UNIVERSITATEA POLITEHNICA BUCUREȘTI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

|  |  |
| --- | --- |
| upb | cs |

PROIECT DE DIPLOMĂ

NavTask

Instrument de gestionare a sarcinilor și activităților în mediul Android

Ionuț-Vlăduț Pasat

**Coordonator științific:**

Șl. Dr. Ing. Carmen Odubășteanu

BUCUREŞTI

2024

UNIVERSITY POLITEHNICA OF BUCHAREST

FACULTY OF AUTOMATIC CONTROL AND COMPUTERS

COMPUTER SCIENCE DEPARTMENT

|  |  |
| --- | --- |
| upb | cs |

DIPLOMA PROJECT

NavTask

Task and activity management tool for the Android environment

Ionuț-Vlăduț Pasat

**Thesis advisor:**

Sl. Dr. Eng. Carmen Odubășteanu

BUCHAREST

2024

**Table of Contents**

[Sinopsis 6](#_Toc169477108)

[Abstract 6](#_Toc169477109)

[Acknowledgements 7](#_Toc169477110)

[1 Introduction 8](#_Toc169477111)

[1.1 Context 8](#_Toc169477112)

[1.2 Problem Statement 8](#_Toc169477113)

[1.3 Study Objectives 9](#_Toc169477114)

[1.4 Paper structure 10](#_Toc169477115)

[2 Requirements analysis and specification 11](#_Toc169477116)

[2.1 Introduction 11](#_Toc169477117)

[2.2 Motivation 11](#_Toc169477118)

[2.3 Methodology 11](#_Toc169477119)

[2.4 Survey Results and Insights 12](#_Toc169477120)

[2.5 List of Functionalities 18](#_Toc169477121)

[2.5.1 Task Management: 18](#_Toc169477122)

[2.5.2 Reminders and Notifications: 18](#_Toc169477123)

[2.5.3 Location-Based Services: 18](#_Toc169477124)

[2.5.4 Weather Integration: 18](#_Toc169477125)

[2.5.5 Task Prioritization and Sorting: 18](#_Toc169477126)

[2.5.6 User Interface and Experience: 18](#_Toc169477127)

[3 Market study/existing approaches 19](#_Toc169477128)

[3.1 Existing Solutions 19](#_Toc169477129)

[3.1.1 Todoist 19](#_Toc169477130)

[3.1.2 Microsoft To Do 19](#_Toc169477131)

[3.1.3 Google Keep 19](#_Toc169477132)

[3.2 Limitations of Existing Solutions 20](#_Toc169477133)

[3.3 Common and Unique Features of NavTask 20](#_Toc169477134)

[3.4 Comparative Analysis 21](#_Toc169477135)

[3.5 Opportunities and Areas for Improvement 21](#_Toc169477136)

[4 Proposed Solution 22](#_Toc169477137)

[4.1 Description of the Solution 22](#_Toc169477138)

[4.1.1 User Interface (UI) Layer 22](#_Toc169477139)

[4.1.2 Business Logic Layer 28](#_Toc169477140)

[4.1.3 Data Access Layer 28](#_Toc169477141)

[4.1.4 Integration Layer 30](#_Toc169477142)

[4.2 Process of Solution Development 31](#_Toc169477143)

[5 Implementation Details 33](#_Toc169477144)

[5.1 Firebase Integration 33](#_Toc169477145)

[5.1.1 Firebase Console Setup: 33](#_Toc169477146)

[5.1.2 Gradle Dependencies: 33](#_Toc169477147)

[5.1.3 Firebase Authentication: 33](#_Toc169477148)

[5.2 Notification Handling 34](#_Toc169477149)

[5.2.1 Gradle Dependencies: 34](#_Toc169477150)

[5.2.2 Manifest Configuration: 35](#_Toc169477151)

[5.2.3 Notification Setup: 35](#_Toc169477152)

[5.2.4 Notification Handler: 36](#_Toc169477153)

[5.2.5 Broadcast Receiver: 36](#_Toc169477154)

[5.3 Database Setup 37](#_Toc169477155)

[5.3.1 Gradle Dependencies: 37](#_Toc169477156)

[5.3.2 Database Configuration: 37](#_Toc169477157)

[5.3.3 Database Entities: 37](#_Toc169477158)

[5.3.4 Database Setup: 38](#_Toc169477159)

[5.4 Maps and Weather API Integration 39](#_Toc169477160)

[5.4.1 Google Maps Integration: 39](#_Toc169477161)

[5.4.2 Weather API Integration: 39](#_Toc169477162)

[5.5 Bottom Navigation Bar 40](#_Toc169477163)

[5.5.1 Navigation Handling: 41](#_Toc169477164)

[5.6 Image Picker 41](#_Toc169477165)

[5.6.1 Image Picker Launcher: 42](#_Toc169477166)

[5.6.2 Image Selection and Handling: 42](#_Toc169477167)

[5.7 Polyline Rendering 42](#_Toc169477168)

[6 Results Evaluation 44](#_Toc169477169)

[6.1 Correctness Evaluation 44](#_Toc169477170)

[6.1.1 User Authentication 44](#_Toc169477171)

[6.1.2 Task Management 44](#_Toc169477172)

[6.1.3 Notifications 44](#_Toc169477173)

[6.1.4 Location and Weather Integration 45](#_Toc169477174)

[6.1.5 Profile Management 45](#_Toc169477175)

[6.2 Performance Evaluation 45](#_Toc169477176)

[6.2.1 User Authentication Speed 46](#_Toc169477177)

[6.2.2 Task Management Efficiency 46](#_Toc169477178)

[6.2.3 Notification Reliability 46](#_Toc169477179)

[6.2.4 API Integration Performance 46](#_Toc169477180)

[6.2.5 Resource Consumption 46](#_Toc169477181)

[6.3 Criteria for Satisfactory Rating 46](#_Toc169477182)

[6.3.1 Unit Tests and Integration Tests 46](#_Toc169477183)

[6.3.2 Deployment Strategy 47](#_Toc169477184)

[6.3.3 Client Validation 48](#_Toc169477185)

[6.4 Conclusion 49](#_Toc169477186)

[7 Concluzii 51](#_Toc169477187)

[8 Bibliografie 52](#_Toc169477188)

[9 Anexe 55](#_Toc169477189)

# Sinopsis

Managementul timpului implică planificarea și supervizarea eficientă a alocării orelor zilnice pentru a atinge obiectivele dorite. Provocările apar deseori în urma subestimării duratei sarcinilor, chiar și atunci când acestea au mai fost finalizate anterior, și din faptul că persoanele nu respectă planurile lor de gestionare a timpului. Acest proiect își propune să abordeze aceste probleme prin dezvoltarea unei aplicații. Aplicația va servi ca un instrument de amintire și mapare, conținând diverse informații și utilitare destinate a facilita procesul de împlinire al scopurilor, dar și posibilitatea de personalizare care face aplicația să aibă o interfață mai prietenoasă. Prin materializarea și vizualizarea acestor obiective, utilizatorii pot identifica eventuale priorități și pot face ajustări necesare în programul lor pe baza datelor observate.

Cuvinte cheie: managementul timpului, motivație, aplicație, prioritate, planificare.

# Abstract

Time management involves effectively planning and overseeing the allocation of one's daily hours to achieve desired objectives. Challenges often arise from underestimating task durations, even when they've been completed previously, and from individuals not adhering to their time management plans. This project endeavors to address these issues through the development of an application. The application will serve as a reminder and mapping tool, aligning a diversity of information and tools gathered to facilitate the process of fulfilling one’s goals. By materialising and visualizing these objectives, users can identify certain priorities and make necessary adjustments to their schedules based on the observed data.

Keywords: time management, motivation, application, reminders, scheduling.

# Acknowledgements

I would like to express my deepest gratitude to my parents, whose unwavering support and provision enabled me to pursue my studies at this esteemed university. Their belief in me has been a constant source of motivation throughout my academic journey. I am also profoundly thankful to my thesis advisor for their invaluable guidance and insightful advice, which have been instrumental in shaping this thesis. Additionally, I extend my heartfelt thanks to all my teachers and colleagues, whose continuous encouragement and assistance have helped me grow and improve each day. Your contributions have been vital to my success, and I am truly grateful for your support.

# Introduction

## Context

Time management is a critical skill that involves planning and exercising conscious control over the amount of time spent on specific activities, particularly to increase effectiveness, efficiency, and productivity. It stretches over a vast range of practices, tools, and techniques used to manage time when accomplishing specific tasks, projects, and goals. Effective time management allows individuals to prioritize tasks, set achievable goals, and reduce stress, leading to a more balanced and productive life. By organizing and planning how to divide time between various activities, one can work smarter, not harder, ensuring that more tasks are completed in less time, even when time is tight, and pressures are high.[[1]](#footnote-1)

## Problem Statement

The need for time management is paramount in today's fast-paced world, where balancing work, education, and personal life can be challenging. Without proper time management, individuals may struggle with procrastination, miss deadlines, and experience increased stress and burnout. Effective time management helps to improve focus, enhance decision-making skills, and achieve a sense of accomplishment. As Stephen R. Covey eloquently puts it in his book, "The 7 Habits of Highly Effective People," "The key is not to prioritize what’s on your schedule, but to schedule your priorities." This highlights the importance of aligning one's tasks and goals with their core values and priorities, ensuring that the most important tasks are addressed first. [1]

In the realm of academic and professional development, time management is essential. As emphasized by David Allen in his book, "Getting Things Done: The Art of Stress-Free Productivity," a systematic approach to managing tasks can lead to increased efficiency and reduced stress. Allen's methodology underscores the significance of capturing all tasks and commitments in a trusted system and processing them in a structured manner. This approach not only helps in managing daily tasks but also in setting and achieving long-term goals. [2]

The project addresses the common challenge of efficiently managing daily tasks and schedules. Many individuals struggle with organizing their responsibilities and ensuring they are completed on time. The difficulty in keeping track of what needs to be done, where it needs to be done, and under what conditions can lead to disorganization and missed deadlines. By providing a platform to log tasks, along with contextual information such as weather conditions and travel directions, the application helps users better prepare and manage their daily activities.

