CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF BIOMEDICAL ENGINEERING

Department of Information and Communication Technologies in Medicine

Multiplatform software for FlexiGuard telemetric system

Bachelor Thesis

Study program: Biomedical and Clinical Technology
Study branch: Information and Communication Technology in Medicine

Bachelor thesis supervisor: Ing. Radim Kliment, Ph.D.

Hoang DOAN

Kladno 2020

CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

BACHELOR'S THESIS ASSIGNMENT

I. PERSONAL AND STUDY DETAILS

Student's name:	**Doan Hoang**	Personal ID number:	**474348**
Faculty:	**Faculty of Biomedical Engineering**		
Department:	**Katedra informačních a komunikačních technologií v lékařství**		
Study program:	**Biomedicínská a klinická technika**		
Branch of study:	**Informační a komunikační technologie v lékařství**		

II. BACHELOR'S THESIS DETAILS

Bachelor's thesis title in English:

Multiplatform software for FlexiGuard telemetric system

Bachelor's thesis title in Czech:

Multiplatformní software pro telemetrický systém Flexiguard

Guidelines:

Based on given documentation and specifications, implement and test multiplatform software for telemetric system FlexiGuard that is applied for the monitoring of physiological functions during physical and psychological stress. The application will work on all main operating systems (Windows, GNU/Linux, MacOS). The software will be designed to recover from error states, and in defined states will not require additional user interaction. The software will allow an online graphical preview of real-time data (monitoring function) and saving the stream of collected data into a file with the possibility of offline browsing. Create the software as modular to ensure the development of future add-ons.

Bibliography / sources:

[1] Alex Banks, Eve Porcello, React: Functional Web Development with React and Redux, ed. 1, O'Reilly Media, 2017, ISBN 9781491954621
[2] FlexiGuard – manual, version 3 technical report FBMI ČVUT, 2018
[3] Crockford, Douglas. , JavaScript: The Good Parts, ed. 1st edition, O'Reilly Media, 2008, ISBN 9780596517748

Name of bachelor's thesis supervisor:

Ing. Radim Kliment, Ph.D.

Name of bachelor's thesis consultant:

Ing. Pavel Smrčka, Ph.D.

Date of bachelor's thesis assignment: **17.02.2020**
Assignment valid until: **19.09.2021**

Karel Hána

doc. Ing. Karel Hána, Ph.D.
Head of department's signature

prof. MUDr. Ivan Dylevský, DrSc.
Dean's signature

DECLARATION

I hereby declare that I have completed this thesis having the topic “Multiplatform software for FlexiGuard telemetric system” independently and that I have attached an exhaustive list of citations of the employed sources.

I do not have a compelling reason against the use of the thesis within the meaning of Section 60 of the Act No.121 / 2000 Sb., on copyright, rights related to copyright and amending some laws (the Copyright Act).

In Kladno, …..…….……...

Hoang Doan

ACKNOWLEDGEMENTS

I would like to greatly thank my supervisor Ing. Radim Kliment, Ph.D. for his extensive support, valuable advice and leadership throughout my work. Moreover, my thanks also go to Ing. Pavel Smrčka, Ph.D. and Ing. Tomáš Veselý for providing their expert insights.

ABSTRAKT

Multiplatformní software pro telemetrický systém Flexiguard

Hlavním cílem této práce bylo vyvinout multiplatformní software pro telemetrický systém FlexiGuard, který je převážné využíván pro výcvik záchranných jednotek. Nejprve byla vykonána analýza systému FlexiGuard za účelem sběru požadavků pro vývoj. Dále byla vybrána vhodná multiplatformní technologie na základě provedené rešerše. Následovaly návrh a implementace softwarové architektury a uživatelského prostředí s důrazem na modularitu a spolehlivost. Poté byl software sestaven pro operační systémy macOS, Windows a GNU/Linux. Nakonec provedené testy prokázaly, že vyvinutý software může být plnohodnotně nasazen do systému FlexiGuard.

Klíčová slova

Electron, React.js, FlexiGuard, multiplatformní, telemonitoring

ABSTRACT

Multiplatform software for FlexiGuard telemetric system

The main aim of the thesis was to develop a multiplatform software for the FlexiGuard telemetric system applied predominantly in the training of first-responders. Firstly, the FlexiGuard system was analyzed to acquire development requirements. Next, suitable multiplatform technology was selected upon conducted research. This was followed by the design and implementation of the software architecture and GUI with the focus on modularity and reliability. Then, the software was deployed for macOS, Windows, and GNU/Linux operating systems. Lastly, the performed tests confirmed that the final version of the software can be coherently integrated into the FlexiGuard system.

Keywords

Electron, React.js, FlexiGuard, multiplatform, telemonitoring

Table of Contents

List of abbreviations

Abbreviation	Importance
API	Application programming interface
COM	Communication port
CSV	Comma-separated values
CTU	Czech Technical University
DOM	Document Object Model
FBME	Faculty of Biomedical Engineering
GPL	GNU General Public License
GPS	Global Positioning System
GUI	Graphical user interface
IPC	Inter-process communication
JSON	JavaScript Object Notation
JSX	JavaScript XML
LGPL	GNU Lesser General Public License
MIT	Massachusetts Institute of Technology (license)
NSIS	Nullsoft Scriptable Install System
NoSQL	Not Only Structured Query Language
OS	Operating system
UC	Use case
UML	Unified Modeling Language
USB	Universal Serial Bus

1 Introduction

FlexiGuard telemonitoring system is a proven long-term project of FBME of CTU in Prague. It is being utilized in a wide range of applications, mainly in the training of rescuers, firemen, soldiers, etc. FlexiGuard's core functionality is providing comprehensive physiological and environmental online data about each monitored individual which increases overall safety and provides key inputs for the decision-making in the scope of the training or mission [1].

The FlexiGuard user application is an essential part of the system, as it processes and visualizes the complex data in a reliable and user-friendly interface. Unfortunately, its foundations were built on multiplatform technology that has recently lost the active support of its author and maintainer. In practice, this would soon bring major functional and security limitations for the continuous development of the project as a whole. As a result, a decision to replace the current application was made.

The output of this thesis is a robust multiplatform application that can be fully incorporated into the Flexiguard system. The application was built around fundamental concepts transferred from the original version. However, great emphasis was put on modularity and revisiting of the user interfaces with the ultimate goal to enhance both developer and user experience. This approach transcended into the whole development of the application which is thoroughly described in this thesis.

2 Aims

The main aim of this thesis was to deliver a full-fledged multiplatform user application for the FlexiGuard system that can be run on Windows, macOS, and GNU/Linux. To achieve the main aim, a list of minor aims was formed and followed throughout the work:

- Understand the inner workings and specifics of the FlexiGuard system.
- Analyze the current user application to aid the forming of requirements and application design.
- Complete the set of application requirements based on initial analysis and input from stakeholders.
- Select suitable technologies for multiplatform development and other requirements.
- Model the application architecture with modularity and reliability in mind.
- Improve the user experience and design the graphical user interface.
- Implement defined requirements and deploy the application for required platforms.
- Carry out tests of the implemented modules and functionalities.

3 Analysis

This chapter is primarily focused on the analysis of the FlexiGuard system, the definition of the application requirements, and the selection of the relevant technology. Moreover, it offers insight into the current state of multiplatform development for desktop environments.

3.1 FlexiGuard system

FlexiGuard is a telemetric system for remote monitoring of physiological and environmental parameters of the human body. The system is generally developed for and used by units of firefighters, soldiers, pilots, rescuers, and other first-responders. The development is led by FBME of CTU in Prague in cooperation with expert bodies in medicine and safety engineering [1].

FlexiGuard allows for individual or collective monitoring of personnel in real-time via personal monitoring units that are equipped with a range of sensors. Processed online results tailored to individual physiological profiles are displayed in the visualization unit. Also, a detailed offline analysis is available as the measured data are being stored. The main components of the system are the personal monitoring units, visualization unit (user application), and the communication module which now supports the monitoring of up to 30 persons at once. Overall, the FlexiGuard system improves the overall safety of monitored personnel and serves as a valuable decision-support instrument in both training and live-action environments [1].

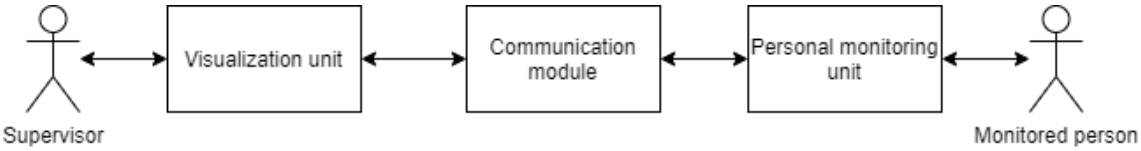

Figure 3.1: Simplified diagram of the FlexiGuard system

3.1.1 Personal monitoring unit

Each monitored person wears the personal monitoring unit on the chest. The unit's case encapsulates a microcontroller, radio communication interface, power supply unit, incident signaling button, and the basic set of sensors. Primary monitored parameters are heart rate, breath rate, physical activity, acceleration, body temperature, both temperature and humidity of the environment [1]. With this basic set of sensors, the physical and physiological state of monitored personnel can be appropriately determined [2]. Moreover, the accuracy of the measurements is higher than with consumer electronics

alternatives [3]. The sensor set can be extended with a GPS receiver and other additional modules. The measured parameters can be continuously stored into an onboard memory card and sent to the visualization unit up to approximately 2 km away. Also, the personal monitoring units broadcast the measured data every 3 seconds in a predefined time window. The case is mechanically rugged and sweatproof [1]. The unit lasts up to 24 hours of active use and does not require intervention or operation from the monitored person [4].

Figure 3.2: Mounted personal monitoring unit [5]

3.1.2 Communication module

The two-way wireless communication between the visualization unit and personal monitoring units is facilitated by FlexiGuard radio modules. The radio modules are connected to the USB ports of the visualization unit and the data are processed through a virtual serial port (COM). The wireless communication is based on the XBee protocol. [6] In practice, 1 or 2 communication modules are used simultaneously. [4]

3.1.3 Visualization unit

Visualization unit (user application) retrieves online data from personal monitoring units through a wireless communication interface. Measured parameters and their trends are displayed in real-time. At the same time, estimations of the physiological state are being calculated specifically for each monitored person. Furthermore, the data are stored on the visualization unit device for later offline analysis.

It is based on the deprecated Chrome Apps platform enabling the application to run within the Google Chrome web browser on Windows, macOS, and GNU/Linux [4]. The measurement records with the support of event logging are stored in CSV files [6].

The graphical user interface is divided into 7 sections: Header, Overview, Details, Map, History, Settings, and Manual [4].

Header – contains the application's navigation and displays the state of communication and storing data.

Figure 3.3: Header – original FlexiGuard software

Overview – provides a simplified online overview of all personal monitoring units and visualizes all essential parameters with easily comprehensible graphical elements (gauges, color indicators, icons, etc.).

Figure 3.4: Online overview – original FlexiGuard software

Details – presents detailed online data and adds graph visualization. Includes user controls for event logging and turning off the incident alarm.

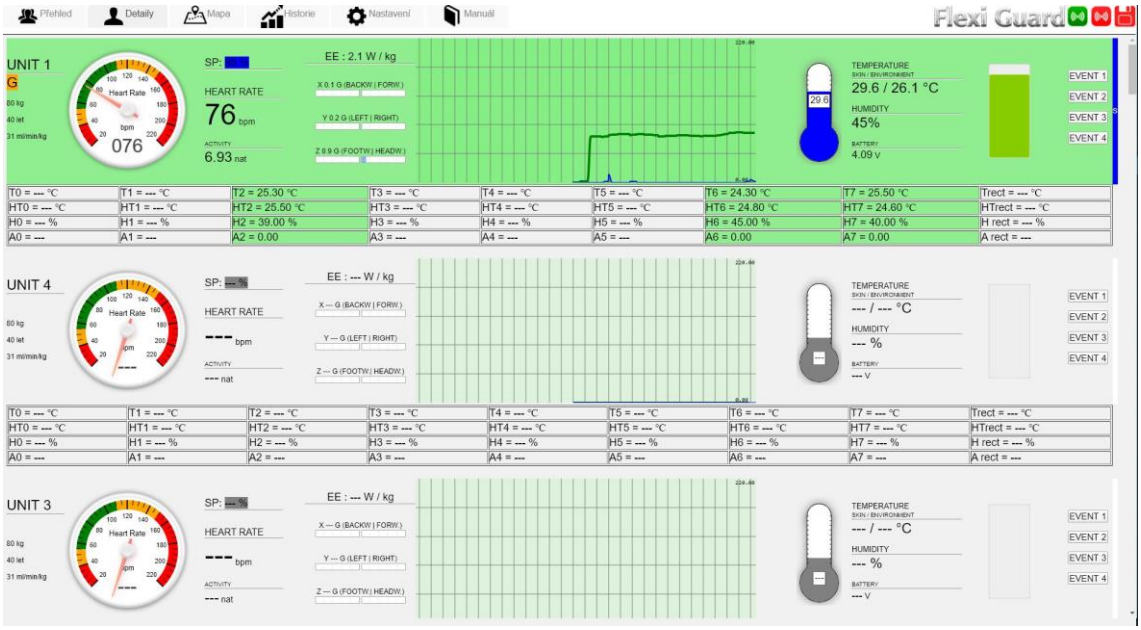

Figure 3.5: Online detail – original FlexiGuard software

Map – shows GPS coordinates of personal monitoring units and their position on the map.

