SVS-Web: Design and Implementation of Cloud Web-Based SVS-EFIE Solver

by

Alireza Niazi

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Master of Science

Department of Electrical and Computer Engineering
The University of Manitoba
Winnipeg, Manitoba, Canada
February 2024

Thesis advisor
Vladimir Okhmatovski

Author
Alireza Niazi

SVS-Web: Design and Implementation of Cloud Web-Based SVS-EFIE Solver

Abstract

Aim of this project is to revolutionize the field of electromagnetic solvers by developing a cutting-edge web application. It leverages the power of multiple programming languages, and frameworks to seamlessly integrate and interact with electromagnetic solvers.

Our goal is to provide widespread access to the EM solver through a web-based platform, catering to researchers and engineers. Instead of relying on traditional command-line interfaces, a modern and intuitive graphical user interface (GUI) has been developed. This user-friendly GUI improves the overall experience for users and removes the requirement for specialized knowledge of command line interface.

This project acts as a bridge between the user and the EM solver, providing a seamless and intuitive interface to input problem parameters, configure simulations, and visualize results. Users can interact with the solver through a browser, eliminating the need for complex software installations and compatibility issues.

Acknowledgments

I am deeply grateful to Professor Vladimir Okhmatovski for his unwavering support and the valuable training he provided throughout my academic journey. His presence and assistance have been indispensable. Equally, I extend my heartfelt thanks to Dr. Dustin Isleifson, my co-advisor, whose ongoing guidance and belief in my research have been a source of constant inspiration.

I also want to express my gratitude to my colleagues and peers with whom I had the privilege of collaborating during my Master's program. A special thanks goes to Jamiu Mojolagbe for his support and guidance in my thesis.

My family deserve a special mention for their endless support and prayers, which have been a cornerstone of my academic pursuits.

This research endeavor would have been significantly more challenging without the involvement and contributions of the G07 and G08 capstone groups. Their close involvement in this project has been invaluable. Finally, my thanks go to Grex for providing the High-Performance Computing resources, which were crucial in generating the results for the test cases in this study.

To My Sister.

Contribution

- **A. Niazi**, S. Zheng, C. Nguyen and V. Okhmatovski, "Full-Wave Analysis of Interconnects in Finite Substrates with Layered Media Formulation of SVS-EFIE for 3D Composite Metal-Dielectric Structures," 2023 IEEE 32nd Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS), Milpitas, CA, USA, 2023, pp. 1-3, doi: 10.1109/EPEPS58208.2023.10314891.

- Shucheng Zheng, Mahsa Shab, **Alireza Niazi**, Vladimir Okhmatovski, and Dustin Isleifson. "Full-wave electromagnetic analysis of scattering on rough surfaces of arctic sea ice in the presence of oil contamination layer". EMTS, 2023 URSI International Symposium on Electromagnetic Theory

- Shucheng Zheng, **Alireza Niazi**, Mahsa Shab, Vladimir Okhmatovski, and Dustin Isleifson. "Electromagnetic Modeling of Multi-layered Rough Surface of Sea Ice Scattering". 2023 IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting

- Mahsa Shab, **Alireza Niazi**, Shucheng Zheng, Vladimir Okhmatovski, and Dustin Isleifson. "Analysis of Electromagnetic Scattering on Oil- Contaminated Arctic Sea Ice for Remote Sensing Applications". 10th IEEE International Conference on Wireless for Space and Extreme Environments (WISEE 2022)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The central purpose of this project is to revolutionize the field of electromagnetic solvers by developing a cutting-edge web application. This web application leverages the power of multiple programming languages, software libraries, and frameworks to seamlessly integrate and interact with electromagnetic solvers.

By utilizing a web-based platform, we aim to make the EM solver publicly accessible to a wide range of users, including researchers, engineers, and enthusiasts. The traditional approach of using command-line interfaces or standalone software is replaced by a modern, user-friendly graphical user interface (GUI). This GUI not only enhances the overall user experience but also eliminates the need for users to have specialized knowledge of the command line interface.

The web application acts as a bridge between the user and the EM solver, providing a seamless and intuitive interface to input problem parameters, configure simulations, and visualize results. Users can interact with the solver through a browser, eliminating the need for complex software installations and compatibility issues.

To achieve this, we employ a combination of programming languages such as PHP, JavaScript, and HTML/CSS. PHP serves as the backend language, handling the core logic

and computation tasks. It interacts with the electromagnetic solver libraries, which are written in C++, to perform the simulations. JavaScript and HTML/CSS are used for developing the front-end components - the GUI elements and interactive visualization.

Additionally, we utilize software libraries and frameworks tailored to web development, such as Laravel framework, Tailwind CSS, Laravel Jetstream, and Alpine JS. These frameworks provide robust tools for building scalable and maintainable web applications, handling tasks like routing, authentication, and data management.

By integrating all these elements, our web application brings convenience and accessibility to users. The combination of advanced computing power, efficient algorithms, and user-friendly interfaces empowers researchers and engineers to tackle complex electromagnetic problems more effectively and efficiently.

1.1 Background Information

The Surface-Volume-Surface Electric Field Integral Equation (SVS-EFIE) formulation is employed to address scattering and radiation issues associated with both dielectric materials and highly conducting metal objects. SVS-EFIE leverages the connection between surface and volumetric currents, simplifying radiation and scattering problems involving objects that can be penetrated, ultimately reducing the problem to determining a single unknown surface current density along the object's boundary. This approach has been explored in [3–7].

When dealing with scattering or radiation problems, there are different methods available. One approach involves directly discretizing relevant partial differential equations (PDEs) using techniques like finite difference (FD) and finite element method (FEM). Alternatively, the problem can be transformed into a surface integral equation (SIE) or a volume integral equation (VIE).

When opting for SIEs as the solution method, several formulations exist, such as the Electric Field Integral Equation (EFIE), Combined Field Integral Equation (CFIE), Magnetic Field Integral Equation (MFIE), and other variations (as mentioned in [8]). These different EFIEs can be further categorized based on the problem domain that contains the unknown field quantities being considered.

The Surface - Electric Field Integral Equation (S-EFIE) localizes the unknown fields to the object's surface, simplifying the need for surface discretization. On the other hand, the Volume - Electric Field Integral Equation (V-EFIE) is formulated with respect to the unknown field quantities localized within the volume of the object, necessitating discretization of the object's volume [8, 9]. In each of these cases, both electric and magnetic field Green's functions play a significant role.

In the case of using SVS-EFIE for seeking a solution, the method of moment (MoM) discretization necessitates the creation of both surface and volume meshes. This is due to the involvement of Surface-to-Volume and Volume-to-Surface operators [3, 6]. Notably, SVS-EFIE exclusively employs the electric field Green's function and localizes its unknown field quantities to the surface of the analyzed scatterer instead of within its volume. Furthermore, SVS-EFIE stands out as it does not involve derivatives, unlike traditional Integral Equations (IEs) [3, 4, 10].

The solutions to scattering and radiation problems for penetrable conducting objects have various practical applications, such as antenna design, high-speed interconnects, and the analysis of plasmonic structures [11–14]. In the realm of biomedical electromagnetic applications, the calculation of the field throughout the volume of lossy body tissues is necessary for determining parameters like specific absorption rate (SAR) or field penetration depth. In such cases, SVS-EFIE is a suitable method for finding solutions [11].

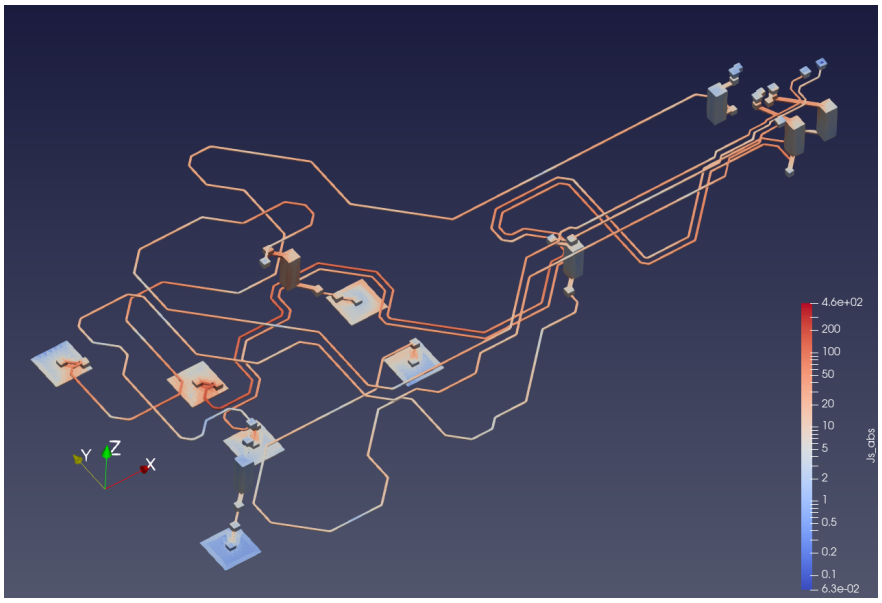

Figure 1.1: The total electric field's magnitude, as determined by the Surface-Voltage-Surface Electric Field Integral Equation (SVS-EFIE) method, for the 6 nets of the IBM interconnect model by [1]

1.2 Purpose of Study

The primary objective of this project is to bring about a significant transformation in the field of electromagnetic solvers. This transformation will be achieved by creating a state-of-the-art web application that harnesses the capabilities of various programming languages and frameworks to seamlessly integrate with electromagnetic solvers.

Our aim is to make electromagnetic solver technology easily accessible to a broad audience, including researchers and engineers, by offering it through a web-based platform. Instead of depending on traditional command-line interfaces, a contemporary and user-friendly graphical interface (GUI) was designed. This intuitive GUI enhances the overall user experience and eliminates the need for users to possess specialized knowledge in electromagnetic theory or programming languages.

This project serves as a bridge connecting users with the electromagnetic solver, providing a smooth and user-friendly interface for inputting problem parameters, configuring

simulations, and visualizing results. Users can interact with the solver directly through a web browser, thus eliminating the complexities associated with software installations and compatibility issues.

1.3 Thesis Structure

This thesis is organized into seven main chapters, each focusing on specific aspects of web architecture, software engineering techniques, Laravel framework, electromagnetic solvers, and the design, implementation, and deployment of the SVSWeb system. The structure is designed to provide a comprehensive understanding of the key elements and methodologies employed in the development of web-based applications and the integration of electromagnetic solvers.

The introductory chapter (Chapter 1) sets the stage for the entire thesis. It provides background information on the subject, outlines the purpose of the study, and introduces the overall structure of the thesis.

Chapter 2 delves into the fundamentals of web architecture, exploring layers, client-server architecture, client-side design and programming, server-side design and programming, and databases design and integration. The goal is to establish a strong foundation in web development principles.

In chapter 3 , various software engineering techniques such as Waterfall, Agile, Scrum, DevOps, CI/CD, Lean Development, and Extreme Programming are discussed. This chapter aims to provide insights into different methodologies and their applicability in software development.

Chapter 4 focusing on Laravel, a popular PHP web application framework, this chapter covers its Model-View-Controller (MVC) architecture, URL mapping and routing, security

features, package development, and deployment strategies.

Chapter 5 introduces the electromagnetic solver, detailing its formulation, model and mesh generation, and the structure of input files required for simulations. Output results and visualization techniques are also explored.

Chapter 6 as core chapter elaborates on the design, implementation, and deployment phases of the SVSWeb system. Topics include system descriptions, UI/UX design, MySQL design process, user management, input generation, job submission, and result visualization.

The final chapter 7 summarizes the key findings and conclusions drawn from the study. Additionally, it outlines potential avenues for future research and development, providing insights into areas that can be explored for further enhancements.

Also appendices serve as a valuable resource for readers seeking a deeper comprehension of the technical aspects underlying the development of the SVSWeb. Each code explanation aims to facilitate a comprehensive understanding of the implemented features and functionalities.

Chapter 2

Web Architecture: Overview of Web Design and Development

Web architecture, a critical component in web design and development, follows the client-server model, separating the user interface from data and application logic. It entails two primary layers: the front-end (client-side) and the back-end (server-side). Front-end development involves creating user interfaces using HTML, CSS, and JavaScript, with frameworks like React, Angular, and Vue.js commonly employed. Meanwhile, the back-end manages data processing, business logic, and interactions with databases, utilizing various programming languages and frameworks, such as Node.js, Laravel, Python, and Django. Databases, a central part of web architecture, store and manage application data, with options ranging from relational databases like MySQL to NoSQL databases such as MongoDB.

Web servers, such as Apache and Nginx, handle client requests and serve web content. APIs, or Application Programming Interfaces, facilitate communication between software components, supporting data exchange and third-party service integration. Ensuring scalability and load balancing is essential to manage increased traffic effectively, while security measures like encryption, authentication, and protection against web vulnerabilities are cru-

cial for safeguarding data and users. Caching optimizes performance by storing frequently accessed data, and Content Delivery Networks (CDNs) speed up content delivery by distributing it across multiple servers in various locations. Web standards compliance and accessibility are imperative aspects of web architecture, and continuous testing and monitoring are key for identifying and addressing performance issues, bugs, and security vulnerabilities.

2.1 Introduction to Web Architecture

A web app architecture orchestrates software components, enabling seamless data delivery through HTTP and ensuring data validity while managing records with permission-based access and authentication. The architecture of a web application consists of key components, including browsers as the user interface, web servers managing resources, and web pages serving as the primary content and functionality delivery mechanism. Web pages are HTML documents that can include client-side scripts like JavaScript to enhance user interactivity. Users engage with web pages by providing input through forms and hyperlinks, triggering events sent back to the server for processing. In server-side processing, business logic, such as data manipulation and interactions with databases or middle-tier components, takes place. This architecture enables dynamic content and functionality delivery in web applications [15]. Selecting the appropriate web application architecture is a pivotal decision in web development, as emphasized by Llyukha [16] and Luchaninov [17]. Luchaninov provides an overview of architecture types, while Llyukha focuses on key considerations, together offering developers valuable insights for creating scalable, reliable, and secure web applications.

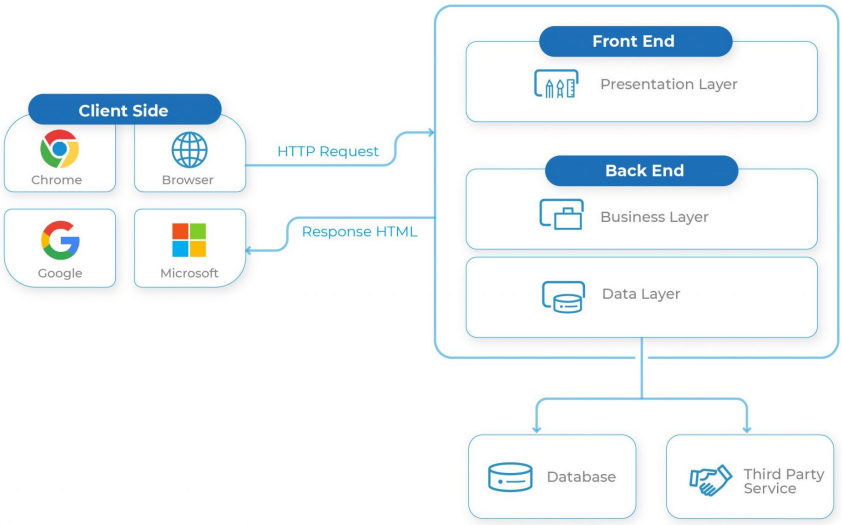

Figure 2.1: A standard web application architecture. Picture is courtesy of ClickIT [2]

2.1.1 Web Architecture's Layers

Web application architecture consists of distinct layers, their count varying based on application complexity as highlighted by Dhaduk [18]. The layers typically include the presentation layer responsible for user interface management utilizing front-end technologies like HTML, CSS, and JavaScript for design, layout, and interactivity. Above this, the application layer manages the application logic and business rules using server-side technologies (e.g., PHP, Java, Node.js) to process requests, perform data validation, and execute core functionality, encapsulating the application's business logic and workflows [18]. The data layer is tasked with overseeing data and the connection to databases within the web application. It incorporates database management systems like MySQL, MongoDB, or PostgreSQL, which are responsible for storing and retrieving data in accordance with the application's needs. The data layer ensures efficient data management, maintains data integrity, and ensures data persistence, thereby enabling the application to reliably store and retrieve information.

In Figure 2.1, intermediary servers receive and process client requests, coordinating with subordinate servers to implement the business logic. The communication between clients and the database is overseen by the intermediate application layer, facilitating client access to data from various database management systems.

The 3-tier architecture prioritizes security, as it prevents direct client access to data. It offers enhanced scalability and performance by enabling the deployment of application servers on multiple machines, which can be independently scaled horizontally. By abstracting the core business logic from the database server, effective load balancing can be achieved, leading to improved data integrity as all data passes through the application server, determining access policies. This architecture also facilitates easy and cost-effective management changes. Employing a thin-client approach at the client layer reduces hardware costs, and its modular nature allows modification of a single tier without affecting the other components.

Emphasizing the significance of constructing web application architecture with individual, adaptable, and modular layers, this approach facilitates simpler maintenance, updates, and future expansion. Separating these layers permits modifications in one layer without influencing the others, enhancing flexibility and versatility. The architecture of web applications encompasses various layers, including presentation, application, data, integration, and security, collaborating cohesively to deliver a secure and operational application. The crucial aspect entails designing these layers to be self-sustaining and expandable, ensuring the ongoing development and upkeep of the web application.

2.2 Client-Server Architecture

A fundamental principle of client-server architecture involves the distribution of tasks between resource providers (servers) and resource requesters (clients). Clients initiate and

terminate sessions by sending requests to servers, which remain in a latent state, processing incoming client requests, and generating responses. When examining these two core components separately, the client-side is relatively straightforward. In this context, the client is the web browser, installed on the user's computer, making request to the server hosting the source code of the application. Clients encompass not only explicitly installed applications but also web pages, serving as clients when using browser-based electromagnetic solvers. This scenario involves a two-tier structure, where the web browser serves as the top-level client, requesting a web page from the server hosting the web application code as well as the EM solver. Upon receiving the request, the web server effectively process the request and return static resources (HTML/CSS/JavaScript) back to the client making the request. As users interact with the pages, the client-side of the electromagnetic solver sends requests to the server, which, in turn, provides feedback to the client application.

2.3 Web Design: Client-Side Design & Programming

In web design, the "client-side" encompasses all elements of a web application that operate on the user's end. This includes the visible components like text, images, and the user interface, as well as any actions carried out within the user's browser[19]. The client-side involves the browser's interpretation of markup languages such as HTML and CSS, and it often involves the incorporation of client-side processes within application architecture, primarily implemented using JavaScript. While the terms "client-side" and "front-end" are not synonymous, the client-side essentially pertains to where these processes are executed, whereas the front-end deals with the user-facing aspects of client-side processes.

2.3.1 Mockups Design

Designing mockups through platforms like Figma [20] and Adobe XD [21] is a pivotal phase in modern web and application development. These tools offer a range of features that facilitate the creation of detailed and interactive visual representations of digital products.

Figma, Adobe XD, and similar tools play a crucial role in user-centered design. They allow designers to focus on creating mockups that prioritize the end user's experience. By providing a highly visual and interactive representation of the final product, these tools ensure that the user interface (UI) and functionality align with user expectations and needs. These tools support user-centered design, streamline collaboration, enhance responsiveness, promote iterative refinement, and provide the means for efficient asset management and handoff to development teams. User testing, version control, and the capacity to export assets add further depth to the capabilities of these design tools.

2.3.2 Client-Side Programming

Client-side programming through the trio of HTML, CSS, and JavaScript is the cornerstone of contemporary web development, enabling developers to fashion dynamic and interactive user interfaces[22]. Each of these components plays a unique and indispensable role in creating immersive web experiences:

2.3.2.1 Hyper Text Markup Language (HTML)

HTML is a unique markup language developed to structure content on web pages. At its core, a markup language distinguishes particular syntax from the main textual content. This differentiation aids in organizing content in a way that various elements - be it text, images, or videos - are positioned purposefully on the webpage. The term HTML is an abbreviation for HyperText Markup Language, and it's a cornerstone for web development since most

web pages are built on it [23]. It operates using a series of elements termed "tags", which are encapsulated within angle brackets. Each opening tag has a corresponding closing tag that starts with a "/". These tags instruct the browser on how to display or process the content enclosed between them. For instance, the "$< p >$" tag indicates a paragraph. Thus, content between an opening "$< p >$" and its corresponding closing "$< /p >$" will be recognized and displayed as a distinct paragraph.

```
<!DOCTYPE html>
<HTML>
    <head>
        <! -- <head> tag contains metadata-->
    </head>
    <body>
        <h1>My First Heading</h1>
            <p>My first paragraph.</p>
    </body>
</html>
```

Listing 2.1: HTML example

2.3.2.2 Cascading Style Sheets (CSS)

CSS emerges as the artistic maestro in the orchestra of web development. This style language empowers developers to orchestrate the visual presentation of web content[24]. With CSS, they wield the power to control layout, color schemes, typography, and the overall styling of HTML elements. Through CSS rules and selectors, developers target specific HTML elements to impart aesthetics. Additionally, CSS facilitates the vital realm of responsive design, guaranteeing the adaptability of web pages to diverse screen dimensions and orientations.

```
1 html {
2   font-size: 10px; /* px means "pixels": the base font size is now 10
     pixels high */
3   font-family: "Open Sans", sans-serif; /* this should be the rest of the
     output you got from Google Fonts */
4 }
```

Listing 2.2: CSS example

2.3.2.3 JavaScript

JavaScript, the dynamic luminary of the client-side stack, infuses life into web pages. As a versatile programming language, it executes within the browser, ushering in real-time user interactions[25]. JavaScript's repertoire spans event handling, form validation, manipulation of the DOM (Document Object Model), and seamless data retrieval from servers via AJAX. It is also the driving force behind captivating animations, the creation of web applications, and the integration of third-party libraries and APIs.

```
1 <script type="text/javascript">
2  alert("Hello World!");
3 </script>
```

Listing 2.3: JavaScript example

2.4 Server-Side Design & Programming

According to Obrizan [26], there are three main types of back-end architecture: monolithic, microservices, and serverless. Each approach offers unique advantages and consider-

ations. Monolithic architecture involves developing the entire application as a single unit, simplifying development and deployment but potentially causing unintended consequences when making changes. Scaling can be challenging since the whole application must scale together.

