Silesian
University
of Technology

BACHELOR PROJECT

”Biomedical Signals Monitoring System”

Gloria ZHOU
304363

Study programme: Biomedical Engineering
Specialization: Information Systems in Medicine

SUPERVISOR
Dr inz. Marcin Rudzki
Department of Medical Informatics and Artificial Intelligence
Faculty of Biomedical Engineering

ZABRZE 2025

Project title:

Biomedical Signals Monitoring System

Abstract:

Remote biomedical signal monitoring systems have become invaluable as they allow for better monitoring or managing of acute or chronic health conditions. Current solutions are often single-user focused and do not allow continuous sharing of data with caregivers or health professionals. To address these limitations, this project presents the development of a web application that facilitates remote biomedical signal monitoring. It uses Fitbit wearable device data via the Fitbit Web API to enable caregivers to monitor the vital signs of their patients or loved ones in near real-time. A full stack web application was developed which utilizes React.js for the front-end and Node.js for the back-end. Key features include near real time data visualization, alert generation for abnormal readings and historical data access.

Keywords:

monitoring, biomedical signal, web application, API, health metrics, wearable devices

Tytuł pracy:

Systemy zdalnego monitorowania sygnałów

Streszczenie:

Systemy zdalnego monitorowania sygnałów biomedycznych stały się nieocenione, umożliwiając lepsze monitorowanie ostrych lub przewlekłych schorzeń lub zarządzanie nimi. Obecne rozwiązania często skupiają się na jednym użytkowniku i nie pozwalają na ciągłe udostępnianie danych opiekunom czy pracownikom służby zdrowia. Aby przezwyciężyć te ograniczenia, w ramach projektu opracowano aplikację internetową ułatwiającą zdalne monitorowanie sygnałów biomedycznych. Wykorzystuje dane urządzenia do noszenia Fitbit za pośrednictwem Fitbit Web API, aby umożliwić opiekunom monitorowanie parametrów życiowych swoich bliskich w czasie zbliżonym do rzeczywistego. Opracowano aplikację internetową typu full-stack, która wykorzystuje React.js jako front-end i Node.js jako back-end. Najważniejsze funkcje obejmują wizualizację danych w czasie zbliżonym do rzeczywistego, ostrzeganie o nieprawidłowych odczytach i dostęp do danych historycznych.

Słowa kluczowe:

monitorowanie, sygnał biomedyczny, aplikacja internetowa, API, wskaźniki zdrowia, urządzenia do noszenia

I’d like to dedicate this work to my parents, Faustina and Martin. Your unwavering support has been my greatest strength, reaching me even across borders. I love you endlessly. To my sisters Rachael and Ruth, you are the pillars of my strength. Your constant presence and support have meant the world to me. I adore you both.

Contents

List of Figures

List of Tables

Listings

1. Introduction

1.1. Background

Biomedical signals are electrical, mechanical, or chemical signals that are generated by the body and are used to diagnose diseases and monitor the health status of a patient [5]. Biomedical signal monitoring involves the continuous or periodic observation and analysis of these signals thus allowing for early diagnosis and intervention in case of any irregularities. The use of wearable health devices has increasingly helped people to monitor their health more effectively. Embedded with various sensors, wearable health devices record and track vital signs.

These technologies are especially useful for the elderly and chronically ill patients to manage and prevent serious health risks by continuously monitoring their vital signs outside of clinical settings. However, most of these systems are designed for individual use; they do not have features that allow family members or caregivers to monitor their loved ones' health. In other words, they are mainly focused on single-user functionality and not family-oriented.

Therefore, the goal of this project is to develop the biomedical signal monitoring system, a web-based application designed for caregivers and family members to monitor their loved ones' health throughout the day from anywhere across the globe. The application will integrate with wearable devices to collect, process, analyze and display important health metrics, sending alerts if abnormal readings are detected.

1.2. Goals/Objectives

The main goal of this project is to develop a web based Biomedical Signal Monitoring System that addresses the limitations identified in chapter 2. The system should provide caregivers and family members with near real-time access to biomedical signal data of their loved ones, utilizing already available wearable devices for continuous monitoring.

The project focuses on achieving the following objectives:

- Enable continuous collection, processing and displaying of patient health metrics to caregivers.
- Implement an alert system so that users receive notifications if abnormal readings are detected.
- Enable users to monitor multiple patients through the application.
- Allow for the integration of Fitbit smartwatch and possibility to scale to other device types in the future.
- Design a user friendly dashboard to enable users to visualize and analyze the health patterns also offering educational insights.
- Incorporate data encryption and user authentication mechanisms to safeguard patient health data and caregiver access.

1.3. Structure of the report

This report is structured into 7 chapters. After this chapter comes chapter 2 which gives an overview of the existing solutions. Chapter 3 describes how the project objectives are to be achieved. For reference, Chapter 4 contains the internal documentation for the implemented application. Chapter 5 presents the user manual to guide users on how to use and navigate the application. In chapter 6, the results of exemplary testing scenarios of the application are presented. Finally, chapter 7 contains the concluding remarks that include a summary of achieved results and future development plans.

2. Existing Solutions

Several solutions for health monitoring exist today. One category is the health monitoring applications that come coupled with wearable devices, such as the Health app for the Apple Watch, the Fitbit app for the Fitbit smartwatch, the Garmin connect app for the Garmin smartwatch and many more. These measure vital signs such as heart rate, blood pressure, respiratory rate, blood oxygen saturation and body temperature [6]. Such solutions provide continuous and accurate monitoring of health metrics.

Another category is applications that facilitate monitoring of vitals without the need for wearable devices. Examples include Biospectal OptiBP [1] presented in fig 2.1 and Veyetals [2] presented in fig 2.2. These solutions utilize the mobile phone camera to measure one's health metrics thus eliminating the use of wearable devices.

Although the above-mentioned solutions offer valuable health insights, a common limitation is that they are single user centric. They do not facilitate access to data by caregivers or family members who wish to monitor their loved ones.

One notable solution which partially addresses this limitation is My Connected Caregiver [3] presented in fig 2.3. It provides tools to monitor the health and safety of loved ones in one collaborative application. It is mainly targeted towards caregivers of the elderly. The application is connected to safety devices which have GPS tracking and fall detector. However, a significant limitation is that it relies on data logged by users rather than embedded sensors for automatic monitoring. As a result, the patients or the elderly being monitored have to continuously log their data for it to be accessible to the caregivers.

Table 2.1 summarizes the pros and cons of each solution.

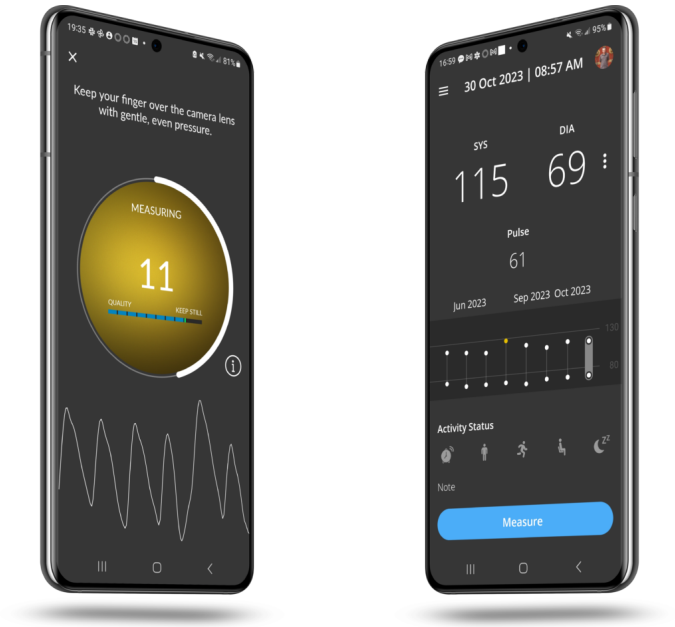

Fig. 2.1. OptiBP [1]

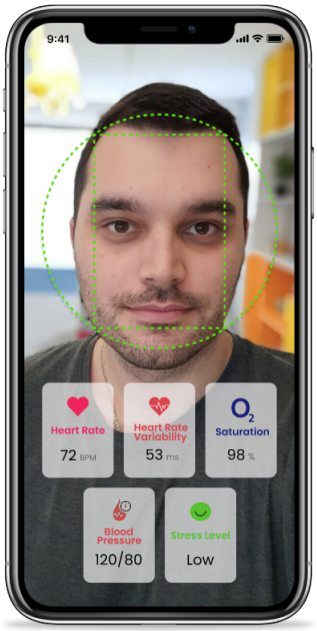

Fig. 2.2. Veyetals [2]

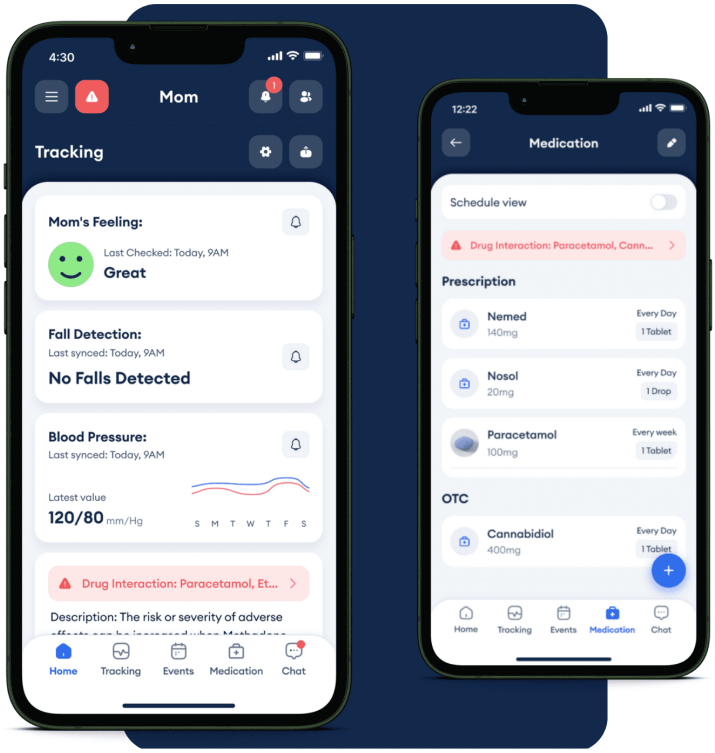

Fig. 2.3. Connected Caregiver [3]

