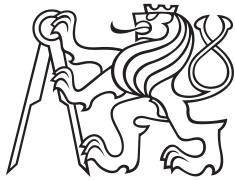


Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Laboratory IS for Measuring Field Biochemical Data of Athletes

Bc. Lukáš Šimon

Supervisor: Ing. Ivo Malý, Ph.D.
Field of study: Open Informatics
Subfield: Software Engineering
January 2024

I. Personal and study details

Student's name: **Šimon Lukáš** Personal ID number: **483810**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Computer Science**
Study program: **Open Informatics**
Specialisation: **Software Engineering**

II. Master's thesis details

Master's thesis title in English:

Laboratory IS for measuring biochemical data of athletes in the field

Master's thesis title in Czech:

Laboratorní IS pro měření terénních biochemických dat sportovců

Guidelines:

Analyze the measurements of biochemical and physiological data outside the laboratory conditions. Focus on the sample collection, sample analysis and transfer of results to athletes and coaches.
Based on the analysis, design a desktop application for direct data collection from measuring devices, an application for visualization and management of the data from measuring devices used by the laboratory technicians, and an application for data visualization by the athletes and coaches.
Implement the proposed applications using appropriate multi-platform development tools.
Test the resulting applications on real-life data from the laboratory.

Bibliography / sources:

Axelson, Jan. Serial Port Complete: The Developer's Guide. Lakeview Research LLC, 2007.
Goodman, Elizabeth, and Mike Kuniavsky. Observing the user experience: A practitioner's guide to user research. Elsevier, 2012.
Flutter Documentation, <https://flutter.dev/>.

Name and workplace of master's thesis supervisor:

Ing. Ivo Malý, Ph.D. Department of Computer Graphics and Interaction FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **14.08.2023** Deadline for master's thesis submission: **09.01.2024**

Assignment valid until: **16.02.2025**

Ing. Ivo Malý, Ph.D.
Supervisor's signature

Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Acknowledgements

I would like to hereby thank Ing. Ivo Malý, Ph.D. for his professional supervision and guidance throughout work on this thesis. I would also like to express my gratitude towards my family, friends and my beloved girlfriend for their unending support.

Declaration

I declare that I have prepared the submitted thesis independently and that I have listed all the information sources used in accordance with the Methodological Guidelines on the observance of ethical principles in the preparation of university final theses.

In Prague, 7. January 2024

Abstract

In the world of competitive sportsmanship, athletes are required to perform at the edge of what their body is capable of. Monitoring and assessing specific biomarkers can give insights into the appropriateness of the training load and possibly aid the adjustment of the preparation. To automate and streamline the currently manual process, an information system consisting of a mobile and desktop application is introduced. The situation is thoroughly analyzed to find flaws that may be fixed and sections of the process that could be optimized. The whole system is designed and implemented using state-of-the-art technologies such as Flutter for performance and multiplatform interfaces and Supabase for real-time data storage. Finally, after iterative development, the solution is adequately tested. The interface was designed with minimalism in mind and to be as intuitive as possible.

Keywords: Flutter, Mobile applications, Desktop applications, Supabase, Dart

Supervisor: Ing. Ivo Malý, Ph.D.
Praha 2,
Karlovo náměstí 13,
E-418

Abstrakt

Ve světě kompetitivního sportu jsou atleti nuceni k výkonu na hranici jejich fyzických schopností. Monitorování a hodnocení specifických biomarkerů může poskytnout vhled do vhodnosti tréninkové zátěže a případné pomoci při úpravě jejich přípravy. Pro automatizaci a zefektivnění aktuálně manuálního procesu je zaveden informační systém skládající se z mobilní a desktopové aplikace. Situace je důkladně zanalyzována k nalezení nedostatků, které by mohli být opraveny, a částí procesu, které by mohly být optimalizovány. Celý systém je navržen a implementován s použitím moderních technologií jako je Flutter pro jeho výkon a multiplatformní možnosti a Supabase pro ukládání dat v reálném čase. Nakonec, po iterativním vývoji, je řešení adekvátně otestováno. Rozhraní bylo navrženo aby bylo co nejvíce jednoduché a minimalistické.

Klíčová slova: Flutter, Mobilní aplikace, Desktopová aplikace, Supabase, Dart

Překlad názvu: Laboratorní IS pro Měření Terénních Biochemických Dat Sportovců

Contents

1 Introduction	1	6 Conclusion	53
2 Analysis	5	7 Future work	55
2.1 Current situation	5	Bibliography	57
2.2 Devices	6	A Installation process	61
2.2.1 Super GL2	7		
2.2.2 Spotchem Arkray	7		
2.2.3 Afias-1	8		
2.2.4 Opti CCA TS2	8		
2.3 User roles	10		
2.4 Requirements	10		
2.4.1 Desktop application	11		
2.4.2 Mobile application	11		
2.4.3 Server	12		
2.5 System communication	12		
3 System design	15		
3.1 System Architecture	16		
3.2 User Interface Design	18		
3.2.1 Desktop application	18		
3.2.2 Mobile Application	26		
3.3 System's user management	29		
4 Implementation	31		
4.1 Flutter	32		
4.2 Libraries	33		
4.2.1 Riverpod	33		
4.2.2 Hive	34		
4.2.3 Lottie	34		
4.2.4 Serial port	35		
4.2.5 Go Router	35		
4.2.6 Bitsdojo window	35		
4.3 Application architecture	36		
4.3.1 Service	36		
4.3.2 Controller	36		
4.3.3 Repository	37		
4.3.4 User Interface	37		
4.4 Supabase	38		
5 Testing	43		
5.1 Software Testing	43		
5.1.1 Test coverage	44		
5.1.2 Found problems	44		
5.2 Expert review	45		
5.2.1 Selected Users	45		
5.2.2 Testing scenarios	46		
5.3 Testing results	49		
5.3.1 Critical errors	49		
5.3.2 Low impact errors	50		

Figures

2.1 Current situation diagram	6
2.2 Devices	9
2.3 Communication schema	12
3.1 System's Architecture	16
3.2 Proposed situation diagram	17
3.3 Dashboard navigation	19
3.4 Event card	19
3.5 Device card	20
3.6 Administrator card	21
3.7 Subject card	22
3.8 Table actions	23
3.9 Data view actions and Devices	23
3.10 Dashboard screen	24
3.11 Device control screen	24
3.12 Subject control screen	25
3.13 Administrator screen	25
3.14 Data control screen	26
3.15 Login screen and trainee screen	27
3.16 Home screen personal	28
3.17 Detail screen	28
3.18 Social login diagram [8]	29
4.1 Application's Architecture	40
4.2 Supabase parse method	40
4.3 Supabase data hierarchy	41

Tables

2.1 User roles	10
5.1 Errors found by unit testing	44
5.2 Critical errors	50



Chapter 1

Introduction

Professional athletes and soldiers are often tested to ensure their training is appropriate and their values, such as cortisol level, blood sugar, and others, are in a reasonable range. To provide those measurements and to deliver valuable feedback, Casri, a scientific and service company for physical education and sport, is working with elite sportsmen and the army to take blood samples, analyze them, and give back reports with information that may lead to adjustments in training load, their dietary regime, and other parts of their preparation that may be affected.

Currently, the collection process and the data delivery are barely automated and, if executed inaccurately, can easily introduce many errors. After taking the samples and measuring them in respective devices, the results of the measurements are written into an Excel sheet value by value by hand from either the display of the machine or from a piece of paper printed by the device. After analysis of the data by a professional technician, the feedback is sent, depending on specifications, to the sportsmen, their trainers, or both to allow them to act on it accordingly and in time. The results are transferred via an E-mail with the Excel sheet attached to it and protected only with a password, which can lead to information leaks and can be tedious. It is also worth mentioning that the Excel sheet, while being sort of an industry standard in some fields, may not be as easy to comprehend for some people.

Overall, this approach can lead to problems in several areas. Firstly, one can easily make typos by writing values manually into the Excel sheet or simply by misreading the values from the devices. Secondly, it makes the whole process unnecessarily slow and cumbersome and burdens medical professionals with repetitive tasks. Moreover, all the measuring devices expose an interface for transferring data to, for example, a laptop, thus creating a great opportunity for improvement and dramatic speedup in process execution. The interface allows for automated data retrieval and visualization, removing the need to read the values from a tiny display or paper and for manual writing, hence minimizing the chance for human errors.

The proposed system aims to use the automation opportunity and, therefore, reduce repetitive tasks and streamline the whole procedure through software. A desktop application should be developed to autonomously read the measured data from medical devices, display them to technicians for evaluation, and potentially export them into various formats to make further analysis possible in other tools made for such purposes. A mobile application for the sportsmen or soldiers and their trainers would then provide in-time insights on the results of the measuring, allowing them to cut the waiting time and potentially avoid inappropriate intensity of training.

The use of state-of-the-art technologies such as Flutter, a software development toolkit created for rapid development of multi-platform applications, and supabase, a backend as a service tool providing scalable data storage with the possibility of real-time data reads and writes, an edge functions solution and analytics console, makes the development time significantly smaller.

User interface design and user experience engineering are also two important aspects of any application development cycle; this thesis and the proposed applications are no exception, and in both, the pressure was put on making them appealing and, more importantly, highly functional while also maintaining familiarity through usage of components traditional for their platforms.

Together with the existing infrastructure, these applications can significantly improve the current situation. By lowering the time needed for collection, analysis, and delivery of measurements and their resulting feedback, as well as providing a system that is less error-prone and more secure, the solution brings a more seamless experience for all parties involved and strives to mitigate the problems.

For the solution to be considered successful, it is vital not to introduce new issues and to avoid complicating the procedure by being overly complex, difficult to understand, or unnecessarily technical. A large amount of the functionality should be effectively hidden from all the users and happen automatically to make sure there is no need for extensive training for both current and future employees, thus making the transition from the old system to the new one quicker.

This thesis covers a comprehensive analysis of the current situation and how the procedure is executed, as well as studying what devices are used, for what purpose, and what biomarkers they measure. It also goes through the roles of users to better understand their needs. This is followed by a design cycle that proposes changes that should simplify the current execution and hide some portions of it by automation. System architecture design is introduced in this part, and interface design finishes the chapter. The solution is developed based on the two previous chapters. Implementation is done using only tools that meet standards for portability, speed, and functionality.

The system is extensively tested throughout the development cycle, ensuring stability. Lastly, the work is summarized in a brief and concise conclusion.

Chapter 2

Analysis

In order to achieve professional levels of performance, athletes must engage in strict training routines that can potentially result in injuries or illnesses. Additionally, their sleeping patterns and dietary habits play a crucial role in their preparation. It is crucial to regularly assess and validate all these factors, as adjustments may be necessary to ensure optimal physical functionality.

The article *Monitoring Training Load to Understand Fatigue in Athletes* [3] states that to minimize the risk of injury, illness, and other problems related to inappropriate training load, it is vital to monitor specific biomarkers. Those may show how well an athlete is adapting to the training program and can signalize a need for a change in intensity. It also emphasizes that even though said monitoring is essential, the system for monitoring needs to be intuitive, and the reported feedback should be simple yet valid. Improving performance can be difficult for elite sportsmen. Adjustments in the frequency and duration of the training sessions are likely to occur, and they can be adjusted often, mentions another part of the article.

As described in the article *Blood-Based Biomarkers for Managing Workload in Athletes* [6], there are many biomarkers that are potentially useful for monitoring the training load, and recent developments show a shift from single biomarker measurement to multiple markers. Established biomarkers like creatine kinase or lactate can prove to be helpful and can be measured quickly. However, their usefulness is not as high as if practitioners capture them along with other markers to assess the situation more thoroughly.

2.1 Current situation

After taking the samples, biomarkers are measured by four types of devices described in section 2.2. Those devices are made by different manufacturers for different purposes and provide unique data in different formats. This makes it harder to automate the process as there is no uniformity in the data output or the data transfer. Ideally, all of that should be hidden from users of the system and form a uniform interface for more effortless work.

After receiving the data, it is written in Excel by hand and it is processed there. This can quickly lead to human errors. The analysis is also conducted using an Excel sheet and does not pose any threat from a technical standpoint; however, the data collection does. It could and should be automated in the system to remove the potential for mistakes in the manual reading and writing of the data and to speed up the whole process by a large margin. The current situation is depicted in figure 2.1.