On the other hand, microservices architecture breaks the application into smaller, loosely coupled services that can be independently developed, deployed, and scaled. It provides scalability and flexibility but can be complex to manage and may involve additional overhead in terms of deployment and maintenance.

Serverless architecture allows developers to build and run applications without managing infrastructure. They focus on writing functions or business logic, and the cloud provider handles infrastructure management and scaling. This approach reduces infrastructure costs and offers automatic scaling based on demand, leading to increased development speed. However, it may lead to vendor lock-in and limitations on the types of applications that can be developed.

Obrizan emphasizes that there's no one-size-fits-all architecture, and the choice should align with the application's specific needs. Smaller, less complex applications may benefit from a monolithic architecture, while larger and more intricate ones can leverage microservices for scalability. Serverless architecture is a suitable option when prioritizing business logic and the advantages of cloud provider scalability. The choice of back-end architecture is pivotal for a web application's success, contingent on factors such as application size, complexity, scalability requirements, development team capabilities, and specific needs. Understanding the trade-offs associated with each architecture type enables developers to make informed decisions aligned with the application's goals and demands.

2.4.1 Server-Side Scripting Languages

In the realm of web development, server-side scripting languages play a pivotal role in crafting dynamic and responsive web applications. Unlike client-side languages that run in the browser, these languages execute on the server, determining how data is processed, stored, or delivered to the end user[27]. Here's an overview of some prominent server-side scripting languages:

2.4.1.1 PHP (Hypertext Preprocessor)

PHP, initiated by Rasmus Lerdorf in the 1990s, transitioned from a humble web-tracking utility to a paramount server-side scripting tool indispensable for dynamic web content creation[28]. Renowned for its open-source ethos and effortless amalgamation with HTML, PHP boasts cross-platform adaptability and compatibility with diverse databases. Its ascent is amplified by an engaged developer community and enriched by frameworks such as Laravel and Symfony. Serving as the backbone for platforms like WordPress, PHP continues to innovate, as seen with its PHP 8 iteration[29]. While security debates have surfaced, its resilience largely depends on diligent deployment, underscoring its enduring relevance in contemporary web development.

2.4.1.2 C# ("C-sharp")

C# serves as a pillar in the server-side web development landscape, particularly when harnessed with the ASP.NET framework. As part of the .NET ecosystem, ASP.NET empowers developers to craft dynamic websites, web services, and APIs, with C# often playing the central role. This union provides access to a rich set of .NET libraries, streamlining operations like data interactions and user management. With the advent of ASP.NET Core, C# has ventured beyond the Windows realm, facilitating web solutions that cater to Linux

and macOS users as well[30]. This modern framework, enriched with features such as flexible middleware, built-in dependency injection, and efficient routing, is further enhanced by C#'s native capabilities like LINQ and async-await patterns. Given the progressive strides by Microsoft in refining and expanding the .NET framework, C# is poised to remain a dominant force in the server-side web development arena for years to come.

2.4.1.3 Python

Python, revered for its elegant syntax and versatility, holds a distinctive space in server-side web development. A key attribute that endears Python to developers is its simplicity, which expedites project timelines, a factor crucial for both budding ventures and tech giants. The language boasts an array of frameworks tailored for web solutions: Django stands out, offering a comprehensive toolkit that encompasses features like an ORM, built-in security measures, and an intuitive admin panel. On the other hand, frameworks like Flask and Pyramid cater to those seeking a minimalistic approach, enabling them to cherry-pick tools for bespoke applications. Beyond the frameworks, Python's expansive ecosystem, populated with a myriad of libraries and plugins, simplifies tasks ranging from data transactions to user session maintenance. With an ever-evolving landscape and unwavering community backing, Python promises to remain a stalwart in the server-side web development arena.

2.4.1.4 Node.js

Within the panorama of server-side technologies, Node.js carves a unique niche. Birthed to bring JavaScript's prowess from the browser to the back-end, Node.js has fundamentally altered how developers view and handle server-side logic[31]. Its signature trait, an event-driven architecture with non-blocking I/O, promotes swift processing of numerous simultaneous requests. This makes it a prime choice for applications that demand high concurrency.

The Express.js framework, a Node.js companion, streamlines the creation of dynamic web services, striking a balance between performance and ease-of-use. Furthermore, Node.js's package ecosystem, known as npm, offers a vast library of modules, turning complex tasks into manageable endeavors. Suited for everything from real-time chat applications to intricate single-page applications, Node.js is a testament to innovation in the web development arena, backed by an ever-growing community of enthusiastic developers.

2.5 Databases Design & Integration

Creating and integrating a database for a web application is a multi-layered process that starts with identifying what data the application will handle[32]. Developers must then design a logical and efficient structure to organize this data. This includes choosing the right database management system and planning how data will be stored and retrieved.

Databases come in two primary forms: relational and non-relational. Relational databases adhere to a structured, table-based format where each row is unique and columns correspond to the attributes of the data[33]. Conversely, non-relational databases, often referred to as NoSQL databases, do not require this rigid, table-like structure and offer more flexibility in how data is stored and connected[33].

2.5.1 Relational Databases

Relational databases are a foundational element in the data management sector, providing a structured environment where information interconnects efficiently. Utilized across various applications, these databases ensure that information is stored in a manner conducive to straightforward retrieval. The structure of a relational database, consisting of records organized in rows with columns as attributes, facilitates a clear distinction of each

data element, making it easily accessible for associated applications.

The strengths of relational databases are manifold. Firstly, they ensure data integrity, maintaining accuracy, completion, and consistency of the information over time. Such databases are inherently user-friendly, boasting an intuitive design that simplifies navigation for users of all experience levels. Moreover, they exhibit remarkable scalability, accommodating the expansion from smaller operations to enterprise-level implementations without substantial restructuring. Lastly, the robust backup and recovery mechanisms inherent in relational databases ensure that they are an asset in mission-critical applications, where downtime is simply not an option[34, 35].

Nevertheless, relational databases are not without their drawbacks. They are not ideally suited for real-time data analysis, as their storage format is not optimized for swift, on-the-fly queries. They also impose restrictions on data types, necessitating the conversion of data into a tabular form for storage, which can be cumbersome. Additionally, they are less space-efficient when dealing with sparse datasets. A significant limitation is the rigidity of the database schema; once set, any modifications require extensive changes across queries and related tables, which can be both time-consuming and prone to errors[36].

2.5.2 Non-Relational Databases

Non-relational databases, commonly known as NoSQL databases, offer an alternative to the rigid structure of relational databases by eschewing the strict schema of tables, rows, and columns[37]. They prioritize performance, scalability, and flexibility, employing a more relaxed approach to data structuring. Without the need for predefined relationships, these databases cater to a variety of data types and structures, streamlining the process of storing and retrieving data to meet the dynamic needs of modern web applications[38, 39].

The benefits of using NoSQL databases are significant, particularly when it comes to

the speed of data operations. They are capable of rapid data retrieval by avoiding the time-consuming joins and queries characteristic of relational databases[40]. This attribute, coupled with their intuitive design, makes them especially suitable for small-scale projects or environments with fewer concurrent users[41]. The scalable nature of NoSQL databases allows them to handle increasing loads by distributing data across multiple nodes, making them ideal for large-scale applications that require processing substantial volumes of data without a predefined schema[42].

On the flip side, the lack of standardization in non-relational databases can pose challenges, particularly when it comes to cross-platform functionality and maintenance. Moreover, some NoSQL databases may struggle with large datasets due to less sophisticated query optimization compared to their relational counterparts. Another potential pitfall is the absence of ACID (Atomicity, Consistency, Isolation, Durability) transactions, which can complicate data updates and integrity. These databases also vary significantly between vendors, necessitating custom code to manage diverse data types and structures, which can complicate application maintenance and integration efforts[43].

2.5.3 MySQL

MySQL stands out as a go-to relational database management system due to its strong reputation for reliability, performance, and ease of use. Its architecture, grounded in the relational model, allows for data to be stored in pre-defined structures of tables and relationships, which are easily manageable and scalable to suit the needs of small to large-scale applications[44]. The system's adherence to ACID principles ensures that transactions are processed securely and consistently, providing a dependable foundation for applications requiring high data integrity[45].

The versatility of MySQL is showcased by its cross-platform compatibility, allowing it to

Feature	Relational Database	Non-Relational Database
Data Structure	Structured (tables)	Unstructured (documents, key-value, graph, etc.)
Schema	Fixed	Dynamic or none
Query Language	SQL	NoSQL (varies by type)
Scalability	Vertical	Horizontal
Transactions	ACID compliant	Varies by system
Best Used For	Complex Queries	Unstructured Data, Big Data
Data Integrity	High (referential integrity)	Varies by system
Performance	High for structured data	High for unstructured data
Flexibility	Low (schema changes are difficult)	High (easy to change structure)
Examples	MySQL, PostgreSQL	MongoDB, Cassandra

Table 2.1: Comparison of Relational and Non-Relational Databases

operate on various operating systems with equal efficacy. This adaptability, coupled with a supportive open-source community and the potential for enterprise-level support, makes MySQL an attractive option for developers. Its integration into the LAMP stack highlights its prominence in web development, serving as the data handling backbone for a multitude of dynamic web services and applications[46].

2.5.4 MongoDB

MongoDB stands out as a NoSQL database for its agile handling of unstructured data, employing a document-oriented storage system which aligns with JSON-like formatting[47]. This structure is inherently schema-less, which grants developers the leeway to incorporate new data types and to modify the database structure on-the-fly. This attribute is especially beneficial for projects that anticipate continuous growth or frequent changes in the data model[48].

The architecture of MongoDB is built for scale, embracing a distributed approach through sharding, effectively managing large datasets by splitting them across multiple servers[49]. This not only facilitates growth but also ensures resilience and high availability. Suited for a variety of use cases, MongoDB is a strong candidate for applications that manage vast amounts of evolving data, from content management systems to complex analytics[50]. Its versatility and performance make it a sought-after solution for modern, data-intensive applications.

Chapter 3

Overview of Software Engineering Techniques

Opting for a specific software development methodology primarily offers a framework and synchronization to the project, fostering a cooperative and productive environment for developers. This organized approach not only enhances teamwork but also ensures regularity in delivering updates and maintaining a reliable schedule for product releases, contributing to a seamless user experience [51]. Well-structured development processes increase the probability of consistent feature enhancements and adherence to projected deployment schedules, which in turn can significantly satisfy user expectations.

This chapter delves into a spectrum of pivotal software development methodologies, encompassing Agile and Scrum, which emphasize iterative progress and team collaboration; Continuous Integration/Continuous Deployment (CI/CD), which automates the steps to deliver applications; the sequential phase-based Waterfall model; DevOps, which integrates development with operations for faster delivery; Lean Development, which prioritizes efficient use of resources and elimination of waste; and Extreme Programming (XP), which focuses on customer satisfaction through continuous feedback and high-quality standards.

Each methodology presents a unique angle on organizing, executing, and optimizing the software development process.

3.1 Waterfall

The Waterfall methodology is a straightforward, sequential approach to software development, characterized by distinct and non-overlapping stages [52]. The process begins with comprehensive requirement gathering, which informs the subsequent design phase. This design stage sets the foundation for what will become the software architecture and hardware requirements. Following design, the implementation phase involves building the software in discrete segments, each undergoing meticulous unit testing.

Once all units are developed and tested, they are amalgamated during the integration phase, where the complete system is then evaluated for defects. Upon successful testing, the software is deployed for use in its intended environment or released to consumers. The final stage is maintenance, where the software is updated and refined to resolve any emerging issues and to improve functionality, ensuring the product remains functional and relevant in the user environment.

The clarity and simplicity of the Waterfall model facilitate its management and use, but it also brings considerable drawbacks. Its rigid structure makes it challenging to accommodate changes once the process is underway, especially after reaching the testing stages. This inflexibility makes it unsuitable for projects where requirements are unstable or likely to evolve, as well as for more complex, iterative, or object-oriented software development projects.

3.2 Agile

The Agile framework is tailored to facilitate the consistent creation of quality software within a designated budget and timeline [53]. A synthesis of various development practices, Agile serves as the foundational structure for several widely-used methodologies, including Scrum [54], Extreme Programming [55], and Kanban [56]. Its philosophy extends beyond prescriptive processes to foster a mindset geared towards continuous improvement and flexibility. Core principles of Agile include incremental funding over upfront heavy investment, aligning engineering efforts with business value to enhance company reputation and client satisfaction, and fostering open communication across technical and business teams to anticipate and resolve issues swiftly. Agile's central tenet is adaptability, encouraging both the development of software and the development teams to remain responsive to technological, organizational, and global shifts.

3.3 Scrum

Scrum, a structured facet of the Agile framework, specializes in iterative progress via manageable increments. It emphasizes clearly defined roles and practices [57]. Central to Scrum is the Scrum Team, typically comprising a Scrum Master, a Product Owner, and developers [58]. The Scrum Master ensures the team embraces Scrum principles effectively, sometimes backed by certifications from Scrum.org, indicating different mastery levels. The Product Owner curates the Product Backlog, a dynamic to-do list detailing the upcoming sprints—short, focused development phases with particular objectives.

This framework thrives on adaptability, with a cycle that begins with the Product Owner breaking down complex project aims into the Product Backlog [59]. The team collectively chooses work from the backlog for each sprint, aiming to transform this work into a value

increment, bringing the final product closer to completion. Post-sprint, the team, and stakeholders reflect on the outcomes to fine-tune future sprints. This cycle repeats until project culmination.

Scrum embodies Agile's core values but with distinct responsibilities for team members, unlike Agile's more open-ended role distribution. Key elements like inspection, transparency, and adaptation align closely with Agile's overarching principles. Scrum's values—commitment, focus, openness, respect, and courage—mirror the 12 key values of Agile, reinforcing its roots in the broader Agile philosophy [60].

3.4 DevOps and CI/CD

DevOps embodies a cultural shift that brings together the realms of software development and IT operations, underscoring the need for their synergy. The philosophy behind DevOps revolves around fostering collaboration, streamlining workflows, and integrating processes between these two traditionally separate units [61]. The goal is to elevate the rate and excellence of software deliveries, as well as to heighten operational proficiency.

At the heart of this movement is the practice of Continuous Integration (CI), a development strategy where team members frequently merge their work—often multiple times per day—into a shared code repository. This frequent integration allows for immediate and automated testing of code, enabling quick detection and resolution of integration issues [62]. Complementing CI, Continuous Delivery (CD) is an approach that ensures software can be deployed to production swiftly and sustainably, facilitating a constant flow of updates to users. This method hinges on a cycle of building, testing, and releasing software at an accelerated pace [63].

Together, CI/CD principles form a critical component of DevOps methodologies, under-

pinning a rapid and iterative deployment infrastructure. While CI concentrates on streamlining and validating code integrations, CD automates the steps to deploy software to various environments, forming a cohesive CI/CD pipeline [64]. This synergistic relationship supports DevOps teams in executing frequent and dependable code updates, ultimately promoting a proactive stance towards ongoing quality assurance. The collective aim is to deliver superior quality software with greater velocity and dependability.

3.5 Lean Development

The Lean Development methodology [65] is an approach to software creation that takes inspiration from efficient manufacturing practices, specifically those that aim to minimize excess while enhancing the overall value delivered to end-users. This philosophy is built upon the idea of doing more with less, streamlining the development process by identifying and eliminating activities that do not directly contribute to the final product's value.

At its core, Lean Development is governed by several guiding principles designed to optimize the development lifecycle. These include relentless waste reduction, promoting continuous learning, postponing decisions to maintain flexibility, accelerating delivery to the customer, empowering the development team, ingraining quality into the process, and adopting a systemic viewpoint of the project workflow [66].

In practical terms, Lean Development emphasizes the importance of understanding and refining the workflow to remove any steps that do not add value. It fosters a culture of continuous improvement, with teams constantly seeking feedback and learning from each iteration of the development process. By pushing decision-making closer to the time of implementation, it allows for more informed and adaptable planning.

Rapid delivery is a hallmark of Lean Development, with the goal of quickly putting the

product in the hands of users to gather actionable feedback. The methodology also champions team empowerment, giving those doing the work the autonomy to manage and direct their efforts. Emphasis on building quality from the outset is achieved through practices such as automated testing and integration.

Lastly, Lean Development promotes an understanding of the development process as a cohesive system. It encourages looking beyond individual components to see the workflow and outputs in their entirety, ensuring that the development process is as efficient and effective as possible.

3.6 Extreme Programming

Extreme Programming (XP) [55] is a disciplined approach to software development that emphasizes frequent releases in short development cycles, which is intended to improve productivity and incorporate new customer requirements flexibly. This agile framework revolves around key principles such as communication, simplicity, and feedback, with a strong emphasis on technical excellence and good design.

Key practices in XP include pair programming, where two programmers work together at one station, continual code review, comprehensive unit testing, and a flat management structure that encourages direct communication. The practice of writing tests before the code (test-driven development) ensures that coding is proactive rather than reactive.

XP places the customer at the heart of the development process, with the flexibility for requirements to evolve and priorities to shift throughout the project lifecycle. The methodology is particularly keen on creating a collaborative atmosphere where respect among team members is paramount, and regular feedback loops are a critical component of the workflow.

By maintaining a focus on the essentials of the task at hand, XP aims to reduce complexity

and unnecessary work. Its core tenet is that the simplest solution that works is the best approach at any given moment. This lean approach to development, combined with the practice of regular and rapid releases, ensures that the product remains aligned with customer needs and quality standards throughout its development.

Table 3.1: Comparison of Software Development Techniques

	Primary Focus	**Key Practices**	**Strengths**	**Weaknesses**
Waterfall	Sequential design process	Sequential phases: requirements, design, implementation, verification, maintenance	Clear structure and milestones, well-documented phases	Inflexible, difficult to adapt to changes, late testing
Agile	Flexibility and adaptability in development	Iterative development, regular adaptation to changing circumstances	High customer satisfaction, ability to manage changing priorities	Less predictability, potential for scope creep
Scrum	Iterative development with fixed-length sprints	Time-boxed sprints, daily stand-up meetings, sprint reviews	Increased control over the project timeline, enhanced transparency	Relies on a mature team with high-level self-organization
DevOps	Collaboration between software development and IT operations	Automation, continuous monitoring, and integration between developers and operators	Improved collaboration and productivity, faster resolution of problems	Cultural shift required, can blur responsibility boundaries
CI/CD	Automated integration and deployment to streamline development	Continuous integration of code and automated testing, continuous delivery to production	Higher code quality, more frequent releases, early detection of issues	Can be complex to set up and requires a robust testing framework
Lean Development	Streamlining production by eliminating waste	Value stream mapping, just-in-time production, and kaizen (continuous improvement)	Waste reduction, focus on delivering value to the customer	Can be too rigid, requires deep understanding of customer value
Extreme Programming	Agile development emphasizing frequent releases in short development cycles	Pair programming, test-driven development, continuous integration, and refactoring	Improved software quality, high adaptability, and constant feedback	Can be more time-consuming, requires constant communication and feedback

Chapter 4

Introduction to Laravel Framework

Laravel, recognized as a comprehensive framework, streamlines complex web application development by automating common tasks such as routing, session management, caching, and authentication. Its full-stack nature allows for a seamless development experience by covering every aspect from server management and database interaction to rendering HTML content. Laravel stands out in the realm of PHP frameworks for its philosophy of 'convention over configuration,' offering a pre-set environment that requires minimal setup before diving into development. This approach, characterized by a standard code structure, eases the learning curve and enhances productivity, despite appearing to limit developer freedom initially. With its out-of-the-box organizational structure complete with directories and files, Laravel not only facilitates a rapid start to projects but also offers extensive customization options, catering to developers who seek both efficiency and flexibility.

This framework includes every element required for a typical MVC web framework, with a high degree of customizability. It's equipped with Eloquent, an ORM that simplifies database interactions, and the Blade templating engine, which streamlines the web development process, allowing for rapid and efficient creation of websites.

4.1 MVC Architecture

Laravel adopts the model-view-controller (MVC) design pattern, which segments the application logic into three interconnected parts, enhancing organization and performance [67]. This structured approach promotes cleaner code and modular design. In Laravel, distinct directories and libraries signify the separate roles of each component in the architecture. Furthermore, the efficiency of Laravel's MVC structure can be evaluated by executing intricate database queries within a single request to the server, demonstrating its capability to handle complex operations efficiently [68].

4.1.1 Model

The model layer in Laravel serves as the foundation for database interactions, employing object-relational mapping (ORM) and active record patterns to facilitate database management [69]. ORM serves as an abstraction that converts data between incompatible systems, transforming database queries into a more understandable syntax by aligning in-memory objects with database entries [70]. Laravel's implementation of ORM, known as Eloquent, aids developers in crafting code that is more organized and legible, thereby enhancing maintainability for ongoing development projects. By using models, developers can perform database operations without writing SQL queries explicitly, making the codebase more maintainable and secure.

4.1.2 View

Views in Laravel are created using the Blade templating engine. These are the templates that render the user interface, and they are kept separate from the business logic. Blade templates allow for typical templating tasks such as echoing out data, looping through data

structures, and extending layouts, with a very clean and readable syntax. The View layer in Laravel serves as the presentation segment, managing the user interface elements and the interactive aspects of the application. It utilizes foundational web technologies such as HTML, CSS, and JavaScript to craft and manage the look and feel of the application. Within this layer, native PHP code can coexist harmoniously alongside Laravel's built-in functions, providing a versatile environment for developers to create dynamic user experiences.

4.1.3 Controller

In Laravel, controllers are responsible for processing incoming requests to the application. They are composed of various actions that respond to distinct HTTP requests related to specific features or sections, like signing up users or presenting a data dashboard. By utilizing models, controllers acquire the data required, implement the core logic of the application, and then transmit that data to the views for display purposes. For instance, a controller might manage a form submission from the view, validate the input, commit the validated information to the database, and then navigate the user to different pages or routes.

