Habit Tracker

Desktop application for effective time management

Report

Aleksandra Wiktoria Bruska, 202400723
Cristina Semikina, 202400720
Jan Jakub Walczak, 202400722
David Fuchs, 202400725

Supervisor: Jørn Martin Hajek

Aarhus University
13 December 2024

~60000 characters

Summary

Written by Aleksandra Bruska

This report presents the development of Habit Tracker, a desktop application designed to facilitate monitoring and tracking the completion of daily habits and tasks. The aim of the project was to provide a prototype version that covers the essential features, which include creating habits, monitoring the progress and obtaining feedback. Moreover, the application should communicate with external components, including database, email system and calendar system.

The primary purpose of the report is to serve as a reference for understanding the design of the application and iterative development process. It provides the initial assumptions and requirements as well as explains how they change over iterations. It focuses on explaining the reasons for decisions and changes that were made. Moreover, the report explains how the software components are designed, implemented and tested. Future development directions are mentioned in order to highlight the project's utilization opportunities. In addition to the description of the technical part, the report also describes the methodology and main challenges encountered by the group members.

Table of contents

Foreword

Written by Aleksandra Bruska

This report was written for the Project Work course at Aarhus University by four exchange students of the International Semester in Software Engineering. The group consisted of Aleksandra Bruska and Jan Walczak, 5th semester students of Computer Science at Gdansk University of Technology, Cristina Semikina, 5th semester student of ISEP - Instituto Superior do Porto and David Fuchs, 7th semester student of Industrial engineering at HTW Berlin. The supervisor of the project was Jørn Martin Hajek. The subject of this report is related to the development of a desktop application for tracking habits. The group worked on the project in the winter semester of 2024/2025 and the submission date was 13th of December 2024.

Introduction incl. problem statement

Background

Written by Aleksandra Bruska

Introducing and maintaining positive habits is a key to self-development and achieving long-term goals. However, due to lack of consistent motivation, many people struggle to develop good routines in their everyday life. Traditional techniques, like notes or journeys often turn out to be insufficient as they do not provide any feedback and do not encourage regularity.

Problem statement

Written by Aleksandra Bruska

How can we help people introduce and maintain positive habits in everyday life?

- How to facilitate habit tracking using a desktop application?
- What are the most important features that the application should provide?
- How to ensure that the application is simple and user-friendly?
- How to provide encouraging and motivating feedback?

Goal

Written by Aleksandra Bruska

The intention of this project is to provide a prototype of an user-friendly desktop application that tracks user's habits and facilitates effective management of free time. It will be used to test the concepts and obtain feedback. It will offer limited functionalities, which should be sufficient to convey the main idea. The application will require connection with a local database stored on the computer.

Requirements

User stories

Written by Aleksandra Bruska (list of user stories was created by the whole group)

At the beginning of the process, we met with the whole group and brainstormed in order to discuss functional requirements that should be provided by the application. Some of them were inspired by other applications that we used in the past. For example, the Streak on Ice was inspired by “Duolingo” and connecting with Google Calendar was inspired by reclaimai. Then we created a list of user stories, which is presented below.

US #1: As a user, I want to sign up with my first name, login, email, password and repeated password.

The application should provide a form for singing in and a form for creating an account. The account details should be requested when creating an account, stored in a database and checked when the user sings in. The form should also be validated to ensure that the data entered by the user is correct (ex. email has an adequate format). Optionally a welcome email could be sent to a user.

Estimated task size: Medium

US #2: As team members, we want to have a nice home screen for our app as a starting point

This requires designing a home screen and then implementing it as a part of the application using JavaFX.

Estimated task size: Medium

US #3: As a user, I want to add my own customized habit.

The main screen of the application should provide a button that triggers a pop-up window with a form allowing users to enter details about the new habit. It needs to be connected with the database so that the new habit is stored and can be displayed later.

Estimated task size: Medium

US #4: As a user, I want to edit personal details on my account (name, surname etc.)

The application should provide a profile view. It should display the information about the user such as name, surname, username, email etc. It should also allow for editing some of those details. The new information should be stored in the database. Optionally, the information could also contain a profile picture and allow for changing it to another image stored on the user's computer.

Estimated task size: Medium

US #5: As a user, I want to connect the app to Google Calendar

The user should be able to connect their Google Calendar to their account. The application should then synchronize the habit data entered by the user with the calendar by finding available time slots and filling it with planned habits performance.

Estimated task size: Medium

US #6: As a user, I want to track my habit with checkboxes

The home screen of the application should include a section with checkboxes that users could use to mark the tasks as done.

Priority: High
Estimated task size: Medium

US #7: As a user, I want to use Streak On Ice, to be able to skip the habit once

The home screen should contain a screen section with a streak counter that would display the information about the number of days in a row that the task was completed by the user. It should also contain an additional "Streak on Ice" feature that would allow the user to skip the task once without losing the streak.

Estimated task size: Small

US #8: As a team member, I want to store the data in the database.

This requires designing and implementing a database as well as connecting it with the desktop application. Users should be able to access their account details without a specific knowledge of how they are stored. Possible queries that users might want to execute are: creating an account and logging to it, deleting an account, setting and changing personal details, like: name, e-mail, age, profile picture etc., creating new habits and deleting old ones, setting and completing goals.

Estimated task size: Large

US #9 As a team member, I want to access the database easily using dedicated services provided as Java classes.
This requires implementing several entity classes and services with public interfaces that could be used by other parts of the application. It would also be a good idea to prepare simple examples of how those methods should be used.

Estimated task size: Medium

US #10: As a User, I want to have an application free from bugs and errors

To ensure that the application is free from unexpected behavior and errors, all major components should be tested using unit tests.

Priority: High
Estimated task size: Medium

US #11: As a user, I want to be able to measure the amount of time spent on performing the habit.

The application should contain a timer that would be started and stopped manually by the user. The amount of time spent on the activity should be stored in the database.

Estimated task size: Medium

US #12: As a user, I want to obtain extensive feedback about my performance.

The application should analyze the data using statistical tools or machine learning. The result should then be displayed as a short weekly summary. The application should also award the users with badges for their achievements.

Estimated task size: Medium

US #13: As a user, I want to be able to connect with my friends using the application.

The application should allow the user to add new friends and store the list in a database. The user should be able to set their habits as either private or public and view their friends' public habits performance. Optionally the users could create common goals and work on them together.

Estimated task size: Large

Non-functional requirements

Written by Aleksandra Bruska

The main non-functional requirements for the application are:

- Usability: User Interface should be simple, intuitive and user-friendly, so that it enables a first-time user to complete core actions like adding a new habit within 5 minutes of app interaction.
- Maintainability: To ensure that the features are isolated, the application should consist of several components with no single module exceeding 500 lines of code.
- Data integrity: The database should use mechanisms such as constraints and data types to ensure that 100% of stored entries conforms to predefined formats.
- Testability: All components should support testing. Unit tests should cover at least the crucial components described later: Login Controller, Create Account Controller, Dashboard Controller, Profile Controller.

Workload division

Written by Aleksandra Bruska

The following tasks were assigned to the individual members of the group:

- **David Fuchs** - creating a design of the home screen and implementing it; providing features related to habits tracking, including creating a new habit, registering performance data with checkboxes and streak counting with additional "Streak on Ice" feature

  User stories: #2, #3, #6, #7

- **Cristina Semikina** - providing components that allow communication with Google Calendar; implementing features that rely on Google Calendar, including finding available time slot and filling it with data provided by the habit tracking application; preparing unit tests

  User stories: #5, #10

- **Jan Jakub Walczak** - planning and creating a relational database in SQL that stores habits and user data; implementing components that connect the application with MySQL database; providing examples of use and helping team members with any concerns about the database; helping with implementing the email system.

