College of Computer Studies | Software Technology Department

MISSION MINISTRIES PHILIPPINES

System Documentation

Team Number	3
Section	S11
Team Members	Azmi, Anwar
	Concio, Tean Jeremy
	Encinas, Robert Joachim O.
	Gamboa, Rafael
	Go, Daphne Janelyn L.
	Inocencio, Rey Vincent H.
	Manlises, Maria Monica
	Pecson, Richard John Jr. Q.

Date Updated	November 25, 2023

Contents

1 Document Management

This paper details the system documentation of the proposed website application for Mission Ministries Philippines Inc. This document will detail the architecture, components, and technical design of the application to aid the transition for future developers, administrators and other stakeholders in further improving the current system.

The main objective of this document is to provide a detailed description of the software features - detailing the capabilities and limitations of the current implementation of the website application. The technical information provided in this document will serve as a manual for developers and other technical stakeholders to understand the step-by-step process as to how the application was developed from start to finish and how it can be scaled further according to the organization's needs.

1.1 Contributors

Please provide details of all contributors to this document

Role	Name
Product Owner	Daphne Janelyn Go
Software Developer	Robert Joachim Encinas
Software Developer	Tean Jeremy Concio
Software Designer and Developer	Richard John Jr. Pecson

2 Overview

The proposed software aims to develop a website application that automates the registration and enrollment processes of the partner organization in order to streamline operations, reduce human-generated errors, and improve efficiency.

The specific objectives of the software are as follows:

- To provide a user-friendly interface for administrators, teachers, and students to easily navigate the system
- To provide a facility for students to enroll in the program and in additional modules
- To provide a centralized database system to store all student and module enrollment data for easy access and management
- To provide a digital registration form with dynamic fields to capture necessary data needed for all incoming student enrollees
- To automate approval workflows, transcript requests, and faculty assignment processes, reducing manual intervention and delays
- To generate various reports for transcript, enrollment and registration processes
- To incorporate real-time error checking and validation mechanisms to reduce data entry errors

2.1 Service Description

This web app streamlines the administration of a training program. It features a user-friendly Registration Form, Faculty Account Creation, and an Admin Dashboard for overseeing accounts and approvals. Students access a dedicated dashboard for grades, enrollments, and transcript requests. Admins manage student records, and module details, and can add new modules. The system enhances efficiency with features like module enrollment approval, downloadable databases, and detailed student information.

The following section details the specific features of the implemented website application:

1. Registration Form for Training Program (via Sign Up Page)
2. Account Creation for Faculty
3. Admin Dashboard (Present only in Admin View)
    a. Faculty and Student Account Approval
    b. Transcript Requests
    c. Module Enrollment Approval
4. Student Dashboard
    a. Record of Grades Acquired
    b. Previous Enrollments
    c. Enroll in Additional Modules
    d. Request for Transcript

5. Enrollment Records (Present only in Admin View)
    a. List of Students filtered based on:
        i. Status
        ii. Year of Enrollment
        iii. Modules Enrolled Currently
    b. Student Details (Clickable Name to show details)
        i. Add Bills and/or Payments to Module Enrollments for a specific student
        ii. Download Database containing student details
6. Student Records (Present in Admin and Faculty View)
    a. List of Students per Module
    b. Download Database containing Module details
7. Module Page
    a. Add New Modules for a New School Year
    b. Assign Teachers to New Modules

All implemented features are based on the improved business process proposed in the figure below.

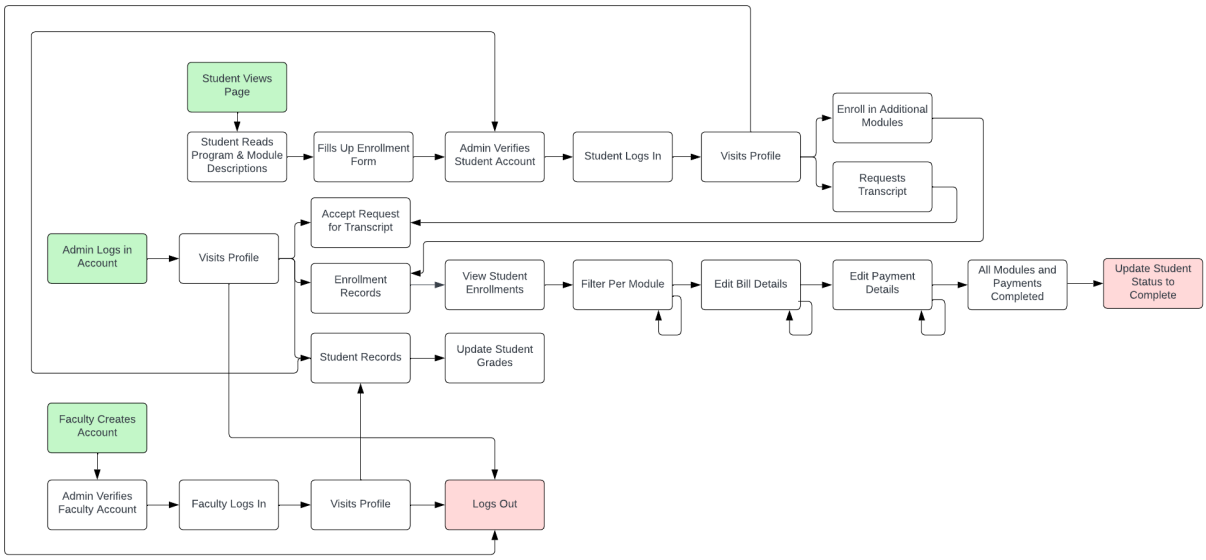


Figure 1. Revised Business Process based on Proposed Software Solution

2.2 Data Model

The application uses a relational database (PostgreSQL) that consists of 10 tables. The tables and their relationships are shown in the database diagram shown below.