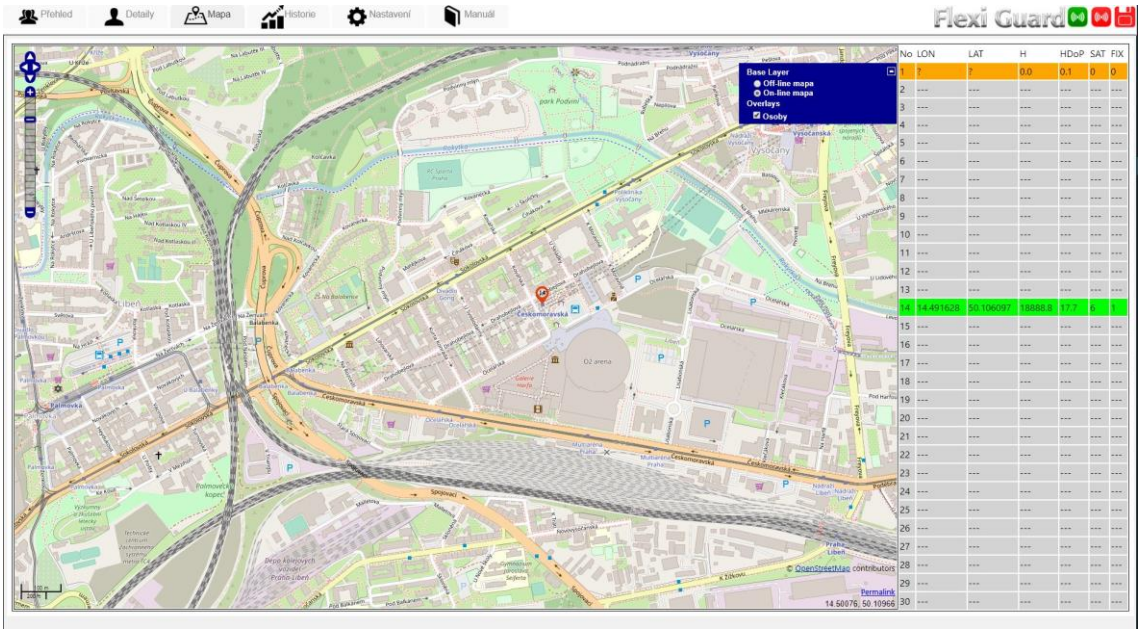

Figure 3.6: Map – original FlexiGuard software

History – visualizes data trends from CSV files upon time window and monitoring unit selection.

Figure 3.7: History – original FlexiGuard software

Settings - allows the user to set the communication modules, choose output CSV file for data storage, and configure physiological profiles. Moreover, broadcast controls for personal monitoring units and debugging console are present.

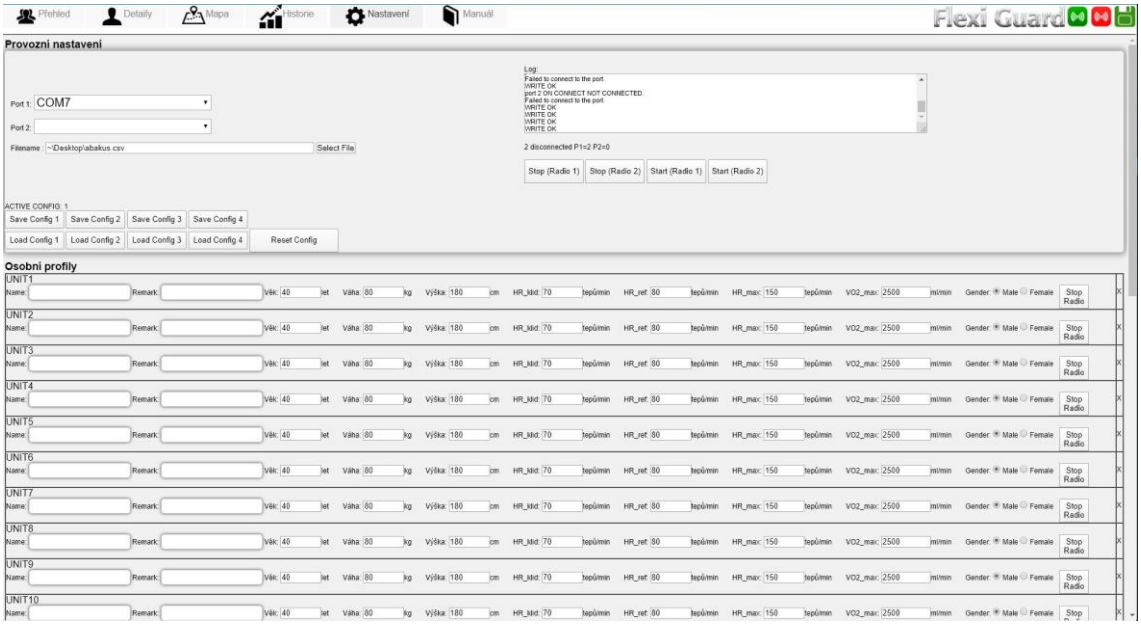

Figure 3.8: Settings – original FlexiGuard software

3.2 Requirements

The requirements were determined in cooperation with assigned consultants who are directly involved in the development of FlexiGuard. Thanks to the user feedback and their developer insight, the original concepts were reinforced. Furthermore, new requirements were coherently introduced in the scope of the whole FlexiGuard system development. Besides, observations from the application analysis were taken into account.

3.2.1 Functional requirements

F1: **Online data visualization** – visualization of online data must be easily readable and intuitive both in a simplified collective view or a detailed individual view. Usage of graphical elements (gauges, graphs, icons, bars, etc.) should be adopted.

F2: **Offline data visualization** – presentation of recorded data must be quickly accessible and should allow for a swift comparison of monitored individuals.

F3: **Location data visualization** – the location data must be displayed on a map along with detail of the GPS coordinates.

F4: **Event logging and data storage** – the application must be capable of storing processed data and events logs into CSV files with a predefined format.

F5: **Offline data management** – CSV files with recorded data must be conveniently manageable.

F6: **User and team profile management** – the application must offer management of monitored personnel and teams.

F7: **Data storage and visualization settings** – data storage and visualization parameters must be adjustable.

F8: **Communication with the FlexiGuard hardware** – the application must be compatible with the FlexiGuard data format, communication modules, and personal monitoring units.

3.2.2 Non-functional requirements

N1: **Multiplatform support** – macOS, GNU/Linux, and Windows operating systems must be able to run the application.

N2: **Touch-friendly graphical user interface** – the graphical user interface must be touch-friendly to further improve the usability and flexibility for the users.

N3: **Modularity and extensibility** – the design and implementation of the functions must be centered around modularity and extensibility to simplify future development.

N4: **Reliability** – the application must be robust and capable of self-recovery from basic errors.

N5: **Straightforward user experience** – the application must require minimal interventions from the user and complicated actions should be abstracted to deliver a simplified user experience.

3.3 Use cases

This section presents the application use case contexts derived from the functional requirements. The distinct use cases are described and represented in the UML diagrams to provide a clear definition of interactions between the user and the application. The use case contexts also served as the basis for the user experience and GUI design.

The expected singular user of the application is the supervisor of the monitoring session or other personnel in related roles. User authorization and authentication were not demanded in the requirements.

3.3.1 Online

Use cases tied to the online flow of the data from personal monitoring units.

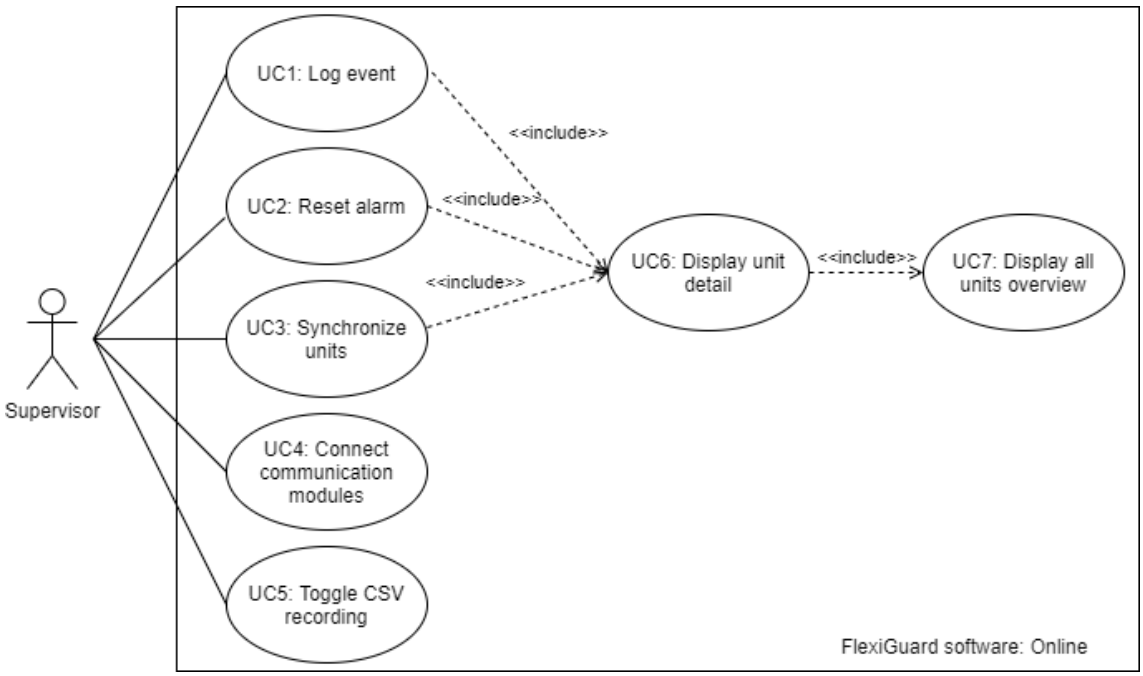

Figure 3.9: Use case diagram – online

UC1: **Log event** – log predefined event into the CSV file.
UC2: **Reset the alarm** – turn off the alarm triggered by the personal monitoring unit.
UC3: **Synchronize units** – broadcast synchronization command to personal monitoring units.

UC4: **Connect communication modules** – set/reset the connection with communication modules.
UC5: **Turn on/off CSV recording** – toggle saving data into the CSV file.
UC6: **Display unit detail** – display detailed data and advanced visualizations.
UC7: **Display all units overview** – display an online overview of all units with simplified visualization.

3.3.2 History

Use cases for the visualization of the offline data from the data storage.

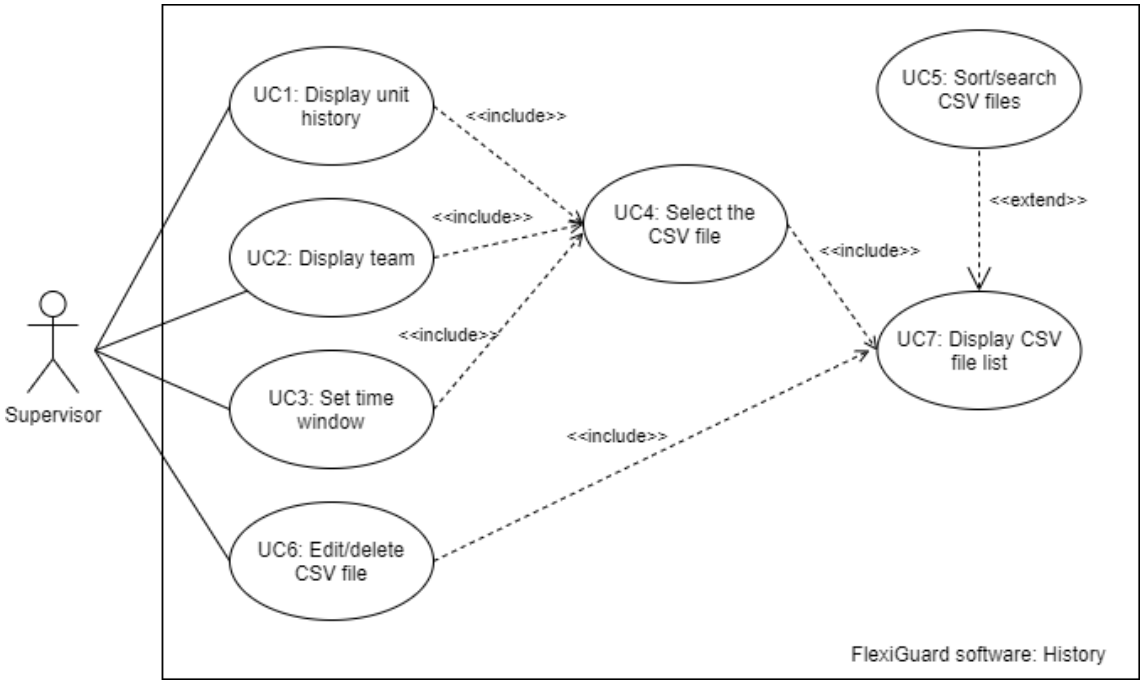

Figure 3.10: Use case diagram – History

UC1: **Display unit history** – display recorded data of a chosen unit from the selected CSV file in the specified time window.
UC2: **Display team** – display the team and its members included in the selected CSV file.
UC3: **Set time window** – set time window for visualization from the selected CSV file.
UC4: **Select the CSV file** – select the CSV file to be visualized.
UC5: **Sort/search CSV files** – sort or search the list of all available CSV files.
UC6: **Edit/delete CSV file** – alter the information about the CSV file (file path, note).
UC7: **Display CSV files list** – list all finished and ongoing CSV files.

3.3.3 Map

Use cases for the visualization of the personal monitoring units' location data.