  User stories: #8, #9

- **Aleksandra Wiktoria Bruska** - providing “sign in” and “create an account” functionalities; implementing the email component that communicates with the e-mail system; implementing the profile screen that allows for displaying and editing information from the database

  User stories: #1, #4

Delimitation

Minimum viable product

Written by Aleksandra Bruska (MVP requirements were discussed by the whole group)

Once we created the list of user stories that we would ideally like to cover, we focused on the minimal viable product version. We discussed the most important functionalities that are necessary for the habit tracking application to be usable. We decided that the essential features for the MVP version are:

- **Habit creation**: The user should be able to create a habit. The simplest version of a habit would have a name without any additional description or category.
- **Tracking one habit with checkboxes**: The user should be able to enter their performance to the application. The easiest way is to use checkboxes to indicate that the habit was performed on a particular day..
- **Streak on ice:** This is the simplest version of a feedback from the application about the user's performance. The idea is to count the number of days in a row when the habit is performed. It should also allow the user to skip the habit once without losing the streak.

Prioritizing user stories

Written by Aleksandra Bruska (prioritizing was discussed by the whole group)

In order to decide which user stories we should focus on first, we divided them into three categories: high, medium and low priority. When developing the application, we first focused on the user stories with the highest priority.

High priority

We decided that the highest priority should be assigned to the user stories that are critical for providing the minimum viable product discussed above. Those are:

- As team members, we want to have a nice home screen for our app as a starting point (US #2)
- As a user, I want to add my own customized habit (US #3)
- As a user, I want to track my habit with checkboxes (US #6)
- As a user, I want to use Streak On Ice, to be able to skip the habit once (US #7)

Moreover, the requirement that must be satisfied for those user stories to be implemented is to have a working database connected to the application. This is related to the following user stories:

- As a team member, I want to store the data in the database (US #8)
- As a team member, I want to access the database easily using dedicated services provided as Java classes (US #9)

Another issue of great importance is to make sure that failures do not make the application unusable. The user story related to that is:

- As a User, I want to have an application free from bugs and errors (US #10)

Medium priority

This category was assigned to the user stories that are important, but not critical for the application. One important issue is to allow the users to have a personal account in the application so that they can manage and customize their profile as well as access their habit data from different devices. The related user stories are:

- As a user, I want to sign up with my first name, login, email, password and repeated password (US #1)
- As a user, I want to edit personal details on my account (name, surname etc.) (US #4)

Another feature that could add significant value to the product would be to connect the application with Google Calendar. While it is not necessary for the basic functionality of the application, it would significantly enrich the experience and distinguish the application from others. The user story is:

- As a user, I want to connect the app to Google Calendar (US #5)

Low priority

Low priority was assigned to the user stories that could enhance the product, but are insignificant and can be postponed for the future versions of the application. They concern the issues of providing better feedback, facilitating entering the performance data and allowing for interaction between users. The user stories related to that are:

- As a user, I want to be able to measure the amount of time spent on performing the habit (US #11)
- As a user, I want to obtain extensive feedback about my performance (US #12)
- As a user, I want to be able to connect with my friends using the application (US #13)

Due to the limited amount of time and low number of team members, we only managed to implement the most important requirements that were assigned a high or a medium priority.

Method and process

UML

Written by Aleksandra Bruska

We created UML class diagrams to visualise the domain model of our project. It helped us get a better understanding of the relationships between entities and make design decisions at the early stages of development.

C4 model

Written by Aleksandra Bruska

To plan and document the structure of our project, we used the C4 model. We created context, component and container diagrams, not including the code diagram. It allowed us to focus on different levels of abstraction. Since we created it at the beginning of the project work, we could use it during the development to avoid confusion about which parts and levels we are focusing on at a given time.

Group formation

Written by David Fuchs

The group was primarily formed due to the limited number of participants in the course. Initially, six people registered, but one person never attended, and another decided to withdraw after the first week. This left the four of us to form the group. Among us, Jan and Aleksandra already knew each other from their studies at Gdańsk University of Technology; however, the rest of us had no prior acquaintance.

Collaboration Agreement

Written by David Fuchs

We created the collaboration agreement together during the process, as we were able to address differences that had arisen in the preceding weeks and discuss them as a team. In hindsight, it would have been better to establish the agreement from the very beginning. This would have allowed us, especially as a team from different countries and cultures, to exchange expectations early on and discuss what constitutes effective teamwork and what matters most to each of us. Additionally, having the collaboration agreement in place from the start would have given us the opportunity to use our retros iteratively to address certain topics in a safe environment and, if necessary, adjust the agreement over time.

The main improvements we noticed after finalizing the collaboration agreement included better punctuality at meetings and more effective use of our project management tool. We also made a point of documenting aspects of our teamwork that were already working well, which helped reinforce positive practices within the group.

The collaboration agreement is provided in the appendix to this report.

Development process

Written by David Fuchs

We chose Scrum as our project management framework because some of our team members had already gained experience with it in previous software development projects. Additionally, Scrum is widely used in the software development industry due to its iterative approach and focus on continuous improvement, making it a fitting choice for our project.

That said, we made some adjustments to the traditional Scrum methodology to better suit our specific circumstances. Since we didn't have an external customer and were working in a very small team, certain roles and processes were adapted accordingly.

Project management

Written by David Fuchs

Alexandra took on the role of the Product Owner. Typically, this role involves managing the product backlog, prioritizing features to maximize the economic added value for the customer of a product to be developed, and acting as the key point of communication between the team and the customer. In our case, since we didn't have a customer, Alexandra's primary responsibility as Product Owner was to facilitate communication with our supervisor, ensuring that their feedback and guidance were integrated into our project.

David assumed the role of Scrum Master, as he had previous experience working with Scrum and using tools like Notion in earlier projects. The Scrum Master is traditionally responsible for ensuring that the team adheres to Scrum principles, facilitating Scrum events (such as sprint planning, daily stand-ups, and retrospectives), and removing any obstacles that could hinder the team's progress. In practice, Scrum Masters often take on dual roles, either contributing as developers or overseeing multiple teams. In our case, due to the small size of our team, David also actively participated in the software development alongside his Scrum Master responsibilities.

By adapting the Scrum framework to our team's unique needs, we were able to effectively manage our project while still benefiting from the core principles of iterative development and collaboration.

How to plan

Written by David Fuchs

The collaboration resulted in 13 user stories with various subtasks using Agile/ SCRUM methodology done through five two-week sprints. User stories were developed to guide the team in sprint planning and assign tasks to each member.

The SCRUM board, which is arranged according to user stories developed by the team, is also stored in Notion, serving as the main guide for accomplishing tasks in Agile Project Management.

After the first sprint, we admittedly got somewhat lost in the coding phase, often discussing our tasks and progress informally during co-working sessions. However, by the third sprint, we realized that this approach sometimes led to miscommunication and misunderstandings. As a result, we collectively decided to improve our process by consistently maintaining and using our Scrum Board. This change allowed us to coordinate more effectively and clearly see what each team member was working on, even when we weren't in the same room.

During this adjustment, we also revisited and refined some of our user stories. For example, we recognized that to make our habit tracker more universally usable, we needed to keep it as simple as possible. This meant avoiding specific metrics such as the number of kilometers run or liters of water consumed. We adapted our Scrum Board to link tasks directly to their respective user stories and assigned a responsible team member to each task. In our sprint planning sessions, we collaboratively created new user stories and iteratively updated older ones, particularly when we realized that certain features were no longer needed. For each user story, we established a priority and provided an effort estimation, considering uncertainty for tasks we had never tackled before. We also tagged each user story with its respective sprint and added rollups, which visually displayed 100% completion when all related tasks were marked as "in review" or "done."

Our Scrum Board was organized as a Kanban board with the following sections: Product Backlog, Sprint Backlog, In Progress, Review, and Done. Additionally, we introduced two useful views for the Scrum Board:

1. **Ownership View**: This provided an overview of all tasks assigned to each individual team member.
2. **User Story View**: This grouped all tasks belonging to a specific user story for clarity.

For our user stories, we implemented two additional views:

1. **Current Sprint View**: This displayed all user stories within the current sprint.
2. **Archived View**: This listed all archives user stories which we didn't need anymore .

These enhancements significantly improved our workflow, making our progress more transparent and aligned, while also enabling us to adapt efficiently to changes in requirements or priorities during the project.