2.3 Technology

The following table details the tech stack used for the website application along with the specific version used to ensure software compatibility across all devices.

	Tech Used	**Version**
General	Node	18.0.0
Build Tool	Vite	4.4.9
Frontend	Vue	3.3.4
Frontend	Flowbite	2.0.0
Frontend	Tailwind	3.3.3
Backend (ORM)	Prisma	5.4.2
Backend (Database)	PostgreSQL	16.0.0

Below are installation guidelines for the provided tech stack, including Node, Vite, Vue, Flowbite, Tailwind, Prisma, and Postgres when running the program locally.. Additionally, instructions for installing nodemon and running `npm install` are also included.

1. **Node.js and npm**
    - Install **Node.js** from the official website: Node.js
    - **Version**: 18.0.0
    - **npm**: npm is included with Node.js. After installing Node.js, npm will also be installed.
2. **Running `npm install`**
    - In your project directory (both client and server), run the following command to install the project dependencies:
      npm install
    - This command will read the `package.json` file in your project and install all the required dependencies listed there.

However, if you want to install the specific version used in the technical dependencies mentioned above, listed below are instructions for installing the specific versions used in the project:

3. **Build Tool - Vite**
    - **Vite**: Install Vite using npm:
      npm install -g create-vite
4. **Frontend - Vue, Flowbite, Tailwind**
    - Install **Vue** globally using npm:
      npm install -g vue@3.3.4
    - Install **Flowbite** using npm:
      npm install flowbite@2.0.0
    - Install **Tailwind** CSS using npm:
      npm install tailwindcss@3.3.3
5. **Backend - Prisma**
    - Install Prisma using npm:
      npm install -g prisma@5.4.2

The figure below illustrates the deployment architecture used for the website application:

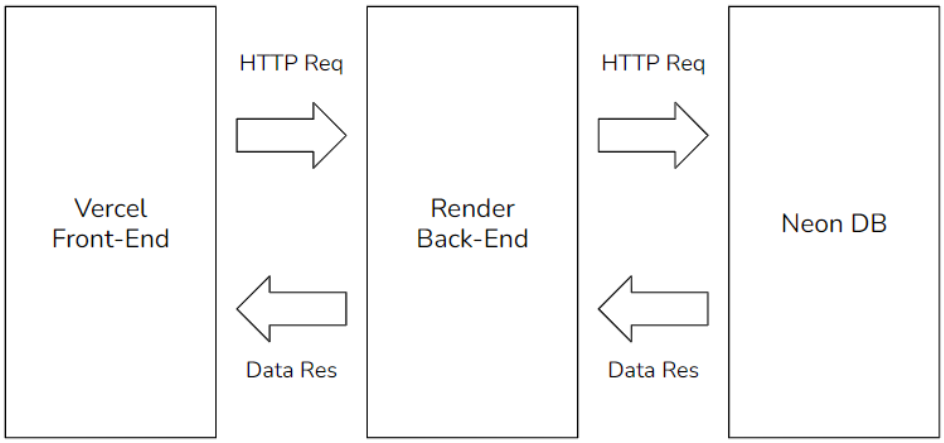


Figure 2. Deployment Architecture

The front end of the website application is hosted via Vercel. Due to its serverless architecture, the backend side had to be deployed using Render. Despite Vercel's support for Postgres, due to its strict limitations in terms of read-and-write storage, the database schema had to be hosted via Neon DB for a more stable and larger read-and-write storage.

2.4 Development Tools

For the development of our application, Visual Studio Code was primarily used as the integrated development environment (IDE). This versatile IDE provides a rich set of extensions that help, facilitate seamless coding, debugging, and version control integration. To ensure the same code formatting, Prettier - Code Formatter v. 10.0.0 is a recommended extension to be installed before using the IDE.

To streamline our deployment process, we have automated deployment using platforms such as Vercel, Render, and Neeon. These tools enable swift and reliable deployment of our application, ensuring that the latest changes are seamlessly propagated to production.

When it comes to building the local environment for debugging, developers can start the server using **'npm run start'** and the client with **'npm run dev.'** To aid in debugging the code for future development, in the latter section, common bugs to look out for will be listed for your reference.

2.5 Interfaces and services

The following tables detail the APIs used for the website application. This will be divided into two sections, client and server side.

Server Side

API Name	Description
Bcrypt	Bcrypt is a widely-used library for hashing passwords in a secure manner. It employs a one-way cryptographic hash function to convert plain text passwords into a hashed representation.
Body Parser	Body Parser is a middleware for Express.js that facilitates the parsing of incoming request bodies. It extracts data from the request payload and makes it accessible in the form of a JavaScript object, simplifying the handling of data submitted through HTTP POST requests.
Cors	Cors (Cross-Origin Resource Sharing) is a middleware that enables secure cross-origin data transfers between a web application and servers. It manages HTTP headers to control and allow or restrict access to resources on different domains.
Dotenv	Dotenv is a zero-dependency module that loads environment variables from a .env file into the process environment.

