UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Informatics

Personal manager of scientific conferences

End of Degree Project

Bachelor's Degree in Informatics Engineering

AUTHOR: Antsyferov , Daniil

Tutor: Sánchez Díaz, Juan

ACADEMIC YEAR: 2022/2023

Resumen

Este proyecto tiene la misión de creación de aplicación móvil para Android que sirve como sistema de gestión de congresos científicos y cubre tanto funcionalidades personales como administrativas. Esta aplicación permitirá a los autores presentar sus artículos al congreso para su evaluación y posible publicación. El sistema debe permitir añadir las revisiones a los artículos para qué sirven como un factor de decisión durante la admisión. Adicionalmente, se espera que la aplicación proporcionara las funcionalidades para configurar nuevos usuarios, gestionar el congreso y notificar usuarios de los resultados.

Aplicación ha sido desarrollada con el lenguaje de programación Kotlin y la librería de interfaz del usuario Jetpack Compose de este modo explorando nuevas tecnologías que están surgiendo en la industria de desarrollo Android.

Palabras clave: Android, Kotlin, Compose, congreso, gestión.

Abstract

This project has a goal of creating an Android mobile application that serves as a management system for holding scientific conferences taking into account both personal and administrative needs. Such an application would allow authors to present their articles to congress for evaluation and further possible publication. The system should allow providing reviews for the articles that would serve as a decision factor for admission. Additionally, the application is expected to deliver management features for new users' setup and conference format configuration as well as notify the author of the results.

The application was developed using Kotlin programming language with Compose Jetpack UI framework thus exploring new technologies that are arising in the industry of Android development.

Keywords: Android, Kotlin, Compose, conference, management.

Table of contents

1. Introduction 8
1.1. Objectives 8
1.2. Methodology 8
1.3. Structure 9
1.4. Relation with studies 10
2. Technology overview 12
2.1. State of mobile development 12
2.2. State of Android native development 13
3. Problem analysis and design 14
3.1. Use cases diagram 14
3.2. Use cases description 16
3.2.1. All roles use cases 16
3.2.2. Author use cases 18
3.2.3. Reviewer use cases 19
3.2.4. Admin use cases 20
3.3. Classes diagram 22
3.4. User interface prototypes 23
3.5. Color palettes 25
3.6. Ethical Analysis 26
4. Implementation of the solution 27
4.1. System architecture 27
4.2. Detailed system design 29
4.2.1. Database structure 29
4.2.2. Client application structure 32
4.3. Technologies 37
4.3.1. Kotlin 38
4.3.2. Jetpack Compose 41
4.3.3. Navigation component 42
4.3.4. Hilt 43
4.3.5. Firebase 44
4.3.6. Other libraries 44
4.3.7. Android Studio 44
4.3.8. GitHub 45
4.3.9. Android 45
5. Jetpack Compose research 47
6. User manual 49
6.1. Common features manual 49
6.2. Author manual 50
6.3. Reviewer manual 52
6.4. Admin manual 53

7. Future work ... 57
8. Conclusions ... 58
9. Bibliography ... 59
Annex A ... 61
Sustainable development goals ... 61
Annex B ... 63
UML notation ... 63

Table of figures

Figure 1 Use cases diagram 15
Figure 2 Classes diagram 23
Figure 3 Home and profile screens prototypes 24
Figure 4 Prototypes of article viewing screens 25
Figure 5 Light color palette 26
Figure 6 Dark color palette 26
Figure 7 Diagram of project architecture 27
Figure 8 Structure of subcollections 31
Figure 9 Domain module structure 34
Figure 10 Model classes 35
Figure 11 Example of callback conversion to suspend function 39
Figure 12 Example of custom scope function 41
Figure 13 Example of DI using Hilt 44
Figure 14 Example of composable function 47
Figure 15 Example of LazyColumn usage 48
Figure 16 User authorization screens 49
Figure 17 Profile screens 50
Figure 18 Article submission screens 51
Figure 19 Evaluation list screens 52
Figure 20 Add review screens 53
Figure 21 Add track screens 54
Figure 22 Article selection screens 55
Figure 23 User configuration screens 56
Figure 24 UML elements 63

Glossary

- **JVM:** Java Virtual Machine. The software responsible for executing bytecode and enabling Java to run on different machines.

- **Garbage collector:** Component of Java programming language responsible for automatically managing memory and freeing unused resources.

