SALN Progressive Web App

Group: LAMIG - AGCAOILI, GUIANG, PEÑAFLORIDA

Client: Rommel Feria

Project blog/website: https://shirokamiqq.github.io/cs191-LAMig-GroupBlog/

Course: CS 192 TBC/HQR1

Instructor: Ligaya Leah Figueroa

System Description

Points of Contact

SALN Progressive Web App Development Team

- AGCAOILI, Lucas Miguel V. - lvagcaoili@up.edu.ph
- GUIANG, Miguel - maguiang@up.edu.ph
- PEÑAFLORIDA, Aeron Dann - appenaflorida@up.edu.ph

Security

OTP Verification

The login and registration processes for the client app make use of an OTP for user verification. They work as follows:

1. The user enters their email to log into/register their account.
2. The user receives a 4-word OTP via email to verify their account.
3. The user enters the OTP, giving them access to the app.

UUID and Email Identification

To uniquely identify users, the app makes use of a universally unique identifier (UUID) and email as the primary key for database operations.

Data Encryption in the Database

Account information and SALN data are encrypted using the built-in encryption functions in the Laravel framework of PHP.

Auto-Deleting Old Data

To ensure that the server does not contain sensitive information for long periods of time, SALNs which have not been updated in 5 days get automatically deleted from the database.

.env file

To configure the server, an `.env` file is needed. These file contains sensitive information, so they are not to be shared or pushed to any public repositories. These could be gotten from the developers stated in the Points of Contact section.

Computer Hardware

The project only requires a computer to run. While in development, the server is hosted on a computer with the `.env` file.

Eventually, the server will be done on the cloud using Digital Ocean. It will use the Linux operating system.

Support Software

To test the project properly, the dependencies for both the client and server side are listed down below:

- Client-side
  - marko/run (0.8.1) - development server for Marko projects
  - marko (6.0.82) - UI framework used to build the app's pages/components
  - pdf-lib (1.17.1) — used for creating and manipulating PDF files (e.g., generating SALN forms)
  - uuid (13.0.0) — generates unique IDs for records and internal identifiers
- Server-side
  - PHP (8.3.6) - main server-side programming language used to run the backend logic

- Laravel (12.33.0) - backend web framework used for routing, controllers, validation, database access, and API endpoints
- Laravel Sanctum (4.2.0) - provides a featherweight authentication system for SPAs and simple APIs.
- Composer (2.7.1) - PHP dependency manager used to install and update backend libraries/packages

Personnel

This section identifies the personnel roles required to maintain SALN Progressive Web App and the skills needed to perform routine maintenance, troubleshooting, and updates.

- Maintenance Roles and Responsibilities
  - Lead Developer
    - Acts as the main contact for the client
    - Decides what gets fixed first (urgent issues vs. nice-to-haves)
    - Keeps the team organized and makes sure releases actually happen
  - Backend Developer (must know PHP and Laravel)
    - Maintains the server-side features and API endpoints
    - Fixes issues related to OTP generation/verification and email sending
    - Handles database migrations and security updates
  - Frontend Developer (must know Marko, CSS, HTML, and JavaScript)
    - Fixes UI issues (forms, validation, layout, responsiveness)
    - Makes sure the PWA still behaves properly across browsers/devices
    - Maintains PDF-related features (generation/export) and client-side flow

Maintenance Procedure

Conventions

In general, the naming procedure used for variables in both the JavaScript (MarkoJS) client and PHP (Laravel) server sides is `camelCase`, with capitalization being used for `Class` names. Field names in JSON messages are also kept on `camelCase`. Additionally, `SCREAMING_SNAKE_CASE` is used for `.env` variables.

In terms of comments, since Javascript, and by extension MarkoJS, is not all that strict with type annotations, it would be helpful to at least comment them out before function definitions. For example:

```
/**
 * Register a new employee
 * @param {string} email
 */
export async function registerEmployee(email)
```

Maintenance Procedure

Upon downloading the GitHub repository, go into the root project directory. In there are two sub-directories: `saln-client` and `saln-server`. We need to set both of them up.

Client Setup

In setting up `saln-client`, all we really need to do is simply install the dependencies using `npm`. `npm` is the package manager of `Node.js`. You could get it from here if you don't have it yet, https://nodejs.org/en/download. Then, install the dependencies:

```
cd saln-client
npm install
```

Then, we could run the developer version of the MarkoJS client in side in `saln-client` directory via

```
npm run dev
```

This allows us to see the app in `localhost:3000`.

Server Setup

In setting up `saln-server`, we need to install MySQL and PHP/Laravel, and then make some changes to the local configuration.

Installing MySQL:

```
sudo apt update
sudo apt install mysql-server -y
sudo service mysql start
```

Installing PHP:

```
sudo apt update && sudo apt upgrade -y
sudo apt install php php-cli php-mbstring php-xml php-bcmath php-json php-zip php-curl php-mysql unzip curl git -y
```

Installing Composer, PHP's package manager:

```
sudo apt install composer -y
```

Installing Laravel:

```
composer global require laravel/installer
echo 'export PATH="$PATH:$HOME/.config/composer/vendor/bin"' >> ~/.bashrc
```

```
source ~/.bashrc
composer install
```

On MySQL, create the database that Laravel will connect to:

```
CREATE DATABASE saln_app_DB;
CREATE USER 'saln_user'@'localhost' IDENTIFIED BY 'saln_password';
GRANT ALL PRIVILEGES ON saln_app_DB.* TO 'saln_user'@'localhost';
FLUSH PRIVILEGES;
```

We also need to configure our Laravel instance to use that newly created database. In `saln-server/.env`:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=saln_app_DB
DB_USERNAME=saln_user
DB_PASSWORD=saln_password
```

We also want to generate a master key to be used for data encryption and create the database tables from the Laravel migrations:

```
php artisan key:generate
php artisan migrate:fresh
```

Lastly, we need to set up a cron job for scheduled tasks.

```
crontab -e
```

Then on the bottom of the file, put these commands:

```
* * * * * cd /Path/to/saln-app/saln-server && php artisan schedule:run >> /dev/null
2>&1
```

Finally, to run the server:

```
cd saln-server
php artisan serve
```

Verification Procedures

When developing on the front-end client, simply running the client `npm run dev` to start the app for visual inspection of the page's elements will do.

Alongside client testing, you would most likely be testing API accesses from the client side. To see all API calls while the Laravel server-side is running, you could look towards the terminal instance running `php artisan serve`.

Additionally, server-side errors could be seen in `saln-server/storage/logs/laravel.log`. This is also where errors coming from MySQL queries will be printed into. For further testing regarding MySQL, you could always log your user into the MySQL server and check the database directly.

Server-side error logs related to the NGINX web server could be found in `/var/log/nginx/error.log`.

Error Conditions

MySQL Errors:

Usually this is caused when the expected payload from client does not match what the Laravel server-side is expecting. This leads to missing fields in the MySQL query. Another potential cause is the other way around: Laravel is not what the client expects. For instance, you changed the schema of some table

in the `migrations` folder of Laravel, but forgot to run `php artisan migrate` to update the schema of the database in the MySQL server.

MarkoJS frontend not updating after code changes: The service worker caches the static Javascript files when the app is being served. This means that even if you refresh MarkoJS after making changes to the code, there is a possibility that the service worker would have already cached the previous code, making you use that said previous code. To fix this, delete the browser's cache at `localhost:3000` and unregister the service worker so that the new one could take its place.