Project administration

Written by David Fuchs

For internal communication, we used a WhatsApp group to organise meetings and discuss things. When we had an online meeting, we met on Discord, so we also had the possibility to share our screen so that we could discuss things better. We use retrotool.io for our retrospectives as David has already had positive experiences with this online tool from a previous project. In addition, retrotool offers a free version with which we were able to export our ideas as a Markdown file to import them into Notion as well.

Meetings

Written by Jan Walczak

We agreed on meeting for co-working sessions on each Friday at 9.30. Usually, every participant showed up and we could discuss problems and help each other.. After each session, one person was responsible for creating notes of how the meeting went and what person did what at that specific meeting.

When we were ready with milestones, we asked our supervisor for a meeting to discuss our current progress. Those meetings usually took place on Mondays between 11.00 and 15.00 and lasted for approximately 30 minutes. Not every person could show up on them, so before each of those meetings we gathered some topics together that should be mentioned.

Conflict management

Early problems

Written by Jan Walczak

As international students, early work on the project has arisen small difficulties from the beginning. We had to find every person who was supposed to be involved in this class. We had sent many emails to possible participants, who had not shown up on the first meeting. With the help of our supervisor, we finally managed to meet up as a group and start working on our project. We talked about our expectations and goals that we would like to achieve.

Conflicts during development

Written by Jan Walczak

Our expectations occurred to be conflicted, some people were not satisfied with how the workload was distributed. Moreover, some people fell behind with updates and had to catch up.

We planned and conducted two dedicated meetings to address those problems. In the first meeting, we discussed our expectations and searched for solutions to our problems. We used RetroTool. The details about the tool and our use of it can be found in the appendix to this report. In the second meeting everyone had to present what they had worked on so far, so people could catch up with the current progress of the project. We made notes to help us work on the report. We have also agreed on changing our collaboration agreement, which we refactor based on most voted thoughts from Retrotool.

Analysis

Written by Aleksandra Bruska

This section describes the most important considerations we had about the software solutions and justifies the decisions we made.

GUI framework: JavaFX

Written by Aleksandra Bruska

When discussing a framework choice for our graphical interface, we were considering two popular Java frameworks: JavaFX and Java Swing. We compared them and came to conclusions:

- Java Swing is considered a legacy technology *(Salo, 2024)* , while JavaFX is a modern and still developing tool. Therefore, Java Swing has a large community of developers and there are many sources available to facilitate learning and solve problems. Its ecosystem has extensive documentation, and better backward compatibility with older Java applications *(ayaan07/GeeksForGeeks, 2024)* .
- Due to being more modern, JavaFX offers richer features, such as a modern and extensive UI component set. It also provides more advanced tools for building visually appealing user interfaces, including CSS styling and FMXL, an XML-based language to define the user interface separately from the business logic *(ayaan07/GeeksForGeeks, 2024)* .
- JavaFX can be used with the Scene Builder, which is a free and open source tool for designing user interfaces. It facilitates the process of creating aesthetic application screens.

Since we did not need our application to be compatible with any legacy systems, we decided to use JavaFX, as it provides many useful tools. We also believe that learning modern technologies might be beneficial for us and useful in the future.

Database API: Java Persistence API

Written by Aleksandra Bruska

For database support, we considered two approaches: Java Database Connectivity (JDBC) and Java Persistence API (JPA). After comparing both tools, we noted that:

- JPA provides a higher level of abstraction than JDBC. (*Fadatare, n.d)*
- JDBC allows for handling queries of the SQL to the database. On the contrary, JPA allows for skipping SQL queries.
- JDBC requires manual handling of database connections, while JPA automates and abstracts much of the database interaction. (*Fadatare, n.d)*
- Unlike JDBC, JPA is built around object-relational mapping, which allows for a more object-oriented approach to persistence and simpler development. (*Fadatare, n.d)*

After careful consideration, we decided that JPA is more suitable for our needs as it has many advantages and allows for easier code maintenance.

Other technologies used

Written by Aleksandra Bruska

This paragraph is devoted to other technologies used in our project. Unlike the technologies described above, the choice of those tools was simpler, either due to the dominance of the chosen solutions on the market or lower significance of those decisions for our project. Therefore, the decision-making process was less extensive and does not require a broad description:

- Git and GitHub - as a version management tool,
- Scrum Board Notion - to support iterative development process,
- SQL language - for creating a database,
- MySQL - as a database management system.

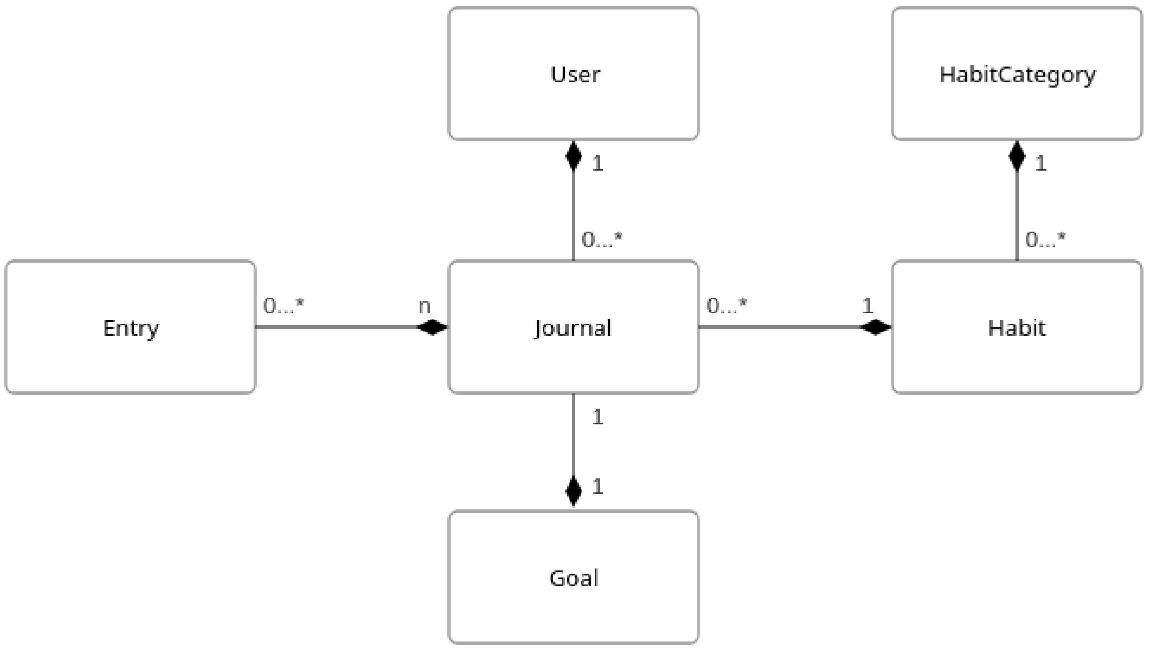
Architecture

Domain model

Written by Jan Walczak

At first, we planned that the entities would be:

- User: Represents a user of the application. User creates an account, logs in, adds habits and uses checkboxes to indicate their performance.
- Habit: Represents a single habit that can be performed, like reading books or eating healthy. It has a name and description as well as a category.
- HabitCategory: Stores the information about the general category a habit can belong to with a short description.
- Goal: Represents the target performance of the habit. For example, if a habit is about running, then the goal might specify the distance that should be run every day.
- Entry: Stores the information about a single session of performing a habit with a date, a value, like distance of the run, as well as the journal it belongs to.
- Journal: Stores information about the user's performance of a particular habit. Holds the information about the user whose performance is checked, a habit that is being performed and a goal that specifies the target performance.