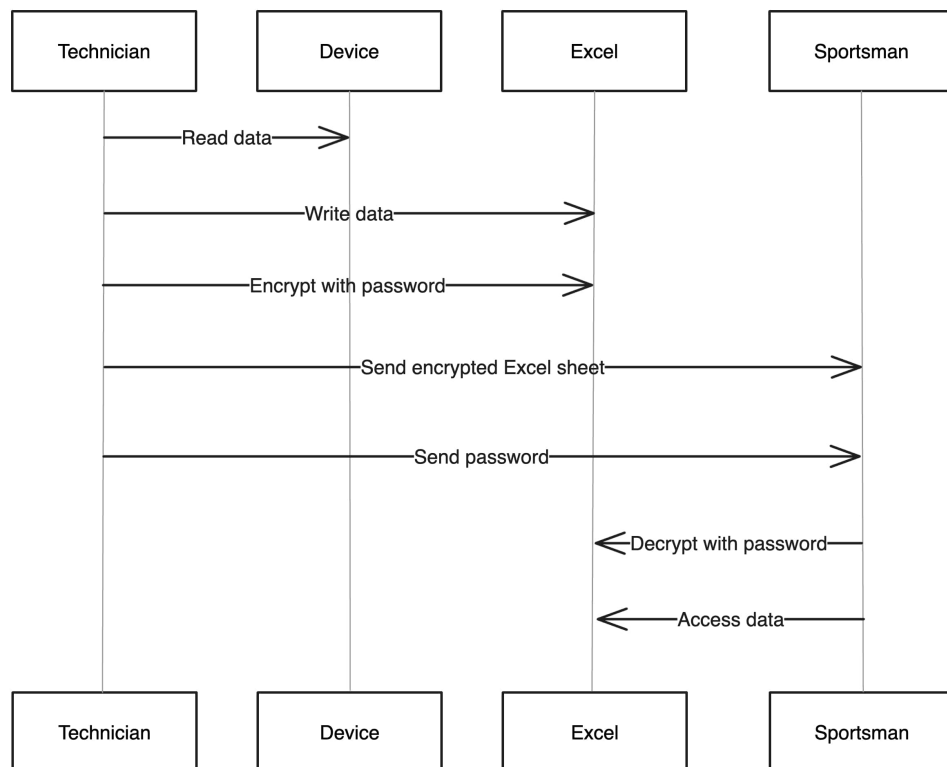


Figure 2.1: Current situation diagram

2.2 Devices

This section is dedicated to going through all four devices that are being used and brief description of them. It will also review the biomarkers that can be measured from the samples and evaluated. Some of those markers are measured, but some are calculated. This thesis does not try to explain those markers in detail as it is out of the scope. The goal is only to illustrate what values are used. The devices are also connected to the laptop differently using various cables. For this purpose, it is necessary to have a hub to connect all of them to the laptop.

■ 2.2.1 Super GL2

Is a device that allows to measure glucose, lactate, and hemoglobin simultaneously from one sample. The device is displayed in the image 2.2d.

Cable used for connection: Six pin RJ

Values being measured:

- Glucose (mmol/L)
- Lactate (mmol/L)

■ 2.2.2 Spotchem Arkray

Is an automated biochemical analyzer using a dry chemistry system depicted in an image 2.2c.

Cable used for connection: Six pin RJ

Values being measured:

- Urea (mmol/l)
- Biliruben ($\mu\text{mol/l}$)
- AST ($\mu\text{kat/l}$)
- ALT ($\mu\text{kat/l}$)
- LDH ($\mu\text{kat/l}$)
- CK ($\mu\text{kat/l}$)
- Albumin (g/l)
- T-Pro (g/l)
- Creatine ($\mu\text{mol/l}$)
- Alkaline phosphatase ($\mu\text{kat/l}$)
- Hdl cholesterol (mmol/l)
- Total cholesterol (mmol/l)
- Glucose (mmol/l)

■ 2.2.3 Afias-1

Is a compact immunoassay analyzer demonstrated in the picture 2.2a.

Cable used for connection: Mini usb

Values being measured:

- Cortisol (nmol/l)
- TSH (uIU/ml)
- CRP (mg/m)
- IL6 (pg/ml)

■ 2.2.4 Opti CCA TS2

Is Blood Gas and Electrolyte Analyzer and could be seen in a picture 2.2b.

Cable used for connection: Usb b

Values being measured:

- pH
- pCO₂ (kPa)
- PO₂ (kPa)
- BE (mmol/l)
- HCO₃ (mmol/l)
- BE_{ef} (mmol/l)
- stHCO₃ (mmol/l)
- SO₂ (%)
- Na⁺ (mmol/l)
- K⁺ (mmol/l)
- Ca⁺⁺ (mmol/l)



(a) : Afias 1 [7]



(b) : Opti CCA-TS2 [27]



(c) : SPOTCHEM EZ SP-4430 [30]



(d) : SUPER GL Compact [33]

Figure 2.2: Devices

2.3 User roles

The software solution developed will be used by users belonging to different roles. Each role is defined with its privileges related to system access as well as what part of the system it has access to. Those roles are listed below, along with their respective privileges and system usage.

Role	Part of the system used	Usage
Technician	Desktop application	Device, data, subject management, data collection, and its delivery to trainers and trainees
Trainer	Mobile application	Results and feedback viewing for events his trainees participated in.
Trainee	Mobile application	Results and feedback viewing for events he participated in.

Table 2.1: User roles

2.4 Requirements

Based on an analysis of the current situation and technical possibilities to automate the processes, the following requirements were raised for a possible automation system. This analysis considered the specific challenges and needs faced in the current workflow. The requirements have been formulated to ensure that the proposed system not only streamlines and optimizes the current processes but also introduces scalability and flexibility to adapt to future demands. Particular emphasis was also placed on the intuitiveness of the whole system. Worth noting is also pressure on the security of the solution.

Based on the analysis, there was found a need for more than one application since the users and their roles require different functionality as introduced in the table 2.1. A desktop application is meant for the data collection and management of the whole process and its parts; this portion of the solution would be used by the technicians. A mobile application is also required as having a desktop application for trainers and trainees would introduce new problems mainly related to vastly different portability for a laptop compared to a mobile phone.

For communication between the applications, a backend was needed. However, since all the logic is situated inside the application, there is no need to develop a custom-made API. For this communication, a backend as a service would be used. Data is written into it by the desktop application and is read by both mobile and desktop applications for synchronization. It is desirable to stream data changes to mobile devices automatically.

■ 2.4.1 Desktop application

- The proposed system has to read and show the results of the measurements automatically.
- The proposed system has to uniform the interfaces for each device to show the same or similar visualization based only on the data read rather than the device output.
- The proposed system has to allow for device management.
 - Add a device.
 - Edit a device.
 - Remove a device.
- The proposed system has to allow easy and intuitive subject creation and alteration of information about the measurement.
- The proposed system has to allow for a quick setup of a scenario (Event).
 - Create a group of sportsmen that will be measured in a scenario.
 - Add devices to the scenario.
 - Synchronize a scenario.
 - Connect to the assigned devices.
 - Download a copy of a scenario.
- The proposed system has to be intuitive and familiar.
- The proposed system has to assign the data automatically to the correct test subject or create a new subject if there is none to assign the data to.
- The proposed system has to allow for automated data transfer to the clients without any intervention of the technicians.
- The proposed system has to allow for the possibility of deciding to whom to send the data.
- The proposed system has to allow for simple work with the roles of the clients (Trainer, Athlete).

■ 2.4.2 Mobile application

- The proposed system has to provide a pleasant, simple, and easy-to-read visualization of the results to the clients.
- The proposed system has to allow the users to connect via social sign-ins.
- The proposed system has to make a clear visual distinction between results for the logged user and for his trainees.
- The proposed system has to distinguish between different trainees of a trainer and clearly separate their results.

2.4.3 Server

- The proposed system has to allow for communication between the mobile and desktop applications.
- The proposed system has to synchronize data changes in the mobile application automatically.

2.5 System communication

There will have to be extensive communication between parts of the system to keep data up to date and to deliver measurement insights on time. For this purpose, a communication schema depicted in figure 2.3 is developed that illustrates how the data would be flowing in the system.

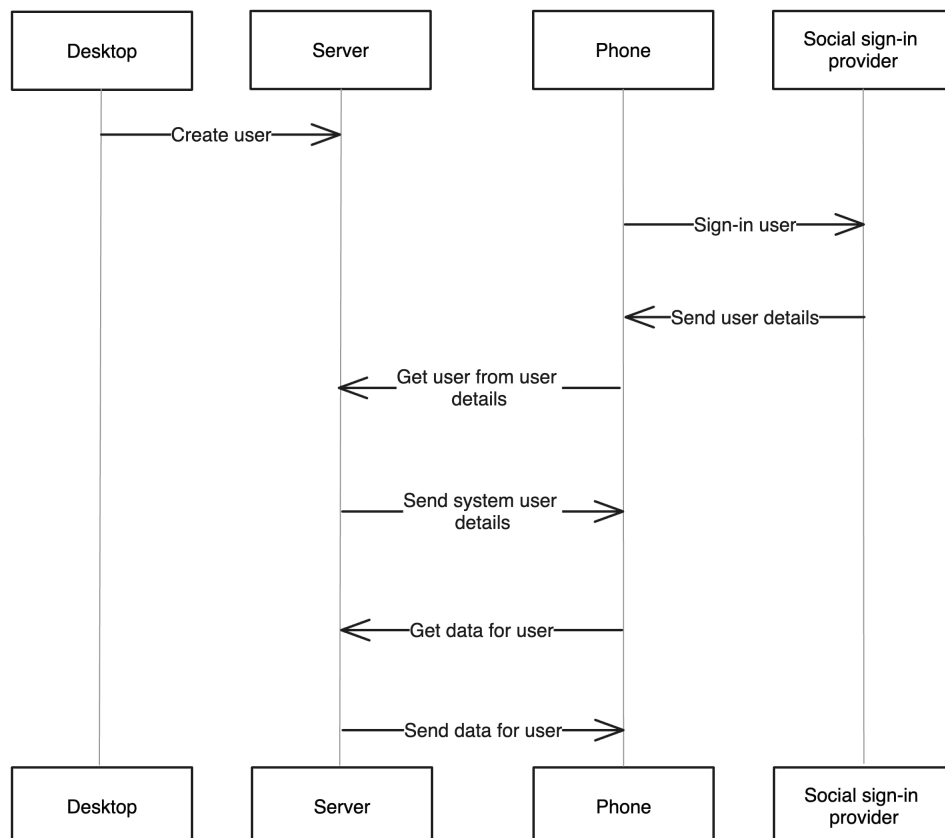



Figure 2.3: Communication schema

The flow begins when a desktop application sends its data to a server, where it is stored; this happens upon the technician's decision that the data is ready to be transmitted to the athletes. The next phase begins when a user opens the mobile application and clicks on login via a social provider; he is redirected to the provider link, where he finishes the login procedure and is redirected back to the application. The application receives user information from the provider, including the email address associated with the account. This email address is then used to match the user on the server. User information present in the database is sent after a match is found. The flow finishes when the mobile application asks for data relevant to the user and obtains it.



Chapter 3

System design

The proposed solution consists of a desktop application for the laboratory technicians to collect and visualize data, a mobile application for the sportsmen and their trainers, and a remote data source. The desktop application provides an interface that abstracts the laboratory machine's interfaces and unifies them to hide the underlying complexity from users. This component facilitates data processing and aids the evaluation of the sportsmen's performance metrics. It also allows for selecting results ready to be transferred to the users and choosing which users should see which data.

The mobile application's primary goal is to enable viewing of the metrics, deliver them on time to the athletes, and track the changes over time. It is also supposed to show data for both the person logged in and data meant for athletes with this person as a trainer.

Last but not least, the remote data source serves as a central repository for the measurements. The mobile application can only read data from this data source and is not allowed to write anything. The desktop application can both read and write to ensure data is synchronized between multiple devices if more than one laptop is being used.

The components combined deliver a more intuitive and pleasant experience while automating many parts of the process. This is described more elaborately in the section on system architecture 3.1.

Firebase authentication service will be used for authorization as it provides a robust and user-friendly implementation of social sign-in, making it easier for end users to log in as they do not need to create a new account. This makes for an easier transition from the old system as it expects less effort from the end users to use the new solution. It will also be used for managing the roles of the users as well as their personal information such as e-mail address, name, or avatar image. Lastly, by using Firebase, the overall security will be improved, increasing trust in the system.