Solution	Pros	Cons
Wearable based apps e.g Fitbit, Health, Garmin connect	• Continuous monitoring • Fitness and activity tracking • Automatic tracking	• High cost of wearbale devices • Limited multi-user access to data
Apps without wearables e.g Otibp, Veyetals	• No need for additional hardware • Less expensive	• No continuous monitoring • Requires active user involvement for measuring • Limited multi-user access to data.
Connected caregiver	• Multi-user access to health and safety data • Safety features like fall detection and GPS tracking	• Relies on logged data • No continuous monitoring • Limited device accessibility

Tab. 2.1. Pros and Cons of existing solutions

3. Problem Analysis

3.1. Sensor Integration

A critical part of the design and development of the system is the integration of the sensor. For the purpose of this project, the Fitbit smartwatch was used as the wearable sensor. Fig 3.1 shows an example of a Fitbit smartwatch. The health metrics measured include breathing rate, heart rate variability (HRV), skin temperature, oxygen saturation (SpO2), and heart rate. These vary depending on the version of the Fitbit device.

The Fitbit device automatically captures the health metrics whenever the user is wearing it. It is connected via Bluetooth to the client device, which is a mobile phone on which the Fitbit application is installed. Data from the Fitbit wearable device is first sent to the Fitbit application and is then forwarded to the Fitbit server [7].

The system will integrate the Fitbit data by incorporating the Fitbit server or Fitbit

Fig. 3.1. Fitbit Versa 3 [4]

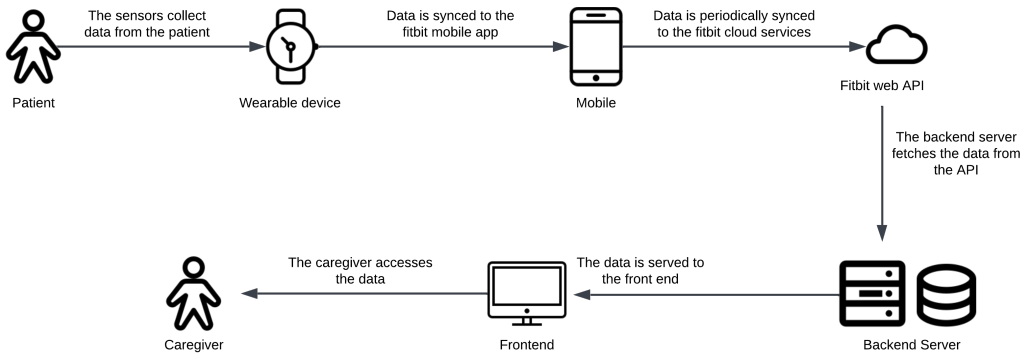

Fig. 3.2. Flow

Web API. Fitbit provides a set of public web APIs that developers may use to retrieve Fitbit user data collected by the Fitbit trackers and smartwatches [8]. Fig 3.2 shows the process involved from when patient data is measured by the sensor to the point it will be accessed by the caregiver.

3.1.1. Considerations

To successfully integrate the Fitbit Web API, there is need to study the Fitbit developer guide and several factors have to be considered which include data accessibility, API rate limits and permissions among others.

Data Accessibility

The Fitbit device and Fitbit mobile application can automatically sync every 15 minutes when the two are within Bluetooth range, the Fitbit application is running on the mobile device, and the mobile device has an active data connection [8]. Because of the synchronization delay, the system will display on the dashboard **near** real-time data, which will be updated in clusters.

Fitbit Web API Rate Limits

An application's rate limit is 150 API requests per hour for each user who has consented to share their data; and it resets, approximately, at the top of each hour. The usage count is incremented for all API requests that use access tokens. The application will receive an HTTP 429 response from the Fitbit API when a request is not fulfilled due to reaching the

rate limit [8]. To avoid exceeding the rate limits, historical data will be stored in the back-end database. When users request their patient's historical data, the server will fetch the data from the local database instead of making external requests to the Fitbit API. More details on the database are presented in section 3.3.

Permissions

The Fitbit Web APIs have the ability to expose a finer granularity of data collected throughout the day. This collection of data is called Intraday data. Intraday data is made available for the Activities, Breathing Rate, Heart Rate, HRV and SpO2 data sets by extending the daily detail-level response. 3rd-party developers who require intraday access to a Fitbit user's data should submit a request and access is granted on a case by case basis [8]. With Intraday access we can get for example heart rate at 15 min, 5 min, 1 min or even 1 sec intervals rather than getting only average resting heart rate. This is important since we want caregivers to effectively monitor their patients or loved one. To enable effective monitoring, the application for intraday data access was submitted to the relevant committees and approval was obtained.

3.2. Technologies used

MongoDB Atlas will be used as the database. Since MongoDB stores data as JSON documents, its a perfect choice for storing health metrics data retrieved from the Fitbit web API which is in JSON format. Unlike relational databases, MongoDB does not require a pre-defined schema for a collection thus data structures can easily be altered.

For front-end development, React.js, a popular JavaScript library will be used. React allows for smooth content updates without page reloads. Its focus on reusable components makes it ideal for real-time applications.

For back-end development, Node.js ,a JavaScript runtime environment will be used. Since Node.js utilizes JavaScript code, its an excellent choice because the same language will be used for both front-end and the back-end development.

The chosen Integrated Development Environment (IDE) is Visual Studio Code. It is the most popular among web developers. It has built-in support for JavaScript and Node.js which are both used in the application.

Other useful tools include Postman which is very beneficial for testing both the application and Fitbit's API endpoints.

3.3. Database Design

As highlighted in section 3.2 the chosen database for the system is MongoDB. There will be 10 collections in the database storing various types of data including user and patient information, alerts generated when abnormal readings are detected and also historical health metrics data. Fig 3.3 showcases the database structure to be employed and table 3.1 gives a description of each collection.

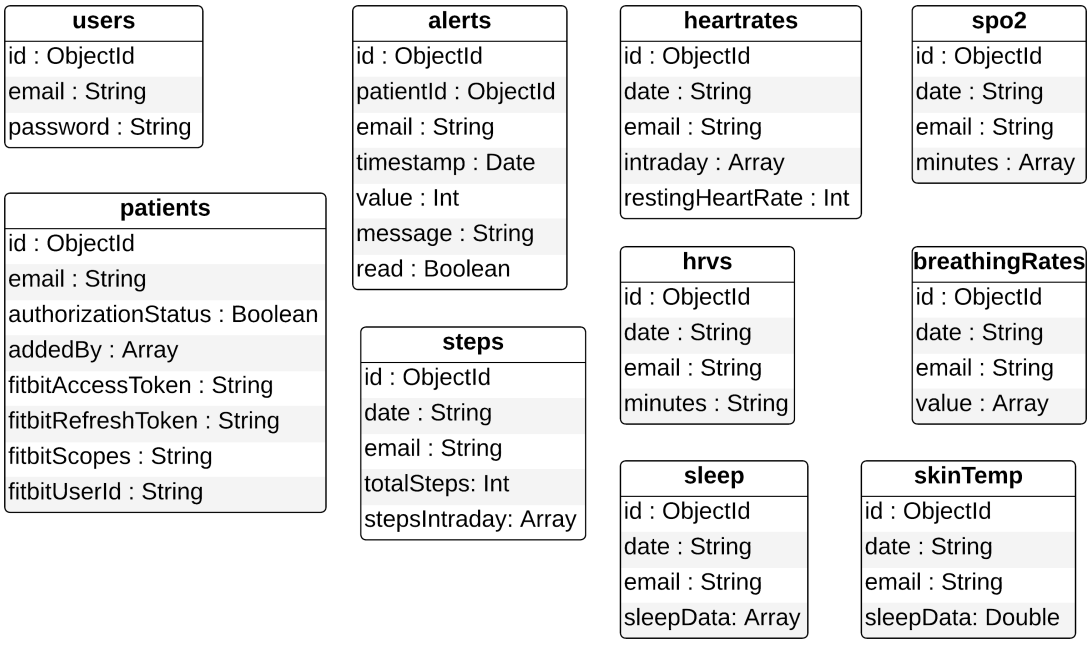

Fig. 3.3. Database Structure

3.4. Security

As the system interacts with Fitbit users' health data, security and privacy issues are a major concern. To ensure data security, the following measures will be implemented.

- Sensitive data stored in MongoDB database is to be encrypted. This includes Fitbit user IDs, access tokens and refresh tokens.
- The web application will be secured using HTTPS (Hypertext Transfer Protocol Secure) and SSL (Secure Sockets Layer).

Collection Name	Description
users	Stores the users' authentication information.
patients	Stores information about the patients being monitored.
alerts	Stores records of all alerts generated by the system.
heartrates	Stores heartrate data for patients.
spo2	Stores oxygen saturation data for patients.
hrvs	Stores heart rate variability data for patients.
steps	Stores steps data for patients.
sleep	Stores sleep data for patients.
breathingRates	Stores breathing rate data for patients.
skinTemp	Stores skin temperature data for patients.

Tab. 3.1. List of collections

- Fitbit's OAuth 2.0 authorization code flow will be integrated to ensure patients explicitly grant the system permission to access their data.
- User authentication through login will be implemented to protect user accounts from unauthorized access.

