Eddy3D Backend

JIB 4329

Abhinn Vegesna, Alexander Withka, Benjamin Dai, Rohit Dudala

Client: Patrick Kastner

Repository: https://github.com/Urbano-io/Urbano-Backend

Table of Contents

List of Figures

Terminology

API (Application Programming Interface): A set of rules and protocols that specify how two applications communicate and interact with each other.

Back-end: The portion of an application that operates behind the scenes and is not directly accessible by the user and handles the application's logic, database, and server-side operations.

C#: An object-oriented programming language developed by Microsoft that is commonly used for building applications on the .NET platform.

Hoppscotch: An API testing platform that allows developers to test RESTful APIs quickly and efficiently by sending various types of requests and analyzing responses.

MongoDB: A NoSQL database that stores data in a flexible format which makes it highly scalable and suitable for applications requiring dynamic schema or real-time data handling.

.NET: An open-source development platform created by Microsoft that provides a robust system of tools, libraries, and frameworks which can be used for building various types of applications.

Simulation Quotas: Limits set within a system to manage and restrict the number of simulations a user can run within a given time frame.

Introduction

Background

Our project aims to build upon the backend for Eddy3D, which is a widely used software that facilitates airflow and microclimate simulations for Rhino and Grasshopper. Our primary objective is to build a robust backend system to track and store user information, enabling valuable insights into software usage patterns. User data will be collected and stored in MongoDB, which will serve as a centralized repository for user metrics. To facilitate backend development and ensure compatibility with the existing backend structure, we will use .NET and C# when writing our code. We will also utilize Hoppscotch for fast and reliable local API testing. This development process will enable us to effectively write and validate API endpoints that support key user functionalities such as logging in, registering, and managing quota tokens. In the later stages of the project, we will design an admin dashboard to visualize key user metrics which will make it easier for the client to monitor software usage. The dashboard will include interactive elements for tracking essential metrics. This will provide a comprehensive view of user behavior and help drive future improvements to Eddy3D.

Document Summary

The <u>System Architecture</u> outlines the interactions between the key components of our system, including API routing, testing with Hoppscotch, the MongoDB database, and user information tracking. It provides an overview of both the static and dynamic aspects of the architecture, highlighting how data flows seamlessly from user interactions through the backend for processing and storage.

The <u>Data Storage Design</u> section outlines how we store user-specific data using MongoDB. User data such as logging activity, account registration, and simulation quota usage is stored securely in MongoDB. Data exchange occurs via secure API endpoints, and security concerns are addressed through encryption and authentication.

The <u>Component Detailed Design</u> section describes the interactions among the specific backend components of our system, including API routing, business logic, and the MongoDB database. Specific methods and classes are showcased including those involved in the functionality of user registration, activity logging, quota management, and metrics tracking.

The UI Design section presents a basic admin interface used to visualize key user metrics. This includes tools to monitor simulation quotas and view user activity.

System Architecture

Introduction

Our backend system is designed with a layered architecture to provide flexibility, modularity, and scalability. This approach enables parallel development across components which allows for faster iteration and easier maintenance. The system's primary goal is to enable effective user tracking and data management by capturing key metrics. By securely storing this data and providing useful metrics, the system aims to support admins when it comes to monitoring user behavior and optimizing performance effectively.

The separation of concerns format of our design ensures that each layer handles a specific responsibility, such as API routing or database interaction which helps in reducing dependencies and simplifying updates. For example, if an API endpoint for user tracking needs to be modified, the respective layer can be updated without affecting other parts of the system. Additionally, this design enhances scalability by enabling individual layers to be scaled independently if needed to handle increased system demands. It also ensures flexibility for future changes such as switching to a new database or updating user tracking features. Additionally, this architecture structure simplifies testing and debugging by isolating issues to specific layers and supports seamless team collaboration since different layers can be developed and maintained simultaneously.

From a security perspective, our architectural choices were specifically designed to ensure data integrity and user privacy. For instance, we will implement authentication mechanisms for updating user data which will allow for secure user authentication and access control. During development, we will perform thorough local testing using Hoppscotch which allows us to simulate and validate API requests and responses efficiently. This ensures that any vulnerabilities or errors in the system can be identified and resolved in a controlled environment before deployment. Furthermore, all communication with our backend code will utilize HTTPS to ensure that data is encrypted in transit. By combining secure development practices and effective local testing, we aim to build a backend system that prioritizes both functionality and user security.