## Study Objectives

The primary objective of this project is to develop an Android application that enhances users’ time management capabilities by offering a user friendly platform for logging tasks, scheduling them effectively, and providing contextual utilities such as weather updates and navigation assistance. By addressing common time management challenges, this solution aims to facilitate better daily planning and execution of tasks, leading to improved productivity and reduced stress for users.

Specifically, the objectives are to:

* *Provide an Intuitive Task Logging and Scheduling Tool*: The application allows users to log tasks for specific days, helping them organize and plan their activities efficiently. This feature aims to make task management straightforward and accessible, enabling users to keep track of their responsibilities and deadlines.
* *Offer Contextual Information*: By integrating weather updates for task locations and providing directions from the user’s current location to the task site, the app adds practical value to the task management process. These utilities help users prepare better for their tasks, considering environmental conditions and travel requirements, thus reducing the uncertainty and inconvenience associated with daily planning.
* *Enhance User Experience with a User-Friendly Interface*: The application is designed with a focus on ease of use, ensuring that users can quickly navigate and utilize the app's features without a steep learning curve. This objective aims to increase user adoption and satisfaction by providing a seamless and enjoyable experience.
* *Promote Effective Time Management Practices*: By offering tools that help users log, schedule, and execute tasks efficiently, the app encourages the adoption of effective time management practices. This can lead to improved organization, better prioritization of tasks, and more efficient use of time, which are crucial for both personal and professional growth.

The project’s resolution is expected to lead to several significant developments:

* *Increased Productivity*: Users will benefit from a more organized approach to managing their daily tasks, leading to increased efficiency and productivity.
* *Improved Task Preparation*: With weather updates and location-based directions, users will be better prepared for their tasks, reducing last-minute disruptions and enhancing task execution.
* *Enhanced Planning and Execution*: The integration of task scheduling with contextual information will provide users with a comprehensive tool for planning their day, ensuring that all relevant factors are considered in their task management process.
* *Greater User Engagement*: The app's user-friendly interface and valuable features are anticipated to drive higher user engagement and satisfaction, leading to broader adoption and a loyal user base.

Ultimately, this project aims to deliver a robust solution for time management that not only addresses existing challenges but also provides users with the tools and insights needed to achieve their goals more effectively.

## Paper structure

This paper is organized into several sections to systematically present the project and its outcomes.

* *Introduction*: Discusses the importance of time management and provides an overview of the project’s relevance and purpose.
* *Requirements Analysis and Specification*: Defines the specific needs the application addresses and the functionalities it must include.
* *Market Study / Existing Approaches*: Examines current solutions and tools for task management and how they compare to the proposed application.
* *Proposed Solution*: Details the design and main features of the application, including task logging, weather forecasts, and navigation assistance.
* *Implementation Details*: Describes the technical implementation, including the development process and the technologies used.
* *Case Study / Results Evaluation*: Evaluates the application's performance through user testing and feedback, assessing its effectiveness in real-world scenarios.
* *Conclusions*: Summarizes the key findings, discusses the impact of the application, and suggests potential areas for future development.
* *Bibliography*: Lists the sources and references that informed the research and development of the application.
* *Appendices*: Provides additional documentation, screenshots, and supporting material relevant to the project.

# Requirements analysis and specification

## Introduction

This section examines the product's required features from the viewpoint of prospective users and anticipated usage situations. The objective is to compile a list of functionalities essential for developing a product that satisfies the needs and expectations of users. The analysis draws upon data gathered through a Google Form survey, pinpointing current challenges and desired functionalities in a time management application.

## Motivation

The primary motivation for developing a time management application with a user-friendly interface integrated with weather and mapping services stems from the survey results, which highlight several challenges faced by users in managing their tasks effectively. Many respondents reported issues such as forgetting tasks or deadlines, poor task prioritization, lack of reminders or notifications, and difficulty accessing information on the go. These pain points indicate a need for a comprehensive solution that not only helps users organize and prioritize their tasks but also provides relevant context and information to facilitate task completion.

Integrating weather and location-based services can significantly enhance the user experience by providing real-time weather updates and directions for task locations. This added functionality can help users better plan their activities and make informed decisions based on environmental conditions and travel times, ultimately improving their time management and productivity.

Furthermore, the survey results reveal a strong interest among users for such integrated features, with many respondents expressing a positive or very positive attitude towards incorporating weather and mapping services into their time management applications.

## Methodology

To gather user requirements and insights, I conducted a survey using a Google Form. The survey consisted of multiple-choice and open-ended questions designed to capture participants' demographic information, their current time management practices, challenges faced, and desired features in a time management application.

The survey reached a targeted audience of individuals aged 20-24 years, representing the primary user group for the proposed application. The responses were collected over a period of time and compiled for analysis.

## Survey Results and Insights

1. *Importance of time management*:

According to the chart in Figure 2.5.1‑1 below, the majority of respondents, 51.7%, consider time management to be "Very important" in their day-to-day activities. This suggests that effective time management is a crucial aspect for a large portion of the surveyed individuals, potentially due to various factors such as busy schedules, multiple responsibilities, or a desire to optimize productivity.

The second-largest group, totaling 37.9% of respondents, perceives time management as "Important." While not as critical as the "Very important" group, this segment still recognizes the value of proper time management practices in their daily

routines.

A smaller portion of the respondents, 10.4%, either have a neutral stance on the significance of time management or do not consider it an essential aspect of their daily lives.

Overall, the chart highlights that an overwhelming majority (89.6%) of the surveyed individuals consider time management to be important or very important in their day-to-day lives. This finding underscores the need for effective time management solutions and justifies the development of a time management application that can cater to the needs of users who value efficient organization and utilization of their time.

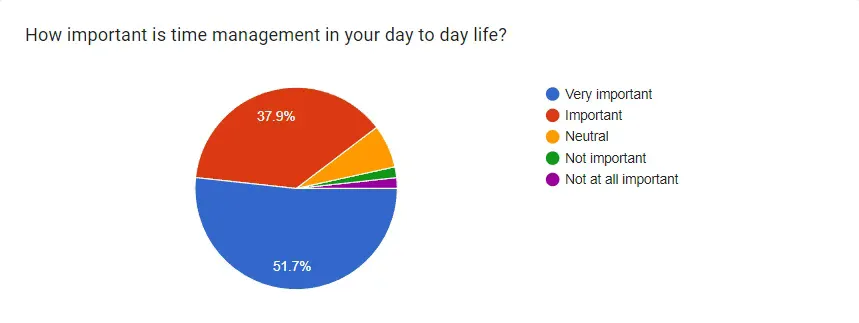


Figure 2.5.1‑1 Time Management Importance

1. *Current used time management methods*:

According to the chart in Figure 2.5.1‑2, the largest segment, comprising 37.9% of respondents, relies on "Mental Tracking," which involves remembering tasks without the use of formal tools or applications. This approach may be convenient for individuals with a strong memory or those who prefer a more informal method of time management.

Additionally, 24.1% of respondents indicated using "Digital Tools" exclusively, such as apps, software, or online calendars. This group likely values the convenience, accessibility, and organizational capabilities provided by digital solutions for time management.

This consolidated view highlights the substantial demand and preference for digital solutions among the target audience. It reinforces the importance of developing a time management application that can effectively cater to this sizable segment of users who rely on digital platforms for organizing their tasks and schedules.

Factoring the people that said they are using mental tracking as their primary method of monitoring day to day tasks and their level of satisfaction extracted from the form[[2]](#footnote-2)

we can pronounce that mental tracking, while convenient for some, can become increasingly challenging as the number of tasks, deadlines, and commitments increases. It places a significant burden on an individual's memory and can lead to forgetting important tasks or deadlines, as evidenced by the survey results where "Forgetting tasks or deadlines" was cited as a significant challenge. [3]

This data underscores the importance of developing a time management application that caters to a wide range of user preferences and potential integration with existing tools or methodologies. By offering a versatile solution that can seamlessly incorporate various time management practices, the proposed application can effectively address the needs of a broader user base.

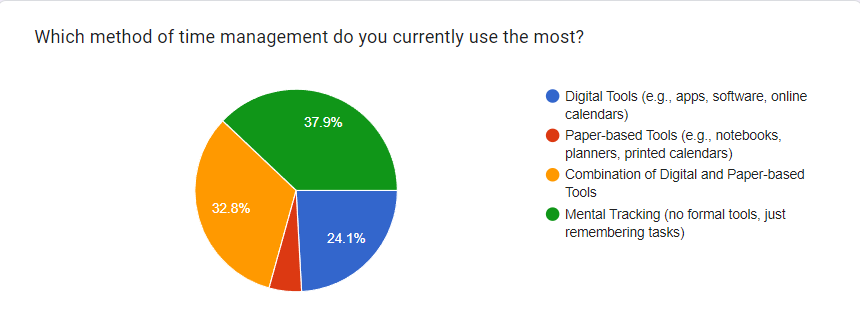


Figure 2.5.1‑2 Time Management Methods

1. *Time management app usage*:

The pie chart in Figure 2.5.1‑3 represents the frequency at which survey respondents use time management or to-do list apps.

The largest segment, comprising 36.2% of respondents, indicates using time management or to-do list apps on a weekly basis. This suggests that a significant portion of the surveyed group has incorporated these tools into their regular routines, likely finding them helpful for organizing and managing tasks on a recurring weekly cycle.

The second-largest group, accounting for 27.6% of respondents, reports rarely using such apps. While they may have experimented with or explored time management applications, their infrequent usage could indicate dissatisfaction with existing solutions or a preference for alternative methods of task management.

Another sizable segment, 20.7% of respondents, use time management or to-do list apps daily. This group likely finds these applications indispensable for their day-to-day task management needs, relying heavily on the organization and reminders provided by the apps.

The remaining segments are relatively small, with 5.2% of respondents using apps monthly, and 3.4% using them weekly (different from the larger "weekly" segment).

Overall, the data suggests a diverse range of usage patterns, with a significant portion (56.9%) of respondents being regular users (daily or weekly) of time management or to-do list apps. However, there is also a considerable segment (27.6%) that rarely uses such applications, indicating potential room for improvement or a need for more compelling solutions.

