BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

SMART TASK PLANNER

INTELIGENTNÍ PLÁNOVAČ ÚKOLŮ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **JAN ZIMOLA**
AUTOR PRÁCE

SUPERVISOR **Ing. JIŘÍ HYNEK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

148580

Institut: Department of Information Systems (UIFS)
Student: **Zimola Jan**
Programme: Information Technology
Specialization: Information Technology

Title: **Smart Task Planner**
Category: Mobile applications
Academic year: 2022/23

Assignment:

1. Get acquainted with the time management theory, study the principles and approaches to time scheduling.
2. Get acquainted with the principles of recommendation systems.
3. Study principles of multiplatform mobile application development.
4. Analyze the requirements for smart time scheduling. Correspond the requirements with existing tools.
5. Based on the requirements, design a mobile application for smart time scheduling.
6. Implement the application.
7. Test the usability of mobile application and time scheduling with selected users.

Literature:

- Clear, J. (2018). *Atomic Habits: An Easy & Proven Way to Build Good Habits & Break Bad Ones*. Random House.
- Allen, D., & Fallows, J. (2015). *Getting Things Done: The Art of Stress-Free Productivity*. Penguin Publishing Group.
- Da'u, A., & Salim, N. (2020). Recommendation system based on deep learning methods: a systematic review and new directions. *Artificial Intelligence Review*, *53*(4), 2709-2748.
- Johnson, J. (2010). *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines*. Morgan Kaufmann Publishers/Elsevier.
- Flutter (2022): Flutter documentation. Online: https://docs.flutter.dev/

Requirements for the semestral defence:
Items 1 - 5.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Hynek Jiří, Ing., Ph.D.**
Head of Department: Kolář Dušan, doc. Dr. Ing.
Beginning of work: 1.11.2022
Submission deadline: 10.5.2023
Approval date: 25.10.2022

Abstract

This thesis aims to create a new planning application that combines various time management techniques. From in-depth research on time management combined with a survey of potential users, diverse techniques emerged that the application needs to provide to satisfy various users' needs. Current applications were insufficient for this purpose. The new implementation mainly combines a todo list and a calendar but also adds support for habits or the Pomodoro technique. On top of it, the user can assign values to projects, labels, or tasks such as priority, deadline, or estimate. These attributes are later used in a machine learning model to score elements, enabling the app to suggest the most relevant tasks to the user. The app was built using the multiplatform framework Flutter and released to App Store and Google Play. Own synchronization mechanism for the app's data is utilized between an Isar database and an Appwrite server. The user-defined events can also be reflected in the user-selected synchronization local calendar.

Abstrakt

Tato práce si klade za cíl vytvořit novou plánovací aplikaci, která kombinuje různé techniky správy času. Z hloubkového výzkumu správy času a průzkumu mezi potenciálními uživateli vyplynulo mnoho různých technik, které aplikace musí poskytnout, aby uspokojila různé potřeby uživatelů. Současné aplikace k tomuto účelu nestačí. Nová implementace kombinuje především seznam úkolů a kalendář, ale také podporuje návyky nebo Pomodoro techniku. Uživatel může přiřadit hodnoty projektům, značkám nebo úkolům, jako je priorita, termín nebo odhad. Tyto atributy jsou později použity v modelu strojového učení pro ohodnocení prvků, což umožňuje aplikaci navrhnout uživateli nejrelevantnější úkoly. Aplikace byla postavena pomocí multiplatformního rámce Flutter a uvolněna na App Store a Google Play. Vlastní synchronizační mechanismus pro data aplikace je využíván mezi databází Isar a serverem Appwrite. Uživatelem definované události mohou být také zobrazeny v lokálním kalendáři vybraném uživatelem pro synchronizaci.

Keywords

time management, mobile app, Flutter, Appwrite, Isar, Pomodoro technique, timeboxing, calendar, todo list, dart, habits, Android, iOS, machine learning

Klíčová slova

správa času, mobilní aplikace, Flutter, Appwrite, Isar, Pomodoro technika, timeboxing, kalendář, seznam úkolů, dart, zvyky, Android, iOS, strojové učení

Reference

ZIMOLA, Jan. *Smart Task Planner*. Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Jiří Hynek, Ph.D.

Smart Task Planner

Declaration

I declare that I have independently prepared this bachelor's thesis under the guidance of Mr. Ing. Jiří Hynek, Ph.D. I have cited all literary sources, publications, and other resources from which I drew information.

.....
Jan Zimola
May 9, 2023

Acknowledgements

I want to thank my family for enabling me to study at the university, thus providing me the space to find what I want to do and become good at it. I also want to thank my supervisor Mr. Ing. Jiří Hynek, Ph.D., who showed incredible patience with me, was speedy to help, provided me with essential feedback about how to write this thesis, and generated ideas that are reflected in the final application. And lastly, I want to thank all who took the time to test the application and provided me with feedback that I used or will use to improve the app.

Contents

Chapter 1

Introduction

We are currently living in an era where digital advancements are creating new possibilities while simultaneously increasing the complexity of the world we live in. This digital age demands more from each of us, imposing greater requirements and expectations. Planning is a key strategy for coping with these demands, and mobile devices have become a popular tool for organizing responsibilities due to their accessibility. By inputting tasks and deadlines into mobile devices, we can effectively manage our daily tasks and remain in control. Although various tools such as calendars and todo lists have a long history of helping people stay organized, many individuals still do not plan regularly, limiting their use to special cases. Others require multiple applications, including todo lists and calendars, to maintain their organization.

This project aims to develop a mobile application that integrates various popular time management techniques such as calendar, todo list, Pomodoro technique, and habit tracker into one platform. The application will be compatible with the most commonly used mobile operating systems. The concept of this application is based on several sources of information including personal experience with time management, prior research on time management techniques, and a survey specifically conducted to gather information about the usage of planning applications.

Chapter 2 provides an explanation of fundamental time management terms and techniques that are directly connected with planning applications and their functionality. In Chapter 3, the search for ways to construct mobile applications and what concrete frameworks can be used is discussed. Chapter 4 delves into the principles of recommendations used to understand how the application can recommend relevant tasks to the user.

In Chapter 5, a conducted survey will be explored to better understand potential users' needs. By combining survey findings with insights from time management research, concrete requirements that the app should provide were defined. These requirements were later compared with current applications, and the resulting requirements form the design outline in Chapter 6, with their implementation explained in Chapter 7. Finally, the testing of the final implementation is described in Chapter 8.

Chapter 2

Time Management Theory

Recently, there has been an increasing demand for time management literature. It became important due to the growing need for fast availability of products [17]. Multiple studies, mostly from schools, have documented its benefits.

- Procrastinating students had a lower perceived control of time, which made them agitated. On the contrary, scholars that used time management principles were less nervous, as they felt more in control [4].
- Time management techniques give people more control, which makes them happier and healthier while reducing stress [17].
- Practicing time management techniques by college students may influence college achievement [11].
- Students, who are planning, report greater work and life satisfaction and less job-induced and somatic tensions [54].
- Time management with leisure activities may be effective at reducing academic stress [61].

Time management is a vital aspect of personal development, comprising various terms and techniques. All these strategies look at the problem from different angles, solving different problems and satisfying various needs.

2.1 Definition Of Time Management

There is no standard definition of time management. However, one of the most approved ones is that time management involves determining needs, setting goals to achieve these needs, and prioritizing and planning tasks required to achieve these goals [49].

Others define time management as planning and exercising conscious control of time spent on specific activities to increase effectiveness, efficiency, and productivity [22].

2.1.1 Efficiency

Efficiency is about doing everything as quickly as possible with the fewest wasted motions [49].

2.1.2 Effectiveness

„Effectiveness means selecting the best task to do from all the possibilities available and then doing it the best way. Making the right choices about how you will use your time is more important than doing efficiently whatever job happens to be around [49].“

2.1.3 Prioritization

Software development requires prioritization since it is impossible to consider all relevant factors [51]. This is also relevant for self-management, where it is possible to define three extremes of how people handle prioritization [49].

- The `overorganized` person makes lists and plans to be confident that he has the perfect plan. As a result, he spends so much time planning that he does only a little.
- The `overdoer` is so focused on doing that he ignores his priorities and the bigger picture. Consequently, he is inflexible and problematic to work with.
- A `time nut` is a person obsessed with time and causes himself and others to be nervous by focusing on never wasting a minute. He makes a schedule for each day that is very detailed and impossible to follow.

2.1.4 Doing things faster

All stereotypes focus on doing things faster, forgetting that pausing and reflecting are critical problem-solving elements [65]. This skill distinguishes undergraduates from math experts, who take longer to decide which approach to choose [15].

2.1.5 Goals

There are two major motivationally relevant goal patterns in goal setting: `ego-involved goals` and `task-involved goals` [27].

- People with ego-involved goals seek to look competent and skilled in the eyes of others. Their minds are filled with questions like „Will I look smart?“ and „Can I outperform others?“
- On the other hand, individuals with task-involved goals focus on mastering tasks and increasing their competence. As a result, their mind is full of questions like „How can I do this task?“ and „What will I learn?“

2.2 Time Management Principles

This section discusses important principles that are reflected in many time management techniques 2.3 that used them as their foundation.

2.2.1 80 / 20 Rule

The Pareto or 80 / 20 principle states that roughly 80% of consequences come from 20% of causes [14]. The phenomenon is seen in many fields, such as money distribution. Research

suggests that 20% of the wealthiest people own 56.72% of the total money, while 20% of the richest countries have 91.62% of the total money [25].

According to physicist Victor Yakovenko of the University of Maryland, the income distribution of the upper class also follows this principle [82].

2.2.2 Parkinson's Law

Parkinson's law states that the work expands to fill the available time for its completion [69]. In two experiments with 24 paid undergraduates, it was shown that students naturally slowed down when given more time [12].

2.2.3 Deep Work

Carl Newport defined the term in his book Deep Work [63] as „professional activities performed in a state of distraction-free concentration that push your cognitive capabilities to their limit. These efforts create new value, improve your skill, and are hard to replicate."

Deep work is rare in today's world, filled with distractions such as social media, Netflix, and YouTube. It is also disappearing from the workspace, where the average worker spends 60% of his time in electronic communication [16].

Constantly switching between tasks decreases our performance because the brain needs to focus intently on a task to be effective [52]. This concentration enables the brain to load information into short-term memory and establish high-intensity communication with it. Communication of this strength helps build myelin sheaths over nerves, which allow the brain to move neurons along them faster [65].

Deep work enables their employees to perform exceptionally [20]. Nevertheless, today's trend of open-spaced workspaces (Facebook, Twitter) significantly increases distractions [80]. A study [8] found that 67% of workers are disrupted by phones ringing, 55% by people talking, and half by office air conditioning or machines. This is important because even a small interruption can significantly increase the time needed to complete a task [58].

2.2.4 Diffuse Mod vs. Focus Mode

Since the start of this century, people have gained a deeper understanding of how the brain works. One of these things is the transition between focused and diffuse modes of thinking — two different strategies for solving problems and finding patterns. Barbara Oakley explains these terms in her book A Mind for Numbers [65].

- During focus mode, the brain solves problems using direct and rational methods. It is associated with the prefrontal cortex's ability to concentrate.
- Diffuse mode involves relaxing attention and allowing the mind to wander freely. People enter this state when doing relaxing activities, such as taking a shower, walking, or sleeping [39]. This ease activates a wide range of brain areas, which allows the production of insights and different perspectives. Children are better at using the diffuse mode of thinking, enabling them to solve a certain puzzle that adults struggle with.

People should focus on a problem first and then allow the mind to relax and find connections. This approach can help us overcome the `Einstellung effect`, which states that people are likely to stick with the initial approach instead of searching for a more

optimal solution. Experts whose experience decreases their ability to act creatively are often affected [10].

2.3 Time Management Techniques

The last section discussed many time management terms 2.2. However, because these principles are difficult to embrace, many techniques have emerged that wrap them in a simple interface.

2.3.1 Pomodoro Technique

The pomodoro technique was developed by Francesco Cirillo in 1980. It involves working for some time and then taking a brief break. After several rounds, it is recommended to take a more extended break. The standard describes working time as 25 minutes, a short break as 5 minutes, and a long break as 15 minutes [64]. The pomodoro technique helps to overcome various problems, often by providing effective relaxation.

Better Concentration

Researchers found that briefly deactivating and activating tasks from focus enables the brain to concentrate for longer [7]. This shift can be achieved by taking a small break.

Improves Problem Solving

It helps to use focused and diffused states of thinking. By entering diffuse mode in a break, people can find a better way to solve a problem [65].

Decrease Effects of Sitting

An office worker spends 15 hours per day sitting [26], which significantly increases his risk of diabetes and mortality [81]. It is recommended to use light-intensity activity to reduce this effect. When taking a break, people are naturally motivated to do light-intensity activities, such as stretching and walking. During a long break, the person can even perform more time-demanding activities, such as walking outside and exercising.

Decreases Eye Strain

Up to 90% of digital users have symptoms of eye strain — especially computer workers or contact lens wearers [19]. Study [75] suggests a lower number of 50%.

One way to combat this problem is utilizing the 20-20-20 rule, which can decrease the effect of digital device usage on eyes [5]. While taking a break, a person is motivated to look away from the device screen, thus naturally utilizing the 20-20-20 rule.

Helpful for Addressing Multitasking

Another issue that hinders productivity is multitasking [56]. A study shows how easily students can be distracted at home if they have technology present. Fifteen-minute observation revealed that students had difficulty focusing on their primary task, averaging less than 6 minutes staying on a task before switching to another task. Put another way. Students stayed on a task on average 10 minutes out of the 15-minute study period. Aside

from getting up and walking around, switching to another task was most often associated with technological distractions, such as texting, Facebook, and watching TV [73].

The study indicates that students found the pomodoro technique helpful for addressing multitasking, even though there was a disagreement on what specifically about it helps. In conclusion, students needed to allow enough time to get used to the technique and adjust it to suit their context [79].

2.3.2 Habits

James Clear established healthy habits to promote his identity change and became influential [18]. He states: „Your behaviors are usually a reflection of your identity. What you do is an indication of the type of person you believe you are — consciously or nonconsciously“. So instead of embracing behavior changes (I will stop smoking), people should look at habits from an identity perspective (I am not a smoker). Nevertheless, this is also true for unhealthy habits. For example, when someone identifies as a smoker, it becomes harder for them to stop smoking.

James Clear proposes that people forget about goals and focus on small habits that compound and have a significant impact.

2.3.3 Eat the Frog

Brian Tracy popularized this idea in his book Eat the Frog. „It has been said that if the first thing you do each morning is to eat a live frog, you can go through the day with the satisfaction of knowing that that is probably the worst thing that is going to happen to you all day long,“ says Brian Tracy, and continues „Your “frog„ is your biggest, most important task, the one you are most likely to procrastinate on if you don’t do something about it. It is also the one task that can have the greatest impact on life and results at the moment“. And adds that „frog eating“ should become a lifelong habit to save energy and make it automatic [78].

Later summarizes: „Successful, effective people are those who launch directly into their major tasks and then discipline themselves to work steadily and single-mindedly until those tasks are completed.“

2.3.4 Timeboxing

Timeboxing is a technique of defining a specific timeframe to achieve particular results. Each timeframe consists of tasks that need priority to clarify which ones could be eliminated if there is not enough time to complete them all [60]. Timeboxing helps organize projects or improve school performance [68].

In development, timeboxing is used as an alternative to the waterfall development strategy. It defines a timebox and associated tasks. Multiple timeboxes should be running simultaneously, each for a specific team. This approach significantly reduces the cycle time between each delivery [44].