4.2 URL Mapping and Routing

Laravel's routing mechanism comes equipped with its own URL rewriting capabilities, enabling the creation of concise and SEO-friendly URLs. A basic route in Laravel can be established with just a URI and a closure function [71]. This routing system empowers developers to define routes corresponding to various HTTP methods. Laravel's routing engine is capable of interpreting six distinct HTTP verbs: GET, POST, PUT, PATCH, DELETE, and OPTIONS [72]. Additionally, routes can be enhanced with middleware—which offers additional functionalities like CSRF (Cross-Site Request Forgery) protection. Forms in HTML

that are directed towards POST, PUT, or DELETE routes are mandated to include a CSRF token field for security measures.

4.3 Security

Firstly, Laravel shields applications from cross-site request forgery (CSRF) attacks. It does so by generating and verifying CSRF tokens, which must be included in forms that submit POST requests. This CSRF token ensures that the request originates from the authenticated user and not an external source.

Another critical security feature in Laravel is protection against cross-site scripting (XSS). Laravel automatically escapes output when using its Blade templating engine, preventing malicious scripts from being injected into the HTML. This measure ensures that any data output to the view is safe to display.

Laravel also secures applications from SQL injection attacks. With Eloquent, the ORM that Laravel provides, all database queries are prepared statements, which are inherently protected against SQL injection. This means that the query structure and data are separate, so users cannot inject malicious SQL through form inputs or URL parameters.

In addition to these, Laravel employs hashed and salted password mechanisms, meaning that passwords stored in the database are encrypted in such a way that even if the database is compromised, the actual passwords cannot be easily retrieved.

Furthermore, Laravel uses secure, encrypted sessions to maintain state between requests. This encryption uses Laravel's own encryption keys, which can be generated using artisan commands and are never stored in the application code, thus adding an extra layer of security.

Laravel's robust approach to security demonstrates its commitment to building secure applications by default. Its architecture encourages developers to write secure code and

provides a wealth of security features out of the box.

4.4 Package Development and Management

In Laravel, package development and management are integral components that extend the framework's capabilities, allowing developers to create reusable code for multiple applications or share with the community.

Developing a package in Laravel typically involves creating a specific directory structure, setting up a service provider that binds the package's functionality into Laravel's service container, and establishing routes, views, and configurations specific to the package. These packages can encapsulate everything from small features, like an image manipulation helper, to entire systems, such as an e-commerce platform.

For managing these packages, Laravel utilizes Composer, a dependency management tool for PHP that facilitates the installation and update of libraries within a project. Composer allows you to declare the libraries your project depends on and it will manage (install/update) them for you.

When developing packages, good practices include adhering to the principles of object-oriented programming, following the PSR standards for coding style, and writing comprehensive documentation to aid other developers in using and contributing to the package.

Once a package is developed, it can be distributed via Packagist, the main Composer repository, which serves as a public registry where packages can be searched and integrated into any project with ease.

In the context of managing existing packages within a Laravel project, developers can specify the required packages in the composer.json file, and Composer will handle the downloading and updating of these resources. This management process ensures consistency and

version control for the libraries being used, contributing to the maintainability and stability of the application.

4.5 Deployment

Deploying a Laravel application involves transferring the code from a local development environment to a production server where it becomes accessible to users. The deployment process is crucial as it must be done securely and efficiently to ensure application stability and availability.

To begin deployment, developers often use version control systems like Git to track changes and collaborate on code. Once the code is ready for production, it's pushed to a central repository from where it can be deployed.

There are various deployment strategies for Laravel, including manual deployment, which involves uploading files directly to a server via FTP, or automated deployment using tools like Laravel Envoyer, which enable zero-downtime deployment, meaning users face no interruption in service as updates are made.

Before deploying, it is essential to set the application's environment to 'production' to optimize performance settings. Configuration files should also be adjusted accordingly, including setting up environment variables, database connections, and mail drivers.

The next step often involves using Composer on the server to install dependencies in the production environment, followed by running Laravel's migration command to set up the database schema.

For automated deployment, Continuous Integration/Continuous Delivery (CI/CD) pipelines are commonly used. These systems automatically test the codebase and deploy it to production if tests pass, ensuring that only tested and stable code is deployed.

After deployment, it's critical to monitor the application for any issues and have a rollback plan in case something goes wrong. Tools like Laravel Forge can help configure server environments and manage deployments directly from the repository.

Deploying a Laravel application is a process that should be carefully planned and executed, with considerations for security, performance, and the end-user experience. It's recommended to have a checklist or a script to ensure that all steps are completed systematically for each deployment.

Chapter 5

Brief Overview of Stratum3D (SVS-EFIE) Electromagnetic Solver

The SVS-EFIE formulation, which integrates the concepts from Single-Source Surface Integral Equation (SSIE) [73, 74] and the conventional volume equivalence principle [8, 75], was initially introduced in references [4–7] for tackling 2-D scattering problems on a variety of objects such as homogeneous, piece-wise homogeneous dielectrics, and highly conductive materials. The process of discretizing the 2-D SVS-EFIE using the Method of Moments (MoM) was detailed in [10].

Moreover, the application of SVS-EFIE extends beyond homogeneous objects, proving effective in the analysis of scattering problems in multi-layered media [76]. While initial studies [4–7, 10] focused on 2-D problem analysis to establish the formulation's accuracy, recent efforts have shifted towards a more comprehensive 3-D scattering problem analysis on homogeneous dielectric objects, employing the MoM discretized SVS-EFIE formulation [3].

The SVS-EFIE method offers several benefits, including the integration of a single electric field-type integral operator, a mixed potential approach free from hypersingular integrals, and other practical advantages in problem-solving and MoM implementation [3–7, 76]. How-

ever, this technique faces challenges, primarily due to the computational demands associated with translating fields from the scatterer's surface to its volume and vice versa [Surface-to-Volume and Volume-to-Surface operator computations]. This dual translation necessitates discretization of both the scatterer's surface and volume. Consequently, at high frequencies, the increase in volumetric mesh elements becomes a significant issue. This has led to arguments in [77, 78] suggesting that the SVS-EFIE method may not be suitable for high-frequency scattering problem analysis.

5.1 Formulation

The STRATUM3D solver employs a combination of the Surface-Volume-Surface Electric Field Integral Equation (SVS-EFIE) and the Mixed-Potential-Integral-Equation (MPIE), collectively known as SVS-S-EFIE, as its foundational equations [11, 79, 80]. In terms of discretization, it utilizes the Method of Moments with the Rao-Wilton-Glisson (RWG) approach for surface triangle meshes and a piece-wise technique for tetrahedral volume meshes.

For efficient computation, STRATUM3D incorporates hierarchical matrices (H-matrices) to expedite both the formation and the solving of the matrix equation. The computational requirements for direct and iterative solutions of the matrix equation are proportional to O(NlogN) for the number of surface degrees of freedom (N) and O(MlogM) for the volume degrees of freedom (M) [81, 82].

Regarding the evaluation of Green's function, the solver uses the Mixed-potential components of the Green's function for multilayered media [83]. This evaluation can be done through either the Discrete Complex Image Method (DCIM) [84] or the Weighted Averages (WA) method [14].

The STRATUM3D tool has seen practical application and validation in electromagnetic

analysis across a range of areas. These include:

- Analysis of microwave circuits, involving structures ranging from 1 to 10 millimeters, within a frequency spectrum of 1 MHz to 20 GHz.
- Examination of high-speed interconnects, featuring structures from 10 to 100 micrometers, in a frequency range of 1 MHz to 60 GHz.
- Evaluation of integrated circuits with structural dimensions between 1 and 10 micrometers, operating in the 1 MHz to 10 GHz frequency range.
- Analysis of power systems, which includes larger structures from 1 to 50 meters, within a lower frequency range of 10 kHz to 2 MHz.
- Application in remote sensing, for structures measuring 0.01 to 1 meter, in a frequency bandwidth of 0.1 to 10 GHz.

5.2 Model and Mesh Generation

Gmsh is a popular open-source 3D finite element mesh generator with a built-in CAD engine and post-processor. Its design is centered around providing a streamlined and flexible tool for mesh generation, which is crucial in computational simulations. Gmsh supports a variety of geometrical definitions and physical attributes, and it is commonly used in academic and research settings due to its versatility and comprehensive documentation available at gmsh.info.

Regarding electromagnetic (EM) simulations, models are initially crafted using the Gmsh CAD software, saved in the .geo format, as provided by the Gmsh website. These models are essential for the subsequent creation of both surface and volume meshes. These meshes are crucial for the effective functioning of the STRATUM3D solver, which requires them for its

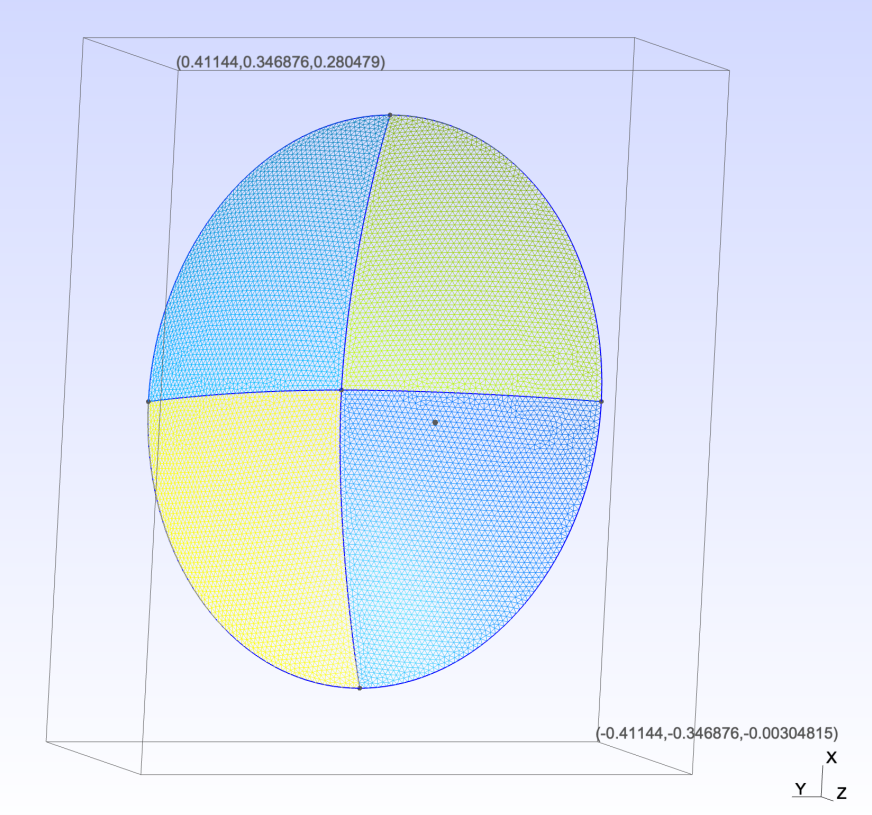

Figure 5.1: Depiction of the mesh for parabolic dish antenna object in Gmsh (See Appendix B For the .geo File scripts)

computational processes. The .geo file needs to be converted into a mesh format, specifically into a "Version 2 ASCII" .msh file format for compatibility and use.

5.3 Input Files Structure

5.3.1 Main Input File

The "input.input" file should be placed in the same folder as the executable, alongside the _INPUT, _KERNEL, and _OUTPUT directories. This file is crucial as it sets most of the control variables for the solver. An Example of the input.input file can be found in Appendix A. The configuration for solving the integral equation formulation requires several key parameters:

- The name and path of the material file relative to the executable.
- The mesh file and its path, also relative to the executable.
- A scale factor for mesh vertex coordinates along the x, y, and z axes, such as factors of 1, 1, 10 to elongate the model tenfold in the z-direction.
- A vector specifying the translation of the geometry in three-dimensional space.
- The path to the wavenumbers file, relative to the executable, which includes a list of excitation frequencies and wavenumbers.
- The chosen testing scheme within the Method of Moments (MoM).
- The selected type of preconditioner.
- The specific type of matrix equation solver.
- The model used for conductor loss.
- The mechanism for excitation, such as 'DipoleViaRWG' for electric dipole excitation near layered media, or 'Dipole' for far-zone locations. The 'PlaneWave' option is compatible only with a 'FreeSpace' kernel setting. To simulate a plane wave in layered media, 'Dipole' excitation with a far-zone dipole position and appropriate orientation for wave polarization and direction is needed. For near-field dipoles with a 'MultiLayer' kernel, 'DipoleViaRWG' should be selected, which involves adding a pair of triangles with an RWG function to the surface mesh. The 'DipoleViaRWG' option determines the dipole's location by the RWG function's position in the mesh (material tag '100'), and while the 'Dipole location' need not be specified, 'Dipole orientation' must be set.
- The polarization type of the incident plane wave.

- The dipole location, which can be specified in Cartesian or spherical coordinates.
- The number of surface ports, which can be defined either by listing vertex indices from the mesh file (using 'points' in Gmsh's 1-based indexing) or by explicitly stating the x, y, z coordinates of the port vertices (using 'meters').
- The file that defines the type of Green's function used in the solution.

5.3.2 Material File

An illustration of a file that specifies material properties aligned with the material tags in the mesh is as follows. The order in which the material properties should be arranged is critical: First, define the materials used in dielectric regions. Next, detail the materials for metallic objects. For metals, although all parameters such as #tag, #eps_real, #mu_real, #sigma, and #thickness need to be assigned nominal values, these are largely overlooked except for #sigma, which is considered if a loss model is mentioned in the .input file. Lastly, the material assigned the tag '100' is to be specified, which is used for elements that bear the impressed current.

```
#tag #eps_real #mu_real #sigma #thickness
3
1 4.5e+0 1.0e+0 0.00077e+0 0.0e+0
2 4.5e+0 1.0e+0 0.00077e+0 0.0e+0
100 4.5e+0 1.0e+0 0.00077e+0 0.0e+0
```

Listing 5.1: An Example of Material File Corresponding to the Mesh with .txt Format

5.3.3 Wave File

The wavenumber k and its Cartesian components kx, ky, kz represent the spatial frequency of the wave, with kx, ky, kz indicating its distribution along the x, y, and z axes, respectively. Additionally, ϕ and θ define the polarization of the wave, while $freq$ denotes its frequency. In terms of data organization, the frequency count should be placed in the first line. Following this, the second line should include the frequency value, the polarization angles ϕ and θ, and the wavenumber components kx, ky, and kz(Parameters are Space-Separated).

```
2
0 1.000000000e+06 0.000000000e+00 1.570796326e+00 -2.095845021e-02
    -0.000000000e+00 -9.785626027e-16
1 2.000000000e+06 0.000000000e+00 1.570796326e+00 -4.191690043e-02
    -0.000000000e+00 -1.957125205e-15
```

Listing 5.2: Wave File Format for the Simulation. Note the Wave file has to be in .txt format

5.3.4 Multilayer File

This file details each layer's specific attributes, like conductivity and permittivity, as well as the layer's position along the z-axis. It is important to note that for each layer, the parameters 'Maximum number of z samples' or 'Maximum number of rho samples' can be assigned a value of -1. This allows the solver to autonomously determine the sampling step in the Green's function kernel database. An example file for defining a layered stack is included in Appendix C.

5.4 Output results and visualization

Computed current distributions on the metallic regions of the model and electric field distributions within the dielectric regions are exported in .vtk format. These outputs can be effectively visualized using Paraview, a versatile, open-source application designed for data analysis and visualization, available at paraview.org. Additionally, the network parameters matrices, including S-, Y-, and Z-parameters, are generated in .m format, suitable for visualization in Matlab.

Paraview stands out as the primary visualization tool in our project, being a multi-platform application. It offers a range of visualization methods, such as 3D Glyphs, Feature Edges, Outline, Point Gaussian, Points, Surface, Surface with Edges, and Wireframe. Among these, the Surface representation is most frequently utilized in our work. The VTK format's compatibility with various lookup tables enables Paraview to display diverse color tables, each representing different features.

Chapter 6

Design, Implementation and Deployment of SVSWeb

The web application consists of a front-end and back-end architecture. The front-end is structured into various modules, each representing a different webpage. These modules operate as single-page applications, heavily utilizing AJAX requests for dynamic content updates. On the back-end, a content management system is in place, allowing administrators to modify and control the content displayed on the front-end.

The website's front-end offers a user interface with several features accessible to both registered and anonymous users. Features open to all include demonstrations, an input configuration generator, a VTK visualizer, an about page, and account creation options. Once a user registers and logs in, they gain access to a personalized profile page. This page allows them to alter their account details, submit electromagnetic problems to a remote solver, validate their submissions, check the status of their current jobs, download results from the solver, or view their uploaded mesh models. Users with administrative rights have the additional capability to access the backend management area, where they can modify various aspects of the website.

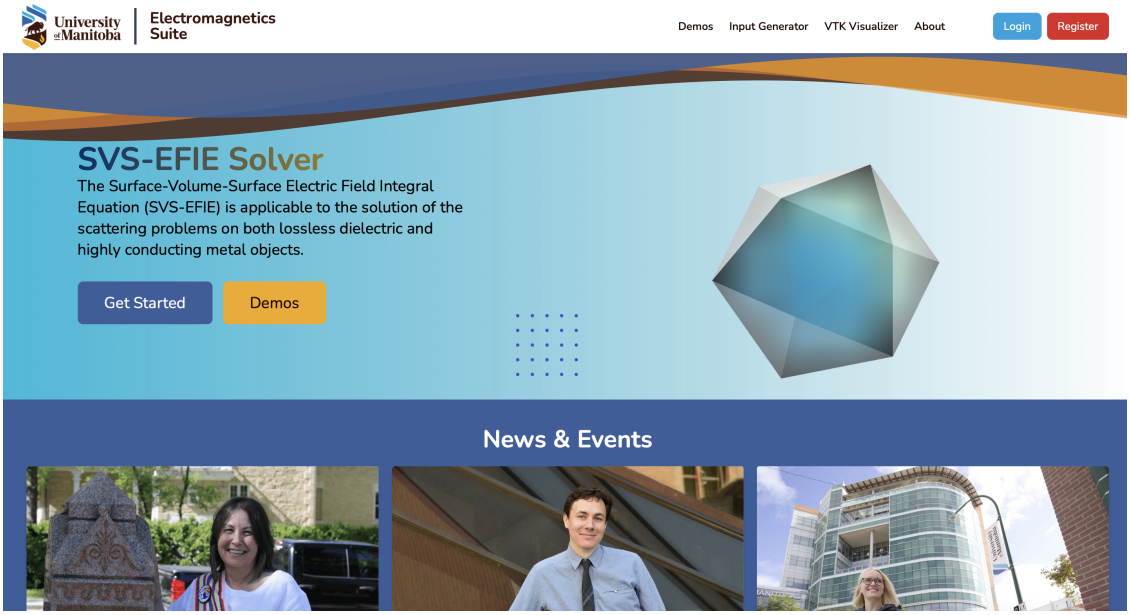

Figure 6.1: SVS-Web Home Page

The back-end is linked to a MySQL database, where all the website's data is stored in different tables. Data manipulation primarily occurs through an Object Relational Mapping (ORM) tool provided by the Laravel framework, although direct SQL queries are sometimes necessary.

Unlike typical implementations, the website's session management is database-driven, with a dedicated table for sessions, avoiding the use of caching systems like Redis. The website's server connects to an electromagnetic (EM) solver node via Secure Shell (SSH). User-uploaded files are stored in a storage service, which could be the web server itself or an external service like Amazon S3. Before being processed by the EM solver node, which is a powerful cluster with numerous CPUs, the job input files are checked and prepared using a Polydata converter and an input visualizer. This processing can be time-intensive.

6.1 System Descriptions

Users of the website can register and log in to access various features. Everyone can view solver demos, which include necessary input and output files, and browse through different categories of electromagnetic problems. Registered users gain additional privileges such as creating and validating input files directly on the website, receiving guidance for each parameter during this process, submitting electromagnetic problems, and getting email notifications about their submissions. They also have the ability to view their models online, and can edit or delete their profiles at their discretion.

Administrative functions on the website are extensive. Administrators can manage user accounts, including deletion and modification of user information, although they cannot change users' passwords. They are responsible for creating, deleting, and editing dynamic web pages, front-end demos, and their associated content. Additionally, admins oversee job submission statuses, and can edit or delete these records. They also maintain the site's settings and manage the categorization of demos.

From a system standpoint, the website is designed for efficiency and user-friendliness. Management pages feature advanced pagination, sorting, and ajax search functions. The front-end is designed to be straightforward and orderly, with files stored externally to save on performance and space. User submissions undergo rigorous validation at multiple stages, ensuring reliability and security. The system records job submission statuses in a database and communicates with the EM solver via APIs. This setup guarantees that the website not only functions smoothly but also maintains high standards of data security and integrity.

