Fachhochschule Dortmund
University of Applied Sciences and Arts
(Digital Transformation)

Optimizing Healthcare Appointment Scheduling: A Software Engineering Approach

Research Project (Thesis)

Authors: Zevalov, Artur (7216660)
All Gharaee, Navid (7216638)
All Gharaee, Saeed (7216639)

Supervisor: Prof. Dr. Christian Reimann

Date: March 2024

Abstract

This research thesis seeks to advance the field of healthcare appointment scheduling and appointment management through the application of modern software engineering practices. The core research problem revolves around the development and evaluation of the "Avicenna" scheduling application. Avicenna leverages Flutter for its front end and Django-based REST API for its back end, with a primary focus on optimizing the management of doctor-patient appointments.

The research methodology follows established software engineering principles, emphasizing techniques such as containerization, continuous integration, and adherence to industry-recognized software design patterns. The study aims to provide valuable insights into the software engineering aspects of Avicenna, with a strong emphasis on architectural robustness, scalability, maintainability, and security.

The research methodology encompasses an exploratory and iterative approach, involving usability testing, performance analysis, and stakeholder feedback. Through these methods, the thesis aims to derive valuable insights into the app's user interface design, system performance, user experience, and the challenges encountered by doctors and patients during its usage.

In summary, this research thesis offers a comprehensive exploration of Avicenna's software engineering aspects within the context of healthcare appointment scheduling. Avicenna's development journey serves as a model for excellence in software engineering, and the findings shall contribute to the broader understanding of software engineering practices in both the healthcare technology domain and overall application development alike.

Keywords: Healthcare, Appointment, Software Development, Flutter, Django, User Experience, User Interface

Table of Contents

Introduction

In the dynamic realm of healthcare, the imperative to optimize appointment scheduling emerges as a critical challenge, with far-reaching implications for both practitioners and patients. Against the backdrop of a growing demand for efficient healthcare services, the streamlining of appointment management processes becomes increasingly important [1]. This thesis undertakes a systematic exploration of this pressing need, grounded in contemporary software engineering practices.

The convergence of software engineering and appointment scheduling presents an alluring avenue for innovation. The synthesis of precision and efficiency not only serves the immediate needs of practitioners and patients but also contributes substantively to the overarching enhancement of healthcare delivery systems.

The reason for this research aligns with the shared efforts of healthcare institutions. As they grapple with the complexities of appointment scheduling, the need to make this process more efficient becomes a collective pursuit. How appointments are scheduled is closely tied to using resources well, keeping patients satisfied, and making sure healthcare systems perform at their best [2]. This research aims to offer insights and solutions that are relevant to the needs of healthcare providers and the wider scientific community.

This writ digs into the challenges of healthcare appointment scheduling. Unraveling the narrative, exploration focuses on the intricacies of Avicenna, an appointment-scheduling app crafted with a focus on employing the latest software engineering principles. The questions posed revolve around usability, performance, security, and the broader impact of integrating Avicenna into healthcare workflows.

Each chapter dissects the intricacies of employing software engineering practices in the development of Avicenna. Beginning with laying the groundwork in the Literature Review, which delves into existing knowledge surrounding healthcare appointment scheduling and the best modern software engineering practices, and the Problem Statement, where the specific challenges in the current landscape are elucidated and a conceptual framework that underpins subsequent empirical investigations is constructed. Leading to the Development of the Application, essentially the fundamentals of the application’s design, the heart of the research where the multifaceted aspects of Avicenna's development and validation are dissected: general information, and essential requirements are explored in the initial sections, laying the foundation for the subsequent exploration. The project's agility and adaptability are unpacked in the Project management, agile, scrum section, providing insights into the

dynamic orchestration of sundry software engineering practices. Containerization/Deployment unravels the technical intricacies of the employed approach, shedding light on the deployment pipeline.

A comprehensive overview of the tools instrumental in Avicenna's creation is presented in the Tools Overview section, offering a perspective into the technological infrastructure employed. The subsections within the chapter delve into the specifics of each tool employed in Avicenna's development, providing granular insights into Dart/Flutter, Python/Django, Docker, Git/GitHub, and more. The Mobile Application and Back-end sections zoom in on the intricacies of Avicenna's architecture, highlighting the considerations and decisions that define its user-facing and server-side components. Usability details the iterative process of refining Avicenna based on user experiences.

The narrative then transitions into the synthesis of findings and insights in Discussion of Results, where the outcomes of this research are critically analyzed against the background of existing knowledge and theoretical frameworks.

As the journey concludes in the Conclusion and Outlook section, the threads of the exploration are drawn together. Here, not only the findings are summarized, but also a course for future research endeavors is charted, acknowledging the dynamic nature of both healthcare technology and software engineering practices.

In navigating this well-structured path, readers shall gain a comprehensive understanding of the quest to optimize healthcare appointment scheduling through the lens of Avicenna and contemporary software engineering practices. Each chapter serves as a waypoint, illuminating a distinct facet of this research, culminating in a cohesive narrative that contributes substantively to the discourse on healthcare technology.

This paper indicates the potential direction for further research including:

- Assessment of user feedback and enhancing survey methodologies to a more advanced level.
- System recommendation based on the reviews and ratings.
- Integration with device-inbuilt services such as voice assistants.
- Accessibility features for disabled users.
- Implementation of the clinic assistants as a new type of user.
- Schedule optimization by expanding the user-to-user interaction [3].

4. Literature Review

In the contemporary landscape of software engineering, the convergence of innovative methodologies, project planning strategies, and efficient organizational structures has become pivotal for the successful development and deployment of software [4]. This literature review aims to explore and synthesize existing knowledge of modern software engineering principles, project planning, management, organizational structures, and DevOps practices such as containerization and deployment. The ultimate objective of this exploration is to inform the development of an application named Avicenna, designed to facilitate appointment scheduling between doctors and patients, as well as generally contribute to the software development field.

The specific objectives include studying:

- Software Engineering Principles:
  - Investigate modern software engineering principles that underpin the development process, emphasizing efficient code architecture, maintainability, and scalability.
- Project Planning and Management:
  - Explore current project planning methodologies and management techniques, with a focus on agile practices, iterative development, and effective team collaboration.
- Organizational Structures:
  - Examine organizational structures within software development teams, considering the impact on productivity, communication, and overall project success.
- DevOps Practices:
  - Investigate DevOps tasks, particularly containerization and deployment, to understand their role in enhancing the efficiency, scalability, and reliability of software applications.
- Application Development (Avicenna):
  - Connect the literature findings to the development of Avicenna, a scheduling application that shall utilize a Django back-end with REST API and a Flutter front-end.
- Usability Testing:
  - Investigate usability testing methodologies and best practices to ensure the Avicenna application meets user needs and expectations.

The literature review serves as the foundational framework for the development of the Avicenna application, offering valuable insights into the latest trends, best practices, and potential challenges within the realms of software engineering from A to Z. By synthesizing existing knowledge, this review aims to guide critical decision-making processes, enhance development strategies, and ultimately contribute to the creation of a robust and user-friendly appointment scheduling application.

The main themes to be explored in this work include the state-of-the-art application of software engineering principles, the dynamic landscape of project planning and management, the impact of organizational structures on software development teams, the integration of DevOps practice, and the application of these insights to the development and usability testing of the Avicenna application. Through an in-depth exploration of these themes, this literature review aims to provide a comprehensive understanding of the contemporary software development landscape and its relevance to the specific goals of the Avicenna project.

4.1. Theoretical framework

4.1.1. Project organization

In the realm of software engineering and project management, several key theories and concepts support the Agile methodology, which is central to this study.

In today's rapidly evolving business landscape, effective project management is of the greatest importance, particularly within the realm of software development. Among the traditional methodologies, the Waterfall approach stands out. A fundamental difference between Waterfall and Agile methodologies lies in their treatment of project requirements. Waterfall methodology requires the comprehensive definition of all requirements at the project's initiation, maintaining their static nature throughout subsequent phases. However, the inherent nature of software development introduces a dynamic environment where details are prone to continual modification [5]. Not only do requirements not provide every detail at the beginning of the projects, but they are subject to change as more questions in the technical part arise. Furthermore, since software development operates without strict rules or clear certainties to rely on, software products often have flaws or areas that could be improved, mainly due to the complexities and uncertainties involved in the development process [6].

Unlike Waterfall, Agile methodologies embrace this flexibility, allowing for iterative requirements adaptation as the project unfolds, accommodating changes even in later stages of development [7]. Consequently, within Agile frameworks, the use of specific requirement descriptions, such as user stories, becomes pivotal in structuring and delineating the diverse scopes and sizes of project requirements.

Given the diverse scope and levels of requirements, it becomes crucial to classify them into a structured hierarchy. These requirements are categorized into epics, user stories, and tasks to clarify their respective scopes and sizes within the project [8]. Therefore, drawing on those conclusions, the requirements for the Avicenna project are defined in chapter 7.2, using those specific categories.

4.1.2. Industry-standard software practices in Django development

The development of software applications in Django, a high-level Python web framework, necessitates a commitment to good software practices to ensure the creation of robust, maintainable, and scalable systems. This section explores the crucial reasons behind the adoption of such practices and provides an in-depth examination of the key aspects within the Django framework that contribute to a successful and sustainable software development process.

- **Maintainability and Readability:**
    - **Rationale:** Django projects often involve collaboration among multiple developers, making code maintainability a paramount concern. Well-structured and readable code facilitates comprehension, ease of debugging, and seamless collaboration. What is more, most widespread mistakes made with regard to accruing technical debt have already been identified and researched, which makes avoiding them a mere matter of familiarizing yourself with the best practices.
    - **Practices:**
        - **Follow PEP 8 Guidelines:** Adhering to Python Enhancement Proposal 8 ensures consistent coding style and enhances code readability.
        - **Use static code analyzers:** Using tools such as pylint, ruff, black, flake8, autopep8 and such, also known as code linting and/or formatting tools, allows to alleviate most mistakes. However, the

tools can only advise, and it is left up for the programmer to act upon their analysis.

        - **"Pay back" technical debt as soon as possible:** Technical debt has a tendency to either being repaid or burying all software development efforts under its weight.
    - **Found in:** [9] and [10]
- **Scalability:**
    - **Rationale:** As projects evolve, scalability becomes imperative. Django projects should be designed to handle increasing complexity and user loads without sacrificing performance.
    - **Practices:**
        - **Optimize Database Queries:** Efficiently structure and optimize database queries to prevent bottlenecks.
        - **Implement Caching Mechanisms:** Leverage caching techniques to reduce redundant computations and enhance response times.
        - **Be aware of the templates' bottlenecks:** While Django's implementation of templates is absolutely adequate for all kinds of use cases, some common pitfalls can be easily avoided by being aware of some of the peculiarities of the template system, such as performance degradation dependent on the number of pieces the templates are broken into.
    - **Found in:** [10], [11] and [12]
- **Security:**
    - **Rationale:** Security is a critical concern in web development, and Django offers inbuilt features to mitigate common vulnerabilities. Adhering to the best practices is vital for safeguarding applications and user data.
    - **Practices:**
        - **Use Django's Authentication System:** Leverage Django's robust authentication system to manage user authentication securely.
        - **Implement CSRF Protection:** Django provides inbuilt protection against Cross-Site Request Forgery (CSRF) attacks. Ensure its proper implementation.
        - **Regularly Update Dependencies:** Stay vigilant about security updates and regularly update Django and its dependencies.

    - **Found in:**
        - [13] and [14]
- **Modularity and Reusability:**
    - **Rationale:** Promoting modularity and reusability ensures that components of a Django project can be easily understood, maintained, and repurposed for future development.
    - **Practices:**
        - **Follow the Don't Repeat Yourself (DRY) Principle:** Minimize redundancy by encapsulating common functionalities in reusable components.
        - **Create Custom Django Apps:** Encapsulate specific functionalities into modular Django apps for easy integration into different projects.
    - **Found in:** [10], [12] and [14]
- **Testing:**
    - **Rationale:** Robust testing is essential for identifying and addressing bugs and ensuring that new features do not introduce regressions.
    - **Practices:**
        - **Use Django's Testing Framework:** Django provides a powerful testing framework for creating unit tests, integration tests, and functional tests.
        - **Implement Test-Driven Development (TDD):** Adopt TDD practices to write tests before the corresponding code, ensuring that the code meets the specified requirements.
    - **Found in:** [10], [15] and [14]
- **Documentation:**
    - **Rationale:** Comprehensive documentation is indispensable for onboarding new developers, maintaining the project, and understanding the purpose and usage of various components, especially so when the developer in questions returns to working on a project after some period time.
    - **Practices:**
        - **Generate and Maintain Documentation:** Utilize tools like Sphinx to generate documentation and keep it up to date with project changes.
        - **Include Comments in Code:** Add inline comments to clarify complex or non-intuitive sections of code.

        - **Adopt a documentation style:** There are several widely used documentation styles available, such as the Google style, the PEP 257 style, and such.
    - **Found in:** [10] and [14]
- **Deployment and DevOps:**
    - **Rationale:** The deployment phase is a critical aspect of software development. Streamlined deployment processes enhance project reliability and maintainability.
    - **Practices:**
        - **Containerization with Docker:** Utilize Docker for containerization to ensure consistency across different environments.
        - **Continuous Integration/Continuous Deployment (CI/CD):** Implement CI/CD pipelines for automated testing and deployment.
    - **Found in:** [16] and [17]

4.1.3. Industry-standard software practices in Flutter app development

Flutter, as a cross-platform mobile app development framework, utilizes Dart primarily as a programming language. This section focuses on the software engineering approaches to develop scalable, maintainable, accessible, and testable software while maintaining a comprehensible and readable code base.

- **Code structure and organization:**
    - **Rationale:** One of the crucial aspects of maintaining high-quality code lies in the commitment to a good coding style. Consistent practices in naming conventions, code ordering, and formatting contribute to a uniform appearance that respects visual coherence. Adopting a uniform coding style throughout the entire enables a more accessible and collaborative environment. Such consistency not only aids in comprehending code but also streamlines the process of learning from and contributing to the codebase developed by others.
    - **Practices:**
        - **Follow conventional naming styles:** Use UpperCamelCase for class names and extensions. For other identifiers such as variables, functions, and parameters, adopt lowerCamelCase. Ensure that

package directories and source files are named using lowercase_with_underscores.

        - **Use Dart Linter:** Utilize the Dart linter to detect potential issues in the Dart code and apply the right Linter rules. These issues include naming conventions, possible errors or mistakes in the code, and pub package setup.
    - **Found in:** [18], [19], and [20]
- **Design Patterns**
    - **Rationale:** It is always a good practice to use design patterns to add simplicity and scalability at the same time in addition to maintaining the code for an extended period.
    - **Practices:**
        - Design patterns highly contribute to the encapsulation and isolation of non-related elements. That comes in handy when there is a need for a change in some part of the app, without modifying the whole system.
        - It is also easier for developers to trace and debug the program, and it saves a great amount of time by structuring the program.
    - **Found in:** [21]
- **Internationalization and Localization**
    - **Rationale:** Accessibility of the app demands consideration of broader user demographics, particularly those who communicate in languages other than English.
    - **Practices:**
        - **Layouts:** In Latin-based languages, documents are structured from left to right, while Arabic and Hebrew scripts follow a right-to-left reading orientation. East Asian languages, such as Japanese, typically adopt a different layout, presenting text from top to bottom and lines from right to left. User interfaces should be tailored to accommodate these linguistic dynamics, and design solutions should align accordingly.
        - **Date formats:** Date formats vary across regions; for example, March 14th, 2000 is expressed as 03/14/2000 in some regions, while some

European countries use the format 14.03.2000. Additionally, different calendar systems, such as Hijri, Chinese, and Korean, introduce further variations. It is crucial to appropriately handle these diverse formats to ensure accuracy in data representation.

        - **Languages:** Instead of hard-coding labels in the presentation layer, organize all language-related strings in a dedicated file. Each language should have its own dictionary, aligning with equivalent strings in other languages.
    - **Found in:** [22] and [23]
- **Accessibility**
    - **Rationale:** An accessible user interface is defined by four key principles: Perceivability, Operability, Understandability, and Robustness. The principle of perceivability aims to make content perceivable for all users, specifically those with impairments like blindness or deafness. It involves testing whether alternatives, such as audio or image options, are available and accessible to individuals with disabilities. Operability guarantees the proper functioning of all features for diverse user groups, regardless of individual limitations. Understandability and robustness define the cognitive ability of the user to comprehend the meaning of the presented information and the adaptability of content in how it is interpreted by various user agents, respectively.
    - **Practices:**
        - Follow platform-specific guidelines such as iOS Human Interface Guidelines for Accessibility by Apple and Android User Interface Guidelines by Google.
        - Follow accessibility guidelines such as WCAG 2.0 (updated to WCAG 2.1 in 2018) and the U.S. Revised Section 508 standards, both of which encompass mobile accessibility.
    - **Found in:** [24] and [25]
- **Testing**
    - **Rationale:** Integration with Test Driven Design (TDD) architecture can be beneficial for projects that are growing progressively. As it appears in the name, Test Driven Design is highly focused on the testability of the code by applying unit test automation. It was first introduced due to agile development methodology.

- **Practices:**
  - It ensures that no production code is written before the test is successful.
  - Each test should be defined to accomplish only one single goal and all the tests performed in the iterative process.
- **Found in:** [26], [27]

4.1.4. Conclusion

The chapter discussed the significance of Agile methodology in software engineering and project management, contrasting it with the traditional Waterfall approach. Agile's flexibility allows for iterative adaptation of project requirements, accommodating changes throughout development, while Waterfall requires comprehensive initial requirements, which may not suit the dynamic nature of software development.

Furthermore, the adherence to good software practices is indispensable in Django development to ensure the longevity, security, and efficiency of projects. From coding style and modularity to security measures and deployment strategies, embracing these practices contributes to the creation of robust and maintainable Django applications. The subsequent practical section of this literature review delves deeper into this topic by applying aforementioned best practices to a real project – Avicenna.

At the same time, considering essential software practices is also critical for effective Flutter app development, emphasizing elements such as code structure, design patterns, internationalization, accessibility, and testing to guarantee robust and enduring projects. Consistent coding style, exemplified through conventions in naming, ordering, and formatting, fosters a collaborative and accessible development environment. Design patterns contribute to code simplicity and scalability, enabling easier maintenance and debugging. Internationalization considerations address linguistic and cultural diversity, while accessibility principles ensure the app meets standards for users with varying abilities. Additionally, integrating Test Driven Design (TDD) in testing practices provides a systematic approach to code testability and development. These practices collectively enhance the quality, sustainability, and user-centricity of Flutter app development.

4.2. Appointment scheduling in healthcare

As mentioned in [28], the process of appointment scheduling holds significant importance, particularly within the domain of outpatient clinics and diverse healthcare services. The complexities inherent in this process demand innovative solutions to navigate the variability and unpredictability associated with service duration and patient arrivals. A fundamental challenge lies in crafting strategies that not only optimize efficiency but also enhance the overall quality, cost-effectiveness, and capacity of healthcare services.

Effective appointment scheduling is more than a logistical puzzle; it is a critical determinant of patient experience and healthcare provider performance. The implications of an inadequate scheduling strategy tend to ripple through the system, leaving a lasting impact on both patients and providers. Long waiting times can erode patient satisfaction, while idle periods for healthcare providers lead to inefficiencies and potential financial repercussions [2].