Express	Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. It simplifies the creation of server-side applications and APIs, offering a streamlined and expressive development experience.
Express Validator	Express Validator is a middleware for Express.js that provides validation and sanitization of incoming request data. It helps ensure that the data sent to the server meets specific criteria, enhancing the security and reliability of the application.
Google APIs	Google APIs refer to various application programming interfaces provided by Google, allowing developers to integrate their applications with Google services such as Google Maps, Google Drive, and Google Calendar, among others, particularly used for Gmail in the application.
Helmet	Helmet is a middleware for securing Express.js applications by setting various HTTP headers to mitigate common security vulnerabilities. It helps protect the application against attacks such as cross-site scripting (XSS) and clickjacking.
Json2CSV	Json2CSV is a library that facilitates the conversion of JSON data to Comma-Separated Values (CSV) format. It is particularly useful when working with data interchange between systems that use different formats.
JsonWebToken	JsonWebToken (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. It is commonly used for authentication and authorization purposes, providing a secure way to transmit information between parties.
Jszip	Jszip is a JavaScript library for creating, reading, and editing zip files. It allows developers to work with compressed archives in the browser, providing functionality to package and extract files in a convenient manner.
Morgan	Morgan is a logging middleware for Express.js that simplifies the process of logging HTTP requests. It provides a configurable logging format and helps in monitoring and debugging web server activities.
Nodemailer	Nodemailer is a module for Node.js applications that facilitates the sending of emails. It supports various transport methods, including SMTP, making it versatile for integrating email functionality into

	applications.
Nodemon	Nodemon is a utility that monitors for changes in files in a Node.js application and automatically restarts the server when changes are detected.

Client Side

API Name	Description
Axios	Axios is a popular JavaScript library that facilitates the making of HTTP requests from the browser or Node.js. It supports promises and provides a simple and consistent interface for performing asynchronous operations, making it widely used for data fetching and interaction with RESTful APIs.
Flowbite	Flowbite is a modern CSS framework designed to streamline the development of responsive and visually appealing user interfaces. It provides a collection of pre-designed components and utility classes, enabling developers to create consistent and responsive web designs efficiently.
Jszip	Jszip is a JavaScript library for creating, reading, and editing zip files. It allows developers to work with compressed archives in the browser, providing functionality to package and extract files in a convenient manner.
Pinia	Pinia is a state management library for Vue.js applications. It offers a flexible and efficient way to manage the state of a Vue.js application by providing a centralized store and reactive state management, enhancing the scalability and maintainability of Vue.js projects.
Autoprefixer	Autoprefixer is a PostCSS plugin that automatically adds vendor prefixes to CSS rules. It ensures cross-browser compatibility by analyzing the CSS code and applying the necessary vendor-specific prefixes, saving developers the manual effort of writing and maintaining these prefixes.
Postcss	PostCSS is a versatile tool for transforming styles with JavaScript plugins. It can be used for tasks such as linting, minification, and prefixing.

As for the application itself, it only provides one API which is used to communicate with the PostgreSQL database server that is hosted on Neon. The API is hosted on Render at https://mmp-ece-server-zrn4.onrender.com/ and is built using NodeJS and Express. The API itself has 11 main routes all of which share the common prefix of /API. The routes themselves can be seen in the table below.

Route	Prefix	Function
Admins	/admins	Contains the endpoints to perform CRUD operations on the Admins table of the database.
Auth	/auth	Contains the login endpoint of the application.
Bills	/bills	Contains the endpoints to perform CRUD operations on the Bills table of the database.
Download	/download	Contains the endpoints for the various database download functionalities of the application.
Module Details	/module_details	Contains the endpoints to perform CRUD operations on the Module_Names and Module_Details tables of the database.
Module Enrollments	/module_enrollments	Contains the endpoints to perform CRUD operations on the Module_Enrollments table of the database.
Modules	/modules	Contains the endpoints to perform CRUD operations on the Modules table of the database.
Payments	/payments	Contains the endpoints to perform CRUD operations on the Payments table of the database.
Students	/students	Contains the endpoints to perform CRUD operations on the Students table of the database.
Teachers	/teachers	Contains the endpoints to perform CRUD operations on the Teachers table of the database.

TOR Requests	/tor_requests	Contains the endpoints to perform CRUD operations on the TOR_Requests table of the database.

Each endpoint in the API is only accessible by certain user types. There are 4 user types that are defined for this API and each of them are denoted by a number. These user types are as follows in the format of "user type (number code)".

- Guest (0)
- Student (1)
- Teacher (2)
- Admin (3)

These permission numeric codes are obtained from the login endpoint of Auth. If a login is not detected on the device or browser, the default permission is level 0. The URL of all the endpoints for each of these routes, HTTP methods, their functionalities, and allowed permission codes are detailed in the tables below.

Admins

URL	HTTP Method	Allowed Types	Functionality
/:admin_id	GET	3	Retrieves an admin by id
/	POST	3	Creates a new admin
/:admin_id	PATCH	3	Updates an admin's info by id
/update_password/:admin_id	PATCH	3	Updates an admin's password by id
/:admin_id	DELEE	3	Deletes an admin by id

Auth

URL	HTTP Method	Allowed Types	Functionality
/login	POST	0, 1, 2, 3	Logs in a user

Bills

URL	HTTP Method	Allowed Types	Functionality
/:bill_no	GET	3	Retrieves an bill by id
/	GET	3	Retrieves all bills
/unpaid	GET	3	Retrieves all bills that are not fully paid
/module/:module_name	GET	3	Retrieves all bills associated with a module
/bill/:module_name/:school_year/:student_id	POST	3	Creates a bill
/:bill_no	PATCH	3	Updates a bill by id
/delete/:bill_no/:module_name/:school_year/:student_id	DELETE	3	Deletes a bill and all its associated payments