In the following sections, we provide an overview of our system architecture using two diagrams. The first is a static diagram (Figure 1) which illustrates the relationships between the core components of our system and offers a high-level view of how the backend is structured. The second is a dynamic diagram (Figure 2) which demonstrates how these components interact during specific workflows which helps provide a m more concrete example of how user data is processed, stored, and utilized.

Static System Architecture

The following diagram illustrates the interactions between the various components of our system within a static system architecture.

Diagram:

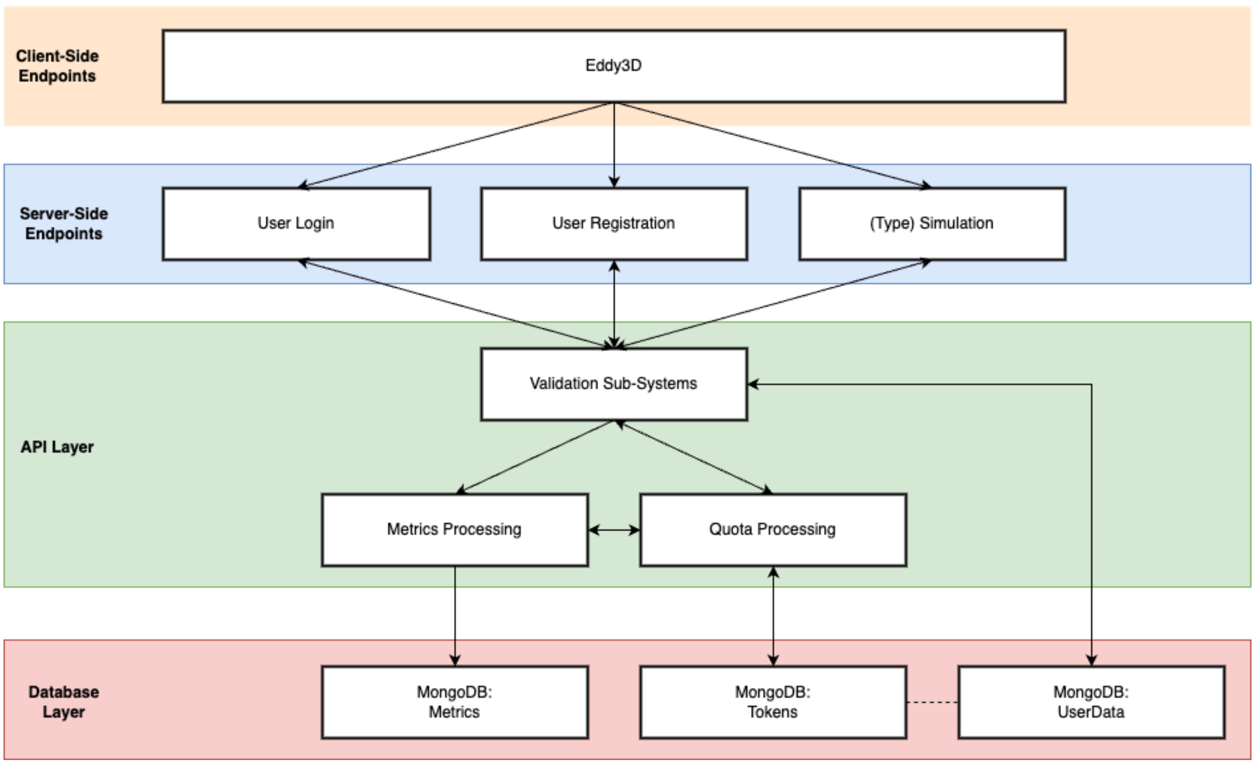

Figure 1 - Static System Diagram

Description:

The Eddy3D client-side endpoints via the front-end communicate with the server-side endpoints to enable user login, registration, and simulations. Validation systems on the API layer, validate token types and amounts to enable the simulations to work properly and introduce quotas. Metrics for usage as well as permanent and continuing storage of tokens for users as well as other data necessary for simulations are kept in the database layer which also communicates with the API layer to give the necessary data to allow for processing of quota tokens and user data to allow the simulations to work as intended.