- **Observer pattern:** System design pattern that allows observer entities to subscribe to changes in one object thus receiving notifications about produced updates.

- **JSON:** JavaScript Object Notation. Data interchange format that is easy to read and write, commonly used to pass data between server and client application.

- **APK:** Android Package Kit. File format used for distributing and installing applications on Android devices.

- **API:** Application Programming Interface. A set of rules and protocols that allows applications and software components to communicate with each other.

- **NPE:** NullPointerException. Runtime exception that occurs in Java programming language when a program attempts to access a null object.

- **LIFO:** Last In, First Out. A processing order where the last item added is the first one to be removed.

- **IDE:** Integrated Development Environment. A software application that provides a set of tools to help developers in writing, debugging, and testing code.

- **VCS:** Version Control System. A software tool that allows developers to manage changes to their code enabling collaboration, version history, and reverting changes if needed.

1. Introduction

This project constitutes a development of a management tool to hold conferences. It is aimed to help the organizer set up processes in a manageable way that would also be easy to use by all parties involved.

It is delivered in the form of an Android application that should be accessible by a wide range of users and function in pair with a server accessible via the network. Some of its functional features are live updates and offline support as well as a simple user interface with smooth animations supporting adequate user experience.

The application provides the necessary tools to set up new users, submit articles, add reviewers, and evaluate submissions. It also has the functionality to notify participants of the results. Additionally, the program allows to set up various flexible tracks for the congress with dedicated deadlines that must be met by the participants.

Overall, the presented solution covers all the basic needs that are required to organize such events and is designed to enable high usability with the use of adequate interface elements that combine into easy-to-learn user interface.

1.1. Objectives

The goal of this project is to create a mobile application that will allow organizations to hold congresses by providing accessible instruments for submitting, evaluating, and managing articles. The application should adapt its functionality for the needs of different user groups such as authors, reviewers, and admins in order to ensure a fair and manageable process of selection of articles. User interface should rely on standard elements that users will expect to find in a mobile application and will be already familiar with. This will ensure a high level of usability and a very low learning barrier for new users.

Apart from functional requirements, it is also set to create an application that will be flexible and easy to maintain which would allow for future extensions to be made or possibly some of the modules to be reused throughout the project. It is also important to ensure adequate choice of technologies so that the complexity of the project will not increase needlessly but it will still be scalable for a potentially growing number of users and functionalities.

1.2. Methodology

Project management methodologies are essential for successful development of the software. They allow to estimate project resources, manage deadlines, prioritize, and focus development efforts on crucial parts of the program. These methodologies can vary in complexity, and some may be more suitable than others for different group sizes and project scopes.

Agile was the choice for this project due to the flexible nature of this work and the ability to quickly introduce changes to the intended solution. It is mainly focused on development cycles and allows to dynamically decide on the next steps to take which suits the format of regular meetings with the tutor where new requirements or problems may come up and need to be prioritized in the next cycle.

Agile can also be implemented in different ways but given that I'm the only developer of this project it was decided to go with a more simplistic approach called Kanban. It is usually implemented with the use of special tables divided into status columns with cards in them that represent different tasks. (Rehkopf, 2023) The number of columns can vary and for this project, three columns were used:

- **To do** – contains all the tasks that have been detected and need to be done. Ordered by priority.
- **In progress** – tasks that are in the process of development.
- **Done** – tasks that have been finished.

All new tasks are added to the backlog and are prioritized against other tasks in it. To update the status of the task developer should move it to one of the other columns. It is usually advised to limit the number of ongoing tasks to not spread effort between too many things risking delivering unsatisfying results. Also, more columns can be added to fit the actual development team like **In review** or **Testing**.

As a tool to implement the described management process, Trello was used. It is free software that allows to use all the essential features and provides some more advanced paid options for bigger teams. Overall usage of Kanban and Trello have helped me and the tutor to keep track of things and navigate through the development of the project.

1.3. Structure

This work consists of nine sections. The first one explains the topic, and what the application is doing and provides general information on the project as well as some explanation of how it is related to the studies.

The second section gives some context to the reader about technologies for mobile development and why the actual approach to implementation was chosen.

The third section is aimed to define the functional requirements of the system with various diagrams and tools without describing the actual implementation.

Forth section provides detailed information on the implementation and functioning of the system. It is divided into subsections and subsequent ones explain the system on more detailed levels so that readers with different technological backgrounds could limit themselves to higher levels of implementation and more proficient readers can get more information on the technologies used.