3.5. Alert System

The alert system is a vital component of the web application since it is important for caregivers to know when their patients are in danger. It involves detecting abnormal readings based on specified threshold values, then sending alerts to users via the system and additionally via email to ensure they do not miss critical alert.

3.6. Data Visualization

Charts will be used to visualize data on the dashboard as they make it easier to understand, interpret and analyze trends.

4. Internal Documentation

4.1. Project Structure

The project consists of the **Client side** (front-end) and **Server side** (back-end). As such, the project files are divided into two main folders: the `backend` folder and the `frontend` folder. Fig 4.1 shows the back-end directories and fig 4.2 shows the front-end directories.

Project Structure Overview

Back-end Structure

- **certs** : Contains SSL certificate and key for HTTPS communication.
- **controllers** : Contains functions that execute the business logic of the application.
- **models** : Defines the MongoDB schema for collections.
- **routes** : Specify the logical paths in the application that determine the server's responses to client requests.
- **services** : Contains functions that facilitate reusable interactions with the database and external APIs.
- **.env** : Holds sensitive information like passwords, client ID and other information that must not be directly written in the code.
- **server.js** : The entry point file of the server.

Front-end Structure

- **assets** : Contains media including images or videos used in the application.

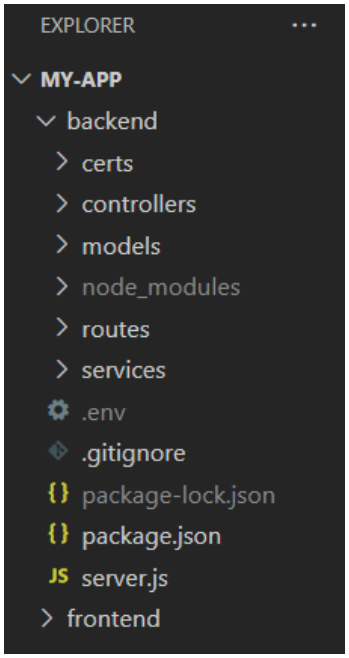

Fig. 4.1. Back-end Structure

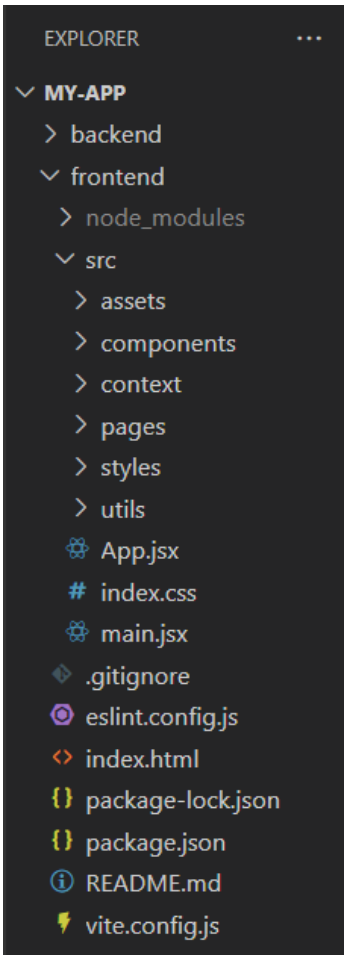

Fig. 4.2. Front-end Structure

- **components** : Contains user interface components such as the health metric cards, header, sidebar etc.
- **context** : Stores React context files which are used across multiple pages.
- **pages** : Contains the functions for respective pages in the application such as the Login Page function.
- **styles** : Contains css files used in the project.
- **utils** : Contains helper or utility functions used throughout the application.
- **App.jsx** : This is the root component of the application.

4.2. Prerequisites

Before running the project, ensure the following.

- Node.js should be installed.
- Install the dependencies highlighted in package.json file. To install the dependen-

cies, use the command `npm install`. If the node-modules directory is already present, this step can be skipped.

- Set up a MongoDB Atlas database and generate a connection string.
- Register the application on the Fitbit developer portal to get a Client ID and Client Secret. Set the redirect URL to *https://localhost:443/oauth2/callback*.
- Configure environment variables in a .env file as shown in listing 4.1.

```
MONGO_URI = [MongoDB connection string]
APP_PASSWORD = [Email used by application]
APP_EMAIL = [Password for the application email]
CLIENT_ID = [Fitbit developer application client ID]
CLIENT_SECRET = [Fitbit developer application client secret]
MONGO_ENC_KEY = [Encryption key for MongoDB]
MONGO_SIG_KEY = [Signing key for MongoDB]
```

Listing 4.1. Environment variables

4.3. Running the back-end

Navigate to the back-end folder in the terminal and start the back-end with the command `node server.js`. Figure 4.3 shows the output on the terminal after successfully executing the command.

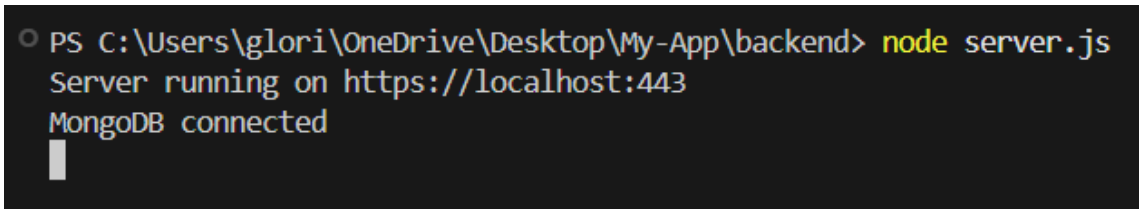

Fig. 4.3. Starting the server

4.4. Running the front-end

Navigate to the front-end folder in the terminal and start the front-end with the command `npm run dev`. Ensure the back-end is already running. Figure 4.4 shows the output on the terminal after successfully executing the command.

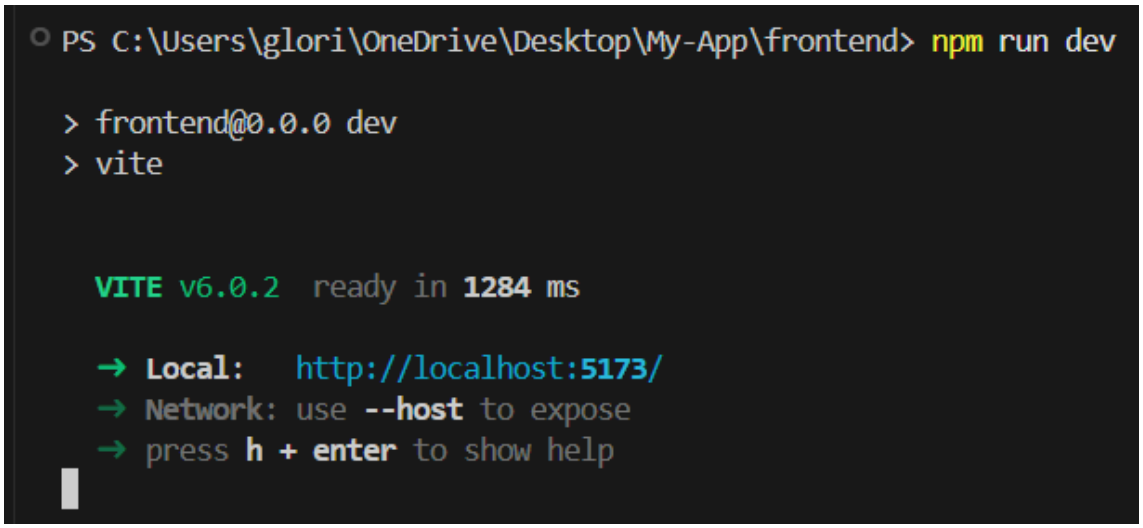

Fig. 4.4. Running the front-end

4.5. Back-end Implementation

4.5.1. Server.js

It is the entry point of the back-end which initializes the Express application, sets up HTTPS communication, defines the routes, configures middle-ware and establishes a connection to MongoDB.

Note: Currently the HTTPS communication appears broken and gives warning messages in the browser because a self signed certificate was used. For deployment of the application, a trusted SSL certificate should be obtained.

Module imports

- `"express"` : Web framework for building the API.
- `"https"` : For creating the HTTPS server.
- `"fs"` : For reading the SSL certificate and key.
- `"mongoose"` : For connecting to MongoDB database.
- `"dotenv"` : For loading environment variable from the .env file.

4.5.2. Database Models

The back-end uses MongoDB Atlas to store and manage data. Mongoose library for MongoDB is used for enforcing the Schema. Each collection in the database has its corresponding mongoose model.

Example: User schema and model

Listing 4.2 shows the implementation of the User schema.

```
import mongoose from "mongoose";
const userSchema = new mongoose.Schema(
  {
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
  },
  { timestamps: true }
);
export default mongoose.model("User", userSchema);
```

Listing 4.2. User Schema in Mongoose

Fields

- email : User's email address.
- password : User's password.
- timestamps : Automatically adds createdAt and updatedAt fields to track record updates.

Data encryption

Sensitive data such as user passwords in the user schema and Fitbit tokens in the patient schema is encrypted using the `mongoose-encrypt` library before storing it in the database.

4.5.3. Routes

A RESTful API was created using `express.Router()` exposing multiple endpoints to handle client requests and responses efficiently. The API provides several GET and POST endpoints allowing the client application or front-end to retrieve data from the server and send data to it.

Route Organization

Routes are organized into separate files within the 'routes' directory. Each file corresponds to a specific resource or functionality.

- **Data routes (fitbitData)** - Fetches real-time Fitbit health metrics data. This is the only route that communicates directly with Fitbit Web API. When this endpoint

is called, the back-end in turn calls the Fitbit API. The Fitbit API documentation is available on the Fitbit developer website [8] and is not covered in this documentation.