Download

URL	HTTP Method	Allowed Types	Functionality
/all	GET	3	Downloads all tables as CSVs
/modules	GET	3	Downloads all module data as CSVs
/student/:student_id	GET	3	Downloads all data of a student as CSVs

Module Details

URL	HTTP Method	Allowed Types	Functionality
/info/:module_name	GET	3	Retrieves the details of a module
/all	GET	3	Retrieves all module names
/all/details	GET	3	Retrieves all module details

/	POST	3	Creates a new module detail and module name entry into their respectives table
/:module_name	PATCH	3	Updates a module's detail

Module Enrollments

URL	HTTP Method	Allowed Types	Functionality
/enrollment/:student_id/:module_name/:school_year	GET	1, 2, 3	Retrieves an enrollment by primary key
/student/:student_id	GET	1, 3	Retrieves all enrollments of a student
/active/:student_id	GET	1, 3	Retrieves all active enrollments of a student
/balance/:student_id	GET	3	Retrieves all finance related info a student
/passed/:student_id	GET	3	Retrieves all passed enrollments of a student
/enrollments/:school_year	GET	3	Retrieves all enrollments in a year
/approval	GET	3	Retrieves all enrollments that are for approval
/enrollments/:module_name/:school_year	GET	2, 3	Retrieves all enrollments in a module
/	POST	1, 3	Create a new enrollment
/:student_id/:module_name/:school_year	PATCH	3	Update all info of an enrollment by primary key
/status/:student_id/:module_name/:school_year	PATCH	3	Update the status of enrollment
/grade/:student_id/:module_name/:school_year	PATCH	2, 3	Edit the final grade of an enrollment
/:student_id/:module_name/:scho	DELETE	3	Delete an enrollment

ol_year			

Modules

URL	HTTP Method	Allowed Types	Functionality
/module/:module_name/:school_year	GET	3	Retrieves a module by primary key
/	GET	3	Retrieve all modules
/school_year/:school_year	GET	3	Retrieves all modules for a year
/student/:student_id	GET	1, 3	Retrieve all modules for a student to take
/program/:program	GET	3	Retrieve all modules of a program
/teacher/:teacher_id	GET	2, 3	Retrieve all modules that a teacher is currently teaching
/report/:module_name/:school_year	GET	3	Retrieve an enrollment report for a module
/report/:school_year	GET	3	Retrieve enrollment reports for all modules in a year
/	POST	3	Create a new module
/:module_name/:school_year	PATCH	3	Update a module
/teacher/:module_name/:school_year	PATCH	3	Updates the teacher of the module
/:module_name/:school_year	DELETE	3	Deletes a module

Payments

URL	HTTP Method	Allowed Types	Functionality
/:or_no	GET	3	Retrieves a payment by id

/bill/:bill_no	GET	3	Retrieves all payments for a bill
/	POST	3	Creates a payment
/:or_no	PATCH	3	Updates a payment
/:or_no	DELEE	3	Deletes a payment

Students

URL	HTTP Method	Allowed Types	Functionality
/id/:student_id	GET	1, 2, 3	Retrieves a student by id
/year/:year	GET	3	Retrieves all student records for a specific year
/module/:module_name/:school_year	GET	2, 3	Retrieves all students for a particular class
/	GET	3	Retrieves all students
/status/:status	GET	3	Retrieves all students by status
/id_name	GET	3	Retrieves all student ids and names
/unpaid_bills	GET	3	Retrieve all student id and names with not fully paid bills
/in_progress	GET	3	Retrieves all student id and name with in progress enrollments
/module/:module_name	GET	3	Retrieves all students enrolled in a specific module
/	POST	0, 3	Creates a new student
/:student_id	PATCH	3	Updates a student
/status/:student_id	PATCH	3	Updates the status of a student

/update_password/:student_id	PATCH	1	Updates the password of student
/:student_id	DELETE	3	Deletes a student

Teachers

URL	HTTP Method	Allowed Types	Functionality
/:teacher_id	GET	2, 3	Retrieves a teacher
/	GET	3	Retrieves all teachers
/all/:status	GET	3	Retrieves all teachers of a certain status
/	POST	0, 1, 2, 3	Creates a new teacher
/:teacher_id	PATCH	2, 3	Updates a teacher
/status/:teacher_id	PATCH	3	Updates the status of a teacher
/update_password/:teacher_id	PATCH	2	Updates the password of a teacher
/:teacher_id	DELETE	3	Deletes a teacher

TOR Requests

URL	HTTP Method	Allowed Types	Functionality
/:req_id	GET	1, 3	Retrieves a TOR Request by id
/	GET	3	Retrieves all TOR Requests
/students/:student_id	GET	3	Retrieves all TOR Requests of a student
/year/:year	GET	3	Retrieves all TOR Requests of a year
/all/:status	GET	3	Retrieves all TOR Requests of

			a certain status
/	POST	1, 3	Create a new TOR Request
/:req_id	PATCH	1, 2, 3	Updates a TOR Request's status
/:req_id	DELETE	3	Deletes a TOR Request

2.6 Access, Authentication and Authorisation

· This section details all authentication details needed to access all deployment services used to deploy the application.

- Web-based (**https://mmp-ece-training-program.vercel.app/**)
- Email Credentials
  - **Account name:** mmp.web.service@gmail.com
  - **Password:** mmpece123
- Github Credentials (Sign in through Google)
  - **Account name:** mmp.web.service@gmail.com
  - **Password:** mmpece123

Sign in to the Github account first before signing in to any of the deployment platforms. **For any changes made to the code, ensure that all changes are pushed to the main branch of the specified GitHub repository.**

- Vercel Credentials (https://vercel.com/)
  - Sign in using Github
  - All credentials in Github should remain as is.
- Render Credentials (https://dashboard.render.com/)
  - Sign in using Github
  - All credentials in Github should remain as is.
- Neon Credentials (https://neon.tech/)
  - Sign in using Github
  - All credentials in Github should remain as is.