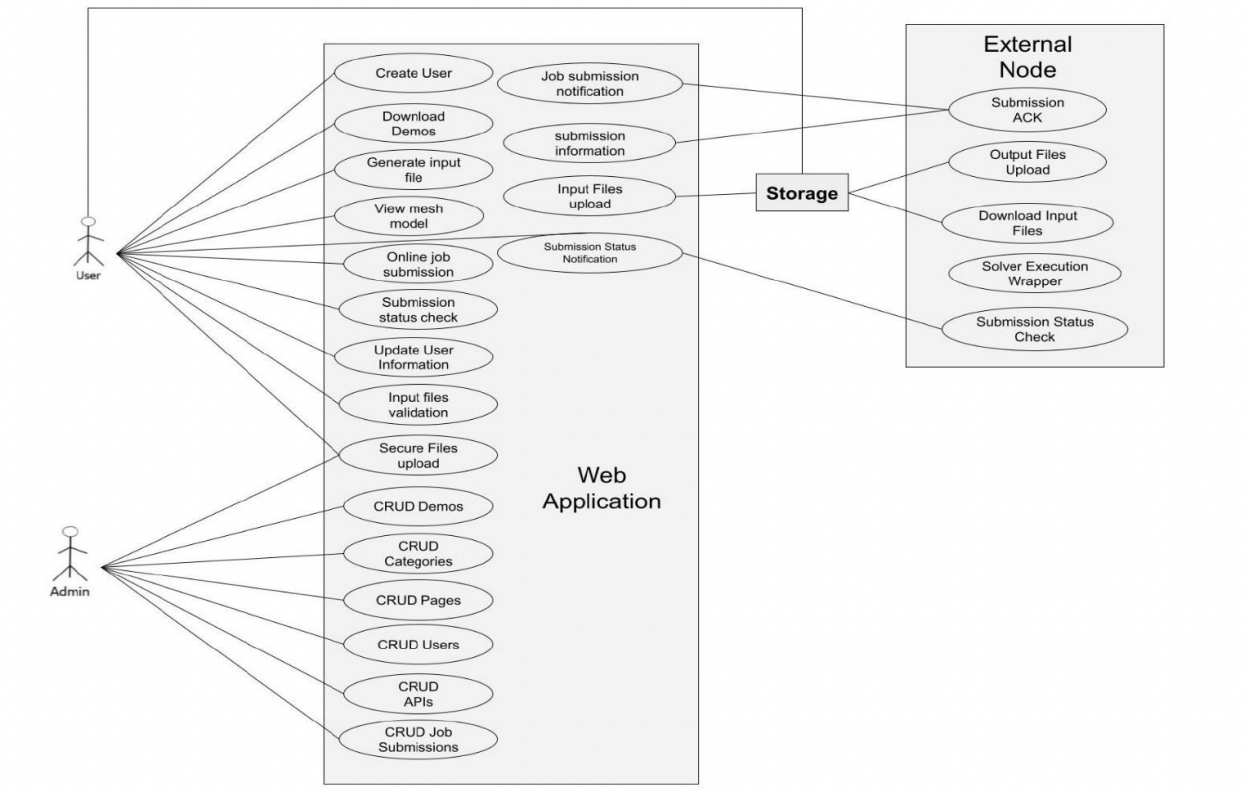

Figure 6.2: System Descriptions in UML (Unified Modeling Language) Diagram. This diagram plays a crucial role in understanding and effectively communicating the structure, behavior, and interactions within the system. The diagram is courtesy of G07's report.

6.2 UI/UX Design and Implementation

To design the new user interface and user experience, Figma was employed for its comprehensive array of design elements and tools, which facilitated the creation of both engaging and user-friendly designs. The initial stages involved developing wireframes, followed by the creation of high-fidelity mock-ups to closely resemble the final product's appearance. After finalizing the mock-ups, the coding phase commenced.

The backend of the web application was developed using Laravel, a widely-used open-source PHP[28] framework. This development phase included updating both the Laravel Blade[85] and the associated Livewire[86] components. The Blade template engine in Laravel compiles templates into PHP code for server execution, with the resulting HTML sent to

the client's browser for rendering. Livewire components, on the other hand, are used for crafting smaller, interactive components of the UI, dynamically integrating into the HTML and managed through Livewire.

The styling of the web application was achieved using TailwindCSS[87], a utility-first CSS framework known for its pre-designed classes that streamline the process of styling HTML elements. Its ease of use contributed to an expedited development process.

6.3 MySQL Design Process

In our web application, various functionalities like user accounts and job submissions necessitate data to be persistently stored. We opted for MySQL as our database of choice in the persistence layer, organizing distinct features into individual or multiple tables to facilitate complex relational analysis. MySQL's use of B+ tree indexing is pivotal; this self-balancing tree structure automatically adjusts the height balance of left and right sub-trees with every addition or removal. This feature enhances performance, particularly in range searches and equality comparisons. Fig 6.3 is an outline of our database implementation in object modeling terms.

6.4 User Management

6.4.1 User Management - Front-end

The User Profile page on our website is a versatile space where users can update their personal information, submit new jobs for processing by the solver, or check the status of their ongoing or completed jobs. Given the page's critical role and functionality, it is expected that users will spend a considerable amount of time there. To enhance user experience, the

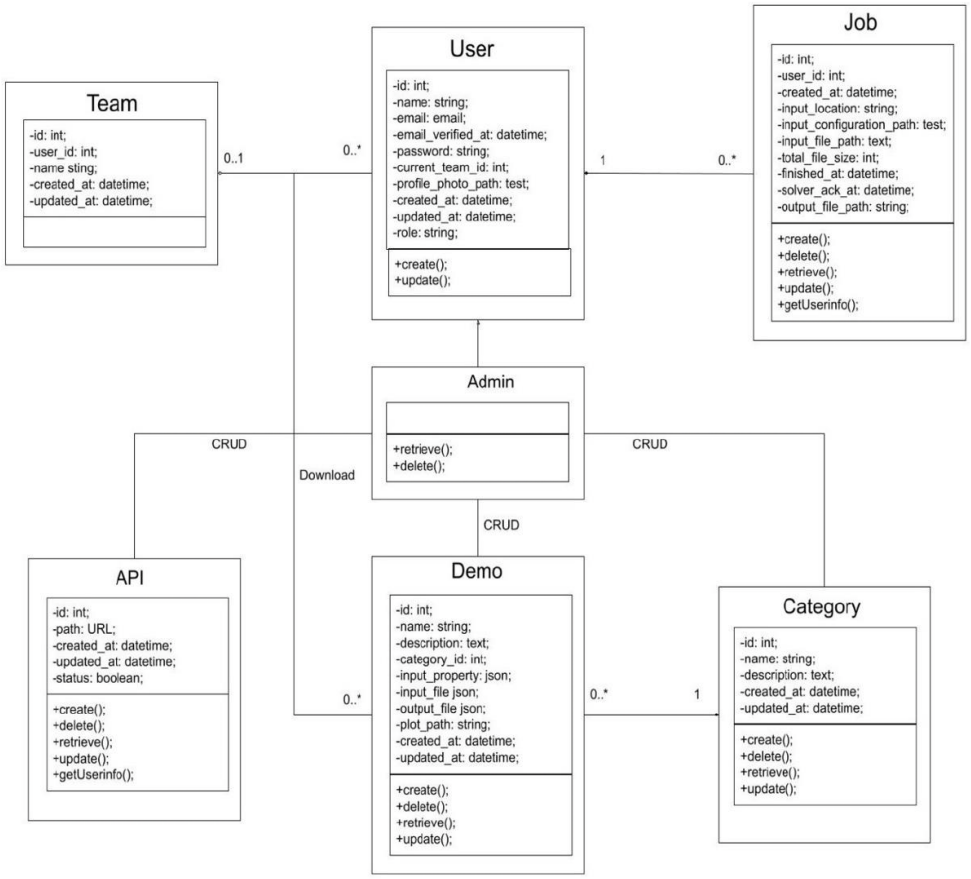

Figure 6.3: The system object model diagram. It provides an overview of how tables in our database are interconnected, without delving into intricate details. We've divided the job-related data across several tables to achieve finer granularity. Additionally, the 'admin' role is not a separate table but rather a distinct permission level within the user table.

job table section on the User Profile page underwent a redesign to present information more clearly. The "Submission Date" column was adjusted to display only the submission day, rather than the full date and time including milliseconds. Furthermore, the "Remote Status" column was revamped for clarity, replacing letter codes with enumerated descriptions for a more comprehensive understanding of the user.

6.4.2 User Management - Backend

The backend management system is equipped with capabilities to modify user details and manage administrative permissions. It has the authority to deactivate or remove a user, alter a user's team, or elevate a regular user to an administrator role. The system also features a table that displays the status of various users, along with an interface for editing these details.

6.4.3 Authentication

Certain web pages on our site, like user profile sections and job submission areas, are deliberately concealed and can only be accessed by verified users. We employ the Fortify package in Laravel as a foundational authentication toolkit. It offers a range of functionalities, including registration, login, password reset, among others. However, we've customized and overridden some of its default features to suit our specific requirements. The initial step involves setting up a user table in our database to house all user-related information.

```
return DB::transaction(function () use ($input) {
        return tap(User::create([
            'name' => $input['name'],
            'email' => $input['email'],
            'password' => Hash::make($input['password']),
        ]), function (User $user) {
            $this->createTeam($user);
        });
    });
```

In our system, each user is uniquely identified by an auto-incremented primary key and an email address serving as the candidate key, ensuring uniqueness across users. Upon account

#	Name	Type
1	id	bigint(20)
2	name	varchar(191)
3	email	varchar(191)
4	role	enum('user', 'admin')
5	status	tinyint(1)
6	email_verified_at	timestamp
7	password	varchar(191)
8	two_factor_secret	text
9	two_factor_recovery_codes	text
10	remember_token	varchar(100)
11	current_team_id	bigint(20)
12	profile_photo_path	text
13	created_at	timestamp
14	updated_at	timestamp
15	permission	int(11)

Figure 6.4: The Database User Table

registration, basic user details like name, email, and password are stored in the database's user table, with other attributes set to default values. Passwords are securely hashed before being saved in the database.

The user authentication process involves logging in with an email and password, sent to our web server via HTTP POST. The server processes these login requests sequentially, applying multiple checks to verify user credentials. Cookies, which are HTTP headers, play a crucial role in storing data client-side. When a client receives a "Set-Cookie" command, the browser saves the cookie as a key-value pair. This cookie persists even after the browser is closed, unless manually cleared or expired.

Given the stateless nature of HTTP, traditional web servers use sessions to maintain user states. Session storage can vary - files, databases, buffer memory, or even encrypted

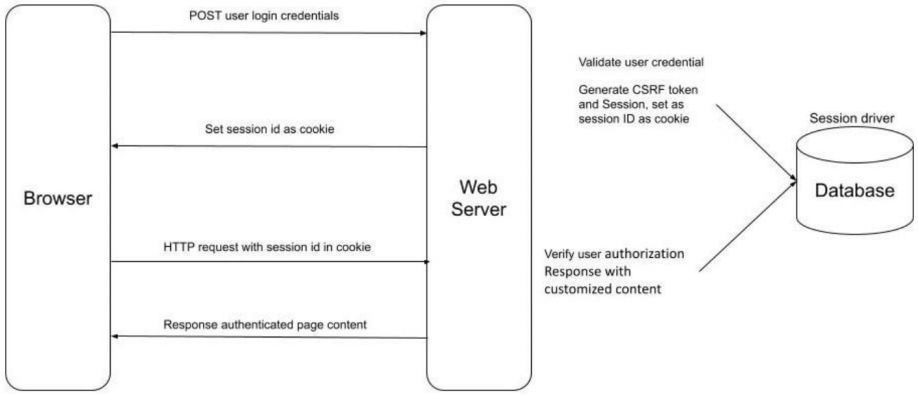

Figure 6.5: Authentication Flow

cookies. Our application employs database-stored sessions. Once a user is verified, the server generates a session ID, storing user information as the session payload, and sends it back in a cookie containing the session ID. This cookie is encrypted in Laravel, ensuring the key-value pairs don't directly correspond to the session ID without decryption.

Subsequent requests include this cookie, enabling the server to authenticate the user and control access to specific URLs. While cookie-session authorization is susceptible to CSRF (cross-site request forgery), where attackers can forge requests in a user's name, Laravel mitigates this risk with anti-CSRF tokens [88], providing robust protection against such attacks. To confirm a user's authentication status, the backend checks the database using the session ID provided in the cookie header. If a match is found and the user is authenticated, the request is authorized for execution. URLs related to the user profile are safeguarded by middleware[89] that screens requests based on their authentication status prior to further processing. Additionally, backend URLs include a permission check middleware to restrict access to non-administrative users. Figure 6.6 illustrates the user profile page as seen by an administrator after logging in.

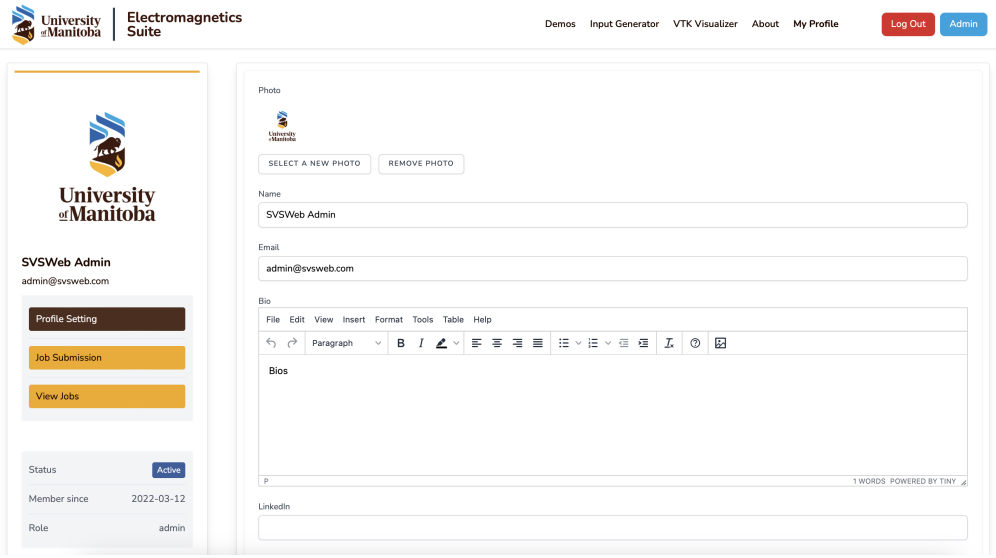

Figure 6.6: User Profile Page

6.5 Input Generator

The front-end of our website features an input configuration generator capable of producing three different file types (based on different types of EM solver), essential for the converter and solver's configuration. The solver utilizes these files to establish parameters and load necessary inputs. Due to the extensive number of parameters, the generator's process is divided into advance and basic options, with hints provided for less obvious parameter names. These parameters are either integers or selectable options, with the option to hide the file name parameter by altering a JSON configuration file, which will be explained subsequently.

The interdependency of input parameters means that the presence of one parameter might hinge on another, or certain parameters could influence the relevance of others. Some parameters, although read by the solver, may not be actively used. To manage this complexity, we designed the input configuration generator as a dynamic web page. It operates by loading a JSON-type configuration file located in the website's application directory. Each

```
"kernel": {
    "title": "Kernel:",
    "type": "select",
    "default": "FreeSpace",
    "validationRule": "nullable|in:FreeSpace,MultiLayer,HalfSpace",
    "options": [
        {"title": "FreeSpace", "value": "FreeSpace"},
        {"title": "MultiLayer", "value": "MultiLayer"},
        {"title": "HalfSpace", "value": "HalfSpace"}
    ],
    "children": {
        "elevation": {
            "_enabled": {"property": "kernel", "value": "HalfSpace"},
            "htmlType": "number",
            "validationRule": "nullable|numeric",
            "title": "Elevation",
            "type": "number",
            "default": "0.000"
        }
    }
},
```


Figure 6.7: Input Generator JSON File Configuration

time a user accesses the input configuration page, the web server dynamically generates the page based on this JSON file. This approach allows for easy future modifications of the generator's parameters by simply updating the JSON file, eliminating the need for source code changes. An explanation of the class file can be found in Appendix G

6.6 Job Submission

The job submission component is a key functionality within our web application. It primarily focuses on three critical activities: uploading and queuing jobs to the Linux cluster and GREX[90], monitoring and updating the status of ongoing jobs remotely, and downloading the results of completed jobs. The entire process of job submission, status updates, and output retrieval is managed autonomously by the server, which interacts seamlessly with

both the database and the computational resources.

Initially, upon submission through the web application, the job details are recorded in the database, and associated elements like converters/solvers are stored on the server. The job transfer from the web application to the server is facilitated by the Laravel scheduler[91]. Once the server receives the files, it executes a script to initiate the solver and places the job in a waiting queue for execution.

In the second stage, once the solver completes the job, the server transfers the output files back to the web application. Simultaneously, the job owner is notified about the completion of their task. This seamless process ensures efficient handling and execution of jobs submitted online.

6.6.1 Laravel Scheduler

The Laravel scheduler[91] serves as a tool for managing scheduled tasks on the server, enabling the web application owner to smoothly set up command schedules within the server and the application. Cron, a task scheduler mechanism in Unix/Linux operating systems, orchestrates tasks based on predefined intervals, such as days, weeks, months, or specific dates and times. The configuration for these schedules is managed through a Crontab, or Cron Table. Each Cron job consists of two essential elements: a Cron expression for setting the frequency and a shell command for execution.

In our application, the task scheduler is configured within the `app/Console/Kernel.php` file's 'schedule' method. This method in the `App\Console\Kernel` class of the web application contains all the scheduled tasks. The process involves SSH-ing into the server to add Cron entries and specify the paths for both the server's remote directory and the local directory for input files.

To keep things straightforward, we've set the scheduling frequency of the web application

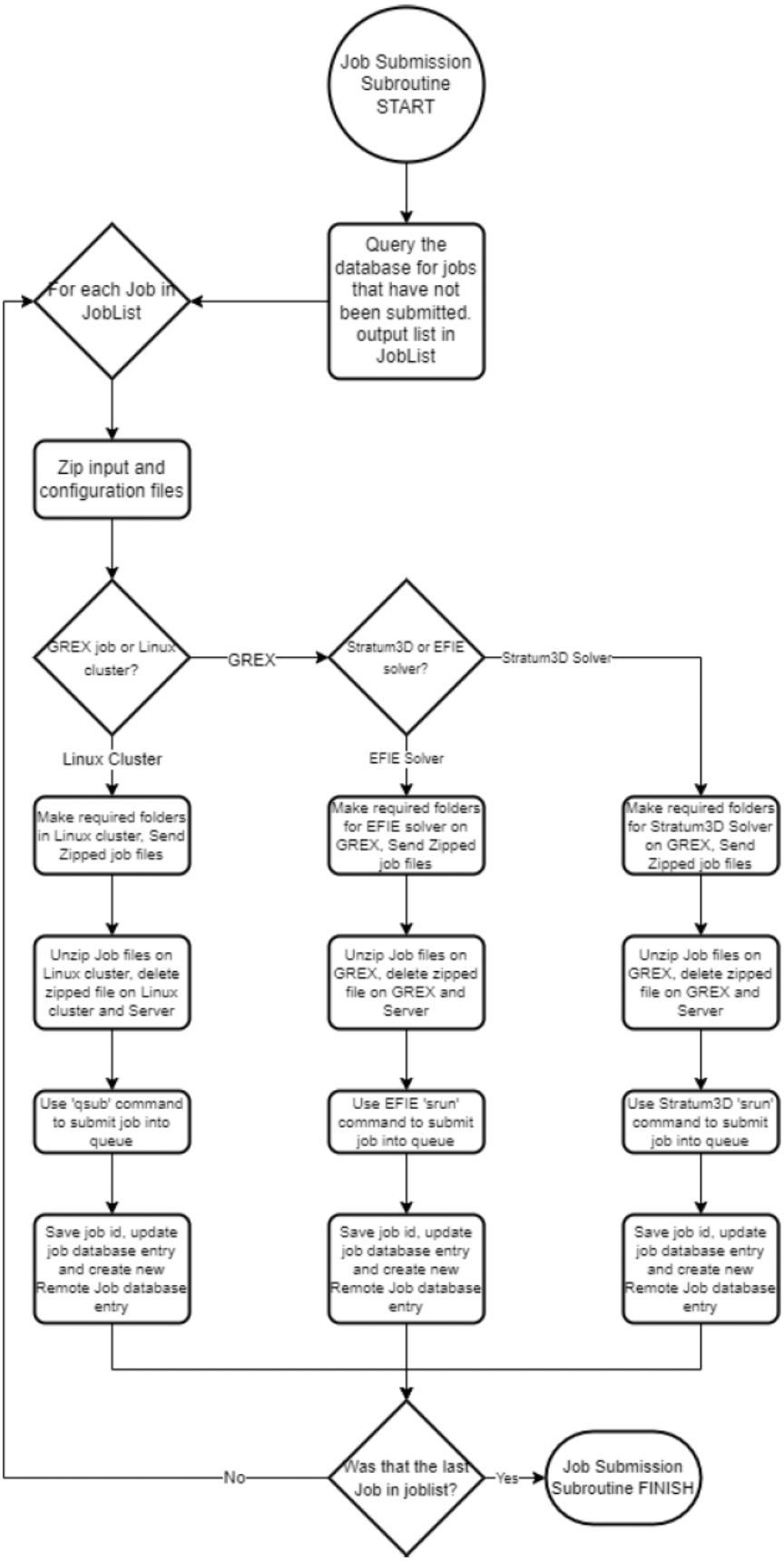

Figure 6.8: Job Submission Flow

to run tasks every minute. This means that the web application activates its tasks at one-minute intervals.

6.6.2 Job Submission Architecture

The job submission process is a crucial routine executed by the scheduler in `kernel.php`, running every minute. Its main objective is to enqueue new jobs onto the Linux cluster or GREX. As depicted in the flow chart in Fig6.8, this routine is initiated by checking the server's database for newly submitted, unqueued jobs. It identifies these by looking for jobs marked as 'Pending' in their progress status, which is the default state for new submissions. The routine then cycles through these jobs, preparing each for submission by compressing its input and configuration files.

The subroutine then gathers specific details about each job. It identifies the desired computational resource (Linux cluster or GREX) by referencing the job's linked `sshserver` database table. The type of solver required for each job, for example, SVS-EFIE or Stratum3D, is determined by the `jobs_solvers` attribute.

The job is then queued for execution—using the `qsub run.pbs` command on the Linux cluster and the `squeue` command for GREX, depending on the solver type. The job's ID is recorded in the database under the `remote_id` attribute of the `remotejobs` table, linked to the job, and the job's status is updated to pending while the remote state is set to queued.

This sequence repeats for each job in the list. If there are no more jobs or if the list was initially empty, the routine ends, conserving server resources. The detailed explanation of the particular classes can be found in Appendices E and F.