- **History routes (/history)** - Retrieves historical health metrics data from the MongoDB database. Due to limit in free MongoDB Atlas storage, data is only stored for 3 days.
- **Authentication routes (/auth)** - Manages user authentication by handling login and registration requests.
- **Alert routes (/alerts)** - Handles retrieval and updating of alerts stored in the database.
- **Email routes (/email)** - Sends authorization emails to patients for granting Fitbit access.
- **OAuth2 routes (/oauth2)** - Handles Fitbit API authorization flow. This is the redirect page that patients are taken to after authorization.
- **Patients routes(/patients)** - Retrieves a list of patients registered in the application.

Each endpoint is defined using the `get()` method for fetching data and the `post()` method for sending data to the server.

Example: Authentication routes

Listing 4.3 demonstrates how POST endpoint for user login is implemented. The endpoint is designed to accept two parameters, email and password from the request body. It is then integrated into the front end to handle user authentication. In a similar manner, all the other endpoints are implemented. Table 4.1 presents all the endpoints exposed by the application.

```
1 import express from "express";
2 const router = express.Router();
3 router.post("/login", async (req, res) => {
4 const {email,password} = req.body;
5 //Authentication logic here
6 }
7 export default router;
```

Listing 4.3. Login endpoint implementation

API Endpoints

Method	Endpoint	Description
GET	`/fitbitData/devices`	Retrieves a list of Fitbit devices paired to the patient's account.
GET	`/fitbitData/sleep`	Returns a list of a user's sleep log entries for current day.
GET	`/fitbitData/spo2`	Returns the SpO2 intraday data for the current day.
GET	`/fitbitData/heartrate`	Retrieves the heart rate intraday time series data for the current day at 5min detail level.
GET	`/fitbitData/hrv`	Returns the Heart Rate Variability (HRV) intraday data for the current day.
GET	`/fitbitData/profile`	Retrieves the patient's profile data.
GET	`/fitbitData/br`	Returns patient's intraday breathing rate data for the current day.
GET	`/fitbitData/steps`	Retrieves the activity intraday time series data for the current day.
GET	`/fitbitData/skinTemp`	Returns the skin temperature data for the current day.
GET	`/history/heartrate`	Retrieves history heart rate data for the last 3 days.
GET	`/history/hrv`	Retrieves history heart rate variability data for the last 3 days.
GET	`/history/spo2`	Retrieves history oxygen saturation data for the last 3 days.
GET	`/history/br`	Retrieves history breathing rate data for the last 3 days.
GET	`/history/steps`	Retrieves history steps data for the last 3 days.
GET	`/history/skinTemp`	Retrieves history skin temperature data for the last 3 days.
GET	`/history/sleep`	Retrieves history sleep data for the last 3 days.
POST	`/auth/login`	Handles user login by verifying credentials.
POST	`/auth/register`	Handles user registration by saving the user details.
GET	`/alerts`	Retrieves alerts from the database.
PATCH	`alerts/:id`	Updates the read status of an alert.
POST	`/email/send`	Sends authorization email to patient.
GET	`/oauth2/callback`	The redirect route which handles Fitbit oauth 2 flow by exchanging authorization code for tokens.
GET	`/patients`	Retrieves a list of patients added by the specified user.

Tab. 4.1. Back-end application endpoints

Error Handling

The common http status codes you will encounter when running the application are presented in table 4.2.

HTTP Status Code	Description
200 OK	Indicates a successful request.
201 Created	Indicates successful creation of a new resource.
401 Unauthorized	Indicates that you are not authorized to access the resource usually due to invalid or expired tokens.
403 Forbidden	Indicates that you do not have permission to access the resource.
404 Not found	Indicates that the resource you are trying to access doesn't exist.
429 Too many requests	Indicates that you have hit the rate limit and should wait for it to reset.
500 Internal server error	Indicates that something is wrong with the server.

Tab. 4.2. HTTP status codes

4.5.4. Alert Generation

The back-end is responsible for alert generation based on predefined health thresholds. This logic is integrated into the data fetching routes. When data is retrieved from the Fitbit API, it is processed to extract the relevant dataset. The dataset is then filtered to identify abnormal readings—values that exceed the upper threshold or fall below the lower threshold. For each abnormal reading, an alert entry is created and stored in the Alert model within the database. These alerts can then be accessed by caregivers and users through the `/alerts` API endpoint.

4.6. Front-end Implementation

The front end application was built with React.js and uses modern libraries to enhance performance. It communicates with the back-end API to fetch, display and manage patient health data, alerts and historical trends.

4.6.1. Key technologies and libraries used

- **React Hooks** - For managing state and side effects.
- **React Context API** - For global state management.
- **React Query** - For fetching and caching API data.
- **React Router** - For handling navigation and private routes.
- **Axios** - For making API requests.
- **Chart.js** - For data visualization.

4.6.2. Use of React Hooks

Several react hooks are used in the application.

- useNavigate - For navigating between pages.
- useState - Manages the state at component level.
- useEffect - Handles side effects in components.
- useContext - For consuming a shared context.

4.6.3. Global State Management with React Context

The Context API is used to manage global state. Global state refers to data that is accessible across multiple components. In the application, a global state is needed to manage the selected patient. This is because all the dashboard components and related pages need to access the selected patient's information to display the data specific to that patient.

The "PatientContext" is created using `createContext()` and provides a way to pass data through the component tree without having to pass props manually at every level. The "PatientProvider" wraps the component tree making the "selected patient" state available to all components that need it. Any component that needs to access the selected patient can use the `useContext` hook to consume the context.

4.6.4. State Management with React Query

The application uses React Query to manage server state. This library simplifies data fetching and catching ensuring efficient and seamless updates to the user interface. This reduces unnecessary API calls and improves user experience. Listing 4.4 shows how the useQuery hook was used for fetching patient data.

```
    export const usePatientData = (email) => {
  return useQuery({
    queryKey: ["patientData", email],
    queryFn: () => fetchPatientData(email),
    enabled: !!email
    refetchInterval: 5 * 60 * 1000,
    staleTime: 5 * 60 * 1000,
  });
};
```

Listing 4.4. useQuery for fetching data

useQuery parameters

- **queryKey** : Identifies the query.
- **queryFn** : The function that the query will use to fetch the data.
- **enabled** : Ensures the query runs only when the email is defined.
- **refetchInterval** : Automatically re-fetches data every 5 minutes.
- **staleTime** : Prevents loading if data is still fresh.

4.6.5. Routing and Navigation

React router is used to manage navigation between pages in the application. The app consists of public and private routes. Private routes can only be accessed by authenticated users. Routes are defined using the "Routes" and "Route" components from React. When a user attempts to access a private route, "PrivateRoute" checks if the user is authenticated.

Public routes

- **/** - Login page
- **/register** - Registration page

Private routes

- **/welcome** - Welcome page
- **/dashboard** - Main dashboard with health metric cards
- **/managePatients** - Patients management page
- **/dashboard/alerts** - Alerts page for abnormal readings
- **/dashboard/insights** - Insights page
- **/dashbaord/patientProfile** - Patient profile page

4.6.6. Integration with Back-end API

The front-end interacts with the back-end through RESTful API calls to the endpoints indicated in table 4.1. Axios library is used for making HTTP requests. Listing 4.5 demonstrates how the axios library is used to fetch data from the back-end, ensuring seamless integration between the front-end and the back-end.

```
//Fetch the list of patients added by the user
const patientsResponse = await axios.get(
`https://localhost:443/patients?userEmail=${email}`
);
```

Listing 4.5. Axios for making HTTP requests

4.6.7. Styling and User Interface Design

The user interface is styled using CSS (Cascading Style Sheets) modules. The dashboard uses a grid style for organizing health metric cards. React chart.js is used for displaying health data on charts.

5. User's manual

Utanostraz is a web application designed for caregivers to remotely monitor their patients or loved ones. This user manual is meant to guide you on how to use and navigate the application. In this guide, the user refers to the person using the application (caregiver/-family member). The patient refers to the person whose data is being monitored.

5.1. Accessing the application

In your chosen browser, navigate to http://localhost:5173

5.2. Login

The login page will appear (Fig 5.1). If registered, login by entering your email address and your password. If not registered, click on "Create account" and navigate to the registration page (Fig 5.2). Register by entering your email address and password of your choice. After registration you will be able to login.

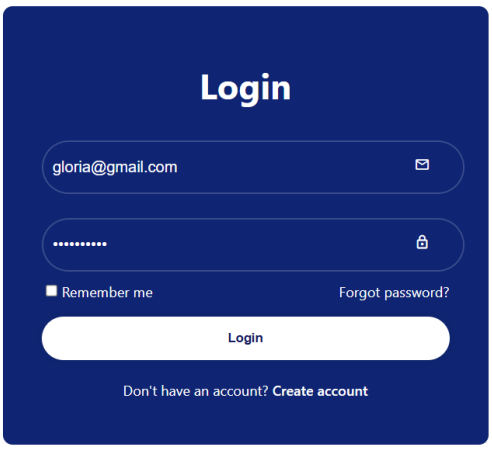

Fig. 5.1. Login page

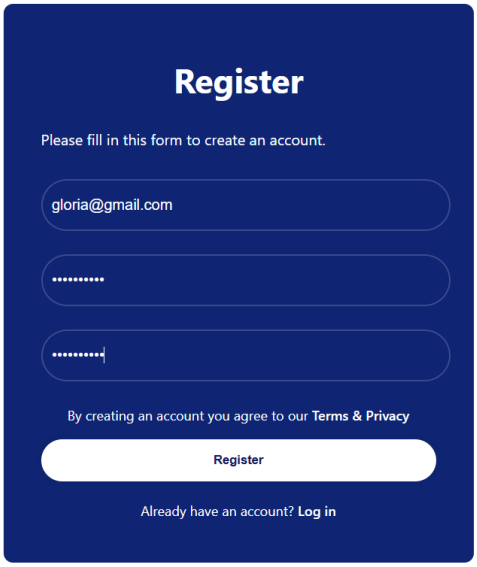

Fig. 5.2. Registration page

5.3. Welcome Page

After successful login, you will be taken to the welcome page (Fig 5.3). Click on the "Manage patients" button to start managing your patients.