Thus, the concept of flexibility takes center stage. A flexible approach acknowledges the inherent uncertainty and variability in healthcare, allowing for adaptability in the face of changing demands and unforeseen circumstances. This flexibility extends beyond mere logistical adjustments; it embodies a mindset that values responsiveness, patient-centricity, and a commitment to optimizing the delicate balance between patient waiting times and provider idle periods [29].

Therefore, appointment scheduling plays a pivotal role when it comes to ensuring efficient and timely access to health services, influencing both patient outcomes and satisfaction levels. Comprehensive exploration in [1] of appointment scheduling systems sheds light on the complexities and challenges inherent in primary care, specialty care, and surgical environments. Understanding the unique features and challenges of each setting is crucial for designing effective appointment management systems.

4.2.1. Primary care appointment scheduling: navigating complexities for enhanced patient access

In primary care, where the majority of patients require services within fixed time slots, appointment scheduling involves the allocation of standard and multiple slots to accommodate different visit types. The introduction of Advanced Access systems, as pioneered by [30], [31], and expanded on in [1], has revolutionized primary care scheduling by aiming to offer same-day appointments, enhancing patient focus and competitive

advantage. The innovative approach prioritizes patient convenience and aims to eliminate the need for a triage nurse by allowing patients to book appointments on the day of their call. However, challenges arise in absorbing variations in daily demand and accurately capturing the true demand for same-day services. For healthcare app development in primary care, addressing these challenges and incorporating features that enhance flexibility and responsiveness to patient needs could significantly improve the overall efficiency of appointment scheduling.

4.2.2. Specialty clinic appointment scheduling: navigating referrals and variable service times

Specialty care clinics, with variable service times and the need for referrals, pose additional complexities. [1] highlights the intricacies of managing referral-based appointments and the necessity to reserve capacity for urgent cases. Surgical appointment scheduling introduces further complications with variable procedure times, pre-surgery appointments, and the coordination of multiple resources. The two-stage process involving patients choosing time windows and subsequent confirmation by physicians exemplifies the intricate nature of scheduling surgeries.

Specialty care clinics, as highlighted by [1] present a distinct set of challenges in appointment scheduling due to variable service times and the need for referrals. Unlike primary care, where services can often be performed within fixed-length slots, specialists' appointments are highly variable and diagnosis-dependent. The coordination of referral-based appointments, often booked by medical assistants at periodic intervals, adds another layer of complexity, as it involves a the two-stage process whereby patients choose time windows which are subsequently sent for confirmation to physicians.

The literature also emphasizes the importance of reserving capacity for urgent appointment requests and maximizing the utilization of specialist time. For the development of a healthcare appointment app catering to specialty clinics, addressing these complexities and incorporating features that facilitate seamless referral-based scheduling, accommodate variable service times, and efficiently manage urgent appointments would be instrumental in enhancing the effectiveness of the scheduling process.

4.2.3. Surgical appointment scheduling: orchestrating resources for optimal efficiency

Surgical appointment scheduling introduces an additional set of obstacles, emphasizing the critical need for effective coordination of resources. [1] outline the two-stage process involved in surgical scheduling, where patients initially choose from a menu of available time windows, and specific appointments are later confirmed by physicians. The variability in procedure times, pre-surgery appointments, and the simultaneous scheduling of multiple resources, such as surgeons, specialized nursing staff, and anesthesiologists, contribute to the complexity. The literature underscores the necessity for surgeons to fit all procedures scheduled for a day within an allocated block of operating room time. For the development of a healthcare appointment app tailored to surgical environments, addressing these unique challenges, and incorporating features that optimize resource coordination, offer intuitive time window selection for patients, and streamline communication among the surgical team could significantly enhance the efficiency of surgical appointment scheduling.

4.2.4. Conclusion

Access delays, both indirect (virtual) and direct (captive), are inherent in appointment systems, impacting patient satisfaction and operational efficiency. [1] emphasizes the importance of well-designed systems that minimize direct waiting times for unscheduled cases without compromising scheduled appointments or resource utilization. The literature underscores the significance of access rules as fundamental components in achieving this delicate balance.

The challenges identified in the literature highlight the potential areas of improvement that an innovative healthcare appointment app could address. A successful app should consider the diverse needs of patients, the variability in service times, and the nuances of different healthcare environments. Leveraging technology to streamline appointment scheduling, minimize waiting times, and enhance communication between patients and providers can contribute to an improved healthcare experience. Additionally, the integration of advanced features, such as real-time updates, personalized preferences, and intelligent scheduling algorithms, could further optimize the efficiency and effectiveness of healthcare appointment systems. These insights underscore the multifaceted nature of healthcare appointment scheduling, emphasizing the need for tailored solutions that can address the nuanced challenges specific to primary care, specialty care, and surgical environments. The integration of advanced technologies and analytics, coupled with a deep understanding of

healthcare workflows, can pave the way for innovative and efficient appointment scheduling solutions.

4.3. Methodological approach

In conducting the literature review for this thesis, a systematic search strategy was employed to identify relevant papers from reputable sources, including the IEEE database and Google Scholar. The search was conducted using keywords such as Django architectural patterns, development best practices, software engineering best practices, Dart, Flutter architectural patterns, state management, user experience (UX), user interface (UI), and accessibility in software. The inclusion criteria were defined to encompass papers within reasonable recency, despite making exceptions for fundamental for the field works. A total of 120 papers were initially identified, with 56 meeting the inclusion criteria.

The data collection process involved a thorough reading of each selected paper, and key information, including theoretical frameworks, methodologies, and major findings, was systematically documented. The synthesis of data involved categorizing, identifying common themes, and noting any variations or contradictions in the literature.

Quality assessment was conducted by considering factors such as peer-reviewed status and the rigor of the methodologies employed in each paper. It is important to note that the literature review process was iterative, with multiple rounds of refinement to ensure the inclusion of the most relevant and high-quality sources.

While this approach provides a comprehensive overview of the existing knowledge on, it is essential to acknowledge potential limitations, such as the inherent biases in the selected literature and the evolving nature of the field.

4.4. Problem Statement

Nowadays, there are numerous challenges in the healthcare system. For instance, a simple medical test could take several months, as some clinics register patients on a long waiting list for even urgent cases. Besides, it is crucial that patients are notified about the schedules to not miss the appointments. Many people forget about the exact time and day, and this is miserable for both ends. On the other hand, appointments lack updated events so patients shall not be aware of cancellations [1].

Normally it is rational to select doctors based on the services and the quality they offer. Surprisingly, most people have a different attitude toward this subject. As mentioned by

Salisbury in [32] there are fundamental issues in the way that people choose their doctors. Some individuals decide based on the distance from their house and it is desirable to come up with the nearest clinic or doctor. Many others see the doctors that their family members visit. The lack of temptation among patients is not because of doctors' shortage but it is because they do not have enough information about the doctors. The doctor's behavior is not the data people can easily collect [32].

4.5. Conclusion

The comprehensive exploration of modern software engineering, project planning methodologies, and organizational structures, alongside an in-depth investigation into healthcare appointment scheduling, has laid a robust foundation for the development of the Avicenna application. This chapter synthesizes critical insights derived from the extensive literature review.

The literature review has established the significance of Agile methodologies, emphasizing their adaptability in the dynamic software development landscape. The theoretical framework, centered around Agile principles, has provided a guiding structure for the Avicenna project. It underscores the importance of iterative requirements adaptation and the use of specific requirement descriptions such as user stories to enhance project flexibility.

Adhering to good software practices in Django development emerges as a crucial aspect. The principles of maintainability, scalability, security, modularity, reusability, testing, documentation, and deployment, drawn from reputable sources, set the stage for the subsequent practical application in the Avicenna project. These practices collectively contribute to the creation of robust and maintainable Django applications.

Likewise, adhering to good software practices in Flutter app development embraces sound software practices. Guided by principles of scalability, maintainability, accessibility, and testability from reputable sources, these foundations lay the foundation for practical implementation in the Avicenna project. Collectively, these practices play a crucial role in crafting robust and easily maintainable Flutter applications.

The complexities of healthcare appointment scheduling, spanning primary care, specialty clinics, and surgical environments, have been thoroughly explored. The literature emphasizes the multifaceted nature of scheduling challenges and the need for flexibility, responsiveness, and efficiency in appointment systems. The Avicenna project is guided to address the nuanced requirements of primary care, specialty care, and surgical scheduling.

Flexibility and adaptability in appointment systems take center stage, acknowledging the dynamic nature of healthcare.

In conclusion, the literature review has served as a robust guide for the Avicenna project. The integration of software engineering principles, good practices in Django development, and insights from healthcare appointment scheduling literature positions Avicenna to be a responsive, adaptable, and efficient scheduling application. Future directions involve the practical application of the identified best practices, ongoing refinement based on emerging research, and the continuous evolution of the Avicenna project to meet the dynamic demands of healthcare scheduling.

5. Development of the Application

This chapter delves into the practical implementation of the Avicenna application, focusing on the development of both the front-end and the back-end. Additionally, it addresses the containerization of the back-end using Docker and the deployment process. The aim is to provide a detailed understanding of the technical aspects involved in the making of a user-friendly, secure, and scalable appointment scheduling system drawing on the analysis derived from the extensive literature review.

The front-end, represented by the user interface but, in principle, not limited to it, of Avicenna is constructed using Flutter, a versatile framework suitable for developing cross-platform applications. Several practical considerations, such as design principles, project architecture, and state management, which guided the creation of the Avicenna front-end are laid out in this chapter. As noted, the front-end extends beyond the user interface, encompassing logic associated with data presentation, management of mobile's local resources, and handling platform-specific APIs.

The Avicenna back-end implemented in Django, a Python web framework known for its robustness and scalability is supplemented by Django Rest Framework (DRF) enabling the exposure of a RESTful API. Architectural decisions, including data models and authentication mechanisms implemented using Django and DRF are laid out as well.

What is more, as Avicenna transitioned from development to a production-ready state, containerization became pivotal. Docker came to the rescue as a tool to encapsulate the Django back-end and its dependencies into a container, ensuring a heightened level of security and portability, as well as facilitating deployment. The chapter explains the dockerization process among other things too.

Beyond code development, this chapter explores usability testing methodologies and iterative refinement processes. Systematic testing and user feedback loops were employed to continuously enhance Avicenna, aligning it with user expectations and ensuring an intuitive and efficient experience.

In the subsequent subchapters each of the aspect points mentioned above are analyzed in greater detail, thus providing further insights into the design, deployment, and iterative refinement process considerations, with the goal being to deliver a pragmatic and effective solution.

5.1. Requirements

As, broadly speaking, the Avicenna application is designed to seamlessly connect doctors and patients for efficient appointment scheduling and management, this section outlines the detailed requirements that shape the functionality and user experience of the application in more detail.

- **User Types:**
  - **All users:** should have the ability to create accounts, log in, access personalized profiles, edit and delete them.
  - **Doctor:** should have specialized access allowing them to manage their schedules and appointment slots. This access should also provide them with the option to accept or reject meeting proposals.
  - **Patient:** should have the ability to look for doctors and make appointments based on the available time slots. Additionally, they should have the capability to leave anonymous reviews about their experiences with a doctor, but only after making an appointment with them.
- **User Authentication:** users, both doctors and patients, should have the capability to create accounts securely, login, and access their individual profiles. On the other hand, the chances of somebody else's obtaining access to private data must be appropriately minimized.
- **Appointment Booking and Schedule Management:** patients and doctors, respectively, should be able to see a straightforward selection of available time slots and efficiently manage them, which entails the ability to create time slots for doctors.
- **Appointment Cancellation and Rescheduling:** users should be able to cancel appointments in case of scheduling conflicts or unforeseen circumstances.
- **Calendar integration:** The app does not impose on users to adopt a new calendar inside the system but rather integrates with the inbuilt calendar app or the default calendar that the user utilizes. Therefore, it results in high consciousness about the latest updates regarding the schedule whether it is a cancellation or time shift and delays. In addition, users can be informed about the navigation to the clinic if the feature is activated on their devices. In short, users, particularly patients, should be able to receive calendar notifications about upcoming appointments to minimize the risk of missed appointments.

- **Integration with Clinic Management Systems:** at a minimum, the application should be ready for seamless integration with clinic management systems, for example by means of containerizing.
- **User Reviews:** patients should have the ability to leave reviews and ratings, contributing to a feedback system that aids other users in making informed decisions. In contrast, each doctor's profile should prominently display its average rating.
- **Localization:** at the very least the app must support both English and German. Additionally, having a stable framework in place which would allow adding more tongues is considered a positive, but ultimately optional, extra. Without any complicated user settings, the app shall automatically identify the user's region preferences and accommodate them accordingly.

5.2. Project management, agile, scrum

GitHub offers a wide variety of collaboration platforms enabling the team to ideate and implement based on the agile criteria. More details are discussed below.

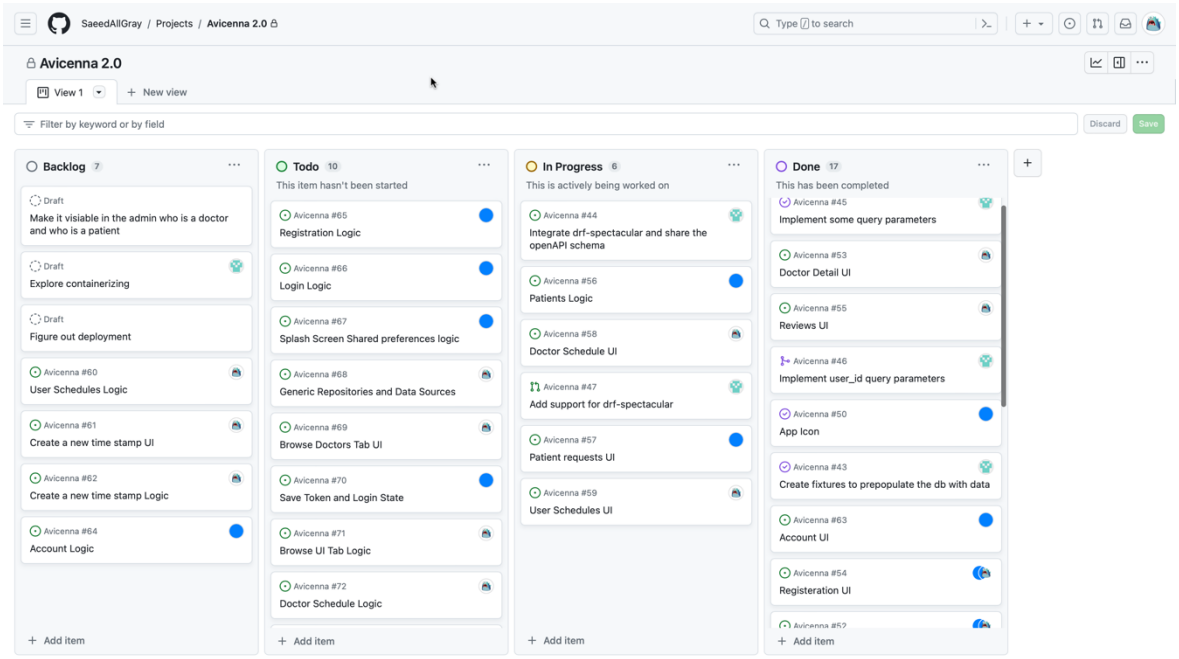


Figure 1 — State of Avicenna Project as of December 31st, 2023

5.2.1. Scrum

Although the team was not large enough to consist of a product owner, a scrum master, and the scrum team, it tried to follow the same flow of sprint sessions and take advantage of its characteristics. Sprint is rather a brief period in which a scrum team achieves one single goal [33]. In Avicenna, one week is agreed to be a sprint period. Based on the requirements, the sprint backlog is written [33]. The diagram below represents a sprint session model.

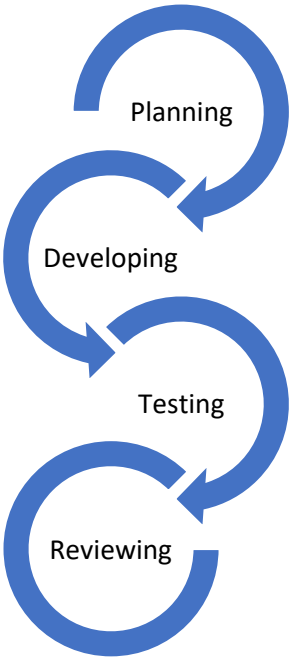


Figure 2 — Sprint Model

5.2.2. Epics

Epics denote large user stories that can consist of multiple user stories. Epics are usually the general idea of a feature, which leads to the definition of the first user stories [34]. Now, we shall establish the epics for the Avicenna App.

- Users consisting of doctors and patients can log into the App.
- Doctors can open timeslots.
- Patients can make appointments.
- Doctors can view and approve the appointments.
- Doctors can accept or decline requests for pending appointments.
- Users can add the appointments to their calendars.
- Users can postpone or cancel the appointment.
- Patients can search for doctors and specializations.
- Patients can rate the doctor and leave reviews.
- Users can delete their accounts.

5.2.3. User stories

User stories serve as a specific type of requirement description. User stories can be established in different linguistic structures, one of which is referred to as the "Canonical Form." This form comprises three essential elements designed to address three fundamental inquiries:

1. Who is utilizing the system?
2. What they expect from the system.
3. Why they seek this functionality from the system.

Answer to these questions defines the persona, the action, and the business benefits respectively [34].

Within our system, the primary users shall encompass doctors and patients. Consequently, these two personas shall feature prominently in our user stories.

- As a doctor, I want to create an account so that I can manage my schedule and provide efficient healthcare services through the platform while maintaining a professional profile.
- As a patient, I want to sign up so that I can access the doctors and specialists.
- As a member, I want to log into the app so that I can access exclusive features, personalized information, and manage my account or services conveniently from anywhere at any time.
- As a patient, I want to look up the doctors so that I can find information about their specialties, experience, and patient reviews to make an informed decision when selecting a healthcare provider who best suits my needs and preferences for treatment.
- As a patient, I want to search for a specific specialization or name so that I can easily find and connect with a doctor who specializes in a particular field or locate a specific healthcare professional by their name to address my medical concerns accurately and efficiently. As a patient, I want to view doctors' information, so that I know the doctors.
- As a patient, I want to make an appointment with a doctor, so that I receive medical care or consultation for my health concerns and ensure timely treatment or guidance for my well-being.

- As a doctor, I want to make timeslots, so that I can efficiently manage my schedule, allocate specific periods for patient appointments, and ensure that I have dedicated time for consultations, examinations, and treatments, maintaining an organized workflow within my practice.
- As a user, I want to cancel or postpone the appointment so that I can adjust my schedule according to unforeseen circumstances.
- As a user, I want to delete my account so that I can permanently remove my personal information, preferences, and any associated data from the platform's database.

The following user stories are established after obtaining user feedback:

- As a user, I want to be guided throughout the app so that I can get a clear understanding of the app's functionalities.
- As a user, I want to see all of my appointments as a list so that I can easily manage and organize my schedule.