The following presents ideas from three blogs that convey their experience with timeboxing.

- When using timeboxing, things are put right in the calendar, which makes it easier to do things at the right time [83]. This ensures that the plan is grounded and realistic. Furthermore, timeboxing helps to combat the need to choose between tasks, a factor

that people struggle with [74]. A study [43] suggests that people are more likely to write better essays or buy jams when offered fewer options.

- Timeboxing makes it easier to do major tasks first by fighting against the `planning fallacy phenomenon` [46]. It states that people are likely to underestimate the time it will take to complete a task. Even though they know that the previous task took longer than planned [13].
- Timeboxing helps combat perfection [47].

2.3.5 Todo List

Todo list helps decompose goals into actionable and manageable tasks, reducing the anxiety of unfulfilled ones [59]. In the productivity system GTD, David Allen uses a todo list as the only inventory for all commitments [4]. He explains, „Your mind is for having ideas, not holding them.“ By keeping track of all commitments, the inventory can enable intuitive prioritization.

David Allen suggests a four-step framework for making todo list clean, complete, and organized. Firstly, people should capture everything, including ideas, tasks, and groceries, and put it into the todo list. The next step is to clarify the items by turning them into projects or actionable tasks with all details. And finally, they should review everything daily, weekly, monthly, and yearly.

Chapter 3

Application Development

People spend 4.6 hours daily on mobile devices, which makes it an essential platform for business [23]. Developers can make mobile apps by using native or cross-platform development. Another option is to create a Progressive Web Application (PWA).

3.1 Development Strategies

Before choosing an approach, it is essential to consider project requirements, such as money, people, time, and possible expansions. If a mobile app were successful, many users would want web and desktop support. And that would be significantly easier to implement with reused code from mobile. For example, Tencent states that 90% of Flutter code can be reused across platforms[1].

Additionally, third-party libraries and the main framework capabilities are important factors. Developers who use frameworks such as Flutter or React Native have ready-made design components. The developers can then adjust them to fit the application use case.

Other libraries help these frameworks use native functionality such as file picker, notifications, and database. These libraries implement functionality for all platforms and then make it callable from the multiplatform framework[2][3][4].

3.1.1 Native Development

Native development is an approach where developers create a native app for each platform. It provides higher performance, especially compared to PWAs [2]. Aside from these benefits, it provides developers with direct access to the native API, eliminating the need to use plugins for it [38]. Moreover, when programmers build an app from platform components, it results in a native look. And as the native components change, it is enough to rebuild the app to get the latest design. This is a significant advantage over Flutter, where the framework supports these components later as it is only emulating them [36, 33].

However, if the app was running on multiple platforms, the development would require separate teams for each platform, making it harder to organize. Furthermore, the codebase grows more quickly because each platform has its unique code. Finally, testing needs to be done for each platform separately [28].

[1] https://flutter.dev/multi-platform
[2] https://pub.dev/packages/awesome_notifications
[3] https://pub.dev/packages/isar
[4] https://pub.dev/packages/file_picker

Android

Applications for Android can be created using Kotlin, Java, or C++ languages. It is recommended to use Android Studio as a development environment with either Kotlin or Java as programming languages. The official distribution platform is Google Play [32].

iOS

iOS applications can be created using either Swift or Objective-C. Swift is recommended because it is designed to be more secure, simpler, and faster. On the contrary, Objective-C is one of the most dreaded languages [67]. The recommended IDE is Xcode. The official distribution platform is App Store [42].

3.1.2 Progressive Web App (PWA)

When supporting multiple platforms is required, a common approach is creating a Progressive Web App (PWA). It is a web application that is responsive enough to support mobile devices and runs in a specific environment. The strategy is cost-effective for developing multi-platform applications because it can run on any device in a browser [2]. Representations are React, Vue, Angular, and Ionic.

The drawbacks are that PWAs can be significantly slower than native implementations. Moreover, developers using PWA rely on browser developers to support native features such as notifications. For example, browsers still need to support iOS push notifications [53].

3.1.3 Cross-Platform Application

It aims to combine the possibilities of sharing code across multiple platforms while providing a way to access the native framework's API. The disadvantages of this approach are that the app may be perceived as less native, have lower performance, and have more problematic access to the native API. React Native and Flutter are the most popular options [57, 28].

React Native

React Native is supported by Facebook. It uses JavaScript/TypeScript, one of the most popular languages [67], and thus it can benefit from its ecosystem. React Native uses the JavaScript bridge to access native components, which is slowing the app to an extent [30, 70]. The components it provides are only the basic ones. This results in developers having to make more customizations than with Flutter.

Flutter

Flutter is supported by Google. It compiles an app into native code, making it faster than React Native [24]. And because of its own rendering engine, Flutter controls every pixel on the screen [30].

3.2 Mobile Application Architecture

Mobile app architecture comprises three main elements: view, state management, and database. In the implementation, each element would represent various tools and languages.

Figure 3.1: Mobile application architecture diagram

Understanding each element helps make better decisions about which concentrate tools to choose.

3.2.1 View

The view is responsible for rendering itself based on state management's data [6]. The user interface (UI) is a bridge between a system and a user. A good UI enables the user to access information that leads to achieving the user's goals, while a bad UI disorients a user. Confusion can cause a complicated menu, inconsistent navigation flow, or insufficient descriptive UI [48].

Bad user experience (UX) costs businesses money because of the need for extensive training and support [41]. To provide users with good UI and UX many design libraries emerged. Its components can be easily tweaked to suit specific application needs. These UI kits follow good UI practices that users are already familiar with. Material Design [35] from Google is a popular option implemented for frameworks such as Flutter or React [62, 34].

3.2.2 State Management

State management is a bridge between a view 3.2.1 and a database 3.2.3. It provides data to a view and reacts to the user's actions received from the view. These actions can modify the data of state management or database. State Management is responsible for notifying a view about these changes.

3.2.3 Database

The term 'database' refers to a shared collection of logically related data and its description, designed to meet an organization's information needs. Smaller databases can be stored on a file system, while larger databases can be hosted on computer clusters or in the cloud. The design of databases spans formal techniques and practical considerations, including data modeling, efficient data representation and storage, query languages, security, and privacy of sensitive data, including the support for concurrent access and fault tolerance. The database is accessed through a database management system (DBMS) that is used to interact with the database. There are two major types of families No-SQL and SQL [21].

SQL databases use a relational model to store data. They organize data into one or more tables of columns and rows, with a unique key identifying each row. Generally, each table represents one „entity type.“ The rows represent instances of that entity type, and the columns represent values attributed to that instance [66]. To summarize, SQL databases have strict schema and enable combining information using a query language.

No-SQL databases expect the developer to denormalize the data because of their schemeless nature. The developer has flexibility regarding how he stores the data, allowing him to store all relevant data in one record. Because the data is much more likely to be reread, changing it at multiple places is acceptable. The relationships are represented in multiple

ways, and it is up to the developer to decide. The trade-off is that the developer can never be certain what data he gets, so the developer needs to code defensively. Many local No-SQL databases have recently been using code-generation to generate the database schema to provide a safe static API[5][6][7]. The No-SQL database can scale horizontally [45].

[5]https://pub.dev/packages/isar
[6]https://pub.dev/packages/realm
[7]https://pub.dev/packages/objectbox

Chapter 4

Recommendation Systems

Recommendation systems are a subclass of information filtering that suggests the most relevant items to a particular user. Often these suggestions are the best guesses of what the user would most likely do, buy or watch next. Tips help when the user needs to choose from a vast range of options [74].

Areas that benefit from recommendation systems are media content providers, online stores, social platforms, and dating apps [40, 72]. For instance, the YouTube recommendation system for videos displays a grid of cleverly chosen options to the user, eliminating the need to search or browse by category. As a result, YouTube's suggestions account for 60% of watch time. Other areas that have seen similar trends include: [55]:

- Forty percent of apps installed on Google Play.
- Thirty-five purchases on Amazon.
- Seventy-five percent of movies watched on Netflix.

Recommendation systems usually have two flavors: *collaborative filtering* and *content-based filtering*. Collaborative filtering uses users' past behavior (such as video ratings, likes, or comments) and other users' decisions to suggest. Content-based filtering uses only current user actions to recommend [3].

4.1 Collaborative filtering

Collaborative filtering assumes that people will like similar things as they wanted in the past. Therefore, the system locates peers with similar interests and histories and uses their past to recommend. Furthermore, it does not rely on machine-analyzable content, so it can accurately recommend complex items without understanding them. Collaborative filtering has four main problems [3]:

- **New Users**: the system needs more data to make accurate recommendations for new users [31].
- **New Items**: the collaborative filtering depends on the user's ratings to recommend. Therefore, it will recommend only items rated a substantial number of times.
- **Sparsity**: the most active users rate only a minor subset of items, so even the most popular entities have a few ratings [50].

For example, when recommending mobile applications to users on Google Play, the system would see that user A likes similar items as user B. User B watched a movie X, while user A did not. Therefore, the system would recommend movie X to user A [77].

4.2 Content-based Filtering

Content-based filtering tries to understand the similarities between items the user likes and recommends similar ones [3, 29]. Content-based filtering has two main problems [3]:

- **New Users**: the user needs to rate a sufficient number of items before the recommender can understand the user's preferences.
- **Overspecialization**: the user is limited to being recommended items similar to those that he already rated [77].

For example, when recommending mobile applications to users on Google Play, the system would see that user A likes healthcare applications and so would recommend another healthcare application to user A [77].

4.3 Hybrid recommendations approach

It utilizes many content-based and collaborative recommendation algorithms to avoid certain limitations. Netflix's recommendation system is a good example [31]. To win the Netflix Prize in 2007, the winners needed to combine 107 recommendation algorithms [9]. Recommendation methods can be combined in four ways [3]:

- Implementing content-based and collaborative methods separately and then combining them using voting or linear combination. Another approach is to choose one and use it.
- By incorporating content-based characteristics in collaborative filtering methods, developers can create and maintain profiles for each user based on their preferences and interests. These profiles can then be used to compare users and make recommendations based on similar content-based characteristics. The technique helps overcome the sparsity problems where only a few users have a significantly similar history. An extra benefit is that it can recommend highly scored items against the user's profile.
- Including collaborative characteristics in content-based methods can result in higher performance than for a purely content-based approach. One way to do this is to use the dimensionality reduction technique on a group of content-based profiles. Then, using latent semantic indexing (LSI), create collaborative views [76].
- Develop a general model with content-based and collaborative characteristics, where developers use content-based characteristics and collaborative characteristics in one classifier.

4.4 Mobile recommendation systems

Mobiles produce a lot of data that is useful for creating suggestions. The problem is that the data is hard to process and noisy [29]. Mobile recommendation systems (MRS) are

using this data and provide users with fast (there is no round trip to the server) offline suggestions while keeping all data private and having a low-power consumption [37].

MRS can be used for recommending routes and parking positions for taxi drivers [29]. The main frameworks are TensorFlow Lite and PyTorch Mobile. They both work on Android and iOS, and support hardware acceleration [37, 71].

Chapter 5

Analysis

This chapter presents an analysis of the target audience and their problems and needs, defined through a survey and previous research on time management techniques and principles. Furthermore, it compares the target audience's requirements with those of existing applications and explains why these applications are inadequate.

5.1 Research

The survey was conducted through social media and gathered responses from 42 individuals, mainly college students. The chart shown in Figure 5.1 indicates that nearly all respondents have experience using planning applications. Only two participants stated that they have never used planning apps and instead prefer physical objects. Furthermore, among those who reported experience using planning apps, only half of them mentioned using them on a regular basis.

In Figure 5.2, it can be observed that calendars, todo lists, and note-taking apps are the most frequently used types of apps by the respondents. Specifically, 30 respondents reported regularly using calendars, 18 reported using todo lists, and 22 reported using note-taking apps. It is noteworthy that, despite finding them helpful, 8 respondents stopped using a todo list app, 10 stopped using a note-taking app, 12 stopped using a timer app, and 10 stopped using a habit-tracking app. In Figure 5.3, it is possible to see that the main perceived benefits of planning are gained clarity about where time is spent, increased productivity, and being reminded about important events.

Figure 5.4 provides a more detailed explanation of the most crucial planning functions for the respondents. It is unsurprising that notifications are perceived as the most critical feature, given the use of planning mediums in the GTD theory to reduce stress and free up mental space. The second most important feature is integration with existing calendars, followed by the ability to create a concrete plan easily, and task organization. While features such as statistics and task suggestions are still important, they are perceived as slightly less crucial.

These findings validate the previous research on time management and provide valuable insights into the specific needs of users. Additionally, it is noteworthy that only 25% of the respondents are opposed to the concept of timeboxing, as mentioned in Figure 5.5.

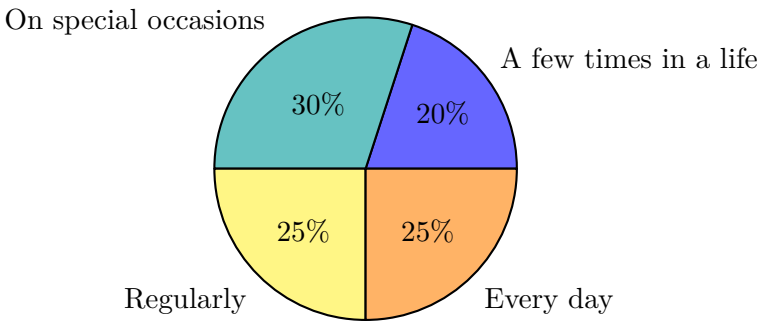

Figure 5.1: How much experience have respondents with planning applications

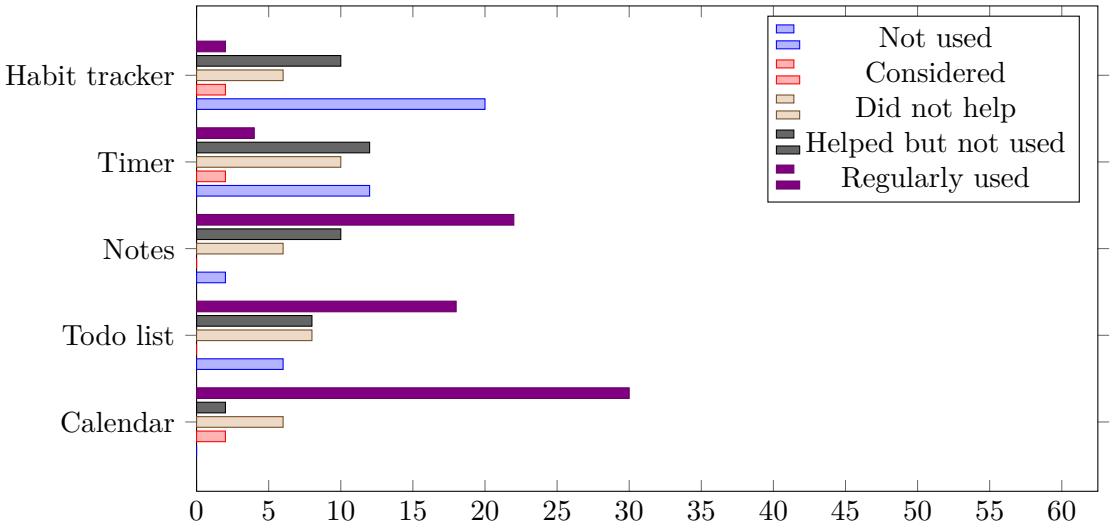

Figure 5.2: What experience have respondents with popular types of planning applications