Dynamic System Architecture

The following diagram illustrates the interactions between the various components of our system within a dynamic system architecture.

Diagram:

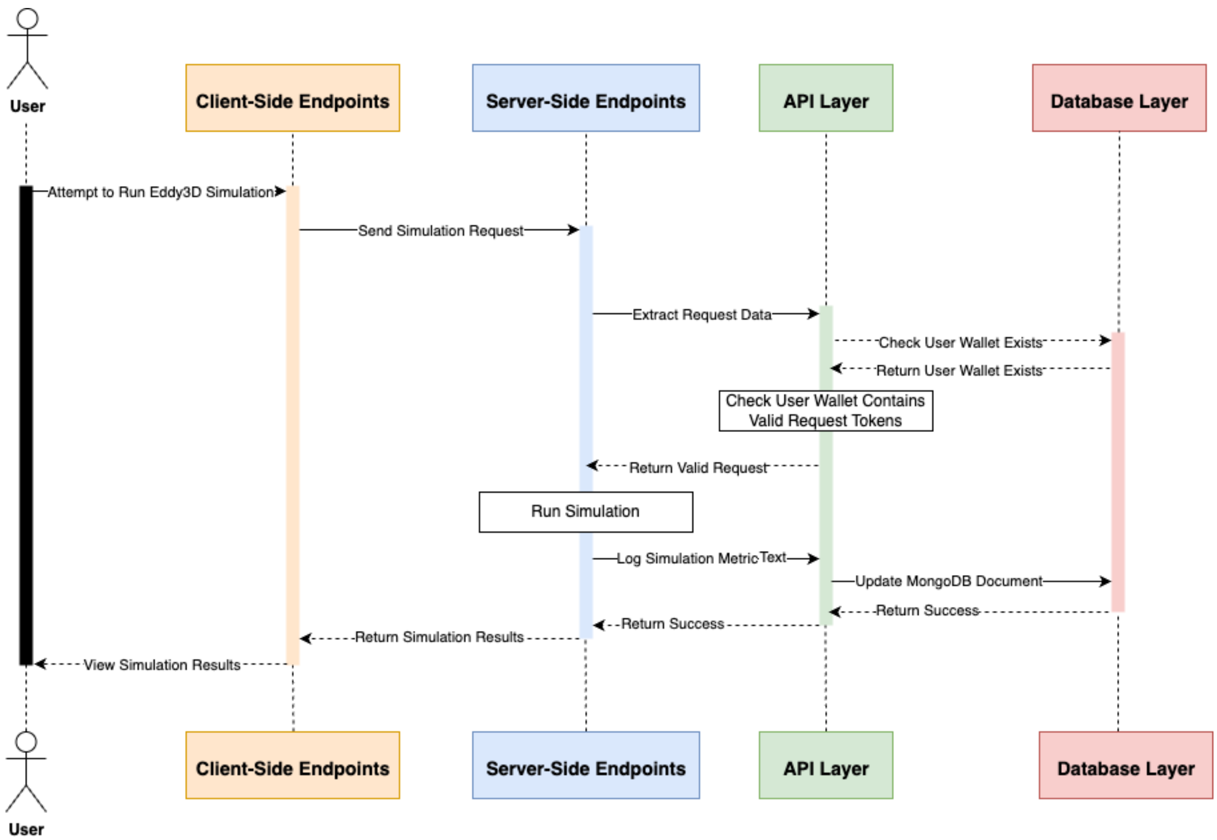

Figure 2 - Dynamic System Diagram

Description:

The System Sequence Diagram above shows what would happen when a user attempts to run a simulation via Eddy3D. The existing server-side infrastructure that Dr. Kastner and his team have developed already runs the simulations themselves, but we expand upon that functionality via our API and Database layers, adding more error-checking as well as the new functionality of tracking simulation metrics for Eddy3D admins to view and analyze.

Data Design

Introduction

In Figure 3, we have given a diagram according to the needs of our backend system. This diagram shows how we utilize MongoDB to save and retrieve metrics for Eddy3D users. The type of database we are using in this project is NoSQL. After the diagram, the file usage, data exchange, and data security details are discussed.