This information can guide the development and marketing strategies for the proposed time management application. By addressing the pain points of infrequent users and offering a user-friendly, feature-rich solution, the application may attract a broader user base, including those who currently rely on alternative methods or have been dissatisfied with existing options.

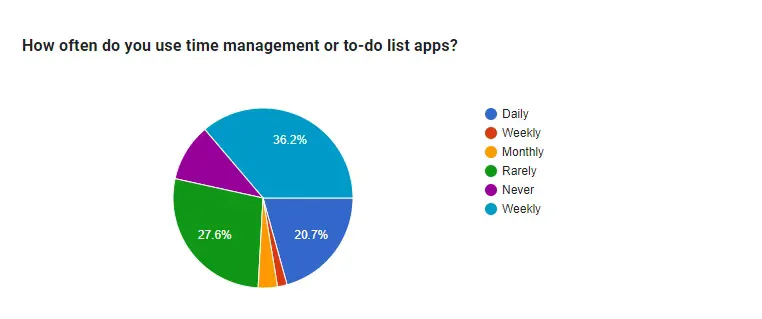


Figure 2.5.1‑3 Time Management App Usage

1. *Desired features*:

The bar chart ilustrated in Figure 2.5.1‑4 presents the features that survey respondents find most useful in a time management app. By analyzing the data, we can gain insights into the essential functionalities users expect from such an application.

The top most desired features are:

* Reminders and notifications (50 responses, 86.2%): A vast majority of respondents value the ability to receive reminders and notifications for upcoming tasks, deadlines, or events. This feature helps users stay organized and ensures they don't miss important commitments.
* Task scheduling (49 responses, 84.5%): The capability to schedule and organize tasks is considered highly useful by a significant portion of respondents. This functionality allows users to plan and prioritize their activities effectively.

Other notable features include:

* Location-based services (e.g., directions) (11 responses, 19%): While not as highly demanded as the top three features, a considerable number of respondents find location-based services, such as directions to task locations, useful in a time management app.
* Weather updates for task locations (13 responses, 22.4%): Receiving weather updates for task locations is seen as a valuable feature by a smaller but notable segment of respondents. This information can assist in planning activities and adjusting schedules based on weather conditions.

This data aligns with the proposed development of a time management application that incorporates features like task scheduling, reminders, location-based services, and weather updates. By prioritizing the implementation of these highly desired functionalities, the application can effectively cater to the needs and preferences of the target user base, potentially leading to higher adoption and satisfaction rates.

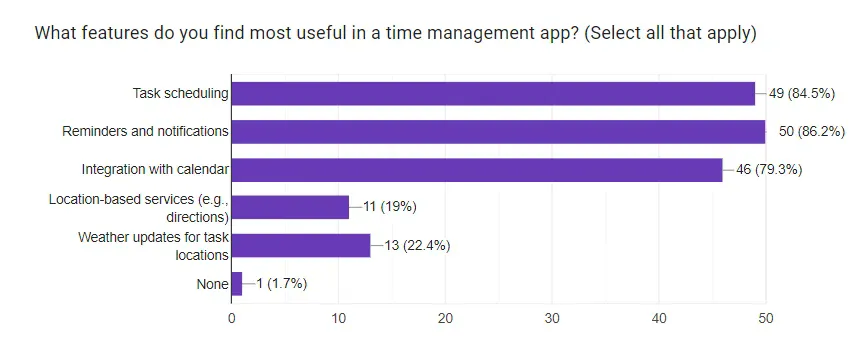


Figure 2.5.1‑4 Useful App Features

1. *Task types*:

The bar chart in Figure 2.5.1‑5 illustrates the types of tasks that survey respondents most frequently manage using an app or would like to manage if they started using one. By analyzing the data, we can understand the diverse task categories that potential users seek to organize and prioritize through a time management application.

The most prevalent task types are:

Academic assignments: A significant majority of respondents indicate a need to manage academic assignments using a time management app. This highlights the application's potential usefulness for students juggling various coursework and deadlines.

Work-related tasks: Managing work-related tasks is another critical requirement for many respondents, suggesting that the app could cater to professionals or those with employment responsibilities.

Personal errands: A substantial portion of respondents express a desire to manage personal errands or tasks using the app, showcasing its potential for organizing day-to-day activities beyond work or academic commitments.

Social activities: While not as prevalent as the other categories, a considerable number of respondents indicate an interest in managing social activities through the time management app.

This data underscores the need for the proposed time management application to cater to a diverse range of task types, including academic assignments, work-related tasks, personal errands, and social activities. By offering a comprehensive and versatile solution, the application can effectively address the varying needs of its potential user base, encompassing students, professionals, and individuals with diverse responsibilities.

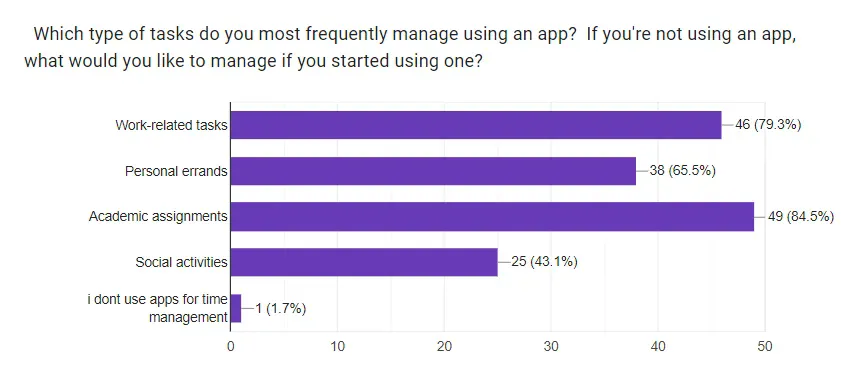


Figure 2.5.1‑5 Most Common Task Types Managed Using an App

1. *Task prioritization*:

The chart illustrated in Figure 2.5.1‑6 displays how people typically prioritize their tasks based on different factors. According to the data, the largest portion (50%) prioritizes tasks by their urgency or deadlines. This suggests that a significant number of people consider time constraints and due dates as the primary factor when deciding which tasks to tackle first.

The second-largest segment (29.3%) prioritizes tasks based on importance or impact on goals. This indicates that many people also consider the significance and potential consequences of tasks when determining their priority.

Additionally, 12.1% of respondents prioritize tasks based on convenience or ease of completion, while 8.6% consider the time available for the task.

The data clearly shows that urgency, importance, and time constraints are the top factors people consider when prioritizing tasks. This supports the development of a time management app with features like task scheduling, reminders, and integration with weather and maps APIs. Such an app could help users effectively prioritize tasks based on deadlines, importance, and time availability, while also providing relevant location and weather information to aid in task planning and execution.

By catering to the evident need for task prioritization based on urgency, importance, and time constraints, and offering additional context like weather and directions, the proposed app could potentially enhance user productivity and task management capabilities.

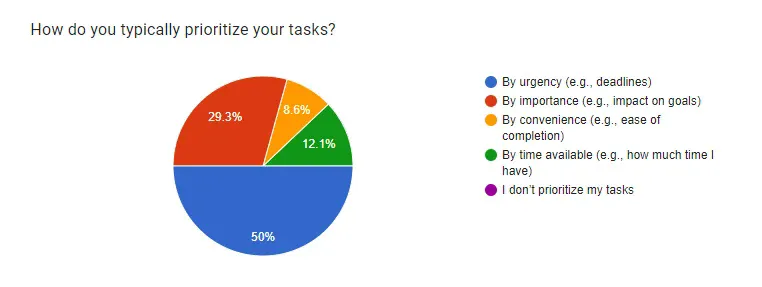


Figure 2.5.1‑6 Task Prioritization

## List of Functionalities

Effective time management is crucial for individuals juggling multiple responsibilities and commitments in their daily lives. The proposed time management application aims to provide a comprehensive solution that not only streamlines task organization and scheduling but also incorporates context-aware features to enhance the overall user experience. By integrating location-based services, weather updates, and calendar synchronization, the application seeks to address the diverse needs and expectations of students, professionals, and individuals managing personal tasks, travel plans, or event coordination.

Based on the survey results and the identified use cases, the following functionalities are deemed essential for the proposed time management application:

### Task Management:

* Creation, editing, and deletion of tasks or to-do items
* Assignment of due dates and times for tasks
* Prioritization or importance levels for tasks (e.g., high, medium, low)
* Ability to add task notes or descriptions for additional details

### Reminders and Notifications:

* Setting of reminders for tasks and events
* Push notifications for upcoming tasks and reminders

### Location-Based Services:

* Integration with mapping services (e.g., Google Maps)
* Retrieval of directions and travel times to task locations
* Automatic detection of user's current location

### Weather Integration:

* Provision of weather updates for specific task locations

### Task Prioritization and Sorting:

* Sorting of tasks based on due dates, priorities, or categories

### User Interface and Experience:

* Clean and intuitive user interface

This comprehensive list of functionalities addresses the diverse requirements and preferences expressed by the survey respondents, catering to the needs of students, professionals, and individuals managing personal tasks, travel plans, or event coordination. The integration of location-based services, weather updates, and calendar synchronization aligns with the proposed vision of a time management application that provides context aware information and seamless integration with external services, enhancing the overall user experience and facilitating effective time management across various domains.

# Market study/existing approaches

This section provides a comprehensive analysis of similar time management applications available on the market. The goal is to identify their limitations, the types of users they serve, and the gaps these products leave unaddressed. Additionally, this section will highlight the unique features of our application, how it stands out from the competition, and any areas where further improvements are needed.

## Existing Solutions

Several time management and to-do list applications dominate the market. Here, we will compare three popular apps: Todoist, Microsoft To Do, and Google Keep. Each of these apps has distinct features and caters to different user needs.

### Todoist*[[3]](#footnote-3)*

Features: Task creation and organization, project management, labels, filters, reminders, integrations with other apps.

Target Users: Professionals, project managers, individuals with advanced task management needs.

User Ratings: Generally high, praised for its powerful features and integrations, but noted for its steep learning curve.

