Project Proposal for Repair Scheduling Website

HEMANTH NANDHAKUMAR
02145227

Introduction

The Repair Scheduling Website is a website to facilitate the repairing scheduling for the users by making it more convenient. It gives an easy-to-use method in choosing the available repair services, setting the appointment, reading the repair details description, and making secure payments. The platform is intended to cover various scopes of repairs and appeal to various types of customers starting from the representatives of electronics industry and car repairing shops up to home appliance fixing services This business proposal outlines the development of a flexible, safe, and functional web-based platform.

What?

The Repair Scheduling Website will:

1. Include a list of available repair services and the times that are suitable for repair.
2. Provide functionality that enables the users to schedule the existing and pending repairs depending on their availability.
3. Include the following: price estimates for repairs to be done, probable time taken for the repair process, and the individuals to carry out the repairs.
4. Enable secure payments.
5. File all the data of the user and the repair in a database.
6. Allow auditing of audit logs and detailed logging for help in tracing and debugging.

Why?

There is the increasing demand for easy-to-use interfaces that people and companies can use to arrange repair schedules. Also, use of traditional approaches is normally characterized by several disadvantages such as being manual and therefore time-consuming and fragmented in their execution. For instance, the Repair Scheduling Website will combine the process of payment, scheduling, and record keeping hence coming up with an integrated system that will help users solve this issue. It is evident that through the use of this platform, repair businesses can improve on customer satisfaction by providing clients with flexible real-time scheduling options as well as tracking on the repair services offered.

Supported Features

1. **User Interface (UI):** This, of course, implies the fully web-based UI accessible via major browsers – Chrome, Firefox, or MS Edge.
2. **User Authentication:** For enhanced and specific security and user-defined.
3. **Repair Listing:** Show the customers repair services being offered and the availability of slots for such a service.
4. **Repair Scheduling:** Select the available slots for the repair and fix an appointment.
5. **Repair Details:** Exhibit more information on the repair costs and services that it offers as well as the technicians.
6. **Payment Services:** Such capacities should include securely paying for the repairs through the integrated payment systems.
7. **Logging:** It also operates the detailed logging which can include: info, debugging, and tracing when needed for auditing and troubleshooting purpose.
8. **REST API:** Specify core functionalities (repair listing, scheduling, payment) as REST API.
9. **HTTPS Support:** Data transfer using the Localtunnel platform safely.

The Not Supported Features

1. **Mobile App Integration:** Although the site will have a ‘responsive’ layout, there won’t be a standalone mobile application at the outset.
2. **Offline Functionality:** Depending on the implementation, this platform will need an active connection to the Internet for scheduling and payments to occur in real time.