Data Storage Diagram

The following diagram illustrates the various data storage entities of our system.

Diagram:

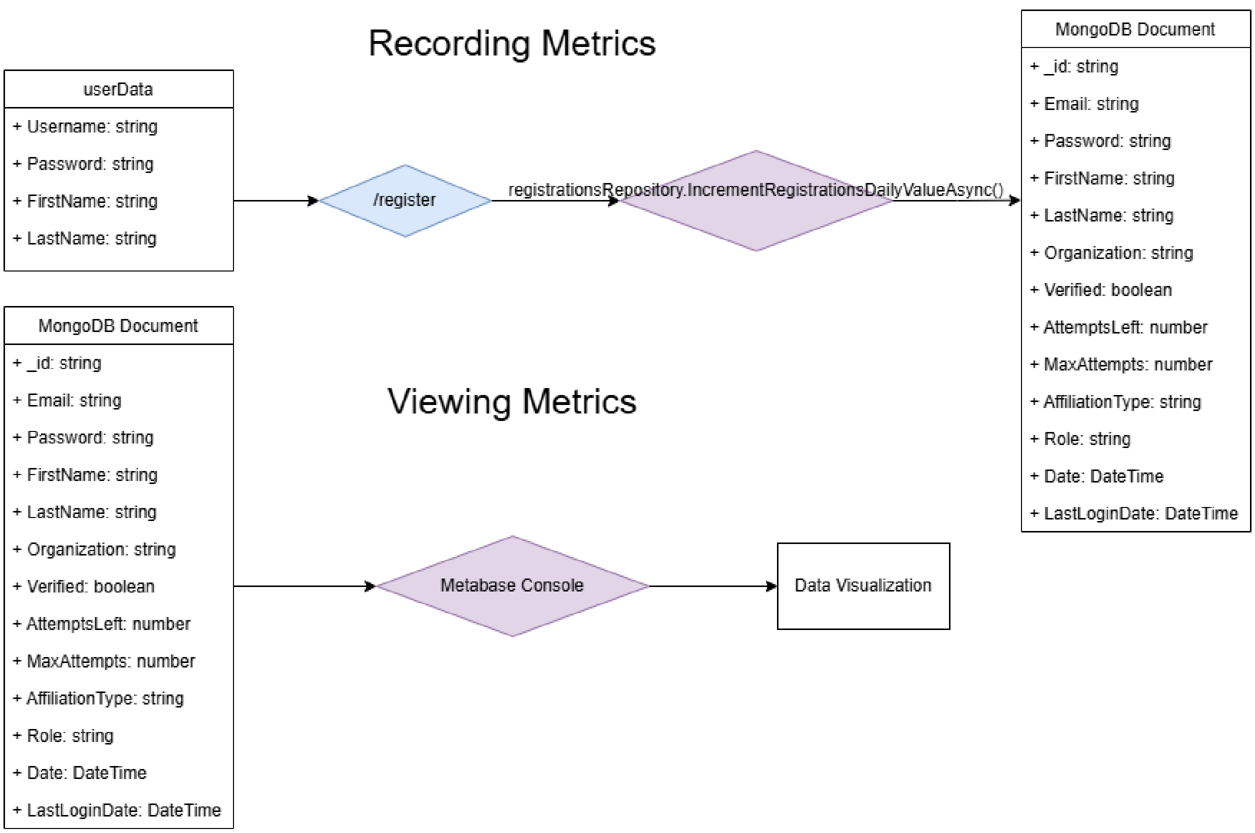

Figure 3 – Data Storage Diagram

Description:

Here is a list of the objects and functions in more details:

1. userData – the DTO object we use to keep track of user data upon hitting the register endpoint
2. /register – the sample endpoint
3. RegistrationsRepository.IncrementRegistrationsDailyValueAsync() – our middleware to handle querying our MongoDB per endpoint hit
4. MongoDB Document – an object that exists as a record per user for Eddy3D
5. Metabase Console – third-party website that allows views of databases like MongoDB
6. Data Visualization – the graphs and visuals that Metabase can create for admins to see how many users Eddy3D has registered

File Use

We do not interact with any files in our application.