5.2.4. Acceptance criteria

- The system should provide a registration form for doctors and required fields include name, contact information, professional details, and a secure password.
- Upon successful registration, the doctor should be able to log in with the created credentials.
- The system should have a patient registration form and required fields including name, contact information, and a secure password.
- After signing up, patients should be able to log in with their credentials.
- The login screen should be accessible from the app's main interface.
- Users should be able to log in using their registered credentials.
- Upon successful login, users should have access to exclusive features and personalized information.
- The system should provide search functionality for doctors.
- The information displayed should include specialties, experience, and patient reviews.
- The search results should be presented in a clear and organized manner.
- The search feature should allow patients to search by specialization or doctor's name.
- Search results should accurately match the entered criteria.

- The system should provide detailed information about the matching doctors.
- The system should display comprehensive information about each doctor.
- Information should include specialties, experience, and patient reviews.
- The system should allow patients to schedule appointments with available doctors.
- Patients should be able to select a suitable time slot from the doctor's availability.
- Confirmation details of the appointment should be provided to the patient.
- The system should provide a calendar or scheduling interface for doctors.
- Doctors should be able to create and manage time slots for patient appointments.
- The system should ensure that no overlapping appointments occur in a given time slot.
- Users should have the ability to cancel or reschedule their appointments.
- The system should notify both the doctor and the patient about any changes made to the appointment.
- There should be an option for users to permanently delete their accounts.
- Deleting the account should remove all personal information and associated data from the platform's database.
- Users should receive confirmation upon successful account deletion.

And here are the criteria added after user feedback:

- Login and sign-up text fields should warn users about illegal characters.
- When users log into the app for the first time, they should be greeted with a quick guiding tour of the app.
- The App should warn the user when they want to log out or delete their account.
- There should be different viewing options for the appointment calendar.
- The App should prompt the user with the default password manager of the phone when in the login and signup screens.

5.3. Modeling and Application Design

In pursuit of streamlining the collaborative efforts between the front-end and back-end development teams, allowing them to work independently of each other in the initial stages of the project, mermaid.js was employed to create an Entity Relationship Diagram (ERD) for the Avicenna project. This diagram, Figure 3 below, serves as a visual representation of the database structure, encapsulating the essential relationships between entities within the system.

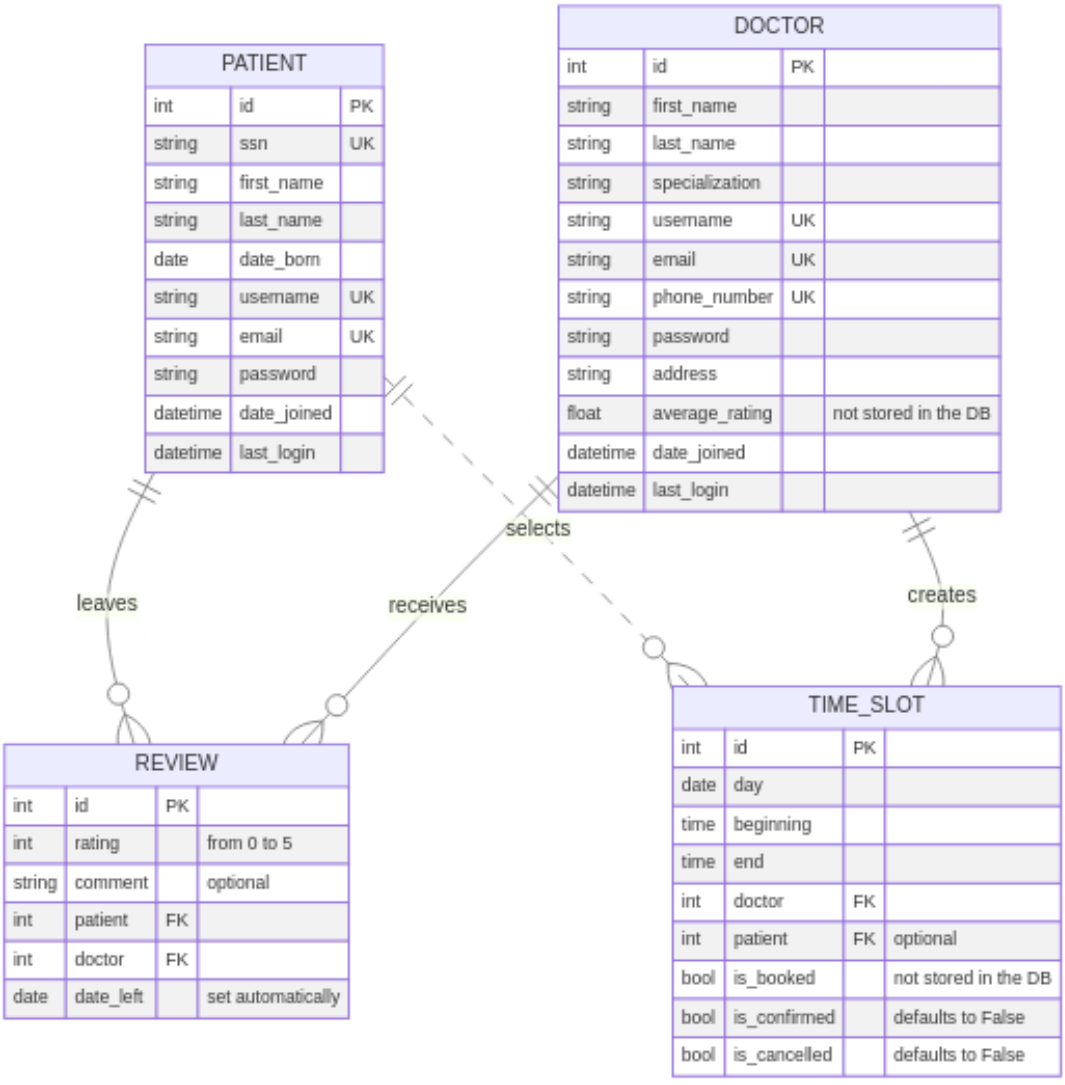


Figure 3 — entity relationship diagram

Mermaid.js, also known as Mermaid Diagramming and Charting Tool [35], distinguishes itself as a powerful and versatile tool for visualizing complex relationships and structures through code-based diagrams. What sets mermaid.js apart and is particularly useful in context of the agile development approach employed for the development of the application, is its simplicity and flexibility in generating diagrams due to its ability to do it directly from

textual descriptions. This streamlined, declarative syntax not only accelerates the diagram creation process but also facilitates easy modifications and updates. What is more, its textual representation can be uploaded to any source control solution, such as GIT, to track changes and collaborate more easily. The tool's adaptability also extends across various diagram types, from flowcharts to Gantt charts, thus making it a comprehensive solution for diverse visualization needs.

ERDs are also commonly known as entity relationship models. There exists an agreed-upon "visual" syntax and each type of "arrow" and "box" of the diagram has a specific meaning [36]. When looking at the diagram above, a perceptive reader might notice that four distinct entity types, further just called **types** as well as **abstract instances**, exist there. ERDs can be as simple as several blocks naming the abstract instances and defining some relationships between them. On the other hand, the specification also allows for the inclusion of fields inside these entities to make models less abstract, in this way bringing them closer to the real behavior of the system.

Each field represents an attribute of the instance type. The notation chosen for the diagram above makes use of at most four columns with the first two being non-negotiable. From left to right these columns are:

1. **Data type:** the type of data that the attribute represents, such as *string* or *integer*. There are no restrictions on the way they are specified, nor any provisions on their exhaustive list. When specifying the type, the maker of the diagram should refer to the established practices in the domain. Thus, well-known types should be preferred.
2. **Name:** a unique identifier within the abstract instance.
3. **Attribute Key:** an optional indicator of whether the attribute is a relationship pointing to another instance type. These are customarily limited to PK, FK, and UK, which stand for Primary Key, Foreign Key, and Unique Key. The PK serves as a single unique identifying attribute of each specific instance, while the FK links an instance to another one by means of pointing to the other one's PK. The UK is an attribute which can uniquely identify an instance.
4. **Comment:** an optional column designed to convey additional meaning. Within the context of Figure 3, this column was mainly employed to specify fields which are not stored in the database, but rather computed on-the-fly, as well as which fields are optional in terms of Django's data model and what their default values are if any.

Thus, from the business logic side of things, both types of users have unique identifies, which allow them to perform actions connected to authorization and further account management, as well as additional attributes that provide more necessary detail, such as the doctor's address or the patient's date of birth. After a successful treatment, a patient is invited to leave a review for the doctor. This review consists of the rating and optional comment among other fields, and links together the patient and the doctor by means of storing their PKs as FKs on the instance.

The timeslot takes a slightly different approach and does not require specifying the patient in order to be created. Instead, the lack of a patient signifies that the timeslot is not books, for which a supporting parameter '*is_booked* is created in order to make deep diving into the complexities of the back-end optimizations irrelevant to the front-end.

Furthermore, the connections between the instances signify not only the existence of a connection, but also the cardinality of such a connection. In this case, all connected models share the same cardinality, which in the Crow's foot notation [37] means the following:

- Two dashes on one of the lines' ends signify that the instance to which it points has to exist as one and only one exemplar to the opposite instance.
- A crow's foot and a zero mean that the that the instance to which it points can exist in the quantity of from zero to infinity through.

What deserves heightened attention is the connection between the patient and the timeslot. Since, as mentioned earlier, the patient is optional for a timeslot, these two instances are in what is called *non-identifying relationship*, hence the dotted line. Unlike in the case of identifying relationships, which are depicted as solid lines, both instances can, in fact, exist independently of each other.

5.4. Tools Overview

This subchapter serves as a foundation for exploring the essential tools employed in the development of the Avicenna application. Within it, the key components that constitute the technological framework supporting the project's architecture and functionality are examined. Each *further subchapter* corresponds to a critical aspect of the development process, beginning with an analysis of Flutter, the app framework, powered by Dart, followed by an exploration of the back-end capabilities offered by Django, a framework of Python. Additionally, Docker is discussed for its role in containerization, Git/GitHub for version control and collaboration management, and ChatGPT for mock data generation.

5.4.1. Flutter/Dart

Introduced by Google in 2016, Flutter is the framework that respects “write once and deploy everywhere”, meaning that it supports all dominant desktop operating systems (Windows, macOS, Linux) as well as iOS, Android, and web applications within a single code base [38]. In fact, Flutter is written in Dart, an object-oriented programming language that supports just-in-time (JIT) compiling and ahead-of-time (AOT) compiling, enabling developers to take advantage of hot reloading and consequently fast development [38]. Although Java Script was intended to be used in the framework at first, Dart was preferred because of its convenience of use, enabling the developers of Flutter to work closely with the developers of Dart. Moreover, Dart is faster than Java Script and is type-safe, preventing programmers from making catastrophic errors [39].

5.4.2. Python/Django

Python is a robust programming language known for its accessibility in education, data science, and general software development. It offers efficient high-level data structures and a straightforward approach to object-oriented programming. Python's clear syntax and dynamic typing, coupled with its interpretive nature, make it an ideal language for scripting and rapid application development across various domains on popular platforms [40], [41]. Moreover, Python provides a wide range of libraries and frameworks that greatly simplify the creation of WEB services, such as Django.

According to such sources as [42], [43], and the Django Software Foundation itself [44], Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It is designed to simplify the creation of complex, database-driven

websites by providing developers with a solid foundation and a range of inbuilt features. Something which is reminiscent of the Python's "batteries included" approach. Django follows the model-view-template (MVT) architectural pattern, which emphasizes the separation of concerns, making it easier to maintain and scale applications over time. This MVT pattern is essentially model-view-controller, albeit with a different name chosen to bind it closer to the specifics of Django. One of the key reasons for Django's popularity is its emphasis on DRY (Don't Repeat Yourself) principles, which reduces redundancy and promotes code reusability, resulting in faster development cycles. Additionally, Django's inbuilt security features, such as protection against common web vulnerabilities like SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF), make it a trusted choice for building secure web applications. Its scalability, versatility, and compatibility with various databases also contribute to its widespread adoption among developers worldwide, making Django a top choice for building everything from simple blogs to complex enterprise-level applications.

Therefore, with its extensive documentation, strong community support, a number of useful features described above, and an array of reusable components known as "apps", Django was an obvious choice to build a robust API for Avicenna.

5.4.3. Containerization/Deployment

According to [45] and [46], Docker is an open-source platform that enables developers to automate the deployment of applications inside containers. Containers are lightweight, portable, and self-sufficient environments that encapsulate application code, runtime, libraries, and dependencies. Docker uses containerization technology to package applications into standardized units, ensuring consistency across different environments, from development to production. Its widespread adoption in both small startups and large enterprises, along with support from leading cloud providers like AWS, Azure, and Google Cloud Platform, cements Docker's position as a cornerstone technology in modern software development. Therefore, owing to Docker's ability to build, ship, and run applications seamlessly across various platforms, it was decided to containerize Avicenna's API. The Dockerfile, a file containing all of the necessary commands to create a container, and instructions are available in the repository (see the following subchapter 5.4.4 Git/GitHub).

As can be seen from the Dockerfile, by default Gunicorn is used as the server runner for the code. According to [47], Gunicorn, short for "Green Unicorn", is a popular WSGI (Web Server Gateway Interface) HTTP server for running Python web applications. It is designed

to be lightweight, simple to use, and highly scalable, making it a preferred choice for deploying Django, Flask, and other Python-based web frameworks in production environments. Gunicorn acts as a middleman between the web application and the outside world, handling incoming HTTP requests, managing worker processes, and ensuring efficient communication between the application and the web server. Its ability to handle multiple concurrent connections and its compatibility with various deployment setups, not to mention the ease of set-up and use, made Gunicorn stand out as a reliable and efficient solution for serving Avicenna's REST API.

However, when deploying the API, the main criterion for the deployment environment was based on the cost. The only readily available and reputationally established solution turned out to be PythonAnywhere, which is both an online Integrated Development Environment (IDE) and a web hosting service specifically designed for Python applications, as stated by [48]. Even though it technically allows writing, running, and deploying Python code directly from the web browser without needing to set up or configure any development environments, the preferred way forward was cloning the GitHub repository within a virtual machine provided by the service and then making the API live. To the developers' team chagrin however, the service did not support containerized application, which meant that the containerized solution had to remain biding its time until the day the system would be deployed onto another provider. One other major gripe with PythonAnywhere turned out to be their lack of support for the latest Python versions that had to be alleviated by making changes to the source code, which is something containerization is capable of preventing.

5.4.4. Git/GitHub

For managing the software development efforts between two teams, back- and front-end, a monorepo approach was chosen. In this way the repository contained both parts of the Avicenna application separated by directories. This approach was preferred as it allowed for closer collaboration on multiple issues concerning the whole system.

In its turn, Git was chosen as the distributed version control system to track changes to the codebase. With GitHub, being a web-based platform built around Git [49], chosen to host Avicenna's code. GitHub provided not only a centralized location for storing, sharing, and managing the code, but also offered project management features such as issue tracking, pull requests, code review, and project management tools, all of which made it easier to organize and guide the project.

The project’s code and diagrams can be obtained by following the link encoded in Figure 4 below.

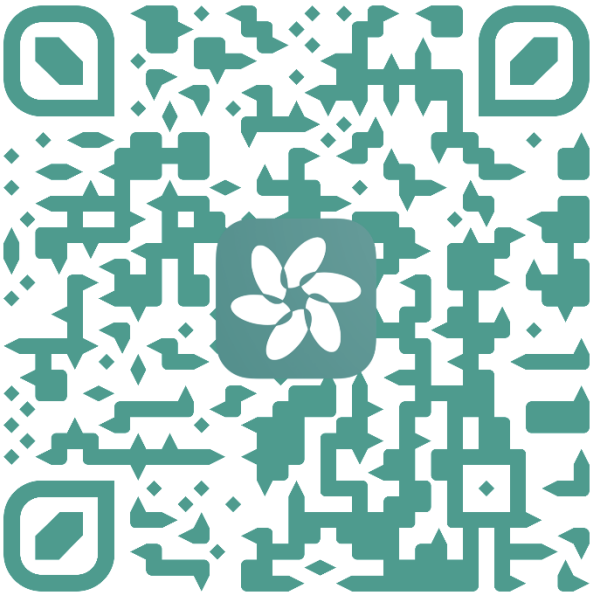


Figure 4 — QR code linking to the repository

In order to uphold principles of transparency, collaboration, and unrestricted access within the scientific community and beyond, the project's code referenced in this research thesis is released under the GNU Affero General Public License Version 3 (AGPLv3). According to [50], this license ensures that the software's source code remains accessible and modifiable by anybody who wishes to use, study, modify, and distribute it. Notably, the AGPLv3 extends the provisions of the GNU General Public License (GPL) to cover software that is accessed over a network, ensuring that modifications made to the software are also made available to users interacting with it over a network, which is especially suitable for software exposing its logic via an API, such as Avicenna.

5.4.5. ChatGPT

ChatGPT is a language model developed by OpenAI, based on the GPT (Generative Pre-trained Transformer) architecture. It has demonstrated its efficacy in generating substantial sets of mock data for the project. In the course of this project, ChatGPT was utilized as a valuable tool for the generation of database data entries.

5.4.6. Conclusion

In essence, this chapter has provided a comprehensive overview of the essential tools utilized in the development of the Avicenna application. By examining each tool in detail, insights into the technological framework that underpins the project's architecture and functionality have been gained.

The exploration began with Flutter and Dart, which enabled cross-platform development with ease and efficiency. Compared to another popular choice, JavaScript, the combination of Flutter and Dart was chosen due to its perceived superiority in terms of performance, developer convenience, and type safety.

Further, Python and Django were employed in the back-end development stack, offering a robust foundation for building a scalable, database-driven web application. No less that due to Django's adherence to the model-view-template architectural pattern, coupled with its emphasis on DRY principles and inbuilt security features, was it an ideal choice for constructing Avicenna's API.

Later, containerization with Docker emerged as a critical aspect of deployment, ensuring consistency and portability across different environments. The utilization of Gunicorn as the server runner further streamlined the deployment process, ensuring efficient handling of incoming HTTP requests and optimal performance.

Eventually, despite encountering challenges with deployment environments the project was deployed to PythonAnywhere, an online hosting solution.

Git and GitHub played indispensable roles as well by facilitating collaboration and version control and enabling seamless coordination between back-end and front-end teams. The adoption of a monorepo approach fostered closer collaboration and streamlined project management, while the decision to release the code under the AGPLv3 license underscored the project's commitment to transparency and community, including scientific, engagement.

Lastly, ChatGPT emerged as a valuable tool for generating mock data, highlighting the potential of AI-driven solutions in augmenting development workflows, and accelerating the development process.

5.5. Mobile Application

5.5.1. User Experience and User Interface

Despite facing various technical challenges across both the back end and mobile stack, our efforts were dedicated to implementing industry-standard user experiences through the

application of best practices in user interface design. An invaluable reference for user interface guidelines is Apple's Human Interface Guidelines, which we consulted extensively. This section examines key recommendations outlined by Apple and explores their implementation within our application.

- "Help people concentrate on primary tasks and content by limiting the number of onscreen controls while making secondary details and actions discoverable with minimal interaction." [51]

  - In the calendar view, we avoid overwhelming the user with excessive details. Instead, we present details to users when they tap on each calendar event tile.