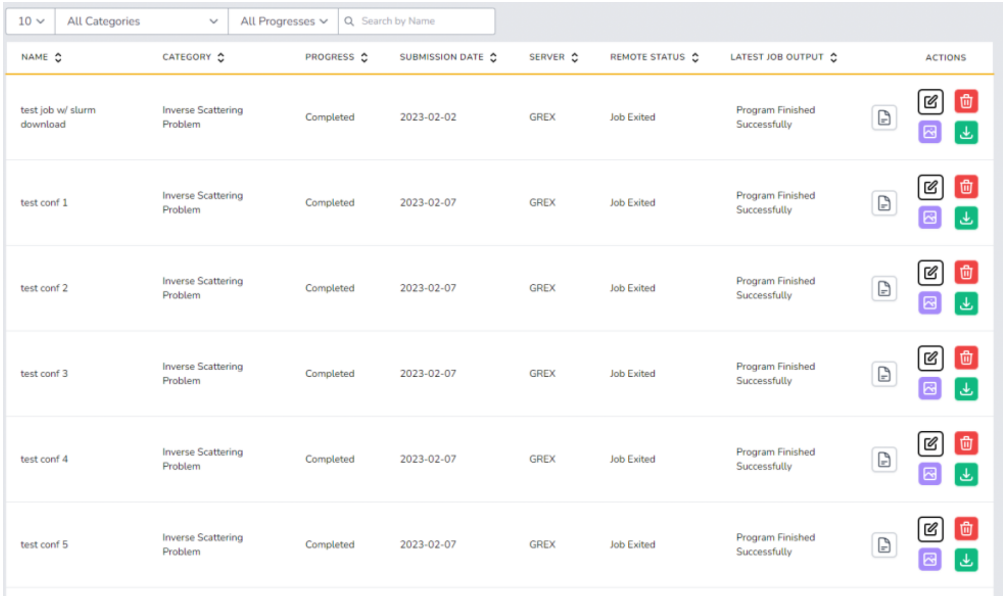

Figure 6.9: User's Submitted Job Through SVS-Web Application

6.6.3 Results Visualization

The output results are presented in vtk format, displaying the distribution of current across the model. VTK, short for Visualization Toolkit, is a freely available open-source tool created by Kitware[92], and designed for the visualization of scientific models. Vtk files, which can be processed and visualized using ParaView[93], another application from Kitware, include metadata such as the vtk version and data type, followed by the substantive data. VTK.js[94] is a JavaScript library designed for scientific visualization within web browsers. We have utilized this library for visualizing jobs on our platform.

Users have the option to view vtk/mesh files using the preview feature. For instance, when users select the preview button, the web application directs them to a page titled 'VTK visualizer'. The 'View Output' button triggers the showVTKmodel function, which requires the JobID, Solver type, and an existing VTK path, if available. When a VTK path is already specified, this function directly loads the visualization page using that path. In cases where the VTK path isn't provided, the vtk representation of the output file needs to be created.

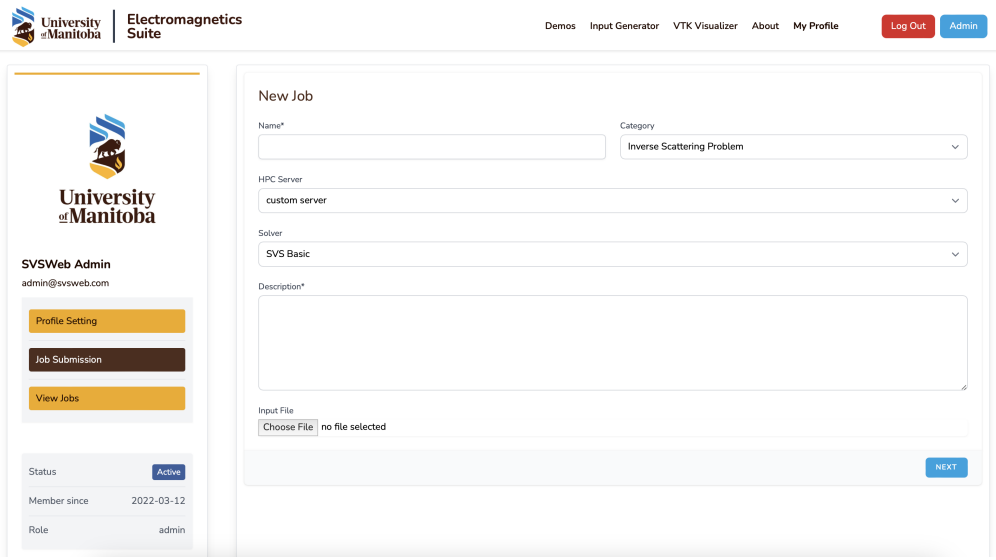

Figure 6.10: Job Submission page

After the VTK file is saved, its new path is recorded in the job's details for quicker access in subsequent visualizations. Subsequently, the user is redirected to the visualization page, where the VTK path is used.

6.7 Submission process from user profile page

After logging into the account, the user will be directed to their user profile page. From there, they can navigate to the "Job Submission" section, typically located in the menu on the left side of the page. Clicking on "Job Submission" will initiate the submission process.

In the submission form (Figure 6.10), the user needs to provide information about the simulation, such as the name and category of the job, select the appropriate HPC server for running the job, and choose the solver to be used. Once all the necessary information is filled out, the user will reach the end of the form.

At this stage, the user is required to upload the master input file, often named "input.input". After uploading the file, the user can proceed to the next step. In the subse-

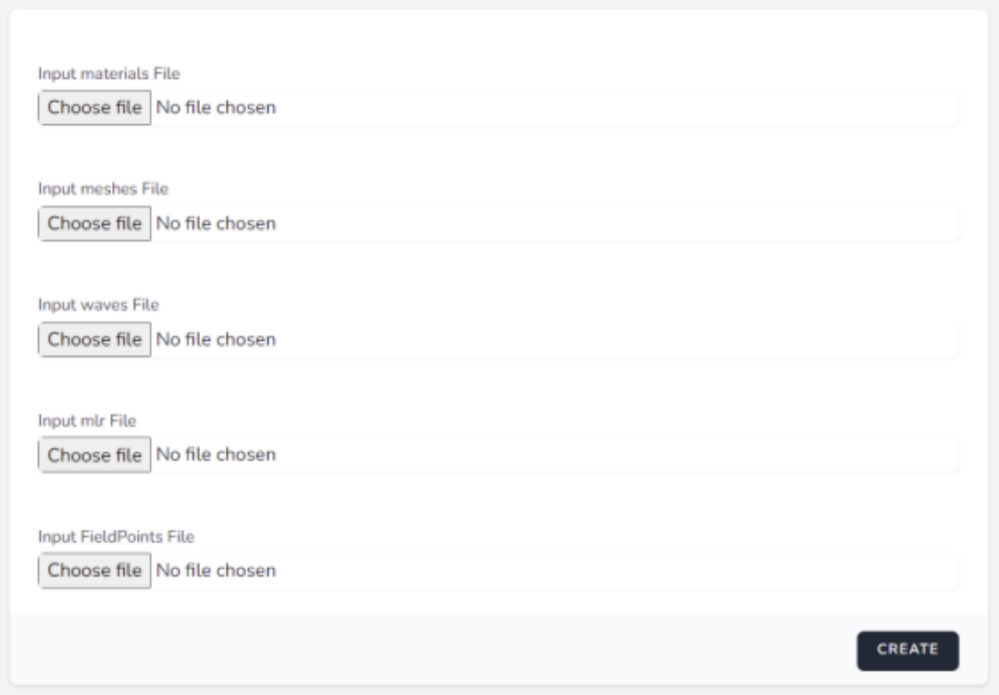

Figure 6.11: Jub's input files selection step

quent step, the user will select additional files related to the simulation (Figure 6.11), such as material files, multilayer files, and wave files. These files are essential for configuring and running the simulation accurately.

Once all the required files are selected, the user can submit the job. Upon submission, the job will be queued in the HPC server's job queue and processed accordingly.

Chapter 7

Conclusion and Further Developments

7.0.1 Convlusion

The Electromagnetic Solver web application's primary objective is to offer an intuitive platform for users to submit, monitor, and retrieve results from jobs. The website is equipped with various functionalities, including demos, an input configuration generator, user profiles, a VTK visualization tool, as well as about and account pages. The demo section showcases various EM examples, which users can download and preview, and these examples can be modified by an admin in the website's backend. The input configuration generator enables users to efficiently create configuration files for the converter and solver, with its appearance and parameters adjustable by the admin through a JSON configuration file located in the website's root directory, without the need to alter the source code.

Upon authentication, users gain access to their profile page, which includes the job submission feature. This page allows users to upload geometry files in Polydata format for visualization in VTK or to submit jobs for converter execution, with automatic redirection to the VTK visualizer. The backend is designed for administrative tasks, enabling site-wide modifications.

The process of online job submission to the solvers involves two key stages. Initially, when a user submits a job, its input files are saved in the database, and solver dependencies are stored on the server. The Laravel scheduler handles the transfer of these files to the server. Once on the server, a script activates the solver, and the job enters a waiting queue. Subsequently, after the solver completes the job, the server sends the output files back to the web application.

7.0.2 Future Works and Suggestions

Looking ahead to the future advancements in Prof. Okhmatovski's research group, I propose a strategic initiative to elevate our framework by seamlessly amalgamating the five existing electromagnetic solvers developed within the group. The objective is to create a unified and comprehensive suite, tailor-made for effortless integration into a myriad of user applications.

Furthermore, We recommend exploring the prospect of integrating our electromagnetic solvers with Amazon AWS, a paramount player in the domain of cloud computing. This integration holds the potential to empower users by allowing them to deploy our solvers on their individual Amazon AWS accounts. This strategic move opens up a versatile and potent avenue for executing simulations on a robust cloud computing platform.

This proposed integration isn't just a technological evolution; it's a paradigm shift. Users would gain the capability to scale their simulations dynamically, adapting to the computational demands of diverse projects. The pay-as-you-go model inherent in AWS ensures a cost-effective approach, with users only incurring charges for the specific computing resources utilized during simulation runs.

Imagine a scenario where a comprehensive solver suite is not just a collection of tools but a seamlessly interconnected system, governed by a user-friendly interface. This graph-

ical user interface becomes the command center, where users, regardless of their technical background, can efficiently orchestrate simulations, navigating the complexities of electromagnetic phenomena with ease.

In tandem with this technological leap, the research group can embark on an outreach initiative, providing comprehensive documentation and educational resources. Workshops and webinars can serve as platforms to not only disseminate knowledge but also foster a community of users adept at harnessing the capabilities of our integrated suite.

In the realm of scientific progress, these proposed enhancements are not merely incremental; they represent a transformative leap forward. This vision of a seamlessly integrated suite, coupled with AWS integration, positions this project at the forefront of innovation, where the boundaries of simulation capabilities are continually pushed, and users are empowered to explore new frontiers in electromagnetic research.

Appendix A

An Example of the input.input file for SVS-EFIE solver

```
    IE-Solver (EFIE || SVS-EFIE):
SVS-EFIE

Name of the material file:
_INPUT/materials/plasma.txt

Name of *.mesh file:
_INPUT/meshes/Model_microstrip.msh
1.00  1.00  1.00
0.0  0.0  0.0

Name of file with incident field information:
_INPUT/waves/k_1pt_50GHz_phi0_theta90.txt

Integration type ("Galerkin" || "Collocation"):
Galerkin
```

```
17
18 Preconditioner type ("Right" || "Left" || "CMP" || "No"):
19 No
20
21 Matrix solver ("LU" || "HLU" || "GMRES" || "HGMRES"):
22 LU
23
24 LossModel Type ("Leontovich" || "sigmaT" || "NOLOSS"):
25 NOLOSS
26
27 Excitation ("PlaneWave" || "Dipole" || "PortS"):
28 PortS
29
30 Plane wave polarization (1-"Theta" || 2-"Phi"):
31 1
32
33 Dipole location (1-"[x y z] in meters" || 2-"[Theta Phi r] in angle and
      multiple of Wavelength"):
34 2  30  180  1000000
35
36 Dipole orientation ([x y z] in meters):
37 0.866  0.0  0.5
38
39 Number of surface ports:
40 1
41 points
42 MicrostripPort
43
44 Surface port polylines ([x y z] in meters):
45 9
```

```
1515
7709
1516
7710
1513
7707
1514
7798
1515

Name of file with RWG coefficients ("Name" || "None"):
None

Kernel ("FreeSpace" || "MultiLayer" || "HalfSpace"):
MultiLayer
0.000

Name of file with Multilayer description ("Name" || "None"):
_INPUT/mlr/plasma_true_layers.mlr

Cubature order to calculate interactions (-1 means orders will be chosen
    according to distance between src & obs):
3

Singularity extraction limit (centre-to-centre distance of src-obs
    triangle pair is larger than N*largest_edge):
4.0

Is computation of scattered field required ("Yes" || "No"):
No
```

```
74
75 Is computation of current at surface points required ("Yes" || "No"):
76 No
77
78 Name of file with observation points ("Name" || "None"):
79 None
80
81 Name of file with observation points for current at the surface ("Name" ||
       "None"):
82 None
83
84 Name of file with RWG definitions ("_INPUT/Processed/RWGinfo.prn" || "None
      "):
85 None
86
87 Name of file with MoM matrix("matrix_mcd" || "None"):
88 None
89
90 Name of file with RHS("RHS_mcd" || "None"):
91 None
92
93 GMRES tolerance:
94 1E-5
95
96 Max # of GMRES Iterations:
97 1000
98
99 GMRES restart:
100 20
101
```

```
ACA or SVD tolerance:
1E-6

H-arithm tolerance ("RELATIVE" || "ABSOLUTE"):
RELATIVE
1E-1

Number of levels in H-Matrix Cluster Tree:
20

Minimum number of elements per leaf node:
32

Eta admissibility condition:
2

Compression type ("ACA" || "GRID" || "SVD"):
SVD

Grid type ("Fixed", "Wavelength" (dictated by Next parameter. min=3)):
Fixed

Number of grid points (X,Y,Z):
4
4
4

Want "one" additional grid point outside the bounding box?:
No

```

```
Interpolation type ("Lagrange"):
Lagrange

Stencil size (X,Y,Z or (-1 -1 -1) - means the same as Num grid points):
-1
-1
-1

Output Matrices ("Yes" || "No"):
No

Clustering Strategy ("0" - Default, "1" - Geometrically balanced, "2" - Geometrically regular (NOT WORKING)
                     "3" - Geom. regular, regular boxes (NOT WORKING), "4" - Cardinality based, "5" - Principal directions):
4

Block Admissibility Criteria ("0" - min adm criterion (default), "1" - weak adm, "2" - max adm):
0

Block Homogeneity ("0" - homogeneous (default), "1" - inhomogeneous):
0

MinRank SVD or ACA:
5

Hmatrix element bounding box ("TRI" - push all 4 points of the triangle pair || "RWG" - push only RWG edge endpoints || "RWG_CENTRE" - only RWG -edge centre):
```

```
157 TRI
158
159 OutputCurrent format ("VTK" || "MATLAB" || "BOTH"):
160 VTK
```

Listing A.1: An Example of the input.input file for SVS-EFIE solver

Appendix B

Parabolic dish antenna - Gmsh scripts

```
SetFactory("OpenCASCADE");
Lc = 0.0068;
R = 0.6699/2; // Major Axis
R1 = 0.5699/2;// Minor Axis
H = 0.1;
F = (4*R*R)/(16*H);
Point (1) = {0, 0, 0, Lc};
Point (2) = {R, 0, H, Lc};
Point (3) = {R/2, 0, 0};
Point (4) = {0, R1, H, Lc};
Point (5) = {0, R1/2, 0};
Point (6) = {-R, 0, H, Lc};
Point (7) = {0, -R1, H, Lc};
Point (8) = {-R/2, 0, 0};
Point (9) = {0, -R1/2, 0};
Point (10) = {0, 0, H};
Point(14) = {Lc/12, 0, F, Lc};
Point(15) = {-Lc/12, 0, F, Lc};
Point(16) = {0, Lc/12, F, Lc};
```

```
Point(17) = {0, -Lc/12, F, Lc};
Line(17) = {14, 17};
Line(18) = {17, 16};
Line(19) = {16, 14};
Line(20) = {16, 15};
Line(21) = {15, 17};
Bezier (1) = {1,3,2,2};
Bezier (2) = {1,5,4,4};
Bezier (3) = {1,8,6,6};
Bezier (4) = {1,9,7,7};
Ellipse(22) = {4, 10, 2, 2};
Ellipse(23) = {4, 10, 6, 6};
Ellipse(24) = {7, 10, 2, 2};
Ellipse(25) = {6, 10, 7, 7};
Curve Loop(7) = {22, -1, 2};
Surface(7) = {7};
Curve Loop(9) = {23, -3, 2};
Surface(8) = {9};
Curve Loop(11) = {25, -4, 3};
Surface(9) = {11};
Curve Loop(13) = {4, 24, -1};
Surface(10) = {13};
Physical Surface(1) = {7, 8, 9, 10};
Recursive Delete {
    Point{3};
    Point{5};
    Point{8};
    Point{9};
    Point{10};
}
```

```
50 Curve Loop(5) = {19, 17, 18};
51 Plane Surface(5) = {5};
52 Curve Loop(6) = {20, 21, 18};
53 Plane Surface(6) = {6};
54 Physical Surface(100) = {5, 6};
```

Listing B.1: Parabolic dish antenna scripts in Gmsh (.geo file format)

Appendix C

Multilayer File Scripts

```
Number of layers: 7
Maximum number of rho samples: 1000

Layer: 6
Top interface [meter]: 0.000848
Permittivity [unitless]: 1.0e+0
Conductivity [Sm/meter]: 0.0e+0
Tan delta [unitless]: -100.0
Permeability [unitless]: 1.0e+0
Bottom interface [meter]: 0.000848
Maximum number of z samples: 20

Layer: 5
Top interface [meter]: 0.000848
Permittivity [unitless]: 1.0e+0
Conductivity [Sm/meter]: 0.0e+0
Tan delta [unitless]: -100.0
Permeability [unitless]: 1.0e+0
Bottom interface [meter]: 0.000748
```

```
Maximum number of z samples: 20

Layer: 4
Top interface [meter]: 0.000748
Permittivity [unitless]: 3.2e+0
Conductivity [Sm/meter]: 0.0e-3
Tan delta [unitless]: 0.035
Permeability [unitless]: 1.0e+0
Bottom interface [meter]: 0.000562
Maximum number of z samples: 20

Layer: 3
Top interface [meter]: 0.000562
Permittivity [unitless]: 4.2e+0
Conductivity [Sm/meter]: 0.0e-3
Tan delta [unitless]: 0.035
Permeability [unitless]: 1.0e+0
Bottom interface [meter]: 0.000162
Maximum number of z samples: 20

Layer: 2
Top interface [meter]: 0.000162
Permittivity [unitless]: 3.2e+0
Conductivity [Sm/meter]: 0.0e-3
Tan delta [unitless]: 0.035
Permeability [unitless]: 1.0e+0
Bottom interface [meter]: 0.0000
Maximum number of z samples: 20

Layer: 1
```

```
Top interface [meter]: 0.0000
Permittivity [unitless]: 1.0e+0
Conductivity [Sm/meter]: 0.0e+0
Tan delta [unitless]: -100.0
Permeability [unitless]: 1.0e+0
Bottom interface [meter]: -0.0001
Maximum number of z samples: 20

Layer: 0
Top interface [meter]: -0.0001
Permittivity [unitless]: 1.0e+0
Conductivity [Sm/meter]: 0.0e+0
Tan delta [unitless]: -100.0
Permeability [unitless]: 1.0e+0
Bottom interface [meter]: -0.0001
Maximum number of z samples: 20

DCIM Parameters (CUSTOM/DEFAULT):
CUSTOM

Number of levels in DCIM:
1

Number of DCIM Images (should be even for two-level DCIM):
12

Tolerance for truncating spectrum:
1E-5

```

```
79 Factor to increase the density of the grid in rho-dimension for S and V
       elements (GridDensityS, GridDensityV):
80 1.0   1.0
```

Listing C.1: An Example of Multilayer File

Appendix D

Explanation of Site Navigation Blade File

The provided code is a segment written in Blade, a templating engine used in Laravel. It constructs a navigation bar with features such as a responsive logo display, site title, mobile-friendly hamburger menu, and navigation links. The integration of Alpine.js facilitates dynamic behavior, allowing for the toggling of the navigation menu's visibility. The navigation bar includes links to various sections of the site, such as "Demos," "Input Generator," "VTK Visualizer," "About," and user-related actions like "Login," "Register," and "Logout." The visibility of the navigation links is controlled by the 'open' variable, making it responsive for both mobile and desktop views. Tailwind CSS classes are employed for styling.

The navigation bar is included on all pages except the error pages like the 404 page.

Here is an explanation of the classes used:

D.1 Alpine.js Integration

```
<div x-data="{ open: false }" class="nav-bar--sticky sticky top-0 z-50 bg-white pt-2">

```

The ‘x-data=“ open: false ”‘ attribute integrates Alpine.js, initializing a data variable ‘open‘ for dynamic behavior. The ‘class=“nav-bar–sticky sticky top-0 z-50 bg-white pt-2”‘ applies various styles, such as making the navigation bar sticky, positioning it at the top, and setting background color.

D.2 Navigation Bar Structure

```
<div class="px-4 lg:px-8 md:w-1xl lg:mx-auto lg:flex lg:flex-row md:items-center md:justify-between">

```

The ‘class=“px-4 lg:px-8 md:w-1xl lg:mx-auto lg:flex lg:flex-row md:items-center md:justify-between”‘ sets padding, width, and flexbox layout for the navigation bar.

D.3 Logo and Site Title

```
<div class="flex md:flex-row md:items-center items-start flex-row">
    <picture class="py-0">
            <source media="(min-width: 350px)" srcset="{{url('/images/UM-logo-horizontal-CMYK.png')}}" alt="{{ __('University of Manitoba') }}" width="150">
            <img src="{{url('/images/UM-logo-vertical-CMYK.png')}}" alt="University of Manitoba" width="90">
     </picture>
```

```
6         <a href="/"
7                 class="site-title inline-block text-2xl ml-5 py-1 pl-5
    border-l-4 border-gray-700 border-opacity-85">
8                  <h1 class="site-title--main font-bold text-2x1"> {{ __('
    Electromagnetics') }} <br>{{ __('Suite') }}</h1>
9         </a>
10    </div>
11
```

The 'class="flex md:flex-row md:items-center items-start flex-row"' applies flexbox styling to arrange the logo, site title, and hamburger icon in a row on larger screens.

D.4 Hamburger Icon and Mobile Navigation

```
1   <div class="ml-5 py-5 inline-block cursor-pointer lg:hidden" @click="
    open = !open">
2       <!-- ... -->
3   </div>
4
```

The 'class="ml-5 py-5 inline-block cursor-pointer lg:hidden"' sets margin, padding, and visibility based on screen size. The '@click="open = !open"' handles the toggle of the 'open' variable when clicked, controlling the visibility of the navigation menu.