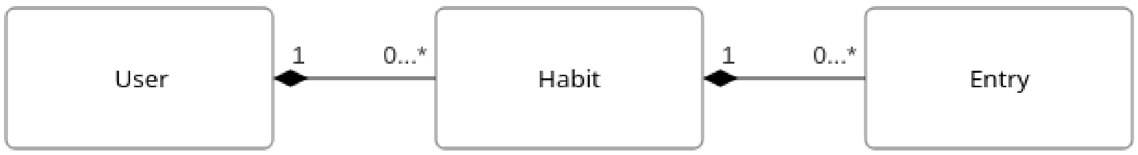
UML class diagram for initial design of domain model

All of those classes were created as JPA entity classes. Names correspond to table names in the database which was fairly easy to implement using JPA. However, managing relations between so many classes, compared to the small amount of data each of them held, ended up complicated. Sometimes adding data to a table required us to manually check and remember entities reference and id’s. For example, when we wanted to add a simple habit to a user we needed following:

1. Activity type – which was supposed to store a simple description of what activity would be about
2. Habit – which needed its own fields: description, name and reference to a habit category
3. Habit category – which needed another name and description
4. User – which needed to be queried from a database or added manually with all its data
5. Goal – which needed a value (unit of measurement) and another description

As we have noticed, such an amount of queries and data was not needed to allow users to create theirs new journals. Despite a lot of queries to the database, we also needed to manage this data inside our code in JPA, as each wrong query threw a custom exception.
The second problem of this design was deleting data. JPA offers a “delete on cascade” method, but we managed to implement this functionality on our own. This led us to even more queries, exceptions and unnecessary effort to manage data properly.

Despite being implemented properly, either in SQL and our application, this effort led us to coming up with another, simpler design, which was supposed to lower the amount of queries and data stored.



UML class diagram for final design of domain model

First goal of this design was to cut a number of references needed to be held in each of the tables. Only two references were held: “User” inside the Habit table, and “Habit” inside the Entry table. Immediately, we needed to remember less data to create new habits for users.

The second advantage of this design, which solved another problem, was deleting data. Smaller amount of references required less amount of deleting methods, which was once again implemented manually, instead of using “delete on cascade”.

Smaller amount of unnecessary data and references allowed us to create habits for users faster and easier. For example, when we wanted to add a habit to a user we needed following:

1. User – which needed to be queried from a database or added manually with all its data
2. Habit – which needed its name, as streak counters was initially set to 0.

As entries were linked directly to a habit, “Habit” table was now considered a “Journal” table from our previous design, as it fulfilled its requirements. We also introduced new methods for managing a given habit, to easily edit its fields, for example incrementing and setting streak counters, which made this approach even more convenient for us to use.

Finally, we have noticed that this design is not perfect. Each table needed bigger chunks of data, which were previously divided between more tables which makes this design less flexible for changes. We came to a conclusion that our new design is better for our requirements and User stories, as it is less complicated, but it should likely change if our application grew in size.

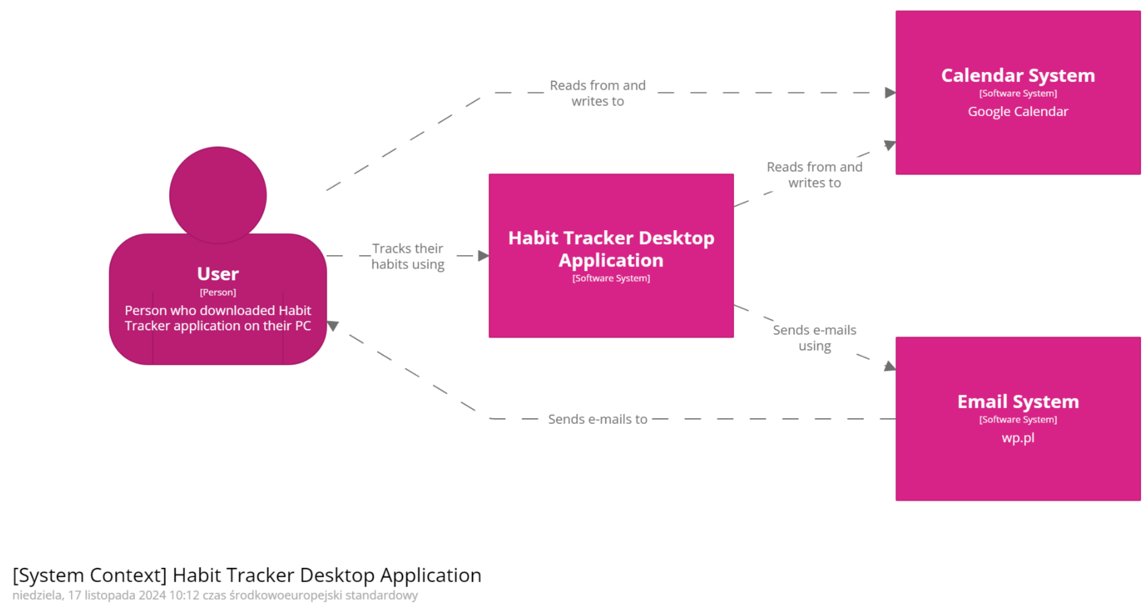
C4 model

Context Diagram

Written by Aleksandra Bruska

The important elements of the context are:

- Habit Tracker Desktop Application - application that tracks User's habits, inserts performance data in the Calendar System and send emails to the User using the Email System
- User - end user that interacts with the application using User Interface. They create accounts, sign in, crete habits and enter data about their performance
- Calendar System - Google Calendar that is synchronized with Habit Tracker. It provides the application with information about free time slots and allows for entering data about planned activities.
- Email System - wp.pl email system that allows for sending emails from the application to users who created an account.



Calendar System
[Software System]
Google Calendar
Reads from and writes to
Reads from and writes to
User
[Person]
Person who downloaded Habit Tracker application on their PC
Tracks their habits using
Habit Tracker Desktop Application
[Software System]
Sends e-mails using
Email System
[Software System]
wp.pl
Sends e-mails to
[System Context] Habit Tracker Desktop Application
niedziela, 17 listopada 2024 10:12 czas środkowoeuropejski standardowy

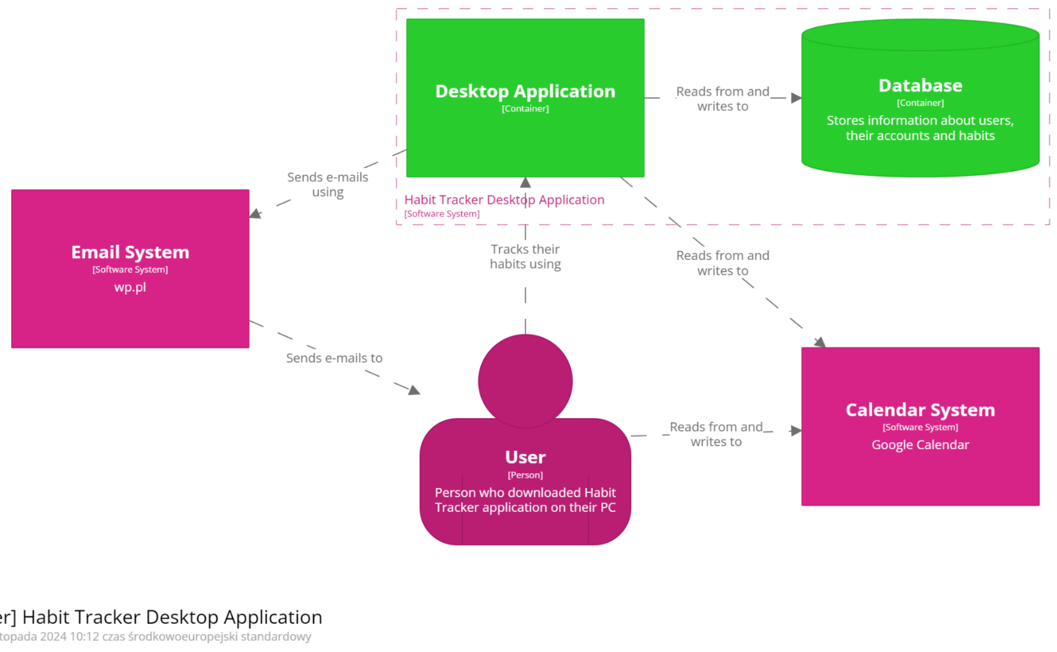
C4 model: Context diagram

Container diagram

Written by Aleksandra Bruska

The most important containers are:

- Desktop Application - application that provides habit tracking functionalities for the Users and interacts with both Calendar System and Email System. It also uses a database to save and read data.
- Database - relational database responsible for storing and managing application data: user information and habits performance results.