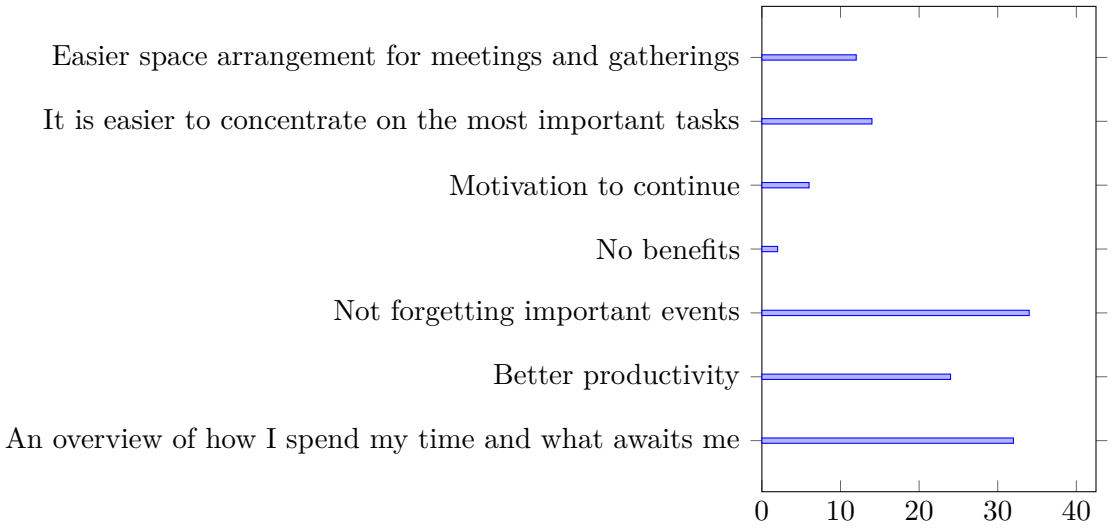

Figure 5.3: The main benefits of planning for respondents

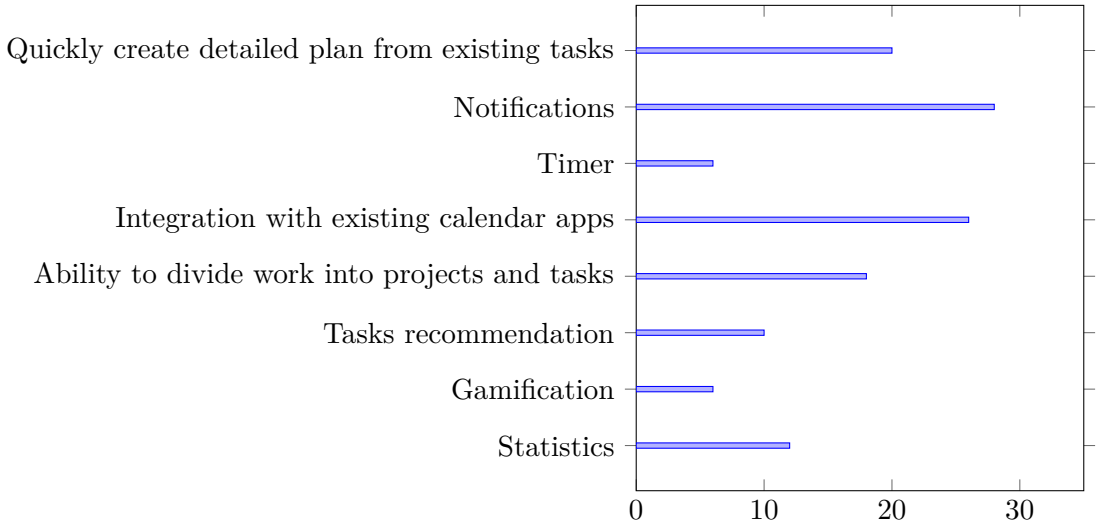

Figure 5.4: The most crucial planning functions for respondents

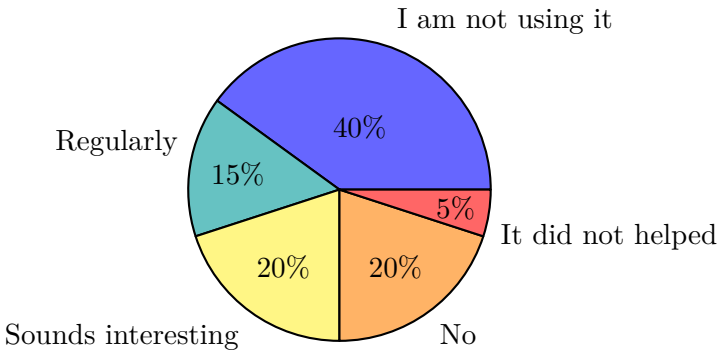

Figure 5.5: Experience with timeboxing of respondents

5.2 Persons

Based on prior research and surveys with potential users, three categories of potential users have been identified: managers, planning students, and non-planning students.

5.2.1 Manager

A manager is responsible for managing a team of colleagues, attending numerous meetings and handling various tasks related to them. To efficiently manage his busy schedule, he needs an app that integrates with his existing calendars, allowing him to view all events in one place and convert them into tasks within the app. Along with work-related tasks, he also has personal projects that he wants to track. The primary requirements of the app for a manager include:

- Ability to view calendar events within the app.
- Option to convert calendar events into tasks within the app.
- Seamless integration between calendar and todo list.

- Flexible organization of multiple projects.

5.2.2 Planning Student

A planning student has a partly fixed schedule (presentations, exercises) that is repeating — beyond that, he has organizational freedom. During this time, he must work on school and personal projects, prepare for exams, care for his health, and enjoy leisure activities. He aims to manage school burdens effectively, so he has time for personal projects. He keeps all school activities in a calendar app that gives him a clear view of the plan. Besides that, he uses a todo list to keep track of duties, but finds it difficult to put tasks from the todo app into the calendar app. The current setup also does not provide quality statistics. Sometimes, he uses the Pomodoro technique to include breaks into his day. He would like to see this included in one app with current tasks in the form of a tracker. The main requirements include:

- Viewing of calendar events in the app.
- Support for recurring events.
- Flexible organization of many projects.
- Tracker.
- Cooperation of calendar and todo list.
- Statistics for self-reflection.

5.2.3 Non-planning Student

A non-planning student uses a calendar only for special events such as tests or project presentations and prefers to keep everything else in mind. However, he is considering using a planning application to increase productivity and gain more clarity. The main challenge for him is using it regularly. To encourage consistent use, he would like features such as a clear perspective on where he spends his time, gamification to give him a sense of progress, and good integration with existing tools. The main requirements for such an app include:

- Ease of use.
- Statistics to track progress.
- Integration with existing calendar apps.

5.3 Existing Applications

The applications to be reviewed are those that best satisfy the previously defined requirements and are popular. Each review will contain two sections: an explanation of the rating and how well it satisfies the target audience, and interesting reviews from Google Play about the app. It is important to consider that managers and planning students are more focused on features, while non-planning students are more focused on ease of use.

5.3.1 Todoist

Figures 5.6 and 5.8 illustrate how Todoist[1] allows users to flexibly organize tasks using features such as task nesting, projects, labels, and searching. However, the app offers very limited statistics. Users can integrate Todoist with their existing calendars to view events, which are converted into tasks with deadlines at the start of the event. When a user creates a task with a specific deadline and duration using natural language processing (NLP), as shown in Figure 5.7, an event is created in the synchronized calendar starting at the specified time and lasting for the specified duration. This feature may be useful for scheduling events and leaving the remaining tasks to be executed based on a todo list, but it can be limiting for timeboxing purposes. Recurring events are easy to create, but they do not support specific time frames. Todoist also lacks a time tracker. Table 5.1 presents the app's rating according to the requirements defined by the target audience.

Users appreciate the ease of entering tasks, the support for labels and searching, and the ability to nest tasks. However, some users would like more calendar features.

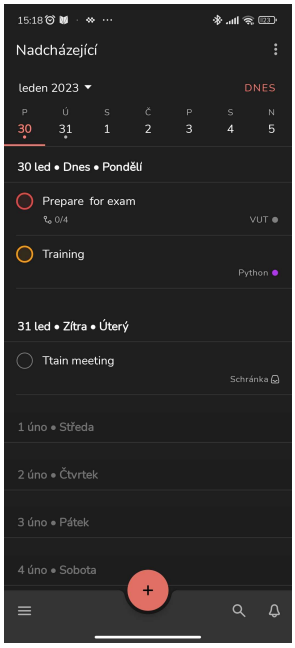

Figure 5.6: Todoist's timeline with todos

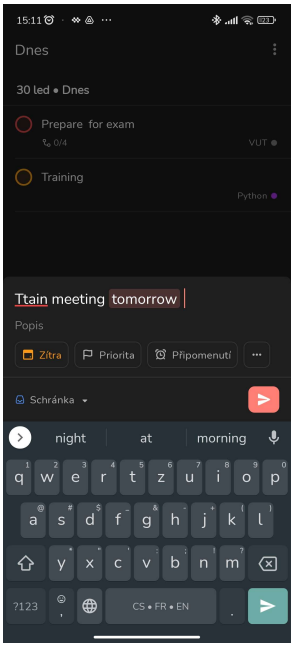

Figure 5.7: Todoist enables the creation of todos using NLP

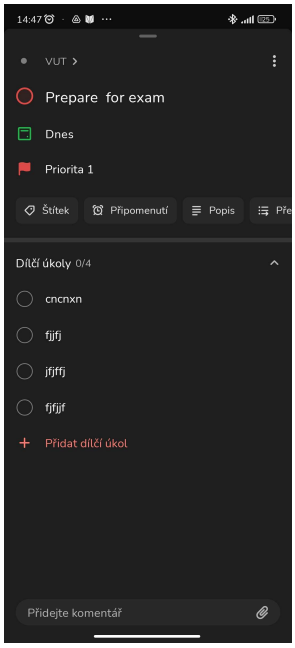

Figure 5.8: The user can modify a todo from a bottom sheet and also add subtodos

5.3.2 Focus Todo

Figures 5.10 and 5.9 demonstrate the features of Focus Todo[2]. The app allows users to create and organize tasks into projects, labels, and periods, while also supporting searching. Users can plan their work using estimates and work through their todo list accordingly. The app provides statistics, daily, weekly, and monthly reviews, and simple gamification through

[1] https://play.google.com/store/apps/details?id=com.todoist&hl=en&gl=US
[2] https://play.google.com/store/apps/details?id=com.superelement.pomodoro&hl=en&gl=US

Functions	Rating (1 – 10)
Ease of use	10
Statistics	2
Integration with existing calendar apps	6
Support for recurring events	7
Flexible organization of many projects	10
Cooperation of calendar and todo list	5
Tracker	1

Table 5.1: How Todoist would satisfy potential users' needs

a tree that grows as tasks are completed and tracked. Figure 5.11 shows the Pomodoro-style timer that users can use to track tasks and derive statistics. While this feature is useful for users who want to track all of their tasks, it may be unnecessary for occasional users. Currently, the app does not integrate with existing calendars. Table 5.2 presents the app's rating according to the requirements defined by the target audience.

Users appreciate the combination of the todo list and Pomodoro technique, as well as the easy-to-view progress and multiplatform support. However, some users would appreciate a dark theme as well.

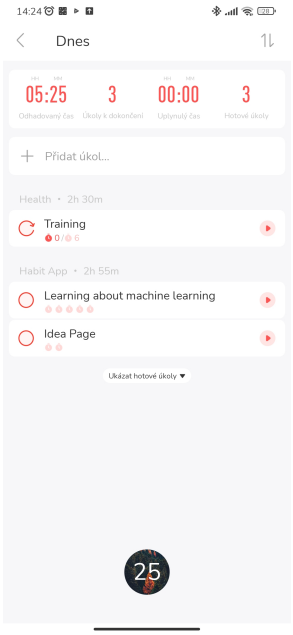

Figure 5.9: Today view shows today's events with their estimates

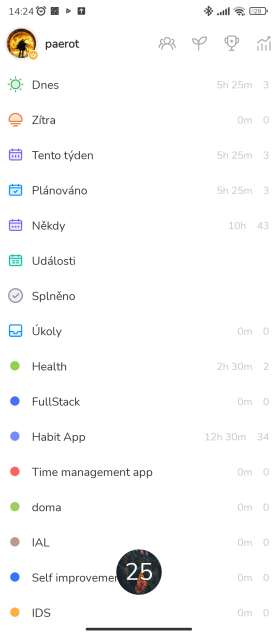

Figure 5.10: The user can also access tasks using views for tasks with a deadline today, this week, or this month

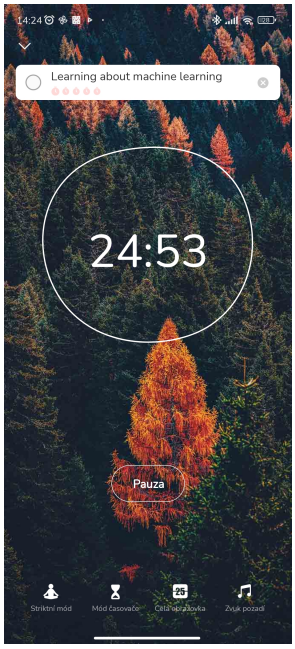

Figure 5.11: The timer in a Pomodoro style

Functions	Rating (1 – 10)
Ease of use	7
Statistics	8
Integration with existing calendar apps	1
Support for recurring events	4
Flexible organization of many projects	8
Cooperation of calendar and todo list	3
Tracker	8

Table 5.2: How Focus Todo would satisfy potential users' needs

5.3.3 Taskito

Taskito[3] is an app that enables users to organize their work into tasks, notes, events, and projects, as shown in Figure 5.13. It can also display calendar events from the device calendars, but it does not have the ability to create events in local calendars. The app's elements and local calendar events are displayed in the timeline, as shown in Figure 5.12. However, the statistics provided by the app are very limited, and it does not include a timer. Table 5.3 presents the app's rating according to the requirements defined by the target audience.

Users appreciate the timeline feature in the app, as well as the app widget shown in Figure 5.14. They also find helpful tips, tags, device synchronization, and templates to be valuable. However, they miss desktop and web support.