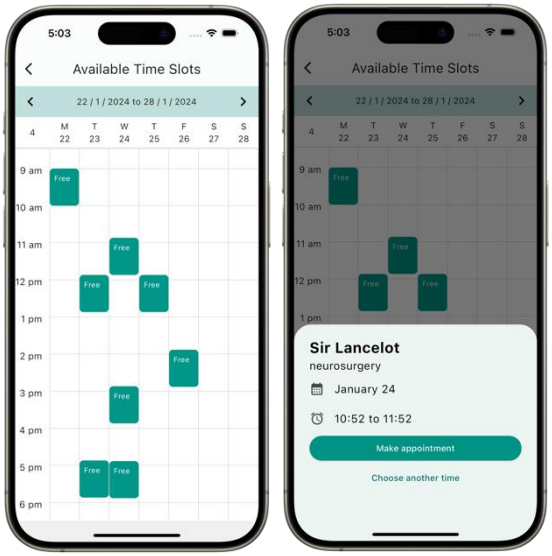


Figure 5 —

- "Support interactions that accommodate the way people usually hold their device. For example, it tends to be easier and more comfortable for people to reach a control when it’s located in the middle or bottom area of the display, so it’s especially important let people swipe to navigate back or initiate actions in a list row." [51]

  - We positioned key controls, such as the 'make appointment' button, date pickers for new time slots, and the feedback sheet in the bottom half of the screen. This placement is designed to facilitate user interactions, particularly for one-handed operation. Additionally, we incorporate swipe gestures to navigate back to the list.

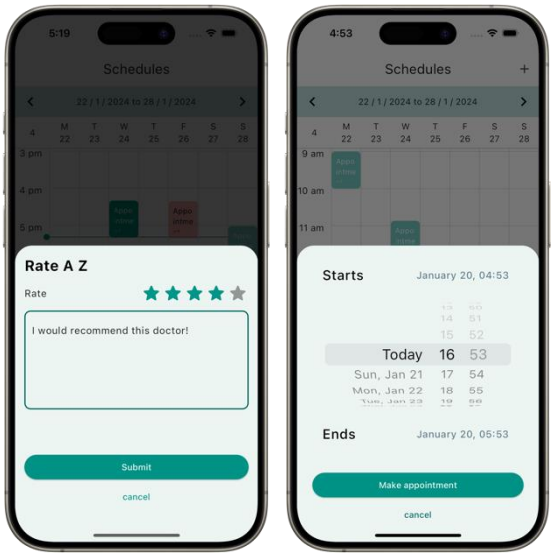


Figure 6 —

- “**Use color sparingly in nongame apps.** In a nongame app, overuse of color can make communication less clear and can distract people. Prefer using touches of color to call attention to important information or show the relationship between parts of the interface.” [52]

  - Deliberate consideration was given to color choices throughout the app to communicate various meanings and behaviors effectively. The primary actions are distinguished by the color #0C8675, secondary actions adopt #66C0B4, and #EC3A42 is reserved for destructive actions, such as account deletion or canceling an event.

- Create a concise and enjoyable user experience by designing a brief onboarding process that avoids overwhelming users with excessive information. Orient users to the main purpose of the app quickly and entertainingly, increasing the likelihood of appreciation and retention. If your app requires access to private data, initiate permission requests during onboarding to explain the necessity and benefits. Consider integrating onboarding elements into the main experience, providing brief descriptions and visual cues when activating features for the first time. [53]

  - Following testing with several participants, it became evident that the absence of an onboarding sequence was apparent. Subsequently, we incorporated a concise onboarding experience that highlights different elements of the user interface along with brief yet effective explanations about their functionalities.

- "In general, use a button that has a visible background for the most likely action in a view." [54]
- "Consider keeping the number of visually prominent buttons to one or two per view." [54]

  - We applied this practice throughout the whole app, which can be seen in the figure below.

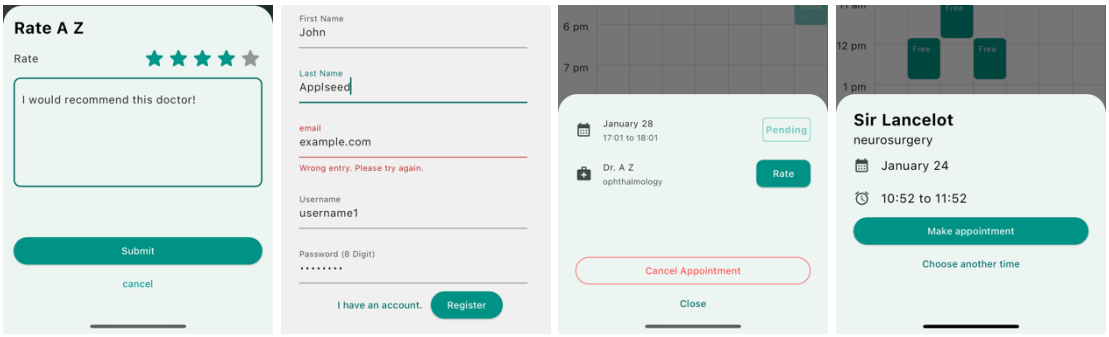


Figure 7 — Use of buttons throughout different features

5.5.2. Widgets

Flutter applications are constructed using widgets, which serve as the fundamental building blocks of the UI. These widgets are responsible for defining the structural elements, encompassing everything visible on the UI, such as texts, buttons, switches, and more. Beyond the widget layer, two additional layers come into play to customize the user interface for iOS and Android platforms.

The Cupertino layer is designed to construct iOS elements, ensuring the app's compatibility with iPhones and iPads. It tailors the user interface to seamlessly integrate with the distinctive design principles of iOS. On the other hand, the Material layer is dedicated to supporting the Material design language, aligning the app's visual and interaction patterns with the standards set by Google.

In summary, the architecture of Flutter extends beyond widget composition, incorporating specialized layers to adapt the user interface to the unique characteristics of iOS and Android. This approach allows for the creation of cohesive applications that align with the design guidelines of each platform, providing a tailored and optimized user experience [39].

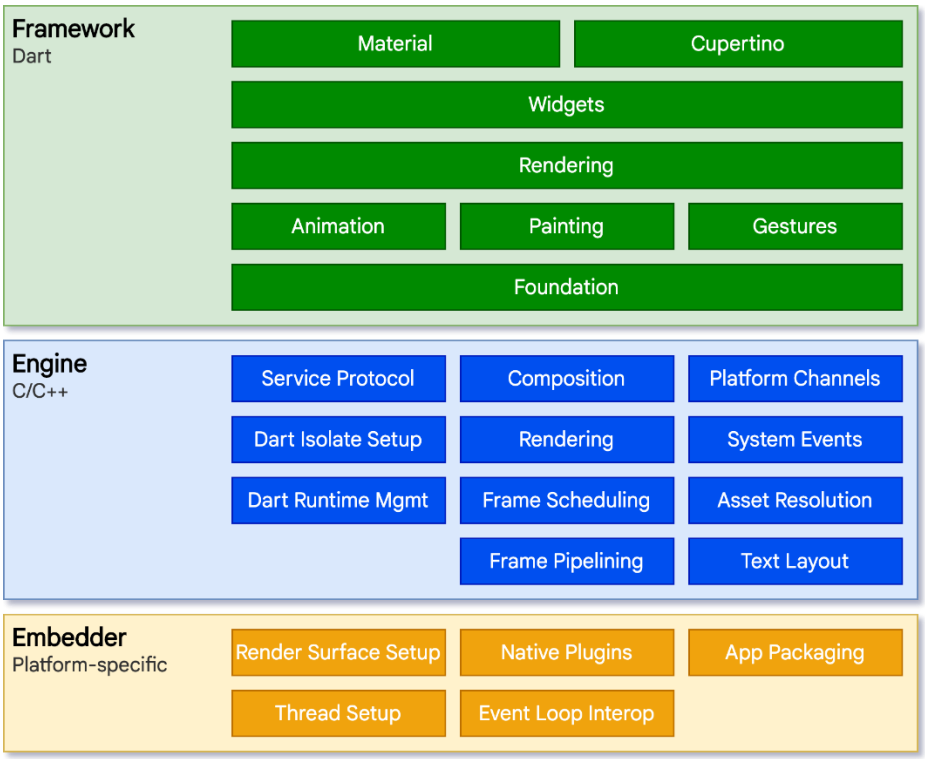


Figure 8 — Flutter Architecture – source: Adapted from [55]

5.5.3. Stateful/Stateless Widgets

In general, in Flutter, widgets can be categorized into two types: stateful and stateless. Stateless widgets represent elements of the user interface, like text and icons, whose values remain constant throughout their life cycle. On the contrary, stateful widgets, such as buttons, switches, or sliders, are designed to undergo changes in their values during their life cycle. Stateful widgets have more methods inside for state management. “setState” method is the primary approach for state management, enabling developers to adjust the value inside the widget. Nonetheless, it is highly recommended to avoid this for state management and use other solutions that is discussed later in this paper [56].

5.5.4. App Architecture

This chapter focuses on the technical approaches employed in the development of the mobile application. Since the beginning, our objective has been to employ domain-driven design principles while crafting an architecture that is scalable, maintainable, testable, and easily comprehensible. In pursuit of these objectives, we encountered an architecture that integrates layered and feature-oriented patterns. First, we need to describe them briefly.

Layered architecture aims to delegate responsibilities to hierarchal layers. The layers are as follows:

- **Data layer:** Takes care of communicating with external sources, which could include remote data sources (i.e. API) or local data sources (i.e. local databases, or user preferences)
- **Repository layer:** Serves as the medium between the data layer and business logic layer. This layer includes repositories for each domain model.
- **Business logic layer:** Includes the business logic surrounding use cases.
- **Presentation layer:** Serves as the medium between the user and the application and includes user interface elements and sends interaction events to the business logic layer while reacting to state changes[57].

Feature-oriented architecture introduces three features that segment the application based on three features:

- **Infrastructure feature:** Communicates with external data sources.
- **Domain feature:** Applies domain-specific business rules to fetched data.
- **Application feature:** Consists of either the business logic layer or the presentation layer[57].

The combined architecture in our application is illustrated in the Figure below.

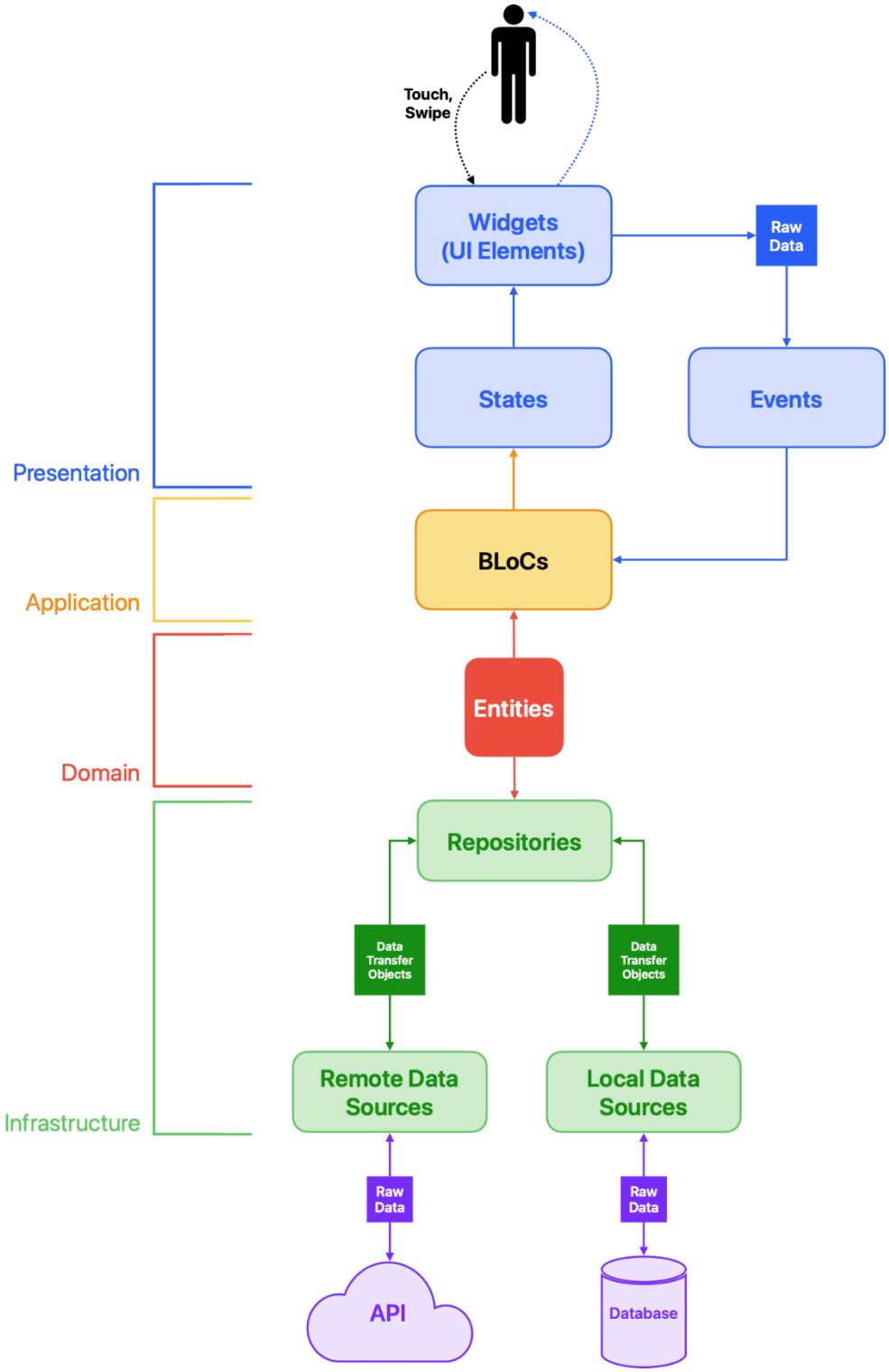


Figure 9 — Mobile app architecture

5.5.5. State Management

As previously highlighted, Flutter's inbuilt state management solution may prove inadequate for large-scale projects. Consequently, various external libraries have emerged to address

this limitation. Since the project requires a straightforward way to separate the presentation layer from business logic, making it reusable and testable, the BLoC library is preferred over other packages. BLoC is also embraced by Google developers [56]. As of now, the latest version of the bloc (^8.0.0) is being used and discussed in this paper. To integrate the Flutter widget into the BLoC, the latest version of the flutter_bloc is added to the project as well (^8.0.0).

BLoC is based on asynchronous programming, featuring two data types called Future and Stream. A Future is a data type that does not compile immediately, as soon as the result is ready, the compiler lets you know about it. Based on the official documentation a Stream is a sequence of asynchronous events, it lets you know when there is an event.

Each Bloc class consists of three main components:

- BLoC (Business Logic Component): This is the core component responsible for managing the state of the application. It contains the business logic and is often responsible for processing events and emitting new states.
- Events: Events are user actions or other occurrences that trigger a change in the application state. Events are dispatched to the bloc, which processes them and produces a new state. They can be considered as inputs.
- States: The state represents the current condition of the application. It is immutable and can only be changed by emitting a new state from the bloc in response to an event. Widgets in the UI listen to changes in the state and rebuild themselves accordingly. They are the outputs from the Bloc core [58].

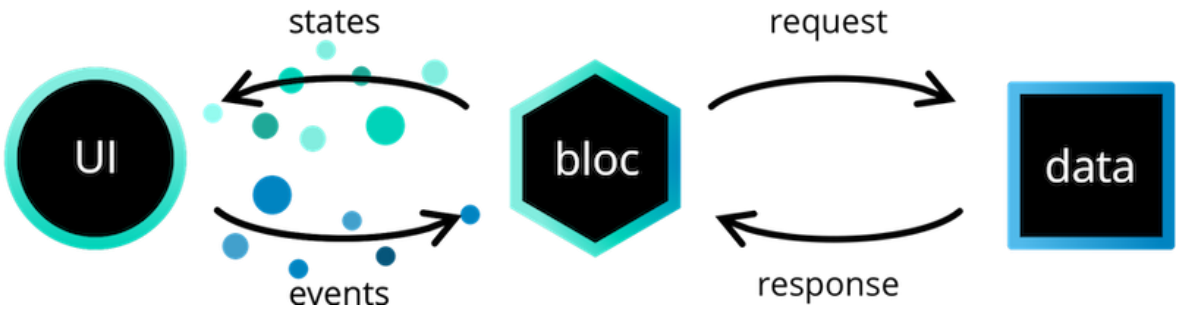


Figure 10 — BLoC Architecture – Source: Adapted from [59]

5.5.6. Cross-platform

The main reason why Flutter was chosen as the primary framework was its excellent support for a wide variety of devices. In order to cater to the diverse range of operating systems, a framework must be capable of accommodating different platforms. Fortunately, Flutter

excels in this aspect as it can be compiled to major mobile operating systems like Android and iOS. Additionally, it extends its compatibility to desktop platforms such as Linux, macOS, and Windows.

One of the outstanding features of Flutter is its ability to support web applications as well. While it may appear straightforward, this feature has proven to be a time-saving for the development team. Without this capability, the front-end department might have faced the challenge of splitting into three separate teams, each dedicated to mobile development, desktop development, and web development, respectively. The versatility of Flutter, therefore, simplifies the development process and streamlines the workflow, making it an optimal choice for projects targeting a broad spectrum of devices and platforms.

5.5.7. Conclusion

Our approach to user experience and user interface design, guided by industry best practices and Apple's Human Interface Guidelines, has resulted in a thoughtfully crafted application that prioritizes user engagement and usability. By implementing principles such as limiting onscreen controls, strategic placement of key features, and careful selection of color, we aimed to enhance the overall user experience and ensure that users can easily navigate and interact with the application.

The discussion on Flutter architecture emphasizes the importance of adapting to the unique characteristics of iOS and Android platforms. The integration of Cupertino and Material libraries allows for a cohesive user interface that aligns with the design language of iOS and Android, which contributes to an optimized experience for users on different devices.

The stateful/stateless widgets and the application's architecture discuss the considerations that ensure scalability, maintainability, and testability. Our layered architecture, consisting of data, repository, business logic, and presentation layers, as well as the feature-oriented architecture with infrastructure, domain, and application features, provides a comprehensive structure for efficient and extendable development and management of the mobile application.

Moreover, adopting BLoC architecture for state management addresses the challenges of large-scale projects. Leveraging asynchronous programming and separating business logic from presentation, the BLoC architecture enhances reusability and testability, aligning with our goal of creating a robust and adaptable application.

Lastly, the Flutter framework's seamless compilation of various operating systems, including mobile, desktop, and web platforms, significantly simplifies our development workflow.

This versatility not only saves time but also ensures a consistent and high-quality experience across a wide spectrum of devices, making Flutter an optimal choice for our project.

5.6. Back-end

In order to provide a clear and concise overview of the back-end architecture and implementation details this chapter delves into the core aspects of back-end development for the Avicenna application by providing a detailed examination of the foundational components and design considerations that underpin the functionality and data management aspects of the system.

Throughout this chapter, the essential elements of back-end development are explored, including overall design considerations, data modeling, and main logic in the form of analysis of the database design, and the implementation of a RESTful API. Both subsections also offer insights into the methodologies, principles, and best practices employed in the development process.

5.6.1. Database Design

Django automatically creates all of the necessary tables for such tasks as user permissions, management, and authentication. Below you can see the tables created manually from the models discussed above.