In terms of the design, the focus is on a clean, simple, and minimal user

interface, striving to achieve an aesthetic and uncluttered experience. The goal is to create a system that even people with limited experience with similar systems or problems handling digital solutions can use smoothly.

The desktop application will be physically connected to the medical devices to receive the data and send the results to the remote database wirelessly. The mobile application will not have any peripherals connected physically and will be accessing the measurements wirelessly from the remote database.

3.1 System Architecture

The system can be divided into two parts: existing infrastructure and newly developed infrastructure. Medical devices for measuring the samples, a remote database currently existing for data archiving, and Firebase together form the existing infrastructure as they will not be implemented, enhanced, or altered in any way. The desktop application serving medical technicians and the mobile application made for sportsmen are newly developed infrastructure as they will be built from the ground up. There will also be a data source mimicking the existing one used for testing to prevent production data corruption.

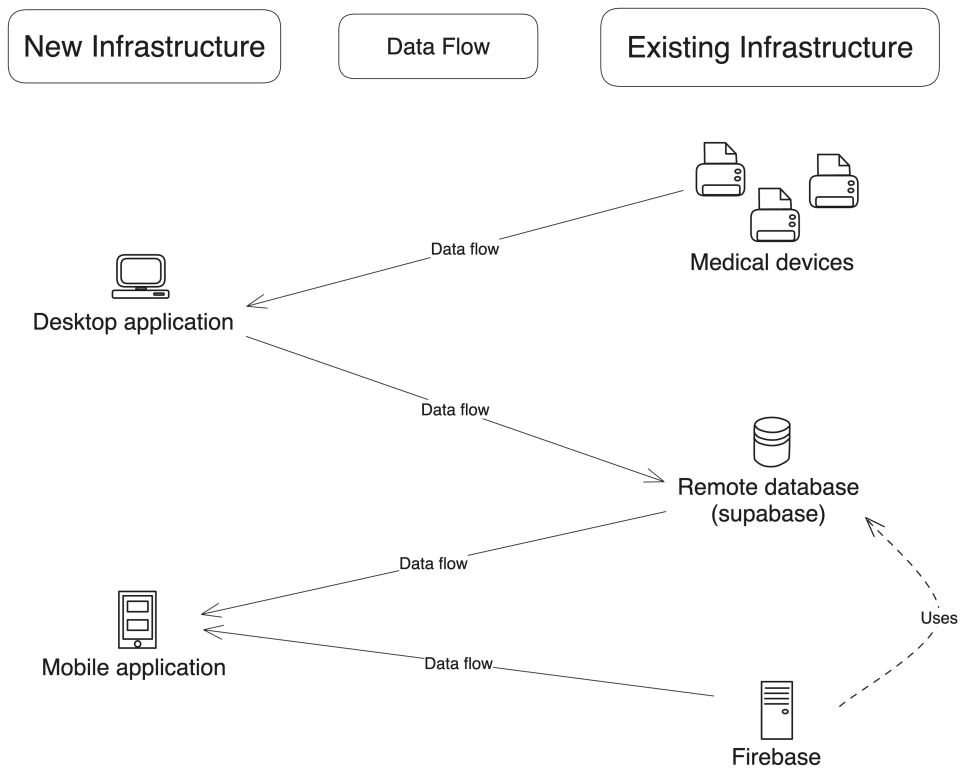


Figure 3.1: System's Architecture

Application of the architecture to the current process

The changes to the procedure are the following: it begins by measuring respective values by third-party medical devices as before. The information those machines provide is then delivered to the first part of the developed system: the desktop application through a serial line. This software parses the data from various formats into application classes. It displays the readings to the laboratory technicians in a table-like interface where the data can be altered, exported, and used in other applications. The system allows simultaneous connection of multiple devices that can be preconfigured at any point in the device control part of the application. The devices are expected not to change frequently and thus be set during the installation once and stay the same or change slightly during usage.

After data analysis, the results can be sent to sportsmen, their trainers, or both, depending on the settings in the table. The onboarding process includes downloading a mobile application, another portion of the system being developed, and registering through social sign-in. This mobile application is then used to view the results of the measurements.

Data that have been analyzed are securely stored in a remote data source that is part of an existing infrastructure. The data is retrieved from this database to be shown to the athletes.

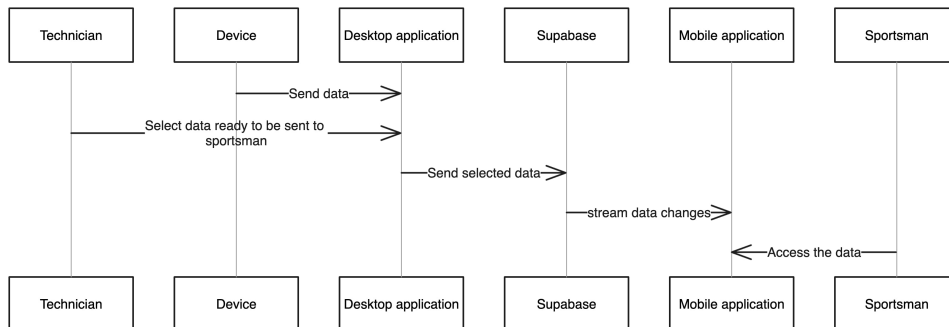


Figure 3.2: Proposed situation diagram

Comparing the developed system diagram 3.2 and the old system diagram 2.1 one can quickly see an enormous difference in workload since the technician now only needs to select data that is ready and send it to respective parties in contrast to the previous multi-step process that included reading

and writing the data manually, creating an excel sheet and encrypting it with a password and sending everything. For the athletes and their trainers, it is also simplified as they don't have to download an Excel sheet, find the password, and decrypt it, but can now open an application and immediately see the results.

■ 3.2 User Interface Design

The design for the application was done in the Figma design tool. The focus when designing both of the newly developed applications was mainly on making the interface minimal, familiar, and easy to navigate for users with diverse technical proficiencies.

■ 3.2.1 Desktop application

The desktop application is divided into a dashboard screen serving as a home page, a device control screen for managing devices, a subject control screen for managing sportsmen and their trainers, an administrator screen for controlling the application and viewing logs, and finally, a data view meant for scenario handling such as assigning devices to the current measuring, automatic data loading into the application and result validation. Those screens are being described below.

■ Dashboard screen

The dashboard screen in figure 3.10 is an entry point of the desktop application, and it offers quick navigation into the device control, subject control, and admin panel, see figure 3.3. The dashboard also offers a download function to synchronize events with the remote data store and, hence, with other devices.

Finally, it displays events in cards as the main content. An event or scenario is an entity representing measurements of a specific group at a specific location and time. Those are ordered in terms of time for quicker access.

An event card is meant to display the most basic information about an event at first glance, this includes the name of the event, dates representing when the event will take place, the amount of subjects and the amount of devices. There is also a bin button to delete the event. After clicking on the card user is routed to the data view and hovering on the button is signaled by changing the color of the card as depicted in the image 3.4.



Figure 3.3: Dashboard navigation

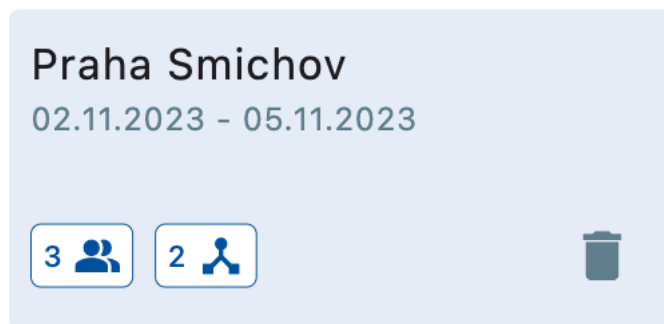


Figure 3.4: Event card

■ Device control screen

The device control screen in figure 3.11 is where all the devices are to be handled. Users can create a new device, delete an existing one, name a device, assign a note, and connect it to a COM port. Devices that are set up here are then available to be assigned to an event and are automatically connected.

Devices of a specific type can be added by clicking on a button in respective column. This creates a new entry with an empty card. This card visualizes information about a device: name of the device, its port and possibly a note. There is also an edit button as well as a delete button, see figure 3.5.

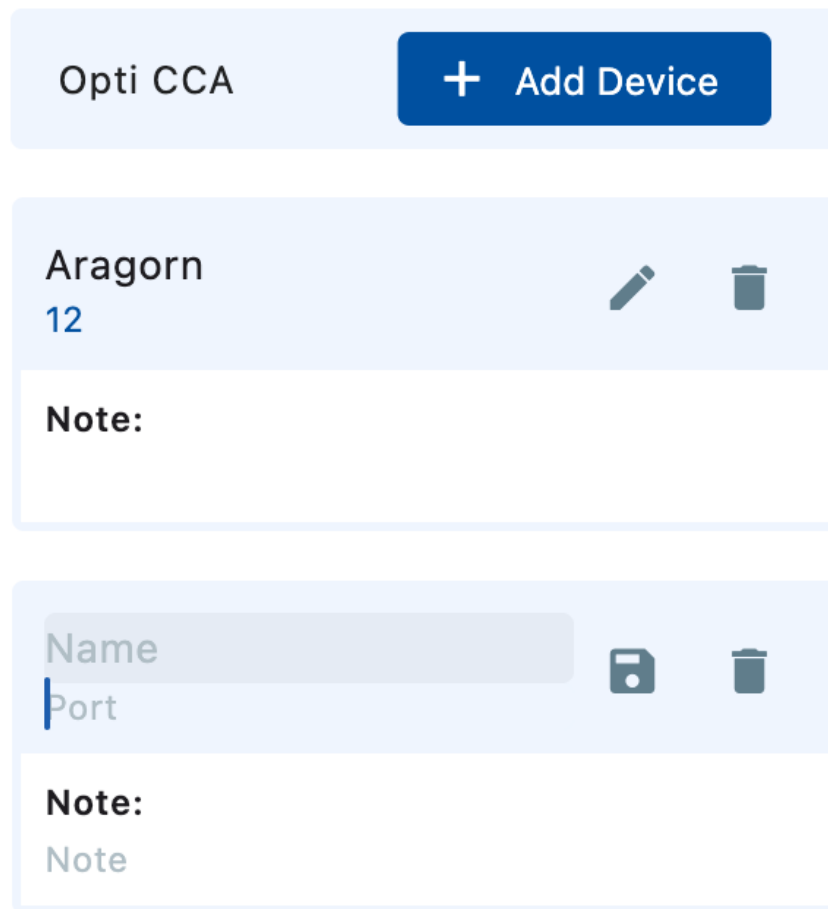


Figure 3.5: Device card

■ Administrator screen

The administrator part of the system is designed for control over more advanced functionalities such as database deletion, viewing logs for data measurements, or going through all the available users in the data store, depicted in figure 3.13.

An example of a card in the administrator view is database card, containing information about numbers of records in the database. There is also a possibility to clear concrete types of records such as clearing only events via a bin icon next to the type that should be deleted. It is possible to delete the whole database and all its records through a delete DB button at the bottom of the card, see figure 3.6.



Figure 3.6: Administrator card

■ Subject control screen

The subject control is for managing sportsmen and their respective trainers and can be seen in figure 3.12. Subjects can be downloaded for synchronization. Technicians can create a new person entity, edit or delete an existing one, and assign trainers for those with sportsman roles. After creation or edition, the entities are automatically synchronized.

Creation of a new person is done via a card with a switch for a person's role and his respective information such as name, e-mail and id. After saving a new entry is created that can be edited via an edit button or deleted by clicking on a bin icon, see figure 3.7.