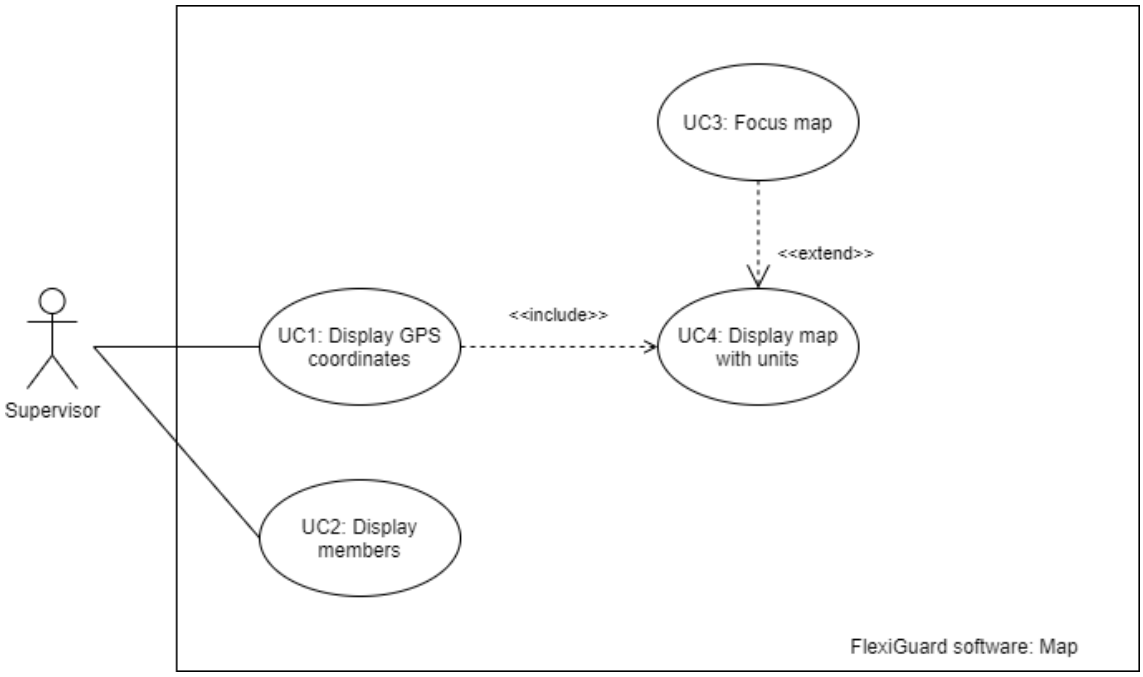

Figure 3.11: Use case diagram – Map

UC1: **Display GPS coordinates** – display GPS coordinates and other details about the unit.

UC2: **Display members** – display a list of team members with simplified location visualization.

UC3: **Focus map** – focus and zoom on the selected unit with a detected location.

UC4: **Display map with units** – display all units with a detected location.

3.3.4 Teams

Use cases for the management of the monitored teams and their members.

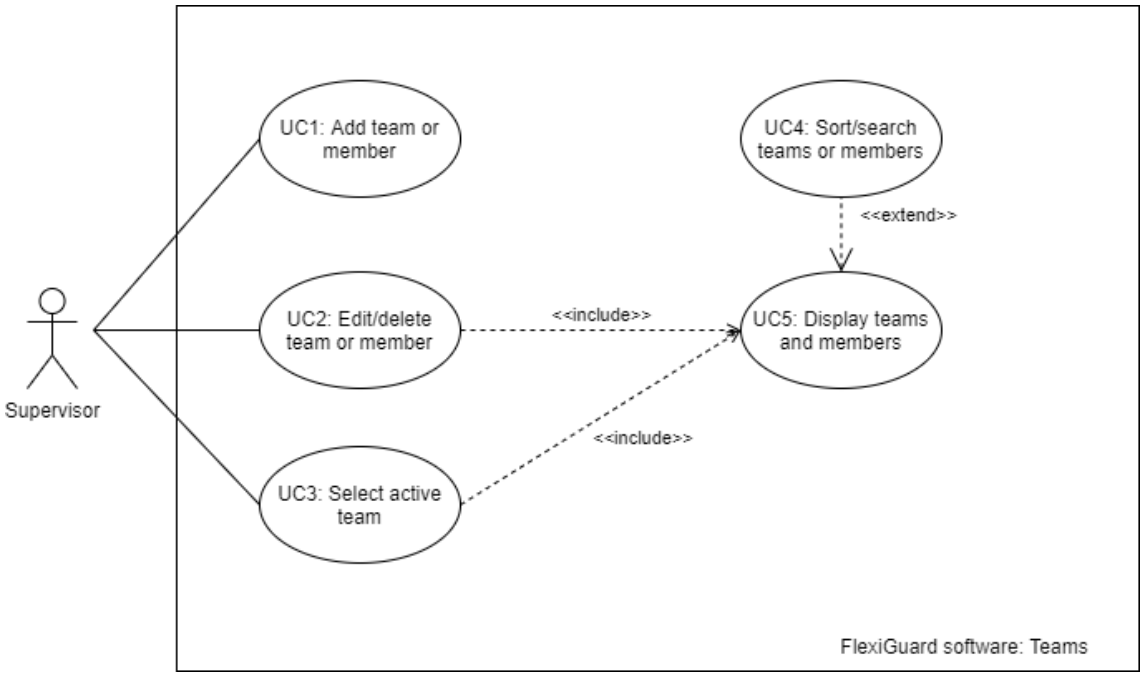

Figure 3.12: Use case diagram – Teams

UC1: **Add team or member** – enter the details and save the entity.
UC2: **Edit/delete team or member** – alter the information about the entity.
UC3: **Select the active team** – select the active team for the performance calculations and profile handling.
UC4: **Sort/search teams or members** – sort or search the lists of users or teams.
UC5: **Display teams and members** – display the lists of all users and teams.

3.3.5 Settings

Use cases for configuring user settings.

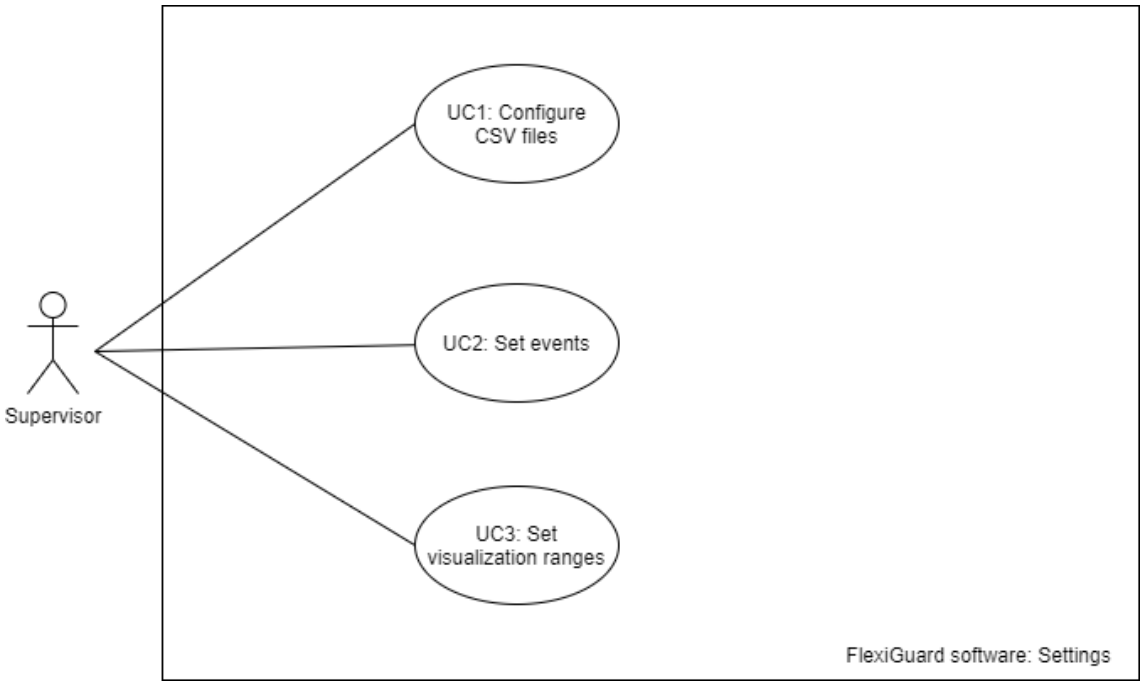

Figure 3.13: Use case diagram – Settings

UC1: **Configure the CSV files** – select the file directory and data sets to be saved into the CSV files.

UC2: **Set events** – define events that can be logged into the CSV file.

UC3: **Set visualization ranges** – set ranges for the graphs, meters, gauges, and other visualization elements.

3.4 Multiplatform development

The integral requirement for the development of the application was the support of multiple platforms - macOS, Windows, GNU/Linux. Accounting the time and resource restrictions of this thesis, the optimal method to meet the requirement was to employ multiplatform (cross-platform) development.

Multiplatform (cross-platform) development delivers software that is capable of fully-functioning on different operating systems with the use of a single codebase. Therefore, major advantages are the acceleration of the development process, reduced costs, and accessibility to a wider user base. On the other hand, universally developed applications can add friction during integration, distribution, and testing. Moreover, the performance and user experience can be inferior to their native counterparts.

3.5 Multiplatform frameworks

Multiplatform frameworks provide essential tools and software interfaces for the development of multiplatform applications. Also, they generally increase the comfort and effectivity of the development as repetitive and low-level tasks are abstracted. As an integral software structure of the developed FlexiGuard software, a suitable framework had to be selected upon the research of available options. These frameworks were considered:

- Electron
- NW.js
- Qt

Table 1: General comparison of multiplatform frameworks [7]

	Electron	NW.js	Qt
Language	JavaScript	JavaScript	C++
Maintainer	GitHub	NW.js community	The Qt Company
Required OS	Yes	Yes	Yes + more
License	MIT	MIT	GPL + LGPLv3

3.5.1 Electron

Electron is a wide-spread open-source framework with strong community backing that leverages web technologies Chromium and Node.js to deliver multiplatform apps. It was originally developed by GitHub for internal purposes [8].

Advantages

- Advanced tools for building and distribution
- Low-level native APIs
- Strong corporate backing by GitHub (Microsoft)
- Most popular open-source multiplatform framework (GitHub stars[1], Npm downloads[2])
- Used in established apps – Visual Studio Code, Twitch, Discord, Slack, etc.
- Flexible ecosystem of libraries and tools

[1] Overall user popularity on the GitHub software development platform [31].

[2] Number of downloads on the Npm package manager [32].

Disadvantages

- Large size of the end application
- Basic documentation
- Demanding on memory resources
- Fragmented packaging for each platform
- No source code encryption

3.5.2 NW.js

NW.js (formerly Node-Webkit) is a renowned open-source framework that allows cross-platform development with web technologies Chromium and Node.js. It was first released by the Intel Open Source Technology Center [9].

Advantages

- High-quality documentation
- Embedded source code protection
- Legacy systems support
- Simple architecture
- Flexible ecosystem of libraries and tools

Disadvantages

- Unsufficient native platform APIs
- Large file of the end application
- Lack of development utilities
- Required patching of Chromium
- Secondary support of multi-context paradigm in Node.js

3.5.3 Qt

Qt is a robust and universal framework with a wide range of supported platforms. It offers domain-specific integrated development environment and tools. It is predominantly used in substantial commercial and industrial projects [10].

Advantages

- Maturity and stability
- Large variety of supported platforms
- Used in complex applications by large corporations
- Low-level native APIs
- Availability of development tools
- Commercially developed and maintained by Qt Group

Disadvantages

- High development complexity
- Resource-heavy build environment
- Closed system of extensions and add-ons
- Non-standard C++ interface
- Requires licensing for commercial use

3.5.4 Multiplatform framework evaluation

To ensure consistent and clear evaluation of the frameworks, a custom rating system was designed focusing on 3 categories, each with 5 sub-categories:

- **Ease of implementation**
    - Thesis author's experience with the framework
    - Thesis author's experience with the technology
    - Gradual learning curve
    - High-quality documentation
    - Extensive native APIs
- **Framework ecosystem and support**
    - Built-in design tools
    - Development tools
    - Advanced distribution tools
    - Wide developers community
    - Commercial backing
- **Extensibility and flexibility**
    - Fully open-source
    - Availability of add-on substitutes
    - Transferability of the technology stack
    - Rich add-on ecosystem
    - Support of mobile platforms

The frameworks could receive a maximum of 5 points in each category, 15 points in total. For each sub-category that was accommodated by the framework, 1 point was awarded.

Table 2: Multiplatform frameworks evaluation by sub-categories

Sub-category	Electron	NW.js	Qt
Framework experience	1	0	0
Technology experience	1	1	0
Gradual learning curve	1	1	0
High-quality documentation	0	1	1
Extensive native APIs	1	0	1
Built-in design tools	0	0	1
Development tools	1	1	1
Advanced distribution tools	1	0	1
Wide developers community	1	1	1
Commercial backing	1	0	1
Fully open-source	1	1	0
Rich add-on ecosystem	1	1	1
Transferability of the tech stack	1	1	0
Availability of add-on substitutes	1	1	0
Support of mobile platforms	0	0	1
Total points	**12**	**9**	**9**

Table 3: Multiplatform frameworks evaluation by categories

Category	Electron	NW.js	Qt
Ease of implementation	4	3	2
Framework ecosystem and support	4	2	5
Extensibility and flexibility	4	4	2
Total points	**12**	**9**	**9**

According to the rating system, Electron was chosen as the best-fitted multiplatform framework for FlexiGuard software development. Generally, its features are sufficiently powerful and the ecosystem provides stable foundations for the development. In comparison with NW.js, which is technologically similarly based, it offers a more comfortable developer experience and better native APIs. The Qt framework is very universal and robust, but its complexity can be overwhelming and excessive for projects of the FlexiGuard's size and scope.

4 Design

The design of the application's principal parts was based on the outputs of the initial analysis.

4.1 Application architecture

The application architecture is largely derived from the structure of the selected multiplatform framework - Electron. Electron allows desktop application development with the usage of web technologies Chromium and Node.js.

Electron's architecture is constituted of two general processes – the Main and the Renderer. In practice, the Main and the Renderer processes are operated by 2 or more operating system level processes that run concurrently and communicate via inter-process communication (IPC). Moreover, these low-level processes have their memory and resources separated. This concept stems from the incorporation of the Chromium web browser, where each browser window owns a separate process, so in a case of a fatal error, only the designated process crashes and not the whole application. As a result, the multi-process paradigm contributes to a more secure and reliable environment [11].