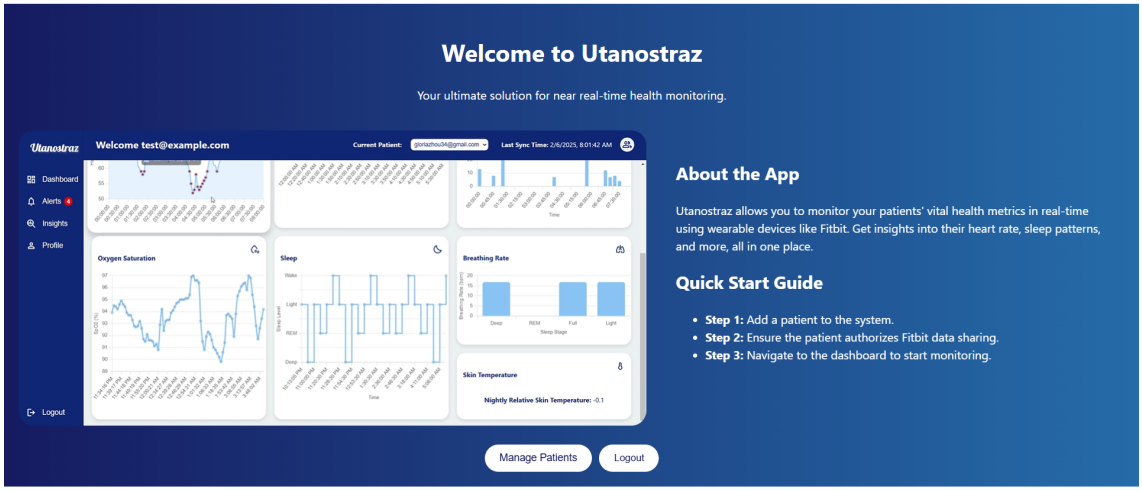

Fig. 5.3. Welcome page with "Manage Patients" button

5.4. Manage Patients

You will be taken to the patients management page (Fig 5.4). Here you will see a list of the patients you added and their authorization status. The authorization status tells whether or not the patient has given the application access to their data. To add a Patient, click on the "Add Patient" button.

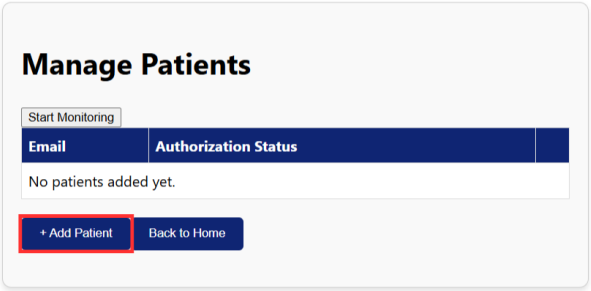

Fig. 5.4. Patients Management page with "Add Patient" button

Upon clicking the Add Patient button, you should see the Add Patient form where you have to add the email address of your patient then click the "submit" button. Make sure the patient has a Fitbit account. Once submitted, the patient will receive an email

requesting them to authorize sharing of their data. It is strongly advised that you that you communicate with your patient prior to sending this request. Fig 5.5 shows the patient management page after successfully adding a patient.

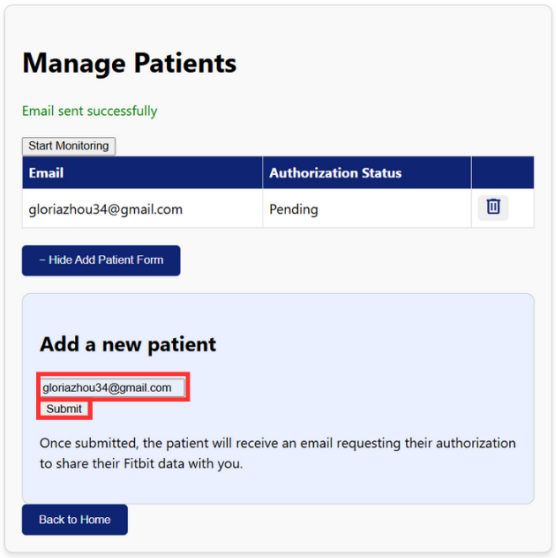

Fig. 5.5. After adding a patient

Patient side

When your patient receives the authorization email, they have to complete the authorization process.

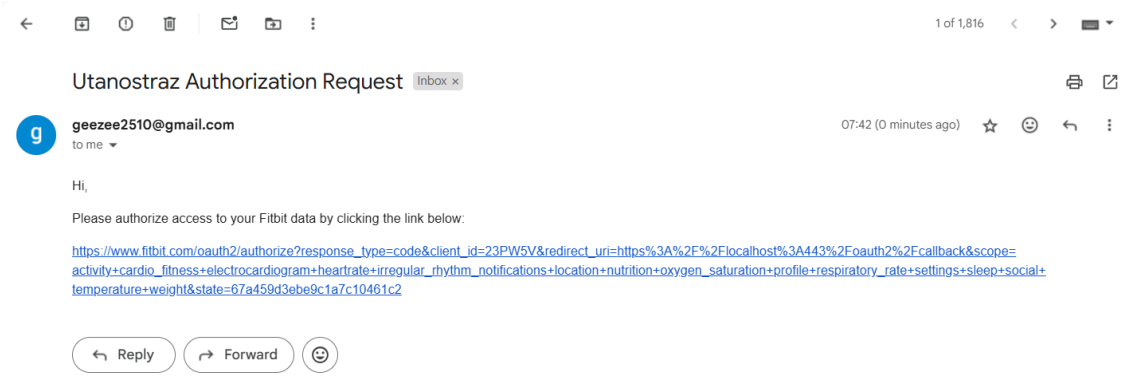

Fig. 5.6. Authorization request email

Fig 5.6 shows the contents of the email including the link. The patient should click the link provided in the email. Upon clicking, they are taken to Fitbit authorization page (Fig

5.7) where they can select the scopes they want to allow access to. After they click allow, they will be redirected to a page where they will see a message indicating successful authorization. Fig 5.8 shows the redirect page showing after successfully completing the authorization process.

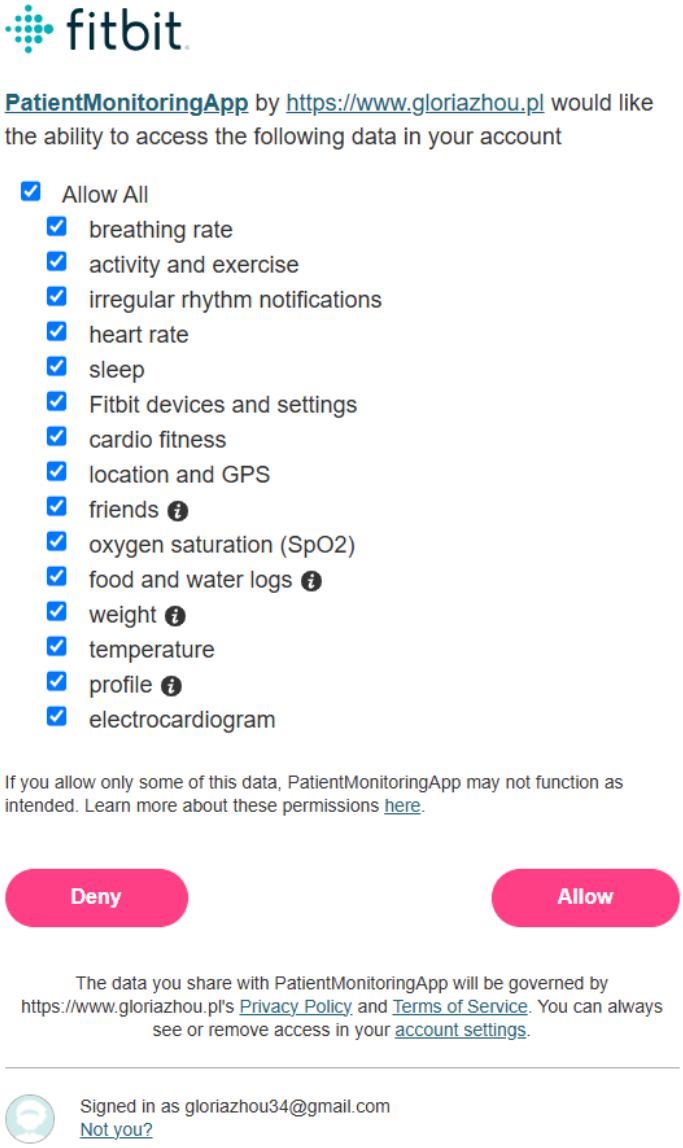

Fig. 5.7. Fitbit authorization page

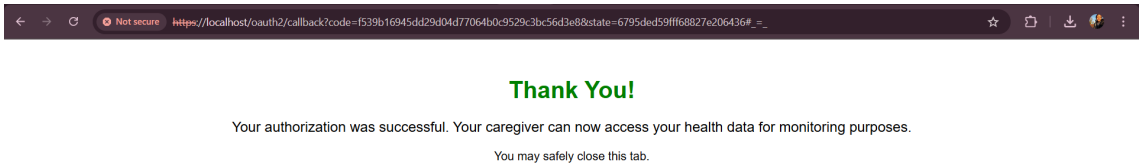

Fig. 5.8. Redirect Page indicating successful authorization

If the authorization process is successfully completed on the patient's side, you now have access to their data and can start monitoring.

5.5. Home Page

On the home page you will see the header, sidebar and dashboard.