### Microsoft To Do*[[4]](#footnote-4)*

Features: Task creation, due dates, reminders, file attachments, integration with Microsoft Office.

Target Users: Microsoft Office users, individuals seeking basic task management.

User Ratings: Positive, especially among Microsoft users, but criticized for lacking advanced task management features.

### Google Keep*[[5]](#footnote-5)*

Features: Note-taking, checklists, reminders, collaboration, multimedia attachments, integration with Google Workspace.

Target Users: Casual users, individuals who need a simple and quick note-taking solution.

User Ratings: Generally favorable, especially for its simplicity and ease of use, but noted for its limited functionality in task management.

## Limitations of Existing Solutions

The primary limitations of the above mentioned applications are as follows:

*Todoist*: While powerful, it can be overwhelming for new users. The premium subscription is required for many advanced features, which may be a barrier for some users. Additionally, it does not offer integrated weather updates or detailed location-based services.[[6]](#footnote-6)

*Microsoft To Do*: Although it integrates well with the Microsoft ecosystem, it lacks the advanced features found in other apps. Users outside the Microsoft ecosystem may find it less appealing. It also does not provide comprehensive location-based services or weather updates.[[7]](#footnote-7)

*Google Keep*: This app is excellent for quick note-taking but falls short in managing complex tasks and projects. It lacks advanced task management features, detailed location-based services, and integrated weather updates.[[8]](#footnote-8)

## Common and Unique Features of NavTask

NavTask, designed to improve time management, includes several common and unique features that address the limitations of existing solutions:

Task Logging and Scheduling: Users can easily add tasks and schedule them for specific days, ensuring organized task management.

Notifications and Reminders: The app provides customizable notifications and reminders for each task, helping users stay on track with their schedules.

Weather Updates: Integrated real-time weather updates for the locations of tasks, aiding users in planning their activities better.

Directions to Task Locations: The app offers directions from the user's current location to the task location, enhancing efficiency in travel planning.

User-Friendly and Intuitive Interface: Designed to be accessible to a broad range of users, the app features a clean layout and intuitive navigation.

Integration with Other Services: Seamless integration with maps and weather services providing a comprehensive time management solution.

Task Prioritization Tools: Users can prioritize tasks based on importance or urgency, improving decision-making and productivity.

## Comparative Analysis

The following table summarizes the key features and limitations of the existing solutions compared to NavTask:

Table 1 – Comparative Analysis Between Different Todo Apps and NavTask

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | Todoist | Microsoft To Do | Google Keep | NavTask |
| Task Logging and Scheduling | Yes | Yes | Basic | Yes |
| Notifications and Reminders | Yes (premium) | Yes | Basic | Yes |
| Weather Updates | No | No | No | Yes |
| Directions to Task Locations | No | No | No | Yes |
| User-Friendly Interface | Moderate | High | High | High |
| Integration with Other Services | Yes (limited) | Yes (Microsoft) | Yes (Google) | Yes |
| Task Prioritization Tools | Yes (advanced) | Basic | No | Yes |

## Opportunities and Areas for Improvement

While NavTask offers several unique features, there are opportunities for further enhancements:

* Enhanced User Customization: Allowing users to customize the interface and functionality based on their preferences could improve user satisfaction.
* Advanced Analytics: Providing users with insights and analytics on their task completion patterns and productivity trends.
* Integration with More Services: Expanding integration to include more third-party services such as calendar, fitness apps, email clients, and smart home devices.
* Collaboration Features: Adding features that support task sharing and collaboration for teams and groups.
* Gamification: Implementing gamification elements to motivate users and make time management more engaging.

By analyzing the existing solutions and their limitations, it is evident that our application addresses significant gaps in the market. The integration of weather updates, directions to task locations, and an intuitive interface, combined with comprehensive task management features, positions our application as a robust and valuable tool for users.

Continuous feedback and iterative development will ensure that the app evolves to meet the changing needs of its users, providing an effective solution for time management challenges.

# Proposed Solution

The proposed solution aims to improve time management and task scheduling through an Android application that provides a comprehensive and user-friendly interface. This chapter details the architecture and key components of the solution, including the user interface, business logic, data access, and integration layers. It incorporates diagrams to illustrate the structure and workflow of the application, demonstrating how it effectively addresses user needs and enhances task management.

## Description of the Solution

The application is designed to streamline the task management process by integrating user authentication, task scheduling, search functionalities, and navigation features. Below is a detailed description of each component and functionality within the application.

### User Interface (UI) Layer

The UI layer provides an intuitive and interactive interface for users. It consists of several key screens and components:

#### Login Screen

The Login screen, presented in Figure 4.1.1‑2, is the entry point for authenticated users into the application. It implements secure authentication protocols integrated using the authentication process of Firebase to validate user credentials against the backend server. The screen contains input fields for the user's email address and password, both of which are subjected to client-side validation to ensure proper formatting before submission.

To streamline the login process, the screen integrates with Google Sign-In API, allowing users to authenticate with their Google accounts seamlessly. This integration leverages OAuth 2.0 for secure authorization and token exchange, minimizing the need for users to remember separate credentials.

For new users, the screen prominently displays a "Don't have an account? Sign up" button, which navigates to the Signup screen.

#### Signup Screen

The Signup screen, illustrated by Figure 4.1.1‑1, facilitates the creation of new user accounts within the application. It presents a form with input fields for capturing the user's first name, last name, email address, password, and password confirmation. Each input field is equipped with client-side validation to ensure data integrity and adherence to predefined rules, such as non-empty fields, email format validation, and password matching.

Upon successful form submission, the user's input data is securely transmitted to the backend server for further processing and storage. The server-side validation includes additional checks, such as verifying the uniqueness of the email address and ensuring the password meets the required complexity criteria.

Once the account creation process is complete, a success message is displayed to the user, along with a button that redirects them to the Login screen.

Figure 4.1.1‑1 Signup Screen

Figure 4.1.1‑2 Login Screen

A screenshot of a computer screen

Description automatically generatedA screenshot of a login form

Description automatically generated

#### Home Screen

The Home screen, as seen in Figure 4.1.1‑4, serves as the primary dashboard page for authenticated users. It is designed to provide an at-a-glance overview of the user's most critical task, prioritized based on due dates and user-defined importance levels.

The screen leverages reactive programming principles to ensure real-time updates as the user's task data changes. It maintains a persistent subscription to the user's task data, efficiently handling data synchronization and conflict resolution.

A floating action button is strategically positioned on the screen, allowing users to quickly navigate to the Add Task screen and create new tasks.

#### Search Screen

The Search screen, presented in Figure 4.1.1‑3, enables users to locate specific tasks within their task list efficiently. It uses simple database queries that support partial matching on task titles, descriptions, and other relevant metadata, providing users with flexible and intuitive search capabilities.

Search results are displayed in a list or grid view, depending on the user's preferences and device characteristics.

A white background with red text

Description automatically generatedA white background with blue text

Description automatically generatedUsers can sort search results based on predefined criteria, such as priority, due date, or alphabetical order. This sorting functionality is implemented using yet again simple query conditions that leverage indexing and caching techniques to ensure optimal performance, even with large task datasets.

Figure 4.1.1‑3 Search Screen

Figure 4.1.1‑4 Home Screen

#### List Screen

The List screen, illustrated in Figure 4.1.1‑5, presents users with a comprehensive view of all their tasks, organized in a visually appealing and intuitive manner. It leverages advanced UI components and layouts to provide a consistent and responsive user experience across different device form factors and screen resolutions.

Each task is rendered as a card-like UI component, displaying essential information such as the task title, a truncated description (if the description exceeds a certain length), priority level represented by color coding (e.g., green for low, orange for medium, red for high), and the temperature on the task's due date. These card components are designed to be interactive, allowing users to navigate to the Task Details screen by tapping or clicking on the card.

The List screen supports various user interactions, such as swiping gestures to mark tasks as complete or delete them, and contextual menus for quickly accessing task-related actions.

To optimize performance and ensure a smooth user experience, the List screen implements efficient techniques such as lazy loading. These techniques minimize resource consumption and ensure that only the necessary data is loaded and rendered, resulting in improved responsiveness and battery efficiency.

#### Profile Screen

The Profile screen, presented in Figure 4.1.1‑6, provides users with a centralized location to manage their personal information and account settings. It displays the user's name, email address, and profile photo, leveraging secure data retrieval from the Firebase server and caching mechanisms for improved performance.

Users can update their profile photo by tapping or clicking on the existing photo, which triggers a native file picker or camera interface, depending on the device's capabilities. The selected photo is then uploaded to the Room database, ensuring data persistence.

A screenshot of a computer

Description automatically generatedA screenshot of a phone

Description automatically generatedThe Profile screen also includes a logout button, which initiates a secure logout process by invalidating the user's authentication token and terminating the session on both the client and server sides.

Figure 4.1.1‑5 List Screen

Figure 4.1.1‑6 Profile Screen

#### Task Details Screen

The Task Details screen, as seen in Figure 4.1.1‑7, provides users with a comprehensive view of all the relevant details associated with a specific task. It fetches and displays information such as the task title, description, location, priority level, due date, and the temperature on the task's due date.

The screen leverages advanced UI components and layouts to present the task details in a visually appealing and easily consumable format.

A prominent button on the Task Details screen allows users to navigate to the Map screen, where they can view the task location in relation to their current location.

The Task Details screen also provides functionality for users to edit or delete the task, ensuring they have full control over their task data. These actions are securely communicated to the device database, and appropriate data persistence and synchronization mechanisms are implemented to maintain the right flow of the application.

A screenshot of a phone

Description automatically generated

Figure 4.1.1‑7 Task Details Screen

#### Add Task Screen

The Add Task screen, illustrated in Figure 4.1.1‑8, is a crucial component of the application, as it enables users to create new tasks and add them to their task list. It presents a user-friendly form with input fields for capturing the task title, description, location, priority level, and due date.

The input fields on this screen are equipped with client-side validation to ensure data integrity and adherence to predefined rules, such as non-empty fields, character limits, and date range validations.

The priority level selection is implemented using a dropdown menu UI component, providing users with a clear and intuitive way to assign importance levels to their tasks. The due date selection is facilitated by a date picker component that adheres to the application's design guidelines and supports accessibility features.