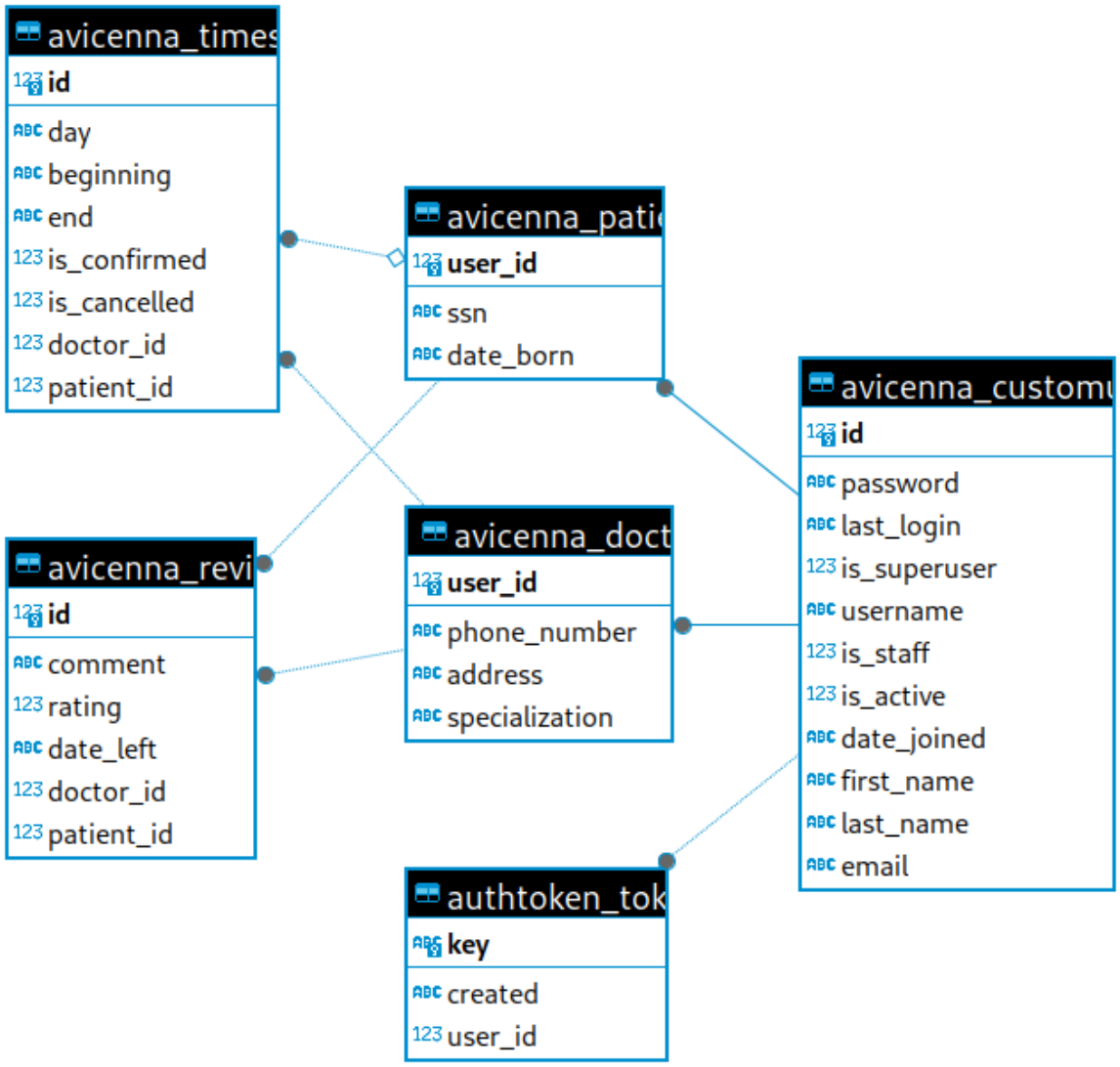


Figure 11 — diagram of the database

Compared to the — entity relationship diagram, the PATIENT and DOCTOR entities are represented by three tables (further reference to the table names ignores the "avicenna-" prefix, which denotes the app's space withing Django's object-relational mapping layer). Also note the absence of those attributes which are marked as "not stored in the DB". The tables are as follows:

- **customuser** — contains the attributes which are common to both patients and doctors, such as having a name, or a password.
- **doctor** and **patient** — contain their respective unique attributes.
- **review** — stores the data associated with feedback.
- **timeslot** — responsible for both the booking and scheduling option.
- **token** — a table storing API tokens uniquely assigned to users to perform any sort of action within the system.

One of the peculiarities of the design implementation is how defining the list of allowed doctor's specializations was approached. It is known that the list is finite and rarely updated. Therefore, it was chosen as a better course of action not to create a separate table, which would lead to more complex queries and degradation of performance, but to predefine a list of choices, a feature akin to ENUMs, but somewhat reserved specifically for Django. Instead of enumerating each of the available specializations, each viable choice is supplanted by a human-readable representation even in the database itself. In other words, the cells inside the column denoting a doctor's specialization do not store the data as numbers (1, 2, 3, …), but rather as lowercase strings. See the full structure:

```
SPECIALIZATION_CHOICES = [
        ("cardiology", "Cardiology"),
        ("dermatology", "Dermatology"),
        ("endocrinology", "Endocrinology"),
        ("gastroenterology", "Gastroenterology"),
        ("neurology", "Neurology"),
        ("oncology", "Oncology"),
        ("orthopedics", "Orthopedics"),
        ("pediatrics", "Pediatrics"),
        ("psychiatry", "Psychiatry"),
        ("urology", "Urology"),
        (
            "Internal Medicine",
            [
                ("nephrology", "Nephrology"),
                ("pulmonology", "Pulmonology"),
                ("rheumatology", "Rheumatology"),
            ],
        ),
        (
            "Surgery",
            [
                ("general surgery", "General Surgery"),
                ("orthopedic surgery", "Orthopedic Surgery"),
                ("neurosurgery", "Neurosurgery"),
            ],
        ),
        (
            "Obstetrics and Gynecology",
            [
```

```
                ("obstetrics", "Obstetrics"),
                ("gynecology", "Gynecology"),
            ],
        ),
        ("ophthalmology", "Ophthalmology"),
        ("otolaryngology", "Otolaryngology"),
    ]
```

Subgroups, such as “Internal Medicine”, show up as user help when accessing the app through the admin interface and are automatically funneled into the openAPI schema.

In this way, even though modifying the specializations necessitates updating the code, compared to how rarely new specializations are created or thrown into obscurity, it was deemed a rational decision to keep the API responsive.

In order to minimize data duplication and inconsistencies, leading to reduced performance and increased database size [60], normalization has been utilized up to the third normal form. The third form was chosen due to the fact that, according to [61] and [62], achieving the third normal form is enough to ensure protection against CRUD (Create, READ, Update, DELETE) anomalies, where the CRUD actions function as the only ones that **matter** in a *real* system, with further forms reserved for special cases or mental exercises, since they often lead to a loss in data access speeds.

Therefore, the database schema has been assessed and proven to be in the third form, defined by the following constraints:

Form 0: Unique rows — uniqueness is guaranteed by Django due to automatic insertion of PKs in each model, and thus, table.

Form 1: Each table cell contains a single non-composite value, and each column has a unique name — the latter is satisfied automatically by Django, while the former by design decisions.

Form 2: No partial dependencies, that is each attribute depends on the whole PK, not a part of it — satisfied by default, due to using IDs as PKs, which are inseparable.

Form 3: No transitive dependencies, that is each attribute is fully dependent on the PK, no attributes depend on anything else — satisfied by design choices.

5.6.2. REST API Design

REST (Representational State Transfer) API (Application Programming Interface) has emerged as a ubiquitous architectural style for designing networked applications, offering a

standardized approach for creating web services that are scalable, maintainable, and interoperable. At its core, a REST API serves as an interface that enables communication and data exchange between different software systems over the internet [63].

According to [64], REST APIs are designed to be founded on a set of principles that emphasize simplicity, scalability, and resource-based interactions. Central to the REST architectural style is the concept of resources, which are identified by unique URIs (Uniform Resource Identifiers) and manipulated through a uniform set of stateless operations. These operations, often referred to as CRUD (Create, Read, Update, Delete), map to the standard HTTP methods: GET, POST, PUT, PATCH, and DELETE, respectively.

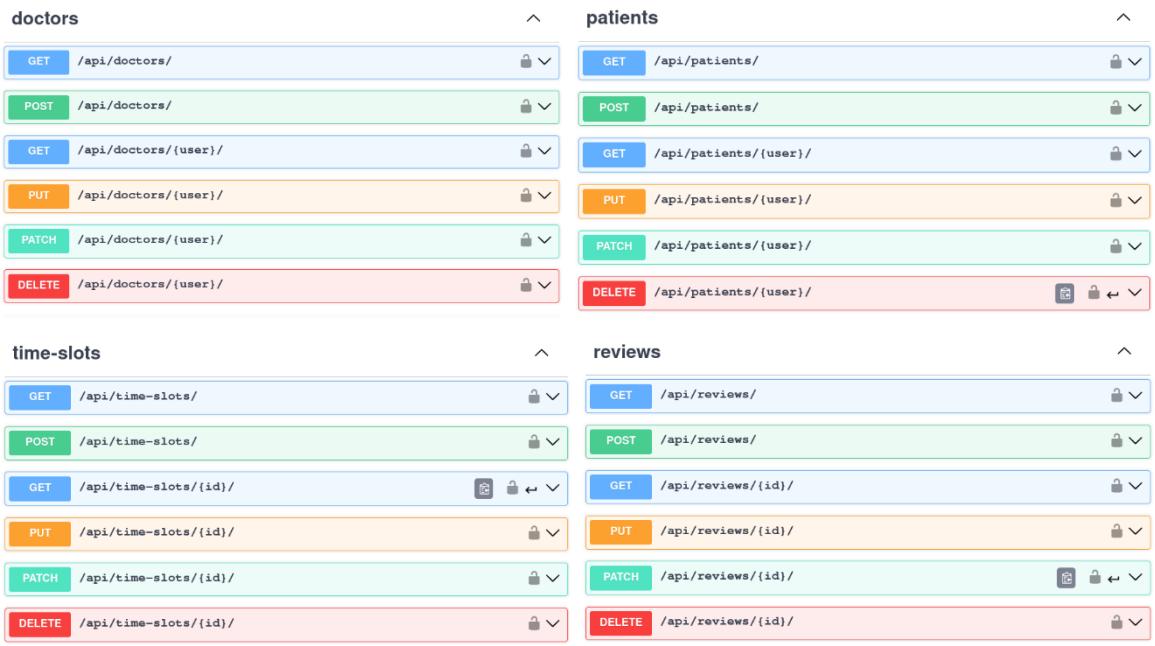
There exists a plethora of documentation on how to best design a REST API; some well-acclaimed examples are [65], [66], [67]. Hence, according to the mentioned sources, a well-designed REST API exhibits several key characteristics that contribute to its effectiveness and usability. Firstly, it adheres to the principles of statelessness, meaning each request from a client contains all the information necessary for the server to fulfill it, without relying on previous interactions. This enhances scalability and reliability, as servers can handle requests independently without maintaining client state. Secondly, a good REST API follows the principle of resource-based interactions, where resources are represented as URIs and manipulated through standardized HTTP methods. This promotes a clear and consistent interface, allowing clients to intuitively navigate and interact with the API. Moreover, a good REST API emphasizes readability and discoverability, with self-descriptive URIs and responses that convey meaningful information about the resources and their relationships. Clear and concise documentation can aid developers in understanding and effectively utilizing the API even further.

While the aforesaid principles of designing a good REST API are fundamental and all-encompassing, there do exist some more specific practices that can enhance the design and usability of the interface. This advice is applicable to both public-facing APIs, where the providers of such are not in direct communication with the consumers, and relatively stand-alone systems alike, where the API is specifically designed to service, for example, a front-end; with the latter being the case for this thesis. The recommendations are as follows:

- **Versioning:** Providing versioning mechanisms to support backward compatibility and facilitate API evolution over time.
- **Error Handling:** Implementing standardized error responses with appropriate HTTP status codes and error messages to aid in debugging and troubleshooting.

- **Authentication and Authorization:** Securing the API endpoints with authentication mechanisms such as OAuth and enforcing access controls through authorization mechanisms.
- **Pagination:** Implementing pagination mechanisms to manage large datasets and improve API performance.
- **HATEOAS (Hypermedia as the Engine of Application State):** Including hypermedia links in responses to enable clients to navigate the API dynamically and discover available resources.

Within the designed system all of the advice above has been applied, but the last one. Despite its being the recommended approach to designing RESTful APIs, as stated in [68], it turned out during the development phase that Flutter lacked adequate tools to support using URLs as identifiers for objects, as opposed to the tried-and-tested use of PKs. Therefore, it was decided that reworking the API to instead expose primary keys was the right way ahead. On the other hand, it has demonstrated that even widely used frameworks still may lack necessary tools for implementing REST APIs by the book. In Figure 7 below you may see the API endpoints with the methods and specific URLs to access them. They support and route to all of the common CRUD operations, and are all bound by permission and authentication management among other things:



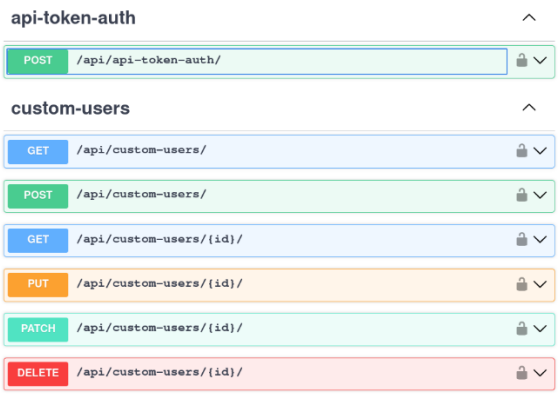


Figure 12 — API endpoints

5.6.3. Conclusion

In summary, the utilization of Django's automated table creation streamlined the initial setup process by setting up some of the essential functionalities such as user permissions, management, and authentication. While Django handled the creation of fundamental tables, specific entities like patients and doctors required manual intervention, resulting in a database structure comprising tables like 'customuser', 'doctor', 'patient', 'review', 'timeslot', and 'token', each serving distinct purposes within the system's functionality. To efficiently manage doctors' specializations, a pre-defined list of choices resembling ENUMs was employed, optimizing performance by avoiding the complexity of additional tables while ensuring human-readable representations within the database. Employing normalization up to the third normal form minimized data duplication and inconsistencies, enhancing database performance, and reducing size, thus effectively guarding against CRUD anomalies, and ensuring data integrity within the system. Overall, this approach culminated in a robust and optimized database schema, supporting the system's functionality with minimal overhead and maximal performance.

Among other things, literature analysis has demonstrated that REST API design perse encompasses principles, characteristics, and best practices aimed at creating scalable, maintainable, and interoperable web services. By adhering to REST principles and incorporating best practices, an intuitive, efficient, and resilient API was created.

During the implementation phase of the Avicenna system, all of the recommended practices for RESTful API design were followed, except for using URLs as object identifiers due to limitations in Flutter's tooling. This underscored the reality that even widely used frameworks may lack full support for all aspects of REST API design.

5.7. Conclusion

Following the best practices for developing the backend and frontend coherently, the development and deployment process becomes straightforward, and the scalability remains simple for the next iteration. Using clear architectures enabled developers to have a great view of layers and collaboration becomes simple and transparent.

The user experience and interface design of the application follow industry best practices and adhere to Apple's Human Interface Guidelines, emphasizing user engagement and usability. Key design principles, such as limiting onscreen controls and strategic feature placement, enhance the overall user experience. The Flutter architecture discussion highlights the adaptation to iOS and Android platforms through the integration of Cupertino and Material libraries, ensuring a cohesive interface across different devices. The application's scalable and maintainable architecture includes stateful/stateless widgets, layered architecture, and BLoC architecture for efficient development and management. Leveraging Flutter's cross-platform compatibility simplifies the development workflow, providing a consistent experience across mobile, desktop, and web platforms.

The initial setup of the system utilized Django's automated table creation for essential functionalities like user permissions and authentication, streamlining the process. Manual intervention was required for specific entities like patients and doctors, resulting in a database structure with tables. The use of pre-defined lists for choices optimized the management of doctors' specializations, while normalization up to the third normal form enhanced database performance and ensured data integrity. The resulting database schema is robust, optimized, and supports system functionality with minimal overhead.

The REST API design followed principles for scalability and maintainability, but limitations in Flutter's tooling prevented the use of URLs as object identifiers, highlighting framework constraints.

6. Avicenna App

6.1. Name and Identity

Avicenna (*Ibn-e-Sīnā*) is a Persian scientist who is considered one of the greatest minds in history and influenced modern medicine. He was born in Afshaneh in the northeast of Persia. He had the privilege to access the royal library as a gift from the Samanid Shah after he treated the prince. That led him to acquire a vast knowledge of medicine, philosophy, and astronomy[69]. For years Avicenna's The Canon of Medicine had been the key reference in Western civilization[70]. Thanks to his achievement for humankind the authors agreed to name the project after his name.

6.2. Visual Identity, Icon, and Color

The Avicenna app plays a vital role for patients during their treatment. Given its crucial role in the healing process, the app must adopt a color scheme aligned with the calming visual elements commonly used in healthcare environments. Hospitals strategically choose colors known for their calming effects on patients, and the Avicenna app should mirror this thoughtful approach by incorporating hues that promote a peaceful and reassuring atmosphere. Research indicates that the use of blue-green colors has a positive impact in reducing fatigue among patients [71].

With all the considerations mentioned above, Persian Green is selected as the primary color.

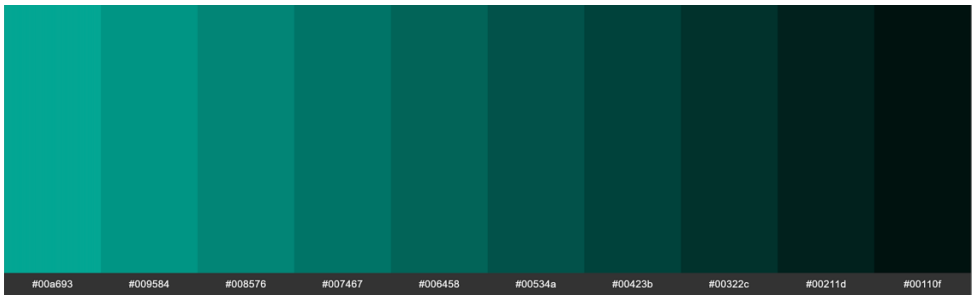


Figure 13 — Persian Green shades – Source: Adapted from [72]

When it comes to the icon for a health app, it is not about boosting sales or attracting more customers. Instead, simplicity is key, aiming for a design that exudes relaxation and tranquility. Research has shown that flowers, in general, play a crucial role in reducing blood pressure, thereby effectively lowering stress levels [73].