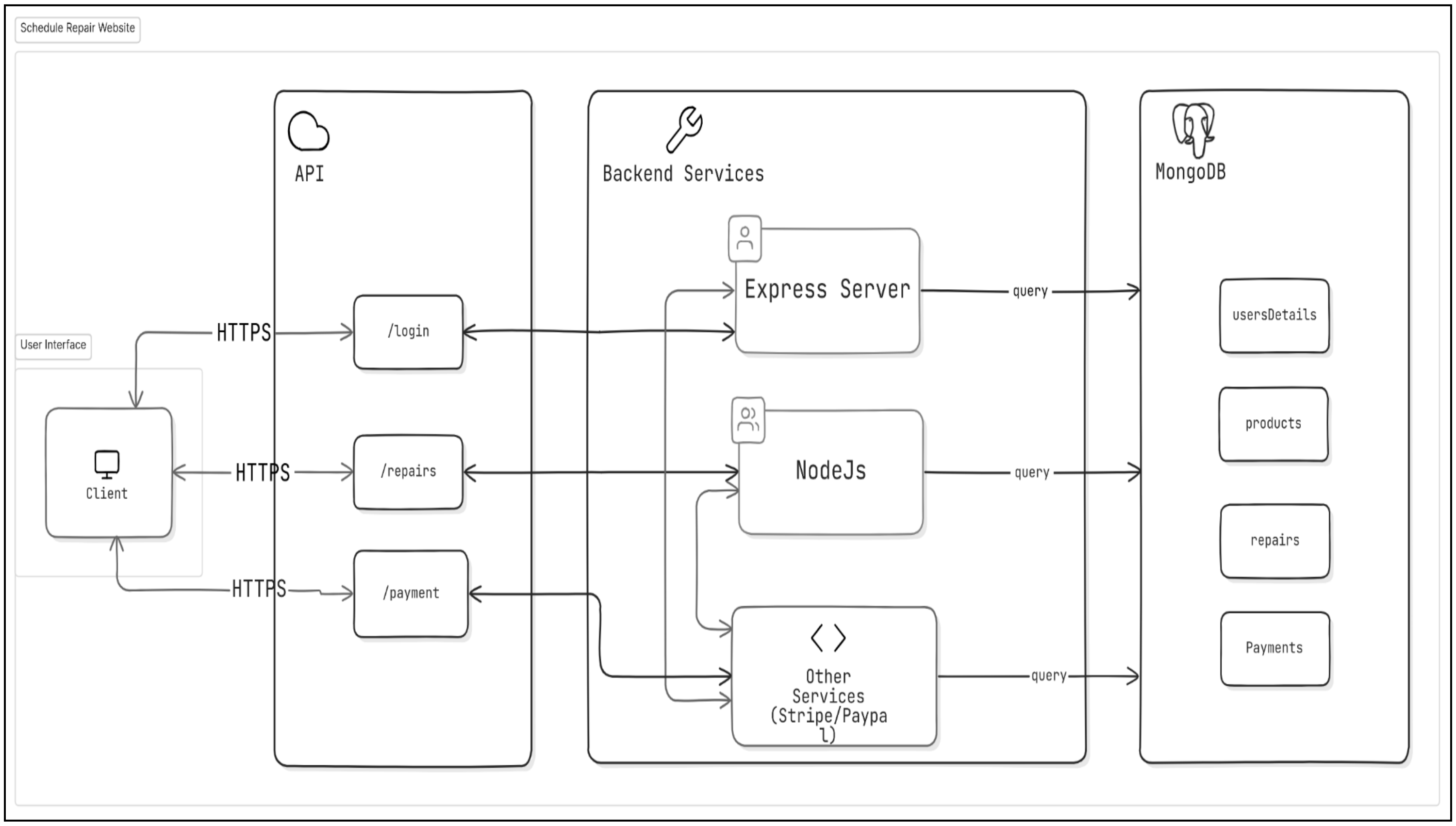
Future Planned Features

1. **Mobile App Development:** Suits with Android and iOS to ease the methods used in accessing information.

2. **Automated Reminders:** Push notifications for the scheduled maintenance and repair services through Short Message Service.
3. **Repair History and Recommendations:** Make it possible for the users to check their previous repair requests and recommend them to undertake next maintenance.
4. **Advanced Analytics Dashboard:** Supply the repair businesses with the trend and performance figures of the repair-related operations.

The How

Project Architecture



Components and Modules

Frontend:

UI elements for repair list views and schedules as well as payment forms.
Authentication.

Backend:

REST API services for create, read, update and delete for repairs, schedules, and payments.

Logging and auditing module.

The database interaction layer which interacts with MongoDB will be established.
Payment Module: Combination with other online payment systems.

Languages used For each Module

1. **Frontend**: HTML, CSS, JavaScript.
2. **Backend**: Node.js or Java.
3. **Database**: MongoDB (NoSQL).

3rd Party/Open-source Modules

Frontend:

1. React (UI library).
2. Bootstrap or Tailwind CSS (for styling).

Backend:

1. Express.js (for REST API in Node.js).
2. Mongoose (MongoDB interaction).
3. Winston or Morgan (for logging).
4. Passport.js or JWT (optional authentication).