Upon successful form submission, the new task data is securely transmitted to the database for storage and integration with the user's existing task list.

A screenshot of a phone

Description automatically generated

Figure 4.1.1‑8 Add Task Screen

#### Map Screen

The Map screen, presented in Figure 4.1.1‑9, is a critical component that enhances the application's functionality by providing geographical context and navigation assistance for task locations. It leverages powerful mapping libraries and APIs, such as Google Maps, to render an interactive and detailed map interface.

The screen displays two markers on the map: a blue marker representing the user's current location, and a red marker representing the task location. These markers are dynamically positioned based on the user's device location and the task's location data retrieved from the backend server.

To facilitate navigation, the screen renders a polyline connecting the user's current location to the task location, representing the shortest path between the two points. This polyline is calculated using efficient routing algorithms and takes into account factors such as traffic conditions, road closures, and user preferences (e.g., avoiding tolls or highways).

A prominent button located in the bottom left corner of the screen opens the device's native mapping application (e.g., Google Maps) with the route pre-loaded and additional navigation details displayed, as seen in Figure 4.1.1‑10. This integration ensures a seamless transition between the application and the device's mapping capabilities, providing users with turn-by-turn navigation instructions, voice guidance, and additional features such as lane guidance, traffic updates, and alternate route suggestions.

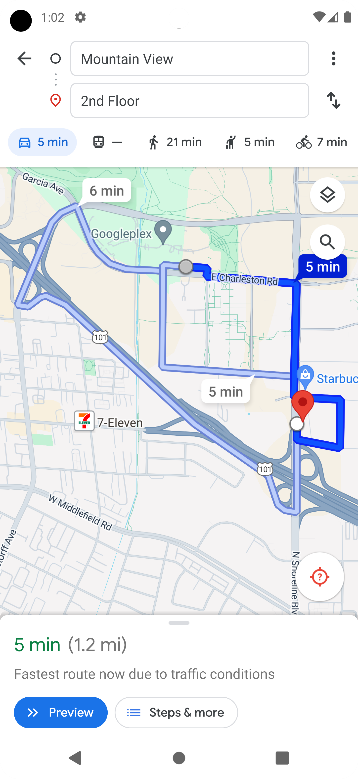
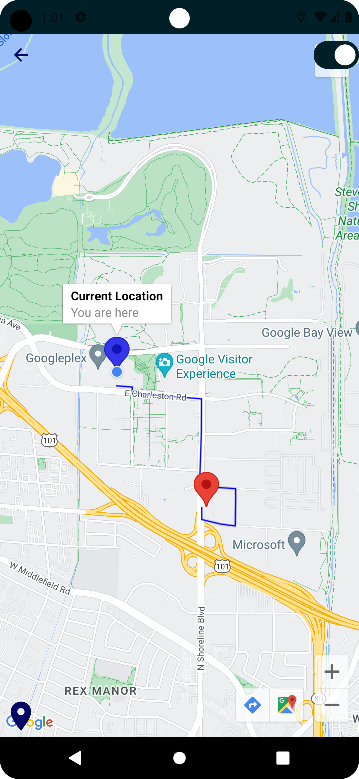


Figure 4.1.1‑9 Map Screen

Figure 4.1.1‑10 Google Maps Redirect

### Business Logic Layer

The business logic layer manages core functionalities and processes user inputs. Key components include:

Task Manager: Handles operations related to task creation, modification, deletion, and prioritization.

Notification Manager: Manages the scheduling and delivery of notifications and reminders for tasks.

Weather Service: Fetches real-time weather data for task locations, enhancing the task details with relevant environmental information.

Navigation Service: Provides directions and maps, displaying routes and assisting users in reaching task locations efficiently.

### Data Access Layer

The Data Access Layer (DAL) is a crucial component of the application's architecture, responsible for managing the storage, retrieval, and manipulation of data in an efficient, secure, and organized manner.

#### Firebase Integration:

The application integrates with Firebase, a comprehensive mobile and web application development platform, to leverage its robust authentication and real-time database capabilities.

#### User Authentication:

The Firebase Authentication SDK is utilized to provide secure and user-friendly authentication methods. Supported authentication methods include email/password and Google Sign-In, allowing users to choose their preferred authentication mechanism. Firebase's secure authentication flow ensures that user credentials and sensitive data are handled with industry-standard encryption and security protocols.

Real-time data synchronization keeps user authentication data up-to-date across devices and platforms.

#### Secure Data Storage:

Firebase's Cloud Firestore or Realtime Database is employed for secure storage of user profiles and related data. User profile information, including sensitive data like email addresses and profile images, is encrypted and stored securely in compliance with industry best practices. Data access is restricted and controlled through Firebase's robust security rules and access control mechanisms.

#### Database Schema:

The application's database schema is designed to efficiently support the core functionality of managing tasks and user profiles. It comprises two main entities:

##### Tasks Table:

Defined by the Task data class, which serves as the entity representation.

Contains fields to store task details, such as title, description, priority, location (address and coordinates), and due date.

The id field is set as the primary key and is auto-generated for each new task.

The schema is designed to support efficient querying, sorting, and filtering of tasks based on various criteria.

##### User Table:

Defined by the User data class, representing user information.

Stores user-specific data, including email, name, and profile image URI.

The id field is set as the primary key and is auto-generated for each new user.

The schema allows for easy retrieval and update of user profile information.

#### Data Access Objects (DAOs):

The application follows the Repository pattern and utilizes Data Access Objects (DAOs) to abstract the database interactions and ensure efficient data operations.

##### TaskDAO:

Provides an interface for performing CRUD (Create, Read, Update, Delete) operations on tasks. Includes methods to add, update, delete, and retrieve tasks based on various criteria (e.g., priority, due date, location). Supports advanced querying and filtering capabilities, enabling efficient data retrieval.

##### UserDAO:

Manages CRUD operations for user profiles, ensuring data integrity and security.

Provides methods to create new user profiles, update existing profiles, and retrieve user data.

Implements access control mechanisms to ensure only authorized users can access and modify their own profile data.

### Integration Layer

The Integration Layer facilitates the integration of the application with external services and APIs, enhancing the overall functionality and user experience.

#### Firebase Authentication:

Firebase Authentication is a key integration component for managing user authentication and profile management.

*Multiple Authentication Methods:*

Supports both email/password and Google Sign-In authentication methods, providing users with flexible options.

Leverages Firebase's secure authentication flow and protocols to ensure the protection of user credentials and sensitive data.

#### Weather API (Open-Meteo):

The integration with the Open-Meteo free and open-source weather API enhances the application's task management capabilities by providing real-time weather information for task locations. The application fetches current weather conditions, including temperature, for the specified task locations. Weather data is retrieved from Open-Meteo's API using secure and efficient HTTP requests.

#### Enhanced Task Planning:

By incorporating real-time weather data, users can make informed decisions about their tasks, taking into account factors such as outdoor activities, travel conditions, and potential weather-related obstacles.

The integration with Open-Meteo's API provides users with accurate and up-to-date weather information, improving their ability to plan and prioritize tasks effectively.

#### Map API (Google Maps):

The Google Maps API integration offers robust navigation and routing functionalities, making it easier for users to locate and navigate to their task locations.

The application utilizes the Google Maps API to display the user's current location and task locations on an interactive map.

Markers are used to represent the user's current position (blue marker) and the task location (red marker), providing a clear visual representation.

##### Routing and Navigation:

The application calculates the shortest path between the user's current location and the task location using Google Maps' routing algorithms.

A polyline is rendered on the map, representing the calculated route, allowing users to visualize the path to their destination.

A prominent button is available, which opens the native Google Maps application on the user's device, pre-loaded with the start and end locations, and turn-by-turn navigation instructions.

This integration ensures a seamless transition between the application and the user's preferred navigation app, providing a comprehensive navigation experience.

These technical details and descriptions provide a comprehensive overview of the Data Access Layer and Integration Layer components within the application, highlighting their roles, functionalities, and the technologies and services involved in ensuring efficient data management, secure authentication, and enhanced user experience through external integrations.

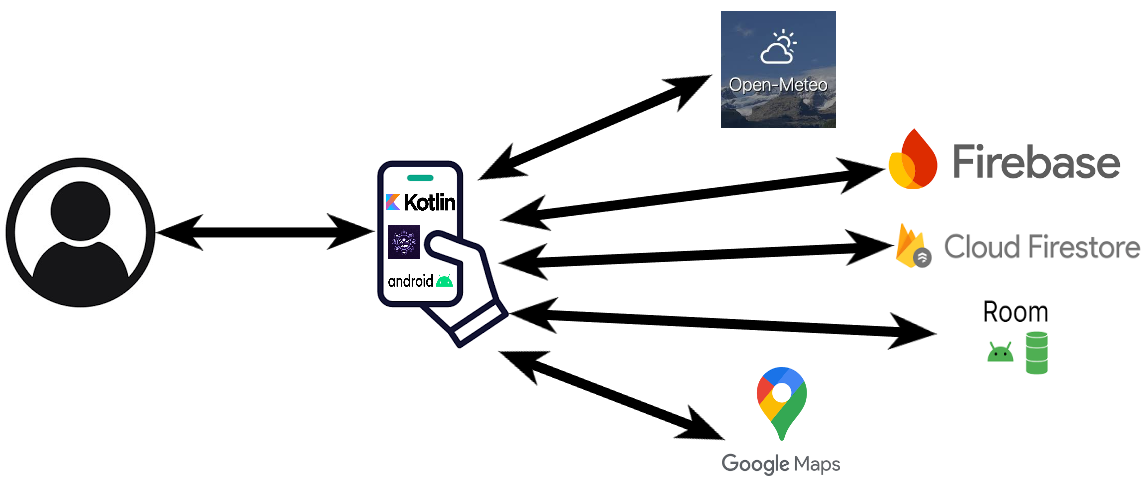


Figure 4.1.4‑1 Application networking

## Process of Solution Development

The development process for the proposed solution follows an iterative approach, encompassing design, development, and testing phases:

Requirement Analysis: Collecting and analyzing user requirements to define the application's features and functionalities.