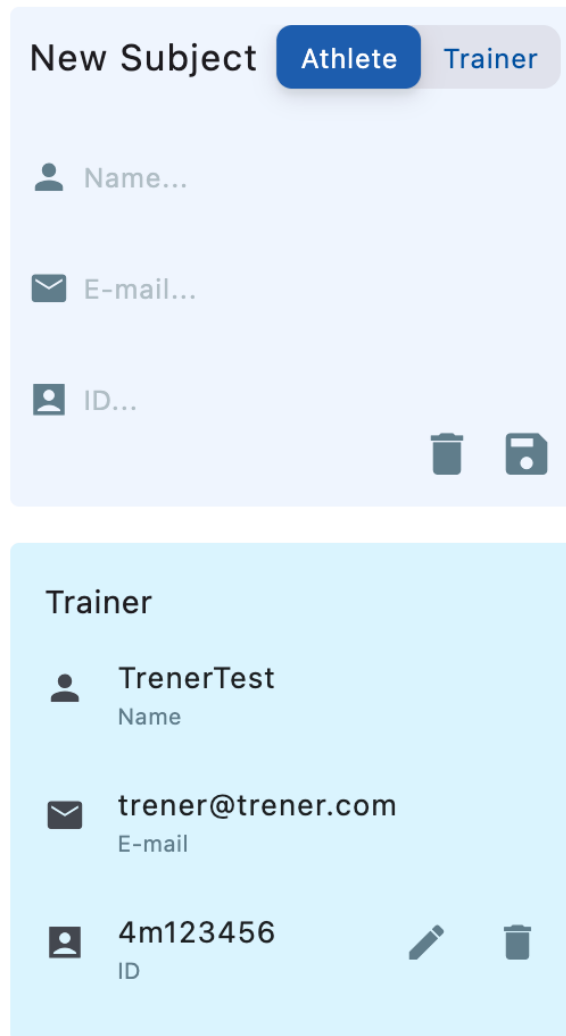


Figure 3.7: Subject card

■ Data view screen

Last but not least, the data view presents all the information for a single event. It is divided into three main parts: the control panel at the top of the screen, a side panel for device assignment on the right side, and finally, a table view of measurements in the center of the screen with table actions above that. The control panel allows for event title and date modification. It also serves for measurement upload and download, showing and hiding the device panel, visible in figure 3.9, and exporting the event into other formats. The side panel adds and removes devices used in the current event and turns on serial line initialization. The table view is meant for creating lines representing a single measurement of a single person, and the data is automatically set to it. The table actions offer group changes of various fields to enhance the experience further and speed up the workflow.

For operations over the table a table actions panel was created, displayed in the image 3.8. Those actions contain: creation of a new blank subject in the table, opening a selection column to conduct multi-row operations, duplication of selected rows while keeping the same values such as date of collection or subject name. Creation of a sequence of IDs is useful for generating multiple ones simultaneously, either from the maximum in the table or from a selected value. Last but not least, a group change of a value can also be done from the panel.



Figure 3.8: Table actions

Glucose	Lactate
12123.0	442211.0

Figure 3.9: Data view actions and Devices

3. System design

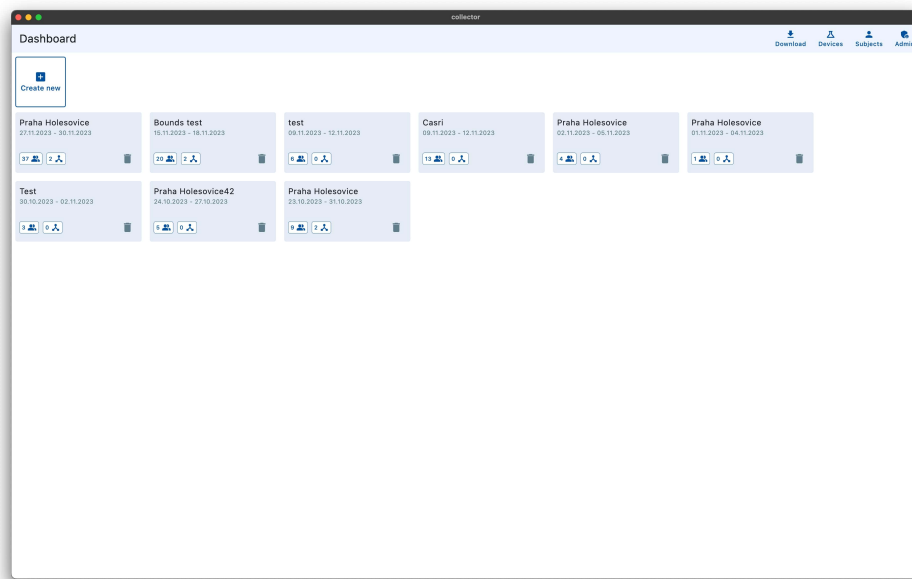


Figure 3.10: Dashboard screen

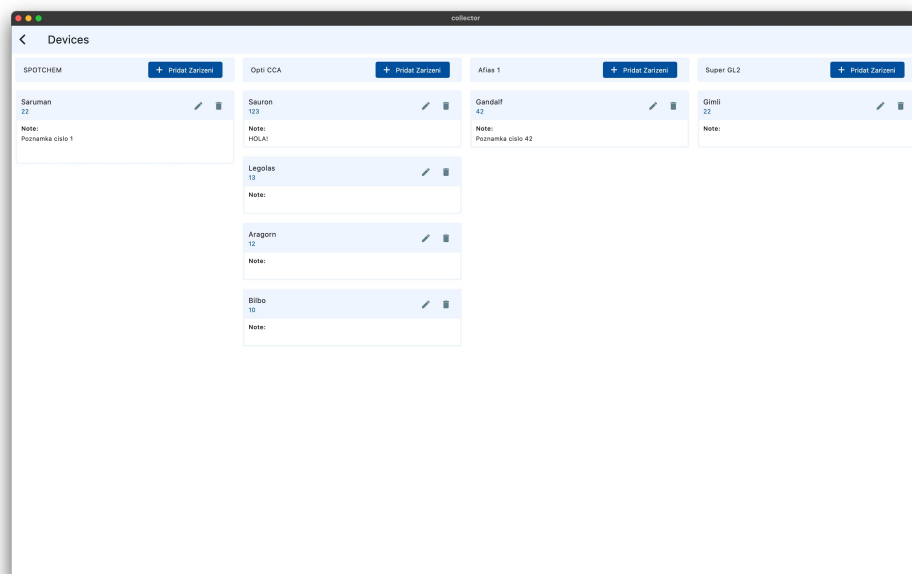


Figure 3.11: Device control screen

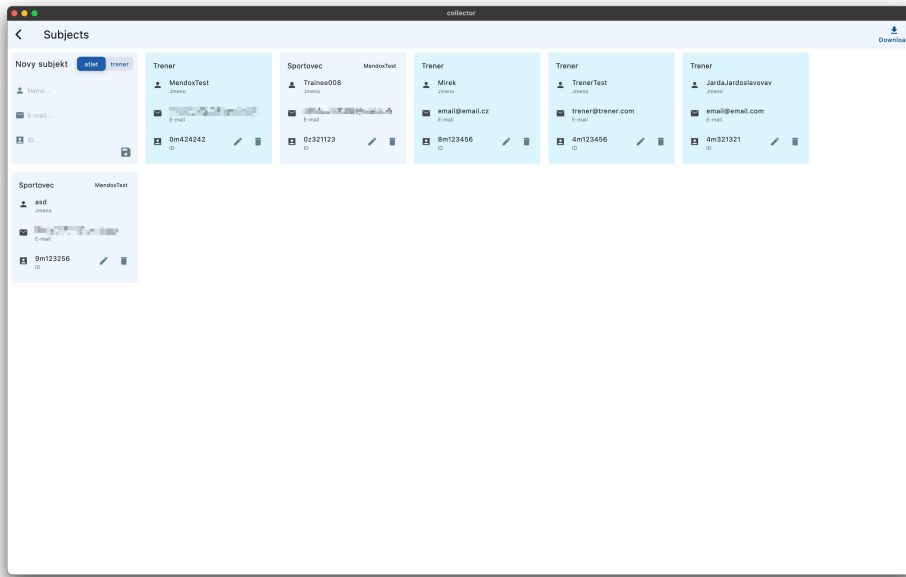


Figure 3.12: Subject control screen

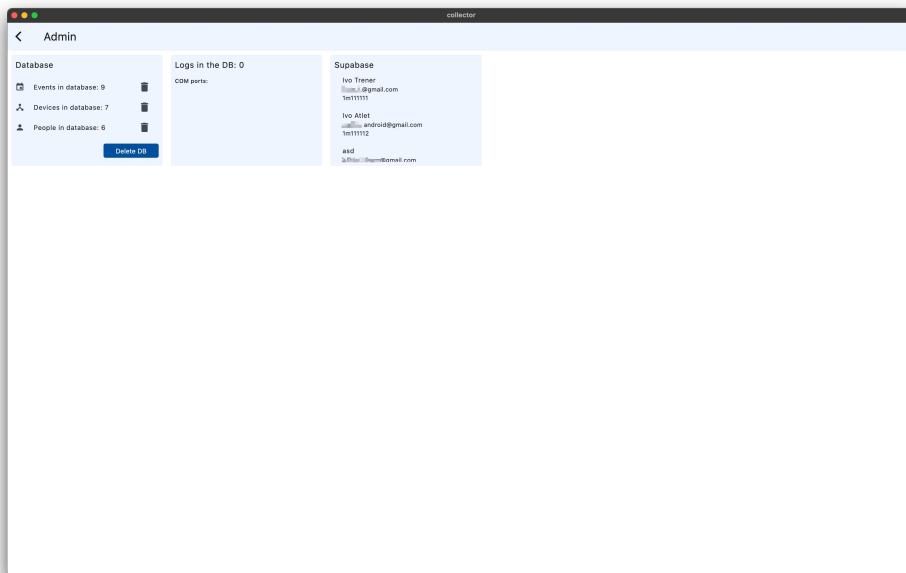
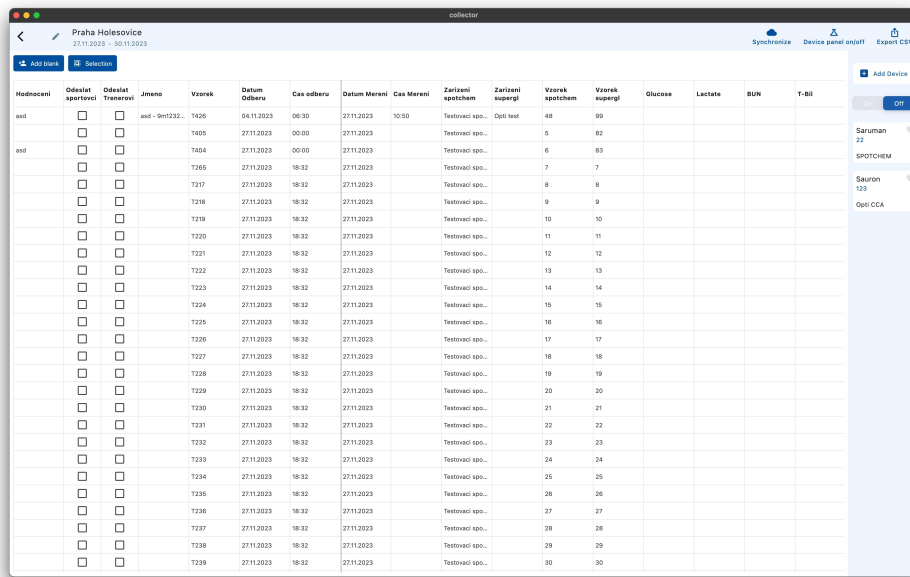