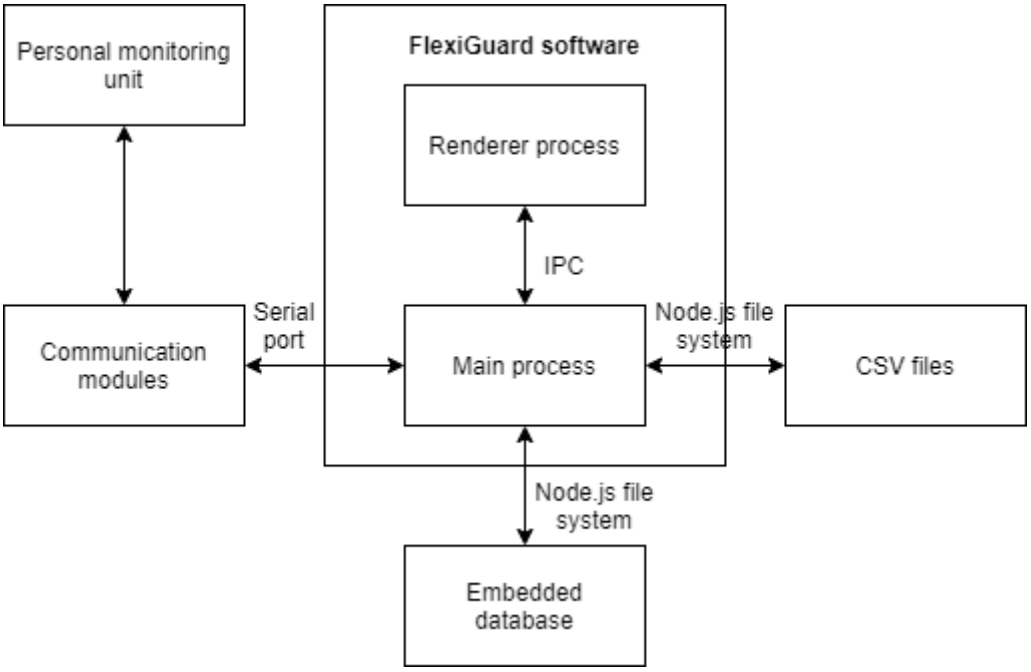

Figure 4.1: Software architecture within the FlexiGuard system

4.1.1 Main process

The Main process is a default entry point and the core of the application. It is responsible for managing the Renderer instances and key application functionalities. Furthermore, it

provides access to native GUI elements, low-level native API, Node.js modules, Node.js API, and also the majority of the Electron API. Also, there can be only one instance of the Main process [8].

4.1.2 Renderer process

The Renderer process manages the user interface of the application, which is practically a web page. It allows access to the DOM API, Node.js API, and some of the Electron API. Multiple Renderer processes can be run at once and they are managed by the Main process [8].

4.1.3 Inter-process communication (IPC)

To provide a communication interface between the Renderer and the Main process, Electron utilizes two-way inter-process communication (IPC). Besides, the communication can be realized asynchronously or synchronously, and it is implemented as an Event Emitter class of Events in Node.js API. The IPC messages contain the arbitrary payload and the channel name to be sent through. The messages are obtained by the channel listeners placed in the processes [8].

4.2 Data storage

To fulfill the defined requirements, the application needs to store user settings, monitored teams and their members' physiological profiles, and most importantly the measured data of each monitoring session.

The original version of FlexiGuard software uses CSV files to store all the aforementioned datasets. However, it does not create connections between physiological profiles, measured data, and the monitoring session. As a consequence, the features for monitoring session management, team management, and offline data analysis were quite limited.

To extend the possibilities with all acquired data and use cases, the application deploys a hybrid approach to data storage utilizing both CSV files and embedded database. The datasets are divided to:

- **CSV files** – measured data, calculated performance data
- **Embedded NoSQL database** – teams, physiological profiles, monitoring sessions, user settings

4.2.1 CSV files

The measured data (acceleration, heart rate, etc.) and calculated performance data are stored in standard CSV files. Each line of the text file is a data record that consists of

fields separated by a delimiter. The CSV format allows for easy readability and transferability even beyond the bounds of the application interface.

4.2.2 Embedded NoSQL database

The monitored teams, user settings, and monitoring sessions' information are stored in an embedded NoSQL database. Embedded database is bundled within the application and does not require any additional setup by the end-user.

The NoSQL database concept was chosen as it does not require a fixed database model which allows for flexibility and fast iterative development. Moreover, the application utilizes a document-based type of NoSQL which simplifies data queries using JSON objects or similar data formats [12].

The data are stored in documents which are organized into document collections:

- Users
- Teams
- Records
- Settings

Although the NoSQL approach does not enforce entity relations and fixed database model, for the needs of the application, a simple model and relations were applied. The application establishes the relations by document nesting and referencing.

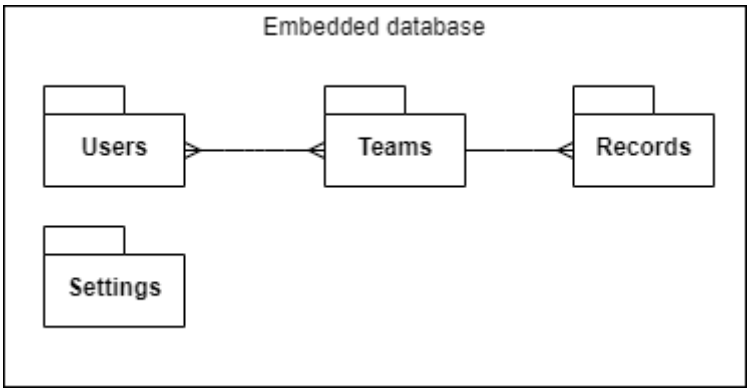

Figure 4.2: Simple NoSQL database model

4.3 Data processing

To ensure modularity and simple extensibility, data processing is fragmented into modules with separated concerns that are orchestrated by the Electron's Main process (described in 4.1.1).

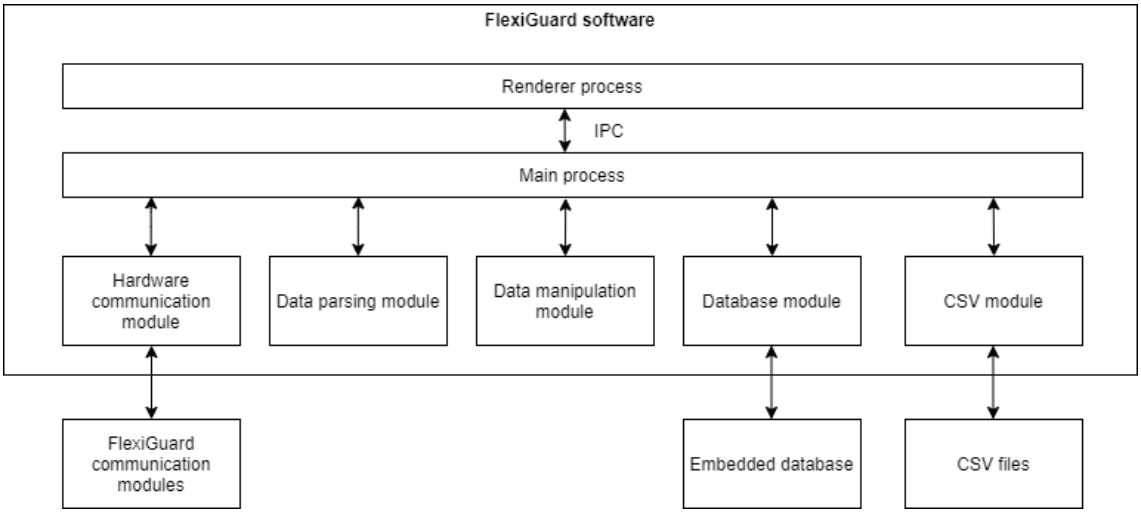

Figure 4.3: Data processing architecture

4.3.1 Hardware communication

Online communication with personal monitoring units is delivered through FlexiGuard communication modules which are connected via USB ports. The data flows through a virtual serial port (COM). In comparison with the original version of the software, the number of simultaneously connected FlexiGuard communication modules is not limited to 2.

The hardware communication module integrates these functions:

- **Automatic detection and connection** – automatically detects FlexiGuard hardware by unique manufacturer identification and initiates the connection. This removes potential user-induced errors, simplifies the user experience, and allows automatic error recovery.
- **Unit synchronization** – synchronizes the broadcasting window for the personal monitoring units. The synchronization is automatically started after the incorrect broadcast is detected or manually by the supervisor.
- **Alarm** – the supervisor can turn off the alarm on a specific personal monitoring unit.

The connection is a critical function of the application, therefore autonomous and robust connection management was designed.

The states of communications are:

A. **Disconnected** – the default state when the communication module is not connected to the application.
B. **Connecting** – the connection configuration is loaded and the application is trying to establish the connection.
C. **Connected** – the communication modules are connected and the application awaits the data.

D. **Reading** – the data are being received and handed over for processing.

The transitions between states are:

1. **Connection configured** – the connected modules are identified and the communication configuration is set.
2. **Port not open** – the connection is not established.
3. **Port open** – the connection is established with the given configuration.
4. **Port closed** – the connection is terminated.
5. **Data flowing** – data are being received.
6. **Synchronization** – the personal monitoring units are being synchronized after a corrupt broadcast window.

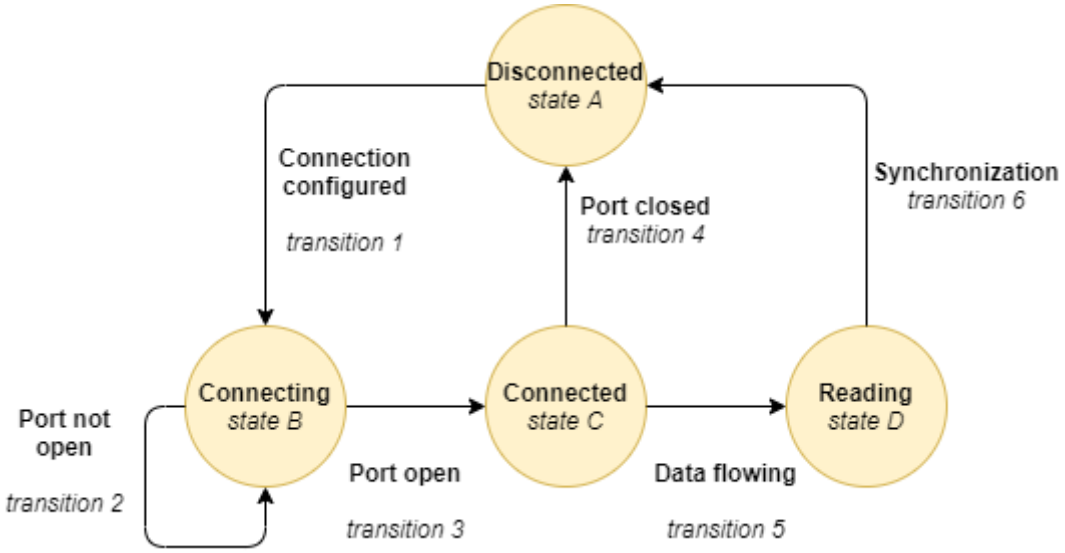

Figure 4.4: State diagram of hardware communication

4.3.2 Data parsing

The module receives raw data in the hexadecimal format. Its main purpose is to output structured data sets in a structured JSON format. The key functions are:

- **Device detection** – personal monitoring units send various types of data packets depending on the device type. The detection is an essential step for further processing.
- **Categorization** – the raw packet is sliced and the data are categorized into basic, location, performance, and additional node datasets.
- **Data conversion** – obtained parameter values and states are converted from hexadecimal domain-specific formats into common units.

4.3.3 Data manipulation

The structured and converted data in JSON format are then separately managed for each personal monitoring unit. It specifically administers:

- **Online data storage** – filters through the incoming data and assigns it to the units. Also, colliding cross-over data from FlexiGuard communication modules are cleared.
- **Graphs** – the data are appended into various online graphs. The time series is automatically adjusted according to the time window length setting.
- **Physiological profiles** – the physiological profiles of the active team are stored.
- **Performance data** – as the module hosts the physiological data, the tailored performance calculations are executed.
- **Renderer state** – updates the state of the components in the Renderer process, as the module manages online data.

4.3.4 Database interface

The module provides endpoints for database manipulation with team management data, CSV files information, and user settings. It initializes the embedded database on every app start and creates the database after the initial application installation.

4.3.5 CSV interface

All operations with the CSV files that store data from monitoring sessions are handled by the CSV interface. Its primary functions are:

- **Creating files** – on every start of online data recording, a new CSV file is created, headers and CSV files properties are set. In case of writing error or output file corruption, a new file is created automatically to prevent data leak.
- **Writing** – writes the stream of data into the designated CSV file.
- **Reading** – loads the chosen CSV file and reads the stream of data.
- **Conversion** – flattens the nested JSON data into CSV format and vice-versa. Also, converts the formatting (delimiters, decimal points) to the Czech locale.

4.4 User interface

This chapter presents the inner structure and the design of the graphical user interface that runs in the Electron's Renderer process (described in 4.1.2).

4.4.1 GUI design

The GUI design process was centered around FlexiGuard's role of robust, unintrusive, and mostly self-maintaining decision support system for safety monitoring. The design focuses on providing essential information with straightforward and low input-dependent

user experience. The GUI utilizes intuitive elements like traffic-light indication, colored icons, meters, gauges, and graphs.

The composition of the GUI is derived from the original FlexiGuard software. The GUI is sectioned into Header, Online, History, Map, Teams, and Settings. The wireframes were sketched in the Wireframe.cc service [13].

Header – the header is present in every major user view. It hosts the navigation, basic controls, and information about the application's status.