Table of Licenses

Library/Module	License Type	Description
React	MIT	Frontend JavaScript library
Express.js	MIT	Web framework for Node.js
Mongoose	MIT	MongoDB object modeling for Node.js
Localtunnel	MIT	Tunneling service for local HTTPS URLs
Winston	MIT	Logging library for Node.js
Stripe	Paid	Payment service for online transactions

3rd Party Services/APIs

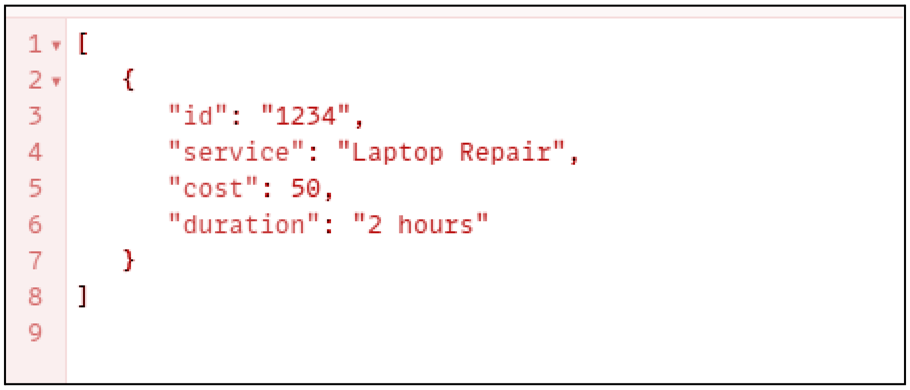
1. **Stripe/PayPal (Paid)**: Secure payment gateways for processing transactions.
2. **Localtunnel (Free)**: Provides HTTPS URL for local development environments.

REST API Endpoints

GET /api/repairs

Description: Get the list of available repairs.

Response:

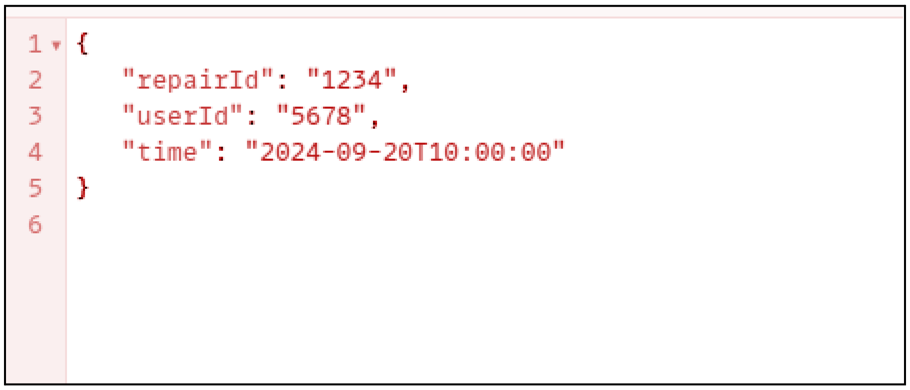


```json
[
    {
        "id": "1234",
        "service": "Laptop Repair",
        "cost": 50,
        "duration": "2 hours"
    }
]
```

POST /api/schedule

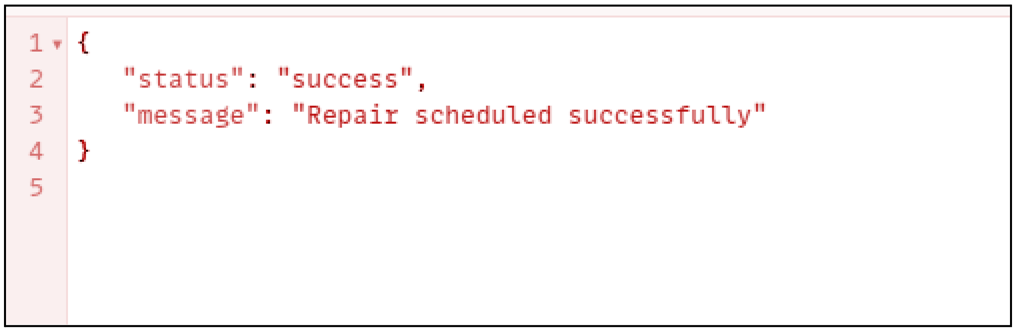
Description: Schedule a repair.