Figure 3.13: Administrator screen

3. System design



Hodnocení	Odeslat spotřeba	Odeslat Testovací	Jméno	Vzorek	Datum Odberu	Čas odberu	Datum Měření	Čas Měření	Zarizení spotřeba	Zarizení kontrol	Vzorek spotřeba	Vzorek kontrol	Glucose	Lactate	BUN	T-Bil
asd	<input type="checkbox"/>	<input type="checkbox"/>	asd - 9m1322...	T426	27.11.2023	06:30	27.11.2023	10:50	Testovací spo...	Opti test	48	99				
	<input type="checkbox"/>	<input type="checkbox"/>		T405	27.11.2023	00:00	27.11.2023		Testovací spo...		5	82				
asd	<input type="checkbox"/>	<input type="checkbox"/>		T404	27.11.2023	00:00	27.11.2023		Testovací spo...		6	83				
	<input type="checkbox"/>	<input type="checkbox"/>		T185	27.11.2023	18:32	27.11.2023		Testovací spo...		7	7				
	<input type="checkbox"/>	<input type="checkbox"/>		T217	27.11.2023	18:32	27.11.2023		Testovací spo...		8	8				
	<input type="checkbox"/>	<input type="checkbox"/>		T218	27.11.2023	18:32	27.11.2023		Testovací spo...		9	9				
	<input type="checkbox"/>	<input type="checkbox"/>		T219	27.11.2023	18:32	27.11.2023		Testovací spo...		10	10				
	<input type="checkbox"/>	<input type="checkbox"/>		T220	27.11.2023	18:32	27.11.2023		Testovací spo...		11	11				
	<input type="checkbox"/>	<input type="checkbox"/>		T221	27.11.2023	18:32	27.11.2023		Testovací spo...		12	12				
	<input type="checkbox"/>	<input type="checkbox"/>		T222	27.11.2023	18:32	27.11.2023		Testovací spo...		13	13				
	<input type="checkbox"/>	<input type="checkbox"/>		T223	27.11.2023	18:32	27.11.2023		Testovací spo...		14	14				
	<input type="checkbox"/>	<input type="checkbox"/>		T224	27.11.2023	18:32	27.11.2023		Testovací spo...		15	15				
	<input type="checkbox"/>	<input type="checkbox"/>		T225	27.11.2023	18:32	27.11.2023		Testovací spo...		16	16				
	<input type="checkbox"/>	<input type="checkbox"/>		T226	27.11.2023	18:32	27.11.2023		Testovací spo...		17	17				
	<input type="checkbox"/>	<input type="checkbox"/>		T227	27.11.2023	18:32	27.11.2023		Testovací spo...		18	18				
	<input type="checkbox"/>	<input type="checkbox"/>		T228	27.11.2023	18:32	27.11.2023		Testovací spo...		19	19				
	<input type="checkbox"/>	<input type="checkbox"/>		T229	27.11.2023	18:32	27.11.2023		Testovací spo...		20	20				
	<input type="checkbox"/>	<input type="checkbox"/>		T230	27.11.2023	18:32	27.11.2023		Testovací spo...		21	21				
	<input type="checkbox"/>	<input type="checkbox"/>		T231	27.11.2023	18:32	27.11.2023		Testovací spo...		22	22				
	<input type="checkbox"/>	<input type="checkbox"/>		T232	27.11.2023	18:32	27.11.2023		Testovací spo...		23	23				
	<input type="checkbox"/>	<input type="checkbox"/>		T233	27.11.2023	18:32	27.11.2023		Testovací spo...		24	24				
	<input type="checkbox"/>	<input type="checkbox"/>		T234	27.11.2023	18:32	27.11.2023		Testovací spo...		25	25				
	<input type="checkbox"/>	<input type="checkbox"/>		T235	27.11.2023	18:32	27.11.2023		Testovací spo...		26	26				
	<input type="checkbox"/>	<input type="checkbox"/>		T236	27.11.2023	18:32	27.11.2023		Testovací spo...		27	27				
	<input type="checkbox"/>	<input type="checkbox"/>		T237	27.11.2023	18:32	27.11.2023		Testovací spo...		28	28				
	<input type="checkbox"/>	<input type="checkbox"/>		T238	27.11.2023	18:32	27.11.2023		Testovací spo...		29	29				
	<input type="checkbox"/>	<input type="checkbox"/>		T239	27.11.2023	18:32	27.11.2023		Testovací spo...		30	30				

Figure 3.14: Data control screen

3.2.2 Mobile Application

The mobile application was developed to serve mainly as a data consumer and to show feedback to the users. Therefore, it is relatively straightforward with a login screen, a home page with personal and trainee tabs with their respective views, and a drawer with additional settings. Lastly, it has a detail screen displaying measurements with an evaluation, all grouped by events.

Since the mobile application is going to be used by people outside of the organization, it was implemented in both light and dark themes. It makes it more visually appealing and readable in direct sunlight when used with light mode and puts less pressure on the eyes and the phone's battery with dark mode.

Login Screen

The login screen, depicted in image 3.17a is an entry point for the application, exposing social sign-ins and letting users access their data. This screen is the most minimal as it only redirects to the sign-in provider's sites and receives the user information it has permission to get.

■ Home screen personal

The Home screen within the personal tab, visible in figure 3.16 lists all the events where the logged-in person was measured as cards. The cards provide information about the event's name and date, and they lead to a detail screen.

■ Home screen trainees

The Home screen within the trainee tab goes through all the events where logged-in person trainees were measured, see figure 3.15b. Again, They are displayed as cards and grouped by a trainee's name.

■ Detail screen

The feedback provided by the medical professionals, along with measured results for individual values, are shown in the detail screen, presented in figure 3.17. Many measurements may be conducted during an event, and that is modeled as dropdown-style opening cards, one for each measurement. Feedback is at the top of every card as it provides personalized information. The values are then listed below, marked with a color corresponding to the ranges they are supposed to fit in.

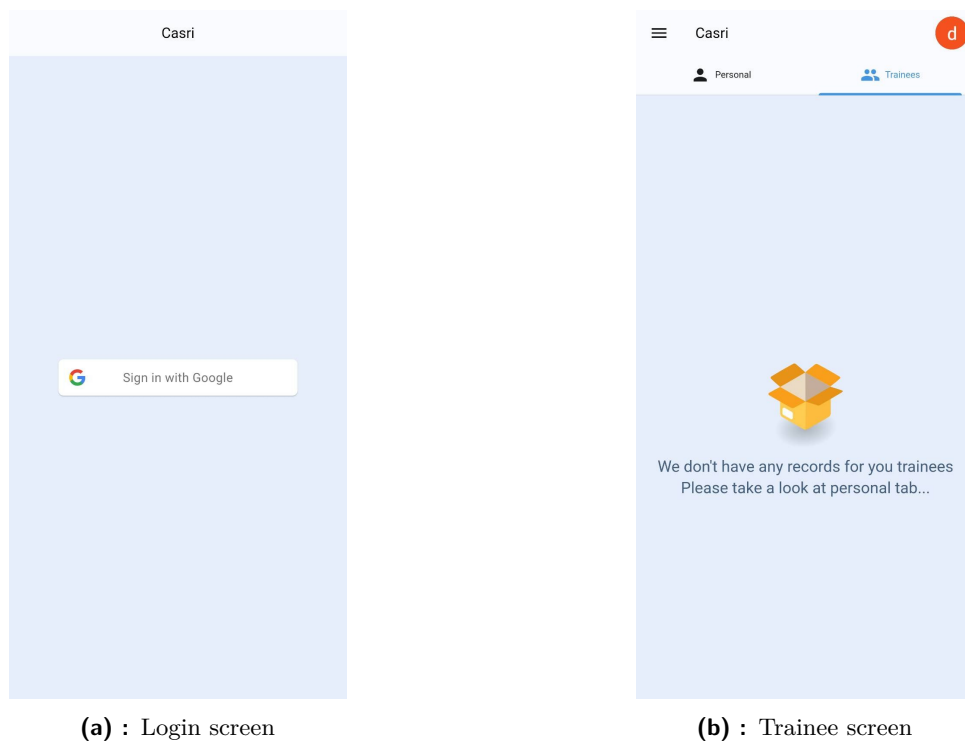
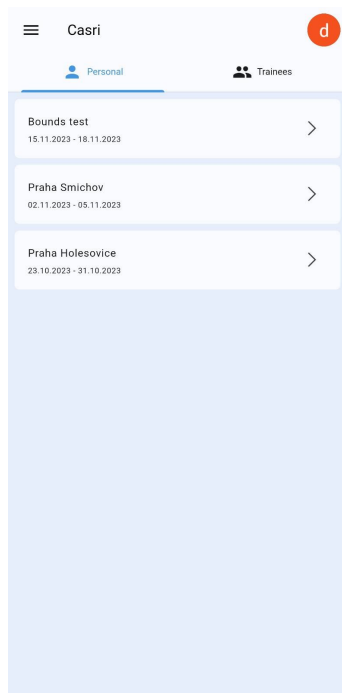
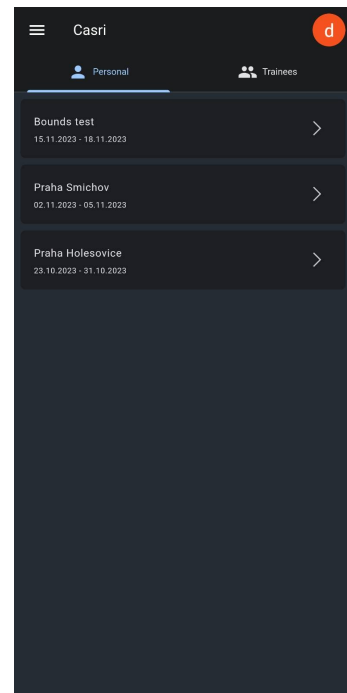


Figure 3.15: Login screen and trainee screen

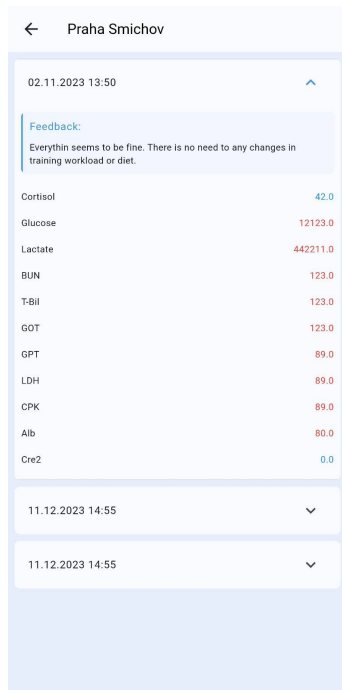


(a) : Dark mode

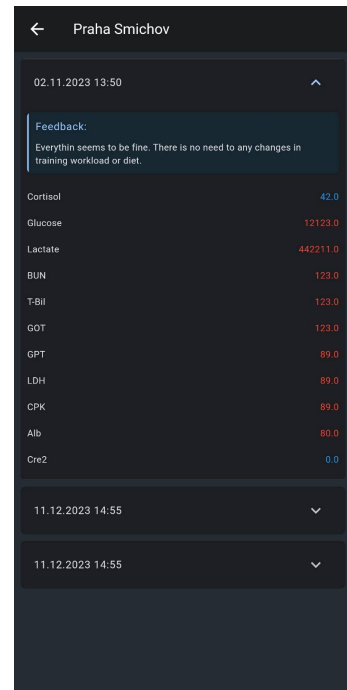


(b) : Light mode

Figure 3.16: Home screen personal



(a) : Dark mode



(b) : Light mode

Figure 3.17: Detail screen

3.3 System's user management

To handle user management without the need to create a new account with new credentials, social sign-in will be used and mapped to an E-mail address and person ID, both of which are already in usage. The usage of social sign-ins also provides additional data such as profile image without the need to bother the user and ensures that the E-mail address is valid and verified. This is supported by an article, *The Rise of Social Login: Bridging the Gap Between Convenience and Security* [34], which covers forms and benefits. The concept is further described in Auth0 documentation [8], and the flow is depicted in the image 3.18.

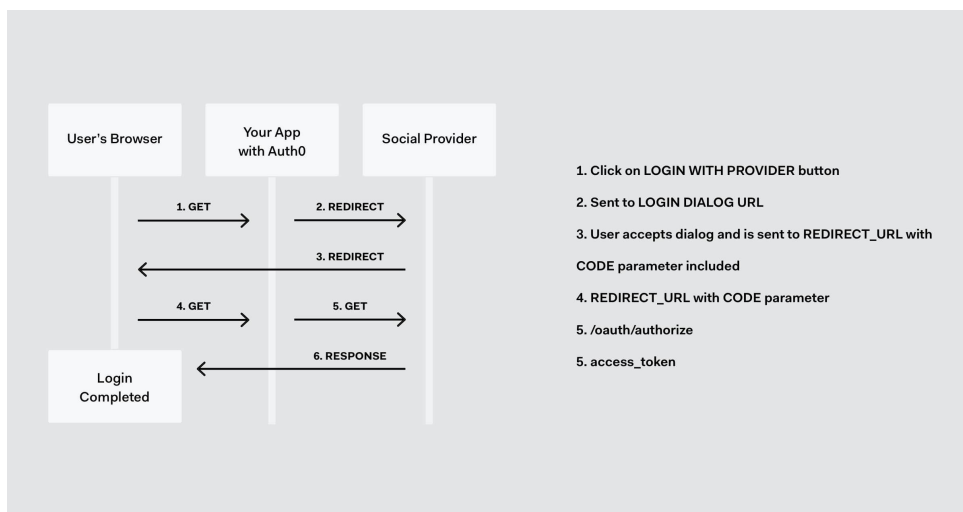



Figure 3.18: Social login diagram [8]



Chapter 4

Implementation

For development, an iterative approach was chosen. After analysis and the first phase of design, a portion of the application was built and tested to gain feedback rapidly and save development time. After every iteration, new requirements arose. Based on those requirements, both the design and the implementation were altered to match the desired behavior.