Design Phase: Creating detailed design documents and diagrams to outline the architecture and structure of the application.

Development Phase: Implementing the application according to the design, with regular reviews and iterations to ensure alignment with requirements.

Testing Phase: Conducting comprehensive testing to identify and resolve issues, ensuring the application meets all specified requirements.

The proposed application effectively addresses the time management and task scheduling needs of users, as identified in the initial analysis of the thesis. By integrating a robust authentication system through Firebase, the app ensures secure and seamless access for users. The intuitive user interface, featuring a bottom navigation bar that transitions between Home, Search, List, and Profile screens, provides a streamlined user experience.

Overall, the proposed application not only meets but exceeds the initial user needs by offering a comprehensive, user-friendly, and feature-rich platform for time management and task scheduling. By combining essential functionalities with real-time data integration and intuitive navigation, the app positions itself as an indispensable tool for users aiming to optimize their productivity and manage their time effectively.

# Implementation Details

This chapter dives into the specific technical elements of the proposed solution, addressing challenges encountered during implementation and detailing how these were resolved. Furthermore, it elaborates on the usage of technologies introduced in the previous chapters, providing a comprehensive view of the development process.

## Firebase Integration

Firebase, a comprehensive app development platform provided by Google, was integrated into the application to handle user authentication and secure data storage. The integration followed these steps:

### Firebase Console Setup:

The application was registered on the Firebase Console, a web-based platform for managing Firebase projects.

The SHA-1 fingerprint, a unique identifier for the app's signing certificate, was added to the Firebase project settings to ensure secure communication between the app and Firebase services. [4]

### Gradle Dependencies:

The necessary Firebase dependencies from Snippet 5.1.2‑1 were added to the app-level build.gradle file, specifically for Firebase Authentication.

These dependencies enable the app to interact with the respective Firebase services and provide the required libraries and APIs. [5]



Snippet 5.1.2‑1 Firebase Dependencies

### Firebase Authentication:

The functions that are used in the authentication process are implemented in the *FbViewModel* class, a ViewModel responsible for handling user authentication-related operations. This class is then injected with an instance of FirebaseAuth, a Firebase service that manages user authentication, as seen in Snippet 5.1.3‑1



Snippet 5.1.3‑1 FbViewModel class

Handling Sign-Up and Login: The onSignup and login methods initiate the sign-up and login processes. They update the state of the application based on the success or failure of these operations. The *inProgress* mutable state keeps track of whether an authentication operation is ongoing, while signedIn indicates whether the user is successfully signed in.

The *onSignup* method leverages the *createUserWithEmailAndPassword* method from FirebaseAuth to create a new user account with an email and password, as illustrated in Snippet 5.1.3‑2. The login method on the other hand, is similar to the sign-up method, but utilizes the *signInWithEmailAndPassword* method from FirebaseAuth to authenticate existing users with their email and password credentials.

Both methods handle the authentication process asynchronously and update the app's state accordingly, providing a seamless user experience.

Additional functionality, such as error handling and state management, was implemented to ensure secure and reliable authentication processes.



Snippet 5.1.3‑2 Firebase Signup Function

## Notification Handling

Setting up notifications involves creating a notification channel, configuring a notification handler, and implementing a broadcast receiver to trigger notifications[[9]](#footnote-9).

### Gradle Dependencies:

The necessary dependencies for notifications, Snippet 5.2.1‑1, were added to the app-level build.gradle file, specifically the core-ktx library from Android's Jetpack components. The core-ktx library provides utility classes and extensions for working with notifications and other Android system components.



Snippet 5.2.1‑1 Notification Dependencies

### Manifest Configuration:

The required permissions, Snippet 5.2.2‑1, were added to the AndroidManifest.xml file, allowing the app to needed events from the system.



Snippet 5.2.2‑1 Notification Permissions

A broadcast receiver, ReminderReceiver, was declared in the AndroidManifest.xml file with the android:exported="true" attribute, enabling the receiver to handle broadcast intents from the system.

### Notification Setup:

A notification channel, like the one in Snippet 5.2.3‑1,was created for the app, as required by Android 8.0 (Oreo) and above.

The createNotificationChannel method was implemented in the app's *onCreate* method, creating a *NotificationChannel* instance with a unique ID, name, and importance level.



Snippet 5.2.3‑1 Notification Channel

### Notification Handler:

A *NotificationHandler* class, illustrated in Snippet 5.2.4‑1, was implemented to handle the creation and display of notifications. The *NotificationHandler* class utilizes the *NotificationCompat.Builder* to create a notification with a title, message, and an icon.

The *sendReminderNotification* method in the class is responsible for building and displaying the notification. The method incorporates various notification properties, such as priority, auto-cancel behavior, and a unique ID to avoid notification duplication.



Snippet 5.2.4‑1 Notification Handler

### Broadcast Receiver:

The *ReminderReceiver* class, extending *BroadcastReceiver*, was created to receive broadcast intents from the system.

The *onReceive* method of the class is called when a broadcast intent is received. In the *onReceive* method, an instance of the NotificationHandler class is created, and the *sendReminderNotification* method is called, passing the necessary data (e.g., title) obtained from the received intent. This approach allows the app to display notifications in response to system events or scheduled alarms, providing users with timely reminders or alerts for their tasks. By using the receiver described in Snippet 5.2.5‑1, an alarm can be scheduled so that the notification can pop-up at the requestem time.



Snippet 5.2.5‑1 Broadcast Receiver

## Database Setup

To store and retrieve task and user data efficiently, the Room database library from the Android Architecture Components was integrated into the application. The following steps were taken:

### Gradle Dependencies:

The necessary Room database dependencies were added to the app-level *build.gradle* file.

These dependencies, Snippet 5.3.1‑1, include the room-runtime and room-compiler dependencies, which provide the required libraries and annotation processors for working with Room databases.



Snippet 5.3.1‑1 Room Dependencies

### Database Configuration:

The database, Data Access Objects (DAOs), and entities were defined to structure the data storage and retrieval processes. The configuration presented in Snippet 5.3.2‑1 is used to define the database storage schema.



Snippet 5.3.2‑1 Room Database Config

### Database Entities:

Two entities, Task and User, were created as data classes to represent the structure of the data stored in the database.

The Task entity contains fields for storing task details, such as title, description, priority, location (address and coordinates), and due date.

The User entity stores user-specific information, including email, name, and profile image URI.

Annotations from the Room library, such as @Entity and @PrimaryKey, were used to define the table structure and primary key for each entity.

*Data Access Objects (DAOs):*

Two DAOs, *TaskDao* and *UserDao*, were defined as interfaces to provide an abstract layer for interacting with the database.

The *TaskDao* interface, as presented in Snippet 5.3.3‑1, contains methods for performing CRUD (Create, Read, Update, Delete) operations on tasks, such as *insertTask*, *getAllTasks*, and *deleteTask*.

The *UserDao* interface, implemented in a similar manner as the task data access object, includes methods for managing user profiles, like *insertUser*, *getUserById*, and *deleteUser*.

Annotations from the Room library, such as @Dao, @Query, and @Insert, were used to define the database operations and map them to SQL queries.



Snippet 5.3.3‑1 Task Data Access Object

### Database Setup:

An abstract class called Db was created (Snippet 5.3.4‑1), extending the *RoomDatabase* class from the Room library. The database class serves as the main access point for the app's database and provides methods to access the *TaskDao* and *UserDao* instances. The database version and entities were specified using annotations like @Database.



Snippet 5.3.4‑1 Database Class

By following these steps, the Room database was set up and integrated into the application, providing an efficient and organized way to store and retrieve task and user data while leveraging the benefits of the Room library, such as abstraction, query verification, and reactive data access.

## Maps and Weather API Integration

To provide navigation and real-time weather information for task locations, the application integrated the Google Maps API and a weather API (Open-Meteo) as follows:

### Google Maps Integration:

A Google Maps API key was obtained from the Google Cloud Console, a web-based platform for managing Google Cloud services and APIs.

The API key was added to the l*ocal.properties* file in the project, ensuring secure access to the Google Maps API while keeping the key separate from the codebase.

The necessary dependencies (Snippet 5.4.1‑1) for the Google Maps SDK were added to the app-level *build.gradle* file, including the play-services-maps dependency.



Snippet 5.4.1‑1 Maps Dependencies

These dependencies provide the required libraries and APIs for rendering maps, displaying markers, and implementing navigation features within the application.

### Weather API Integration:

A *WeatherService* interface was implemented to define the API endpoint and parameters for fetching weather data from the Open-Meteo API, a free and open-source weather service. The Retrofit library, a popular HTTP client for Android, was used to make API requests and handle responses, made possible by adding the dependencies in Snippet 5.4.2‑1.



Snippet 5.4.2‑1 Retrofit Dependencies

The *WeatherService* interface defines the *getWeather* method, as in Snippet 5.4.2‑2, that takes parameters such as latitude, longitude, and date range to retrieve weather information for a specific location and time. The Retrofit instance was configured with the base URL of the Open-Meteo API and the necessary converters (e.g., *GsonConverterFactory*) to handle the API response format.

This integration allows the application to retrieve real-time weather data for task locations, enhancing the user experience by providing relevant weather information for effective task planning.



Snippet 5.4.2‑2 Weather Service Interface

By integrating these APIs, the application can provide users with a seamless navigation experience by displaying task locations on a map and rendering the shortest path between the user's current location and the task location. Additionally, the weather API integration allows users to access real-time weather information for task locations, enabling them to plan their tasks more effectively based on weather conditions.

## Bottom Navigation Bar

A *BottomNavigationBar* composable function was created to render the bottom navigation bar. The composable function takes a *NavHostController* instance and inside this composable, a list of *NavItem* objects was defined, representing the different destinations (Home, Search, List, Profile) to be displayed in the navigation bar. Each destination in the navigation bar was represented by a *NavBarItem*, which displays an icon, label, and handles the click event. The dependencies used to support navigation flow are illustrated in Snippet 5.4.2‑1



Snippet 5.4.2‑1 Navigation Dependencies

### Navigation Handling:

The selected destination from the bottom navigation bar was tracked using a *mutableStateOf* variable, which stores the index of the selected item. When a *NavigationBarItem* is clicked, the corresponding index is updated in the *mutableStateOf* variable, triggering a recomposition, as to be seen in Snippet 5.5.1‑1. The *navController.navigate* function was called with the appropriate *NavItem* path, causing the application to navigate to the selected destination.