Desktop Application
[Container]
Reads from and writes to
Database
[Container]
Stores information about users, their accounts and habits
Habit Tracker Desktop Application
[Software System]
Sends e-mails using
Email System
[Software System]
wp.pl
Tracks their habits using
Reads from and writes to
Sends e-mails to
User
[Person]
Person who downloaded Habit Tracker application on their PC
Reads from and writes to
Calendar System
[Software System]
Google Calendar
[Container] Habit Tracker Desktop Application
niedziela, 17 listopada 2024 10:12 czas środkowoeuropejski standardowy

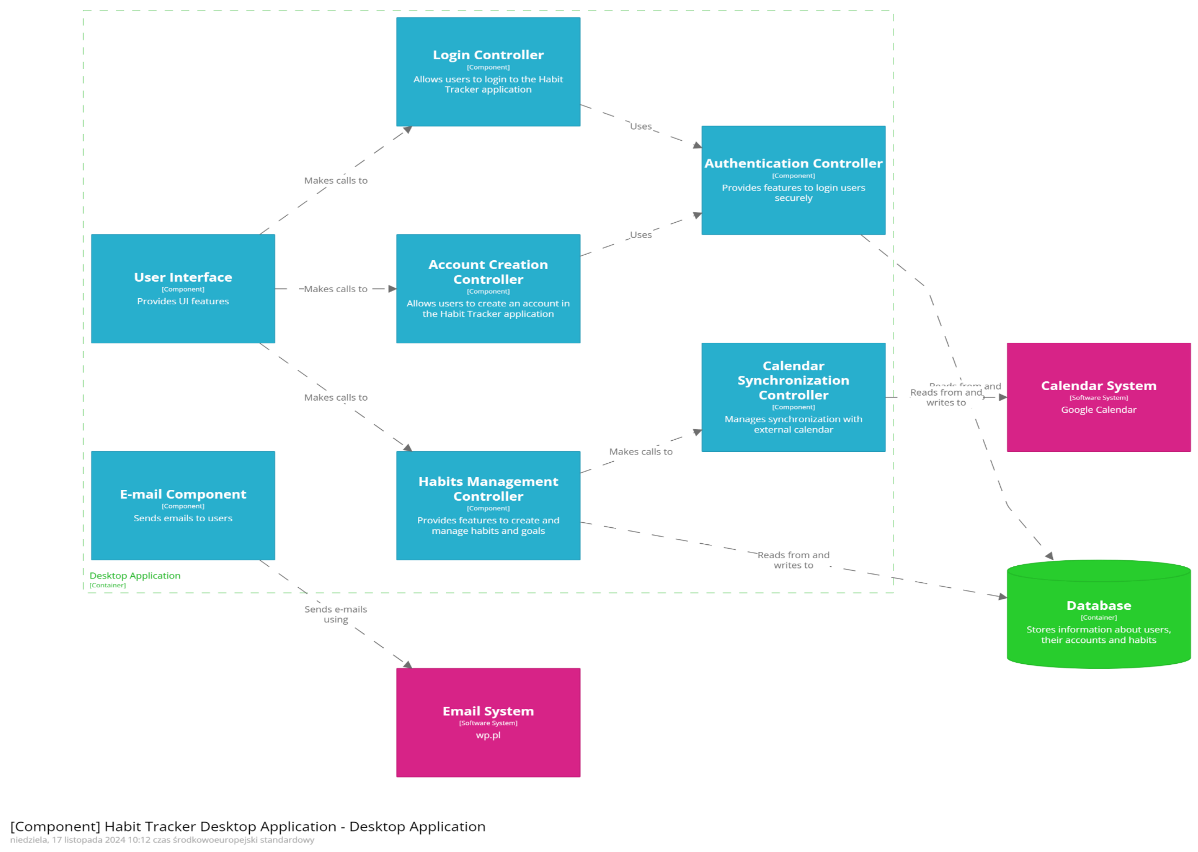
C4 model: Container diagram

Component Diagram

Written by Aleksandra Bruska

The important components of the application are:

- User Interface - UI features displayed to the user
- Email Component - responsible for communication with the Email System in order to send emails to the User
- Habits Management Controller - provides functionalities connected with tracking habits, including creating and deleting habits and entering performance information using checkboxes; implemented as Dashboard Controller
- Login Controller - allows the User for signing in to an existing account
- Account Creation Controller - responsible for creating User's account on Habit Tracker application
- Authentication Controller - provides authentication using password to the account
- Calendar Synchronization Controller - responsible for communication with the Calendar System


Login Controller
[Component]
Allows users to login to the Habit Tracker application
Uses
Authentication Controller
[Component]
Provides features to login users securely
Makes calls to
Uses
User Interface
[Component]
Provides UI features
Makes calls to
Account Creation Controller
[Component]
Allows users to create an account in the Habit Tracker application
Calendar Synchronization Controller
[Component]
Manages synchronization with external calendar
Reads from and writes to
Reads from and writes to
Calendar System
[Software System]
Google Calendar
Makes calls to
Makes calls to
E-mail Component
[Component]
Sends emails to users
Habits Management Controller
[Component]
Provides features to create and manage habits and goals
Reads from and writes to
Desktop Application
[Container]
Database
[Container]
Stores information about users, their accounts and habits
Sends e-mails using
Email System
[Software System]
wp.pl
[Component] Habit Tracker Desktop Application - Desktop Application
niedziela, 17 listopada 2024 10:12 czas środkowoeuropejski standardowy

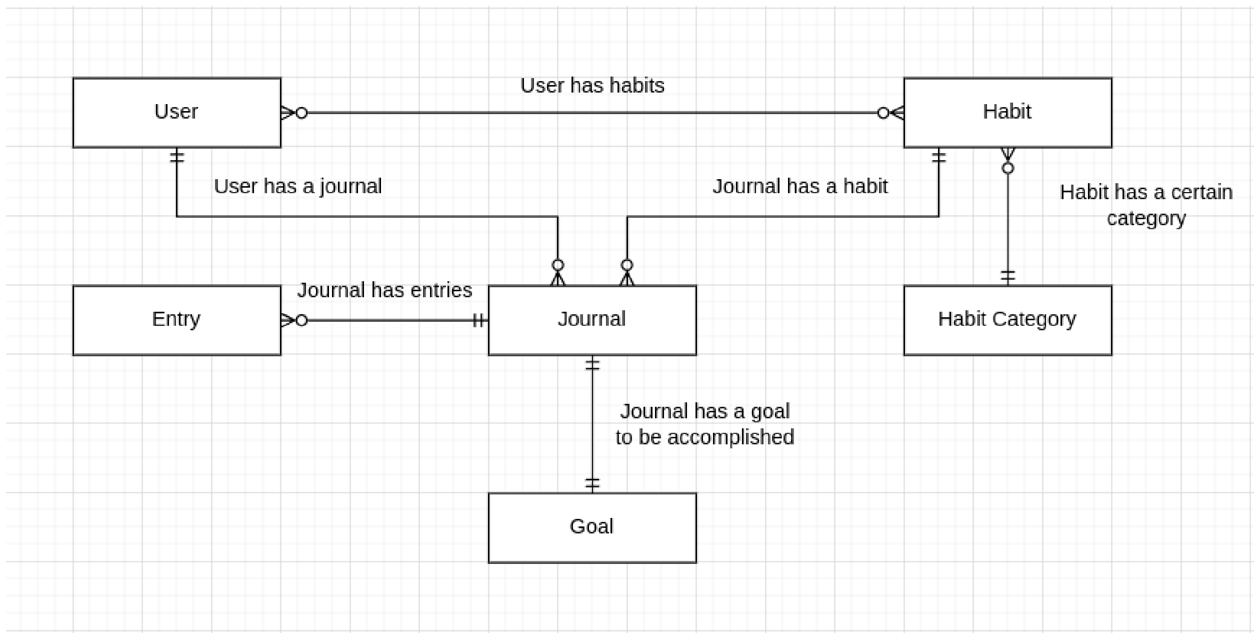
C4 model: Component diagram

Architecture of the database

Written by Jan Walczak

There should be a separate package for a database. Other packages should be able to send queries to the database package and get simple responses.