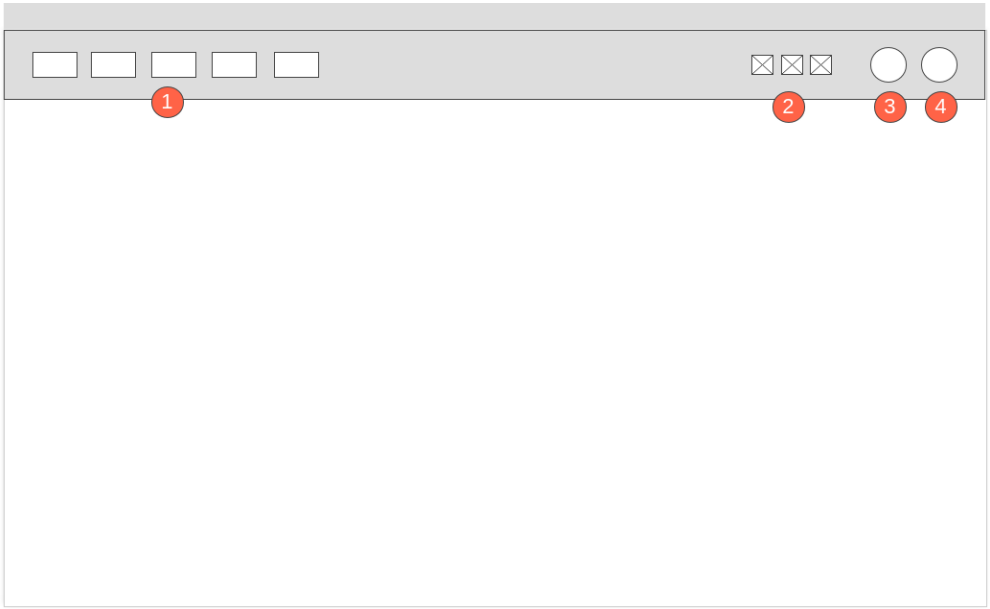

Figure 4.5: Header – wireframe

1) **Navigation panel** – contains icon buttons that switch user views (sections).
2) **Status icons** – colored icons display the state of communication module connection and data storing.
3) **Record button** – switches the state of data storing into CSV file.
4) **Reset dropdown** – allows synchronization of monitoring units and reset of the communication module connection.

Online overview – displays an online overview of all units with a simplified visualization of data.

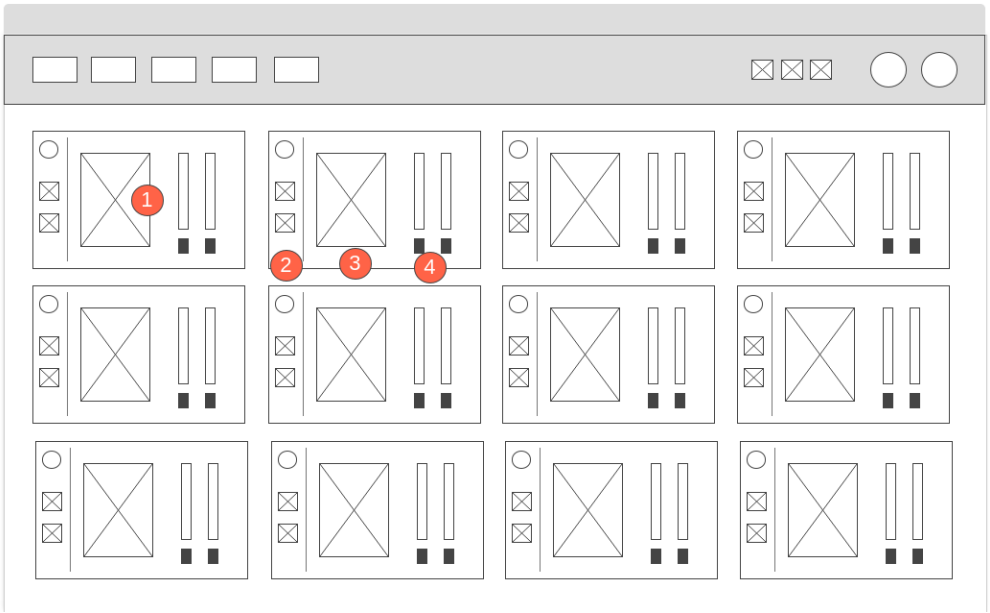

Figure 4.6: Online overview – wireframe

1) **Unit box** – encompasses the online data about the unit. Clicking on the box opens a detailed online view of the unit.
2) **Unit status bar** – uses icons to display unit number, battery state, GPS state, presence of additional modules. Signals connection states and alarms are presented via background color.
3) **Heart rate gauge** – numerically and graphically shows the online heart rate with colors for different thresholds.
4) **Meters** – visualize the skin temperature and physical activity with colors for different thresholds.

Online detail – displays detailed online data for the selected unit with advanced visualization. In contrast with the original software, only the selected unit is visualized to save the processing resources.

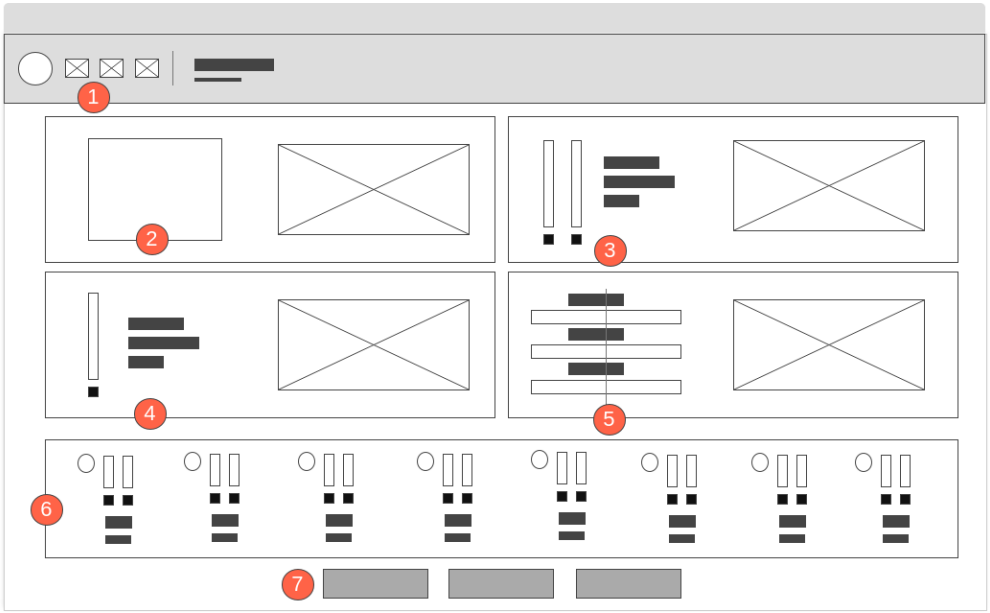

Figure 4.7: Online detail – wireframe

1) **Unit status bar** – uses icons to display unit number, battery state, GPS state, presence of additional modules. It contains the name of the assigned unit user. Signals connection states and alarms via background color.
2) **Heart rate detail –** visualizes the heart rate using a gauge and graph.
3) **Performance detail** – informs about the unit user's physiological profile and performance through meters with color indication and a graph.
4) **Temperature detail** – shows information about humidity, skin and environment temperature via meter with color indication and a graph.
5) **Acceleration detail** – displays acceleration in 3 axes through meters and a graph.
6) **Node detail** – visualizes the temperature and physical activity details of additional sensor nodes. It includes colored meters for skin temperature and physical activity.
7) **Log & alarm buttons** – log predefined events into the CSV file and turn off the alarm (alarm button only present when the incident alarm is on).

History overview – provides swift access to individual offline unit data and monitoring session management.

Figure 4.8: History overview – wireframe

1) **CSV file information** – displays the information about the selected record and provides time selectors for time window setting. The newest record is pre-selected automatically.
2) **User chips** – presents the unit users (name, unit number) for the selected CSV file. Clicking on the user chip opens the detailed view of measured parameter trends for the selected unit.
3) **List of CSV files** – lists all CSV record files with name, note, start time, end time, unit users' details.
4) **Search & Load** – allows searching CSV record files and loading of external CSV files that are not stored in the database.
5) **Manage buttons** – selects the record for visualization, edits, or deletes the CSV file.

History detail – displays the offline and user data of the selected unit.

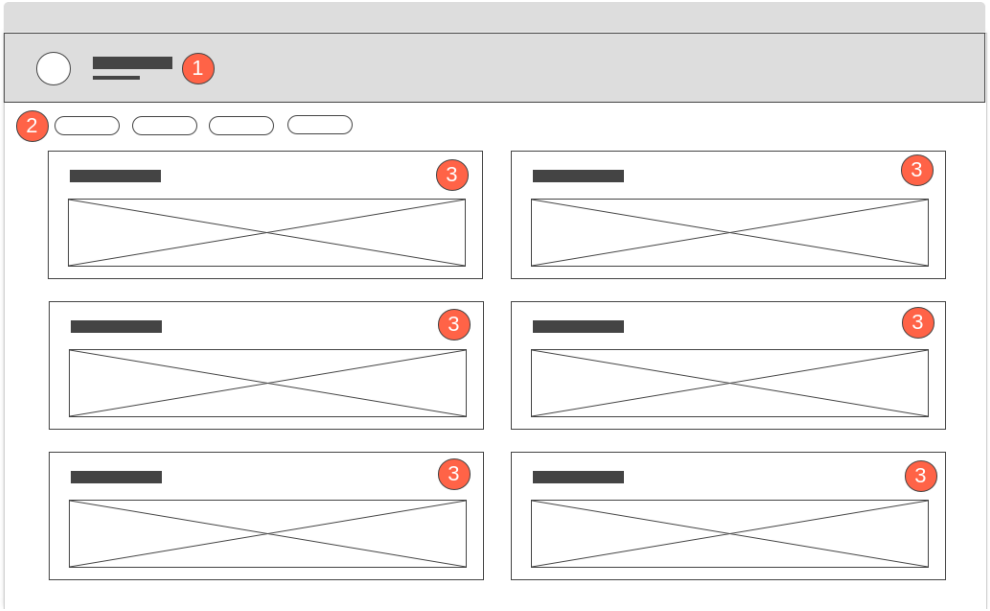

Figure 4.9: History detail – wireframe

1) **Information bar** – displays unit user name and unit number.
2) **Physiological details** – show the unit user's physiological profile.
3) **Offline graphs** – visualize the data trends in the specified time window for physical activity, acceleration, heart rate, humidity, skin and environment temperature**.**

Map – visualizes online location data.

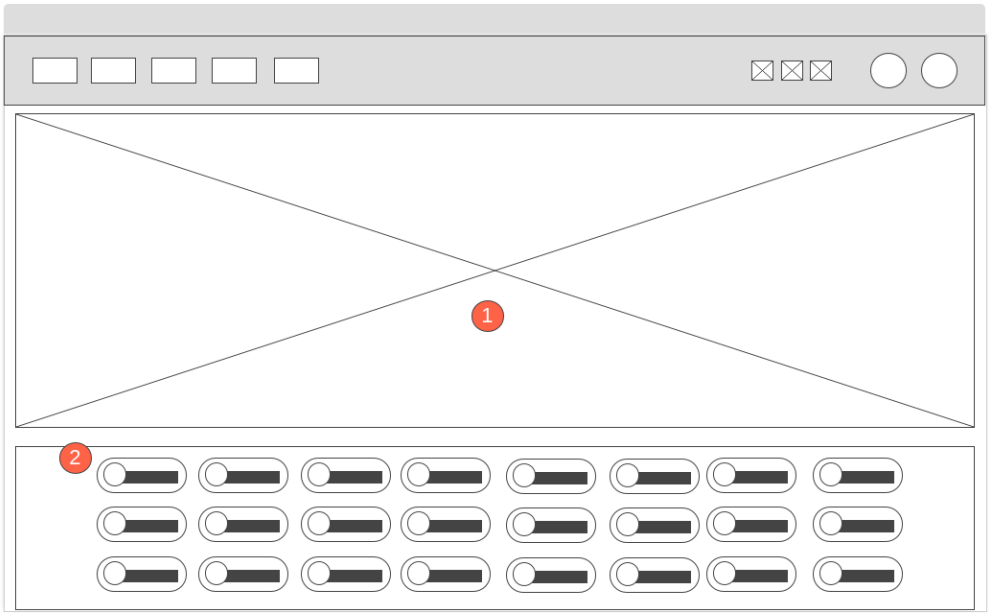

Figure 4.10: Map – wireframe

1) **Interactive map** – displays an interactive map with markers and details about each unit.
2) **User chips** – presents the unit users (name, unit number) and basic color visualization of location detection and incident alarm. Clicking on the chip focuses the map on the selected unit.

Teams – allows for the management of monitored teams and their members.

Figure 4.11: Teams – wireframe

1) **List of teams** – shows all teams and their detailed information.
2) **List of users** – shows all unit users and their detailed information.
3) **Search bar** – allows searching through the list by all the detailed information.
4) **Team select button** – selects the active team for online visualization and data storing (performance calculations, unit user details)

Settings – changes the settings for data storage and visualization.