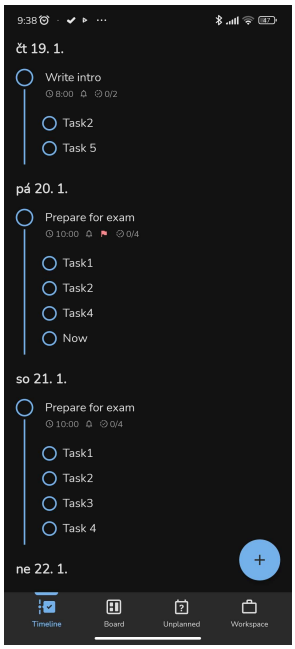

Figure 5.12: The timeline in the Taskito App

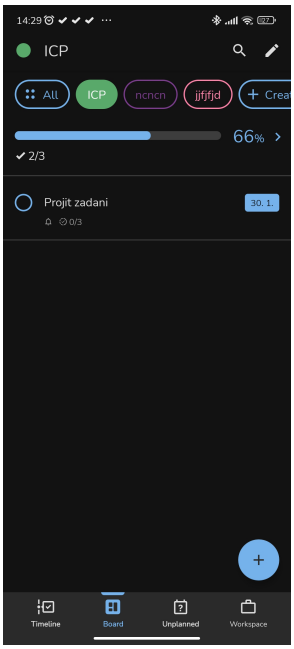

Figure 5.13: This view represents how the user can access tasks for a project

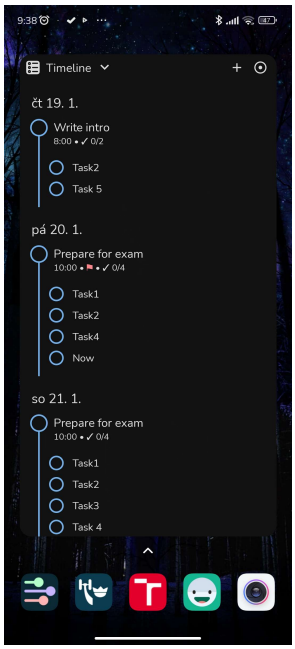

Figure 5.14: Taskito's app widget on Android

[3]https://play.google.com/store/apps/details?id=com.fenchtose.reflog&hl=en_US

Functions	**Rating (1 – 10)**
Ease of use	7
Statistics	2
Integration with existing calendar apps	5
Support for recurring events	6
Flexible organization of many projects	8
Cooperation of calendar and todo list	7
Tracker	1

Table 5.3: How Taskito would satisfy potential users' needs

5.4 Conclusion

The current productivity applications in the market do not fully satisfy the needs of the target audience, as they tend to focus on either a todo list or a calendar approach, but not both techniques together. While these applications are useful in their own ways, they do not cater to users who prefer the timeboxing technique with the flexibility of a todo list. Moreover, they often lack comprehensive statistics or gamification features that can motivate non-planners to continue using the app. Some apps also do not offer timer functionality, or have functionalities that depend on it. Therefore, there is a need to develop a solution that can better meet the requirements of the target users.

Chapter 6

Design of the Solution

This chapter presents the app design that aims to meet the needs of the target audience identified in the previous chapter, while also learning from the shortcomings of current implementations. The chapter begins by dividing the app into four main parts: the server database, local database, mobile device, and local calendar API. It then proceeds to define the requirements for the server and local databases, along with an explanation of how they should interact. Next, it defines the different categories of pages for the mobile device and their corresponding functionalities based on the target audience's needs. Finally, the chapter introduces planning algorithms that will help users prioritize tasks and schedule events more effectively.

6.1 Architecture

This section provides a high-level overview of the main components of the application, which will be discussed in detail in the following sections. Figure 6.1 depicts the architecture of the application.

6.1.1 Mobile Device

For reasons explained in Chapter 3, the primary platform for the application will be mobile. As Android and iOS are the two most widely used mobile platforms, it is reasonable to utilize cross-platform development that will enable the app to be created for both platforms with one codebase.

6.1.2 Local Database

Since the application will be working extensively with data locally, a local database is crucial. The main purpose of the local database will be full-text search, monitoring changes, and querying. It is essential for the database to handle relationships and be fast, especially considering it may potentially work with a large amount of data. Caching alone is not sufficient as it could have issues with queries, and not all data needs to be cached. Therefore, the application must store all user data locally.

6.1.3 Server Database

After analyzing existing apps, it was found that users highly value the ability to share data between multiple devices and have it securely stored in the cloud. Therefore, the application

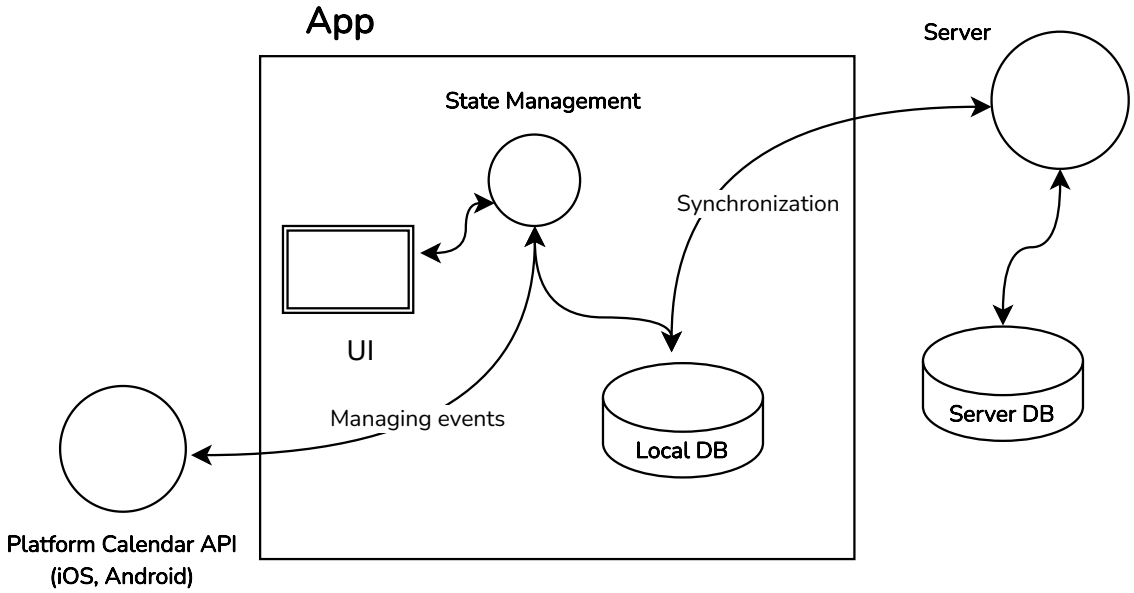

Figure 6.1: Application architecture

should include a server component to enable data synchronization between multiple devices and support authentication to ensure data privacy.

6.1.4 Local Calendar API

The integration between the app and existing calendar applications is crucial for users, as demonstrated in Figure 5.4. Hence, the application should have a framework that allows it to reflect events created within the app in existing calendars and display events from those calendars in the app. As iOS and Android offer APIs for accessing local calendars, it is sensible to implement this feature using those APIs, as the app can accomplish all the necessary actions through them.

6.2 Mobile Application

The mobile application is the primary component of the application where the user interacts with it. As shown in Figure 6.2, the complete navigation map was created based on the requirements of the target audience. This section will focus on discussing the four primary components of the mobile application.

6.2.1 Timeline

Based on user feedback, the calendar feature was found to be the most useful planning tool. Hence, the app will prioritize the calendar functionality, which will be implemented using a timeline view as shown in Figure 6.3. Since the todo list is also a frequently used feature, it is logical to integrate both planning strategies into a single application. This will enable users to define events in the timeline using tasks created through the todo list functionality.

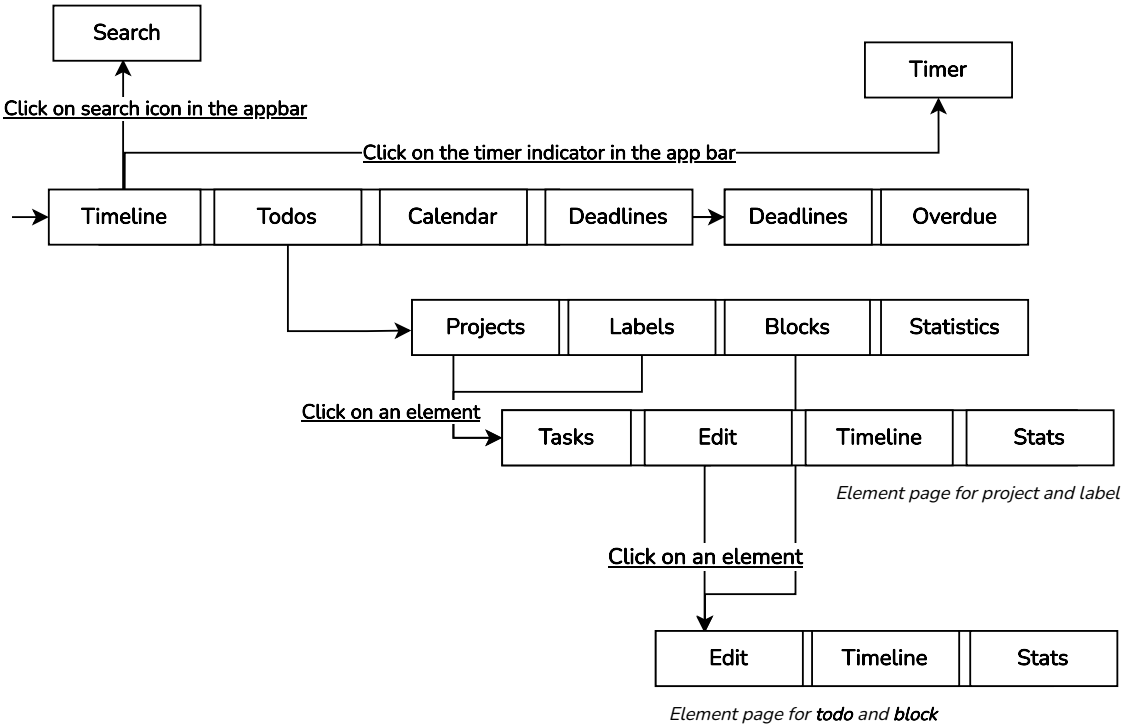

Figure 6.2: Application navigation schema

Events will only store the start and end times along with a link to the corresponding planning element. This way, the presentation of the event can be mostly delegated to the planning element. Many users have abandoned calendar applications due to their complexity in creating plans. Therefore, the app should provide a more user-friendly interface. It will recognize free spaces in the user's plan and offer various functions to flexibly fill them. When the user clicks on a free space, the app will show the following options:

- **Add to Timeline** – the user selects a task and its duration, and the app creates an event at the beginning of the free space that ends after the selected duration.
- **Add to Timeline with Custom Range** – the user selects a custom time range and a task to create an event that spans the selected time range.
- **Fill the Space** – the user selects a task, and the app creates an event that spans the entire free space.
- **Expand Above All** (if the event can expand above) – the selected event fills the available free space above. Similarly, `Expand Below All` fills the free space below.
- **Expand Above Pick** (if the event can expand above) – the user selects how much the event should expand above, and the app increases the event by that amount. Similarly, `Expand Below Pick` increases the event below.
- **Clever Planning** – the app offers an algorithm to automatically fill the free space with multiple events more efficiently.

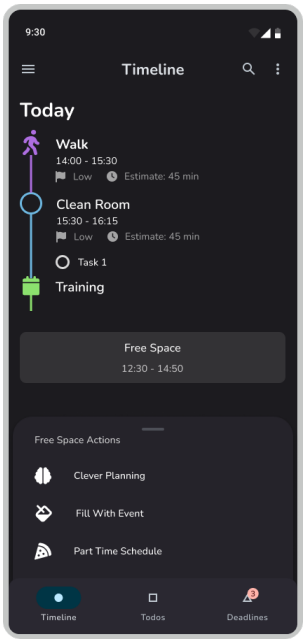

Figure 6.3: Home page timeline design including todo and block events, but also free space

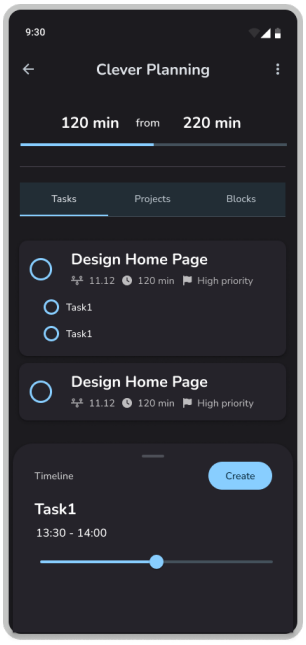

Figure 6.4: Clever planning page design

Figure 6.5: The deadline page design

Clever Planning

Clever Planning, shown in Figure 6.4, exists to enable the user to fill free space with multiple events more easily. It is a special environment that will suggest the most important elements (tasks, projects, labels, and blocks) that the user can work on. The user selects the elements they want to work on and can flexibly modify them. It should offer functions on individual elements, such as filling remaining space or removing them. Moreover, changing the order of elements or spacing them evenly is also an important option.

Deadline Timeline

Users highly value notifications and reminders to ensure they do not forget about important deadlines. Therefore, it is crucial to provide a feature that displays all their deadlines in one place. The design for the deadline timeline is presented in Figure 6.5. This feature will include all tasks, projects, and labels with a deadline, and show how they occur over time.

6.2.2 Pages for Organizing Tasks

The second most frequently used planning functionality is a todo list, which, as mentioned earlier, will cooperate with the timeline to enable the user to create plans. To make the organization flexible, the app needs to enable the creation of hierarchies. To accomplish this, the user will be able to group tasks into projects. However, for larger projects, grouping tasks into projects may not be sufficient. For this reason, the app will also include labels,

which will be in a many-to-many relationship with todos. This will enable users to create subgroups within projects and group similar tasks from multiple projects.

In addition to tasks, projects, and labels, the app will contain an element called a block, which will be useful for filling up space with often-performed actions such as reading, eating, or running. Since the app can potentially contain a large number of tasks, it should also include searching functionality, as shown in Figure 6.6. The bottom sheet design shown in Figure 6.8 will be used to create all the elements mentioned earlier.

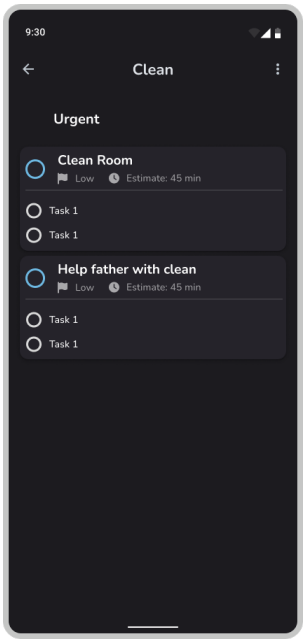

Figure 6.6: Search

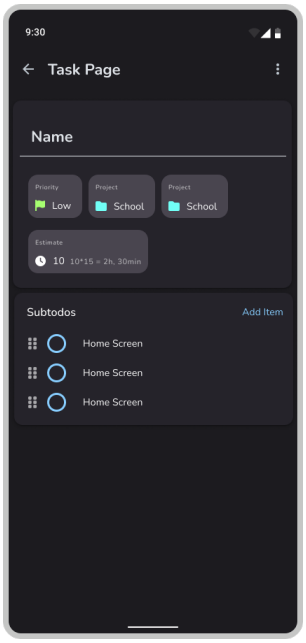

Figure 6.7: Edit page design

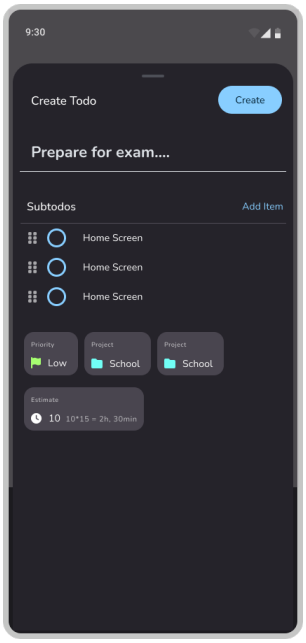

Figure 6.8: The design for a bottom sheet used to create a todo item

Viewing Planning Elements

The app will provide a page for viewing and editing planning elements, such as tasks, projects, and labels, as shown in Figure 6.7. Each planning element will have common attributes, including priority, creation date, and deadline, enabling the user to prioritize and manage tasks efficiently. Therefore, the app will allow users to sort and group planning elements by these attributes. To achieve this, an abstract element that contains these attributes will be created, making sorting and viewing reusable across the application, as shown in Figure 6.10.

Summary Statistics

To provide users with an overview of their time expenditure, the app will offer summary statistics that analyze planned time, how many times planning elements were planned, and how many related elements were completed for the label and project. These statistics will be presented in the form of a list or pie chart, and users will be able to choose the time range (day, week, month, year, or total) for which they want to view the data.