For all changes related to the front end and back end, in order for modifications to reflect on the website, manual redeployment of Render is done by clicking its Manual Deployment drop-down and then clicking Deploy latest commit. There is no need to manually

redeploy for Vercel as this is triggered automatically for each commit in the main Github repository.

For any changes related to restructuring the database you may edit the schema.prisma file found in the server folder of the source code. To implement these changes in the love database you must first connect to the Neon database server on your machine by changing the values of 'DATABASE_URL' and 'DIRECT_URL' in the environment with the URLS provided by Neon. Then run the command "npx prisma generate" on the terminal in the server directory. This command will generate a prisma client that is connected to the Neon database. From here to implement the changes through a migration, run "npx prisma migrate dev" in the command line interface of your local machine. Note that this may have side effects, such as destroying data in the database. Ensure that you have a backup of your data before doing this. Furthermore, for specific changes in the data stored in the database, you may directly edit this through a SQL Query in Neon's built in SQL Editor.

2.7 Delivery

As mentioned above, the website is hosted via Vercel. Hence, changing the domain of the website will be done through the settings in Vercel. With this, Render only needs to be manually redeployed when there are pertinent changes in the program functionalities committed in the main Github repository. In the same way, pertinent changes to the database structure and data should be done through Neon.

3 Support details

3.1 Documentation

This section details the external documentation of Prettier, which serves as the code format or standard that was used for the code.

```
//                                                    60 chars -->   |
function makeComponent() : int {
  return {
    longCall() {
      complicatedFunction(importantArgument(), secondaryArgument())
      weirdStyle({ prop: 1 },
        1, 2, 3);
    },
    render() {
      const user = {
        name: "James"
      };
      return <div>
        hello ${name}! JSX is <strong>supported</strong>
      </div>;
    }
  };
}
```

Figure 3. Unformatted Code

```
//                                                    60 chars -->   |
function makeComponent(): int {
  return {
    longCall() {
      complicatedFunction(
        importantArgument(),
        secondaryArgument()
      );
      weirdStyle({ prop: 1 }, 1, 2, 3);
    },
    render() {
      const user = { name: "James" };
      return (
        <div>
          hello ${name}! JSX is <strong>supported</strong>
        </div>
      );
    }
```

Figure 4. Formatted Code using Established Formatting Rules

Figure 3 and 4 details some of the formatting rules reinforced in the code, such as maximum line length as well as tab and space formatting. Prettier reinforces a coding standard, but is open to the customization of the developers based on their preferred customized standard as shown in the figure below.

JSON:

```
{
  "trailingComma": "es5",
  "tabWidth": 4,
  "semi": false,
  "singleQuote": true
}
```

Figure 5. Basic Configuration of Prettier in JSON Format

Based on this syntax, the developer can configure his/her preferred formatting in Prettier to be automatically reinforced throughout his whole code through the help of the Prettier extension.

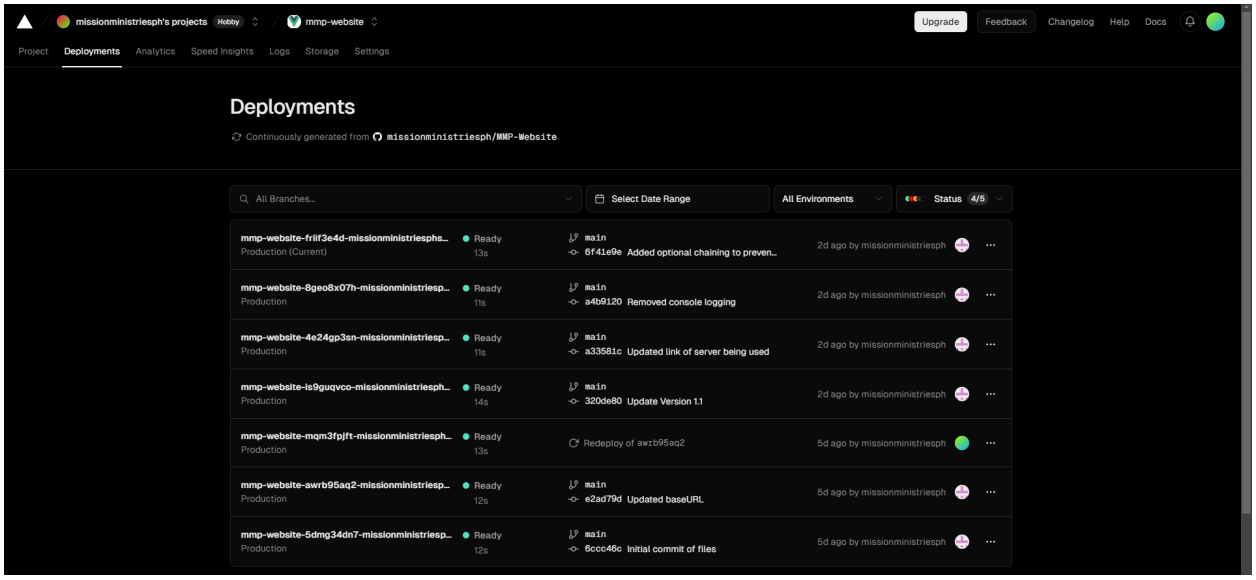
For further information, refer to Prettier's official documentation (https://prettier.io/docs/en/).