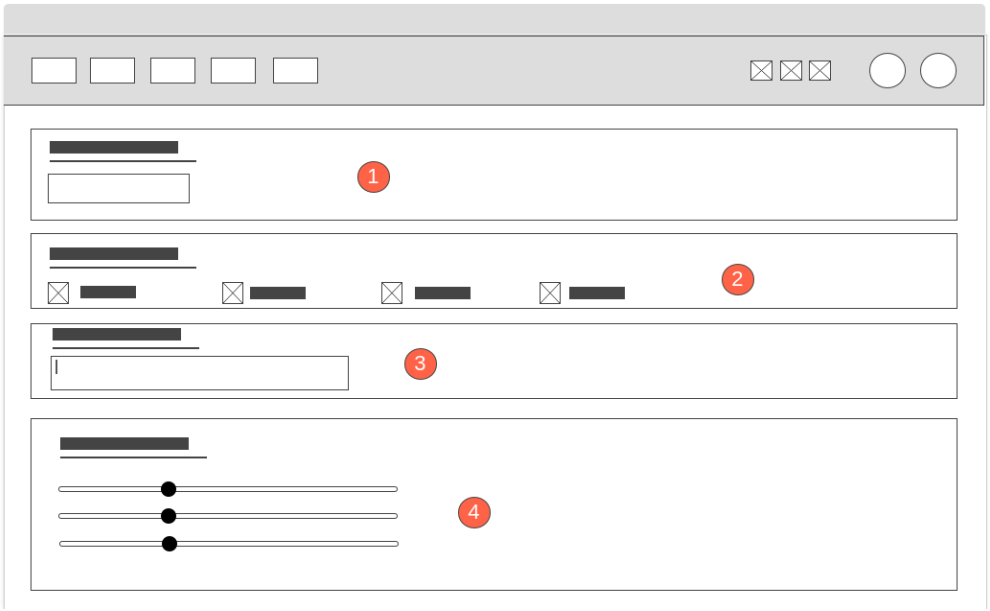

Figure 4.12: Settings – wireframe

1) **CSV directory selector** – selects the default directory for the automatic creation of CSV record files.
2) **CSV components checkboxes** – choose which data sets to store in the CSV record files (basic data, location data, performance data, node data)
3) **Events input** – set the events that can be logged into the CSV file.
4) **Range sliders** – adjust ranges for visualization via meters and graphs.

4.4.2 Component-based GUI

To accommodate the requirements of high modularity and extensibility, the application's user interface is built on the component-based paradigm. The paradigm is specific by constructing systems from highly-independent and reusable components. The fundamental principles are:

- **Separation of concerns** – each level of the system is composed of loosely-coupled components that are self-contained. Consequently, this approach enhances software development in terms of flexibility, maintainability, and reusability.
- **Single responsibility** – every component should execute only 1 key responsibility, but reliably and effectively [14].

The application's user interface incorporates the component-based paradigm with the React.js GUI JavaScript library. It is an open-source library maintained by Facebook and it is the pioneer in component-based GUI web development [15]. It was chosen for its

large developer community, extensive support, and rich ecosystem. The alternative solutions are Vue.js, AngularJS, or Ember.

As React.js uses the component-based paradigm, the GUI is built from components that are organized in a tree structure. The components can store data and pass it to other components as properties (React Props). Moreover, the components are persisted in virtual DOM that abstracts manual updates, event handling, and attribute handling within DOM. The React.js components are implemented with JSX which is a JavaScript's syntax extension [15].

The application's component tree branches out from the container components which represent the main user sections: Online, History, Map, Teams, and Settings.

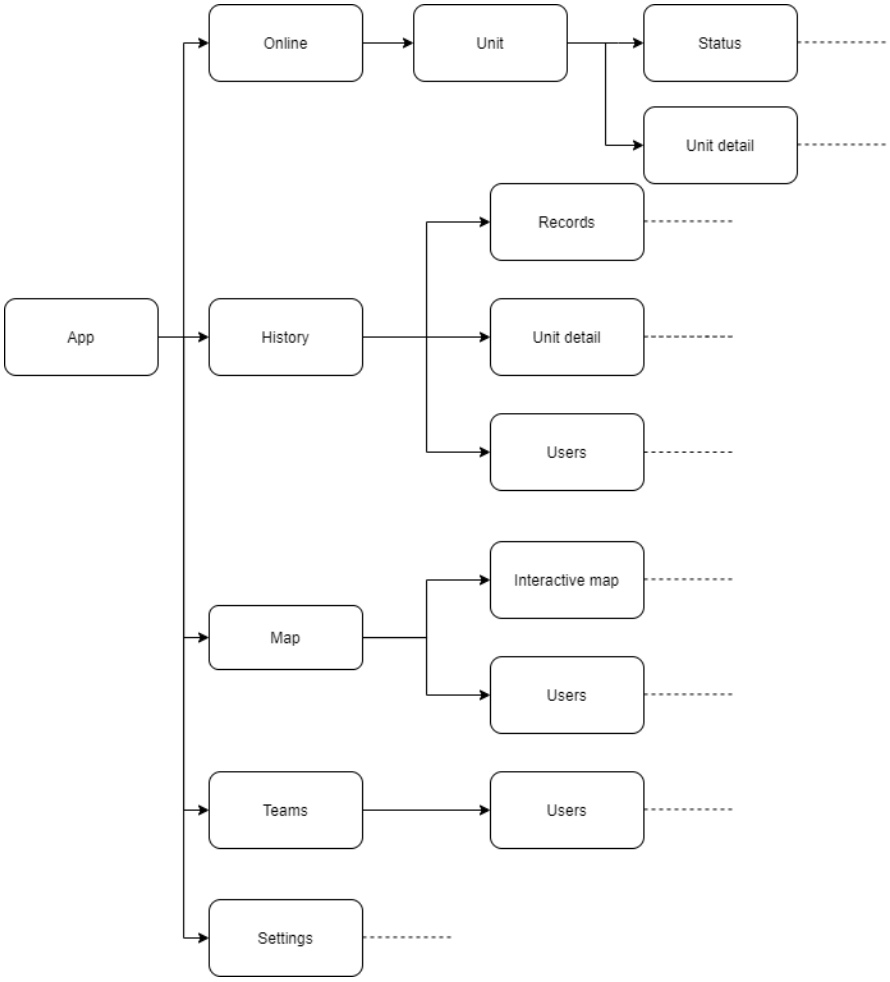

Figure 4.13: Part of the component tree (simplified)

5 Implementation

This chapter describes the implementation of key application parts following the designed architecture. The source code is distributable under the MIT license [16].

5.1 User interface

The designed user interface was built with the React.js GUI library and its additional modules.

5.1.1 Components

The application's GUI is assembled from a tree of React.js components. All types of React.js components were adopted:

- **Classes** – the components are written as JavaScript classes and they can store data which means they are stateful.
- **Functions** – serve for the creation of simpler components without data storage (stateless components). They are integrated as JavaScript functions.
- **React Hooks** – allow usage of the functional structure but can retain data at the same time (stateful components) [15].

Classes and React Hooks also include React lifecycle methods for resource management which track if the component is mounted to the DOM [15].

```
// Menu for user - syncing devices or resetting receivers
export default function ResetMenu() {
    const [anchorEl, setAnchorEl] = React.useState(null);

    const handleClick = event => {
        setAnchorEl(event.currentTarget);
    };

    const handleClose = (event) => {
        setAnchorEl(null);

        // Sends the signal for device sync
        if(event === 'devices') ipcRenderer.send("sync-devices");
        // Reconnects all receivers
        if(event === 'receivers') ipcRenderer.send("connect-ports");
    };

    return (
        <div>
            <Fab onClick={handleClick} size="small" aria-label="reset" >
                <AutorenewIcon />
            </Fab>
            <Menu
                id="reset-menu"
                anchorEl={anchorEl}
                keepMounted
                open={Boolean(anchorEl)}
                onClose={handleClose}
            >
                <MenuItem onClick={() => handleClose('devices')}>Sync devices</MenuItem>
                <MenuItem onClick={() => handleClose('receivers')}>Find receivers</MenuItem>
            </Menu>
        </div>
    )
}
```

Figure 5.1: Reset menu implemented as React Hook

5.1.2 Tools

To streamline the implementation of the GUI, proven open-source React.js modules were put to use. Consequently, the development of advanced features was swifter and more robust.

Material-UI

The wireframed design was implemented using Material Design. It is a complex GUI design system with guidelines and predefined component designs made by Google [17]. As a result, cohesive and consistent user experience can be delivered throughout the whole application. To fulfill the Material Design principles, the Material-UI React.js library was used [18].

Material-table

Various parts of the GUI display and work with tabular data. The data tables were created using the Material-table library which offers a rich feature set for tabular data management. At the same time, it is flexible as it allows for the addition of custom features. Also, its design conforms with Material Design [19].

React-Leaflet

The map component was created using the React-Leaflet library [20] which is a React.js wrapper for the Leaflet mapping solution [21]. The result is an interactive Leaflet map component with Open Street Map as a tile source.

React-vis

The graphs for both online and offline visualization were made using the React-vis charting library which is maintained by Uber. Its performance and wide options were suitable for defined use cases [22].

5.2 Database

The NoSQL database was implemented using the Lowdb package. It is a compact embedded JSON database with simple API developed specifically for Node.js and Electron applications. The data are stored in a single human-readable JSON file that is initialized in the application's system folder [23]. This solution brings straightforward portability, flexibility, and accessibility.

5.3 Data processing

The modular architecture of data processing (as described in 4.3) was realized by separating concerns into independent JavaScript classes. This enables added flexibility in swapping any data processing component and the software libraries within them.

5.3.1 Hardware communication

The data are transferred from the FlexiGuard communication module via a virtual serial port (COM). The basic serial port communication is based on the Node.js library Node-serialport which extends the state machine (described in 4.3.1) [24]. The state machine is implemented as a set of recursive functions.

```
// Returns parser for selected port
    async getParser() {

        // Configuring parser for reading from port
        this.port = new SerialPort(this.com, {
            baudRate: 115200,
            autoOpen: false
        });

        const parser = this.port.pipe(new Delimiter({ delimiter: [0] }));

        return await parser;

    }
```

Figure 5.2: Function for initialization of serial port

5.3.2 Data parsing module

The output of the data parsing is a nested JSON object with categorized data that are converted from hexadecimal format using a series of parsing functions. Furthermore, each data category is parsed by a separate function. Consequently, the data which are handed for further processing or writing to CSV files can be simply configured and modified. Also, the introduction of new data categories is convenient.

```
function parseBasicData(rawBasicArray) {

    const basicJSON = {
        timestamp: Date.now(),
        devId: rawBasicArray[0],
        heartRate: parseInt(rawBasicArray[1]),
        tempSkin: convertTemprature(rawBasicArray[2], rawBasicArray[3]),
        tempCloth: convertTemprature(rawBasicArray[4], rawBasicArray[5]),
        humidity: convertHumidity(rawBasicArray[6]),
        accX: convertAcceleration(rawBasicArray[13]),
        accY: convertAcceleration(rawBasicArray[14]),
        accZ: convertAcceleration(rawBasicArray[15]),
        batteryVoltage: parseEZ14(rawBasicArray[18], rawBasicArray[19]),
        batteryPercentage: convertBatteryVoltage(parseEZ14(rawBasicArray[18], rawBasicArray[19]))
    }

    return basicJSON;
}
```

Figure 5.3: Function for parsing basic dataset

5.3.3 Data manipulation

The data manipulation module is elemental for online visualization as it stores and organizes the online data. In practice, it delivers state management for the GUI components as it directly sends the data to the Renderer process (described in 4.1.2).

```
// Sending the packet to the Renderer
    sendData(devId) {

        const devSlot = devId - 1;

        const packet = this.packets[devSlot];
        const activityGraph = this.activityGraphs[devSlot];
        const heartRateGraph = this.heartRateGraphs[devSlot];
        const tempSkinGraph = this.tempSkinGraphs[devSlot];
        const accXGraph = this.accXGraphs[devSlot];
        const accYGraph = this.accYGraphs[devSlot];
        const accZGraph = this.accZGraphs[devSlot];

        const data = {
            packet,
            activityGraph,
            heartRateGraph,
            tempSkinGraph,
            accXGraph,
            accYGraph,
            accZGraph
        };

        this.event.sender.send(devId.toString(), data);
    }
```

Figure 5.4: Function for sending specific unit data to Renderer

5.3.4 Database interface

Although the implemented Lowdb database is conceptually simple, it offers a powerful API for comfortable data queries [23]. The database interface initializes the database and executes predefined data queries that are accessible as class methods. Moreover, the ShortId library is used for the generation of unique database document keys [25].

```
addRecord(start, path) {

        let record = {
            id: shortid.generate(),
            path,
            start,
            end: null,
            team: this.getSelectedTeam(),
            note: null
        }

        this.db.get('records')
            .push(record)
            .write()

        return record.id;
    }
```

Figure 5.5: Function for adding CSV record information to database

5.3.5 CSV interface

The main function of the CSV interface is to write and read CSV files in a performant way as the monitoring session records' size can be extensive (up to GBs). The CSV input/output operations are executed as Node.js Streams. As a result, the data are handled sequentially and not all at once in the memory, enabling faster work with large datasets [26]. The general CSV formatting is abstracted via Csv-write-stream [27] and Csv-parser [28] libraries.

With extensive datasets, graph visualization can become ineffective as an excessive amount of data points is being rendered. Therefore, additional polyline simplification of the dataset is performed by the Simplify.js library. It removes excess data points while retaining the shape of the polyline [29]. At the moment, only part of the stored data are visualized, but the functions are already capable of processing all measured parameters into graphs.

```
createCsvWriter(recordingStart) {

        const date = recordingStart.toISOString().split("T");
        const time = recordingStart.toTimeString().split(":");

        const fileName = date[0] + "-" + time[0] + time[1] + ".csv";
        this.filePath = path.join(this.directory, fileName);
        this.writer = csvWriter({
            separator: ';',
            newline: '\n',
            headers: this.createHeaders()
        })

        this.writer.sendHeaders = fs.existsSync(this.filePath) ? false : true;

        this.writer.pipe(fs.createWriteStream(this.filePath, { flags: 'a' }));
    }
```

Figure 5.6: Function for initialization of CSV writer

5.4 Code versioning

The Git versioning system was used for source code management. Currently, it is the most frequently used versioning system with many capabilities that make development faster and more comfortable. The key feature is the ability to version code into different branches and later merging them [30]. The app's Git repository is hosted on the GitHub service [31]. Furthermore, it supports direct integration into the majority of code editors and integrated development environments.

5.5 Deployment

The application's deployment into the production environment includes compiling, minifying, building, and packaging the source code into the installation package. The final deployment output for the FlexiGuard software is a set of installation packages for required platforms – macOS, Windows, and GNU/Linux. The deployment firstly requires the building of the React application which is later packaged into the Electron environment. The Npm development tool was utilized for Node.js package management and running the development scripts [32].