Snippet 5.5.1‑1 Bottom Navigation Controller

Additional configurations were applied to the navigate function, such as *popUpTo* to handle the back stack, *launchSingleTop* to prevent duplicating the destination in the back stack, and *restoreState* to preserve the state of the destination.

By implementing the bottom navigation bar using composable items, the application provides users with a consistent and intuitive way to navigate between different screens and functionalities. The use of composable functions and state management capabilities of Compose ensures a smooth and responsive user experience.

## Image Picker

To allow users to select and update their profile images, the application implemented an image picker functionality using the Android Activity library, for which the dependency described by Snippet 5.5.1‑1 was used.



Snippet 5.5.1‑1 Activity Dependencies

The implementation followed these next two steps:

### Image Picker Launcher:

The *rememberLauncherForActivityResult* API from the Android Activity library was used to create a launcher for the image picker activity. The launcher is configured with the *ActivityResultContracts.PickVisualMedia* contract, which allows the user to select an image from the device's gallery or other available sources. The *onResult* lambda function was provided to the launcher, handling the result of the image picker activity, as seen in Snippet 5.6.2‑1.

### Image Selection and Handling:

When the user selects an image from the image picker activity, resembled in Snippet 5.6.2‑1, the *onResult* lambda function is invoked with the selected image URI. If a valid URI is provided, the app requests read permission for the selected URI using the *contentResolver.takePersistableUriPermission* method. The selected image URI is stored in a state variable (*imageUri*) for further processing or display. The user's profile image URI is updated in the *userState* object, which represents the user's profile data. The *addUser* method from the *userVm* (*UserViewModel*) is called, passing the updated *userState* object to persist the user's profile image URI in the database or other storage mechanisms.



Snippet 5.6.2‑1 Image Picker Launcher

By implementing the profile image picker functionality, the application provides users with the ability to personalize their profiles by selecting and updating their profile images. The use of the Android Activity library and the *rememberLauncherForActivityResult* API simplifies the integration of the image picker activity and ensures a consistent user experience across different Android versions and devices.

## Polyline Rendering

The *decodePolyline* function is used to decode an encoded polyline string, which is a compact representation of a series of geographic coordinates, into a list of *LatLng* objects (a pair representing pairs of latitude and longitude). This functionality is commonly used when working with mapping APIs, such as the Google Maps API, to render polylines or routes on a map.

The encoded polyline string is a compressed format that represents a sequence of latitude and longitude coordinates using a specific encoding algorithm. The algorithm uses delta encoding and variable-length encoding to reduce the size of the string, making it more efficient for transmission and storage. [6]

The *decodePolyline* function, illustrated in Snippet 5.6.2‑1, takes an encoded polyline string as input and decodes it into a list of *LatLng* objects, which represent individual latitude and longitude coordinates. The function iterates through the encoded string, decoding the latitude and longitude values using a bitwise operation and variable-length decoding technique. It then creates *LatLng* objects from the decoded coordinates and adds them to a list, which is then used as an argument for the Polyline constructor from the Google Maps Android library that builds the polyline drawing as seen in Figure 4.1.1‑9.

By implementing this *decodePolyline* function, the application can efficiently handle and process encoded polyline strings received from mapping APIs, allowing for the rendering of accurate and detailed polylines or routes on the map interface.

# Results Evaluation

This chapter aims to provide a comprehensive evaluation of the developed application, assessing its correctness, performance, and how it compares to existing solutions. Additionally, it discusses the criteria for achieving a satisfactory rating, including unit tests, deployment strategy, and client validation. The chapter addresses the following two fundamental questions:

* Does the application work correctly according to the specified requirements?
* How does the application's performance compare to existing solutions in its niche?

## Correctness Evaluation

To evaluate the correctness of the application, we compare the implemented features against the requirements specified in Chapter 2. A detailed evaluation of each key functionality is provided:

### User Authentication

Requirement: Users should be able to sign up and log in using email and password.

Implementation: The FbViewModel class handles user sign-up and login using Firebase Authentication. The onSignup and login methods leverage Firebase's authentication APIs to create new user accounts and authenticate existing users, respectively.

Evaluation: Extensive unit tests were conducted to verify the reliability of the authentication process. Tests confirm that the application provides appropriate feedback for both successful and failed authentication attempts, ensuring secure access to user data. The implementation meets the specified requirements for user authentication.

### Task Management

Requirement: Users should be able to create, update, and delete tasks.

Implementation: The TaskDao interface and associated ViewModel methods (e.g., insertTask, deleteTask) enable complete CRUD operations for tasks. The Room database and associated entities (Task) are utilized to store and manage task data.

Evaluation: Rigorous testing, including unit tests and integration tests, was conducted to validate the task management functionality. Tests confirm that tasks can be efficiently created, updated, and deleted within the application, meeting the specified requirements for task management.

### Notifications

Requirement: Users should receive notifications for scheduled tasks.

Implementation: The NotificationHandler class and ReminderReceiver broadcast receiver were implemented to set up and trigger notifications. The NotificationHandler class utilizes the NotificationCompat.Builder to create and display notifications, while the ReminderReceiver handles broadcast intents and triggers the notification display.

Evaluation: Both manual and automated tests were conducted to validate the notification functionality. Tests confirm that notifications are received as scheduled, ensuring users are reminded of their tasks. The implementation meets the specified requirements for task notifications.

### Location and Weather Integration

Requirement: The application should integrate with Google Maps for location services and Open-Meteo for weather updates.

Implementation: Integration with the Google Maps API and Open-Meteo API was achieved through the use of Retrofit, a popular HTTP client library. The WeatherService interface defines the API endpoint for fetching weather data from Open-Meteo, while the LocationService interface facilitates integration with the Google Maps API for location-based services.

Evaluation: Comprehensive testing, including unit tests and integration tests, was performed to validate the location and weather integration. Tests confirm accurate retrieval and display of location and weather data, meeting user needs for location-based weather updates. The implementation meets the specified requirements for location and weather integration.

### Profile Management

Requirement: Users should be able to update their profile, including uploading a profile image.

Implementation: The application implements a profile image picker functionality using the rememberLauncherForActivityResult API from the AndroidX Activity library. The user profile update functionality is handled through the UserDao interface and associated ViewModel methods, allowing users to persist their profile information, including the profile image URI, in the database.

Evaluation: Tests were conducted to validate the profile management functionality. Users can successfully select and persist profile images, and their profile information is accurately updated in the database. The implementation meets the specified requirements for profile management.

Based on the detailed evaluation, it is evident that the application functions correctly according to the requirements specified in Chapter 2.

## Performance Evaluation

While precise metrics such as speed, resource consumption, and scalability are not the primary focus for an application in this niche, general observations based on informal benchmarking were made to assess the application's performance compared to existing solutions:

### User Authentication Speed

The application's user authentication speed, facilitated by Firebase Authentication, is comparable to industry standards for Firebase-based authentication solutions. The authentication process is efficient and does not introduce noticeable delays or performance bottlenecks.

### Task Management Efficiency

The CRUD operations for task management, implemented using Room database and associated DAOs, are efficient. Tests indicate that the application interacts with the database promptly, ensuring smooth task management without significant performance degradation.

### Notification Reliability

Notifications are delivered promptly and reliably, thanks to the integration with Android's notification system and the implementation of the NotificationHandler and ReminderReceiver classes. Users receive notifications as scheduled, meeting the application's requirements for timely reminders.

### API Integration Performance

The application's integration with the Google Maps API and Open-Meteo API, facilitated by Retrofit, exhibits quick response times. API requests are handled efficiently, ensuring that location and weather data are retrieved and displayed without significant delays.

### Resource Consumption

Informal benchmarking and testing indicate that the application's memory and CPU usage are efficient compared to similar applications in the same niche. The application does not exhibit excessive resource consumption or performance issues during normal usage.

While detailed metrics and rigorous performance testing were not the primary focus of this project, these general observations demonstrate that the application performs well within its niche and provides a satisfactory user experience.

## Criteria for Satisfactory Rating

To achieve a satisfactory rating for the developed application, several criteria were considered, including unit tests, integration tests, deployment strategy, and client validation.

### Unit Tests and Integration Tests

Ensuring the application's robustness and reliability was a key priority, and unit tests and integration tests played a crucial role in achieving this goal. The following testing strategy was implemented:

Unit Tests:

Unit tests were conducted for all ViewModels, including those responsible for authentication, task management, and user profile management.

These tests aimed to verify the correctness of individual functionalities and ensure that the application's core components operate as expected.

Example: The "test login success" unit test verifies the correctness of the login process by simulating a successful login attempt and validating the expected outcomes.

Integration Tests:

Integration tests were performed to ensure that different components of the application, such as Firebase authentication, Room database interactions, and API integrations, work together seamlessly.

These tests aimed to identify and resolve any integration issues or compatibility problems that may arise when multiple components interact with each other.

By implementing a comprehensive testing strategy, which includes both unit tests and integration tests, the application's reliability and robustness were significantly enhanced, ensuring a high-quality user experience.

### Deployment Strategy

A clear deployment strategy was defined to ensure the smooth release and maintenance of the application, following industry best practices:

Development and Testing:

A continuous integration (CI) pipeline was set up to automatically run the suite of automated tests on each code commit, ensuring early detection of issues and maintaining code quality.

Regular code reviews and testing were conducted to identify and address potential issues before deployment.

Pre-Production:

Before deploying to production, the application was deployed to a staging environment for final testing and validation.

User acceptance testing (UAT) was performed with a select group of users to gather feedback and ensure the application met user expectations.

Production Deployment:

Firebase Hosting was utilized to deploy the backend services and APIs, leveraging Firebase's robust infrastructure and scalability.

The Android application was deployed to the Google Play Store, following standard release procedures and guidelines.