5.5.1. Header

On the header (Fig 5.9), you will see a drop-down which displays the current patient. If you added more than one patient, you can switch between patients to display their data. Besides the drop-down, you will see the last sync time. This shows the most recent time that data from the Fitbit device is assured to have been recorded in the application. The button on the far right is the manage patients button. Click on it to be to be directed to the Patient Management page.

Fig. 5.9. Header

5.5.2. Sidebar

The sidebar helps you navigate between pages. Clicking on the Dashboard, Alerts, Insights and Profile buttons on the sidebar will direct you to the respective pages. Fig 5.10 shows the sidebar. At the bottom you will find the logout button. Click this button to end your session.

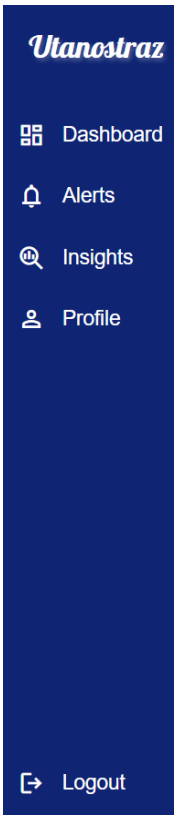

Fig. 5.10. Sidebar

5.5.3. Dashboard

On the dashboard (Fig 5.11 and 5.12), you can clearly visualize your patient's vital signs for the current day. It consists of 7 cards each corresponding to a health metric being monitored.

- **Heart rate card** : Here you can see the intraday heart rate trend for the current day from 00:00 to 23:59 at 5 min detail level. The data is continuously updated throughout the day.
- **Step card** : Here you can see the number of steps taken by your patient through out the day at 15 min intervals. The steps data is continuously updated throughout the day.
- **Heart rate variability card** : Here you can visualize the HRV data. Note that the HRV data is only collected during sleep.
- **Oxygen saturation card** : Here you will find Spo2 data. Note that spo2 data is only collected during sleep.
- **Sleep card** : Here you can see and analyze the sleep stages of your patient for their main sleep.

- **Breathing rate card** : Here you will see the average breathing rate for each sleep stage.
- **Skin Temperature card** : The temperature card displays the nightly relative skin temperature for the patient.

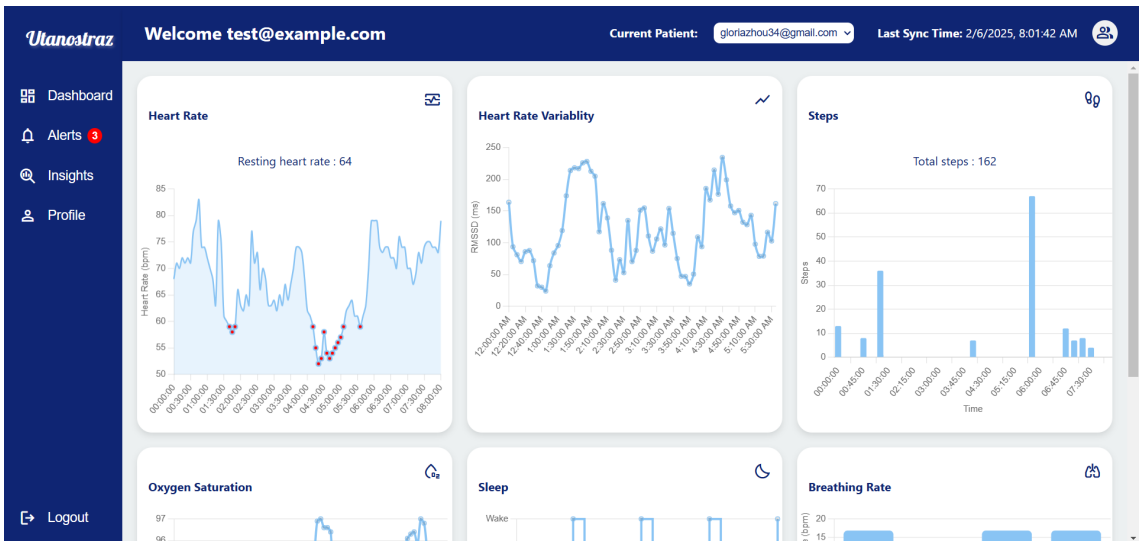

Fig. 5.11. Dashboard 1st part

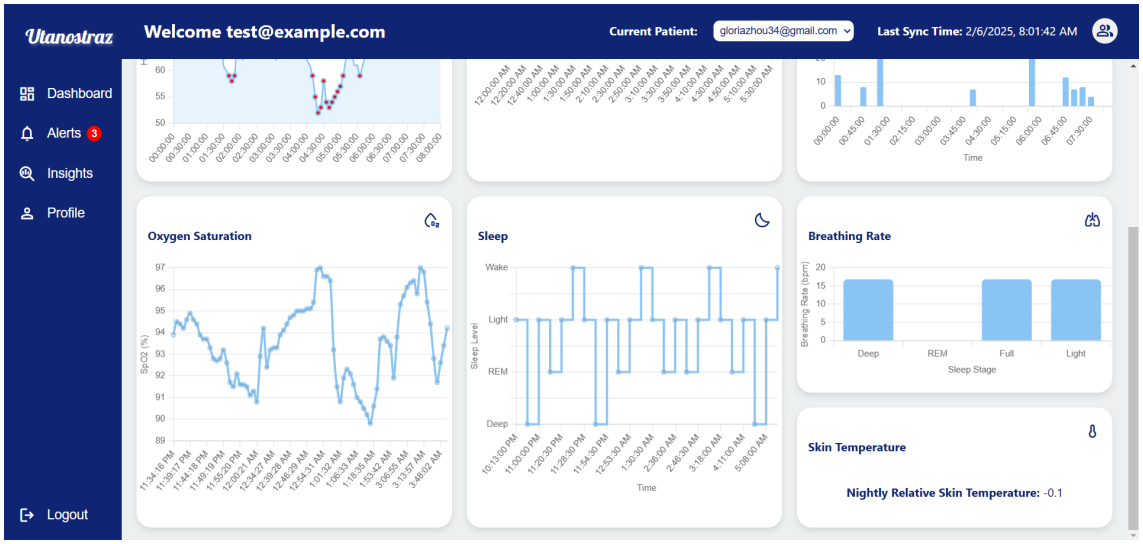

Fig. 5.12. Dashboard 2nd part

5.6. Insights and history

From the Insights page (Fig 5.13) you can access historical patient data for all the measured health metrics. Historical data is available for the last 3 days. At the top of the page, you will see the tabs to help you navigate between insights. You will also find more

educational information about each measured health metric and why it is important to monitor it.

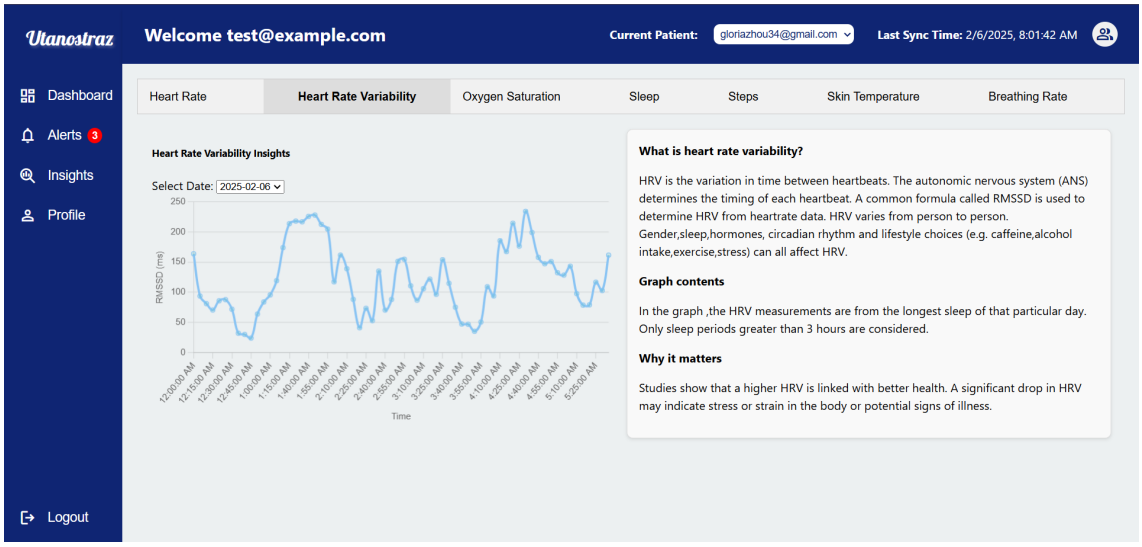

Fig. 5.13. Insights Page

5.7. Alerts

On the Alerts Page (Fig 5.14), alerts will be displayed whenever abnormal readings are detected in the data. Each alert will have its timestamp and details. After reading the alert, you can click on the "Mark as read" button. You may also click the "Dismiss" button to archive the alert and clear your inbox. In addition to this, you will also receive alerts via email to ensure you don't miss any critical alerts when you are not logged in.