5.5.1 React

The React environment in the application was set up from Create-react-app software boilerplate. It includes a pre-configured deployment environment for building React apps with the usage of Webpack packaging library [33]. After running the building script, a build folder is added with the production version of the application.

5.5.2 Electron

The Electron environment was built on top of the software boilerplate Electron-quick-start which provides the minimal configuration of the Electron app [34]. For the building and distribution of the installation packages, the Electron-builder package was used. The Electron-builder contains an advanced packaging feature set with wide support of target formats [35]. After packaging, the installation packages in the chosen target formats are added to a distribution folder.

These installation packages were built into these target formats:

- **NSIS** – default open-source installer for Windows. Administrator privileges are required to operate with CSV files outside of the application's root folder scope.
- **AppImage** – the widely supported format for GNU/Linux distributions.
- **Pkg** – standard format for macOS platforms.

6 Testing

The application's capabilities were tested using various approaches focusing on specific functionalities. The tests were executed continuously to eliminate convoluted debugging. The identified bugs were gradually removed and no major errors were found. The packaged applications were successfully run and evaluated on these platforms:

- Windows 10
- macOS Catalina 10.15.1
- Ubuntu 20.04 LTS (GNU/Linux)
- Open Suse Leap 15.1 (GNU/Linux)

6.1.1 Integration testing

Integration tests predominantly focus on correct communication and cooperation of components that are critical in module-oriented architecture. The tests were conducted continuously and manually by the author using both mock and real data of CSV files, embedded database, and online visualization.

The use cases could be correctly executed and the behavior of the components conformed with the application design.

6.1.2 End-to-end testing

The most suitable technique for testing the application within the whole FlexiGuard system was end-to-end testing. It checks the functionality throughout the whole system which meant including the FlexiGuard communication modules and personal monitoring units. The reliable cooperation of these system parts is integral, therefore these aspects were thoroughly assessed:

- **Communication modules** – the autonomous detection and management of the communication modules were tested.
- **Alarm** – the registration of the incident alarm and turning it off by the supervisor.
- **Synchronization** – correct broadcast window synchronization of personal monitoring units, when induced by invalid packet detection or the supervisor.

The functionality was tested with various models of communication modules and personal monitoring units. The interaction with the hardware was standard and consistent.

6.1.3 Viewport testing

The GUI was implemented as responsive which means the layout is being dynamically adjusted according to the window size and resolution. This ensures simple readability and comprehensibility on a variety of devices. However, the online collective user view

requires that all components are visible without user input (scrolling, resizing, etc.) to visualize complete critical data at all times. These parts of the GUI were created as adaptive and change the components' size. Therefore, readability and comprehensibility tests were executed for standard desktop resolutions:

- 1280 × 720
- 1366 × 768
- 1920 × 1080

The interfaces were tested using the Chrome DevTools[3] and all components demonstrated clear readability without any deformations or abnormalities.

[3] Web development tools integrated in the Google Chrome

7 Discussion

This thesis proves that a multiplatform framework can be effectively used to develop a stable fully-fledged desktop application that can serve such critical functionality as the FlexiGuard system. At the same time, it demonstrates the transferability of the original FlexiGuard application's concepts that are implemented in the new version with evolutionary improvements and modifications.

However, a notable improvement that can be soon achieved is the performance optimization of the Main process with asynchronous operations in the data parsing module. The synchronous complex data parsing operations are restricting the completely smooth GUI rendering as JavaScript is based on the concept of the event loop. This can be eliminated for example with the open-source Electron-remote library [36]. Moreover, as the states and data flow between GUI components might get more complicated with deeper component nesting and new features, implementation of a React state management library like Redux is recommended [37]. Also, a user manual is not yet available as the software GUI changes are expected upon further feedback during a gradual transition from the original software. In the context of future application updates, the automatic update feature of Electron can be leveraged to deliver the latest software versions and seamlessly improve the user experience.

As the application design was centered around modularity, both hardware and software modifications can be easily accommodated. The author of the thesis predicts that the feature development of the software will continue in the direction of further leveraging the improved team management and physiological profiles to provide deeper insight and benchmarking in both offline and online data analyses.

8 Conclusion

The thesis is based around the development and deployment of a multiplatform software for the FlexiGuard telemonitoring system adopted mainly in military, rescue, and firefighting training.

Firstly, the analysis of the FlexiGuard system and its software was conveyed to sufficiently understand the domain-specific needs of the application. Secondly, essential input from the stakeholders was implemented into forming the requirements and the application design. As the key requirement was the support of multiple platforms, the research of available cross-platform technologies was conducted and to select the best-suited solution. Next, the application architecture was adjusted to provide reliability and flexibility which also transcended into the design of the GUI. In the end, the application was deployed for all required platforms and thoroughly tested. Also, the source code is distributed under the MIT license [16]. The end product of the thesis is a fully-capable FlexiGuard software that can be seamlessly integrated into the current FlexiGuard system.

References

[1] KLIMENT, Radim, Pavel SMRČKA, Karel HÁNA, Jakub SCHLENKER, Vladimír SOCHA, Luboš SOCHA a Patrik KUTÍLEK. Wearable Modular Telemetry System for the Integrated Rescue System Operational Use. *Journal of Sensors* [online]. 2017, **2017**, 1-12 [cit. 2020-05-20]. DOI: 10.1155/2017/9034253. ISSN 1687-725X. Available at: https://www.hindawi.com/journals/js/2017/9034253/

[2] KUTILEK, Patrik, Petr VOLF, Slavka VITECKOVA et al. Wearable systems for monitoring the health condition of soldiers: Review and application. In: *2017 International Conference on Military Technologies (ICMT)* [online]. IEEE, 2017, s. 748-752 [cit. 2020-05-20]. DOI: 10.1109/MILTECHS.2017.7988856. ISBN 978-1-5090-5666-8. Available at: http://ieeexplore.ieee.org/document/7988856/

[3] RYŠLAVÝ, Erik. *Measuring Options and Evaluation of Physiological Parameters of Integrated Rescue System Unit Employees*. Kladno, 2018.. Bachelor thesis. Czech Technical University in Prague.

[4] SMRČKA, Pavel. *FlexiGuard verze 2.2: Popis a souhrn parametrů*. Prague, 2019.. Czech Technical University in Prague.

[5] *Mediatéka* [online]. Prague: CTU in Prague, Computing and Information Centre, 2018 [cit. 2020-05-20]. Available at: www.media.cvut.cz/cs/foto/20180523-projekt-flexiguard-fbmicvut

[6] SCHLENKER, Jakub, Vladimir SOCHA, Pavel SMRCKA et al. FlexiGuard: Modular biotelemetry system for military applications. In: *International Conference on Military Technologies (ICMT) 2015* [online]. IEEE, 2015, s. 1-6 [cit. 2020-05-20]. DOI: 10.1109/MILTECHS.2015.7153712. ISBN 978-8-0723-1977-0. Available at: http://ieeexplore.ieee.org/document/7153712/

[7] *XPDA: Cross-Platform Desktop Apps* [online]. XPDA, 2020 [cit. 2020-05-19]. Available at: https://xpda.net/

[8] *Electron: Build cross-platform desktop apps with JavaScript, HTML, and CSS* [online]. San Francisco: GitHub, 2020 [cit. 2020-05-19]. Available at: https://www.electronjs.org/

[9] *NW.js* [online]. NW.js Community, 2020 [cit. 2020-05-19]. Available at: https://nwjs.io/

[10] *Qt: Cross-platform software development for embedded and desktop* [online]. Helsinki: The Qt Group, 2020 [cit. 2020-05-19]. Available at: https://www.qt.io/

[11] NOKES, Cameron. Deep dive into Electron's main and renderer processes. *Cameron Nokes* [online]. Provo: Cameron Nokes, 2016 [cit. 2020-05-19]. Available at: https://cameronnokes.com/blog/deep-dive-into-electron's-main-and-renderer-processes/

[12] *NoSQL Databases Explained* [online]. New York: MongoDB, 2020 [cit. 2020-05-19]. Available at: https://www.mongodb.com/nosql-explained

[13] *Wireframe CC: A design tool fine-tuned for wireframing* [online]. Panama: Wireframe CC, 2020 [cit. 2020-05-19]. Available at: https://wireframe.cc/

[14] MRÁZ, Marcel. *Component-based UI Web Development*. Brno, 2019.. Bachelor thesis. Masaryk University in Brno.

[15] *React: A JavaScript library for building user interfaces* [online]. San Francisco: Facebook, 2020 [cit. 2020-05-19]. Available at: https://reactjs.org/

[16] *The MIT License* [online]. Palo Alto: Open Source Initiative, 2020 [cit. 2020-05-20]. Available at: https://opensource.org/licenses/MIT

[17] *Material Design: Create intuitive and beautiful products with Material Design* [online]. Mountain View: Google, 2020 [cit. 2020-05-19]. Available at: https://material.io/design

[18] *Material-UI: React components for faster and easier web development.* [online]. Paris: Material-UI, 2020 [cit. 2020-05-19]. Available at: https://material-ui.com/

[19] *Material-table: React data table component that based on material-ui* [online]. Istanbul: Mehmet Baran, 2020 [cit. 2020-05-19]. Available at: https://material-table.com

[20] *React-Leaflet: React components for Leaflet maps* [online]. London: Paul Le Cam, 2020 [cit. 2020-05-19]. Available at: https://react-leaflet.js.org/

[21] *Leaflet: An open-source JavaScript library for mobile-friendly interactive maps* [online]. Kyiv: Vladimir Agafonkin, 2019 [cit. 2020-05-19]. Available at: https://leafletjs.com/

[22] *React-vis: A composable charting library* [online]. San Francisco: Uber, 2020 [cit. 2020-05-19]. Available at: https://uber.github.io/react-vis/

[23] *Lowdb: Small JSON database for Node, Electron and the browser* [online]. Typicode, 2020 [cit. 2020-05-19]. Available at: https://github.com/typicode/lowdb

[24] *Node Serialport* [online]. New York: Node SerialPort, 2020 [cit. 2020-05-19]. Available at: https://serialport.io/

[25] *ShortId: Amazingly short non-sequential url-friendly unique id generator* [online]. Arlington: Dylan Greene, 2020 [cit. 2020-05-19]. Available at: https://github.com/dylang/shortid

[26] *Stream: Node.js v14.2.0 Documentation* [online]. San Francisco: OpenJS Foundation, 2020 [cit. 2020-05-19]. Available at: https://nodejs.org/api/stream.html

[27] *Csv-write-stream* [online]. Beverly Hills: Max Ogden, 2020 [cit. 2020-05-19]. Available at: https://github.com/maxogden/csv-write-stream

[28] *Csv-parser* [online]. Copenhagen: Mathias Buus, 2020 [cit. 2020-05-19]. Available at: https://github.com/mafintosh/csv-parser

[29] *Simplify.js: Tiny high-performance JavaScript polyline simplification library* [online]. Kyiv: Vladimir Agafonkin, 2013 [cit. 2020-05-20]. Available at: http://mourner.github.io/simplify-js/

[30] *Git* [online]. New York: Git Project, 2020 [cit. 2020-05-19]. Available at: https://git-scm.com/

[31] *GitHub* [online]. San Francisco: GitHub, 2020 [cit. 2020-05-20]. Available at: https://github.com/

[32] *Npm: Build amazing things* [online]. Oakland: Npm, 2020 [cit. 2020-05-20]. Available at: https://www.npmjs.com/

[33] *Create React App: Set up a modern web app by running one command* [online]. Menlo Park: Facebook, 2020 [cit. 2020-05-19]. Available at: https://create-react-app.dev/

[34] *Electron-quick-start* [online]. San Francisco: Electron, 2020 [cit. 2020-05-19]. Available at: https://github.com/electron/electron-quick-start

[35] *Electron-builder* [online]. Develar, 2020 [cit. 2020-05-19]. Available at: https://www.electron.build/

[36] *Electron-remote* [online]. Berkeley: Electron Userland, 2019 [cit. 2020-05-20]. Available at: https://github.com/electron-userland/electron-remote

[37] *Redux: A Predictable State Container for JS Apps* [online]. Menlo Park: Dan Abramov, 2020 [cit. 2020-05-19]. Available at: https://redux.js.org/

List of figures

List of tables

Attachment A: Implemented GUI

The designed GUI (described in 4.4.1) was implemented using the React.js GUI library and its add-ons (described in 5.1)