3.2 Standard tasks

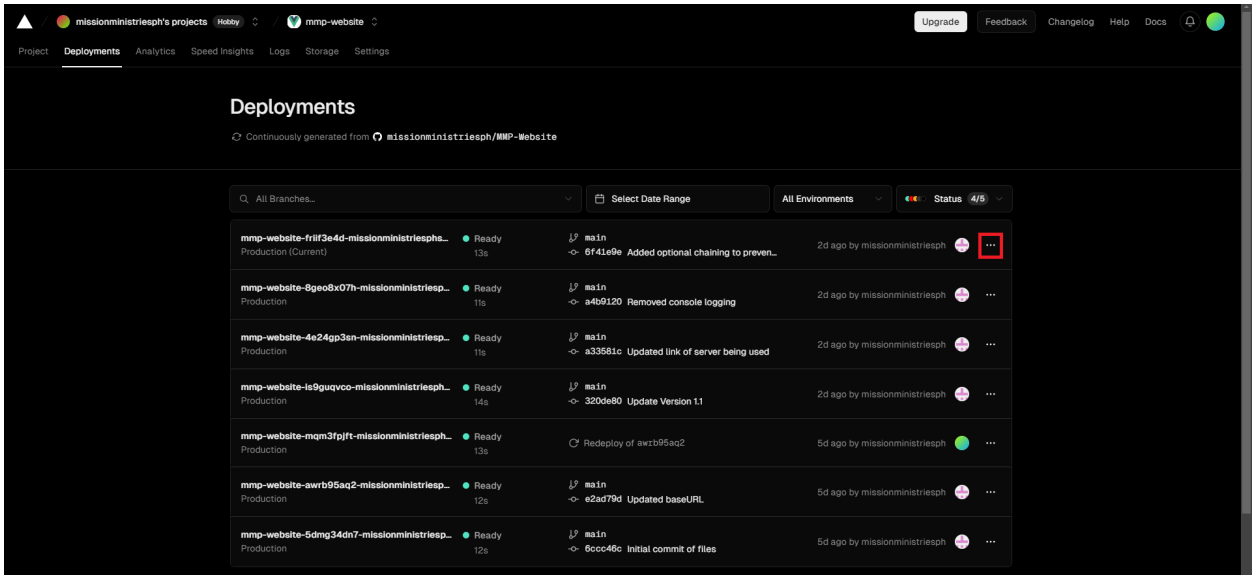
Should the automatic deployment of Render and Vercel not function for new changes, a manual deployment must be done. Below are the steps to perform manual deployment for Render and Vercel respectively.

Vercel

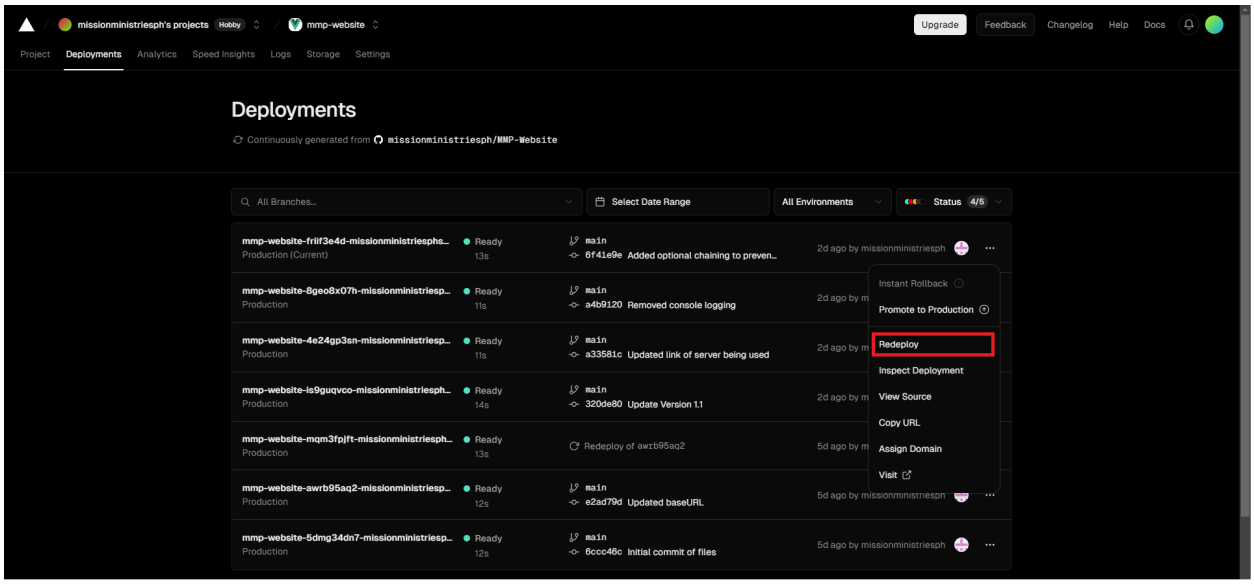
1. Navigate to the deployment tab of the Vercel project.



