Python Programming and Data Visualization Project Report

Konrad Skarżyński, Inga Maziarz

January 25, 2024

1 Design

Overall visuals are created to maintain maximum simplicity by showing data that is essential to potential users at first glance without any distracting additions such as icons, different

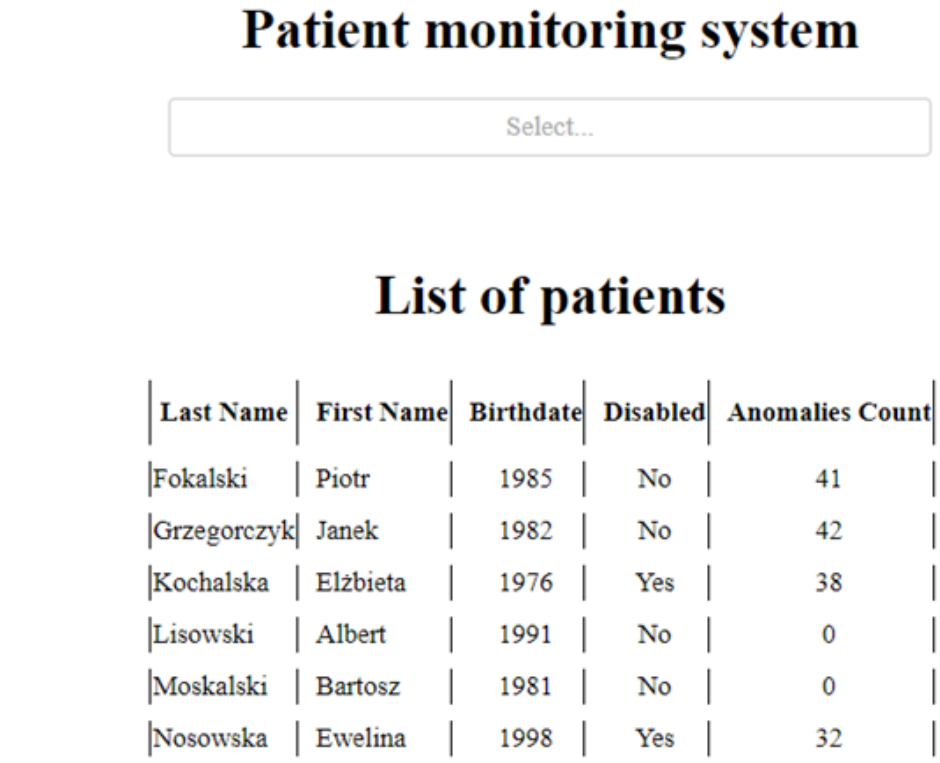

Figure 1: First part of the webpage

Webpage Title: At the top, there's a title "Patient monitoring system" in bold black letters, centered on the page. This gives users an immediate understanding of the web page's purpose.

Figure 2: Dropdown menu

Dropdown Menu: Below the title, there's a dropdown menu labeled "Select". This interactive element allows users to choose from different options (patients) that are recorded in the database.

Subheading: Underneath the dropdown menu, there's another title "List of patients" in bold black letters. This indicates the beginning of the patient list.

Data Table: The main part of the webpage is a table with five columns: Last Name, First Name, Birthdate, Disabled, and Anomalies Count. The column headers are bolded for emphasis. The table now contains six rows of patient data. Each row corresponds to a patient, with details matching the column headers. The 'Disabled' column entries are either 'Yes' or 'No', indicating the disability status of each patient. The 'Anomalies Count' column lists numerical values representing each patient's count of anomalies.