D.5 Navigation Links

```
1   <div :class="{ 'hidden': !open }" class="hidden lg:block">
2       <x-site-link href="{{ url('/demos') }}"
```

```
3                               :active="request()->routeIs('demos') ||
    request()->getPathInfo() == '/demos'">
4                       {{ __('Demos') }}
5                   </x-site-link>
6                   <x-site-link href="{{ url('/input-generator') }}"
7                               :active="request()->routeIs('input-generator')
    || request()->getPathInfo() == '/input-generator' || request()->routeIs
    ('xml-input-generator')">
8                       {{ __('Input Generator') }}
9                   </x-site-link>
10                  <x-site-link href="{{ url('/vtk-visualizer') }}"
11                              :active="request()->routeIs('vtk-visualizer')
    || request()->getPathInfo() == '/vtk-visualizer'">
12                      {{ __('VTK Visualizer') }}
13                  </x-site-link>
14                  <x-site-link href="{{ url('/about') }}" :active="request()
    ->getPathInfo() == '/about'">
15                      {{ __('About') }}
16          </x-site-link>
17      </div>
18
```

The ':class="'hidden': !open"' applies the 'hidden' class dynamically based on the truthiness of the 'open' variable. The 'class="hidden lg:block"' ensures that the navigation links are hidden on smaller screens and visible on larger screens.

This code also includes Blade directives ('@if', '@auth', '@endif', etc.) for conditional rendering based on user authentication and role. The Blade syntax is specific to the Laravel framework and is used for server-side rendering.

Appendix E

Explanation of Job Schedule Class

The JobSchedule class plays a crucial role in coordinating job execution and overseeing file management on remote servers. Its methods encompass vital functionalities, including job submission, status updates, and the secure transfer of files. The UML diagram (Figure E.1) depicts the JobSchedule class as a central element in the system architecture. Its responsibilities include the management and scheduling of jobs, interaction with remote servers, and the facilitation of file transfers. The diagram serves as a transparent representation, elucidating the structural composition and operational aspects of the class.

E.1 Submit Method

The 'submit' method within the 'JobSchedule' class is integral for orchestrating the submission of jobs to remote servers. It encompasses a series of steps tailored to the specifics of the server type and the nature of the job, ensuring a seamless execution process.

The 'submit' method encapsulates a comprehensive set of actions required for the successful submission of jobs to both Linux clusters and the Grex servers. It adapts its behavior based on the server type and the nature of the job, handling file uploads, unzipping, remote

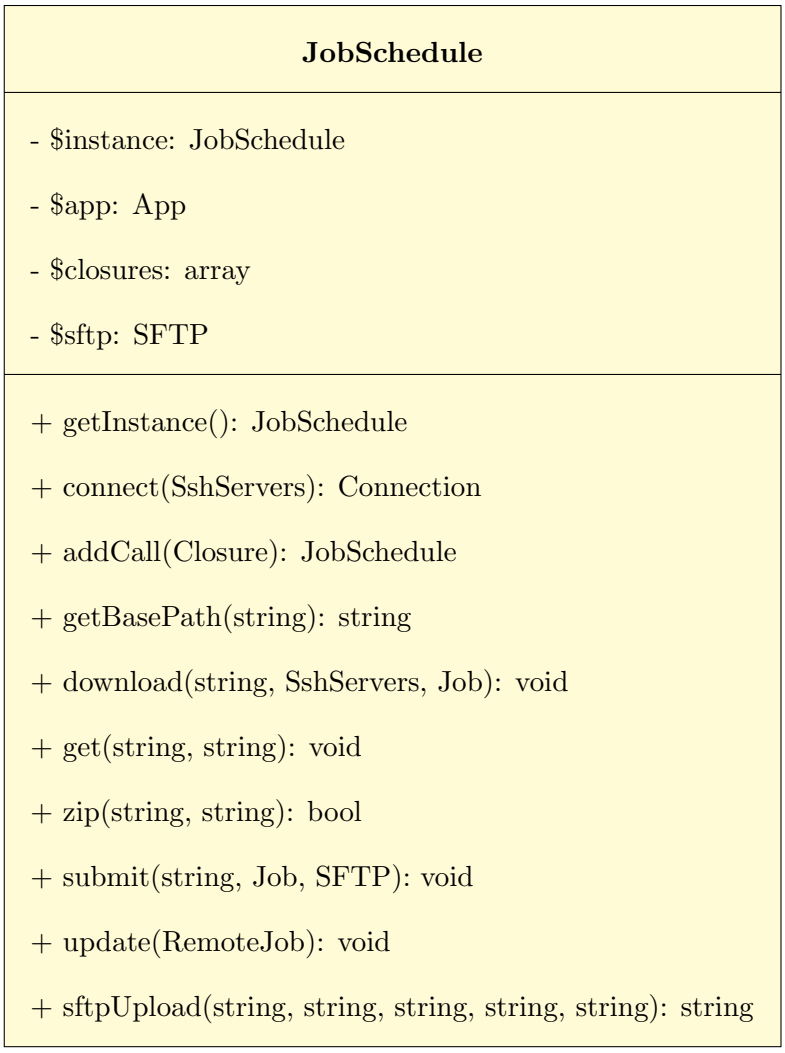

Figure E.1: JobSchedule Class UML Diagram

command execution, and status updates. The method's modular design allows for extensibility and customization based on specific requirements.

E.1.1 Parameters

$filePath (string): The local file path where the job files are stored.

$job (Job): An instance of the 'Job' class containing job-related information.

$sftp (SFTP): An instance of the 'SFTP' class, presumably facilitating secure file transfers.

E.1.2 Execution Steps

E.1.2.1 Initialization and Echo Statement

The method begins with an echo statement signaling the initiation of the submission process. This serves as a helpful marker for monitoring and debugging.

```
1     echo "starting submit\n";
2
```

E.1.2.2 Initialization

Instantiates the 'App' class to access application-specific configurations and settings.

```
1     $app = new App();
2
```

E.1.2.3 Server Type Switch

Utilizes a switch statement based on the type of SSH server associated with the job.

```
1    switch ($job->sshservers->first()->id) {
2
```

E.1.2.4 Linux Cluster (Server Type 1)

In the scenario where the server type is a "linux cluster" (case 1), the method verifies whether the job pertains to a Stratum3D solver

```
1    case 1: // linux cluster
2        if ($job->jobs_solvers == 2) {
3
```

E.1.2.5 Job File Paths

Constructs local and remote paths for the zipped job file.

```
$jobFileLocal = realpath(sprintf('%1$s/%2$s/%2$s.zip', public_path('storage/jobs'), $job->id));
$jobDir = str_replace('{JOB_ID}', $job->id, $app->getAttribute('remote_dir'));
$jobFileRemote = sprintf('%s/%s.zip', $jobDir, $job->id);

```

E.1.2.6 File Upload and Unzipping

Uploads the zipped job file to the remote server using SSH. Finds and unzips the file on the remote server using SSH commands.

```
SSH::put($jobFileLocal, $jobFileRemote);
// ... (unzipping logic)
 $cmd = [];
 $cmd[] = sprintf('find %s', $jobFileRemote);
 $cmd[] = implode('; ', [
     'while read filename',
     sprintf('do unzip -o -d 'dirname %s' "$filename"', $jobFileRemote),
     'done;'
 ]);
 $cmd = implode(' | ', $cmd);
 SSH::run([$cmd], function ($line) {
     echo ($line);
 });


```

E.1.2.7 Remote Batch Job Submission

Triggers the remote batch job submission using the 'qsub' command.

```
$c = JobSchedule::connect($server);
$c->run([sprintf('qsub solver.pbs -v var=%s', $job->id)], function ($line) use ($job) {
    // ... (callback logic)
}, 30);

```

E.1.2.8 Status Update and Save

Updates the job status and saves relevant information.

```
$job->progress = Job::STATUSES[1];
$job->save();

```

E.1.3 Case: Grex (Server Type 2)

Distinguishes if it's Stratum3D solver for the Grex server and follows the same procedure as it was explained for Linux Cluster.

```
 case 2: // grex
     if ($job->jobs_solvers == 1) {

```

E.2 Update Method

The "update" method is responsible for managing the status update of a remote job based on the type of SSH server it is associated with. This method specifically handles updates for jobs on a "linux cluster" (case 1) and "GREX" (case 2). Below is an explanation of the code:

E.2.1 Echo Statements

The method starts with echo statements to provide information about the beginning of the update process and the type of job being processed (linux cluster or GREX).

```
1         echo "starting update\n";
2
```

E.2.2 Switch Statement

The switch statement is used to handle different cases based on the SSH server's type associated with the remote job.

```
1         switch ($remoteJob->job->sshservers->first()->id) {
2             case 1: // linux cluster
3                 // ...
4                 break;
5             case 2: // GREX
6                 // ...
7                 break;
8         }
9
```

E.2.2.1 In the case of a “linux cluster” (case 1)

The method starts by creating a command template designed for querying the job state, utilizing the “qstat” command. In this template, placeholders are strategically positioned to later accommodate the specific remote job ID. Subsequently, the actual command is crafted by substituting these placeholders with the unique identifier of the remote job.

To execute this command on the remote server, the method relies on the “run” method. This step is crucial for obtaining real-time information about the job’s status and progression. If, during this inquiry, the job is not found and is marked as ‘Unknown,’ this implies that the job has reached completion or has been deleted.

In response to this scenario, the method takes a series of actions. Firstly, it initiates the download of pertinent job files using the ‘JobSchedule::download’ method, ensuring the retrieval of relevant data. Following this, it proceeds to update the local job status, committing these changes to the database for future reference. Simultaneously, the state of the remote job is adjusted to “E,” signifying its completion, and these alterations are saved to maintain an accurate record.

However, if the job is found to be still in progress, the method dynamically adapts the state of the remote job to reflect its ongoing status. This ensures that the system remains synchronized with the real-time progress of the job on the remote server.

```
1     case 1: // linux cluster
2         $commandTemplate = "qstat -f %s | grep job_state | awk -F ' ' '{
    print \$NF}'";
3         $command = sprintf($commandTemplate, str_replace(["\n", "\r"], '',
     $remoteJob->remote_job_id));
4         $server = $remoteJob->job->sshservers->first();
5         $c = JobSchedule::connect($server);
6         $c->run([$command], function ($line) use ($server, $remoteJob) {
```

```
7               // ...
8           });
9           break;
10
```

E.2.2.2 In the case of "GREX" (case 2)

The method initiates by constructing a command designed to query information about the job using the "squeue" command. Two distinct commands are formulated based on the nature of the job, distinguishing between configuration solvers and input solvers. Each command is tailored to address the specific Slurm output location corresponding to its job type.

Execution of both the Slurm and squeue commands on the remote server is facilitated through the "run" method, allowing for simultaneous retrieval of relevant data. The method then meticulously processes the output of the Slurm command, dynamically updating the local job's verbose status by extracting information from the last lines of the output.

Following this, the output of the squeue command is analyzed to determine the job's state. If, during this assessment, the job is not located and is designated as 'Unknown,' this signifies that the job has reached completion or has been deleted. In response to this scenario, the method meticulously orchestrates a series of operations. Firstly, it triggers the download of pertinent job files to ensure the retrieval of crucial information. Subsequently, it proceeds to update the local job status, effecting these changes in the database for accurate record-keeping. Simultaneously, the state of the remote job is modified to "E," denoting its completion, and these adjustments are saved to maintain a comprehensive historical record.

However, if the job is discerned to be still in progress, the method adeptly adapts the state of the remote job based on the information gleaned from the squeue command. This

ensures that the system remains dynamically synchronized with the real-time progress of the job on the remote server.

```
1     case 2: // GREX
2         $command = "squeue -u " . $server->username . " | grep " .
    $remoteJob->remote_job_id . ' || echo "Unknown"';
3         $slurmCommand = ($remoteJob->job->jobs_solvers == 1)
4             ? "tail --lines=10 " . "/home/" . $server->username . "/
    Executable_CFIEHFMM_serial/slurm-" . $remoteJob->remote_job_id . ".out"
5             : "tail --lines=10 " . self::getBasePath($server->username) .
    $remoteJob->job->id . "/slurm-" . $remoteJob->remote_job_id . ".out";
6         $c->run([$slurmCommand], function ($line) use ($server, $remoteJob
    ) {
7             // ...
8         });
9         $c->run([$command], function ($line) use ($server, $remoteJob) {
10            // ...
11        });
12        break;
13
```

E.2.3 Output Handling

The method captures the output of the executed commands and processes it to determine the status of the remote job.

```
1         if (str_contains($result, 'Unknown')) {
2             $job = $remoteJob->job;
3             JobSchedule::download($job->id, $server, $job);
4             $job->previous_progress = $job->progress;
5             $job->progress = Job::STATUSES[2];
```

```
6               $job->save();
7               $remoteJob->job_state = "E";
8               $remoteJob->save();
9           } else {
10              $remoteJob->job_state = str_replace(["\n", "\r"], '', $result)
    ;
11              $remoteJob->save();
12          }
13
```

E.2.3.1 Job and RemoteJob Updates

Throughout the process, the method updates attributes of both the local job and the remote job, reflecting changes in job status, progress, and additional information.

```
1           $job->save();
2           $remoteJob->save();
3
```

The method provides a comprehensive mechanism for updating the status of remote jobs on different types of servers, including handling completion, deletion, and ongoing execution scenarios.

E.3 Zip Method

The primary goal of this function is to compress files on the local machine and facilitate their transfer to a remote server. The detailed description emphasizes its pivotal role in the compression of files in preparation for subsequent transmission. Now, let's dissect the essential components of the provided code:

The function is equipped with two parameters: '$filepath' signifies the path to the local directory containing the files earmarked for compression, while '$fileName' designates the name to be assigned to the resultant zip file. Notably, the default value for '$fileName' is intelligently set to 'zip.'

To systematically gather files, a recursive iterator is employed, navigating the directory structure adeptly. This iterator is specifically configured to selectively consider leaf nodes, distinguishing individual files from directories.

The boolean variable '$response' is initialized with a value of true. Its purpose becomes apparent as it becomes the indicator of the overall success or failure of the zip operation.

An instance of the ZipFile class is instantiated to take charge of the compression process. The code bravely endeavors to integrate files from the iterator into the zip archive, subsequently saving it with a name formed by appending ".zip" to the specified name.

Should a ZipException rear its head, signaling an anomaly during the compression procedure, the resilient response mechanism kicks in—setting '$response' to false. Irrespective of the operation's outcome, the 'finally' block ensures the proper closure of the ZipFile instance, exemplifying prudent resource management.

In the ultimate act, the function gracefully returns the '$response' variable, effectively communicating the triumph or setback encountered during the zip operation.

```
1    public static function zip(string $filepath, string $fileName = 'zip')
     : bool
2    {
3        $rootPath = $filepath;
4        $files = new \RecursiveIteratorIterator(
5            new \RecursiveDirectoryIterator($rootPath),
6            \RecursiveIteratorIterator::LEAVES_ONLY
7        );
```

```
8         $response = true;
9         $zipFile = new ZipFile();
10        try {
11            $zipFile->addFilesFromIterator($files)->saveAsFile($rootPath.
    $fileName.".zip");
12        } catch(ZipException $e) {
13            $response = false;
14        } finally {
15            $zipFile->close();
16        }
17
18        return $response;
19     }
20
```

In summary, this meticulously crafted function encapsulates the intricate dance of compressing files from a designated local directory, adorning the resultant archive with a user-defined or default name. Its judicious use of exception handling elevates its robustness, and the boolean return value serves as a transparent messenger of the compression outcome.

E.4 Get Method

This function plays a pivotal role in facilitating the upload of input files from the local machine to a remote server. The intricate process involves several steps, each contributing to the seamless transfer of files. Here's a breakdown of the key elements:

The '`JobSchedule::\$sftp->chdir(\$remotePath);`' function is employed to effectuate a change in the directory on the remote server. This step ensures that subsequent operations are carried out in the designated remote path.

The '`JobSchedule::\$sftp->nlist(".", true);`' function takes center stage in listing all files in the current directory on the remote server. This comprehensive listing serves as the foundation for the subsequent file transfer process.

The '`File::makeDirectory(\$localPath, 0755, true, true);`' function is invoked to create a directory on the local machine. This step ensures that the local destination for file transfer is properly structured.

The code then meticulously traverses through each file in the remote directory. For each file encountered, a series of actions are initiated:

- Exclusion of the directories ".." and "." ensures the code operates only on actual files.
- The directory structure is replicated on the local machine using '`File::makeDirectory`' if the file is not in the root directory.

In essence, this function orchestrates a sophisticated dance of directory navigation, file listing, directory creation, and file transfer. Its robust design ensures the systematic replication of the remote directory structure on the local machine, culminating in the successful upload of files. The code elegantly combines directory management and file transfer operations, showcasing a well-orchestrated mechanism for seamless data transfer between local and remote environments.

```
1    public static function get(string $remotePath, string $localPath)
2    {
3        JobSchedule::$sftp->chdir($remotePath);
4        $files =  JobSchedule::$sftp->nlist(".", true);
5        File::makeDirectory($localPath, 0755, true, true);
6
7        foreach ($files as $file) {
8            if ($file != ".." && $file != ".") {
```

```
                $a = dirname($file);
                if ($a != ".") File::makeDirectory($localPath . $a, 0755, true, true);
                JobSchedule::$sftp->get($remotePath . $file, $localPath . $file);
            }
        }
    }

```

E.5 Download Method

This static function, known as "download," plays a pivotal role in the essential task of transferring files from a remote server to the local machine within the context of job management and associated data transfers. The breakdown of its key components reveals a comprehensive and adaptable approach.

E.5.1 Initialization and Connection

The function initiates an instance of the App class, presumably serving as a configuration and settings handler. It then ensures the presence of critical parameters, specifically checking for the availability of $jobID and $server. The subsequent establishment of an SFTP connection utilizes the provided server details, including host, port, username, and password.

E.5.2 Conditional Handling for Job Types

The function incorporates conditional statements contingent on the job type (`$job->jobs_solvers`). For configuration solver jobs, denoted by `$job->jobs_solvers` equaling 1, it defines dis-

tinct remote and local paths for output and converter-related data. The execution of the `JobSchedule::get` function facilitates the secure download of files from the remote server to the local machine for both sets of paths. An additional conditional check, examining the server ID, discerns GREX jobs (ID 2) and mandates the download of the Slurm output file (slurm-output.out).

For input solver jobs, the determination of the base path for file retrieval precedes the use of `JobSchedule::get` to download output files. Furthermore, a specific function (`ShowForm::gridToPolydata`) is invoked for additional processing on a VTK file (`exc0_CurrentV0.vtk`). The download of the Slurm output file is contingent on the job type and ID, showcasing a nuanced approach to handling diverse job scenarios.

E.5.3 Dynamic Path Construction

Throughout the function, local paths are dynamically constructed using placeholders like `{JOB_ID}`. These placeholders are subsequently replaced with the actual job ID, enhancing the adaptability and modularity of the code.

In summary, the download function emerges as a robust and adaptable mechanism for orchestrating the retrieval of files from a remote server, considering diverse job types. Leveraging SFTP for secure file transfers, the function provides detailed logs and employs conditional logic to ensure appropriate handling for varied job scenarios, reflecting a sophisticated approach to downloading job-related data.

```
1 public static function download(string $jobID, SshServers $server, Job
    $job)
2 {
3     $app = new App();
4
5     // Check if essential parameters are provided
```

```
6     if ($jobID && $server) {
7         // Establish an SFTP connection to the remote server
8         JobSchedule::$sftp = new SFTP($server->host, $server->port);
9
10        // Attempt login with provided credentials
11        if (!JobSchedule::$sftp->login($server->username, $server->
    password)) {
12            exit('Login Failed');
13        }
14
15        // Conditional handling based on job types
16        if ($job->jobs_solvers == 1) { // Configuration solver (
    Executable_CFIEHFMM_serial)
17            // Define remote and local paths for output and converter-
    related data
18            $remotePath = "/home/". $server->username . "/
    Executable_CFIEHFMM_serial/OUTPUT/" . $jobID . "/";
19            $localPath = str_replace('{JOB_ID}', $jobID, $app->
    getAttribute("local_path"));
20
21            // Log and initiate file transfer for output data
22            echo "Moving " . $remotePath . " to " . $localPath . "\n";
23            JobSchedule::get($remotePath, $localPath);
24
25            // Define paths for additional data
26            $remotePath = "/home/". $server->username . "/
    Executable_CFIEHFMM_serial/" . $jobID . "/_OUTPUT/";
27            $localPath = str_replace('{JOB_ID}', $jobID, $app->
    getAttribute("converter_local_path"));
28
```

```
29            // Log and initiate file transfer for converter-related data
30            echo "Moving " . $remotePath . " to " . $localPath . "\n";
31            JobSchedule::get($remotePath, $localPath);
32
33            // Additional check for GREX job and download Slurm output
    file
34            if ($server->id == 2) {
35                $remotePath = "/home/" . $server->username . "/
    Executable_CFIEHFMM_serial/slurm-" . $job->remoteJob->remote_job_id .
    ".out";
36                $localPath = str_replace('{JOB_ID}', $jobID, $app->
    getAttribute("local_path"));
37
38                // Log and initiate Slurm output file transfer
39                echo "Moving " . $remotePath . " to " . $localPath . "\n";
40                JobSchedule::$sftp->get($remotePath, $localPath . "slurm-
    output.out");
41            }
42
43        } else { // Input solver
44            // Determine the base path for file retrieval
45            $basePath = self::getBasePath($server->username);
46
47            // Define remote and local paths for output data
48            $remotePath = $basePath . $job->id . "/_OUTPUT/";
49            $localPath = str_replace('{JOB_ID}', $jobID, $app->
    getAttribute("local_path"));
50
51            // Log and initiate file transfer for output data
52            echo "Moving " . $remotePath . " to " . $localPath . "\n";
```

```
53              JobSchedule::get($remotePath, $localPath);
54
55              // Invoke a specific function for additional processing on a
    VTK file
56              ShowForm::gridToPolydata($localPath . 'exc0_CurrentV0.vtk');
57
58              // Define remote path for Slurm output file based on job type
59              $remotePath = ($job->jobs_solvers == 1) ?
60                  "/home/" . $server->username . "/
    Executable_CFIEHFMM_serial/slurm-" . $job->remoteJob->remote_job_id . "
    .out" :
61                  $basePath . $job->id . "/slurm-" . $job->remoteJob->
    remote_job_id . ".out";
62
63              // Log and initiate Slurm output file transfer
64              echo "Moving " . $remotePath . " to " . $localPath . "\n";
65              JobSchedule::$sftp->get($remotePath, $localPath . "slurm-
    output.out");
66          }
67      }
68  }
69
```

E.6 Get Base Path Method

This private static function, aptly named "getBasePath," fulfills the role of retrieving the base path associated with a given username. The function operates within a context where the dynamic construction of paths is essential, contributing to the adaptability and

flexibility of the overall code.

```
1 private static function getBasePath(string $username) : string
2 {
3     $app = new App();
4     return str_replace(‘{username}’, $username, $app->getAttribute("
    grex_base_path"));
5 }
```

Appendix F

Explanation of Job Show Form Class

The ShowForm class (Figure F.1) encapsulates the logic and functionality for managing job-related data and interactions within a Livewire component. It integrates with other Laravel features, such as file storage, model relationships, and routing. The class is structured to handle various aspects of job management, including updates, deletions, withdrawals, and file downloads, providing a comprehensive user interface for job-related operations.