2. Click the three dots to the right of the deployment with the correct branch (usually the latest deployment).



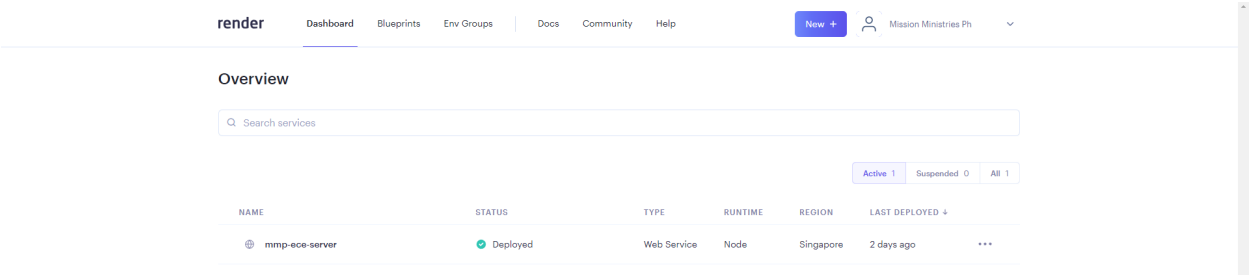
3. Click redeploy



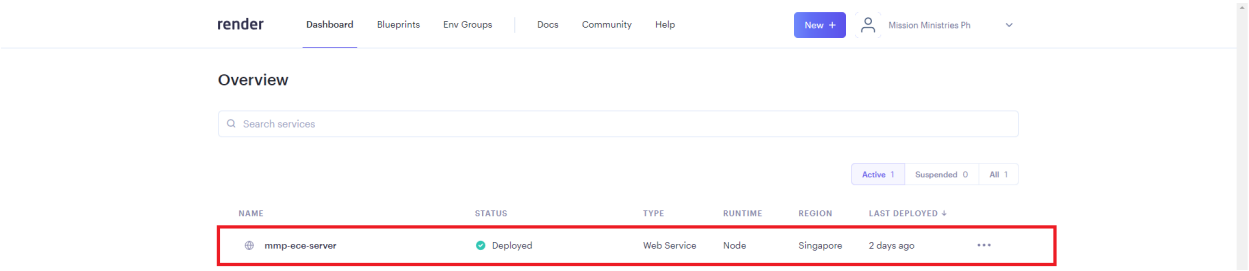
4. Vercel should now deploy the previous failed deployment