Data Exchange

We have no data transfer between physical devices.

Data Security

We deal with password storing via MongoDB. Our database encrypts passwords using SHA-256 and then stores them, keeping user account passwords secure. User login is required via Eddy3D plugin components – our backend system is not responsible for any security concerns with the existing Eddy3D frontend functionalities. Our system does not handle any financial data. Personally identifiable information is limited to first and last name, for which users will be provided basic protection via the security provided by MongoDB.

Component Detailed Design

Introduction

Our backend system is designed to provide flexibility, modularity, and scalability. This approach enables parallel development across components which allows for faster iteration and easier maintenance. The system's primary goal is to enable effective user tracking and data management by capturing key metrics. By securely storing this data and providing useful metrics, the system aims to support admins when it comes to monitoring user behavior and optimizing performance effectively.

In the following sections, we provide an overview of our component design using two diagrams. The first is a static diagram (Figure 3) which illustrates the relationships between the core components of our system and offers a deeper dive of how the backend is structured. The second is a dynamic diagram (Figure 4) which demonstrates how these components interact during specific workflows which helps provide a more concrete example of how user data is processed, stored, and utilized.

Static Component Design

The following diagram illustrates the various components of our system in more detail.

Diagram:

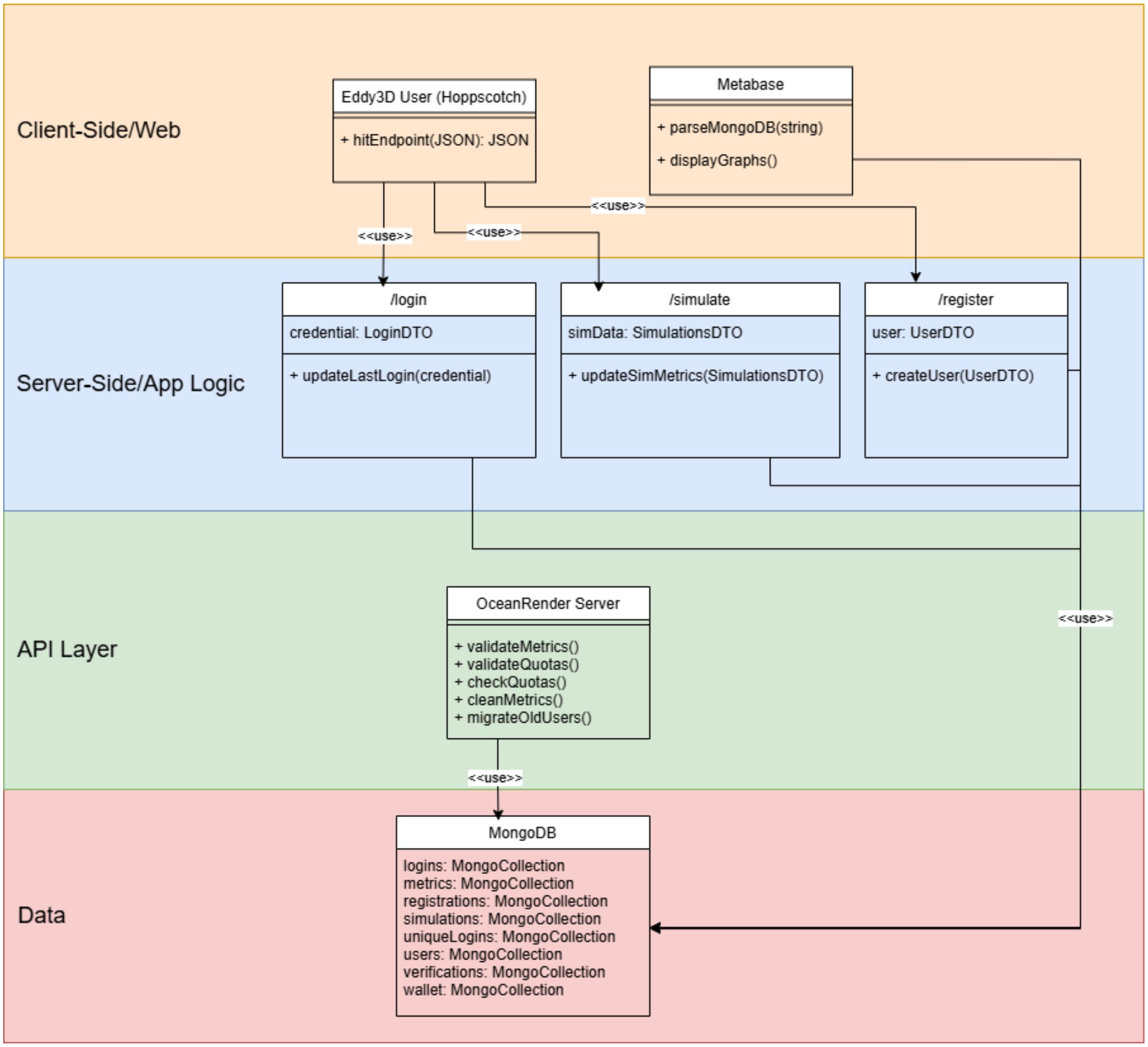

Figure 4 - Static Component Diagram

Description:

The diagram uses color-coding consistent with previous diagrams' color schemes, portraying the 4 separate main layers of the backend. The Eddy3D client-side endpoints via the front-end communicate with the server-side endpoints to enable user login, registration, and simulations. Hoppscotch simulates the actions of a normal Eddy3D user during development, and Metabase is a 3rd-party tool that can visualize MongoDB databases. Validation systems on the API layer validate token types and amounts to enable the simulations to work properly and introduce quotas. Metrics for usage as well as permanent and continuing storage of tokens for users as well as other data necessary for simulations are kept in the database layer which also communicates with the API layer to give the necessary data to allow for processing of quota tokens and user data to allow the simulations to work as intended. Almost all components interact with and alter the MongoDB database, which holds the core of all metric data being measured for Eddy3D.

Dynamic Component Design

The following diagram illustrates the interactions occurring between components within our system during a standard workflow of simulation call from a user to being added to metrics displayed on Metabase dashboard.

Diagram:

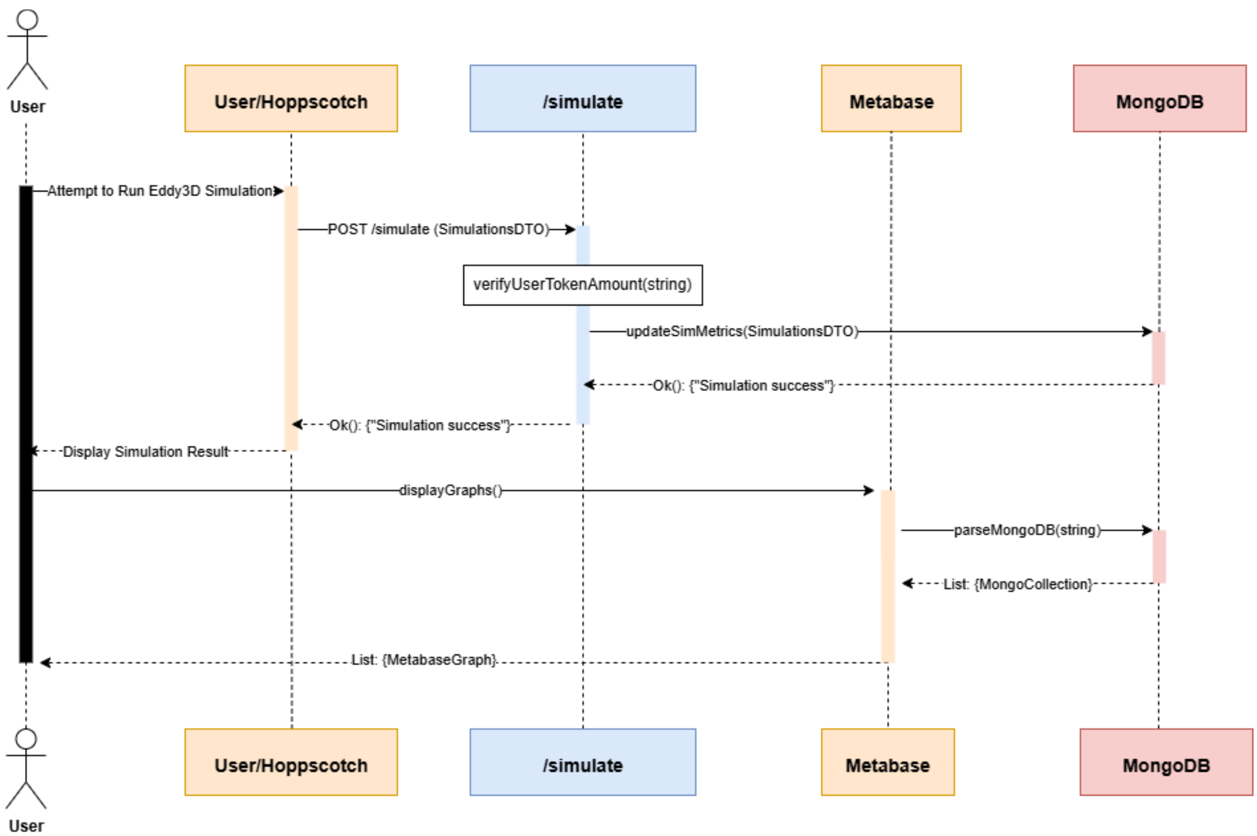

Figure 5 - Dynamic Component Diagram

Description:

The System Sequence Diagram above shows what would happen when a user attempts to run a simulation via Eddy3D, as well as retrieving the reflected metric changes due to the simulation call within Metabase. The Eddy3D user puts in a request for a simulation, sending a POST request to the /simulate endpoint, which verifies the user's quota token amount is valid for the request. The endpoint itself updates MongoDB, adding one to the total simulation count and relevant counts. After this, the user opens Metabase, a website, which in turn accesses the MongoDB and returns a display of all the metrics on the Metabase dashboard.

UI Design

Our project is primarily a backend implementation with a small frontend aspect involving Metabase as an admin dashboard to view the metrics collected by our backend implementation. As such, we do not have a frontend UI to show that is made by us, so we will be displaying a sample UI frontend that an Eddy3D admin may see while using Metabase to query our backend statistics.

When an admin connects to the MongoDB backend and views it through Metabase, they will be greeted with a dashboard like the one displayed in Figure 6 at the top. It would customizable, and example graphs could be displayed for metrics like user logins and registrations per a selected timeline.