Figure 6.9: Timer page design

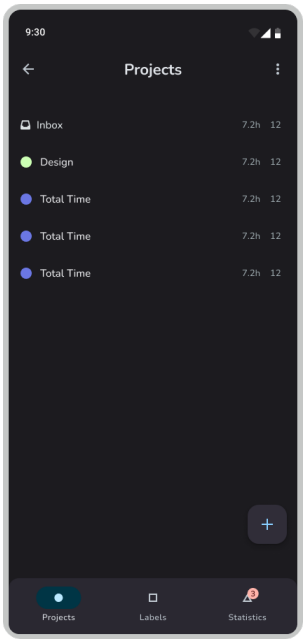

Figure 6.10: The design of viewing planning elements in a list

Specific Statistics for Planning Elements

In addition to summary statistics, the app will also offer specific statistics for individual planning elements. Users will be able to view a line chart that shows how an element was executed over time, as well as information on how many times it was planned and how long it was planned in total.

6.2.3 Timer

The majority of respondents did not use a timer or used it but stopped, according to the chart in Figure 5.2. However, a significant number of users found the timer feature helpful. To help users better manage their time and maintain focus on tasks, the app will include an independent timer feature. This feature will allow users to track their work sessions, breaks, or other time-related activities. The design of the timer page is presented in Figure 6.9.

6.3 Databases

The application heavily relies on database implementations, particularly on the local database that handles a substantial amount of data. However, the synchronization process between the server and local database assumes critical importance given the need to synchronize across multiple devices and backup data in the cloud, rendering the server database equally significant.

6.3.1 Local Database

The local database is crucial in three main contexts.

Timeline

The timeline loads all events scheduled for a particular day and promptly notifies the user of any changes in the query. It then displays these events, along with available slots in the user's schedule, and keeps track of all tasks with deadlines on that day.

Task Organization Pages

The search function is a crucial feature in the app as it can contain numerous tasks that can be challenging to locate. The search function should recognize matches from the beginning and end of words, and it should not require full matches. It should also be able to find elements that match all searched words when multiple words are entered. Additionally, it should be able to filter by the element's completion state.

When the user selects a project, the app should only display relevant tasks related to that project in the search results. Similarly, the user can select multiple labels and filter by them. The app should allow filtering by project and labels simultaneously to display subgroups within a project.

Statistics

To provide effective statistics, the application's database must support aggregation functions, in addition to indexes. It should also be able to locate linked objects, which places even more emphasis on the database's speed. For instance, to compute statistics related to labels, the database must first locate all the related tasks and then identify any associated events.

6.3.2 Server Database

In order to achieve proper authentication and synchronization between the server and the local database, it is recommended to utilize a Backend as a Service (BaaS) platform to expedite development. However, it is important to avoid utilizing services such as Firebase, which rely heavily on local caching, as this approach may not align with the local database requirements outlined earlier.

6.3.3 Data Model

Figure 6.11 illustrates the default data model that is required for the app's functionality. The data model consists of four main components: Todo, Project, Label, and Block, all of which have attributes such as priority, deadline (except for Block), and creation date. Projects with Todos are in a one-to-many relationship, while Labels with Todos are in a many-to-many relationship. Subtodos are stored in a list within these elements, eliminating the need to handle their relationship and allowing the user to search for words included in these subtodos. Another important attribute is estimateInMinutes, which enables the user to make a more realistic plan and is also used for creating recommendations. Furthermore, Todos and Blocks are connected to TodoEvent, BlockEvent, TodoRecurringEvent, and BlockRecurringEvent, which represent concrete executions in a one-to-many relationship.

The Rrule attribute stores information about event recurrence and is a string representation of the `rrule` used by local calendars such as ICalendar[1], along with a date when the event should end.

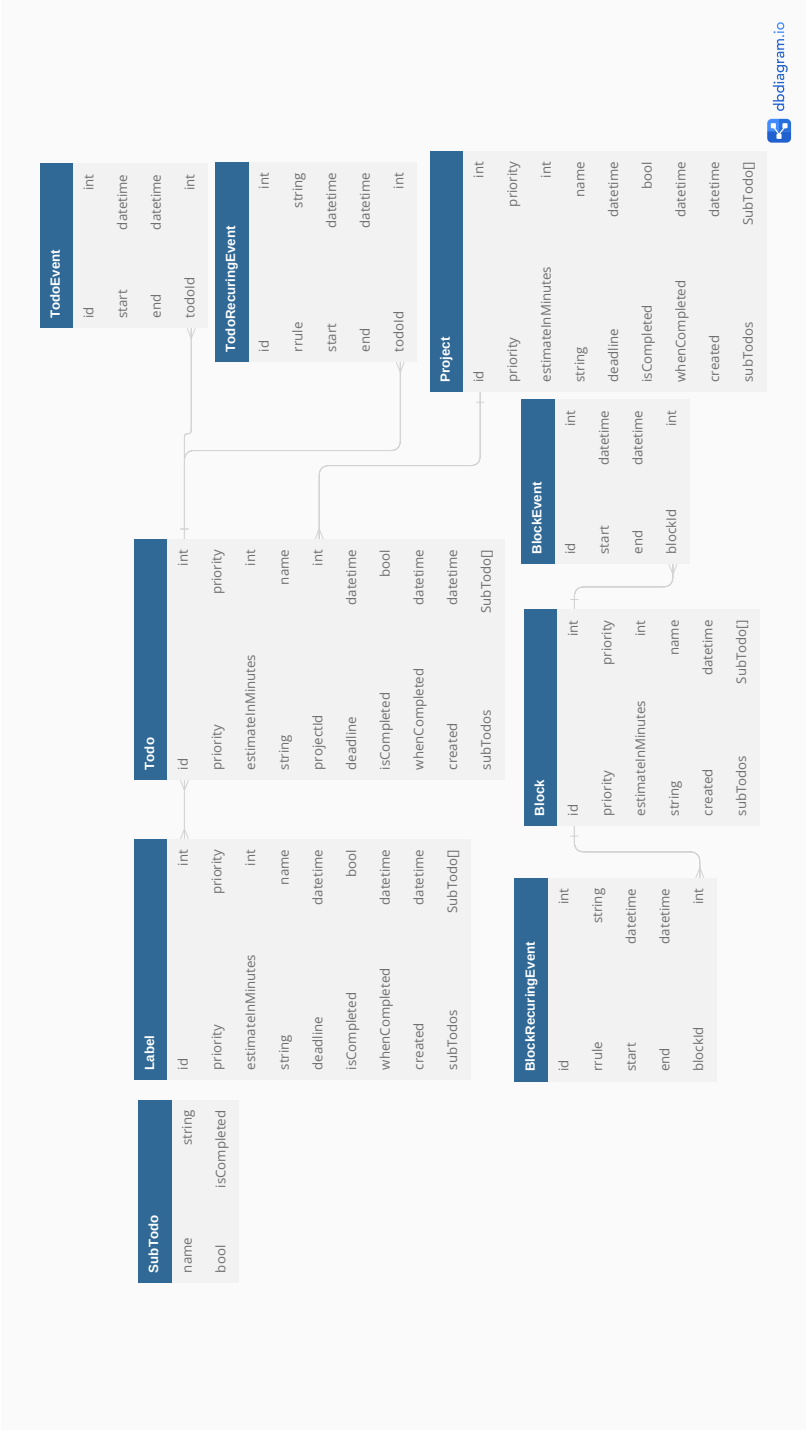

Figure 6.11: The app's data model.

[1]https://github.com/jakubroztocil/rrule

6.4 Smart Planning

To enhance the user experience, it is crucial to prioritize the development of intelligent planning functionality. This feature should be implemented only after the app's foundation is established, and the basic requirements of the users are met. Intelligent planning has the potential to significantly improve the app's usability and can be achieved in two ways:

6.4.1 Recommendation of Most Crucial Tasks

Implementing a recommendation system based on the priority, deadline, and estimate fields assigned to each organizational element in the app can enhance the user's productivity and aid in focusing on the most crucial tasks. The system can determine the significance of each element and display them in order of importance. However, creating the logic for this process can be challenging, so a data-driven approach is recommended. This involves generating a reasonable number of possible states and assigning values to them. A regression machine learning model can then be trained and converted to code. This approach can be used as another grouping option in the app or to recommend tasks when appropriate. It is important to note that classification is not recommended as it may be confused with an assigned priority, which could make the app more complex.

When assessing tasks, the deadline should hold the greatest weight as it signifies a necessary completion. It is important to distinguish between tasks with a deadline in a month versus those due in a day, which means that even a low-priority task within a low-priority project that has a deadline today should receive a high score. By implementing a recommendation system based on these factors, the app can help users prioritize their work and manage their time more effectively.

6.4.2 Suggesting Possible Event Terms

The app can suggest preferred times for the user to perform a selected action based on their history and available free spaces in their schedule. To achieve this, a histogram can be used to loop over all related events and store extracted information in a helper structure, shown in Listing 6.1.

```
class ScoreStructure {
  //<1 - 7> Count of events for each weekday
  List<int> days = [];
  //<1, 31> Count of events for each day of month
  List<int> daysInMonth = [];
  //<1, 5> Count of events for each week of a month
  List<int> week = [];
  //Example: {"13:00 - 15:00" : 3}
  Map<TimeFrame, int> timeFrames = {};
  //PartOfDay: morning, noon, evening
  Map<PartOfDay, int> partOfDay = {};
}
```

Listing 6.1: The basic scoring structure design

The app would then search for all available free spaces within the user's selected time span (although it cannot look infinitely into the future) and search for parts of free spaces

that would satisfy them using the found time frames. It would then score all found parts using other discovered information.

Chapter 7

Implementation

This chapter presents the implementation details of the app's functionality discussed earlier. It commences by discussing the app's architecture and the development tools employed in its creation. Then, it delves into the app's data synchronization mechanism and how it integrates with the calendar synchronization process. Furthermore, the chapter explains the implementation of the app's user interface components and the supporting structures and libraries used in the project. Finally, it describes the intelligent planning algorithms that the app utilizes to aid its users. The app was created using Flutter, a cross-platform framework, due to its fast performance, availability of relevant libraries, and the author's previous experience with it.

7.1 Architecture

In order to achieve sustainable development, the app is divided into four layers: infrastructure, domain, application, and presentation, as shown in Figure 7.1. This section will provide an overview of the architecture and its key components, which will be referenced later.

7.1.1 Infrastructure

The infrastructure layer is responsible for syncing the app with the server database, local calendars, and manipulating local data. It communicates with other layers through interfaces and returns and accepts domain models. The details of how this layer works are abstracted from other layers. The `DatabaseRepository` is an example of this layer, which retrieves data from the local database, modifies elements through synchronization repositories, and communicates with the Calendar API using the `device_calendar` plugin[1].

The `IDatabaseRepository` interface, shown in Listing 7.1, is used by the application layer to communicate with the infrastructure layer. This interface is primarily implemented using the Isar[2] package. Using this approach, it is possible to modify the infrastructure layer without refactoring the other layers.

[1] https://pub.dev/packages/device_calendar
[2] https://pub.dev/packages/isar

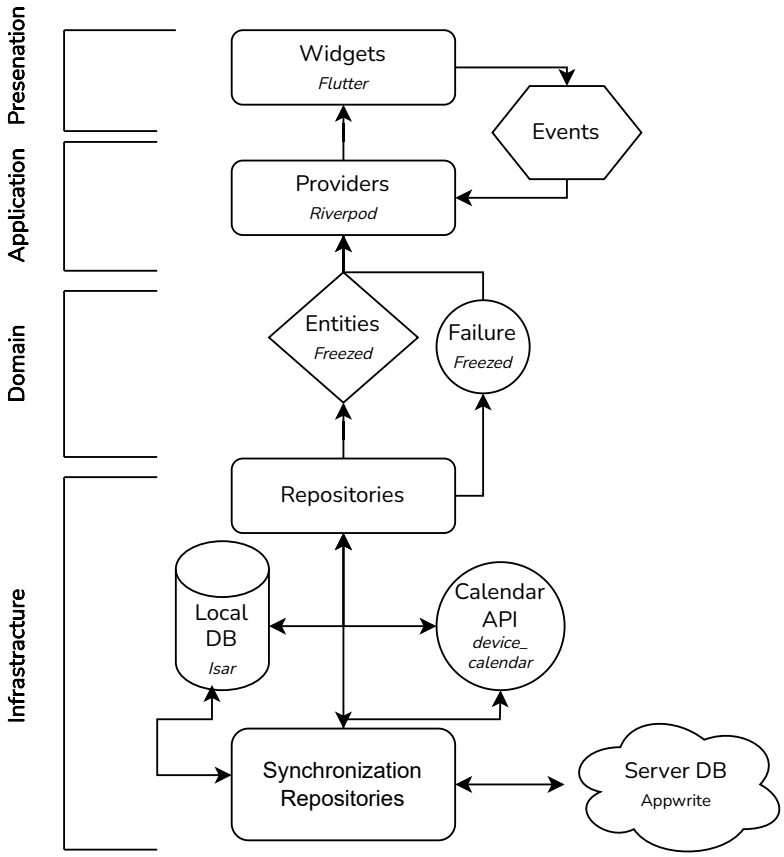

Figure 7.1: Mobile app architecture diagram

```
abstract class IDatabaseRepository {
  Stream<Todo> watchTodo(int todoId);
  bool createTodo(...);
}
```

Listing 7.1: A sample of the `IDatabaseRepository`

7.1.2 Domain

The domain layer is responsible for representing the business logic of the application, by encapsulating the business entities, rules, and operations that define the behavior of the application. It sits on top of the infrastructure layer and is decoupled from the presentation layer. Its main responsibility is to transform the raw data received from the infrastructure layer into a more meaningful and expressive representation that can be used by the application layer.

In this project, the domain layer uses the `freezed`[3] package to generate immutable classes that represent the various entities of the application. The use of immutable classes ensures that the objects can be shared between different parts of the application without risking data corruption. The most important classes defined in the domain layer are the

[3]https://pub.dev/packages/freezed

`Todo`, `Project`, `Label`, and `Block` classes, which represent the different elements shown in Listing 7.2 of the task management system.

Finally, the domain layer defines the `Failure` union, which represents the different types of errors that can occur in the application. This allows errors to be handled in a more systematic and uniform way throughout the application, making it easier to maintain and debug.

```
class Elements {
  Elements.todo(...) = Todo;
  Elements.block(...) = Block;
  Elements.project(...) = Project;
  Elements.label(...) = Label;

  // The identification of an element
  ElementId get elementId => ElementId(type: type(), id: id)
}
```

Listing 7.2: A pseudocode of the `Elements` union

7.1.3 Application

The application layer is responsible for managing the state and providing the presentation layer with access to local data as domain models. This project uses the Riverpod[4] framework, which is based on the principle of providers. Providers are objects that expose values and allow the state to be updated, making it easy to share and manage state across the app. An example of a provider in this app is the `TodoProvider`, shown in Listing 7.3. It provides the UI with the `Todo` domain model and automatically updates it when it changes, as the `IDatabaseRepository` exposes a stream. The `TodoProvider` also provides methods for modification.

```
class TodoProvider {
  Stream<Todo> build(int id) async* {
    //Listens for stream through the IDatabaseRepository interface
    await for (final v in ref.watch(databaseRep).watchTodo(id)) {
      yield v;
    }
  }
  //Method for modification
  void changeName(String newName) {...}
```

Listing 7.3: Pseudocode for the TodoProvider provider

7.1.4 Presentation

The presentation layer is responsible for displaying user information and responding to their actions by making changes through the application layer. To access Riverpod's providers, the UI uses a parameter called `ref`. The example provided in Listing 7.4 illustrates how the UI can obtain required information using the `ElementId` and then use the same `ElementId` to call the relevant modification method.