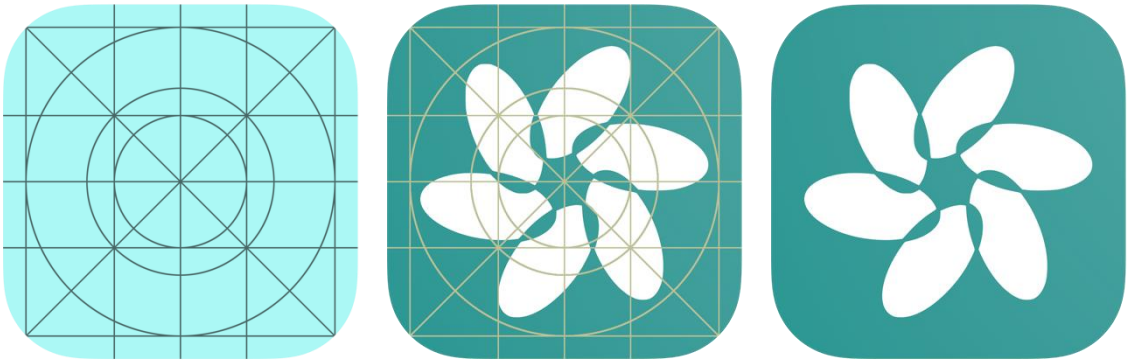


Figure 14 — App Icon, The Visual Identity

6.3. Features and Functionality

6.3.1. Splash Page

A splash page is a page that is thrown at the user upon the launch of the application. Typically, this page shortly represents the App name, in addition to the logo and in some cases, the company that developed the app. The functionality of this page is to initialize the processes that need to be done before the app starts operation, such as internet access, or availability of user, cashes, and more [74]. What makes Avicenna's splash screen unique is the platform adaptation drawing native iOS and Android screens.

Figure 15 — Splash Page

6.3.2. Authentication Page

After the splash page, the user encounters the authentication page, which presents options to log into the app or create a new account, depending on whether they have registered in the app before or if they are a new user.

Many more details have been taken into consideration to enhance the user experience. First, since doctors and patients have distinct data requirements, checking the "I am a doctor" checkbox triggers the display of

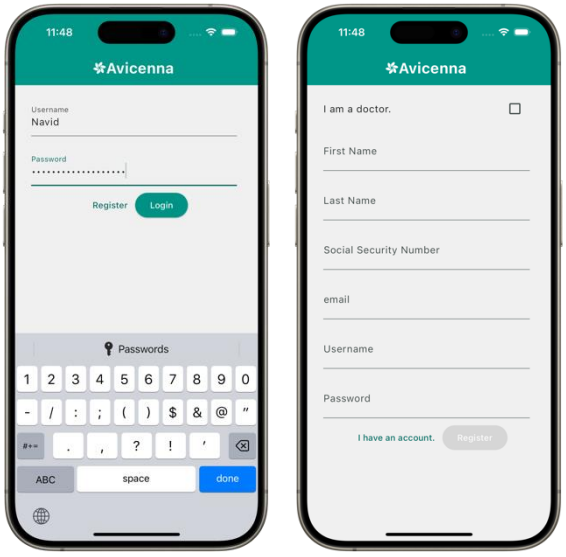


Figure 16 — Login UI (Left) and Registration UI (Right)

relevant text fields specific to that user type. As these text fields are mandatory, the register and login buttons remain disabled until the user completes the required information. Second, all the text fields adapt to the type of text field. For instance, when the user enters the email address, the keyboard shows extra buttons such as "@" and ".", while the phone number text field triggers number pad inputs. What is more is that the password text field is also obscured. Lastly, we simplified the login process for all users, meaning that they do not need to specify whether they are doctors or patients.

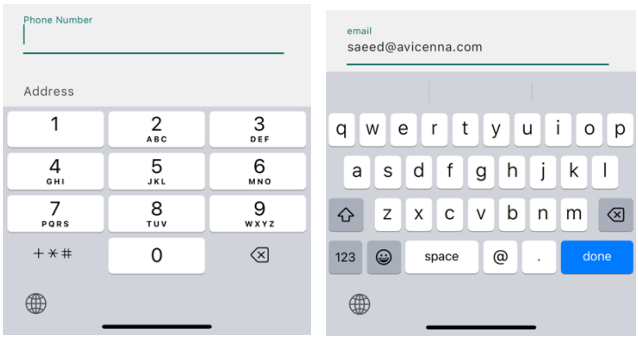


Figure 17 — The keyboard adapts to different types of input.

Upon successful authentication, the app navigates to a new page, which is the primary screen of the app called home page. Depending on whether the user is a doctor or a patient, they see different options. Doctors get three tabs which include schedules, appointments, and account tabs, while patients' home page includes browse, schedules, and account tabs.

6.3.3. Browse Tab

Browse is the first section exclusively dedicated to allowing patients to browse through all doctors registered in the database. In addition, patients can browse the doctors based on their specialization. The text field on the top is responsible for filtering doctors according to their name, last name, and profession.

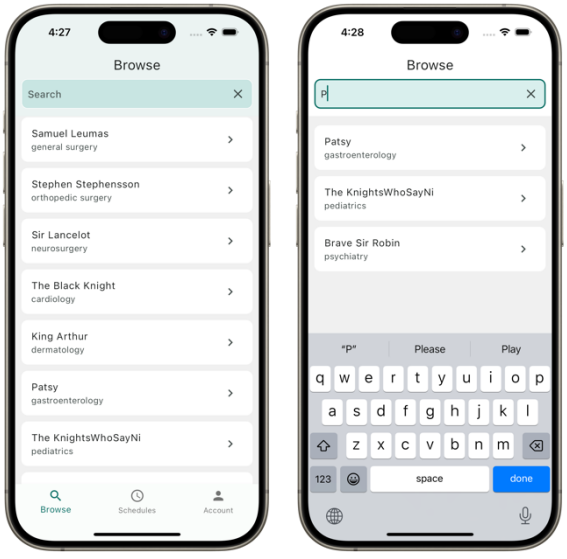


Figure 18 — Browse tab and search functionality

By tapping on each doctor tile, the user is navigated to the doctor details page where they can find any useful information about the doctor. There is a button at the center of the screen, where the user can access the time slots when the doctor is available for a visit.

Moreover, a rating system is designed to receive feedback from previous patients making it easy for users to select among the doctors in the same area. Feedback is anonymous and only visible to patients.

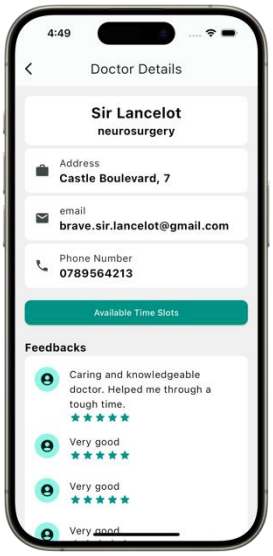


Figure 19 — Doctor Detail Screen

Figure 20 — Doctor's available time slots

This is the calendar view demonstrating the free time slots. It is designed in a way that is familiar to any smartphone user, including a simple four-direction gesture recognizer for incoming hours, days, and weeks. In fact, each page represents the weekdays in a row on the top and hours along the left side of the screen. There is also a current-day indicator to provide context to users. Teal rectangles represent the selected doctor's available time slots. The vertical dimension of the rectangle represents the duration of the time slot, a variable element influenced by the doctor's selection.

If the user taps on every time slot, a bottom sheet modal comes up on down screen enabling the user to preview the information including the doctor's name, and the time. If it is suitable for them, they can book it straight away. Otherwise, they can dismiss and choose another appointment. After booking, the user should wait for the doctor to approve the appointment.

Figure 21 — Appointment

6.3.4. Schedules Tab

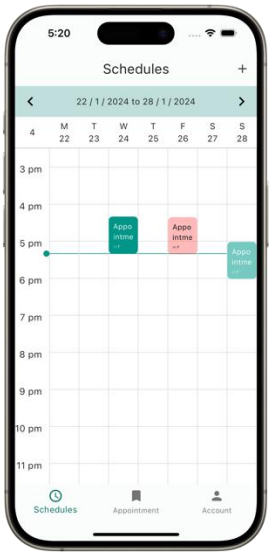


Figure 22 — Schedules tab

Schedules Tab is common among doctors and patients. However, there are slight differences for each type of user that we discuss as we go further.

As it is shown, there are three types of time slots on the calendar, highlighted with three colors:

- Pale teal: The appointments defined by the doctor but have not been booked by any user or user should wait for the doctor's approval.
- Teal: The appointments that have been made by the patient and have been approved by the doctor.
- Pale red: The appointments that are rejected or canceled by the doctor or patient.

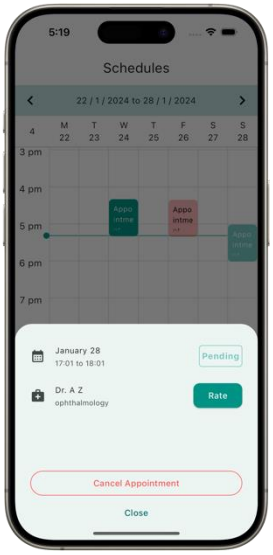


Figure 23 — Preview Appointment

After scheduling an appointment, the status can be tracked, and the doctor can be rated upon booking. If there is a change in plans, the appointment can be canceled. After the approval by the doctor, the event can be added to the calendar with a simple button tap. Rating and feedback are enabled when the appointment is successfully made. By clicking each event all the related information can be fully previewed.

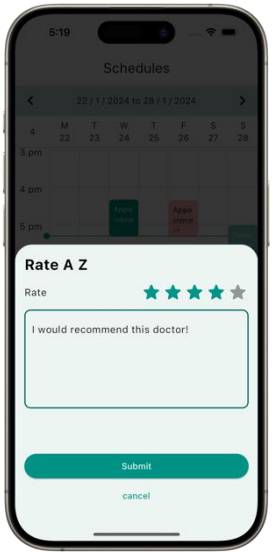


Figure 24 — Feedback

The feedback system is designed for a general assessment of each doctor. It consists of a five-star rating and a review textbox to enrich the user's opinions and points of view anonymously. It is not mandatory to submit a review and it is only possible after the booking. Reviews are not visible to doctors.

The time slots should be defined by the doctor before patients book them. There is a plus button on the top exclusively for doctors and by clicking it, a bottom sheet shows up to set the start and end time. End time is preset to one hour upon the start time by default. However, doctors are free to set their own end time.

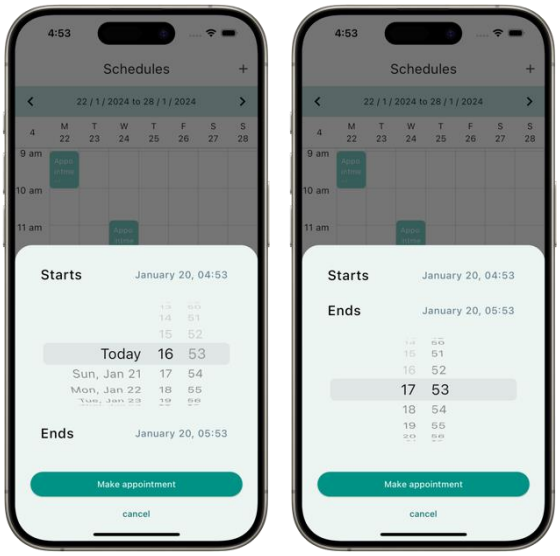


Figure 25 — Set Appointment

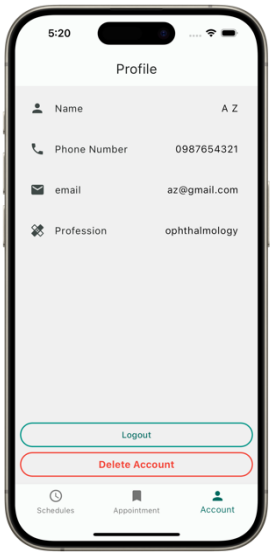


Figure 26 — Profile

6.3.5. Profile Tab

The last tab provides user information enabling users to log out or delete their accounts. When the user logs out, all the local storage on the phone is deleted permanently and they are navigated to the login page. The same thing happens if the user deletes the account, in addition to account removal from the servers.

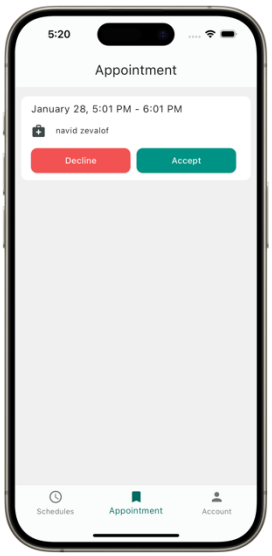


Figure 27 — Approval

6.3.6. Appointment Tab

On the middle tab, the appointment tab is located exclusively for doctors to approve the appointments booked by patients. When patients make an appointment, it needs to be approved by the respective doctor. After approval, the event shows up in the calendar view of both parties as an upcoming event. In case the doctor declines the appointment proposal, the event is displayed as a canceled appointment.

6.3.7. Calendar Integration

Avicenna communicates with the phone's inbuilt calendar to add or update events. Therefore, the user is notified about upcoming events with the minimum of fuss and the navigator suggests the routes to the hospital consequently.

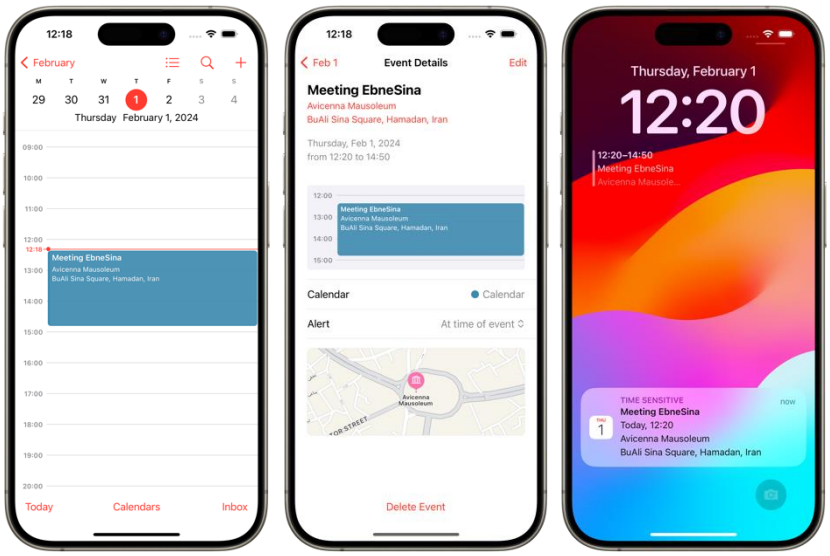


Figure 28 — iOS built-in calendar app showcasing the meeting schedule

6.3.8. Language and Localization

In the authors' point of view, adding support for more languages is one of the extraordinary features to achieve the friendliness of the user interface and ease of use. Hence, it is decided to add three more major languages including German, Persian, and Russian. These languages are carefully selected to cover all the possible app testers. There is no need for configuration

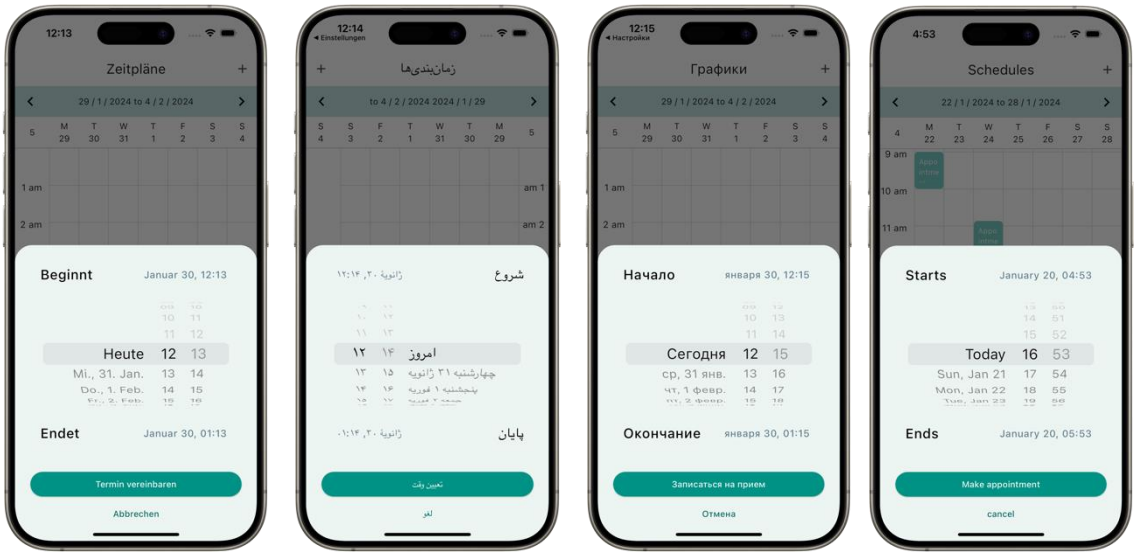


Figure 29 — From left to right: German, Persian, Russian and English

from the user since the app automatically adapts based on the language and the region that the device is set to.

Each language has its own challenges. Persian and Russian require Cyrillic and Farsi alphabets, respectively. Moreover, Persian has its own numeric Unicode, and unlike the other three languages the direction is right to left. Therefore, all the widgets and user interface elements are compatible with the right-to-left layout, thanks to flutter localization.

6.4. Accessibility

The app has been intentionally designed to ensure compatibility with accessibility features such as Android's TalkBack and iOS's VoiceOver. These features accommodate users with visibility impairments, providing audio descriptions that narrate the on-screen events for a more inclusive user experience.

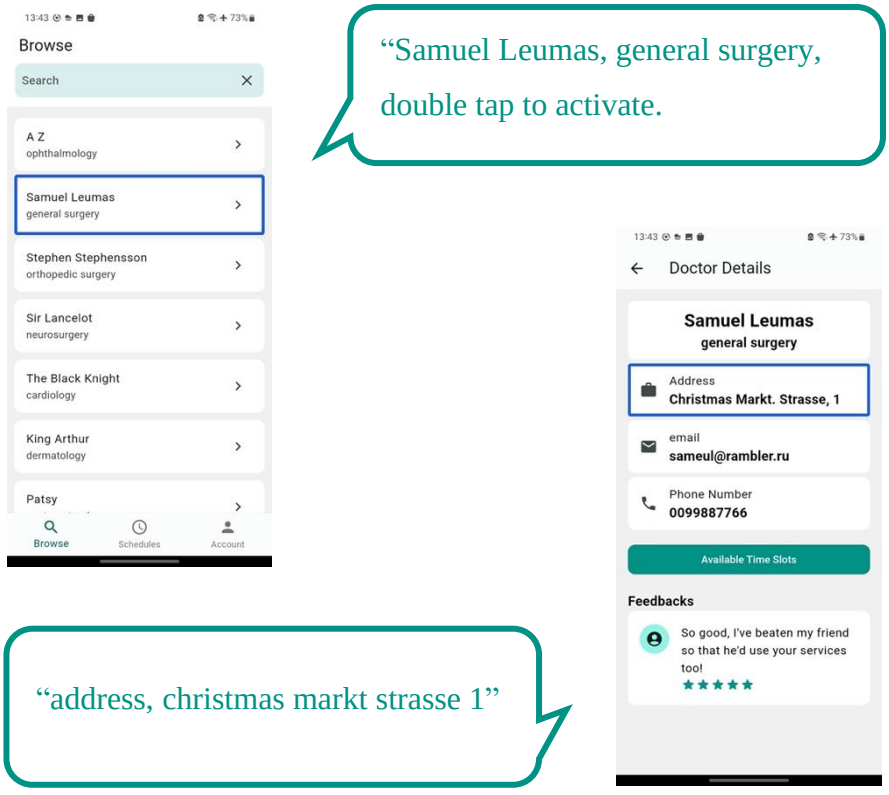


Figure 30 — TalkBack accessibility feature showcased on Android

6.5. Real-world Scenario

In the exploration of the app's functionalities, we demonstrate specific scenarios of interactions between two fictional characters – Navid Zevalov, the patient utilizing the application, and Samuel Leumans uses the app as a Doctor. Each step is also demonstrated in the figure Figure 31.

i. Navid opens the application and logs in.
ii. Navid taps on the search bar and inputs the term "general."
iii. The top result introduces him to Dr. Samuel Leumans.
iv. Tapping on Dr. Leumans' name, Navid is directed to a detailed page displaying contact details, address, and user reviews. After reviewing the information, Navid proceeds to check the doctor's available time slots.
v. On the next page, Navid views all available time slots set by the doctor and decides to request a visit for February 2nd.
vi. After confirming his choice, the app shows a message to inform the user that the appointment request has been made successfully.
vii. Navid tracks the appointment request he made on the schedules tab, where the event set for February 2nd at 11:24 is visible with a pale teal color.