The whole system is developed in Dart programming language, using Flutter as a framework of choice for its speed, intuitiveness, and robustness. The desktop application is designed for Windows, and the mobile application is developed for Android and iOS operating systems. It is worth noting that by using Dart and Flutter, a significant portion of the code can and is being reused throughout the parts of the system thanks to the multi-platform options provided by Flutter. The applications perform native-like because of the Flutter's architecture mentioned more in detail in section 4.1.

The system connects to the medical devices via a serial line, which delivers data in different formats for each device, making it more challenging to uniform. This data is parsed into dart classes using device-specific parsers and stored locally in a database on the laptop. Every device has a COM port [13] assigned to it in the application, as well as a name and possibly a note, making it easier to set up the whole system in the place of measuring as the device is recognized by the application due to the same COM port [13] and by the technician through the name. The system, therefore, does not need any pairing wizard to connect all the devices every time it is used. A dart library that internally uses FFI [16] is used to work with a serial line. More specifically, it allows initialization of a port, reading of the data via a stream, and manipulation with the port.

To store all the data that has been read, Supabase is used. It uses PostgreSQL for data storage, which is used in the current production database, hence ensuring the data can be migrated easily. Supabase also comes with real-time access, making the data transfer from a desktop application to a mobile one a simple and quick task.

4.2 Libraries

As in any other software project, common problems are already solved adequately by experts in the specific field and exposed by respective libraries. For a smoother experience and to not reinvent the wheel, libraries are also used in the proposed system, and some notable ones are listed below.

A significant benefit of Dart and Flutter is the extensiveness of high-quality libraries. Those libraries are also frequently reviewed by Google experts, and the most complete and beneficial ones receive a marker, making it easier to differentiate between sketchy libraries and mature ones.

4.2.1 Riverpod

Riverpod [29], a provider successor, is a library managing the application's state. State management in Flutter is dealt with by many libraries, and their approach differs a lot. The main differentiator is often the completeness of the solution, verbosity of the code necessary to be written, ease of understanding the solution, and correct usage. Among the various solutions, Riverpod [29], Provider [28], Bloc [12], GetX [20], and GetIt [19] are the most used by the community and described, for example, in *State Management in Flutter: A Comprehensive Guide* [31] and *Demystifying State Management in Flutter: An In-Depth Exploration* [15].

GetX [20] is discredited by using antipatterns and being used for too many things simultaneously, making it too robust to work with, error-prone, and unsuitable for complex applications. It also forces users to depend on a library that solves many more problems than they might need.

Meanwhile, GetIt [19] is simple, readable, and easy to use and understand. It does not provide enough tools to deal with state management in large and complex codebases by itself. Therefore, it is often used with other libraries that fix the problem, but they introduce a dependency on more libraries to solve one problem.

Flutter bloc [12], one of the most used, is excellent for clean architecture and testability but brings a significant amount of boilerplate code that essentially slows the development speed.

Finally, Provider [28] and Riverpod [29] are packages developed and tailored specifically for Flutter, making them the most declarative solutions that are the most familiar to those used to Flutter concepts. Since Riverpod is a successor to Provider and fixes many problems that the architecture of Provider simply has not allowed, it is superior.

the animation's motion.

Compared to GIFs, Lottie files are much smaller, making the app bundle size smaller and load times lower. Lottie files can also be tweaked using various tools available to match the design exactly.

■ 4.2.4 Serial port

For work with serial line, multiple libraries were necessary to be used. FFI [16] and win32 [38] are used to initialize the serial port on Windows with the correct settings [2], such as start and stop bits, baud rate, or parity. Flutter lib serial port [18] is used for communication with the port and reading the data from it as it is streamed by the devices.

The second was chosen for being the most mature among serial port libraries. It generally works as a wrapper around the dart lib serial port [14] that itself works by exposing the API of libserialport C library [24]. This solution seemed to give granular enough control over the port and communication with it; however, during development, it was found that the library was flawed, and it did not set the settings correctly. This fact motivated the usage of above mentioned win32 library that works on a lower level and allows for proper settings changes.

■ 4.2.5 Go Router

Routing in Flutter has shifted from Navigator 1 API to Navigator 2 API [26]. However, because of its complexity and boilerplate code, it is often used with third-party libraries to make the code more readable and maintainable. Navigator 2 marks a shift from imperative navigation to a declarative one.

There are many libraries solving this issue. The most mature include Beamer [9] and Go router [21]. While Beamer was considered the more mature and complete due to considerable community support, Go router exceeded it and is the recommended approach.

Go router requires the declaration of possible routes and providing this as a configuration for the root of the application. This configuration allows the application to determine what pages should be on the stack. It also makes it simpler to send data between the routes.

■ 4.2.6 Bitsdojo window

To work with the application windows received from the platform and change its looks and functionality, a package called Bitsdojo window [11] is used. It

triggers application state changes, thus forcing the user interface to rerender. The controllers are also responsible for getting data from supabase and writing into it.

The controller's functions are triggered in the user interface, and outside of that can only be called inside their layer. They are exposed in the same way as services, through simple providers or via Notifiers. If a functionality falls into a service domain, the work is delegated to it. Otherwise, it is handled inside the controller's layer. Controllers also have to keep updating the repository layer during their work to visualize the progress to the user.

■ 4.3.3 Repository

A repository layer is introduced for storing and editing the application's state. It is built from raw components provided by the Riverpod library. Providers and Notifiers are mainly used as they provide the best value and are also the preferred way in the documentation.

Simple Providers are used when the interface only needs to read a value, be it from a remote database or a local one; however, that value may not be changed. Notifiers are used where more complex operations are required, where data needs to be edited or deleted, or any other additional requirement is posed. All components from the Riverpod library can be used together and combined in any way to achieve the desired behavior.

The state is generally immutable in those components and has to be re-assigned to ensure the change takes effect. This makes for a cleaner API that triggers notification of state changes automatically precisely when it should.

Where asynchronous operations are performed, a handy wrapper class `AsyncValue` is used. It offers methods to handle data, loading, and error states [37] at the interface layer, making sure the application does not enter an invalid state that it does not know how to react to, hence rendering correctly.

The repository layer cannot call methods in other layers and is called only from the controller layer and listened to from the interface layer. It is generally just a passive layer exposing the data and holding the state that is changed from controllers.

■ 4.3.4 User Interface

The visual part of the application is in the user interface layer, which is divided into pages and widgets. The elements that form the page are in the components folder attached to every page. A widget is the smallest building block used. Depending on the developer's design, it can be a simple

■ Supabase database

Supabase's database is the most crucial functionality for the system developed in this thesis as it is used to store all the data and stream them to mobile devices. Multiple tables, depicted in figure 4.3, were created within this database: one for people, keeping information about athletes, trainers, and their connections; one for events, storing data such as the date of the event, its name, and subjects. Finally, a table for subjects is used for all the measurements along with the IDs used for correct data assignment, measured values, the ID of a person the data belongs to, and dates of collection and measurement.

Since the database runs PostgreSQL, the tables were created using SQL scripts. Those scripts can be used to ease a potential data migration to another backend. Most of the fields were defined as non-null with default values to avoid null pointer errors and also to mimic data classes in the applications that strictly follow no null value policy that is then enforced by Dart's sound null safety.

To work with the data from supabase in Flutter applications multiple components were necessary to be built. All the classes that were stored in the Supabase database had to implement serialization methods `toSupabase()` and `fromSupabase()`, visualized in figure 4.2; those methods used new Dart syntax, allowing for more straightforward and readable parsing. The other component was a Supabase service that built all the queries that accessed the data.

Important supabase library methods used for building the queries:

- `Supabase.from("tableName")` - Creates a query builder for the selected table name
- `table.stream(primaryKey: ['id'])` - Creates a stream of data from the selected table and uses the 'id' to ensure proper real-time streaming when data is updated or deleted
- `dataStream.eq('propertyName', value)` - Filters streamed data with the property name and respective value
- `dataStream.inFilter('propertyName', values)` - Filters streamed data with multiple possible values
- `dataStream.contains('propertyName', values)` - Filters stream data where a column contains a list of elements and validates if a values list is contained within that column
- `table.select<List<Map<String, dynamic>>>()` - Instead of returning a stream, returns a future with a snapshot of current data

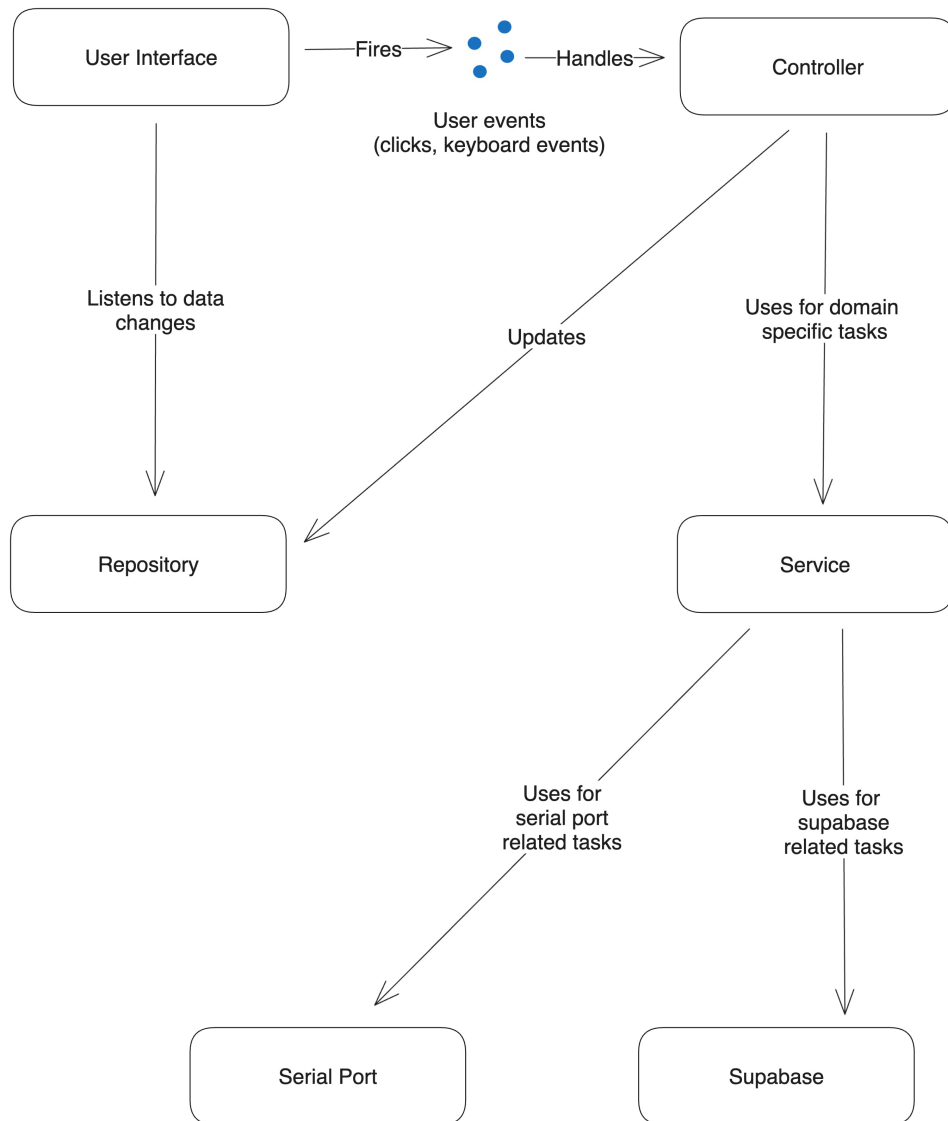


Figure 4.1: Application's Architecture

```

factory Event.fromSupabase(Map<String, dynamic> map) {
  if (map
    case {
      'id': final String id,
      'location': final String location,
      'dateFrom': final String dateFrom,
      'dateTo': final String dateTo,
    }) {
    return Event(id: id, location: location, dateFrom: dateFrom, dateTo: dateTo);
  }
  throw Exception('Person from supabase format error!');
}
  
```

Figure 4.2: Supabase parse method

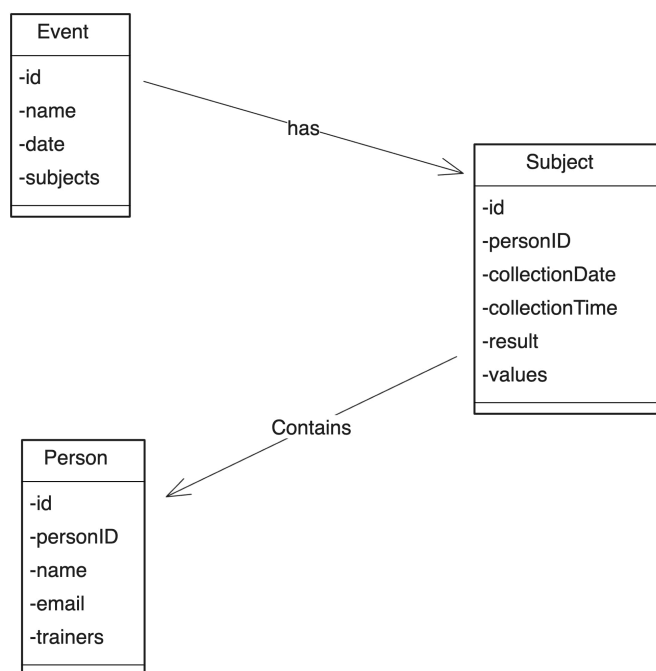


Figure 4.3: Supabase data hierarchy

Chapter 5

Testing

Testing is vital for any system and is also very important for application development. It serves as a quality assurance for both the developers trying to make the best possible product as well as for customers wanting software that is reliable and solves the problem at hand. It is also meant to secure the system, as software without bugs is more challenging to exploit. Testing not only helps find errors in the code but can also work as prevention before regression errors.