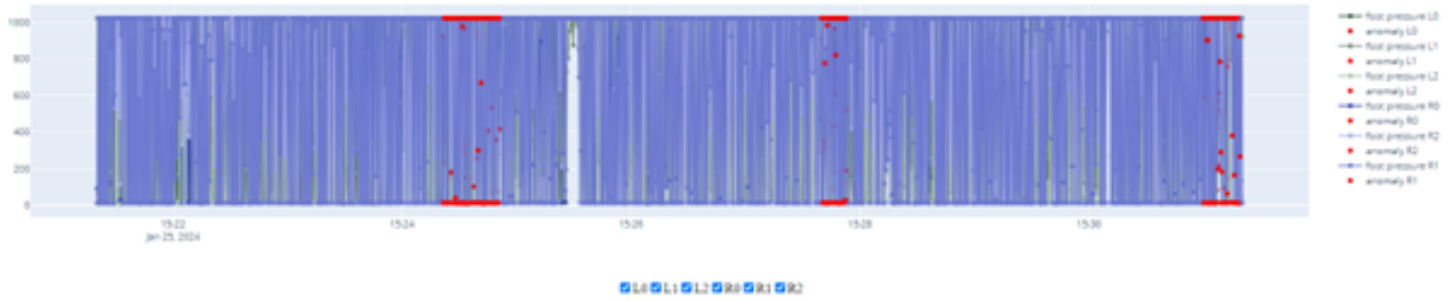

Figure 3: Graph of sensor values

Figure 4: Graph of sensor values

Graph: Graph shows graph with vertical blue lines in 3 different shades for the representation of right foot pressure points and green lines in 3 different shades for the representation of left foot pressure points, red markers to point out anomalies, and annotations once we hover over one of the points. The x-axis is labeled with timestamps of certain hours and dates. The y-axis ranges from 0 to 1000 but the labels. users can select the timeframe they want to inspect by simply selecting it with their mouse and dragging across the graph to zoom the desired fragment.

Legend: On the right side, there is a legend indicating different types of foot pressures like “foot pressure L1”, “anomaly L2”, etc., each associated with different colors. This helps users understand the meaning of the different colors used in the graph.

Control Icons: Below the graph, there are icons representing controls for choosing which pressure points will be displayed on the graph. Users can choose any combination of pressure points to make it more readable or check how pressure measurements look for specific sensors.

Figure 5: Feet diagram - custom component

Foot Diagrams: At the bottom part of the page, there are two outlined drawings of feet with dots on them. Each dot represents a different sensor located on either left or right foot. Dots and their sizes and colors are displayed

Figure 6: Feet diagram with anomalies

based on real-time data received from sensors. Each dot size depends on the pressure sensor sent to the page - The higher the pressure the bigger the dot is - to allow users to easily visually see how much pressure is put on certain parts of the feet. Colors on this custom component correspond with colors represented on the graph so the users have an easier time connecting data on the graph with the diagram. If anomalies occur then the color of the dots is red instead of shades of blue or green to further point out that anomalies are occurring at this very moment.

2 Architecture implementation

Programme is python script implementing a patient monitoring system using Flask and Dash for creating a web-based user interface. The system retrieves data from an external API, stores it in a SQLite database, and displays real-time graphs and patient information.

Database Creation:

The script creates an SQLite database named `patients.db` with two tables: `patients` and `anomalies`.

Data Fetching and Storage:

The `fetch_and_store_data` function continuously retrieves patient data from an external API, processes it, and stores it in the `patients` table of the database. It uses the `requests` library to make API calls and the `pandas` library to manipulate and store data in the database.

Anomaly Detection and Copying:

The `copy_anomalies` function runs in the background, identifying patients with anomalies and copying corresponding data to the `anomalies` table.

Web Application with Dash:

The Dash framework is used to create a web application with a user interface. The main layout includes patient information, a patient list, a real-time graph, a sensor checklist, and a custom widget (probably for displaying sensor values). The web application updates every second (`dcc.Interval(id='interval', interval=1 * 1000, n_intervals=0)`).

Callbacks:

Dash callbacks are used to update the web application dynamically based on user interactions and data changes. Callbacks are triggered by events such as selecting a patient, changing sensor preferences, or the defined time interval.

Graphical Representation:

The real-time graph is plotted using the `plotly` library, showing sensor values over time for selected sensors. Anomalies are marked in red.

Multithreading:

Multithreading is employed to run the data fetching and anomaly detection functions concurrently in the background, ensuring that the web application remains responsive.

Widget and Patient List:

The web interface includes a custom widget (`widget.Widget`) and a patient list table displaying relevant patient information.

Server Setup:

The Flask server is initialized with the `Flask(__name__)` line, and the Dash application is created with `app = Dash(__name__, server=server)`.

Execution:

The script checks if it's the main module and, if so, starts the Flask server to run the web application.