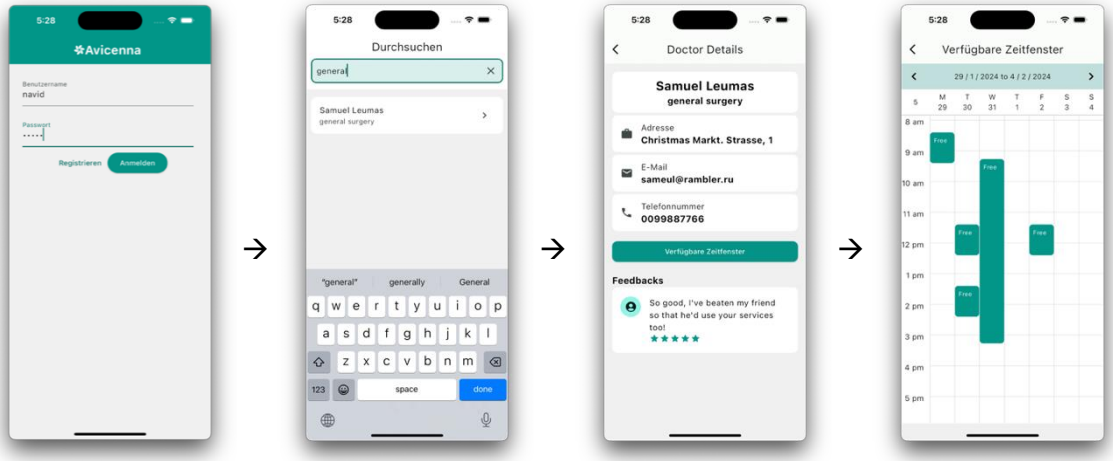
On the doctor's side, Samuel Leumans experiences the following:

i. After logging in, Samuel is prompted with the schedules page, where he can view upcoming appointments and create new free time slots.
ii. Upon navigating to the appointment tab, Samuel views Navid’s appointment request, which can be either accepted or declined. Samuel accepts the request.

The outcome is observed by both parties.

i. Navid, when checking the appointment status again, notices that the appointment request is no longer labeled as pending. The event for February 2nd is now confirmed and is no longer displayed in pale teal.
ii. Navid may also cancel the appointment or give feedback based on his experience with the doctor.

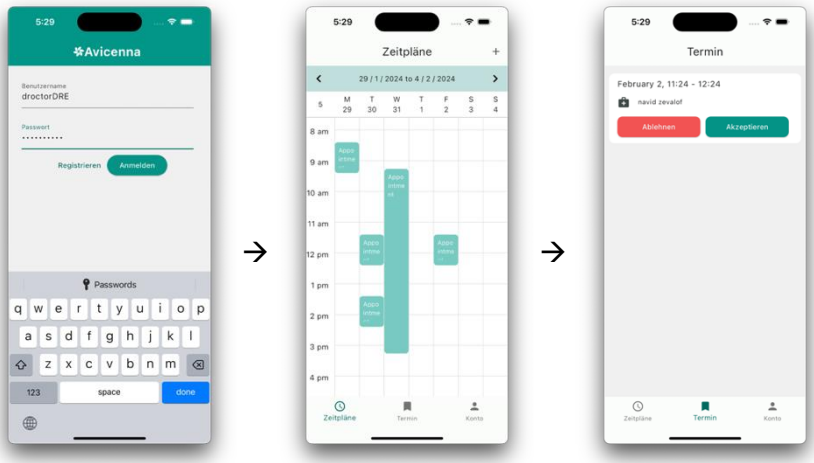
On the patient side



i ii iii iv

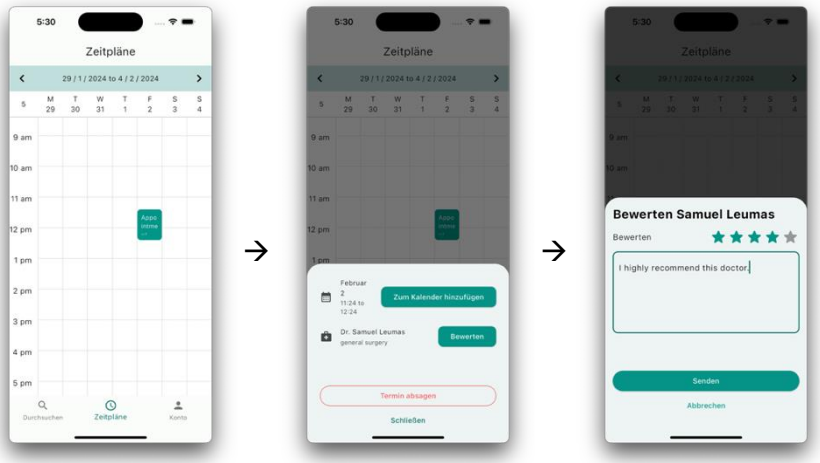
v vi vii vii

On the doctor's side



i i ii

On the patient side



i i ii

Figure 32 — Demonstration of a real-world scenario

6.6. Conclusion

When it is all set and done, the app is on a level that meets the app store's minimum requirements. Features are operating on every platform with backward compatibility, meaning that any update shall not end in app crashes and failures. By integrating with inbuilt features, the user experience reaches the highest order, and the sense of familiarity keeps the user involved.

The Avicenna app, named after the influential Persian scientist Ibn-e-Sīnā, represents a thoughtful integration of historical significance and modern healthcare principles. The visual identity, characterized by a Persian Green color and a flower-themed icon, reflects the app's focus on peace and user well-being.

The Avicenna app offers features designed for healthcare professionals and patients. Starting with a splash page featuring a unique platform-adaptive design, the app transitions seamlessly to the authentication page with user-specific fields. The browse tab allows choices for patients for the exploration of doctors based on specialization and integrates a rating system. The detailed calendar view simplifies appointment scheduling, and the schedules tab provides a comprehensive overview with a feedback system for general assessments. The profile tab demonstrates user control and privacy, while the appointment tab enables communication between doctors and patients. Calendar integration enables smooth event management, and language and localization features enhance user-friendliness by supporting four languages.

Accessibility is considered with the app designed to be compatible with accessibility features like Android's TalkBack and iOS's VoiceOver, which enables an intuitive user experience for individuals with visibility impairments.

7. Usability

Usability, a fundamental concept in Human-Computer Interaction (HCI), refers to the degree to which a product enables intended users to efficiently, effectively, and satisfactorily accomplish predetermined objectives within a defined context of usage [75].

A product designed for usability aims at three key outcomes: firstly, ensuring users become familiar and proficient with it from their initial interaction; secondly, facilitating users in accomplishing their objectives effortlessly through its use; and thirdly, enabling users to easily remember and recall the interface and its functionalities during subsequent visits or usage instances. These goals encompass the seamless integration of user-friendly design, intuitive navigation, and efficient task completion, fostering a positive and enduring user experience [76].

7.1. Definition of Objective and Goals

- **Objectives**: Evaluation of the usability of the healthcare app by testing the key user flows related to account creation, profile management, and appointment scheduling.
- **Goals**: Identifying areas of confusion, measure task completion rates, and gather feedback on the overall user experience.

7.2. Scenarios and Tasks

- **Scenario 1**: You are a neurosurgeon, and you visit patients on Mondays and Tuesdays from 9:00 AM to 12:00 AM. Every visit takes one hour.
    - **Task 1**: As a neurosurgeon, create a professional account on the platform, providing the necessary information.
    - **Task 2**: Log into the app.
    - **Task 3**: As a dentist define the schedules for your patients.
    - **Task 4**: Approve the bookings from your patients.
    - **Task 5**: Log out from the app

- **Scenario 2**: You need to see a neurosurgeon.
    - **Task 1**: As a patient, create an account on the platform, providing the necessary information.
    - **Task 2**: Log into the app as a member to explore neurosurgeons and view a doctor's information to understand their background and qualifications.
    - **Task 3**: Select the neurosurgeon based on the ratings and feedback and make an appointment.
    - **Task 4**: Write a review about the doctor and rate the dentist.
    - **Task 5**: Launch your calendar and find your appointment.
    - **Task 6**: Delete your account.

7.3. Analysis and Synthesis of Results:

After completing the usability testing sessions, the collected data was subjected to a comprehensive analysis to derive insights into the app's overall usability and user experience. As a developer, it is common that some trivial details are overlooked, and this might lead to inconvenience in user experience and ambiguity in user input. Thanks to the testers, the following points are concluded for the next app iteration.

- **User Experience:**
    - **User Input:** During the authentication process, users were not provided with hints to input data accurately into text fields. Notably, there were no validators for the email address, password, phone number, and birthdate fields. Additionally, the error message stated "Wrong Entry," lacking clarity and specificity for the user.
    - **Warnings:** There were no cautionary prompts for potentially destructive actions like "Logout" or "Delete Account," leading to the possibility of users tapping on these buttons and executing unintended actions.
    - **Password autofill:** Testers encountered difficulty with the repetitive input of usernames and passwords during sign-in attempts, expressing an expectation for the text fields to behave like other applications that utilize default password manager prompts.

- **Lack of explanation and onboarding:** Testers expressed a lack of precise understanding regarding the functionality of certain elements within the user interface.

- **Performance:**
  - **Server Overloads:** User exceptions are inevitable, especially in the authentication process. In a simple login process, if the user keeps the username field empty, there should be no request to the server because ultimately it ends in exceptions from the server. However, this was not the case for the first version. If there is a way to handle all the exceptions by employing the device intelligence instead of the backend, it can save time and money for users and developers, respectively.
  - **Device Exceptions:** For testing purposes, Samsung Galaxy A01 is used for Android. However, it appeared that the inbuilt calendar integration failed to operate well on the tester's device. The first tester had a Fairphone 5, and the calendar feature did not ask for permission.

7.4. Iteration and Modifications:

- To remedy user input issues, about ten different validators were added to prevent overload. Moreover, the user interface is clear about this and lets users know by highlighting the exact text field that causes the error.

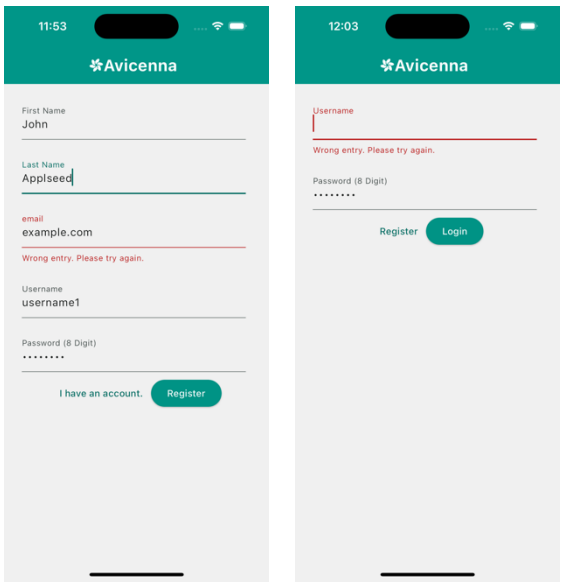


Figure 33 — In device error handling and text field validators

- On the other hand, to prevent users from deleting their accounts unintentionally the next iteration comes up with an alert dialog pop-up to ensure users intend to proceed.
- When users want to enter a username and password in the login and registration pages, the inbuilt keyboard shows an option to auto-fill the fields from the phone's default password manager.

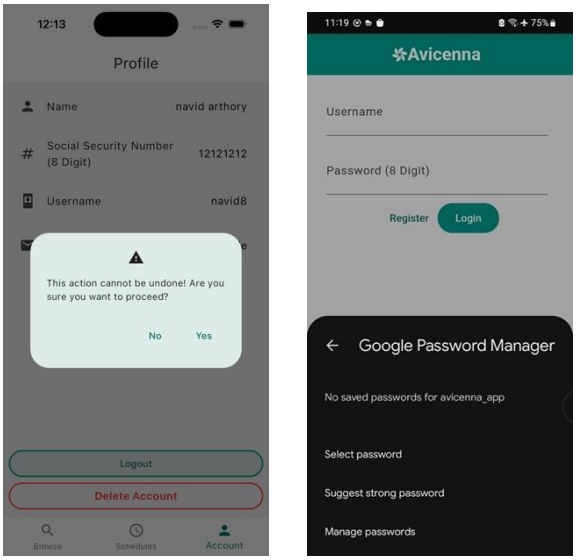


Figure 34 — Confirmation alert for a destructive action (left), autofill password (right)

- When the user installs the app for the first time, after successful registration, the app guides the user through various elements of the user interface which may not be obvious at first sight.

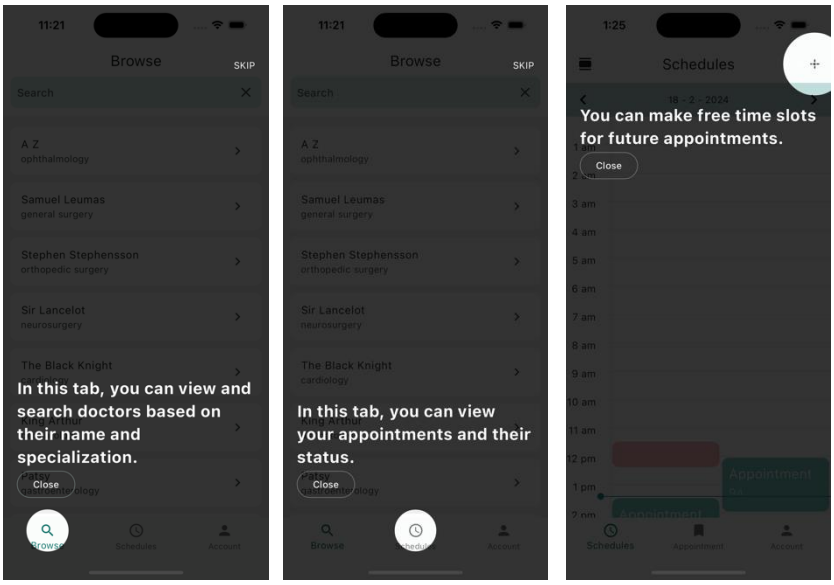


Figure 35 — App onboarding

- The application now features two distinct calendar views. The day view displays all scheduled events, allowing users to navigate between days by swiping left or right to access corresponding events.

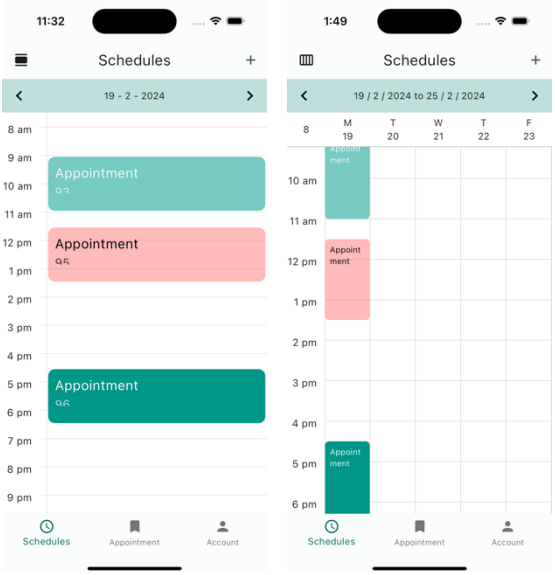


Figure 36 — Calendar day view and week view

7.5. Conclusion

In conclusion, the usability evaluation of our healthcare app played a key role in refining and enhancing the overall user experience. Our commitment to usability aimed to ensure users could efficiently, effectively, and satisfactorily achieve their objectives within the defined context of usage.

The definition of objectives established the stage for evaluation, focusing on user flows related to account creation, profile management, and appointment scheduling. The identified scenarios and tasks simulated real-world situations for healthcare professionals and patients, providing valuable insights into the application's usability across different user roles.

The analysis and synthesis of results revealed specific areas for improvement, underlining the importance of attention to user input, warnings, and overall performance. The identified issues, ranging from lack of input hints to server overloads, were addressed through a thoughtful iteration process.

The iterative improvements, such as adding validators for user input fields, cautionary prompts for potentially destructive actions, and enhancements to the authentication process, were implemented to resolve identified shortcomings. Furthermore, the introduction of onboarding elements for new users and the incorporation of distinct calendar views

demonstrated our commitment to refining the user interface and ensuring a more intuitive and familiar experience.

8. Discussion of Results

The results of this exploratory research consist, firstly, of an analysis of the available literature discussing both the theoretical side of making appointments per se, as well as the outlined specifics pertaining to the healthcare field. Secondly, the software project, consisting of the design, requirements, code, diagrams, and conducted user testing, itself is a result, too.

The requirements for the project were well met and, by applying the feedback from users, the general app experience ended with intuitive user navigation and transparency of user actions. Missing guidance within the app was addressed and ambiguity reached a minimum. Effortless device optimizations proved positive in reducing server overloads and user waiting times. The user interface follows standards and pixel-by-pixel perfection.

However, some parts of the system remained outside of the scope of the undertaking, thus they remain as milestones of potential future development.

In their turn, the results of the literature review fueled the development of the application, as in light of the analysis conducted, it became obvious that the healthcare market is in search of solutions capable of alleviation some of issues and inefficiencies experienced within the appointment management domain.

8.1. Future Directions

The following functionalities were deliberated upon by the development team. However, they were not incorporated into the final implementation due to time constraints and technical complexities beyond the scope of the project.

- **Chat Service Integration:** One noteworthy feature under consideration is the incorporation of a chat service within the application. This feature facilitates effective communication between doctors or doctor representatives and patients, which enables a seamless channel for inquiries, updates, and clarifications. The inclusion of such a communication platform has the potential to enhance patient-doctor interactions and contribute to an improved healthcare experience.
- **Push Notifications:** Another potential for future development involves the implementation of push notifications, which aim to streamline the communication process by promptly notifying doctors and patients of any alterations or updates to their scheduled appointments. By leveraging push notifications, users can stay

informed in real-time, reducing the likelihood of missed appointments and improving overall appointment management efficiency.