Early design of a database structure should look like this.

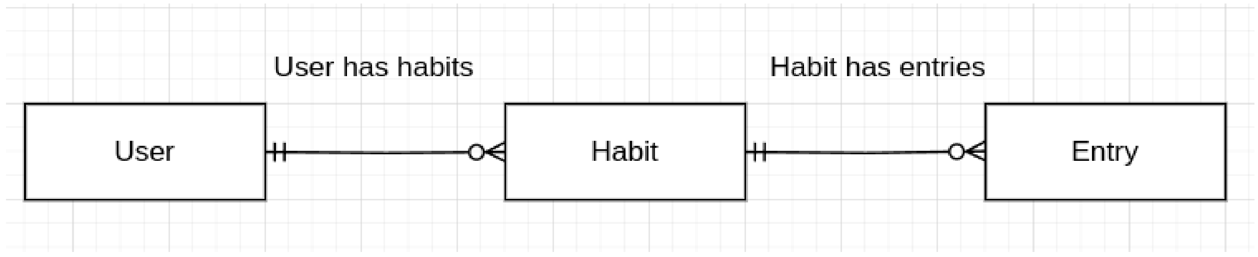


Relationship diagram for initial design of database model

For the prototype purpose, data would be stored on the user's localhost using their own Database Management System, which is considered as hardware since data is saved on a hard drive. Our software would be able to work with this system properly.

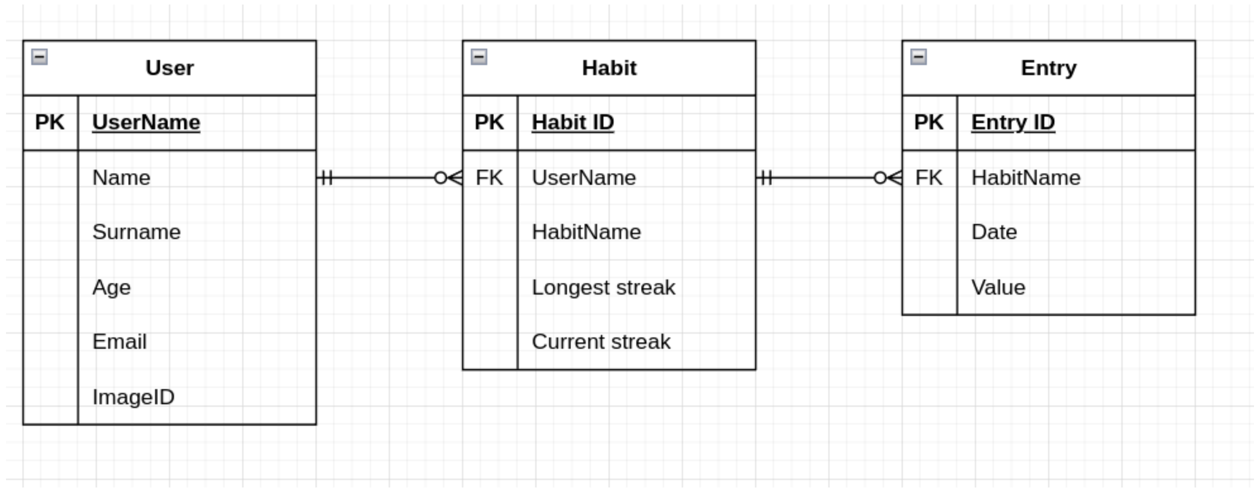
Each table in the database is supposed to have its own class and a service class in the application package, so that other packages could call them easily. For the purpose of deleting entries a 'deleter' service is required, so that references inside the database would not get lost at some point.

As we progressed, we all noticed that this design may be too complicated for the small amount of data which we need to store. We created a simpler design.



Relationship diagram for final design of database model

This design lowers the number of needed tables, but increases the amount of fields needed for each one. A “Habit” table is responsible for holding all details that the previous “Journal”, “Goal”, “Habit category” and “Habit” tables did before. It also eliminates a problem with many-to-many relationship between a “User” and a “Habit” from previous design, so extra connecting table is not needed.



	User
PK	**UserName**
	Name
	Surname
	Age
	Email
	ImageID

	Habit
PK	**Habit ID**
FK	UserName
	HabitName
	Longest streak
	Current streak

	Entry
PK	**Entry ID**
FK	HabitName
	Date
	Value

Table content for final design of database model

Design

Design of the components

Written by Aleksandra Bruska

Most of the provided components are split in two parts:

- The View - Handles the presentation layer. It consists of the FXML file that defines how the user interface elements are displayed to the user. The naming convention for those files is “[component_name]-view.fxml” and they are stored in the resources directory in the project. The presentation layer also includes CSS files that are used by the FXML views to ensure the aesthetic appearance of the UI elements. To facilitate the working process, we did not work with the FXML files directly - instead we used the Scene Builder tool.
- The Controller - handles the business logic of the component. It processes user input from the view and makes sure that the view responds to the user’s actions. It consists of java files. The naming convention for those files is “[ComponentName]Controller.java” and they are stored in the java directory in the project.

This division applies to most of the components. However, since the Email Component does not require the views, only the controller is provided.

What is more, the controllers communicate with the data access layer, that is implemented by the java files in the database directory of the project. It is provided in the form of service files that provide a public interface. Those files perform all necessary operations on the database including reading and saving data concerning users and their habits. The directory also contains java classes for entities stored in the database. More information about the services and connection of the database with the rest of the project is provided in the implementation part of the project.

Implementation

Programming language

Written by Aleksandra Bruska

When it comes to creating software, one of the first decisions we had to make was about the programming language we would use for our desktop application. Technologies that are currently the most widely used for this purpose are C# and Java. We compared them and finally chose Java, version 21.0.5. The main reasons for that were:

- While C# applications are primarily limited to the Windows platform, Java can run on any operating system with a Java Virtual Machine. It allows the creation of cross-platform software without any additional effort.
- Java is object oriented and has an uncomplicated syntax, which makes it easy to use.
- We are aware that Java provides useful tools for network programming, developing desktop applications and working with databases. Some members of our group have used them before.

As a project management tool, we chose Apache Maven 3.9.6.

Connecting the database

Written by Jan Walczak

Initially, we tried to create a database package directly in our project. To do so, we needed to define persistence unit and add following dependencies:

- MySQL
- Hibernate
- Jakarta.persistence

(*Hibernate and Jakarta documentation*)

Database itself was created on the localhost using SQL language, and imported to the project using IntelliJ's database tool. By doing so, we managed to create needed @Entity classes. When we tried to instantiate library classes some errors occurred. We were not sure if those errors are associated with dependencies, maven's version or dependencies themselves. To distinguish that, we have created another project dedicated only to the database, with the intention of connecting it as a .JAR file to our application after fixing those errors. After completing developing this separate project, we noticed that database and use of library functions worked perfectly fine, so the problem was dependencies. Finally we managed to connect the database as a separate package to our application by declaring needed requirements in the "module-info.java" file, which was the problem from the beginning.

For easy use of JPA we created a services' package inside the database package. Its purpose was to allow other parts of the application to get needed data easily and secure invalid queries. Every @Entity class was associated with its specific service. Each service consisted of some methods like "get", "set", "getAll" etc. If the query from another module was invalid, for example entry in the database did not exist, each method in services was supposed to catch those errors and throw a custom exception.

For deleting data a separate "DeleterService" was created. Creating it was a way to handle erasing references that existed in the database between tables. The way this service worked could be compared to "delete on cascade" in SQL.

To store users' passwords securely, we used the SHA-256 algorithm to ensure that the database administrator cannot view passwords as plain text. This algorithm performs one-way hashing, so when a user signs in, we hash their password and compare it to the hash stored during the registration process.

Connection to Google Calendar

Written by Cristina Semikina

The integration of Google Calendar into the habit tracker app allows users to sync their daily habits with their calendar. This feature provides convenience by automatically creating reminders for habits, ensuring users stay on track with their goals.