[4]https://pub.dev/packages/riverpod

```
return ColorCard(
    //Based on the type call the appropriate method
    onChange: (newColor) => type.when(
      //Using ref access the provider
      block: () => ref
          .read(block(elementId.id).notifier).color = newColor;
    ),
    //Access elements's color, while being notified when it changes
    color: ref.watch(eleColor(elementId)),
```

Listing 7.4: The pseudocode of the ElementSelectionColorCard widget

7.2 Infrastructure

This section provides an explanation for the selection of Isar[5] as the local database and Appwrite[6] as the server database. Additionally, the section provides details on how the application's data is synchronized with the Appwrite server and how events in the app are reflected in the synchronization calendar.

7.2.1 Databases

The app's reliance on local data underscores the importance of having a dependable local database. Extensive research and benchmark testing led to the selection of a suitable local database. Additionally, it was crucial that the local database could synchronize seamlessly with the server database to ensure consistency between the two.

Local Database

In the SQL realm, there is a plugin sqflite[7] which is the classic port of SQLite from C to dart. SQLite is considered robust and battle-tested over time[8]. There are two major wrappers for sqflite in Flutter - drift[9] and floor[10], which were chosen based on pub points. The biggest advantage is the relational structure and the core sqflite that will stay even if the used wrapper project will not be maintained.

- Drift adds code generation, caching, and threading support.
- Floor adds code generation.

When it comes to the No-SQL databases, there are several options available for handling relationships as required by the app. ObjectBox[11], Isar[12], and Realm[13] (in Beta) are among the most popular choices (based on pub points).

[5] https://pub.dev/packages/isar
[6] https://pub.dev/packages/appwrite
[7] https://pub.dev/packages/sqflite
[8] https://www.sqlite.org/index.html
[9] https://pub.dev/packages/drift
[10] https://pub.dev/packages/floor
[11] https://pub.dev/packages/objectbox
[12] https://pub.dev/packages/isar
[13] https://github.com/realm

- ObjectBox is partly commercial software with high speed, ACID semantics, and relationships, and is statically typed. The optional synchronization option is questionable because you can ask for a free trial for a month and additional information is missing[14].
- Isar is a database that is statically typed and supports ACID semantics, queries, asynchronous, full-text support, relationships, and migration support.
- The Realm is a port of a very popular framework to dart owned by MongoDB. It offers ACID, synchronization, relationships, and queries.

To evaluate the performance of the chosen local database, a set of tests were conducted using modified tests from the Isar author that incorporated relationship data[15]. Additional tests were conducted to compare Isar with drift (a sqflite wrapper) and to evaluate how Objectbox, Isar, and Realm handle relationships. The non-relational tests ran on 50000 items, while the relational tests were conducted on 200 projects linked to 20000 tasks.

The findings from the conducted tests, presented in Figure 7.2, indicate that Isar outperforms ObjectBox and Realm in various areas, including CRUD operations, filtering, and relationship handling. ObjectBox performed slightly slower, while Realm exhibited significantly slower performance. Furthermore, Isar maintained a more compact database than ObjectBox, while Realm often encountered crashes when working with a large number of objects.

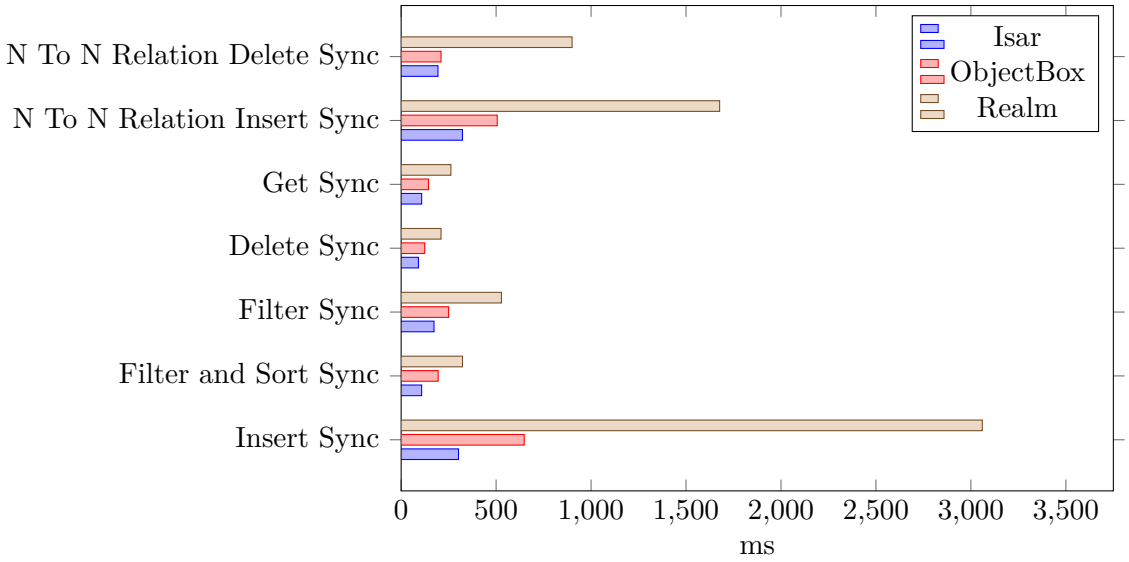

Figure 7.2: Comparison of No-SQL databases: Realm, Isar, ObjectBox for Flutter

Figure 7.3 shows the comparison of Isar with Drift. Drift takes only half the space as Isar. Outside of that, Isar is significantly faster. Based on the benchmarks, it is evident that Isar performs better than other databases in all key operations. The only comparable option is Realm, which has built-in synchronization capabilities that can greatly reduce development time.

[14] https://objectbox.io/sync/
[15] https://github.com/Musta-Pollo/isar_benchmark

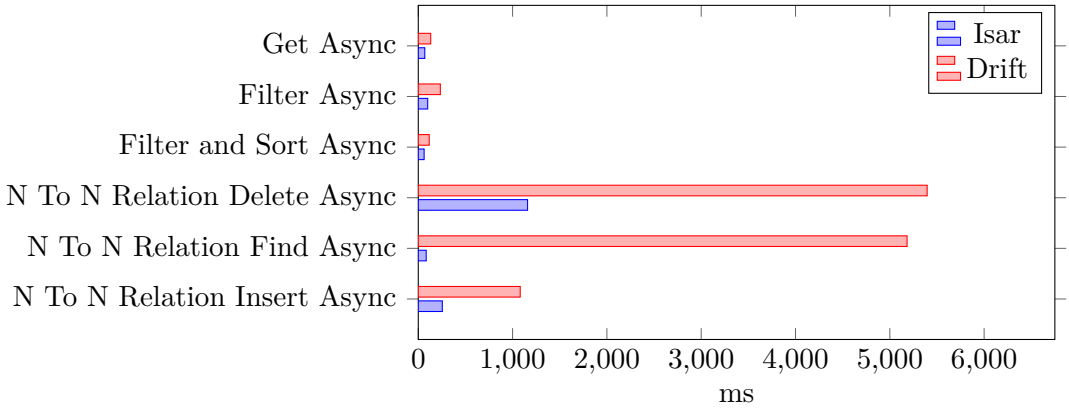

Figure 7.3: Comparison of Isar with Drift

Server Database

To save time and focus on the app features it is possible to use BaaS that implements almost all needed functions. BaaS is a model for providing web app and mobile app developers with a way to link their applications to backend cloud storage and APIs exposed by backend applications while also providing features such as user management. These services are provided via the use of custom software development kits (SDKs) and application programming interfaces (APIs)[16].

As previously mentioned, the application requires server storage with authentication. Firestore, which was also previously mentioned and can be found at[17], is a popular candidate that provides a sturdy document-based architecture, allowing for flexible handling of relationships, synchronization, and integration with Flutter. However, due to the use of caching in synchronization, Firestore would not be used in this application.

An alternative to Firestore for server storage with authentication is Supabase[18]. It is built on top of a PostgreSQL database and offers additional features such as edge functions, real-time subscriptions, and authentication. However, it does not support offline functionality, and the Flutter plugin does not support the Linux platform[19].

Another option to consider is Appwrite[20]. It aims to simplify backend management by providing authentication, a database, cloud functions, and out-of-the-box self-hosting. While Appwrite uses MariaDB as its database, it tries to emulate a No-SQL database, based on Wix's experience when they started using MySQL as a key-value storage system [1]. However, offline support is not available.

MongoDB is a popular choice for backend storage, and its integration with Realm provides features such as object modeling that allows for the translation of code-generated models to MongoDB schema with relationships and synchronization[21]. However, despite offering synchronization, offline-first storing, and authentication features, Realm will not be used for this application due to previous research. The non-static API for querying and poor performance of Realm make it an unsuitable choice for the app's needs. After eval-

[16] https://www.cloudflare.com/en-gb/learning/serverless/glossary/backend-as-a-service-baas/
[17] https://firebase.google.com/docs/firestore/quickstart#dart
[18] https://supabase.com/
[19] https://pub.dev/packages/supabase_flutter
[20] https://appwrite.io/
[21] https://www.mongodb.com/docs/realm/sdk/flutter/

uating various synchronization servers, it became apparent that using Isar in conjunction with a synchronization server is a more suitable option, despite the added time required for implementation. This is because a slow database would be a significant issue for the application.

The available options have been narrowed down to Supabase and Appwrite. Appwrite is the preferred choice because of its user-friendly interface that eliminates the need for SQL coding. Furthermore, it supports relationships through list structures, which are not as powerful as using a SQL relational structure but are still feasible and easier to integrate with the local database. In addition, Supabase has a limited number of real-time subscribers[22] compared to Appwrite. With Appwrite, all data changes can be loaded at the start of the app, and the server parameters are the only limiting factors when listening for changes using WebSockets.

7.2.2 Synchronization

As previously mentioned, synchronization is a crucial aspect of the app, enabling data to be synchronized between the local Isar database and the Appwrite server while also reflecting these changes in a local synchronization calendar. The synchronization process uses helper local attributes, including `lastUpdated`, `lastSynced`, `deviceId`, `lastCalendarSync`, `eventId`, `calendarId`, and `ownerDeviceId`, which are stored on synchronized elements. On the server, Appwrite adds a `lastUpdated` attribute to every document that is updated or created. This attribute is used in conjunction with the stored `deviceId` to retrieve updates from the server. Each document on the server is identified by a `documentId`. The local attributes used in synchronization have the following meanings:

- `LastUpdated` (local) represents the most recent modification time of the element.
- `LastSynced` represents the last time the element was successfully updated on the server.
- `DeviceId` identifies the device that last modified the element.
- `LastCalendarSync` represents the last time the element was synchronized with the synchronization calendar.
- `CalendarId` and `eventId` identify the corresponding calendar event.
- `OwnerDeviceId` identifies the device responsible for synchronizing the event in the synchronization calendar.

Application Data Synchronization

Upon launching or returning from the background, the app synchronizes with the server using the `totalUpdate` function as shown in Listing 7.5. This function contacts the server and retrieves the `lastUpdated` value of the record associated with the current `deviceId` using the Databases API. The `lastUpdated` value represents the last time when the `totalUpdate` function was successfully finished and the server was contacted to store this value, marking a point in time when the local database and server database were fully synchronized.

[22] https://supabase.com/pricing

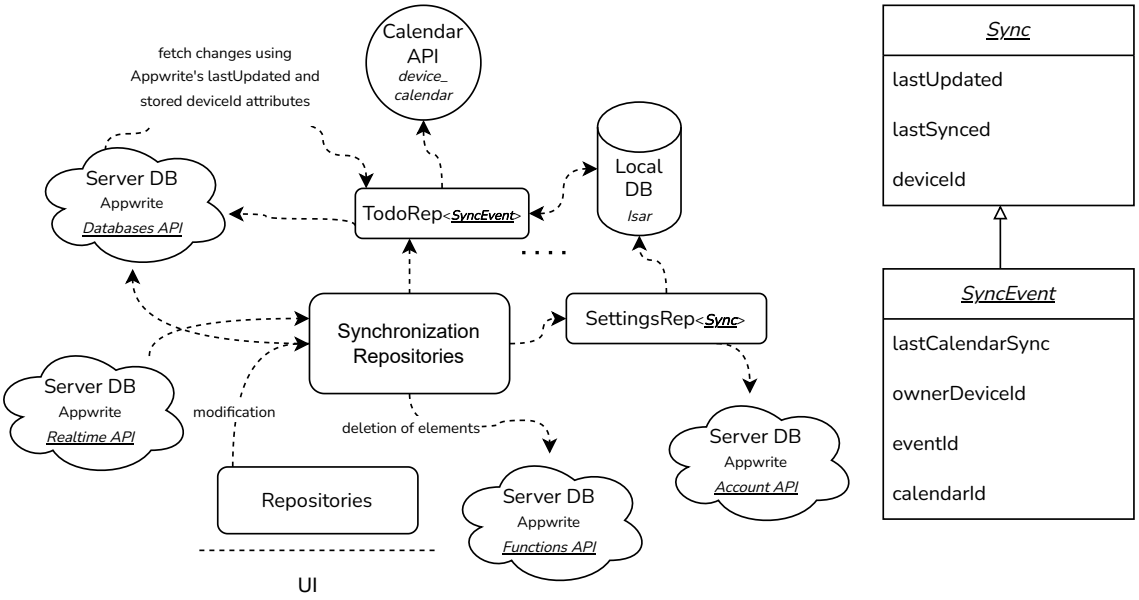

Figure 7.4: Synchronization schema consisting of synchronization repositories, the local database, the calendar API and the server

The app then requests all deletion records created after that datetime (Databases API) and deletes all local elements associated with them. A deletion record stores all documentIds for each app's element that were using a Functions API previously deleted. Next, each of the synchronization repositories requests all updated records after that datetime, converts them to the app's elements, and compares them with the currently stored data. If the updated records are newer, they are updated in the app's database. Any elements not recognized by the database are created as new entries. It recognizes elements using the server's `documentId` and uses it also to handle relationships with the help of Isar's links[23].

Important to note that `getChanges` and `updatedFromChanges` are deliberately separated steps. Because it is necessary to update elements in a particular order due to relationships it is more effective to load all changes at once and only after all changes are received, updated them in a specific order, as the most time-consuming part is waiting for the server responses.

```
Future<void> totalUpdate() async {
  DateTime lastUpdated = await getLastUpdated(); //When was last
completed totalUpdate
  await getDeleted(lastUpdated); //Get all deleted records
  await getUpdateSettings(); //Using Account API ask for preferences
  await getChanges(lastUpdated); //Repositories ask for changes
  await updateFromChangesAll(this); //Repositories save changes
  await updateLastUpdated(start);
}
```

Listing 7.5: The pseudocode of the totalUpdate function

[23] https://isar.dev/links.html

Furthermore, there is a background service that regularly runs a function shown in Listing 7.6 that scans for any new or updated elements. It specifically searches for unsynchronized elements (with `lastUpdated` not equal to `lastSynced`). Afterward, it sends requests to either create or update these elements (Databases API). For deleting elements, a cloud function (Functions API) is utilized to ensure that elements are deleted in a particular order due to relations. A deletion record is also created to enable other devices to reflect the deletion in their local database 7.6.

```
Future<void> pushChangesToServer() async {
  await Future.wait([
    ...repositories.map((e) => e.pushChangesToServer(this)),
    deleteFromServer(), //It creates DeleteRecord and deletes elements
    pushSettingsToServer() //It stores settings in the user's
preferences using Account API
  ]);
}
```

Listing 7.6: The pseudocode of the pushChangesToServer function

At the same time, a realtime service utilizing Realtime API listens for changes and then delegates modifications to appropriate synchronization repositories. As WebSockets are closed after time off inactivity, the app closes this service after the app goes to the background. The service starts listening again when the app is reopened and after `totalUpdate` is executed.