- **Health Data Integration with Apple's Health Kit and Google Fit:** The integration of Apple's Health Kit for iOS and Google Fit for Android emerges as a pivotal future direction. These platforms provide a set of APIs that enable developers to seamlessly integrate health and fitness data into our application. Doctors can gain valuable insights into patients' health statuses before appointments, which allows for a more understanding of their well-being.
- **Optimization for Larger Screens:** Acknowledging the evolving landscape of devices, there is an opportunity to optimize our application for larger screens, such as iPads and tablets. This adaptation aims to enhance the user experience for individuals utilizing larger devices, providing a more immersive and user-friendly interface.
- **Multi-Person Appointment Scheduling:** Recognizing the diverse needs of our users, a feature aimed at allowing individuals to make appointments for more than one person is a valuable consideration, which is beneficial for users managing appointments for family members or dependents, such as elderly parents. Enabling multi-person appointment scheduling aligns with our commitment to providing a flexible and accommodating healthcare scheduling solution.
- **Integration of a Recommendation system:** Based on the comments and ratings a system can be developed to automatically recommend doctors by analyzing the data.
- **Integration of multiple user roles:** doctors would benefit from being able to offload time-management questions to their assistants.
- **Integration with existing healthcare providers:** stress testing in a real environment and direct requirements elicitation from clinics and such is needed.

9. Conclusion and Outlook

In conclusion, while the research successfully culminated in the development of an intuitive and efficient application aimed at simplifying appointment management for doctors and patients, as well as researching and adhering to industry recommended software development practices, certain constraints prevented the full realization of the initial optional goals. Despite the application's seamless functionality in facilitating individual doctors' scheduling needs, its limitations in seamlessly integrating with larger hospital-run systems and accommodating personal assistants' involvement in schedule management are acknowledged. The decision to prioritize simplicity and usability for individual practitioners over complex integration requirements was strategic, aimed at ensuring a robust foundation for initial deployment. However, the fact that widespread adoption within larger healthcare institutions necessitates enhanced interoperability and support for multi-user management is recognized.

Moving forward, further development efforts should focus on refining the application to seamlessly integrate with existing clinic software systems and empower personal assistants to manage doctors' schedules. Additionally, scalability enhancements are imperative to accommodate the diverse needs of various healthcare settings, paving the way for broader adoption and impactful utilization of the proposed solution across the healthcare landscape. Accessibility features are expected to expand to cover all the potential users and user-centered design becomes the core of the project. Through iterative refinement and strategic collaboration with stakeholders, the development team behind Avicenna remains committed to advancing the application to meet the evolving demands of modern healthcare practices.

Apart from the purely programmatical value, the paper also offers its readers a comprehensive overview of such things as the most employed and recommended software development practices in the field, while also following them to a T, and the deep dive into both the historical and the state-of-the-art literature covering appointments per se, as well as specifics of healthcare pertaining to the project.

The code for the project, as well as accompanying it diagrams, have been released under an opensource license, therefore anybody is free to continue expanding the project either by directly contributing to it or forking it.

10. Table of Contribution

Chapter	Contributor
Abstract	Arthur, Navid, Saeed
Introduction	Arthur, Navid, Saeed
4. Literature Review	Arthur, Saeed
4.1. Theoretical framework	--
4.1.1. Project organization	Navid, Saeed
4.1.2. Industry-standard software practices in Django development	Arthur
4.1.3. Industry-standard software practices in Flutter app development	Navid, Saeed
4.1.4. Conclusion	Arthur, Navid, Saeed
4.2. Appointment scheduling in healthcare	Arthur
4.2.1. Primary Care Appointment Scheduling: Navigating Complexities for Enhanced Patient Access	Arthur, Saeed
4.2.2. Specialty Clinic Appointment Scheduling: Navigating Referrals and Variable Service Times	Arthur
4.2.3. Surgical Appointment Scheduling: Orchestrating Resources for Optimal Efficiency	Arthur
4.2.4. Conclusion	Arthur
4.3. Methodological approach	Arthur, Navid, Saeed
4.4. Problem Statement	Saeed
4.5. Conclusion	Arthur, Navid, Saeed
5. Development of the application	Arthur, Navid, Saeed
5.1. Requirements	Arthur, Navid, Saeed
5.2. Project management, agile, scrum	Saeed
5.2.1. Scrum	Saeed
5.2.2. Epics	Navid
5.2.3. User Stories	Saeed, Navid
5.2.4. Acceptance Criteria	Saeed
5.3. Modelling and application design	Arthur, Navid, Saeed
5.4. Tools overview	Arthur
5.4.1 Flutter/Dart	Saeed
5.4.2. Python/Django	Arthur
5.4.3. Containerization/Deployment	Arthur
5.4.4. Git/GitHub	Arthur, Saeed
5.4.5 ChatGPT	Navid
5.4.6. Conclusion	Arthur, Saeed, Navid
5.5. Mobile Application	Navid, Saeed
5.6. Back end	Arthur
5.7. Conclusion	Navid, Saeed, Arthur
6. Avicenna App	Navid, Saeed
7. Usability	Navid, Saeed
8. Discussion of Results	Navid, Saeed, Arthur
9. Conclusion and Outlook	Navid, Saeed, Arthur

11. Bibliography

[1] D. Gupta and B. Denton, “Appointment scheduling in health care: Challenges and opportunities,” *IIE Transactions*, vol. 40, no. 9, pp. 800–819, Jul. 2008, doi: 10.1080/07408170802165880.

[2] I. Almomani and A. AlSarheed, “Enhancing outpatient clinics management software by reducing patients’ waiting time,” *J Infect Public Health*, vol. 9, no. 6, pp. 734–743, Nov. 2016, doi: 10.1016/j.jiph.2016.09.005.

[3] A. Kuiper, J. de Mast, and M. Mandjes, “The problem of appointment scheduling in outpatient clinics: A multiple case study of clinical practice,” *Omega (United Kingdom)*, vol. 98, 2021, doi: 10.1016/j.omega.2019.102122.

[4] V. Garousi, K. Petersen, and B. Ozkan, “Challenges and best practices in industry-academia collaborations in software engineering: A systematic literature review,” *Inf Softw Technol*, vol. 79, pp. 106–127, Nov. 2016, doi: 10.1016/J.INFSOF.2016.07.006.

[5] S. Ashmore and K. Runyan, *Introduction to Agile Methods*. 2014. Accessed: Nov. 29, 2023. [Online]. Available: https://books.google.de/books?id=hE7iAwAAQBAJ&lpg=PR7&ots=FxHOLwoEqm&dq=agile%20methodologies&lr&pg=PP1#v=onepage&q=PMID&f=false

[6] V. Szalvay, “An introduction to agile software development,” *Danube Technologies*, no. June, 2004.

[7] S. Ashmore and K. Runyan, *Introduction to Agile Methods*. 2014. Accessed: Nov. 29, 2023. [Online]. Available: https://books.google.de/books?id=hE7iAwAAQBAJ&lpg=PR7&ots=FxHOLwoEqm&dq=agile%20methodologies&lr&pg=PP1#v=onepage&q=PMID&f=false

[8] M. E. Moreira, *Being Agile*, vol. 9781430258407. Berkeley, CA: Apress, 2013. doi: 10.1007/978-1-4302-5840-7.

[9] G. van Rossum and A. Coghlan, “PEP 8 – Style Guide for Python Code | peps.python.org.” Accessed: Dec. 07, 2023. [Online]. Available: https://peps.python.org/pep-0008/

[10] P. Classon, “Managing Technical Debt in Django Web Applications,” 2016.

[11] “Performance and optimization | Django documentation | Django.” Accessed: Dec. 07, 2023. [Online]. Available: https://docs.djangoproject.com/en/5.0/topics/performance/

[12] “Applications | Django documentation | Django.” Accessed: Dec. 11, 2023. [Online]. Available: https://docs.djangoproject.com/en/5.0/ref/applications/

[13] “Security in Django | Django documentation | Django.” Accessed: Dec. 11, 2023. [Online]. Available: https://docs.djangoproject.com/en/5.0/topics/security/

[14] D. R. Greenfeld and A. R. Greenfeld, *Daniel & Audrey Feldroy - Two Scoops of Django 3.x (2021, Two Scoops Press)*.

[15] “Testing in Django | Django documentation | Django.” Accessed: Dec. 11, 2023. [Online]. Available: https://docs.djangoproject.com/en/3.2/topics/testing/

[16] “How to deploy Django | Django documentation | Django.” Accessed: Dec. 11, 2023. [Online]. Available: https://docs.djangoproject.com/en/5.0/howto/deployment/

[17] W. S. Vincent, “Django for Beginners Build websites with Python & Django,” 2018. [Online]. Available: http://leanpub.com/djangoforbeginners

[18] “Effective Dart: Style.”

[19] “Effective Dart.” Accessed: Feb. 07, 2024. [Online]. Available: https://dart.dev/effective-dart

[20] “Linter Rules.” Accessed: Feb. 07, 2024. [Online]. Available: https://dart.dev/tools/linter-rules

[21] W. B. McNatt and J. M. Bieman, “Coupling of design patterns: Common practices and their benefits,” in *Proceedings - IEEE Computer Society’s International Computer Software and Applications Conference*, 2001. doi: 10.1109/cmpsac.2001.960670.

[22] “Internationalizing Flutter apps.” Accessed: Feb. 07, 2024. [Online]. Available: https://docs.flutter.dev/ui/accessibility-and-internationalization/internationalization

[23] R. W. Collins, “Software localization for internet software: Issues and methods,” *IEEE Softw*, vol. 19, no. 2, 2002, doi: 10.1109/52.991367.

[24] S. Yan and P. G. Ramachandran, “The current status of accessibility in mobile apps,” *ACM Trans Access Comput*, vol. 12, no. 1, 2019, doi: 10.1145/3300176.

[25] M. Ballantyne, A. Jha, A. Jacobsen, J. Scott Hawker, and Y. N. El-Glaly, “Study of accessibility guidelines of mobile applications,” in *ACM International Conference Proceeding Series*, 2018. doi: 10.1145/3282894.3282921.

[26] B. S. Mattu and R. Shankar, “Test driven design methodology for component-based system,” in *Proceedings of the 1st Annual 2007 IEEE Systems Conference*, 2007. doi: 10.1109/SYSTEMS.2007.374646.

[27] D. S. Janzen and H. Saiedian, “On the influence of test-driven development on software design,” in *Software Engineering Education Conference, Proceedings*, 2006. doi: 10.1109/CSEET.2006.25.

[28] A. Kuiper, M. Mandjes, J. de Mast, and R. Brokkelkamp, “A flexible and optimal approach for appointment scheduling in healthcare,” *Decision Sciences*, vol. 54, no. 1, pp. 85–100, Feb. 2023, doi: 10.1111/deci.12517.

[29] A. Ala and F. Chen, “An Appointment Scheduling Optimization Method in Healthcare with Simulation Approach,” in *2020 IEEE 7th International Conference on Industrial Engineering and Applications (ICIEA)*, IEEE, Apr. 2020, pp. 833–837. doi: 10.1109/ICIEA49774.2020.9101995.

[30] M. Murray and C. Tantau, “Redefining open access to primary care,” *Manag Care Q*, vol. 7, no. 3, p. 45—55, 1999, [Online]. Available: http://europepmc.org/abstract/MED/10620958

[31] M. Murray and C. Tantau, “Same-Day Appointments: Exploding the Access Paradigm,” *Fam Pract Manag*, vol. 7, no. 8, pp. 45–50, Sep. 2000, Accessed: Feb. 26, 2024. [Online]. Available: https://www.aafp.org/pubs/fpm/issues/2000/0900/p45.html

[32] C. J. Salisbury, “How do people choose their doctor?,” *Br Med J*, vol. 299, no. 6699, 1989, doi: 10.1136/bmj.299.6699.608.

[33] A. Srivastava, S. Bhardwaj, and S. Saraswat, “SCRUM model for agile methodology,” in *Proceeding - IEEE International Conference on Computing, Communication and Automation, ICCCA 2017*, 2017. doi: 10.1109/CCAA.2017.8229928.

[34] M. E. Moreira, *Being agile: Your roadmap to successful adoption of agile*, vol. 9781430258407. 2013. doi: 10.1007/978-1-4302-5840-7.

[35] K. Sveidqvist and Contributors to Mermaid, “Mermaid: Generate diagrams from markdown-like text.” Dec. 2014. [Online]. Available: https://github.com/mermaid-js/mermaid

[36] S. Bagui and R. Earp, “Database design using entity-relationship diagrams,” p. 357.

[37] G. Everest, “Basic data structure models explained with a common example,” *researchgate.netGC EverestProc. Fifth Texas Conference on Computing Systems*, 1976, Accessed: Jan. 29, 2024. [Online]. Available: https://www.researchgate.net/profile/Gordon-Everest-2/publication/291448084_BASIC_DATA_STRUCTURE_MODELS_EXPLAINED

_WITH_A_COMMON_EXAMPLE/links/57affb4b08ae95f9d8f1ddc4/BASIC-DATA-STRUCTURE-MODELS-EXPLAINED-WITH-A-COMMON-EXAMPLE.pdf

[38] E. Windmill, *Flutter in action*. 2020.

[39] L. Dagne, "Flutter for Cross-Platform App and SDK Development," *Metropolia University of Applied Sciences*, no. May, 2019.

[40] G. van Rossum, "Python tutorial." Jan. 01, 1995.

[41] "The Python Tutorial — Python 3.12.2 documentation." Accessed: Feb. 24, 2024. [Online]. Available: https://docs.python.org/3/tutorial/index.html

[42] W. S. Vincent, "Django for Beginners Build websites with Python & Django," 2018. [Online]. Available: http://leanpub.com/djangoforbeginners

[43] D. R. Greenfeld and A. R. Greenfeld, "Daniel & Audrey Feldroy - Two Scoops of Django 3.x (2021, Two Scoops Press)".

[44] "Django overview | Django." Accessed: Feb. 24, 2024. [Online]. Available: https://www.djangoproject.com/start/overview/

[45] "Docker overview | Docker Docs." Accessed: Feb. 24, 2024. [Online]. Available: https://docs.docker.com/get-started/overview/

[46] N. Poulton, *Docker Deep Dive*. Nielson Book Services, 2023.

[47] "Gunicorn - WSGI server — Gunicorn 21.2.0 documentation." Accessed: Feb. 24, 2024. [Online]. Available: https://docs.gunicorn.org/en/stable/

[48] "Host, run, and code Python in the cloud: PythonAnywhere." Accessed: Feb. 24, 2024. [Online]. Available: https://www.pythonanywhere.com/

[49] S. Chacon and B. Straub, "Pro Git."

[50] "GNU Affero General Public License - GNU Project - Free Software Foundation." Accessed: Feb. 29, 2024. [Online]. Available: https://www.gnu.org/licenses/agpl-3.0.en.html

[51] "Designing for iOS." Accessed: Feb. 25, 2024. [Online]. Available: https://developer.apple.com/design/human-interface-guidelines/designing-for-ios

[52] "Human interface Guidelines, Color." Accessed: Feb. 25, 2024. [Online]. Available: https://developer.apple.com/design/human-interface-guidelines/color

[53] "Human interface Guidelines, Onboarding." Accessed: Feb. 25, 2024. [Online]. Available: https://developer.apple.com/design/human-interface-guidelines/onboarding

[54] “Human interface Guidelines, Buttons.” Accessed: Feb. 25, 2024. [Online]. Available: https://developer.apple.com/design/human-interface-guidelines/buttons

[55] Google, “Flutter Architectural Layers,” https://docs.flutter.dev/resources/architectural-overview. Accessed: Feb. 01, 2024. [Online]. Available: https://docs.flutter.dev/resources/architectural-overview

[56] M. Szczepanik and M. Kedziora, “State management and software architecture approaches in cross-platform flutter applications,” in *ENASE 2020 - Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering*, 2020. doi: 10.5220/0009411604070414.

[57] Alejandro Ferrero, “Development of a Large-Scale Flutter App,” 2022. Accessed: Feb. 01, 2024. [Online]. Available: https://www.politesi.polimi.it/handle/10589/186288

[58] Dmitrii Slepnev, “STATE MANAGEMENT APPROACHES IN FLUTTER.”

[59] F. Angelov, “Bloc Library.” Accessed: Feb. 01, 2024. [Online]. Available: https://bloclibrary.dev/#/coreconcepts

[60] E. Eessaar, “The Database Normalization Theory and the Theory of Normalized Systems: Finding a Common Ground,” *Baltic J. Modern Computing*, vol. 4, no. 1, pp. 5–33, 2016, Accessed: Feb. 22, 2024. [Online]. Available: https://www.researchgate.net/publication/297731569

[61] C. J. Date, “An Introduction to Database Systems, eighth ed. Addison-Wesley Logman,” 1999.

[62] H. Lee, “Justifying database normalization: a cost/benefit model,” *Inf Process Manag*, vol. 31, no. 1, pp. 59–67, Jan. 1995, doi: 10.1016/0306-4573(95)80006-F.

[63] “2023 State of the API Report | Brought to You by Postman.” Accessed: Feb. 23, 2024. [Online]. Available: https://www.postman.com/state-of-api/

[64] R. Thomas Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” University of California, Irvine, 2000.

[65] M. Masse, “REST API Design Rulebook,” 2011, Accessed: Feb. 23, 2024. [Online]. Available: https://search.worldcat.org/title/1302275708

[66] “Web API design best practices - Azure Architecture Center | Microsoft Learn.” Accessed: Feb. 23, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design

[67] “Best Practices in API Design.” Accessed: Feb. 23, 2024. [Online]. Available: https://swagger.io/resources/articles/best-practices-in-api-design/

[68] M. Nally, “API design: Why you should use links, not keys, to represent relationships in APIs | Google Cloud Blog.” Accessed: Feb. 23, 2024. [Online]. Available: https://cloud.google.com/blog/products/application-development/api-design-why-you-should-use-links-not-keys-to-represent-relationships-in-apis

[69] A. Zargaran, A. Mehdizadeh, M. M. Zarshenas, and A. Mohagheghzadeh, “Avicenna (980-1037 AD),” *Journal of Neurology*, vol. 259, no. 2. 2012. doi: 10.1007/s00415-011-6219-2.

[70] A. Aciduman, U. Er, and D. Belen, “Peripheral nerve disorders and treatment strategies according to Avicenna in his medical treatise, Canon of medicine,” *Neurosurgery*, vol. 64, no. 1. 2009. doi: 10.1227/01.NEU.0000335779.27115.D3.

[71] M. Saito, “‘Blue and seven phenomena’ among Japanese students,” *Percept Mot Skills*, vol. 89, no. 2, 1999, doi: 10.2466/pms.1999.89.2.532.

[72] Colorswall, “Shades of Persian Green Color.” Accessed: Feb. 01, 2024. [Online]. Available: https://colorswall.com/palette/27464

[73] H. Mochizuki-Kawai, I. Matsuda, and S. Mochizuki, “Viewing a flower image provides automatic recovery effects after psychological stress,” *J Environ Psychol*, vol. 70, 2020, doi: 10.1016/j.jenvp.2020.101445.

[74] W. Jackson, “User Interface Design Interactivity: Event Handling and Imaging Effects,” in *Pro Java 9 Games Development*, 2017. doi: 10.1007/978-1-4842-0973-8_10.

[75] P. Weichbroth, “Usability of mobile applications: A systematic literature study,” *IEEE Access*, vol. 8, 2020, doi: 10.1109/ACCESS.2020.2981892.

[76] L. Punchoojit and N. Hongwarittorrn, “Usability Studies on Mobile User Interface Design Patterns: A Systematic Literature Review,” *Advances in Human-Computer Interaction*, vol. 2017. 2017. doi: 10.1155/2017/6787504.