For this integration it was necessary to use different tools and libraries:

- **Google Calendar API:** To interact with Google Calendar.
- **Google Client Library:**
  - `google-api-client`
  - `google-oauth-client`
  - `google-oauth-client-jetty`
  - `google-api-services-calendar`

Before starting implementing the code, it was necessary to set up OAuth 2.0 Authentication. A project was created in the **Google Cloud Console.** The Google Calendar API was enabled and the credentials were saved in a .json file placed inside the ‘resources’ directory of the app.
This file is used to request authorization from the user’s Google account.

The app stores tokens locally in the `tokens` directory, allowing reuse without repeated authorization.

For the implementation of this feature 2 classes were essential: *GoogleCalendarAuth* and *HabitDetailsController*.

GoogleCalendarAuth handles authentication and provides a `Credential` object for authorized API access.

The sync functionality is implemented in the *HabitDetailsController* class and it adds a habit as events to Google Calendar for 7 consecutive days.

Events were successfully added to Google Calendar for the next 7 days.
Users could see the events in their calendar with the correct title, description, time, and duration.
Handled cases like missing credentials and insufficient permissions by displaying meaningful error messages to the user.

There were some complications for this implementation since initially, the app used the `CalendarScopes.CALENDAR_READONLY` scope, which did not permit creating events. The scope was updated to `CalendarScopes.CALENDAR`, and tokens were regenerated. Besides that, existing tokens were invalidated due to scope changes. The `tokens` directory was cleared, and new tokens were generated.

Email system

Written by Jan Walczak (implemented by Aleksandra Bruska and Jan Walczak)

Adding an email system for our application was a must, since we wanted to send welcoming messages to users, who are creating accounts in our application. Email system would also allow us to implement sending verification and advertising messages in the future.

Initially, we used Mailtrap, to get to know how to handle sending emails in Java.
It allowed us to capture and view emails sent during the development and testing stages of an application without actually sending them to real recipients. Mailtrap, like real email providers, provides SMTP (Simple Mail Transfer Protocol). To send emails, we needed to specify which port we would be using. The best choice for this purpose was port “587” which provides STARTTLS and TLS (Transport Layer Security). Both of those protocols ensure safe connection between two computers where emails are being sent.
(*Get Started - Mailtrap Email Testing*)

As Mailtrap is an artificial environment, we needed an actual provider, which would work in real time, and send emails to receivers. We have searched through a lot of popular domains like “gmail.com” and “outlook.com” but they required either phone number or national ID to create an account, which we were not willing to give. Finally, we have found the “wp.pl” domain which requires only a “trusted email” to be passed, for creating an account.

After creating an account, we had to specify which email protocols we would be using. There were following:

1. POP3 (Post Office Protocol version 3) – A protocol used by email clients to retrieve emails from a mail server
2. IMAP (Internet Message Access Protocol) – A protocol used for server-side management of emails

(*skc161931/GeeksForGeeks*)

For authentication, in Java code, we had to create a session for authentication. To do so, initially we stored our login and password to the email domain as a hard-coded text in the authentication method. We have all agreed that this is definitely not a smart idea, since we wanted to make our Github repository public in the future, and someone would eventually log in to our account.
We have created a “secrets.properties” file, which was added to “.gitignore”, so we could store it safely on our computers, without worries that our login and password would be leaked.

Tests

Written by Cristina Semikina

This paragraph details the testing process for the JavaFX application, specifically how the implemented user stories were tested and which tests correspond to each user story. Additionally, it highlights the improvements made by integrating the *FXMLLoader* with the application's FXML views and addresses persistent issues with certain tests.

We tested almost every US implementation successfully, achieving significant improvement in the performance of the application by handling some errors that were occurring during the development of the project. As the development of the project was occurring, we let the tests run as the last part, knowing that is a mistake. For future work, we would start testing at the same time as the development of the US for better debugging and not overloading the application with possible bugs.

Testing Overview

The following tests were conducted for the controller classes in the application:

1. **CreateAccountControllerTest:** This test verifies the functionality for user account creation and login, including input validation. It directly addresses **US #1**, testing the form validation (e.g., email format, password matching) and database interactions for storing user details. This user story was fully implemented and tested, with successful results after integrating the *FXMLLoader*. Also **US #8** was tested since the creation of the account runs successfully .

2. **LoginControllerTest**: This test focuses on the login functionality, ensuring that the provided credentials are checked against the database and proper feedback is given for success or failure. It is also tied to **US #1** and confirms that users can log in using the implemented form.

3. **ProfileViewControllerTest:** This test covers the functionality for viewing and editing personal details. It validates the ability to fetch user data from the database and update it successfully. This test corresponds to **US #4**, which was implemented and verified. Optional features were not tested, as they were not implemented.

4. **DashboardControllerTest:** This test focuses on habit management, including adding, tracking, and deleting habits. It primarily corresponds to **US #3** (adding habits), **US #6** (tracking habits using checkboxes), and **US #7** (streak tracking). Although the *FXMLLoader* resolved many issues, this test still exhibits unexpected behavior. The main problems stem from the programmatically generated circles for streak tracking and occasional database connection failures, which cause intermittent test failures.

For the GoogleCalendarConnection we found it really hard to perform tests since it was necessary to perform them according to the Google API documentation. This caused many troubles because the version used in our project was not the same as the documentation due to Gradle issues and therefore we didn’t have enough time to test it out.

Unimplemented and Untested User Stories

The following user stories were not implemented and, therefore, were not tested:

- **US #11**: A timer for measuring time spent on habits was not implemented.
- **US #12**: Weekly feedback, performance analysis, and badge awarding were not implemented.
- **US #13**: Social features for connecting with friends and sharing habits were not implemented.

Improvements from FXML Integration

A significant improvement was observed after switching to the *FXMLLoader* to load and bind FXML views to their respective tests. Before this change, many tests failed due to discrepancies between the test environment and the actual application. With the integration of *FXMLLoader*, all tests started functioning as expected, with the exception of the *DashboardControllerTest*. The latter still shows intermittent failures due to issues with programmatically generated UI elements and database synchronization.

The database is a critical component, and occasional failures during tests highlight the need for robust error handling and possibly mock database environments for testing. At the moment we are only testing using the real database with a mock user “test”.

Using *FXMLLoader* significantly improved the accuracy and reliability of the tests by aligning them more closely with the application's runtime behavior. But, despite improvements, the test still encounters issues related to programmatically generated UI components and inconsistent database connections. In the future, this implies the need for further debugging and possible refactoring of the dashboard logic.

Also, several user stories remain unimplemented such as advanced analytics (**US #12**). These features, once implemented, will require new test cases to ensure their functionality.

The Results

Written by Cristina Semikina

At this stage of the project, significant progress has been made in implementing and testing various features of the application. Most of the implemented user stories have been successfully tested, with notable improvements after resolving configuration issues and integrating better testing practices. However, there remain areas requiring additional attention and development.

The *FXMLLoader* integration with the tests provided accurate synchronization between the JavaFX FXML views and the controller logic, allowing tests like *CreateAccountControllerTest*, *LoginControllerTest*, and *ProfileViewControllerTest* to pass reliably. These tests validated user account creation, login functionality, and the ability to edit personal details, ensuring these core user stories (e.g., US #1 and US #4) were successfully implemented and tested.

Features related to habit management (US #3, US #6, US #7) have been partially validated through *DashboardControllerTest*. Users can add habits, track them with streak counters, and utilize the "Streak On Ice" feature (in an early stage version). While these functionalities worked under specific conditions, the dynamic generation of circles and intermittent database connection issues caused some test failures. Further debugging and optimization are needed for this feature.

A critical blocker during testing was JavaFX configuration, which required the addition of **`--add-exports javafx.graphics/com.sun.javafx.application=ALL-UNNAMED`** to the JVM arguments. This resolved module access issues and enabled seamless testing with JavaFX components.