Calendar Synchronization

To reflect app events in calendar apps and make it more understandable, the user needs to select a device on which he wants to do the synchronization and a calendar in which the app should create events. As the app synchronizes, any changes made are also reflected in the synchronization calendar. The user can also view events from device calendars in the app. It utilizes device_calendar plugin[24] that provides helper functions such as `createOrUpdateEvent`, `deleteEvent`, `deleteEventInstance`, `requestPermissions`, `retrieveCalendars`, and `retrieveEvents`.

The retrieval of events in the application is managed by the `CalendarRepository`, which utilizes the helper functions of the device_calendar library to retrieve events from the user-selected calendars and stores them in the local database for caching purposes. This caching mechanism results in a faster loading of the user interface, with updates made only when there are differences in the calendar events.

In addition, a background service frequently checks for new or updated app events. It searches for unsynchronized elements (`lastUpdated` not equal to `lastCalendarSync`) with `ownerDeviceId` equal to null or the current `deviceId`. After the update or creation of an event, it stores the `eventId`, and the `calendarId` of the created event together with the `ownerDeviceId` on the element while updating `lastCalendarSync` to be equal to the `lastUpdated` value.

[24] https://pub.dev/packages/device_calendar

7.3 Mobile Application

All features specified in the app design have been successfully implemented. This section focuses on the key libraries and algorithms used to implement the user interface and associated logic. The app heavily relies on Flutter's Material Components for efficient development and to enhance the consistency and clarity of the user interface.

7.3.1 Timeline

The timeline feature of the app combines app events with calendar events to generate a list of events, represented by the `Events` union. The user's defined `startWorkingTime` and `endWorkingTime` are used to calculate free time slots in the user's schedule, and the free time slots and events are displayed in the UI. To render the events in the UI, the timeline_tile[25] library is used. If the user clicks on an event, a bottom sheet is displayed for event modification, using the modal_bottom_sheet[26] library.

The clever planning page features a draggable sheet for creating events, using the snapping_sheet[27] library. This sheet is displayed persistently and can be dragged to different positions on the screen.

7.3.2 Pages For Organization Of Tasks

When the app is showing elements in a list and enables them to be grouped by their attributes it uses the grouped_list[28] library.

7.3.3 Statistics

Statistics use heavily the fl_chart[29] plugin that enables it to be interactive. It specifically uses fl_chart's elements: `PieChart`, `BarChart`, and `LineChart` and their associated structures.

7.3.4 Timer

The timer state and settings in the app are stored in the `TimerSettingsInfo` union, which allows for synchronization across all devices for the same user. This ensures that the user has the same timer state on different devices. The timer is based on the Pomodoro technique 2.3.1 and its state includes the timer configuration and the current timer state number `timerStateNum`. When a user starts a timer, the app stores the datetime when the timer should end as `finalTimerDatetime` in the `TimerSettingsInfo` union. On another device, the provider that holds the timer is rebuilt when the state is synchronized, and it uses `finalTimerDatetime` to start counting down to that datetime. It also checks the `timerStateNum` and the timer configuration to determine if it is a working time, a break, or a long break.

When an user pauses the timer, the `tickerNum` is stored in the `TimerSettingsInfo` union, representing the number of seconds until the timer's end. On another device, this

[25] https://pub.dev/packages/timeline_tile
[26] https://pub.dev/packages/modal_bottom_sheet
[27] https://pub.dev/packages/snapping_sheet
[28] https://pub.dev/packages/grouped_list
[29] https://pub.dev/packages/fl_chart

information is used to determine that the timer is paused and how many seconds remain until the end.

7.4 Smart Planning

The application incorporates two distinct planning mechanisms that assist the user in prioritizing important tasks or finding available time when scheduling events. The first utilizes machine learning algorithms to score planning elements based on their importance and subsequently recommends tasks to the user in that order. The second examines the history of related elements and the user's available free space in their plan to suggest appropriate scheduling slots for the element being scheduled.

7.4.1 Recommendation Of Most Crucial Tasks

This algorithm organizes elements based on the user's input, such as priority, deadline, estimate, and related projects and labels. It uses a regression machine learning model that was trained on created data sets. It contains two datasets: the project and label dataset and the todo dataset.

The datasets are rated in a way that elements with a deadline must be completed. Because of that deadline has the most impact on the score. Very important is also project and label score as it is assumed that all related tasks should be affected. Medium influence on the score has a priority and the least important is considered an estimate. The maximum score is 100 and the lowest is 0.

To generate the project and label dataset, priority, days until the deadline, and the difference between the estimate and the elapsed duration were used, as shown in Figure 7.7. To facilitate scoring, only the most important members for each value were included. It is assumed that the model will interpolate the missing values, which are represented as -1, as machine learning models may struggle with missing data. Idea and none priorities were not included, as they receive an automatic score of zero.

```
#0 - urgent, 1 - important, 2 - low
priority = [0.0, 1.0, 2.0]
daysTillDeadline = [0.0, 7.0, 21.0, -1.0]
estimateTimeInMinutesMinusElapsedProjectLabel = [800.0, 3000.0, -1.0]
```

Listing 7.7: Used values for the generation of project and label dataset

The Todo dataset, depicted in Figure 7.8, adopts a similar approach to the Project and Label dataset, but it also takes into account the project and assigned labels of the todo. The scores for the project and labels are the output values of the machine learning model for project and label. For the label score, it uses the highest importance label assigned to the todo. This guarantees that even a low-priority task that belongs to a highly important project will receive a high score.

priority	daysTillDeadline	estimateTimeInMinutesMinusElapsed	score
0	0	3000	100
2	-1	-1	0
1	7	-1	60
2	21	800	42
0	-1	-1	30
1	0	3000	80
2	21	-1	40

Table 7.1: Project and label dataset example values

priority	daysTillDeadline	estimate...	projectScore	maxLabelScore	score
1	7	800	20	90	73
2	-1	800	20	90	25
1	21	800	90	60	55
2	21	60	60	4	40
0	-1	800	20	90	38
0	7	800	4	4	65
0	-1	60	20	90	35

Table 7.2: Todo dataset example values

```
#0 - urgent, 1 - important, 2 - low
priority = [0.0, 1.0, 2.0]
daysTillDeadline = [0.0, 7.0, 21.0, -1.0]
estimateTimeInMinutesMinusElapsedTodo = [60.0, 800.0]
projectImportance = [4, 20.0, 60.0, 90.0]
maxLabelImportance = [4, 20.0, 60.0, 90.0]
```

Listing 7.8: Used values for the generation of todo dataset

The machine learning model was selected from the scikit-learn library, a popular library that interfaces well with code-generation libraries. To choose the best machine learning model for the project, the machine learning map[30] from the scikit-learn documentation was consulted, and after testing, the `RandomForestRegressor`[31] was selected for its alignment with the scoring design and its negligible impact on the UI. The machine learning model is converted to Dart using the m2cgen library[32].

7.4.2 Suggest Possible Event Terms

This feature heavily relies on the UI to support it. Without it, there would be no value in it for the user as he would only see a suggestion and nothing else. It cooperates with the calendar page in the app that shows all events and available free spaces in a day, three days, or a week layout. When the user enters this mode, part of the calendar page is delegated to show a list of suggestions represented as cards. When then the user clicks on one of

[30] https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
[31] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
[32] https://github.com/BayesWitnesses/m2cgen

the suggestion cards, he is navigated to that specific time in the calendar. Because of this integration with a calendar, the user can see what events are planned around it, enabling him to make a more confident decision about the value of this suggestion. At the same time, he can also choose non-suggested time frames. It uses modified 7.9 structure as described in the application design. In addition, it also gives a higher score to parts that are closer to the average duration of past events.

```
class ScoreStructure {
  List<int> days = [];
  List<int> daysInMonth = [];
  List<int> week = [];
  DateTime? lastEventDate;
  List<int> intervalInDaysBetweenEvents = [];
  Map<TimeFrame, int> timeFrames = {};
  Map<PartOfDay, int> partOfDay = {};
}
```

Listing 7.9: Modified scoring structure from the design of solution

Chapter 8

Testing

This chapter discusses the feedback gathered from a survey and personal interactions regarding the application. It also includes the opinions of potential users at Excel@FIT[1].

8.1 User Testing

The application was shared with potential users, primarily college school students, via Facebook. They were first asked to try the app and provide feedback. After a week, they were asked to submit a survey using a link to a Google Forms survey. Thirteen respondents tested the app and submitted the survey. Figure 8.1 shows that all respondents used the app for over five minutes, and almost half used it for over an hour. The app was understandable, with an average score of 3.9 out of 5. The design was also attractive, scoring 4.75 out of 5, if one outlier rating of 1 is excluded. Nine respondents tried the testing environment, and all of them appreciated it. Others missed it or did not use it. The users' most significant issue was the lack of classic todo list functions, such as a view for tasks with a deadline today, this week, or this month. The second biggest issue was creating events or adding subtasks and notes.

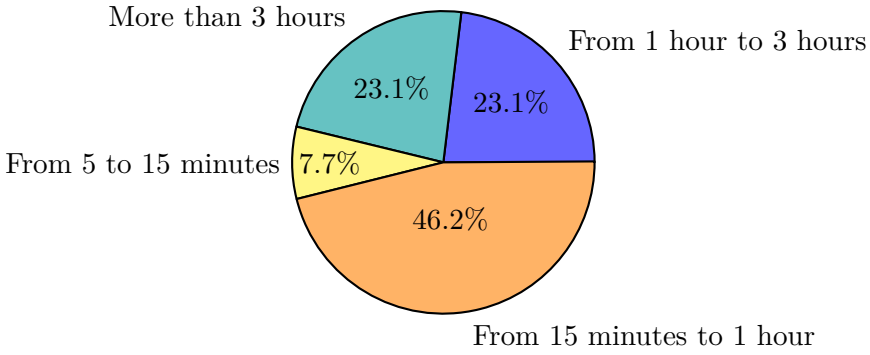

Figure 8.1: How long the respondents used the app before submitting the survey

On the other hand, Figure 8.2 shows that users appreciated mainly the app design, organization hierarchy, clever sorting of tasks, and the ability to view calendar events in the app. Lastly, when the testing users were asked how well the application satisfies their needs, their average score was 3.8 out of 5.

[1] https://excel.fit.vutbr.cz/

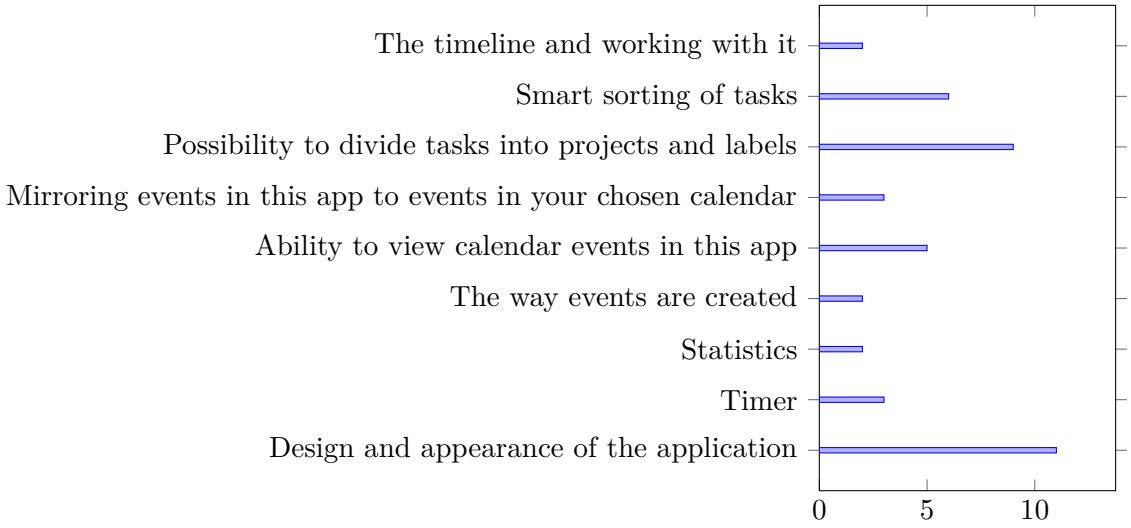

Figure 8.2: What respondents of the survey found interesting about the app

8.2 Personal Feedback From Potential And Actual Users

Outside of the survey the app also got feedback from users who were previously asked to test the app using Facebook, but chose their opinion express using personal messages. Outside of visual or translation issues, they also requested more quality notifications for events, but also for tasks with a deadline.

Another source of feedback was a student conference called Excel@FIT[2] at the Faculty of Information Technology of VUT. The feedback was from people to whom the application was shortly presented. The app attracted attention by its design and its many features. On closer look, viewers appreciated the idea of combining todo list with a calendar, integration with existing calendar applications, and recommendations of tasks.

8.3 Conclusion

After analyzing the feedback received from the testing phase, it has been identified that the application needs improvements in two main areas: addressing translation issues and enhancing the notification system for planning elements and events. While the app's complexity can be attributed to its multifunctionality, the feedback suggested that there is room for improvement in the user interface to enhance its usability. One of the respondents reported feeling overwhelmed by the amount of information presented, stating that they got lost in the app at times. To tackle this issue, one potential solution could be to use the showcaseview plugin[3] to demonstrate and showcase the application's features. The feedback received is considered valuable and provides guidance on how to make the app more user-friendly and effective.

[2]https://excel.fit.vutbr.cz/
[3]https://pub.dev/packages/showcaseview

Chapter 9

Conclusion

The objective of this project was to develop an application that combines various time management techniques to provide a better planning experience for users compared to existing implementations. To achieve this goal, popular time management terms, such as the todo list or calendar, were studied, which enabled the understanding of why these techniques work and how they could be combined. A survey was conducted, which revealed that the calendar and todo list were the most important techniques for users, and many respondents were not against using timeboxing to a certain extent.

Based on the insights gained from the study, three target user personas were defined, and their needs were analyzed against existing applications. However, the current implementations failed to meet the diverse needs of the target audience in a single app. Therefore, the application was created using the Flutter multiplatform framework, which allows for easy expansion to all major platforms. The app is currently available on Google Play[1] and App Store[2].

The application that has been developed offers a combination of classic todo list functionality with a timeline and traditional calendar views. Additionally, it provides a Pomodoro tracker and enables users to view their planning history through statistics. The app uses user-defined attributes to recommend the most appropriate tasks and offers time frames for planning events based on related element history. The architecture of the application is layered, allowing for easy extension and infrastructure modification without affecting other layers. To support the local database requirements, the app utilizes self-managed synchronization. This implementation makes it less expensive to maintain long-term since the user only downloads server changes.

Sharing the app with potential users showed that it resonated with many of them, especially those with more extensive planning experience who were hoping for this combination of ideas and techniques. Similar responses were also received at Excel@FIT[3], from college students, doctors, and even representatives of sponsoring companies. To ensure the app's future success, it is important to address users' proposals and work on solutions to their issues. One of the key areas that the app must focus on is improving its usability, as some users found the interface overwhelming. It is also essential to consider whether the app will support sharing tasks across users, as Appwrite currently supports teams and database relationships. By addressing these issues, the app can gain more trust, which is crucial for its future existence.