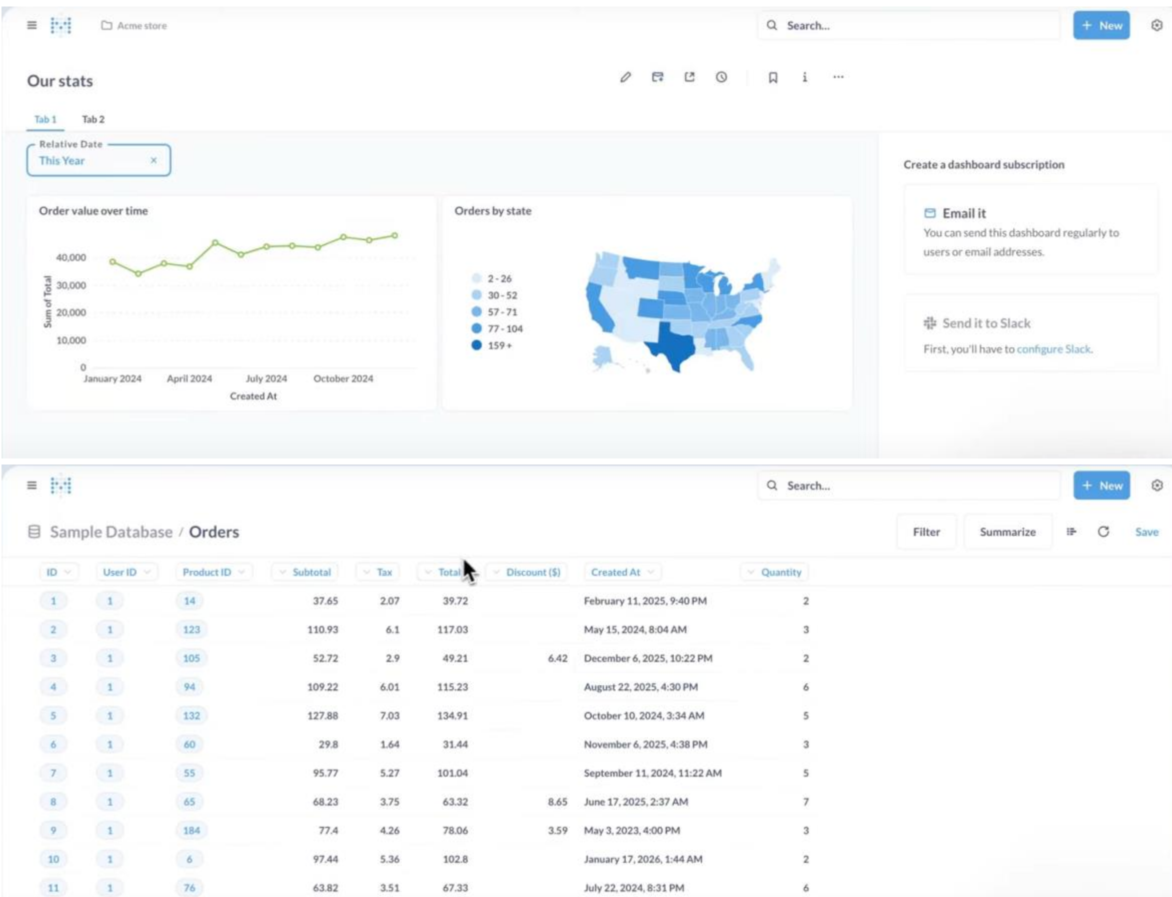

Figure 6 – Metabase Dashboard

In addition to the admin dashboard and raw MongoDB data views, Metabase provides options to create different individual views of the user metric data as shown in Figure 7. These views can be put together on the dashboard, providing the admins a comprehensive overview of all Eddy3D backend statistics.

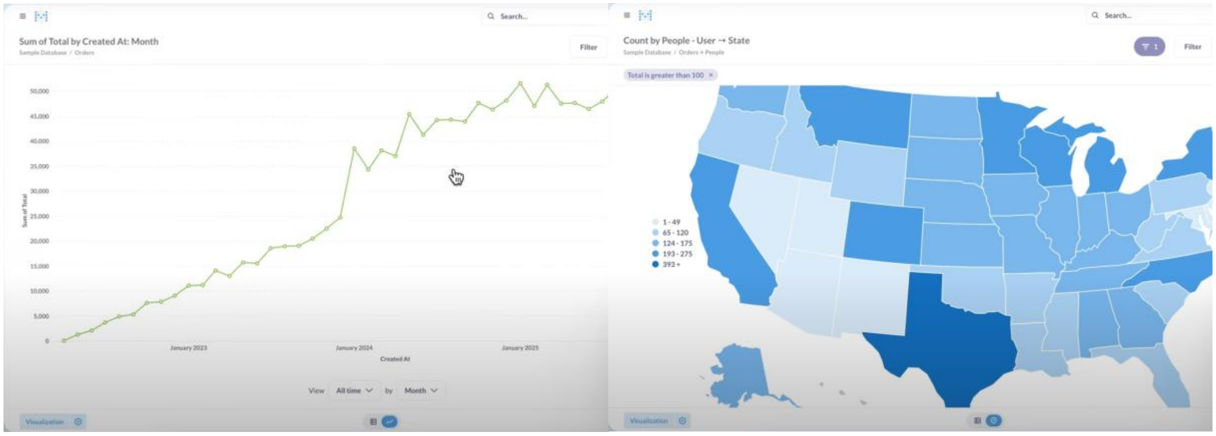

Figure 7 – Metabase Data Views

Appendix

Team member contributions:

Alexander Withka

- Team project manager
- Large contributor to group cohesion, client communication, overall project scope, and various components within the project codebase
- awithka3@gatech.edu

Abhinn Vegesna

- Large contributor to overall project scope alignment and many aspects of the project codebase
- avegesna8@gatech.edu

Rohit Dudala

- Large contributor to project codebase and positively contributes to team alignment
- rdudala3@gatech.edu

Benjamin Dai

- Contributor to project codebase and main documentation author/scribe
- bdai@gatech.edu