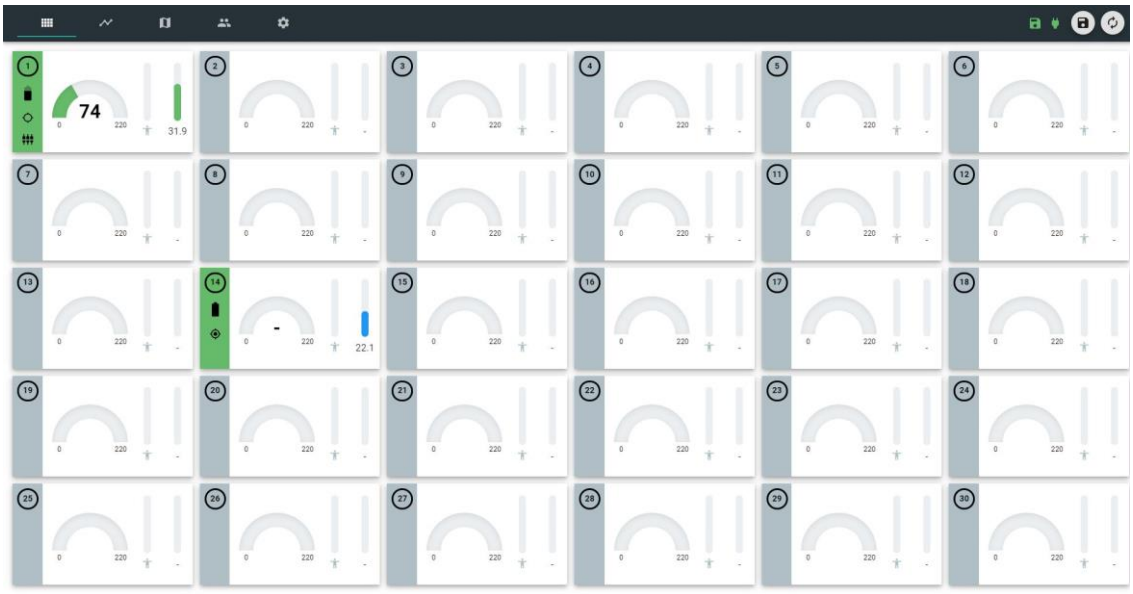

Figure A.1: Online overview – implemented GUI

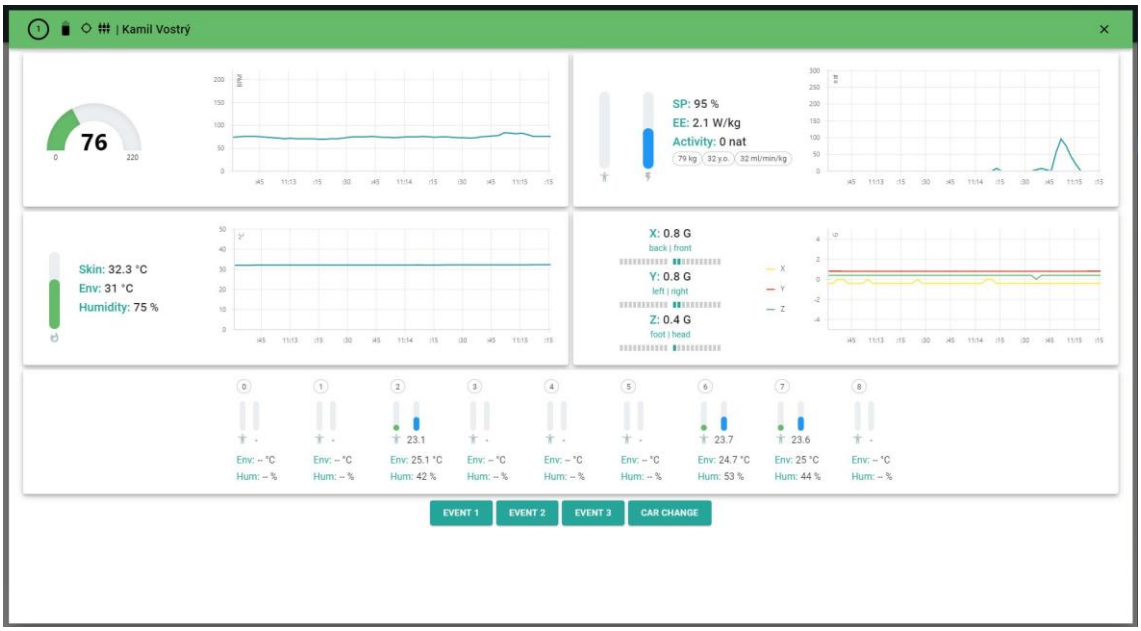

Figure A.2: Online detail – implemented GUI

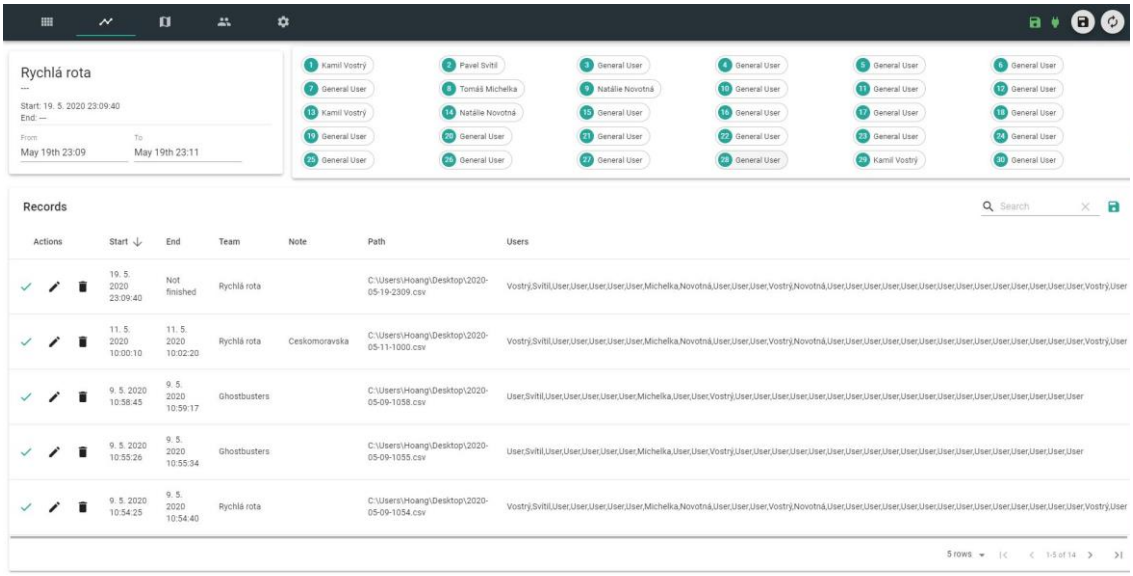

Figure A.3: History overview – implemented GUI

Figure A.4: History detail – implemented GUI

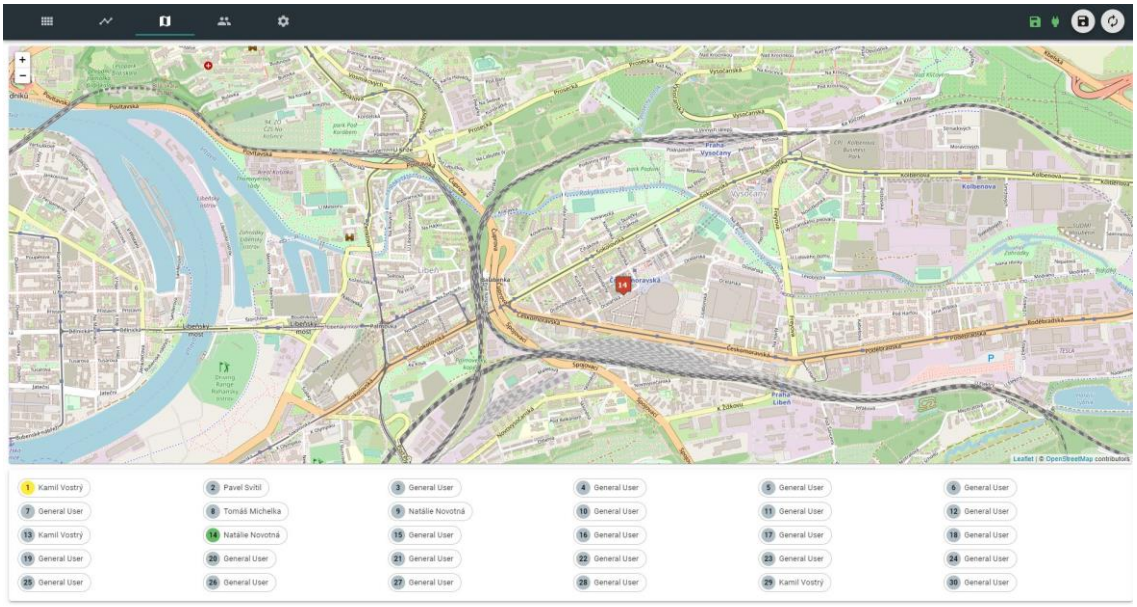

Figure A.5: Map – implemented GUI

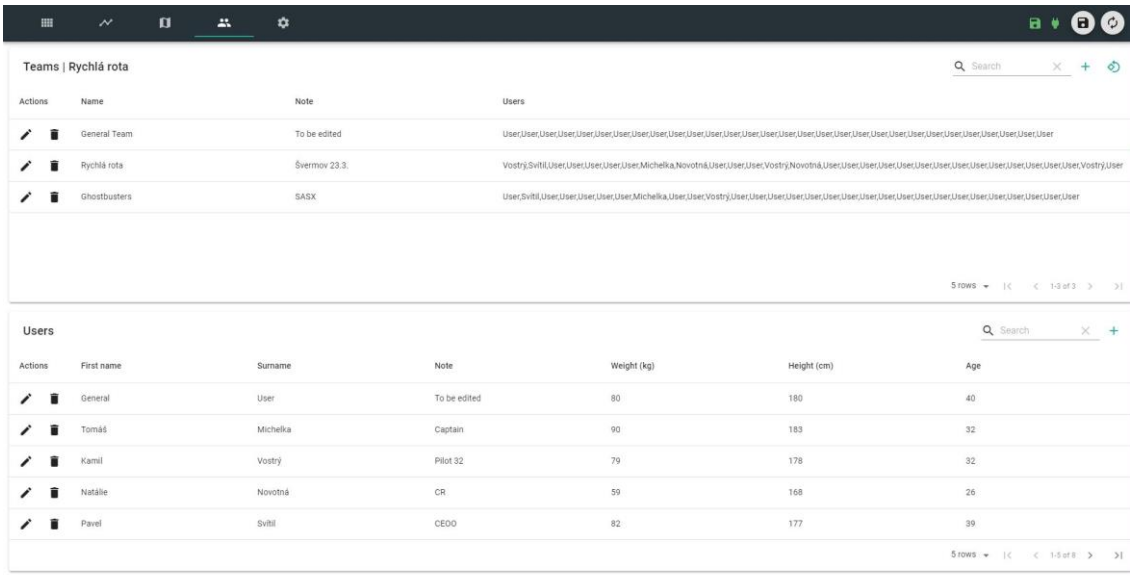

Figure A.6: Teams – implemented GUI

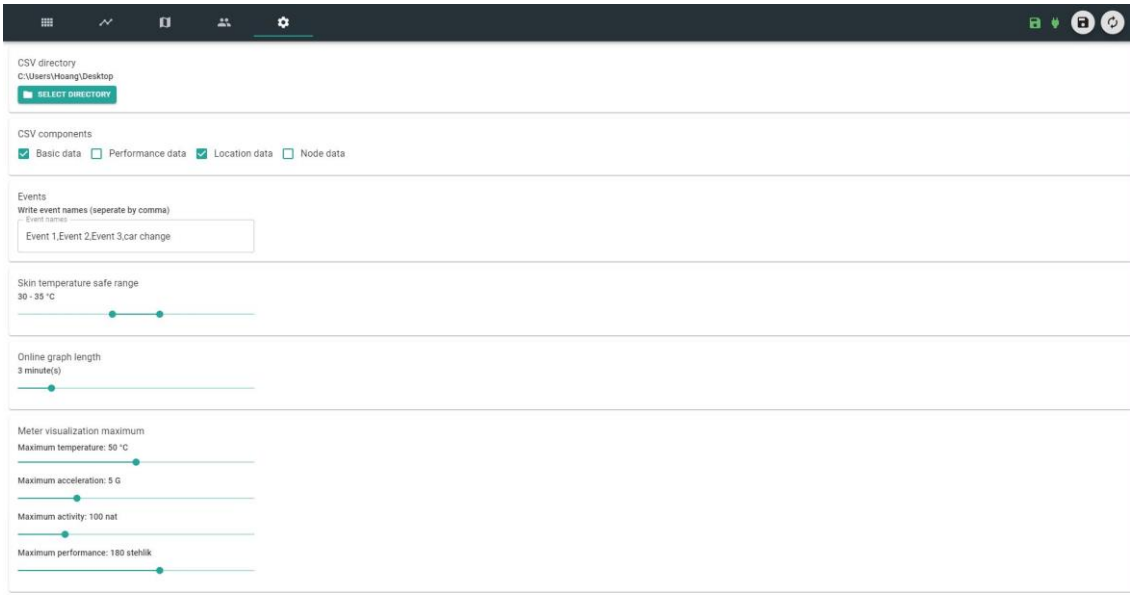

Figure A.7: Settings – implemented GUI

Attachment B: Development & packaging manual

This manual instructs on how to run the application in the development environment and how to build an installation package with technologies described in 5.5. At the moment, the target platform installation packages can be built only from selected origin platforms:

- **Windows**[4] – buildable from macOS, GNU/Linux, Windows
- **macOS** – buildable only from macOS
- **GNU/Linux** – buildable from macOS, GNU/Linux

Follow these steps to run the application in the development environment:

1. Install the latest Node.js available[5].
2. Transfer the source code folder to your computer from the enclosed attachment.
3. Navigate to the source code directory in the command line.
4. Run ***npm install*** command to install all dependencies.
5. Run ***npm run start*** command to start a development Node.js server.
6. Run ***npm run electron*** command in a new command line to start the application (in the source code directory).

Continue with these steps, if you would like to build an installation package:

7. Run ***npm run build*** command to build the React app.
8. Run ***npm run dist -wml*** [6] command to build a packaged installation file.
9. Run the packaged installation file in the ***dist*** directory within the source code directory to install the application.

[4] Already built Windows installator is on the enclosed CD (Attachment C: Content of the enclosed CD)

[5] Available at: https://nodejs.org/en/

[6] According to targeted platforms (Windows: -w, Linux: -l, macOS: -m, all: -wml)

Attachment C: Content of the enclosed CD

Abstract_CZ.pdf	Thesis abstract in Czech
Abstract_ENG.pdf	Thesis abstract in English
Assignment.pdf	Thesis assignment in Czech
CSV_sample.csv	Sample monitoring record
DB_sample.json	Sample database file
FlexiGuard_src.zip	FlexiGuard software source code
FlexiGuard_win_setup.zip	FlexiGuard installer for Windows (x64)
Keywords.pdf	Thesis keywords in English and Czech
Thesis.pdf	Complete thesis