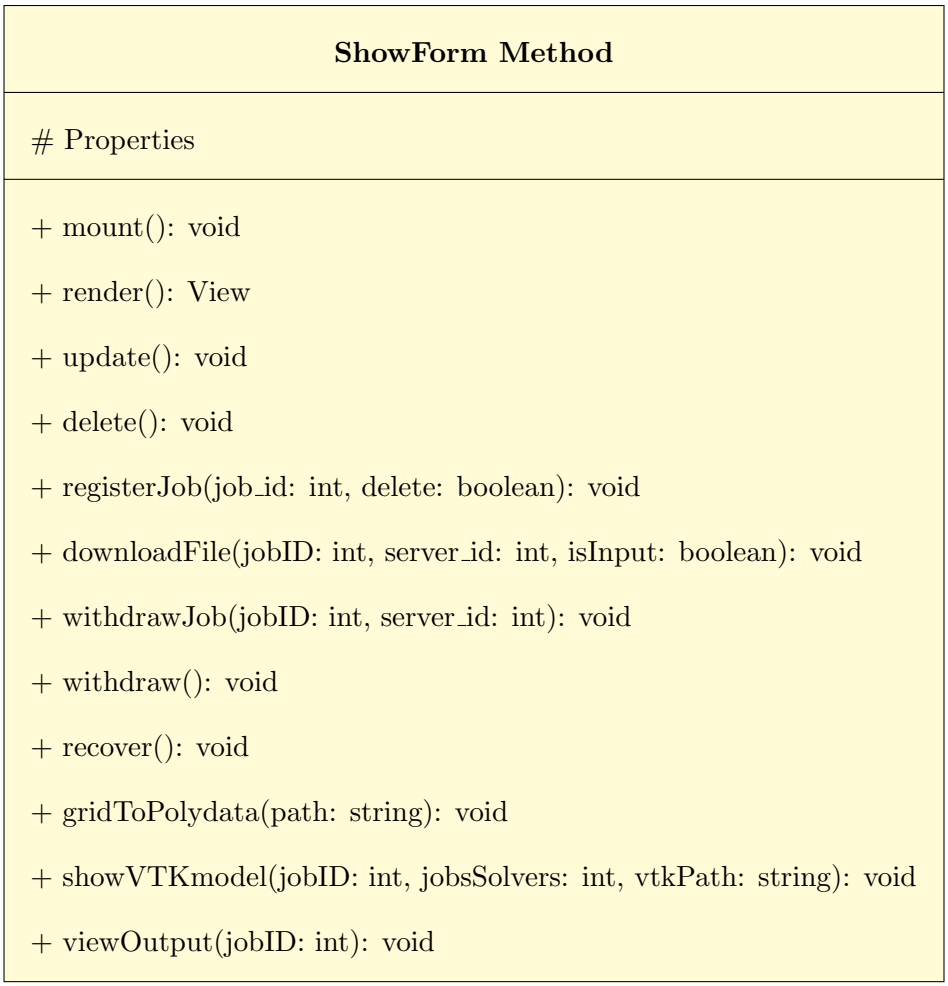

Figure F.1: UML Diagram of the ShowForm Method class

F.1 mount() Method

The mount() method sets up various properties within the component, such as the current route name, categories, and user ID. Additionally, if there is a specific job ID (`$jobID` is not -1), it registers the job using the registerJob method.

```
1          public function mount()
2     {
3          $this->pathName = request()->route()->getName();
```

```
4
5           if($this->pathName != "jobs" && $this->pathName != "jobs.all") {
6               $this->pathName = 'userprofile';
7           }
8
9           $categories = Category::all();
10          foreach ($categories as $category)
11          {
12              $this->categories[$category->id] =  $category->name;
13          }
14          $user = auth()->user();
15          $this->userID = $user->id;
16
17          if($this->jobID !== -1)
18          {
19              $this->registerJob($this->jobID, false);
20          }
21
22      }
23
```

F.1.1 Initialization of Properties

The mount method initializes various properties within the component. The `$pathName` property is set to the name of the current route obtained using `request()->route()->getName()`.

```
1  $this->pathName = request()->route()->getName();
```

F.1.2 Route Name Adjustment

An if condition checks if the current route name is neither “jobs” nor “jobs.all.” If true, it sets `$pathName` to `userprofile`. This conditional logic seems to handle adjustments to the route name.

```
if($this->pathName != "jobs" && $this->pathName != "jobs.all") {
    $this->pathName = 'userprofile';
}
```

F.1.3 Category Retrieval and Mapping

All categories are retrieved from the database using the Category model, and a foreach loop maps category IDs to their respective names in the `$categories` associative array within the component.

```
$categories = Category::all();
foreach ($categories as $category)
{
    $this->categories[$category->id] =  $category->name;
}
```

F.1.4 User ID Assignment

The authenticated user’s ID is assigned to the `$userID` property of the component.

```
$user = auth()->user();
$this->userID = $user->id;
```

F.1.5 Job Registration

If `$jobID` is not equal to -1, the registerJob method is called on the component, passing `$jobID` and false as arguments. This suggests that there is specific logic related to job registration.

```
if($this->jobID !== -1)
{
    $this->registerJob($this->jobID, false);
}
```

F.2 render() Method

The render method is responsible for preparing and returning the data to be rendered by the Livewire component. It checks if a specific job is set and, if so, retrieves its data and configuration. Otherwise, it retrieves a list of jobs based on various criteria, including search terms, category, progress, status, user role, and sorting preferences. The final result is passed to the component's Blade view for rendering.

F.2.1 Initialization and Job Information Check

The render method begins by initializing an empty array named `$jobs = []`. Next, it checks if a specific job (`$this->job`) is set, and the job ID is not -1. If true, it retrieves information about the job, decodes its configuration, and sets properties related to input file JSON and display editability.

```
$jobs = [];
if(isset($this->job) && $this->jobID !== -1)
{
```

```
    $data = Job::find($this->jobID)->toArray();
    $configuration = json_decode($data['configuration'], true);

    $this->inputFileJson =  $configuration['input_file_json'];
    (
    count($this->uploadFields) == count($this->inputFileJson['fileName'])
    -1)?
    $this->displayEditable = true : $this->displayEditable = false;
}
```

F.2.2 Job Query, Pagination, and View Rendering

If no specific job is set or the job ID is -1, the method constructs a query to retrieve jobs based on various criteria such as name search, category search, progress, and status. It handles conditions based on user roles and permissions.

```
else
{
    $jobs = Job::where('jobs.name', 'like', '%'.$this->nameSearch.'%')->
    with('belongsToUser');

    // Additional conditions based on category, progress, and status...

    // Conditions based on user role, permission, and path name...

    // Ordering based on properties...

    $jobs = $jobs->paginate($this->pageNum, ['jobs.*']);
}
```

Finally, the method returns the rendered view, passing the list of jobs to the Livewire component's Blade view.

```
    return view(self::COMPONENT_TEMPLATE, [‘jobs’ => $jobs]);
```

F.3 update() Method

The ‘update‘ method is a part of a Laravel application, responsible for updating job information. The method begins by validating incoming data and then proceeds to update attributes of a specific job, such as its name, description, category, and status. It also handles the upload of input files associated with the job, storing them in the public disk and updating file-related information. Exception handling is implemented to manage potential errors during the file update process, providing a flash message if an exception occurs. The method concludes by updating the job's configuration and status, with the latter being determined based on the presence of uploaded files corresponding to specified upload fields. If any discrepancies are found, the job status is set to 0; otherwise, it is set to 1, indicating successful update with valid file uploads.

F.3.1 Form Validation

The update method starts by validating the incoming data using the Livewire `$this->validate()` method. This ensures that the data adheres to the specified validation rules.

```
public function update()
{
    $this->validate();
    // Validation rules and logic...
}
```

F.3.2 Job Update

If a specific job (`$this->job`) is set, the method proceeds to update various attributes of the job, such as name, description, category ID, and status.

```
if (isset($this->job))
{
    $this->job->name = $this->jobAttr['name'];
    $this->job->description = $this->jobAttr['description'];
    $this->job->category_id = $this->jobAttr['category_id'];
    $this->job->status = $this->jobAttr['status'];
    // Job attribute update logic...
}
```

F.3.3 Handling Input Files

The method then attempts to update input files associated with the job. It iterates through the `$this->inputFiles` array, processes each file type, and updates file-related information in the job's configuration.

```
foreach ($this->inputFiles as $fileType => $file)
{
    if(isset($this->inputFiles[$fileType]))
    {
        // File handling logic...
    }
}
```

F.3.4 File Storage and Exception Handling

Within the file handling logic, the code deletes existing files from storage, updates file-related information, and catches any exceptions that may occur during the process.

```
try
{
    // File deletion and storage logic...
}
catch (\Exception $e)
{
    // Exception handling...
    return redirect()->route(self::REDIRECT_ROUTE);
}
```

F.3.5 Job Status Update

Finally, the method updates the job's status based on the presence of uploaded files and their correspondence to specified upload fields.

```
$data = job::find($this->job->id)->toArray();
$data['configuration'] = json_decode($data['configuration'], true);
$data['configuration']['input_file_json'] = $this->inputFileJson;

if(count($this->inputFileJson['fileName']) - 1 == count($this->
    uploadFields))
{
    // Job status update logic...
}
```

F.4 delete() Method

The delete method is part of the application, responsible for deleting a specific job and handling related operations. The method first checks if a job instance exists and has a valid ID. The result of the deletion operation is captured in the variable `$deleted`. Depending on the success of the deletion, a flash message is set, indicating either the successful deletion of the job or an error message if something went wrong. The flash message style is also set accordingly, with 'success' for successful deletions and 'danger' for errors. Finally, if the job is successfully deleted, the method redirects the user to the appropriate route based on the `pathName` property. Below is the code snippet for the delete method:

```
1  public function delete()
2  {
3      if ($this->job && $this->job->id)
4      {
5          $deleted = $this->job->delete();
6          $msg =  $deleted ? 'Job successfully deleted!' : 'Ooops! Something
       went wrong.';
7          $flag = $deleted ? 'success' : 'danger';
8
9          session()->flash('flash.banner', $msg);
10         session()->flash('flash.bannerStyle', $flag);
11
12         if ($deleted)
13         {
14             if($this->pathName == 'userprofile')
15                 return redirect()->route($this->pathName, ['currentModule'
       => "jobs"]);
16             else
```

```
17                 return redirect()->route($this->pathName);
18             }
19         }
20 }
```

F.5 registerJob() Method

This method is responsible for setting up the state based on a given job ID. This method is a crucial setup step for displaying or managing a specific job within the component. Depending on the context, it prepares the component for either viewing or deleting a job. Let's break down the code into sections:

F.5.1 Job Retrieval and Validation

The method begins by retrieving a job using the provided `$job_id with Job::find($job_id)`. If the job is not found (i.e., it's null), a 404 HTTP response is triggered using Laravel's `abort(404)` function.

```
1 $this->job = Job::find($job_id);
2 if(is_null($this->job))
3 {
4     abort(404);
5 }
```

F.5.2 Job Deletion Confirmation

If the `$delete` parameter is true, it means the method was called with the intention of deleting the job. In such a case, the property `$this->confirmingJobDeletion` is set to

true. This is likely used in the component's view to trigger a confirmation dialog for job deletion.

```
if($delete)
{
    $this->confirmingJobDeletion = true;
}
```

F.5.3 Job Attributes Setup

If the job is not being deleted, the method proceeds to set up various properties based on the retrieved job. It extracts attributes such as name, description, category ID, and status from the job and assigns them to the corresponding properties.

```
else
{
    $this->jobID = $this->job->id;
    if (isset($this->job))
    {
        $this->jobAttr['name'] = $this->job->name;
        $this->jobAttr['description'] = $this->job->description;
        $this->jobAttr['category_id'] = $this->job->category_id;
        $this->jobAttr['status'] = $this->job->status;
```

F.5.4 Configuration and Upload Fields

Additionally, the method handles a job's configuration. It decodes the JSON configuration stored in the job, specifically extracting an array of upload fields. It initializes the `$this->uploadFields` and `$this->inputFiles` properties based on this configuration.

```
    $configuration = json_decode($this->job->configuration, true);
    $this->uploadFields = $configuration["input_property_json"];
    foreach ($this->uploadFields as $fileType => $extension){
        $this->inputFiles[$fileType] = null;
    }
```

F.6 downloadFile() Method

It handles the download of files related to a specific job. This method facilitates the download of files associated with a specific job, provided that the job's progress status is 'Completed' or 'Cancelled'. It zips the files and streams the zip file for download. Let's break down the code into sections:

F.6.1 Job Retrieval and Validation

The method begins by retrieving the job with the specified `$jobID using Job::find($jobID)`. It then checks whether the job's progress status is `Completed` or `Cancelled`. If true, it proceeds with the file download; otherwise, it skips the download.

```
    try {
        $this->job = job::find($jobID);
        if($this->job->progress == 'Completed' || $this->job->progress ==
    'Cancelled') {
```

F.6.2 File Path and Zip Initialization

If the job progress condition is met, the method defines the output name for the zip file and constructs the local path where the files to be zipped are located. It then uses the

PhpZip library to create a zip file and add files from the specified directory to it.

```
1         $output_name = sprintf("%d_output.zip", $jobID);
2         $local_path = public_path()."\storage\jobs\\".$jobID."\output\
    solver_output\\";
3         $local_path = str_replace('\\', '/', $local_path);
4         $files = File::allFiles($local_path);
5         $zipFile = new \PhpZip\ZipFile();
```

F.6.3 Stream Download Response

The method then uses Laravel's `response()->streamDownload` to initiate the download. Inside the closure passed to `streamDownload`, it adds each file to the zip file. Finally, it outputs the zipped content and closes the zip file.

```
1         return response()->streamDownload(function () use($output_name,
    $length, $files)
2         {
3             $zipFile = new \PhpZip\ZipFile();
4             try{
5                 foreach ($files as $file) {
6                     $zipFile->addSplFile($file);
7                 }
8
9                 echo $zipFile ->outputAsString();
10                $zipFile->close();
11            }
12            catch(\PhpZip\Exception\ZipException $e){
13                // handle exception
14                dd($e);
15            }
```

```
16              finally{
17                  $zipFile->close();
18              }
19          },$output_name);
```

F.6.4 Exception Handling

The entire code is wrapped in a try-catch block to handle exceptions. If an exception occurs during the file download process, it will be caught, and you might see a dump of the exception message for debugging purposes.

```
1       }
2       catch(\Exception $e) {
3           dd($e->getMessage());
4       }
```

F.7 withdrawJob() Method

This method is used to initiate the withdrawal or recovery process for a job based on its progress status. If the job is in `Pending` or `In Progress`, it confirms the withdrawal; if the job is `Cancelled`, it confirms the recovery. Additionally, it assigns the provided `$server_id` to a property, although there might be a typo in that assignment. Let's break down the code into paragraphs:

F.7.1 Job Retrieval and Status Check

The method starts by retrieving the job with the specified `$jobID using Job::find($jobID)`. It then checks the progress status of the job. If the progress is `Pending` or `In Progress`,

it sets the `$confirmingJobWithdraw` variable to true, indicating that the withdrawal of the job is being confirmed.

```
1     $this->job = job::find($jobID);
2     if($this->job->progress === 'Pending'||$this->job->progress === 'In
    Progress'){
3         $this->confirmingJobWithdraw = true;
4     }
```

F.7.2 Recovery Confirmation for Cancelled Jobs

If the job's progress is 'Cancelled', it sets `$confirmingJobWithdraw` to false and `$confirmingJobRecove` to true. This suggests that, in the case of a cancelled job, the method is confirming the recovery of the job.

```
1     else if($this->job->progress === 'Cancelled'){
2         $this->confirmingJobWithdraw = false;
3         $this->confirmingJobRecover = true;
4     }
```

F.7.3 Server ID Assignment

The method sets the `$server_id` property to the provided `$server_id`. It's worth noting that there might be a typo in the code where `$this->$server_id` appears, and it could be corrected to `$this->server_id = $server_id` unless this is intended.

```
1     $this->$server_id = $server_id;
```

F.8 withdraw() Method

the withdraw method cancels a job, terminates associated remote jobs on SSH servers, and then updates the job's status and redirects the user to a specific route.

F.8.1 Progress and Previous Progress Update

The method starts by updating the progress property of the job to `Cancelled`, and it stores the previous progress status in the `previous_progress` property.

```
1
2     $this->job->previous_progress = $this->job->progress;
3     $this->job->progress = 'Cancelled';
```

F.8.2 Terminating Remote Jobs on SSH Servers

The code then checks if the job has a related remote job (`$this->job->remotejob`) with a valid remote job ID. If so, it iterates over SSH servers associated with the job (`$this->job->sshservers`) and attempts to terminate the corresponding remote job on each server.

```
1     if ($this->job->remotejob && $this->job->remotejob->remote_job_id) {
2         $remotejob = $this->job->remotejob;
3         $remote_id = $remotejob->remote_job_id;
4         $pid_shell = "qsig -s SIGKILL %s".$remote_id;
5         foreach ($this->job->sshservers as $server) {
6             // ... SSH connection and command execution logic
7         }
8     }
```

F.8.3 SSH Connection and Command Execution

Inside the loop over SSH servers, it establishes an SSH connection to each server and attempts to execute a command to terminate the remote job using `qsig`. After terminating the job, it deletes the corresponding remote job.

```
1             $c = new Connection($server->server_name, $server->host.":".
    $server->port, $server->username, ["password" => $server->password]);
2             $c->run([
3                 $pid_shell
4             ], function($line) use ($server, $remotejob) {
5                 // ... Inner SSH connection and command execution logic
6             });
```

F.8.4 Job Save and Redirect

After handling the termination of remote jobs, the method saves the changes made to the job and redirects the user to a specified route (`$this->pathName`).

```
1
2     $this->job->save();
3     return redirect()->route($this->pathName);
```

F.9 recover() Method

The recover method is designed to recover a job by restoring its previous progress status, saving the changes, and redirecting the user to a specific route. This could be useful in scenarios where a job was canceled but needs to be reactivated with its previous progress information.

```
public function recover(){
        $this->job-> progress = $this->job -> previous_progress;
        $this->job->save();
        return redirect()->route($this->pathName);

    }
```

F.10 gridToPolydate() Method

The `gridToPolydata` method reads a VTK unstructured grid dataset file, modifies specific lines to convert it to a polydata dataset, and then updates the file with the new content.