Testing can be done on multiple layers with various levels of granularity. This ranges from simple unit tests covering small encapsulated units of code to complete interface tests going through complex portions of the application. It can also be divided into software and user testing. Tests conducted by appropriately chosen testers can quickly prove to be valuable as they mimic the real usage of the system.

For the system developed as this thesis, testing was divided into automated unit tests covering the logic of the code and ensuring no regression appears in the future and complex testing of the whole interface done with experts from the field to have a solution that is robust enough to handle current and future workload and one that tackles the problem in an easy to use way.

5.1 Software Testing

For unit testing in Flutter, a package called Flutter test is used as it provides all the necessary primitives, as shown in the article [Unit Testing in Flutter \[35\]](#). Tests can thus be grouped, and chunks of code can be run before and after test execution for data initialization and resource creation before each test or before a group of tests and also to clean up after the tests finish.

The main focus was given to testing data parsers as they have the most logic in them, and the whole system relies on their correct implementation. The core functionality of the data parsers is in their parse methods; therefore, they were subjected to thorough limit tests.

The most essential testing primitives that were used for unit testing are the following:

- `group` - Grouping multiple smaller tests into a larger unit testing a bigger portion of the system.
- `test` - The smallest testable unit, testing one functionality with typically one case.
- `expect` - Primitive used for ensuring the correct value was indeed present.
- `setUp` - Used for preparing necessary resources before tests are run.
- `tearDown` - Used for cleaning up everything after tests finish running.

■ 5.1.1 Test coverage

Testing criteria for the system under test needed to be defined to properly measure how well is the system covered in terms of tests and whether testing met its goal. The need for proper definition is highlighted in Software Unit Test Coverage and Adequacy article [1]. Since the user interface was under constant expert review, which is explained in section 5.2, software testing of the application's interface was omitted, and the main focus was targeted to testing the logic. As previously mentioned, a large portion of the system logic and the most critical part lies in parsers and the implementation of their parse method. This functionality has, therefore, a hundred percent test coverage. It was also important to test formatting functionality as it caused some problems with date parsing during development. The service layer is mainly calling API from third party libraries that are tested internally, thus making testing of this functionality redundant.

■ 5.1.2 Found problems

Automated unit testing found multiple issues in the code in data parsing and date formatting, which are depicted in the table 5.1.

Problem	Solution
Date parsing for samples created by automatic subject creation when no matching row in the data table was found thrown exception for incorrect format	Incorrect string serialization of the <code>DateTime</code> class was used, resulting in improper formatting, changed to a correct one
Index out of bounds when parsing data from Afias 1 device	Corrupted data that does not contain all necessary fields is skipped and logged
An incorrect format error was raised in parsing data for the Supergl device	White space bytes signaling the beginning and the end of a measurement was cut before being parsed

Table 5.1: Errors found by unit testing

■ 5.2 Expert review

To test if the process automation works as expected, comprehensive interface testing was done with future users of the application. Multiple sessions were conducted, with each of them yielding valuable feedback that led to many changes in the functionality and user interface layout and design. It also ensured that there was a need for a solution.

Validating that there is a problem to be fixed and that the developed system is solving a real issue is essential to avoid product disasters. User testing can significantly aid in this manner, as described in *The Importance of User Testing* [36].

The testing was mostly done in a similar manner, forming a type of scenario described below. The main goal was to ensure both the development team and future users had the same understanding of what the result would be. It was also vital in exposing errors in the system as well as keeping the complexity of the interface low and the whole system understandable. Finally, some portions of the system could not be thoroughly tested without using real devices for measurement collection. This was primarily for the serial connection and the communication protocols and settings used by the machines.

Data collected before implementation was used for testing throughout the iterations and served as a template for parser development as well as a base for the data table. Once the skeleton of the desktop application was finished, the testing transitioned to the use of data provided by physical devices in real time.

During testing with the devices, problems with data transmission were found, namely data not being sent after collection or the usage of multiple formats by a single device. Documentation did not provide any solution or explanation of such behavior, thus requiring contact with customer support. Finally, the issues were fixed by changing the configuration of the device. For multiple format problem, the corresponding parser had to be altered to switch based on what data he reads and adjust to it.

■ 5.2.1 Selected Users

Generally, in user testing, it is critical to select the correct people, ideally those who would afterward use the product. There should also be a reasonably large amount of those testers to ensure high-quality feedback. However, given that the system is tailored to a very specific scenario, it was decided that it would be tested within the company for which it was being developed. Due to the iterative nature of the development process, this testing was conducted many times throughout the implementation. This was crucial for maintaining

a clear path to the final solution and a scalable application supported by the article Best practices for building scalable Flutter applications [10].

■ 5.2.2 Testing scenarios

User testing was done without specific step-by-step scenarios, precisely leading people through the application. However, all functionalities needed to be used to expose any vulnerabilities or bugs that could be present in the system. Similar patterns emerged during the early stages of testing that were afterward loosely followed. Those patterns could be formalized into the following scenarios.

■ First scenario - Management of subjects

Part of system: Desktop application

Area of testing: Subject management

Description: Validation of correct user creation with all the fields. The user should be created after clicking on the save button, and the state of the form should be reset. The user can be created as a Trainer or a Trainee; those roles should be visually distinct. A Trainer user should be assignable to a Trainee.

Value thresholds: ID - digit followed by a character m,z followed by 6 digits, E-mail - valid email

1. Go to the subject control screen
2. Fill in the fields and create a subject with a role *Trainer*
3. Fill in the fields again and create a subject with a role *Trainee*
4. Start edit mode of a subject with a Trainee role
5. Choose a trainer and assign him
6. Save the Subject

Expected outcome: There are two newly created users, one with a trainee role and one with a trainer role. The trainer is assigned to a trainee.

■ Second scenario - Management of devices

Part of system: Desktop application

Area of testing: Device management

Description: Validation of proper logic when creating a device. A new device should appear after clicking the add button in the respective column.

1. Go to the device control screen

2. Create a new device for all four types
3. Fill in the name of each new device
4. Fill in the port for all new devices
5. Save all newly created devices

Expected outcome: There are four new devices in the application, one of each type.

■ Third scenario - Data viewing and creation of events

Part of system: Desktop application

Area of testing: Data view and event creation

Description: Event creation works, and users can assign devices and create subjects. The name and date of the event can be changed, and subjects can be altered as well.

Prerequisites: Opti and Supergl devices created in the application.

1. Create a new event
2. Open the device panel and add a SuperGl device and an Opti device
3. Create a new subject and change its properties
4. Edit the name of the event and change it to a unique value
5. Save the event's name

Expected outcome: There is a new event with a unique name; it has two devices assigned to it and a subject.

■ Fourth scenario - Automatic collection of data

Part of system: Desktop application and physical devices

Area of testing: Automatic data collection

Description: Devices should signal their status through their connection color. Data should be automatically assigned when a subject has a valid ID, and a device is connected. A new one should be created automatically if no subject matches the data.

Prerequisites: Opti and supergl devices created in the application and assigned to an event. The event has at least one subject.

1. Open an event with devices assigned and at least one subject present
2. Assign the sample ID from the physical opti device to the subject

3. Assign a supergl ID based on the current sequence from the physical supergl device
4. Connect the physical devices to the laptop
5. Ensure there is a supergl and an opti device assigned and turn them on
6. Start the data measuring on the physical devices and input the same sample ID into the opti device
7. Start another data measuring on the physical devices and input an incorrect sample ID into the opti device

Expected outcome: The First measurement should appear in the first subject, correctly assigning the values to the table. The second measurement should be on a newly created line with data from the measurement automatically assigned.

■ Fifth scenario - Viewing events in mobile application

Part of system: Mobile application

Area of testing: Login and event viewing

Description: A user should be able to sign in to the mobile application with his Google account credentials. He should also be able to see events that he is part of in case of a trainee role and events that his trainees are part of in case of a trainer role.

Prerequisites: There needs to be a user created in the desktop application with an E-mail address corresponding to the one used in the sign-in.

1. Open the Casri mobile application and log in with your Google account
2. Select the personal view and validate that all events where you were measured are present
3. Select the trainee view and validate that all events where your trainees were measured are present

Expected outcome: User should be able to sign in with his existing account and see all events that he has access to.

■ Sixth scenario - Viewing details of events in mobile application

Part of system: Mobile application

Area of testing: Event detail

Description: User should be able to open the event that he participated in and view the results.

Prerequisites: There needs to be a user created in the desktop application with an E-mail address corresponding to the one used in the sign-in. There needs to be an event that the user is part of.

1. Open an event from the home page
2. Validate that there are all the measurements you went through during the opened event
3. Go through the measurements and validate that they have the correct values

Expected outcome: All the measurements for an event should be present in the detail screen. They should be collapsible and contain correct values.

■ Seventh scenario - Synchronization of data

Part of system: Mobile application, Desktop application, Database

Area of testing: Data synchronization

Description: Data should be synchronizable across the applications and be automatically pulled without user interaction needed.

Prerequisites: There needs to be an event with some measurements in it.

1. Login to the mobile application and open the home page
2. Open the data view screen in the desktop application
3. Assign a user to a measurement (Use the same user used in the mobile application)
4. Synchronize the event

Expected outcome: There should automatically appear a new event in the mobile application in case no other measurement was previously assigned. In case there was already another measurement for the user in the selected event a new measurement entry should occur.

■ 5.3 Testing results

As previously mentioned, testing was done during the whole development process, and it helped expose problems early. There were significant issues found; the most impactful was one with serial line communication and its library. The most critical problems found will be listed below, along with their respective solutions.

■ 5.3.1 Critical errors

Multiple critical errors that threatened the basic functionality of the system were found. Those errors are listed in the table 5.2 and then described more elaborately.

Problem	Solution
The serial line package was not working properly with a baud rate other than 9600, making the data sent from a device unreadable	A different package was used to initialize the serial port with other settings
Google sign-in was not working correctly on iOS, rendering the application inoperable	Platform files for iOS devices were re-configured
Data was not being parsed correctly	A view for logs was implemented and extra white characters were found that confused the parsers

Table 5.2: Critical errors

In the early stages, problems with serial port were found. The data received was not readable, and after extensive testing and debugging, it was found that the serial ports were misconfigured and the library used was flawed. Multiple solutions were considered, and another library using Windows primitives in the Dart language was chosen. This library first sets the correct settings for the port, and then the data is read as before.

The data read from the device seemed correct at first glance, but the parsers were throwing exceptions. Even extensive investigation did not show any progress. This motivated the implementation of more advanced logging that could be viewed inside the application. After going through those logs, multiple unexpected white space characters were found, and the parsers were altered to take that into account.