Request Payload:



```json
{
    "repairId": "1234",
    "userId": "5678",
    "time": "2024-09-20T10:00:00"
}
```

Response:

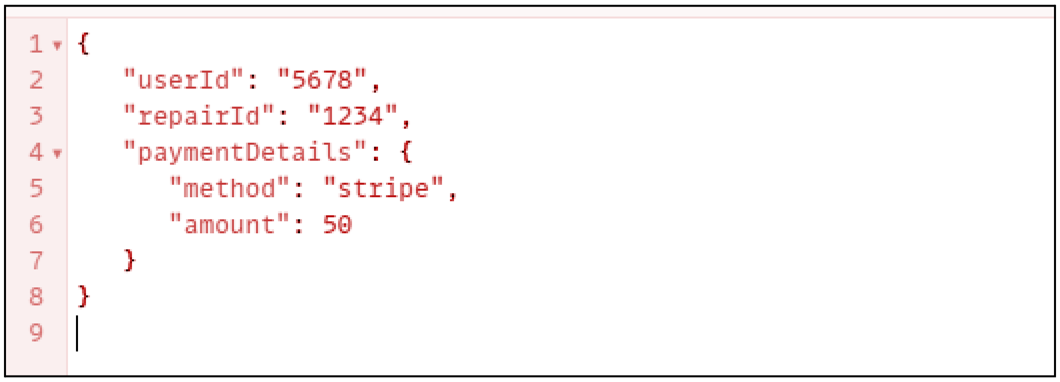


```json
{
   "status": "success",
   "message": "Repair scheduled successfully"
}
```

POST /api/pay

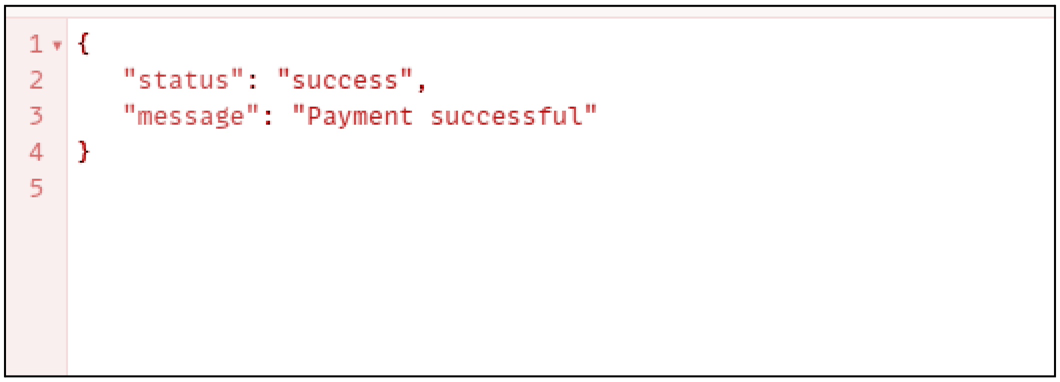
Description: Complete payment for a repair.

Request Payload:



```json
{
   "userId": "5678",
   "repairId": "1234",
   "paymentDetails": {
      "method": "stripe",
      "amount": 50
   }
}
```

Response:



```json
{
   "status": "success",
   "message": "Payment successful"
}
```

Build Steps/Scripts

Frontend:

1. Build React app using npm run build (if using React).
2. Minify and bundle assets.

Backend:

1. Use npm to install dependencies: npm install.
2. Build Node.js server using npm run build.

Install Steps/Scripts

Step 1: Installing Node.js **node -v** and **NVM npm -v**

Step 2: initiate a React application using the **npx-create-react-app** command

Step 3: Install dependencies using **npm install**

Step 4: Initialize the backend by using **npm init -y**

Step 5: Create a MongoDB account and install mongoose using **npm i mongoose**

Step 6: Start the Project using local tunnel by using the command **lt --port 3000**

Git Information

Link: https://github.com/hemanthchowdary1121/comp5130/tree/week3

References

MongoDB documentation. (n.d.). MongoDB Documentation. https://docs.mongodb.com/

Express - Node.js web application framework. (n.d.). https://expressjs.com/

Stripe API reference. (n.d.). https://stripe.com/docs/api

Index | Node.js v22.9.0 Documentation. (n.d.). https://nodejs.org/en/docs/