[1] https://play.google.com/store/apps/details?id=com.zimolajan.timenoder
[2] https://apps.apple.com/us/app/timenoder/id1640957069?platform=iphone
[3] https://excel.fit.vutbr.cz/submissions/2023/051/51.pdf

Bibliography

[1] Abrahami, Y. Scaling to 100M: MySQL is a better NoSQL. *Wix Engineering (utolsó frissítés: 2018. 07. 17.) Web: https://www. wix. engineering/post/scaling-to-100m-mysql-is-a-better-nosql (2019. 11. 30.)*. 2016.

[2] Adetunji, O., Ajaegbu, C., Otuneme, N., Omotosho, O. J. et al. Dawning of Progressive Web Applications (PWA): Edging Out the Pitfalls of Traditional Mobile Development. *American Academic Scientific Research Journal for Engineering, Technology, and Sciences*. 2020, vol. 68, no. 1, p. 85–99.

[3] Adomavicius, G. and Tuzhilin, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*. IEEE. 2005, vol. 17, no. 6, p. 734–749.

[4] Allen, D. *Getting things done: The art of stress-free productivity*. Penguin, 2015.

[5] Alrasheed, S. H. and Alghamdi, W. M. Impact of an educational intervention using the 20/20/20 rule on Computer Vision Syndrome. *African Vision and Eye Health*. AOSIS. 2020, vol. 79, no. 1, p. 1–6.

[6] Angelov, F. *Architecture*. Accessed: 2022-11-29. Available at: https://bloclibrary.dev/#/architecture.

[7] Ariga, A. and Lleras, A. Brief and rare mental "breaks" keep you focused: Deactivation and reactivation of task goals preempt vigilance decrements. *Cognition*. Elsevier. 2011, vol. 118, no. 3, p. 439–443.

[8] Banbury, S. P. and Berry, D. C. Office noise and employee concentration: Identifying causes of disruption and potential improvements. *Ergonomics*. Taylor & Francis. 2005, vol. 48, no. 1, p. 25–37.

[9] Bell, R. M., Koren, Y. and Volinsky, C. The bellkor solution to the netflix prize. *KorBell team's report to Netflix*. 2007.

[10] Bilalić, M., McLeod, P. and Gobet, F. Inflexibility of experts—Reality or myth? Quantifying the Einstellung effect in chess masters. *Cognitive psychology*. Elsevier. 2008, vol. 56, no. 2, p. 73–102.

[11] Britton, B. K. and Tesser, A. Effects of time-management practices on college grades. *Journal of educational psychology*. American Psychological Association. 1991, vol. 83, no. 3, p. 405.

[12] Bryan, J. F. and Locke, E. A. Parkinson's law as a goal-setting phenomenon. *Organizational Behavior and Human Performance*. Elsevier. 1967, vol. 2, no. 3, p. 258–275.

[13] Buehler, R., Griffin, D. and Peetz, J. The planning fallacy: Cognitive, motivational, and social origins. In: *Advances in experimental social psychology*. Elsevier, 2010, vol. 43, p. 1–62.

[14] Bunkley, N. Joseph Juran, 103, pioneer in quality control, dies. *New York Times*. 2008, vol. 3, p. 50–55.

[15] Chi, M. T., Glaser, R. and Rees, E. *Expertise in problem solving*. Pittsburgh Univ PA Learning Research and Development Center, 1981.

[16] Chui, M., Manyika, J. and Bughin, J. *The social economy: Unlocking value and productivity through social technologies*. McKinsey Global Institute, 2012.

[17] Claessens, B. J., Van Eerde, W., Rutte, C. G. and Roe, R. A. A review of the time management literature. *Personnel review*. Emerald Group Publishing Limited. 2007.

[18] Clear, J. *Atomic habits: An easy & proven way to build good habits & break bad ones*. Penguin, 2018.

[19] Coles Brennan, C., Sulley, A. and Young, G. Management of digital eye strain. *Clinical and experimental Optometry*. Taylor & Francis. 2019, vol. 102, no. 1, p. 18–29.

[20] Colquitt, J., Lepine, J. and Wesson, M. *Organizational Behavior: Improving Performance and Commitment in the Workplace (4e)*. New York, NY, USA: McGraw-Hill, 2014.

[21] Connolly, T. and Begg, C. *A practical Approach to design, implementation, and management*. Addison-Wesley, Reading, 2005.

[22] Cottrell, S. *The study skills handbook*. Macmillan International Higher Education, 2013.

[23] Data.ai. *The State of Mobile in 2022: How to Succeed in a Mobile-First World As Consumers Spend 3.8 Trillion Hours on Mobile Devices*. Accessed: 2022-12-21. Available at: https://www.data.ai/en/insights/market-data/state-of-mobile-2022/.

[24] Demedyuk, I. and Tsybulskyi, N. *Flutter vs Native vs React-Native: Examining performance*. 2020. Accessed: 2022-12-21. Available at: https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980.

[25] Dunford, R., Su, Q. and Tamang, E. The pareto principle. University of Plymouth. 2014.

[26] Dunstan, D. W., Howard, B., Healy, G. N. and Owen, N. Too much sitting–a health hazard. *Diabetes research and clinical practice*. Elsevier. 2012, vol. 97, no. 3, p. 368–376.

[27] Eccles, J. S., Wigfield, A. et al. Motivational beliefs, values, and goals. *Annual review of psychology*. Palo Alto. 2002, vol. 53, no. 1, p. 109–132.

[28] Foundation, K. *Native and cross-platform app development: how to choose?* Accessed: 2022-12-21. Available at: https://kotlinlang.org/docs/native-and-cross-platform.html#debugging-some-popular-myths-about-cross-platform-app-development.

[29] Ge, Y., Xiong, H., Tuzhilin, A., Xiao, K., Gruteser, M. et al. An energy-efficient mobile recommender system. In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2010, p. 899–908.

[30] Gilder, T. *Flutter's Key Difference: Owning Every Pixel*. 2019. Accessed: 2022-12-21. Available at: https://medium.com/flutter-community/flutters-key-difference-owning-every-pixel-e2135b44c8a.

[31] Gomez Uribe, C. A. and Hunt, N. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*. ACM New York, NY, USA. 2015, vol. 6, no. 4, p. 1–19.

[32] Google. *Application Fundamentals*. Accessed: 2022-12-21. Available at: https://developer.android.com/guide/components/fundamentals.

[33] Google. *Compose Material 3*. Accessed: 2022-12-21. Available at: https://developer.android.com/jetpack/androidx/releases/compose-material3.

[34] Google. *Material Components widgets*. Accessed: 2022-11-29. Available at: https://docs.flutter.dev/development/ui/widgets/material.

[35] Google. *Material Design*. Accessed: 2022-11-29. Available at: https://m3.material.io/.

[36] Google. *Material Design Develop*. Accessed: 2022-12-21. Available at: https://m3.material.io/develop.

[37] Google. *TensorFlow Lite* [https://www.tensorflow.org/lite/guide]. Accessed: 2023-01-25.

[38] Google. *Writing custom platform-specific code*. Accessed: 2022-12-21. Available at: https://docs.flutter.dev/development/platform-integration/platform-channels.

[39] Gruber, H. E. and Bödeker, K. *Creativity, psychology and the history of science*. Springer, 2005.

[40] Gupta, P., Goel, A., Lin, J., Sharma, A., Wang, D. et al. Wtf: The who to follow service at twitter. In: *Proceedings of the 22nd international conference on World Wide Web*. 2013, p. 505–514.

[41] Hartson, R. and Pyla, P. S. *The UX Book: Process and guidelines for ensuring a quality user experience*. Elsevier, 2012.

[42] IBM. *IOS app development*. Accessed: 2022-12-21. Available at: https://www.ibm.com/cloud/learn/ios-app-development-explained.

[43] Iyengar, S. S. and Lepper, M. R. When choice is demotivating: Can one desire too much of a good thing? *Journal of personality and social psychology*. American Psychological Association. 2000, vol. 79, no. 6, p. 995.

[44] Jalote, P., Palit, A., Kurien, P. and Peethamber, V. Timeboxing: a process model for iterative software development. *Journal of Systems and Software*. Elsevier. 2004, vol. 70, 1-2, p. 117–127.

[45] Kerpelman, T. *Cloud Firestore Data Modeling (Google I/O'19)*. Accessed: 2022-11-26. Available at: https://www.youtube.com/watch?v=lW7DWV2jST0&t=578s&ab_channel=Firebase.

[46] Kirpalani, N. *What's the 1 Productivity Tool? For Me, It's Timeboxing*. [https://hbr.org/2021/09/whats-the-1-productivity-tool-for-me-its-timeboxing]. Accessed 16/08/2022.

[47] Kos, B. *Timeboxing: A simple and powerful technique to improve your productivity* [https://www.spica.com/blog/timeboxing]. Accessed 16/08/2022.

[48] Laila, S. N., Sabariah, M. K. and Suwawi, D. D. J. UI design of collaborative learning app for final assignment subject using goal-directed design. In: IEEE. *2016 4th International Conference on Information and Communication Technology (ICoICT)*. 2016, p. 1–6.

[49] Lakein, A. and Leake, P. *How to get control of your time and your life*. New American Library New York, 1973.

[50] Lee, S., Yang, J. and Park, S.-Y. Discovery of hidden similarity on collaborative filtering to overcome sparsity problem. In: Springer. *International Conference on Discovery Science*. 2004, p. 396–402.

[51] Lehtola, L., Kauppinen, M. and Kujala, S. Requirements prioritization challenges in practice. In: Springer. *International Conference on Product Focused Software Process Improvement*. 2004, p. 497–508.

[52] Leroy, S. Why is it so hard to do my work? The challenge of attention residue when switching between work tasks. *Organizational Behavior and Human Decision Processes*. Elsevier. 2009, vol. 109, no. 2, p. 168–181.

[53] Luzniak, K. and Hartuna, M. *All You Need to Know About PWA on iOS* [https://neoteric.eu/blog/all-you-need-to-know-about-pwa-on-ios/]. 2021. Accessed 20/12/2022.

[54] Macan, T. H., Shahani, C., Dipboye, R. L. and Phillips, A. P. College students' time management: Correlations with academic performance and stress. *Journal of educational psychology*. American Psychological Association. 1990, vol. 82, no. 4, p. 760.

[55] MacKenzie, I., Meyer, C. and Noble, S. How retailers can keep up with consumers. *McKinsey & Company*. 2013, vol. 18, no. 1.

[56] Madore, K. P. and Wagner, A. D. Multicosts of multitasking. In: Dana Foundation. *Cerebrum: the dana forum on brain science*. 2019, vol. 2019.

[57] Majchrzak, T. A., Biørn Hansen, A. and Grønli, T.-M. Progressive web apps: the definite approach to cross-platform development? 2018.

[58] Mark, G., Gudith, D. and Klocke, U. The cost of interrupted work: more speed and stress. In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 2008, p. 107–110.

[59] Masicampo, E. and Baumeister, R. F. Consider it done! Plan making can eliminate the cognitive effects of unfulfilled goals. *Journal of personality and social psychology*. American Psychological Association. 2011, vol. 101, no. 4, p. 667.

[60] Miranda, E. Time boxing planning: buffered moscow rules. *ACM SIGSOFT Software Engineering Notes*. ACM New York, NY, USA. 2011, vol. 36, no. 6, p. 1–5.

[61] Misra, R. and McKean, M. College students' academic stress and its relation to their anxiety, time management, and leisure satisfaction. *American journal of Health studies*. Expert Health Data System, Inc. 2000, vol. 16, no. 1, p. 41.

[62] MUI. *Move faster with intuitive React UI tools*. Accessed: 2022-11-29. Available at: https://mui.com/.

[63] Newport, C. *Deep work: Rules for focused success in a distracted world*. Hachette UK, 2016.

[64] Noteberg, S. *Pomodoro technique illustrated: The easy way to do more in less time*. Pragmatic Bookshelf, 2009.

[65] Oakley, B. A. *A mind for numbers: How to excel at math and science (even if you flunked algebra)*. TarcherPerigee, 2014.

[66] Oracle. *A Relational Database Overview*. Accessed 20/1/2023. Available at: https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html.

[67] Overflow, S. *2020 Deleveloper Survey*. Accessed: 2022-12-21. Available at: https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages-professional-developers.

[68] Ovesen, N., Eriksen, K., Tollestrup, C. et al. Speeding up development activities in student projects with time boxing and scrum. In: *DS 69: Proceedings of E&PDE 2011, the 13th International Conference on Engineering and Product Design Education, London, UK, 08.-09.09. 2011*. 2011, p. 559–564.

[69] Parkinson, C. N. and Lancaster, O. Parkinson's Law: or the Pursuit of Progress. Murray London. 1958.

[70] Pinedamg. *How Flutter & Dart Code Gets Compiled To Native Apps* [https://pinedamg.medium.com/how-flutter-dart-code-gets-compiled-to-native-apps-c4612ea0ef0e]. Accessed: 2022-11-26.

[71] Research, F. A. *PyTorch Mobile* [https://pytorch.org/mobile/home/]. Accessed: 2023-01-25.

[72] Ricci, F., Rokach, L. and Shapira, B. Recommender Systems: Techniques, Applications, and Challenges. *Recommender Systems Handbook*. Springer. 2022, p. 1–35.

[73] Rosen, L. D., Carrier, L. M. and Cheever, N. A. Facebook and texting made me do it: Media-induced task-switching while studying. *Computers in Human Behavior*. Elsevier. 2013, vol. 29, no. 3, p. 948–958.

[74] Schwartz, B. The paradox of choice: Why more is less. *New York*. 2004.

[75] Sheppard, A. L. and Wolffsohn, J. S. Digital eye strain: prevalence, measurement and amelioration. *BMJ open ophthalmology*. BMJ Specialist Journals. 2018, vol. 3, no. 1, p. e000146.

[76] Soboroff, I. and Nicholas, C. Combining content and collaboration in text filtering. In: Sn. *Proceedings of the IJCAI*. 1999, vol. 99, no. 1999, p. 86–91.

[77] TenserFlow. *Content-based filtering & collaborative filtering (Building recommendation systems with TensorFlow)* [https://youtu.be/v90un9ALRzw]. 2021. Accessed 20/12/2022.

[78] Tracy, B. *Eat that frog!: 21 great ways to stop procrastinating and get more done in less time*. Berrett-Koehler Publishers, 2017.

[79] Usman, S. A. *Using the Pomodoro Technique® to help undergraduate students better manage technology-based multitasking during independent study: A design-based research investigation*. Lancaster University (United Kingdom), 2020.

[80] Vischer, J. C. The effects of the physical environment on job performance: towards a theoretical model of workspace stress. *Stress and health: Journal of the International Society for the Investigation of Stress*. Wiley Online Library. 2007, vol. 23, no. 3, p. 175–184.

[81] Wilmot, E. G., Edwardson, C. L., Achana, F. A., Davies, M. J., Gorely, T. et al. Sedentary time in adults and the association with diabetes, cardiovascular disease and death: systematic review and meta-analysis. *Diabetologia*. Springer. 2012, vol. 55, no. 11, p. 2895–2905.

[82] Yakovenko, V. M. and Silva, A. C. TWo-class Structure of Income Distribution in. *Econophysics of Wealth Distributions: Econophys-Kolkata I*. Springer Science & Business Media. 2005, p. 15.

[83] Zao Sanders, M. *How Timeboxing Works and Why It Will Make You More Productive*. 2018. Accessed 22/1/2023. Available at: https://hbr.org/2018/12/how-timeboxing-works-and-why-it-will-make-you-more-productive.

Appendix A

Contents of The Attached Memory Media