Despite the implementation for Google Calendar synchronization (US #5), testing was unsuccessful due to connection issues and technical challenges in integrating the Google Calendar API with the Junit testing and dummy data. Despite having this feature implemented, the tests still remain incomplete which requires future work and special attention for bugs and unexpected behaviour.

Alos, several user stories, including advanced analytics (US #12), social features (US #13), were not implemented and therefore not tested. Their development will require additional design, coding, and rigorous testing in the future.

The use of *FXMLLoader* was a pivotal change, aligning the test environment more closely with the runtime behavior of the application. This reduced discrepancies between mock and real components, ensuring higher reliability of test results. Additionally, the switch to using real FXML views exposed dependencies and areas in the code that needed better modularization, particularly in the dashboard.

Overview and Next Steps

The project is in a strong position, with foundational features like account management, habit tracking, and basic profile editing tested and validated. However, the intermittent issues with the dashboard and the absence of key features like stats analytics highlight the need for further development.

Our next steps should focus on:

- Debugging and stabilizing the dashboard features, particularly dynamic UI generation and database synchronization.
- Revisiting Google Calendar integration and resolving API-related challenges.
- Implementing and testing uncompleted user stories like analytics and social features.
- Enhancing error handling and improving the modular design of the application for more robust testing.

With these improvements, the application can achieve its intended scope and provide a polished user experience.

Discussion of results

Written by Aleksandra Bruska

The most obvious result of our work is a desktop application. It is simple, but provides a solid foundation for further development and could be enhanced by more advanced functionalities.
We managed to cover all the user stories marked as high or medium priority. The application allows for creating a profile as well as for changing the information in the profile screen. The most interesting part is that the user is able to change the profile picture to one of the pictures stored on their computer at any location. Logged-in users can then use the habit tracking functionalities, including adding and performing habits as well as counting the streak.

We are particularly proud that apart from basic functionality, we managed to add some extra features that enrich user experience. One of them is the possible connection with Google Calendar. This allows for a better visualization of planned activities and facilitates the process of accessing the data. This is a significant improvement, since Habit Tracker can only be accessed on a computer, while Google Calendar allows for viewing the habit related data on a mobile phone. Another noteworthy feature is the connection with the email system that allows the application for sending emails to users. In the current version the application only sends a welcome email. However, this has a great potential of development and could be used for example to send code in order to verify email address or recover the password.

Furthermore, we managed to create a database and connect it with our application, which was one of the priorities for this project. It was a challenge and we are satisfied that we managed to do it soon enough so that there was still time for coding the features that use the database as well as adapting the database to changing goals. Our database was relational, which differs from what we learned in the Mobile Application Development course, where we used a non-relational database. Even though it was different, the knowledge that we have gained through this course was useful.

Moreover, the work produced results that are difficult to quantify, which includes increasing the technical skills of group members. For most of us, this was the first project created using the JavaFX platform. For this reason, we had to learn how to use it at a level that is sufficient for creating a working application. It was also an opportunity to increase our qualifications concerning managing databases and using them in a Java application.

For some of the members, this project was the first opportunity to work according to the Scrum methodology. They were able to experience not only the benefits, but also the challenges of it. They also learned about tools useful for managing an iterative process, including Notion board.

This project gave us a chance to apply the knowledge obtained during this semester’s courses in practice. When planning the design and architecture, including creation of C4 model diagrams, we mostly relied on the information gained during the Software Design course.

Moreover, when we created an account for email system purposes, we had to store login information safely. To do so, we have created a “secret.properties” file, which was added to “.gitignore” so that this data would not become leaked to the web. We learned about this approach on the Mobile Application Development course.

Conclusion

Written by Jan Walczak

Our intention was to provide a prototype of a desktop application, to help people perform their daily routine. We can call this part a success, as our prototype is a way for a user to effectively manage their time in an easy way. We managed to provide all crucial functionalities for this type of application as well as some advancement, including Google Calendar connection. Most important features for this purpose were:

1. Creating user profile
2. Creating custom habits
3. Tracking the completion of daily routine

To ensure that the application is easy to use, we kept the design simple, including only necessary UI components. Unfortunately, we did not manage to provide users with feedback other than streak counting.

We consider our project as a “prototype” more than “product” because some functionalities still work locally. We were considering what we should do to deploy our application as a final product. We have all agreed that our application would need to be installed, rather than be manually opened via the user's terminal. Database management system should be located on a server, that logging in functionality would make sense and a user would not be obliged to store theirs data locally.

During the process, we had to make changes and refactor some packages. We completely changed the design for managing database in order to ease other participants' work and to make code more clear for them. We also had to adjust other packages, for example screens of our application, in order to join them all together in a whole thing. The aim of this was to ensure that each team member feels confident working with the code we created. This way of cooperating was especially useful, as it is a main goal for every project work.

Process of creating this project may be described as rough. As international students, who would be in Aarhus for only one semester, we had to quickly get to know each other in order not to waste valuable time that should have been dedicated to working on our group project.
This was achieved by creating a “Collaboration agreement” and dividing work at the beginning, so we would have known our capabilities and strengths. This method was especially valuable for us, as it is very similar to real-life scenarios, when you start a project with coworkers, who you do not know. We scheduled our meetings to occur at regular intervals, so we would know what others were currently working on and what our current progress was. However, this method failed at some point, so we planned a special meeting, dedicated to resolving conflicts that had occurred, and we succeeded.

One of the major challenges was to stick to deadlines set for individual tasks. In some cases we did not manage to finish the work that was planned for a particular sprint by its end. One of the reasons is that as exchange students, we found it hard to predict the workload distribution during the semester and the amount of time that we would be able to spend on this project. What is more, we underestimated the size of some of the tasks. The solution was to add an additional fifth sprint to ensure that we would be able to finish the scheduled work.

Finally, during this process we have learned how to cooperate with others effectively, how to resolve conflicts and how to distribute workload accordingly. We have also succeeded in refactoring project at places that someone has failed.

Future work

What remains to be done?

Written by Jan Walczak

For this moment, our application remains as a locally working product. It is not ready for a deployment for a wider audience, as a user would have to create a database themselves on their computer. The thing that must be done before deployment is to rent some hosting for data and connect it to our application.

The other thing is to give users feedback about their performance. For now, the user is able to see when they performed their habits, but it might be useful for them to see some statistics like total time spent, longest session etc.

It would be convenient for a user to use “Streak on ice” functionality. We have managed to implement it, and it works perfectly with the connection with Google Calendar. However, users might not connect their accounts to Calendar, so we have to improve this functionality. It would increase the probability that a user would not quit using our application after losing their streak.

The key aspect of keeping users using any application is to create some relatable society around the product. By creating a functionality by which users could add their friends and see their performance, we would ensure that users would spend more time inside the application.

What opportunities are available in the project?

Written by Jan Walczak

Our project could be possibly transferred to other platforms. For example, instead of developing it as a desktop application, we could try developing it as a mobile application. Access to it might be easier and more convenient for a user by doing so. Another idea might be to create a webpage with the same functionality. By doing so, our product would be available for both desktop and mobile devices with the same workload needed to improve the application in the future.

Reference list

Salo, Lilli (2024) *Comparing Java GUI frameworks: Vaadin, JavaFX, and Swing*
Available at: https://vaadin.com/blog/comparing-java-gui-frameworks-vaadin-javafx-and-swing
(Accessed 11.10.2024)

ayaan07/GeeksForGeeks (2024), *Java AWT vs Java Swing vs Java FX*
Available at: https://www.geeksforgeeks.org/java-awt-vs-java-swing-vs-java-fx/
(Accessed 11.10.2024)

Fadatare, Ramesh (n.d), *Difference between JDBC and JPA*
Available at: https://www.javaguides.net/2023/11/difference-between-jdbc-and-jpa.html
(Accessed 11.10.2024)

Get Started - Mailtrap Email Testing
Available at: https://help.mailtrap.io/article/109-getting-started-with-mailtrap-email-testing

skc161931/GeeksForGeeks (2024) Differences between POP3 and IMAP
Available at: https://www.geeksforgeeks.org/differences-between-pop3-and-imap/

Hibernate and Jakarta documentation
Available at: https://hibernate.org/orm/documentation/6.6/