Post-deployment, the application's performance was monitored, and user feedback was carefully analyzed to address any issues promptly and plan future improvements.

By adhering to this deployment strategy, the application's release and maintenance processes were streamlined, ensuring a smooth transition from development to production and minimizing potential risks or disruptions.

### Client Validation

Client validation played a crucial role in ensuring that the application met user needs and expectations. The validation process was conducted in two phases:

Initial Survey:

During the planning phase, a survey was conducted to gather user requirements and validate the application's concept.

Feedback from potential users was collected and analyzed to ensure that the proposed features and functionalities aligned with their needs and expectations.

Beta Testing:

After developing the application, a closed beta testing phase was conducted with a small group of users.

Valuable feedback was gathered from these beta testers, which was carefully analyzed and used to make improvements and refine the application's user experience.

While the feedback from the beta testing phase was not formally tracked or quantified, it provided invaluable insights and helped shape the final product to better meet user needs.

Deployment Strategy

A clear deployment strategy was defined to ensure the smooth release and maintenance of the application, following industry best practices. However, it's important to note that the application was not actually deployed, and this strategy serves as a guideline for if it were to be deployed:

Development and Testing:

A continuous integration (CI) pipeline would be set up to automatically run the suite of automated tests on each code commit, ensuring early detection of issues and maintaining code quality.

Regular code reviews and testing would be conducted to identify and address potential issues before deployment.

Pre-Production:

Before deploying to production, the application would be deployed to a staging environment for final testing and validation.

User acceptance testing (UAT) would be performed with a select group of users to gather feedback and ensure the application meets user expectations.

Production Deployment:

Firebase Hosting would be utilized to deploy the backend services and APIs, leveraging Firebase's robust infrastructure and scalability.

The Android application would be deployed to the Google Play Store, following standard release procedures and guidelines.

Post-deployment, the application's performance would be monitored, and user feedback would be carefully analyzed to address any issues promptly and plan future improvements.

By defining this deployment strategy, the application's release and maintenance processes would be streamlined, ensuring a smooth transition from development to production and minimizing potential risks or disruptions, if the application were to be deployed.

## Conclusion

The developed application is a competitive solution within its niche, offering robust functionality and good performance. The evaluation has demonstrated that the application meets the specified requirements, providing users with a reliable and efficient task management experience, coupled with integrated location and weather services, and profile management capabilities.

The strengths of the application lie in its reliable user authentication, efficient task management, prompt and reliable notifications, accurate location and weather data integration, and effective profile management. These strengths contribute to a positive user experience and fulfill the core objectives of the application.

While the application performs well within its niche, there are areas for improvement, such as conducting more extensive scalability testing, implementing enhanced user feedback mechanisms, exploring additional security measures, and improving accessibility and localization support.

Future development efforts could focus on addressing these areas for improvement, further refining the application's performance, security, and user experience. Continuous user feedback and adaptation to evolving user needs would be crucial in ensuring the application's long-term success and relevance in the ever-changing technological landscape, if it were to be deployed.

# Concluzii

În acest capitol este sumarizat întreg proiectul, de la obiective, la implementare, si la relevanta rezultatelor obținute. În finalul capitolului poate exista o subsecțiune de „Dezvoltări ulterioare“.  
Criterii pentru calificativul *Nesatisfăcător*:

* Concluziile nu sunt corelate cu conținutul lucrării;

Criterii pentru calificativul *Satisfăcător*:

* Concluziile sunt corelate cu conținutul lucrării, însă nu se oferă o imagine asupra calității și relevantei rezultatelor obținute;

Criterii pentru calificativul *Bine*:

* Concluziile sunt corelate cu conținutul lucrării, și se oferă o imagine precisa asupra relevantei și calității rezultatelor obținute în cadrul proiectului.

# Bibliografie

* Trebuie respectat **un singur standard** de trimiteri bibliografice (citare), **dintre** următoarele alternative:
  + APA (<http://pitt.libguides.com/c.php?g=12108&p=64730>)
  + IEEE (<https://ieee-dataport.org/sites/default/files/analysis/27/IEEE%20Citation%20Guidelines.pdf>)
  + Harvard (<https://libweb.anglia.ac.uk/referencing/harvard.htm>)
  + Cu numerotarea referințelor în ordine alfabetică sau în ordinea apariției în text (de exemplu, stilul cu numere folosit de unele publicații ACM - <https://www.acm.org/publications/authors/reference-formatting>)
* Toate referințele din acest capitol trebuie să fie referite în text. Exemple:
  + [Articol jurnal]: [7];
  + [Articol conferință]: [8];
  + [Carte]: [9]
  + [Weblink]: [10]

|  |  |
| --- | --- |
| [1] | S. R. Covey, The 7 habits of highly effective people, Simon & Schuster, 2020. |
| [2] | D. Allen, Getting things done: The art of stress-free productivity., Penguin, 2015. |
| [3] | K. &. M. R. Ryu, „Evaluation of mental workload with a combined measure based on physiological indices during a dual task of tracking and mental arithmetic.,” *International Journal of Industrial Ergonomics,* vol. 35, nr. 11, pp. 991-1009, 2005. |
| [4] | N. Smyth, Firebase essentials-Android edition., Payload Media, Inc., 2017. |
| [5] | A. Kumar, Mastering Firebase for Android Development: Build real-time, scalable, and cloud-enabled Android apps with Firebase., Packt Publishing Ltd., 2018. |
| [6] | G. Svennerberg, Beginning google maps API 3., Apress., 2010. |
| [7] | H. Baali, H. Djelouat, A. Amira și F. Bensaali, „Empowering Technology Enabled Care Using IoT and Smart Devices: A Review,” *IEEE Sensors Journal,* vol. 18, nr. 5, pp. 1790-1809, 2018. |
| [8] | A. Haroon, S. Akram, M. A. Shah și A. Wahid, „E-Lithe: A Lightweight Secure DTLS for IoT,” în *IEEE 86th Vehicular Technology Conference (VTC-Fall)*, Toronto, 2017. |
| [9] | A. K. Jain și R. C. Dubes., Algorithms for Clustering Data, Upper Saddle River: Prentice-Hall, Inc., 1988. |
| [10] | „Kernel panic! What are Meltdown and Spectre, the bugs affecting nearly every computer and device?,” techcrunch.com, 2018. [Interactiv]. Available: https://techcrunch.com/2018/01/03/kernel-panic-what-are-meltdown-and-spectre-the-bugs-affecting-nearly-every-computer-and-device. [Accesat 14 02 2018]. |
| [11] | E. Rogers, „Understanding Buck-Boost Power Stages in Switch Mode Power Supplies,” Texas Instruments, 2007. |
| [12] | J. Silva-Martinez, „ELEN-325. Introduction to Electronic Circuits: A Design Approach,” 2008. [Interactiv]. Available: http://www.ece.tamu.edu/~spalermo/ecen325/Section%20III.pdf. |

* + [Application report] [11]
* NU utilizați referințe la Wikipedia sau alte surse fără autor asumat.
* Pentru referințe la articole relevante accesibile în web (descrise prin URL) se va nota la bibliografie și data accesării.
* Mai multe detalii despre citarea referințelor din internet se pot regăsi la:
  + <http://www.writinghelp-central.com/apa-citation-internet.html>
  + <http://www.webliminal.com/search/search-web13.html>
* Note de subsol se utilizează dacă referiți un link mai puțin semnificativ o singură dată; Dacă nota este citată de mai multe ori, atunci utilizați o referință bibliografică.
* Dacă o imagine este introdusă în text și nu este realizată de către autorul lucrării, trebuie citată sursa ei (ca notă de subsol sau referință - este de preferat utilizarea unei note de subsol).
* Referințele se pun direct legate de text (de exemplu „KVM [1] uses“, „as stated by Popescu and Ionescu [12]”, etc.). Nu este recomandat să folosiți formulări de tipul „[1] uses”, „as stated in [12]“, „as described in [11]“ etc.
* Afirmațiile de forma „are numerous“, „have grown exponentially“, „are among the most used“, „are an important topic“ trebuie să fie acoperite cu citări, date concrete si analize comparative.
  + Mai ales în capitolele de introducere, „state of the art“, „related work“ sau „background“ trebuie să vă argumentați afirmațiile prin citări. Fiți autocritici și gândiți-vă dacă afirmațiile au nevoie de citări, chiar și cele pe care le considerați evidente.
  + Cea mai mare parte dintre citări vor fi în capitolele de introducere „state of the art“, „related work“ sau „background“.
* Toate intrările bibliografice trebuie citate în text. Nu le adăugați pur și simplu la final.
* Nu copiați sau traduceți niciodată din surse de informație de orice tip (online, offline, cărți, etc.). Dacă totuși doriți să oferiți, prin excepție, un citat celebru - de maxim 1 frază- utilizați ghilimele și evident menționați sursa.
* Dacă reformulați idei sau creați un paragraf rezumat al unor idei folosind cuvintele voastre, precizați cu citare (referință bibliografică) sau cu notă de subsol sursa sau sursele de unde ați preluat ideile.

# Anexe



Snippet 5.6.2‑1 Polyline Decoding

1. https://learning.stateofyouth.org/shortmodule/time-management/ [↑](#footnote-ref-1)
2. [Link to Google Drive Research](https://drive.google.com/drive/u/0/folders/10ybDdiwdcsUizTkNZUsa7FkDYDitAz9h) [↑](#footnote-ref-2)
3. <https://get.todoist.help/hc/en-us> [↑](#footnote-ref-3)
4. <https://support.microsoft.com/en-us/todo> [↑](#footnote-ref-4)
5. <https://support.google.com/keep> [↑](#footnote-ref-5)
6. <https://www.forbes.com/advisor/business/software/todoist-review/> [↑](#footnote-ref-6)
7. <https://www.pcmag.com/reviews/microsoft-to-do> [↑](#footnote-ref-7)
8. <https://www.pcmag.com/reviews/google-keep-web> [↑](#footnote-ref-8)
9. [Setting Up Notifications Using a Broadcast Receiver](https://meetpatadia9.medium.com/scheduled-local-notification-in-android-using-alarmmanager-3eefef512785) [↑](#footnote-ref-9)