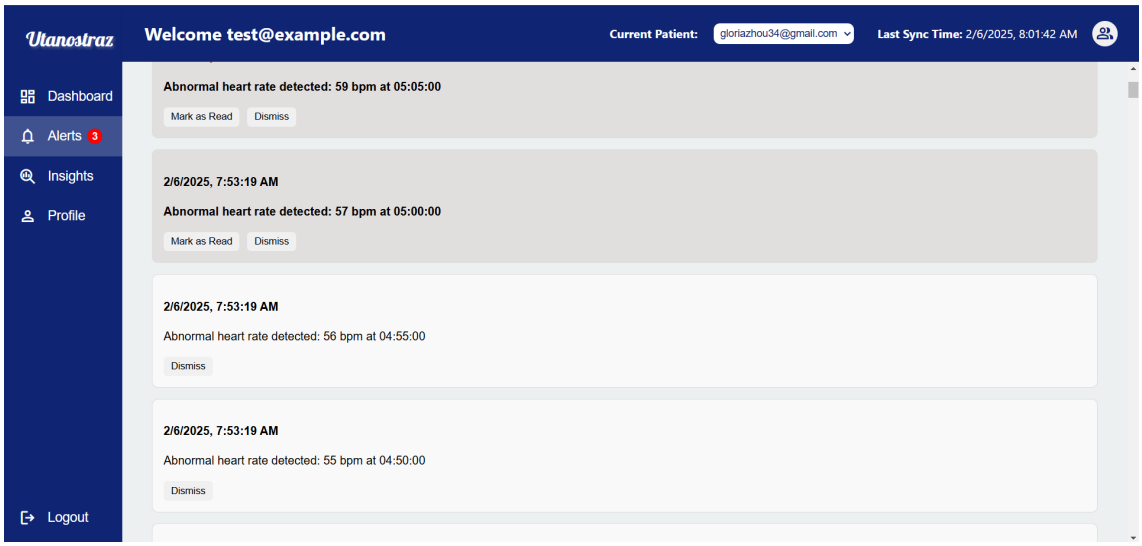

Fig. 5.14. Alerts Page

5.8. Profile

The profile page (Fig 5.15) lets you view the personal information of your currently selected patient. You will also find more information about your patient's wearable device.

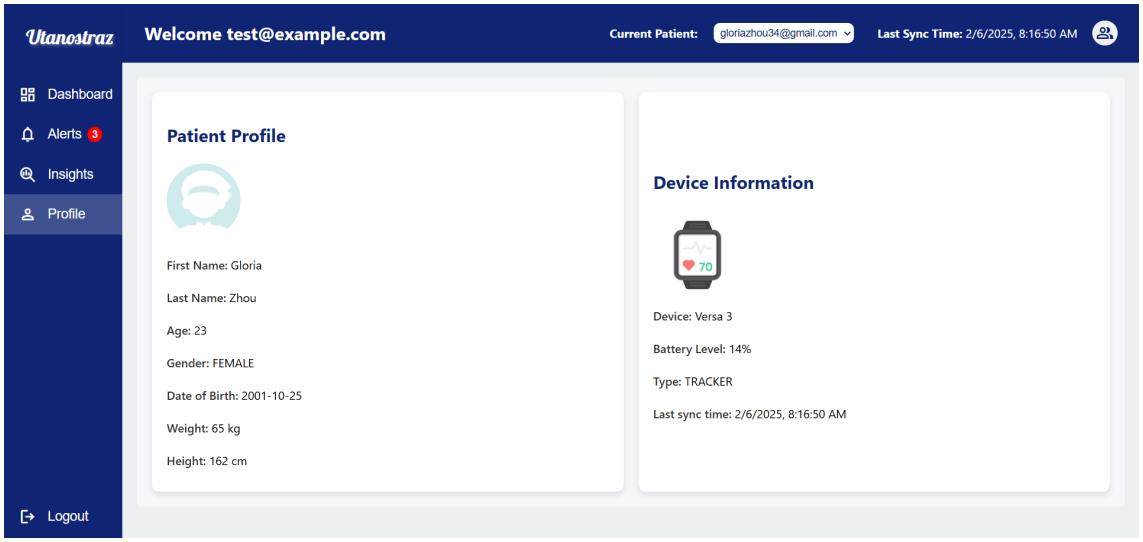

Fig. 5.15. Profile Page

6. Testing

Software testing is the process of running an application with the intent of finding software bugs (errors or other defects) [9]. There are two categories of testing which are Manual Testing and Automated Testing. Manual testing involves testers executing test cases manually, without the use of software tools, to verify the functionality of the software. On the other hand, automatic testing involves the use of software tools to execute tests automatically, without the need for human intervention [10]. For this project, manual testing was carried out. A test plan and test cases were designed. This chapter presents exemplary testing scenarios the results.

6.1. Functionality testing

Functional testing checks an application, website, or system to ensure that it is doing exactly what it is meant to [11]. Various application functionalities were tested to evaluate if they work correctly.

6.1.1. User authentication

The purpose of this test is to verify if user authentication in the application works correctly, allowing only verified users to access private routes and preventing unauthorized users from doing so. Table 6.1 shows the testing results. Fig 6.1 shows that the Invalid login test scenario obtained the expected result.

Test Scenario	Test Steps	Expected Results	Status
Valid Login	1. Open Login Page 2. Enter valid credentials 3. Click the Login button	Redirection to Welcome page/-Manage Patients Page	Pass
Invalid Login	1. Open Login Page 2. Enter invalid credentials 3. Click the Login button	Invalid login message should be displayed	Pass

Tab. 6.1. User authentication testing results

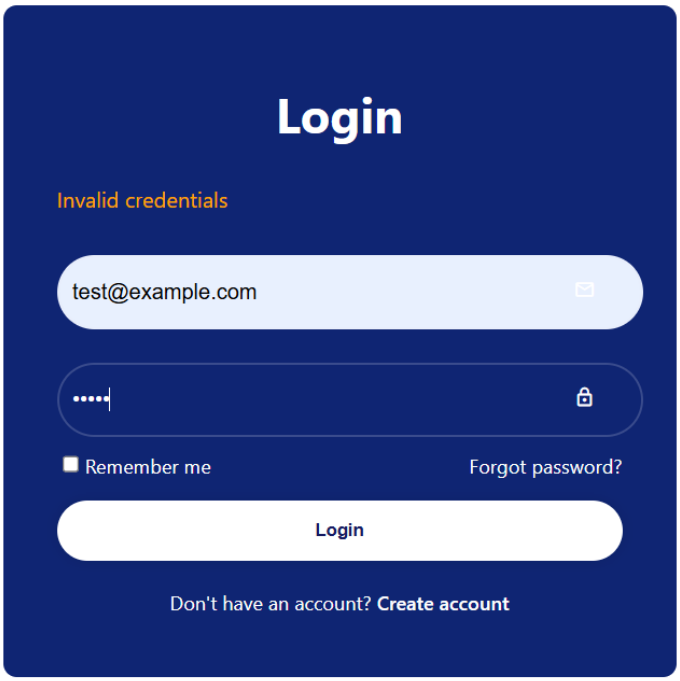

Fig. 6.1. Invalid login

6.1.2. Adding a patient

The purpose of this test is to evaluate whether users are able to seamlessly add patients to start monitoring. The results of the test are presented in table 6.2 and in fig 6.2 we can see that the process of adding the patient works correctly.

Test Scenario	Test Steps	Expected Results	Status
Sending fitbit authentication email	1. Open the Manage Patients Page 2. Click the Add Patient button 3. Fill in the form with the email of the patient 4. Click submit	The patient should receive an email with the authorization link	Pass

Tab. 6.2. Patient Management testing results

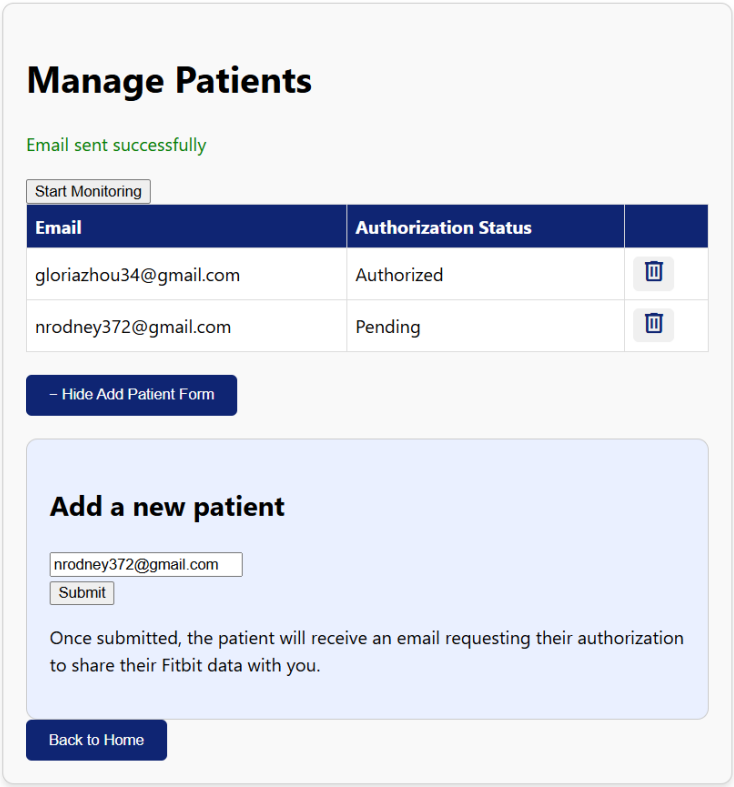

Fig. 6.2. Email successfully sent

6.1.3. Switching between multiple patients

This test investigates whether the application works correctly for users monitoring multiple patients. In this case 2 patients were used. One has got a Fitbit versa 3 smartwatch which has been is actively collecting data, the other does not have a device but has a Fitbit account. Table 6.3 presents the testing steps and results. As expected, data changes dynamically depending on the selected patient. Patient 1's dashboard (fig 6.3) displays data whist Patient 2's dashboard (fig 6.4) lacks data since they do not have a sensor.