The method starts by opening the specified file (`$path`) in read mode (`'r'`) and assigns the file resource to the variable `$myFile`. If the file cannot be opened, it outputs an error message. The method reads the file line by line using a while loop until the end of the file (feof). For each line, it checks if the line contains the string `"/DATASET UNSTRUCTURED_GRID/"` or `"/CELLS/"`. If a match is found, it modifies the line accordingly. Specifically, it replaces `"/DATASET UNSTRUCTURED_GRID/"` with `"DATASET POLYDATA"` and manipulates the line related to cell information. After modifying the content, the method creates a new content string by concatenating the modified lines. It then writes the new content back to the same file, effectively updating the file content.

```
public function gridToPolydata($path)
    {
        $myFile = fopen($path,'r') or die("Unable to open file!");
        $unstructuredGrid = "/DATASET UNSTRUCTURED_GRID/";
        $cells = "/CELLS/";
        while(! feof($myFile))
```

```
7         {
8             $line = fgets($myFile);
9             if(preg_match($unstructuredGrid, $line)){
10                $line = "DATASET POLYDATA\n";
11            }
12            else if (preg_match($cells, $line)){
13                $words = explode(" ",$line);
14                $line = implode(" ", ["POLYGONS", $words[1], $words[2]]);
15            }
16            $lines[] = $line;
17        }
18        $new_content = implode('',$lines);
19        //dd($new_content);
20        file_put_contents($path, $new_content);
21    }
```

F.11 showVTKmodel() Method

The `showVTKmodel` method checks the input parameters, determines the type of solver, executes the appropriate actions, and handles redirects or error logs based on the outcomes. The specific details of file handling and solver execution depend on the solver type. The method takes three parameters: `$jobID`, `$jobsSolvers`, and `$vtkPath`. It first checks if `$vtkPath` exists. If true, it attempts to redirect to a route named `'vtk-visualizer'` with the given VTK path. If `$vtkPath` is not present, the method proceeds to generate the VTK model. The method checks the type of solver based on the value of `$jobsSolvers`. If `$jobsSolvers` is equal to 1, it indicates a "CFIEHFMM Solver". If it's not equal to 1, it assumes an "Stratum3D solver." If it's a "CFIEHFMM Solver," the method prepares

the necessary directories and files, copies input files, executes an external executable, and checks the result code. If the result code is 0, it updates the VTK path in the database and attempts to redirect to the `vtk-visualizer` route. If the result code is not 0, it sets up an error log for display. If it's an "Stratum3D Solver," the method prepares the polydata file, converts it using the `gridToPolydata` method (assumed to be defined elsewhere), updates the VTK path in the database, and attempts to redirect to the `'vtk-visualizer'` route.

```
public function showVTKmodel($jobID, $jobsSolvers, $vtkPath)
{
    if ($vtkPath) {
        redirect()->route(‘vtk-visualizer’, [‘vtkPath’=> $vtkPath]);
    } else {
        // Code for generating VTK model
    }
    if ($jobsSolvers == 1) {
        // Code for CFIEHFMM solver
    } else {
        // Code for Stratum3D solver
    }
    // Code for conf solver
    exec($exe_path.‘ ’.$workDirectory, $this->errorLog, $resultCode);
    if ($resultCode == 0) {
        // Update VTK path and redirect
    } else {
        // Display error log
    }
    } else {
        // Code for input solver
        $polydataFile = // Set the polydata file path
        $this->gridToPolydata($polydataFile);
```

```
24          // Update VTK path and redirect
25      }
```

F.12 viewOutPut() Method

The `viewOutput` method checks for the existence of a specific output file, reads its contents if it exists, and sets properties or displays a flash message accordingly. The method checks whether a specific output file exists using Laravel's Storage facade. It looks for the file at the path `'jobs/' . $jobID . '/output/solver_output/slurm-output.out'`. If the file exists, it proceeds to read its contents. If the file doesn't exist, it displays a flash message notifying the user that the output file couldn't be opened. If the output file exists, the method uses Laravel's Storage facade to retrieve the file contents and assigns them to the `$jobOutput` property. It also sets a boolean flag, `$viewJobOutput`, to true. This flag might be used to conditionally display the file contents in the user interface.

```
1  public function viewOutput($jobID)
2      {
3          if(Storage::disk('public')->exists('jobs/'. $jobID .'/output/
       solver_output/slurm-output.out')){
4              $this->jobOutput = Storage::disk('public')->get('jobs/'.
       $jobID .'/output/solver_output/slurm-output.out');
5              $this->viewJobOutput = true;
6          }else{
7              session()->flash('flash.banner', "Unable to open output file")
       ;
8              session()->flash('flash.bannerStyle', 'danger');
9          }
10     }
```

```
11 }
```

Appendix G

Explanation of Input File Generator Class

The ‘InputGenerator’ class is a Livewire component, extending the ‘Livewire\Component’ class. It serves as a container for various public properties designed to store information related to input, validation rules, and the overall state of the component. These properties play a crucial role in managing and controlling the behavior of the Livewire component.

G.1 mount() Method

During the initialization of the component, the ‘mount’ method is automatically invoked. This method serves as a crucial entry point, responsible for loading input information and initializing the underlying system. By leveraging the ‘mount’ method, the component ensures that essential setup tasks are executed when it is first brought into existence.

```
1 public function mount()
2 {
3     $this->loadInputInfo();
```

```
4       $this->initSys();
5       $this->totalSize = count($this->inputInfo_filter);
6       $this->windowUpdate();
7   }
```

G.2 render() Method

The 'render' method is designed to provide the associated view for the Livewire component. This view is crucial for rendering the component's user interface, defining how it appears and interacts with users.

```
1 public function render()
2 {
3       return view(static::COMPONENT_TEMPLATE);
4 }
```

To enhance code organization and maintainability, the 'InputGenerator' class incorporates several private methods, each encapsulating specific functionalities. These include 'windowUpdate', 'loadInputInfo', 'initSys', 'updatedInputValue', 'refreshESeq', 'refreshRepeatNum', and 'fileGenerationHelper'. These private methods contribute to a modular and organized code structure, allowing for easier comprehension and maintenance of the component's logic.

G.3 generateFile() Method

The core functionality of generating an input file is encapsulated in the 'generateFile' method. This method leverages the component's current state to dynamically create an

input file, and it further facilitates the response mechanism for streaming the generated file to users as a downloadable resource.

G.3.1 Input Validation

At the beginning of the method, `'$this->validate()'` is called. This Livewire method triggers the validation of the component's input data based on the rules defined in the `'rules'` method. This ensures that the input is valid before proceeding with file generation.

```
$this->validate();
```

G.3.2 File Generation Logic

The method then initializes an empty string `'$writeStr'` to store the content of the generated file. It iterates through the properties defined in `'$this->inputInfo['properties']'`. For each property, it checks certain conditions to determine whether the property should be included in the generated file.

```
$writeStr = '';
foreach ($this->inputInfo['properties'] as $key => $property) {
    // Display conditions and property title addition
    // ...

    // Main and children value handling
    // ...

    // Iterating through children
    // ...

    // Line break handling
```

```
13      // ...
14  }
```

G.3.2.1 Display Conditions

If the property has a `'displayOnEnable'` attribute and it is enabled, the property is skipped if the condition is not met. If the property has a `'fileDisplay'` attribute set to `'NONE'`, it is also skipped.

```
1  if (isset($property['displayOnEnable'])) {
2      // Display condition based on 'displayOnEnable'
3  }
4
5  // OR
6
7  if (isset($property['fileDisplay'])) {
8      // Display condition based on 'fileDisplay'
9  }
```

G.3.2.2 Adding Property Title

For each property that passes the display conditions, the method appends the property's title to `'$writeStr'`. If the `'newline'` attribute is set in `'$this->inputInfo'`, a newline character is added after the title.

```
1  $writeStr .= $property['title'];
2  if ($this->inputInfo['newline']) {
3      $writeStr .= "\n";
4  }
```

G.3.2.3 Handling Main and Children Values

The method then iterates through the values associated with the property, distinguishing between the main value (‘`$vKey === "main"`’) and any children values (‘`$vKey === "children"`’). For the main value, it checks for null or empty values, assigning the default value if available. The value is then added to ‘`$writeStr`’ using the ‘`fileGenerationHelper`’ method.

```
foreach ($this->inputValue[$key] as $vKey => $value) {
    if ($vKey === "main") {
        // Main value handling
    } else if ($vKey === "children") {
        // Children value handling
    }
}
```

G.3.2.4 Iterating Through Children

If the property has children values, a nested loop iterates through each child, checking for null or empty values and assigning defaults when necessary. The ‘`fileGenerationHelper`’ method is called to append the child values to ‘`$writeStr`’.

```
foreach ($value as $cIndex => $cValues) {
    foreach ($cValues as $cName => $cValue) {
        // Child value handling
    }
}
```

G.3.2.5 Line Break Handling

Finally, if the ‘linebreak‘ attribute is set in ‘`$this->inputInfo`’, and it is ‘true’, a newline character is added to ‘`$writeStr`’.

```
if (isset($this->inputInfo['linebreak']) && $this->inputInfo['linebreak'])
    {
    $writeStr .= "\n";
}
```

G.3.3 Streaming the File for Download

The method concludes by using Laravel's response mechanism to stream the generated content for download. The `'streamDownload'` method is employed, and a closure is provided to echo the content (`'$writeStr'`). The file is streamed with a specified output filename from `'$this->inputInfo['outputFileName']'`.

```
return response()->streamDownload(function () use ($writeStr) {
    echo $writeStr;
}, $this->inputInfo['outputFileName']);
```

The `'generateFile'` method meticulously constructs an input file based on the component's state and specifications, taking into account various conditions and properties for inclusion or exclusion in the final output. The resulting file is then streamed for the user to download.

G.4 toggleAdvanced() Method

Lastly, the `'toggleAdvanced'` method provides a mechanism to toggle the `'advancedStatus'` property. This property likely influences the advanced settings or features within the Livewire component, allowing users to switch between different states or modes based on their preferences or requirements.

```
1 public function toggleAdvanced()
2     {
3         $this->advancedStatus = !$this->advancedStatus;
4     }
```

Bibliography

[1] Alireza Niazi, Shucheng Zheng, Chris Nguyen, and Vladimir Okhmatovski. Full-wave analysis of interconnects in finite substrates with layered media formulation of svs-efie for 3d composite metal-dielectric structures. In *2023 IEEE 32nd Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, pages 1–3, 2023. doi: 10.1109/EPEPS58208.2023.10314891.

[2] Web application architecture: The latest guide 2024. `https://www.clickittech.com/devops/web-application-architecture/`. Accessed: 2022, March.

[3] Farhad Sheikh Hosseini Lori, Anton Menshov, Reza Gholami, Jamiu Babatunde Mojolagbe, and Vladimir I. Okhmatovski. Novel single-source surface integral equation for scattering problems by 3-d dielectric objects. *IEEE Transactions on Antennas and Propagation*, 66(2):797–807, 2018. doi: 10.1109/TAP.2017.2781740.

[4] Farhad Sheikh Hosseini Lori, Anton Menshov, and Vladimir I. Okhmatovski. New vector single-source surface integral equation for scattering problems on dielectric objects in 2-d. *IEEE Transactions on Antennas and Propagation*, 65(7):3794–3799, 2017. doi: 10.1109/TAP.2017.2705226.

[5] Anton Menshov and Vladimir Okhmatovski. Novel surface integral equation formulation for accurate broadband rl extraction in transmission lines of arbitrary cross-section. In

2012 IEEE/MTT-S International Microwave Symposium Digest, pages 1–3, 2012. doi: 10.1109/MWSYM.2012.6259361.

[6] Anton Menshov and Vladimir Okhmatovski. New single-source surface integral equations for scattering on penetrable cylinders and current flow modeling in 2-d conductors. *IEEE Transactions on Microwave Theory and Techniques*, 61(1):341–350, 2013. doi: 10.1109/TMTT.2012.2227784.

[7] Anton Menshov and Vladimir I. Okhmatovski. Surface–volume–surface electric field integral equation for magneto-quasi-static analysis of complex 3-d interconnects. *IEEE Transactions on Microwave Theory and Techniques*, 62(11):2563–2573, 2014. doi: 10.1109/TMTT.2014.2360838.

[8] Scott L Ray Andrew F Peterson and Raj Mittra. *Computational methods for electromagnetics*, volume 2. 1997.

[9] Roger F Harrington and Jan L Harrington. *Field computation by moment methods*. Oxford University Press, Inc., 1996.

[10] Anton Menshov and Vladimir Okhmatovski. Method of moment solution of surface-volume-surface electric field integral equation for two-dimensional transmission lines of complex cross-sections. In *2012 IEEE 16th Workshop on Signal and Power Integrity (SPI)*, pages 31–34, 2012. doi: 10.1109/SaPIW.2012.6222905.

[11] Kush Agarwal and Yong xin Guo. Interaction of electromagnetic waves with humans in wearable and biomedical implant antennas. *2015 Asia-Pacific Symposium on Electromagnetic Compatibility (APEMC)*, pages 154–157, 2015. URL `https://api.semanticscholar.org/CorpusID:37096165`.

[12] Raul Enriquez-Shibayama, Benjamin Lopez-Garcia, Luis Nathan Perez-Acosta, and Ana Sanchez-Granados. Design of very low distortion dense routings for skew compensation in high speed interconnects. *IEEE Electromagnetic Compatibility Magazine*, 5(4):100–103, 2016. doi: 10.1109/MEMC.2016.7866246.

[13] Barışcan Karaosmanoğlu and Özgür Ergül. Modified combined tangential formulation for stable and accurate analysis of plasmonic structures. In *2017 International Applied Computational Electromagnetics Society Symposium - Italy (ACES)*, pages 1–2, 2017. doi: 10.23919/ROPACES.2017.7916304.

[14] Raul Enriquez-Shibayama, Benjamin Lopez-Garcia, Luis Nathan Perez-Acosta, and Ana Sanchez-Granados. Design of very low distortion dense routings for skew compensation in high speed interconnects. *IEEE Electromagnetic Compatibility Magazine*, 5(4):100–103, 2016.

[15] Jim Conallen. Modeling web application architectures with uml. *Commun. ACM*, 42 (10):63–70, oct 1999. ISSN 0001-0782. doi: 10.1145/317665.317677. URL `https://doi.org/10.1145/317665.317677`.

[16] Llyukha, V. the guide to web application architecture. `https://jelvix.com/blog/guide-to-web-application-architecture`. Accessed: 2020, Juni 15.

[17] Luchaninov, Y. web application architecture in 2023: Moving in the right direction. `https://mobidev.biz/blog/web-application-architecture-types`. Accessed: 2022, October 30.

[18] Dhaduk, H. an ultimate guide to web application architecture. `https://www.simform.com/blog/web-application-architecture/`. Accessed: 2023, April 27.

[19] What do client side and server side mean? — client side vs. server side. https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/.

[20] Documentation. https://help.figma.com/hc/en-us. Note: Documentation Page.

[21] Welcome to the xd user guide. https://helpx.adobe.com/ca/xd/user-guide.html. Note: User Guide Page.

[22] Common client-side web technologies. https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-client-side-web-technologies.

[23] Html5 differences from html4. http://www.w3.org/TR/html5-diff/. Accessed: 2012.

[24] The importance of css in web development. https://www.plasmacomp.com/blogs/importance-of-css-in-web-development/.

[25] The importance of javascript in modern web development. https://designoptimise.com/the-importance-of-javascript-in-modern-web-development/.

[26] Serverless, microservices, or monolith: Choosing back end architecture for your startup. https://leanylabs.com/blog/serveless-microservices-monolith/. Accessed: 2023.

[27] A beginner's guide to server-side scripting. https://www.scrums.com/blog/a-beginners-guide-to-server-side-scripting-for-web-development.

[28] History of php. https://www.php.net/manual/en/history.php.php, .

[29] Php 8.3: What's new and what's changed in the latest release. https://kinsta.com/blog/php-8-3/, .

[30] What is asp.net core? https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core.

[31] What is node.js? here's how to use server-side javascript. https://www.makeuseof.com/node-js-server-side-javascript/.

[32] How to set up and implement database for web design and development. https://axiomq.com/blog/how-to-set-up-and-implement-database-for-web-design-and-development/.

[33] What's the difference between relational and non-relational databases? https://aws.amazon.com/compare/the-difference-between-relational-and-non-relational-databases/.

[34] 8 advantages of a relational database. https://www.easasoftware.com/insights/8-advantages-of-a-relational-database/, .

[35] Relational database benefits and limitations (advantages disadvantages). https://databasetown.com/relational-database-benefits-and-limitations/, .

[36] Disadvantages of a relational database. https://www.techwalla.com/articles/disadvantages-of-a-relational-database.

[37] Non-relational data and nosql. https://learn.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data, .

[38] Nosql databases: A practical approach for web developers. https://www.searchmyexpert.com/resources/web-development/nosql-databases, .

[39] What is nosql? https://www.oracle.com/ng/database/nosql/what-is-nosql/, .

[40] Nosql databases: Advantages and disadvantages. https://www.dataversity.net/nosql-databases-advantages-and-disadvantages/, .

[41] Best nosql databases. https://www.g2.com/categories/nosql-databases, .

[42] A B M Moniruzzaman and Syed Akhter Hossain. Nosql database: New era of databases for big data analytics - classification, characteristics and comparison, 2013.

[43] Navigating the limitations and challenges of nosql databases. https://kerelka.com/blogs/navigating-the-limitations-and-challenges-of-nosql-databases, .

[44] 8 major advantages of using mysql. https://www.datamation.com/storage/8-major-advantages-of-using-mysql/, .

[45] Acid explained: Atomic, consistent, isolated durable. https://www.bmc.com/blogs/acid-atomic-consistent-isolated-durable/, .

[46] What is a lamp stack? https://aws.amazon.com/what-is/lamp-stack/, .

[47] Managing unstructured data. https://www.mongodb.com/scale/managing-unstructured-data, .

[48] Make schema-less and relational databases work together. https://towardsdev.com/make-schema-less-and-relational-databases-work-together-44b8a2b78d55, .

[49] Sharding in mongodb. https://www.mongodb.com/basics/sharding, .

[50] Best 7 real-world mongodb use cases. https://hevodata.com/learn/mongodb-use-case/, .

[51] The benefits of team-based organizations and faster growth. https://risepeople.com/blog/team-based-organizations/.

[52] Waterfall methodology: A comprehensive guide. https://www.atlassian.com/agile/project-management/waterfall-methodology.

[53] What is agile? https://www.atlassian.com/agile.

[54] What is scrum? https://www.scrum.org/resources/what-scrum-module, .

[55] Extreme programming. https://en.wikipedia.org/wiki/Extreme_programming.

[56] How the kanban methodology applies to software development. https://www.atlassian.com/agile/kanban.

[57] Agile project management with scrum. https://www.pmi.org/learning/library/agile-project-management-scrum-6269, .

[58] Agile scrum roles and responsibilities. https://www.atlassian.com/agile/scrum/roles, .

[59] Product owner. https://scaledagileframework.com/product-owner/, .

[60] The key values and principles of the agile manifesto. https://resources.scrumalliance.org/Article/key-values-principles-agile-manifesto, .

[61] Devops. https://en.wikipedia.org/wiki/DevOps.

[62] What is continuous integration. https://learn.microsoft.com/en-us/devops/develop/what-is-continuous-integration, .

[63] What is continuous delivery. https://www.ibm.com/topics/continuous-delivery, .

[64] What is ci/cd in devops? https://www.jetbrains.com/teamcity/ci-cd-guide/devops-ci-cd/, .

[65] Lean software development. https://en.wikipedia.org/wiki/Lean_software_development, .

[66] 7 principles of lean software development. https://builtin.com/software-engineering-perspectives/lean-software-development, .

[67] Rashidah Funke Olanrewaju, Thouhedul Islam, and Nor'ashikin Ali. An empirical study of the evolution of php mvc framework. 2015. URL https://api.semanticscholar.org/CorpusID:54115166.

[68] Rajeev Patrick Das and Dr. Lakshmi Prasad Saikia. Comparison of procedural php with codeigniter and laravel framework. 2016. URL https://api.semanticscholar.org/CorpusID:201027699.

[69] Ren Yu He. design and implementation of web based on laravel framework. In *Proceedings of the 2014 International Conference on Computer Science and Electronic Technology*, pages 301–304. Atlantis Press, 2015/01. ISBN 978-94-62520-47-9. doi: 10.2991/iccset-14.2015.66. URL https://doi.org/10.2991/iccset-14.2015.66.

[70] Xianjun Chen, Zhoupeng Ji, Yu Fan, and Yongsong Zhan. Restful api architecture based on laravel framework. *Journal of Physics: Conference Series*, 910(1):012016, oct 2017. doi: 10.1088/1742-6596/910/1/012016. URL https://dx.doi.org/10.1088/1742-6596/910/1/012016.

[71] M. Stauffer. *Laravel: Up and Running: A Framework for Building Modern PHP Apps*. O'Reilly Media, Incorporated, 2019. ISBN 9781492041214.

[72] Http routing. https://laravel.com/docs/5.2/routing. Note: Laravel documentation.

[73] David R Swatek. Investigation of single-source surface integral equations for electromagnetic wave scattering by dielectric bodies. 1999.

[74] Zhi Guo Qian, Weng Cho Chew, and Roberto Suaya. Generalized impedance boundary condition for conductor modeling in surface integral equation. *IEEE Transactions on Microwave Theory and Techniques*, 55(11):2354–2364, 2007. doi: 10.1109/TMTT.2007.908678.

[75] J.M. Jin. *Theory and Computation of Electromagnetic Fields*. IEEE Press. Wiley, 2011. ISBN 9781118088111.

[76] Shucheng Zheng, Anton Menshov, and Vladimir I. Okhmatovski. New single-source surface integral equation for magneto-quasi-static characterization of transmission lines situated in multilayered media. *IEEE Transactions on Microwave Theory and Techniques*, 64(12):4341–4351, 2016. doi: 10.1109/TMTT.2016.2623625.

[77] Utkarsh R. Patel, Sean V. Hum, and Piero Triverio. A magneto-quasi-static surface formulation to calculate the impedance of 3d interconnects with arbitrary cross-section. In *2017 IEEE 21st Workshop on Signal and Power Integrity (SPI)*, pages 1–4, 2017. doi: 10.1109/SaPIW.2017.7944006.

[78] Yu Zhao, Min Tang, Shang Xiang, and Junfa Mao. $\mathcal{H}$-matrix accelerated contour integral method for modeling multiconductor transmission lines. *IEEE Transactions on Electromagnetic Compatibility*, 60(2):552–555, 2018. doi: 10.1109/TEMC.2017.2722820.

[79] Mario Bebendorf. Approximation of boundary element matrices. *Numerische Mathematik*, 86:565–589, 2000. URL `https://api.semanticscholar.org/CorpusID:206858339`.

[80] M. Bebendorf. *Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems*. Lecture Notes in Computational Science and Engineering. Springer Berlin Heidelberg, 2008. ISBN 9783540771470.

[81] Mario Bebendorf and Sergej Rjasanow. Adaptive low-rank approximation of collocation matrices. *Computing*, 70(1):1–24, 2003.

[82] Steffen Börm, Lars Grasedyck, and Wolfgang Hackbusch. Introduction to hierarchical matrices with applications. *Engineering Analysis with Boundary Elements*, 27(5):405–422, 2003.

[83] Yaniv Brick, Vitaliy Lomakin, and Amir Boag. Fast direct solver for essentially convex scatterers using multilevel non-uniform grids. *IEEE Transactions on Antennas and Propagation*, 62(8):4314–4324, 2014.

[84] Weng Cho Chew. *Waves and fields in inhomogeneous media*. IEEE press, 1995.

[85] Blade templates. `https://laravel.com/docs/10.x/blade`, .

[86] Livewire quick start. `https://laravel-livewire.com/docs/2.x/quickstart`.

[87] Get started with tailwind css. `https://tailwindcss.com/docs/installation`.

[88] Csrf protection. `https://laravel.com/docs/10.x/csrf`.

[89] Middleware documentation in laravel. `https://laravel.com/docs/10.x/middleware`.

[90] University of manitoba high-performance computing system grex. `https://umanitoba.ca/information-services-technology/research-computing/um-high-performance-computing-system-grex`.

[91] Task scheduling. `https://laravel.com/docs/10.x/scheduling`, .

[92] Kitware, inc. https://github.com/Kitware.

[93] Paraview, getting started document. https://www.paraview.org/resources/.

[94] Vtk.js - the visualization toolkit for javascript. https://github.com/Kitware/vtk-js.