After testing the mobile application on Android devices, testing on iPhones was conducted. It was quickly found that Google sign-in was not working on iOS as on Android. After a long search, a solution was found, mentioning the necessary configuration that the documentation does not mention. The platform files were reconfigured, and the sign-in started working.

5.3.2 Low impact errors

Multiple low to medium-impact errors were also found throughout the system and were quickly fixed. Those include improper data assignment to measurements after data parsing, caused by data being overwritten multiple times and deleting the correct data. A simple merging mechanism was introduced to join partial results rather than rewrite them. Another example is a mobile app showing more data than it was supposed to, potentially causing confusion. This problem was fixed by strictly defining values meant for customers. The last example would be an incorrect generation of sample IDs, where immensely large IDs were generated. This issue was caused by storing the currently highest ID and incrementing it every time it was accessed. The problem

meant that when all the IDs were regenerated from a value, for example, value one, the stored ID stayed the same during the whole lifetime of the event. Another causality was when the event was changed on a different machine, the locally stored ID was not present, causing ID generation from the initial value. This was fixed by not keeping any IDs in a database but instead consistently generating them based on the context, ergo from currently present values in the table.



Chapter 6

Conclusion

The work done during this thesis was divided into phases very loosely following waterfall project development. The first phase consisted of understanding the problem at hand, going through the process, and figuring out its weak and strong parts. A thorough analysis of what devices are used, how they function, and what they measure was conducted, giving a rough estimate of what system might aid in automating the procedure. Multiple consultations also occurred during this phase, further advancing the understanding of the domain. Finally, research on available technologies, their limitations, and their benefits took place. Multiple problems in the current execution of the process arose, most of them being related to human-made errors or natural differences in effectiveness between a human technician and a machine reading the data. Those issues motivated the need for multiple applications forming a software system; one application meant for the company's employees and running on a laptop, reading the data and automatically assigning them to the correct athletes; the other application, a mobile one, targeted for the sportsmen. Those applications need to communicate together, serving the results of the measurements; thus, a remote database is required. Flutter was chosen for the applications and supabase as the remote data store, both of them because of their cutting-edge performance, ease of use, and excellent fit for the situation.

The second phase was mainly concerned with designing the whole system. The first part of the phase further narrowed the technology stack, deciding what portions of the problems would be a good fit for the usage of a library; examples include a local database, advanced animations, or state management, and what portions would be solved internally, for example, data buffering and data manipulation. In this phase, it was also decided that social sign-in via Firebase would be used. After the technical design, a user interface was designed; the central theme of that design was minimalism and a clutter-free interface that would be highly functional.

The last phase was dedicated to implementation and testing. This phase was internally iterative, consisting of sprints that were mostly one to two weeks long and contained both development and testing. This iterative

process started by creating a skeleton of the desktop application and building on top of that by adding more and more functionality. Once a base of the desktop application was finished, a supabase instance was created and configured, and a mobile application base was implemented. The system was then extended sprint after sprint until the requirements were fulfilled.

The whole solution developed in this thesis automates a large portion of the process and streamlines the workflow of the technicians; it also implements the requirements defined during the analysis and builds on top of them.



Chapter 7

Future work

The system developed in this thesis is a complete solution for all the identified problems, aiding in making the whole process of data measurement, its collection, visualization, and analysis streamlined, as well as speeding up the delivery of valuable feedback to the athletes and their trainers, while subsequently making the data delivery more secure and the results more readable. All the proposed requirements are fulfilled by this solution, and many of them are also extended, adding more functionality. However, just as with any other solution, there can be more improvements and there can be additional functionality built on top of the current one. Those facts motivate this chapter by highlighting possible future enhancements for both developed applications. Firstly, the mobile application is currently automatically receiving all relevant data without the need for user input, but this only happens when the application is running in the foreground, thus inspiring notifications that could be implemented to inform the user about new data he might want to read through. This could be done via the Firebase cloud messaging platform since Firebase is already connected to the applications and is used in user management. The mobile application could be further extended with more graphical representations of the measurements, supplementing the textual form currently present. Multiple enhancements are proposed: a graph showing values for a selected biomarker and its development over time within an event; another chart could serve as a showcase of trends among all measured data from all events, quickly pointing out the long-time development of selected markers. For the desktop application, a data table could be improved by adding more operations across rows and columns, making some operations even faster. Those actions would have to be consulted with experts from Casri to ensure their usefulness and avoid cluttering the interface.

During late phases of testing a few enhancements were deemed necessary and should be considered high priority in future development. Namely, for some values that were under or over certain threshold, the devices are sending extra characters. The parsers are not expecting those characters and assume those are invalid. This can be solved by finding the extra characters in, for example, logs in the application and altering the parser behavior. Secondly, it was found that values that are whitelisted to be shown in the mobile

7. Future work

application have to be altered and some of them must be blacklisted as they are not to be visible for the consumers. This could be also extended to allow users of the mobile application to pick and choose which values should be visible from those that the application allows.



Bibliography

- [1] Hong Zhu, Patrick AV Hall, and John HR May. “Software unit test coverage and adequacy”. In: *Acm computing surveys (csur)* 29.4 (1997), pp. 366–427.
- [2] Jan Axelson. *Serial Port Complete: The Developer’s Guide*. Lakeview Research LLC, 2007.
- [3] Shona L Halson. “Monitoring training load to understand fatigue in athletes”. In: *Sports medicine* 44.Suppl 2 (2014), pp. 139–147.
- [4] Lukáš Šimon. “Mobilní Flutter aplikace pro řešení výukových úloh v systému EduARd”. Bakalářská práce. České vysoké učení technické v Praze, Fakulta elektrotechnická, katedra počítačů, 2021.
- [5] Richard Rose. *Flutter and Dart Cookbook*. O’Reilly Media, Inc., 2022. ISBN: 9781098119515.
- [6] Nils Haller et al. “Blood-Based Biomarkers for Managing Workload in Athletes: Perspectives for Research on Emerging Biomarkers”. In: *Sports Medicine* (2023), pp. 1–15.
- [7] *AFIAS-1*. URL: <https://www.boditech.co.kr/en/product/instruments/id/1> (visited on 12/01/2023).
- [8] *Auth0 social login*. URL: <https://auth0.com/learn/social-login> (visited on 12/01/2023).
- [9] *Beamer documentation*. URL: <https://pub.dev/packages/beamer> (visited on 12/01/2023).
- [10] *Best practices for building scalable Flutter applications*. URL: <https://verygood.ventures/blog/scalable-best-practices> (visited on 12/01/2023).
- [11] *Bitsdojo Window documentation*. URL: https://pub.dev/packages/bitsdojo_window (visited on 12/01/2023).
- [12] *Bloc documentation*. URL: <https://bloclibrary.dev/#/> (visited on 12/01/2023).
- [13] *Configuration of COM Ports*. URL: <https://learn.microsoft.com/en-us/windows-hardware/drivers/serports/configuration-of-com-ports> (visited on 12/01/2023).

- [14] *Dart Libserialport documentation*. URL: <https://pub.dev/packages/libserialport> (visited on 12/01/2023).
- [15] *Demystifying State Management in Flutter: An In-Depth Exploration*. URL: <https://medium.com/@ubermenschdeveloper/demystifying-state-management-in-flutter-an-in-depth-exploration-a9724af89e7d> (visited on 12/01/2023).
- [16] *FFI documentation*. URL: <https://pub.dev/packages/ffi> (visited on 12/01/2023).
- [17] *Flutter architecture*. URL: <https://docs.flutter.dev/resources/architectural-overview> (visited on 12/01/2023).
- [18] *Flutter Libserialport documentation*. URL: https://pub.dev/packages/flutter_libserialport (visited on 12/01/2023).
- [19] *GetIt documentation*. URL: https://pub.dev/packages/get_it (visited on 12/01/2023).
- [20] *GetX documentation*. URL: <https://pub.dev/packages/get> (visited on 12/01/2023).
- [21] *Go Router documentation*. URL: https://docs.page/csells/go_router/navigation (visited on 12/01/2023).
- [22] *Hive documentation*. URL: <https://docs.hivedb.dev/#/> (visited on 12/01/2023).
- [23] *HiveDB documentation*. URL: <https://docs.hivedb.dev/#/> (visited on 12/01/2023).
- [24] *Lib serial port*. URL: <https://sigrok.org/wiki/Libserialport> (visited on 12/01/2023).
- [25] *Lottie — an open-source animation rendering tool*. URL: <https://medium.com/@MarianaN./lottie-an-open-source-animation-rendering-tool-743e02fc369e> (visited on 12/01/2023).
- [26] *Navigation and routing*. URL: https://docs.flutter.dev/ui/navigation?gclid=CjwKCAiAvdCrBhBREiwAX6-6UkXcm_7CKrHEZq85Fa0IXax-5PzaZjemNiCTo-XtpQe10a0jxi05zhoCM5EQAvD_BwE&gclidsrc=aw.ds (visited on 12/01/2023).
- [27] *Opti CCA-TS2*. URL: <https://www.optimedical.com/en/products-and-services/analyzers/opti-cca-ts2-analyzer/> (visited on 12/01/2023).
- [28] *Provider documentation*. URL: <https://pub.dev/packages/provider> (visited on 12/01/2023).
- [29] *Riverpod documentation*. URL: <https://riverpod.dev/> (visited on 12/01/2023).
- [30] *SPOTCHEM EZ SP-4430*. URL: https://www.arkray.eu/english/products/biochemistry_testing/dry_chemistryclinical_chemistry/sp-4430.html (visited on 12/01/2023).

- [31] *State Management in Flutter: A Comprehensive Guide*. URL: <https://medium.com/@enitinmehra/state-management-in-flutter-a-comprehensive-guide-7212772f026d> (visited on 12/01/2023).
- [32] *Supabase documentation*. URL: <https://supabase.com/docs> (visited on 12/01/2023).
- [33] *SUPER GL Compact*. URL: <https://dr-mueller-geraetebau.de/en/products/super-gl-compact/> (visited on 12/01/2023).
- [34] *The rise of social login*. URL: <https://medium.com/@thepulsewallet/the-rise-of-social-login-bridging-the-gap-between-convenience-and-security-a3497abc902e> (visited on 12/01/2023).
- [35] *Unit testing in flutter*. URL: https://medium.com/@Ikay_codes/unit-testing-in-flutter-19dea7214c7b (visited on 12/01/2023).
- [36] *User testing*. URL: <https://www.toptal.com/designers/prototyping/user-testing-prototypes> (visited on 12/01/2023).
- [37] *Why we need asyncvalue of riverpod*. URL: <https://chooyan.hashnode.dev/why-we-need-asyncvalue-of-riverpod> (visited on 12/01/2023).
- [38] *Win32 documentation*. URL: <https://pub.dev/packages/win32> (visited on 12/01/2023).

Appendix A

Installation process

To run and possibly deploy the applications it is necessary to follow the following instructions:

1. Download the codebase and unpack it.
2. Create an instance of Supabase either local or a remote one in the cloud. Specific instructions may change and can be found in the official documentation ¹.
3. Create tables from the schemas located in resources folder.
4. Add Firebase to the mobile application. For specific instructions, please follow the official documentation ².
5. Configure social sign-in for the mobile application. For specific instructions, please follow the official documentation ³.
6. Configure both applications for supabase integration. Specific instructions are in the official documentation ⁴.
7. Get dependencies with the get command ⁵.
8. Generate files in both the mobile application and desktop application with the build command ⁶.
9. To deploy the applications, please follow the instruction from the official documentation ⁷.

¹<https://supabase.com/docs/guides/getting-started>

²<https://firebase.google.com/docs/flutter/setup?platform=android>

³<https://firebase.flutter.dev/docs/auth/social/>

⁴<https://supabase.com/docs/guides/getting-started/quickstarts/flutter>

⁵`flutter pub get`

⁶`dart run build_runner build --delete-conflicting-outputs`

⁷<https://docs.flutter.dev/deployment>

■ Requirements

To be able to run the applications locally, the following requirements need to be met:

- Docker installed. Can be done from official page ⁸.
(Developed with version 4.25.2)
- Flutter installed. Can be done from official documentatation ⁹.
(Developed with version 3.16)
- Dart installed. Comes bundled with Flutter.
(Developed with version 3.2)

⁸<https://www.docker.com/>

⁹<https://docs.flutter.dev/get-started/install>