Test Scenario	Test Steps	Expected Results	Status
Single user multiple patients	1. Open the dashboard 2. Observe patient 1's data 3. Locate the patient drop down and switch to a different patient 4. Observe patient 2's data	The displayed data should change based on the selected patient	Pass

Tab. 6.3. Testing Functionality for Switching Between Multiple Patients

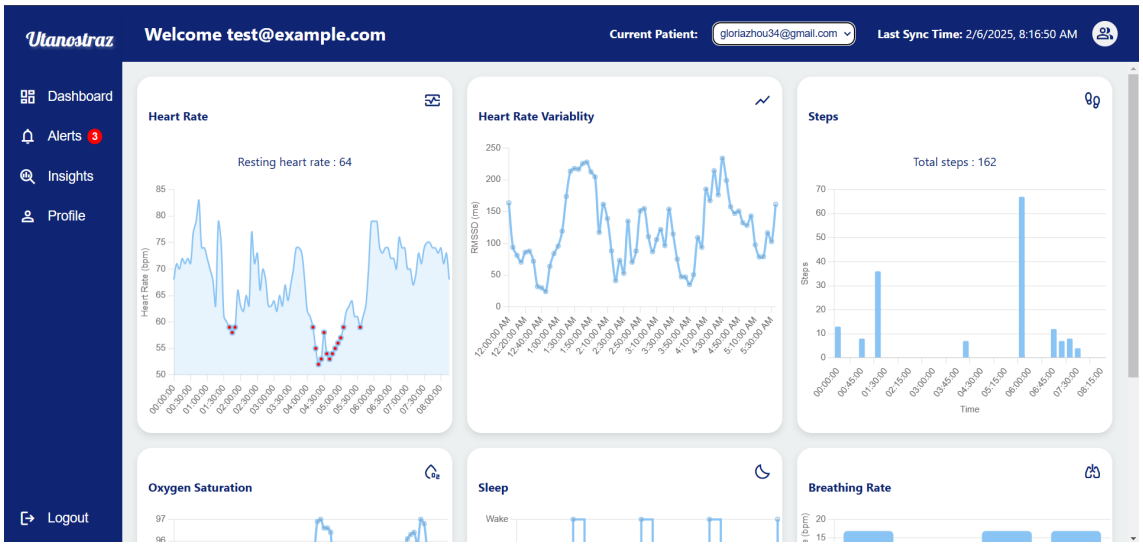

Fig. 6.3. Patient 1's dashboard

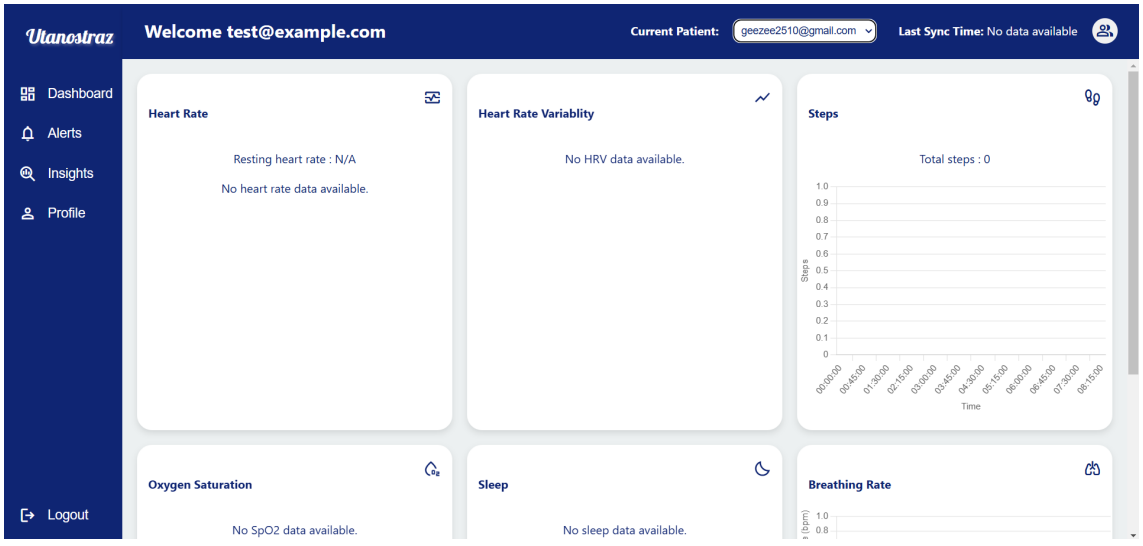

Fig. 6.4. Patient 2's dashboard

6.1.4. Automatic updates of data

In this test we want to find out if the patient's health metrics are updated automatically from time to time. The test steps and results are presented in table 6.4.

Test Scenario	Test Steps	Expected Results	Status
Continuous automatic data updates	1. Open the dashboard 2. Observe the last sync time from the header and the health metric cards (target heart rate card) 3. Wait for max 15 minutes	The last sync time and patient data should be automatically be updated at regular intervals	Pass

Tab. 6.4. Testing for automatic updates

The results prove that all the tested functionalities work as expected.

7. Conclusion

The aim of the project was to develop a web based application capable of collecting data from biomedical sensors, processing, displaying and sending alerts for abnormal readings. Through conducting thorough research in the field biomedical monitoring and employing modern software development best practices, the overall goal of this project was successfully realized. The application integrates the Fitbit device to monitor biomedical signals such as heart-rate, heart rate variability, oxygen saturation, breathing rate etc. The system's ability to allow caregivers/family members to continuously monitor their patients or loved ones remotely makes it stand out among other solutions. The application was tested and proved to be accessible, user friendly, efficient and secure.

However, there were some limitations that were noted. The application heavily relies of Fitbit Web API, so if their server is down temporarily, it will also be affected. Additionally, the current authentication for Fitbit scopes gives data access to the whole application rather than to specific users. It is possible to control this but rather complex, therefore due to time constraints, the issue was not addressed.

For further development, the following could be incorporated to make the application even better:

- Use of machine learning algorithms for giving personalized feedback and predictive insights.
- Use of web sockets to enable real time monitoring.
- Scaling the application to include other wearable device vendors especially those measuring other biomedical signals like blood pressure etc that are not not measured by the Fitbit.

Bibliography

[1] "Biospectal optibp: The ai-driven, medically certified blood pressure monitor app," https://biospectal.com/, accessed: 2025-02-03.

[2] "Veyetals: Health and wellness checks," https://veyetals.com/, accessed: 2025-02-03.

[3] "Connected caregiver application," https://myconnectedcaregiver.com/, accessed: 2025-01-13.

[4] F. Sklep, "Fitbit versa 3 image," https://fitbitsklep.pl/produkt/fitbit-versa-3-czarny, accessed: 2025-02-07.

[5] J. Spinrad, "Biomedical signal and its processing techniques," *Biomedical Engineering and Medical Devices*, vol. 8, p. 256, 2023. [Online]. Available: https://www.longdom.org/open-access/biomedical-signal-and-its-processing-techniques-99446.html

[6] D. Dias and J. Paulo Silva Cunha, "Wearable health devices—vital sign monitoring, systems and technologies," *Sensors*, vol. 18, no. 8, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/8/2414

[7] A. Al-Sabaawi, K. Al-Dulaimi, Y. Zhao, and L. Simpson, "Investigating data storage security and retrieval for fitbit wearable devices," *Health and Technology*, 2024. [Online]. Available: https://doi.org/10.1007/s12553-024-00885-0

[8] Fitbit, Inc., “Fitbit web api reference,” https://dev.fitbit.com/build/reference/web-api/, accessed: 2025-01-22.

[9] N. Anwar and S. Kar, “Review paper on various software testing techniques & strategies,” *Global Journal of Computer Science and Technology*, pp. 43–49, 05 2019. [Online]. Available: https://computerresearch.org/index.php/computer/article/view/1873

[10] K. Thant and H. H. K. Tin, “The impact of manual and automatic testing on software testing efficiency and effectiveness,” *Indian Journal of Science and research*, vol. 3, pp. 88–93, 05 2023. [Online]. Available: https://www.researchgate.net/publication/370526552

[11] BrowserStack, “Manual testing tutorial for beginners,” https://www.browserstack.com/guide/functional-testing, 2024, accessed: 2025-01-31.

A. Software used

Below is a list of software used, along with information about their licenses. The full contents of the licenses for all dependencies can be found within the source code, specifically in the node_modules directories.

Visual Studio Code version 1.96.4, licensed under MIT License, available at https://code.visualstudio.com/.

Postman version 11.30.4, licensed under Postman EULA, available at https://www.postman.com/.

React version 18.3.1, licensed under MIT License, available at https://react.dev/.

Node.js version 20.18.1, licensed under MIT License, available at https://nodejs.org/.

Express.js version 4.21.1, licensed under MIT License, available at https://expressjs.com/.

CORS version 2.8.5, licensed under MIT License, available at https://github.com/expressjs/cors.

Axios version 1.7.9, licensed under MIT License, available at https://axios-http.com/.

dotenv version 16.4.7, licensed under BSD-2-Clause License, available at https://github.com/motdotla/dotenv.

Mongoose version 8.8.3, licensed under MIT License, available at https://mongoosejs.com/.

mongoose-encryption version 2.1.2, licensed under MIT License, available at https://github.com/joegoldbeck/mongoose-encryption.

node-cache version 5.1.2, licensed under MIT License, available at https://github.com/node-cache/node-cache.

node-cron version 3.0.3, licensed under ISC License, available at https://github.com/node-cron/node-cron.

Nodemailer version 6.9.16, licensed under MIT License, available at https://nodemailer.com/.

qs version 6.13.1, licensed under BSD-3-Clause License, available at https://github.com/ljharb/qs.

@tanstack/react-query version 5.65.0, licensed under MIT License, available at https://tanstack.com/query.

Chart.js version 4.4.7, licensed under MIT License, available at https://www.chartjs.org/.

react-chartjs-2 version 5.2.0, licensed under MIT License, available at https://github.com/reactchartjs/react-chartjs-2.

React Router DOM version 7.0.2, licensed under MIT License, available at https://reactrouter.com/.

B. Additional project files

Contents of the `.zip` archive:

- `Gloria-Zhou-304363-ProjInz2025.pdf` - project report in electronic form,
- `/source-code/` - directory containing full source code of the web application developed as part of the project,
- `readme.txt` - contains additional information about the project, including installation and setup instructions, launch procedures, and important notes on testing,
- `demo-video.mp4` - a video demonstrating the functionality of the application in action.