Render

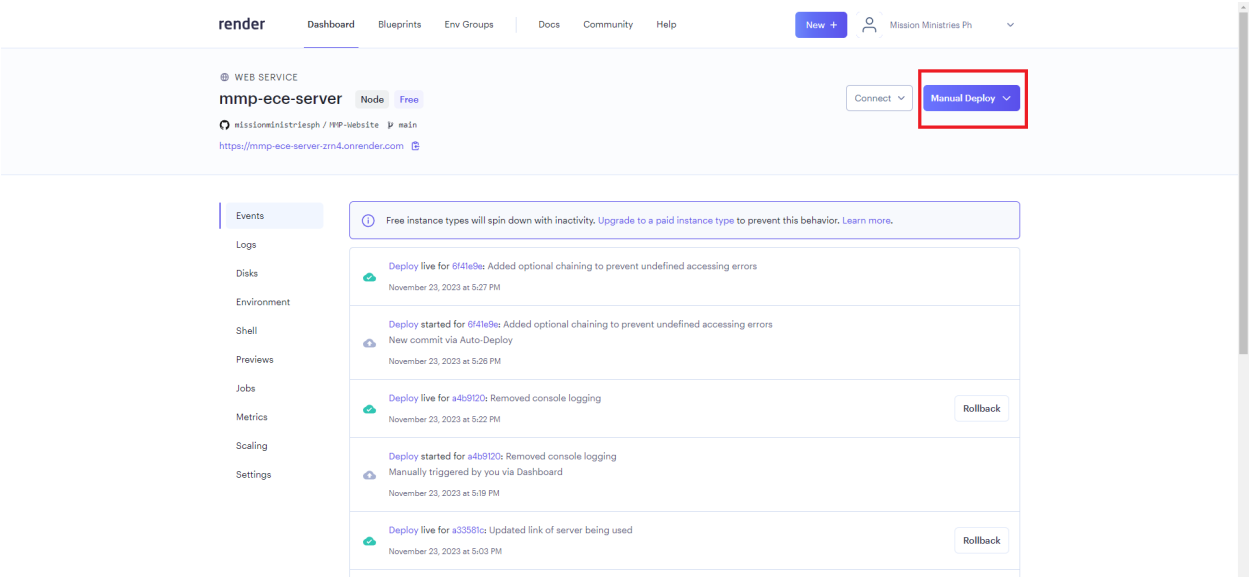
1. Navigate to the Render website



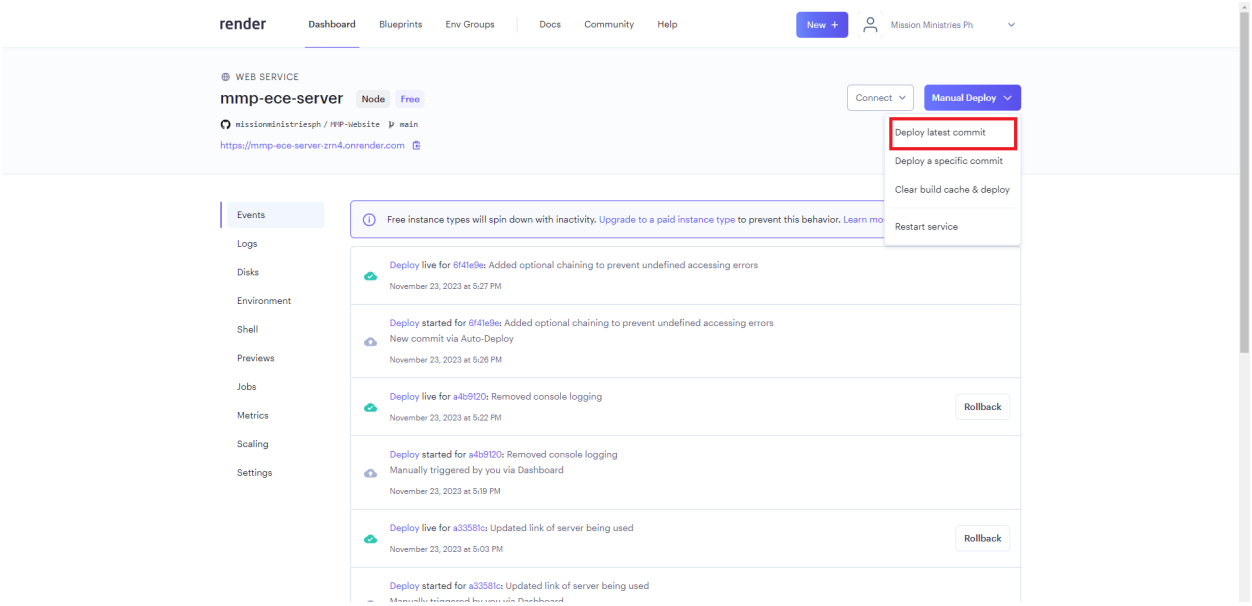
2. Click the deployment of the web API



3. In the deployment's page click "Manual Deploy"



4. Click "Deploy Latest Commit"



3.3 Troubleshooting Guide

Below listed are some common errors that may be encountered and how to fix them

Error: Invalid prisma.(...) invocation

Explanation: These types of error messages means that the issue lies with the database updating/querying through Prisma. Some of the common issues are as follows:

- Violation of primary key constraint - the primary key of the data being inserted into a table already exists. This would most likely be caused by the automatic ID assignment functions (for the student, teacher, admin, bill, payment, and tor request endpoints) or by user input error for the other tables.
- Violation of foreign key constraint - the data being inserted into the table is referencing another table's field but the referenced data does not exist. This would most likely be caused by the wrong insertion order of data or wrong deletion order of data.
- Invalid/unknown argument - the syntax of the Prisma command in the API is wrong.

Solution: Each of the variants of this error are solved in different ways.

- Violation of primary key constraint can be fixed by changing the code for the automatic id generation function or ensuring correct user inputs in the frontend.

- Violation of foreign key constraints can be fixed by reordering the insertion or deletion of data to ensure that referential integrity is maintained.
- Invalid/unknown argument is a highly case to case basis as this is an issue with the syntax of the command. Refer to the Prisma documentation (https://www.prisma.io/docs) for more information

Error: The http response is status 500 and the response body's data contains errors: {...}

Explanation: These types of error messages means that the request body being passed to the post endpoint has erroneous data formatting or is missing some fields.

Solution: Add the missing fields to the request body or change the formatting of the data based on the error message returned.

Error: The loading spinner has been displaying for an extended period of time

Explanation: This error may be because of 3 causes

- The Render server
- An error in the retrieval of data
- Your internet connection

Solution:

- The Render server usually takes a while to load after a period of inactivity, this is normal and would take usually 5 to 10 minutes. If the error persists, please check the console of your Render server deployment for any crashes.
- An error in the retrieval of data can be either in the frontend or the backend as such fixing this issue would be on a case by case basis. This error may either be a minor syntax error in the returned fields of the API or a large issue with the structure of the data being retrieved.

Error: A certain page is not loading but other pages work

Explanation: Check the console of your browser for the issue. If the issue stated in the console mentions something about a field in the HTML being unreadable, the issue may lie in using the "this" keyword in the template of your .vue files.

Solution: Remove the “this” keyword in the reactive data accessed in the template html code. Note that for all variables or functions in the HTML the “this” keyword must be omitted, whereas all variables or functions in the script tags must contain the “this” keyword.

Error: No data is being loaded

Explanation: This may be because of 3 cases: 1) There is no data in the database deployed, 2) The API is not working properly, 3) The Neon database is down.

Solution: The solutions for each case are as follows

1. Check the deployed database, specifically the tables tab if the data is there or not.
2. Check the console of the deployment of Render to see if the server has crashed.
3. Check the official channels of Neon for outages.

Error: Changes are not being saved

Explanation: This may be the case if either the frontend code is wrong, the backend code is wrong, or the server is down.

Solution: The troubleshooting steps are as follows

1. Check first if the server is running on Render.
2. If the server is running check the network tab of the dev tools of your browser to see the error with the request to help narrow down the search.
3. Check the frontend code of the vue component, specifically the code for the request by logging the updated data being sent.
4. Check the backend code of the appropriate endpoint to see if the data sent is being received correctly.

4 Future Developments

Change Password Functionality

This feature is highly recommended to be implemented as all endpoints for updating a user's password are in place. Additional considerations would be an email prompting the user to verify the change in password for extra security.

Remember Me Functionality

This feature is in line with session management. However, since there was no two factor authentication implemented, a remember me functionality would instill more redundancy in terms of user experience. Nonetheless, this may be implemented to reinforce further security on user data management and the security of the application as a whole.

Document and Image Upload

Due to limitations in DB Storage, a document and image upload was not possible in this current iteration of the application. If the organization decides to purchase a Database with capabilities to store large amounts of binary data such as the Amazon S3 Simple Cloud storage, a document and image upload is highly recommended to further streamline the registration process of all student enrollments.

Generate PDF Functionality

Exporting all data into pdf is another functionality that may be implemented. This is highly similar to the export database functionality currently in place. However, some considerations would be designing a predefined PDF format and using an external third party API to aid the process of exporting.