The fifth section explains some of the discoveries regarding the main new technology employed in the project which is Jetpack Compose. It provides more information on how it is functioning and how it is better than XML.

The sixth section presents the user manual to the reader which explains in detail how users can interact with the application and achieve their respective goals.

The seventh section provides some thoughts and ideas on possible future work in the context of this project.

Finally, conclusions are presented with some thoughts on the project and key takeaways. Additionally, a section with references to useful sources is included. At the end of the document, the reader can also find annexes with information on ODS and UML notation.

1.4. Relation with studies

During the development of this project, it was necessary to deploy knowledge and skills that were obtained from various subjects as well as those I have learned independently. In this section, I would like to highlight some of the most useful subjects.

On one hand, *Ingeniería del software* was very helpful in understanding how to design a complex system with many components and how to formalize the solution. It allowed me to gain an understanding of how such a system could function and what approaches could be implemented.

On the other hand, subjects like *Bases de datos y sistemas de información, Redes de computadores,* and *Tecnología de sistemas de información en la red* have contributed to the development of the database and establishing communication between client application and server.

Other subjects like *Interfaces persona computador* and *Introducción a los sistemas gráficos interactivos* have helped with gaining an understanding of how to design and develop appealing user interface. Also, *Gestión de proyectos* has helped with organizing the process of development. Another useful subject is *Ciberseguridad en dispositivos móviles,* it helped with implementing best practices and avoiding popular mistakes during application development.

Amount the skills that I have learned independently I could name Android development and all the components and libraries that were employed during the development of this project. Finally, Jetpack Compose could be highlighted in this section as a new framework that had to be mastered completely from the start for the purpose of developing this application

2. Technology overview

2.1. State of mobile development

Nowadays developers have access to a rich variety of technologies and frameworks when choosing to create mobile applications. Two fundamental approaches are native and cross-platform development. Both have strong and weak points to consider when deciding on which approach to implement. Native development suggests writing code for Android and IOS separately using respective programming languages while cross-platform development allows writing projects in one language which is then translated into native implementations that operating systems can understand.

This fundamental difference is the source of all the benefits and drawbacks that can be defined between the two approaches. Typically, cross-platform development allows to get a working version of the product on the market faster, and covering both Android and IOS systems also makes it so that startups can engage with a wider audience. The speedup in development is obtained through the reuse of a single codebase for both platforms. However, cross-platform solutions tend to suffer from several problems, namely, performance issues, weaker security, delayed software updates, and difficulties with the implementation of hardware features.

Since the code needs to be translated from one language to the native languages of both platforms, for example, Kotlin and Swift, the performance of the solution tends to be significantly lower than direct native implementations. As for security issues and later updates, the reason for these problems is that any new feature introduced on the native platform is available there out-of-the-box but to get access to this feature in a cross-platform environment it needs additional updates which are not guaranteed to come on time. Finally, cross-platform frameworks can struggle with implementations for hardware-dependent features and some modules may need to be developed natively anyways.

To sum up, cross-platform development is a great tool to deliver faster results for more users but when it comes to writing a high-quality application that uses device resources efficiently, relies on security updates, or is hardware-focused, native development is the preferred way to go. Ultimately any decision will always represent some sort of trade-off and developers need to select appropriate technologies for their needs.

For this project, I have chosen to target the Android platform using native development. The reasoning behind this decision is to introduce a proof-of-concept solution that would correspond to industry quality standards and make use of the latest available features and technologies to provide the best user experience. This does suggest that the audience of the IOS platform will not have access to the initial version of the application, however covering users of the Android platform will be enough to achieve initial goals of verifying the concept and to make further decisions on the development of the project.

2.2. State of Android native development

It is also important to provide context on the state of Android native development. In general, it can be done using Java or Kotlin languages but the preferred one is Kotlin due to many beneficial features and better support of new libraries and tools. It is also interoperable with Java, so a developer doesn’t lose any possible functionalities when choosing Kotlin.

The most useful tool that is available in the Android framework is Android Jetpack. It is a collection of libraries, tools, and guidelines that are offered by Google to developers to simplify complex common tasks and provide a way of writing high-quality applications more easily.

Jetpack introduces a collection of libraries that are divided into categories of architecture, UI, foundation, behavior, and testing components. Each group contains useful tools and libraries to solve specific problems. This project does not make use of all of them, but developers need to have a general idea of what tasks can be achieved by making proper use of available tools.

Another key aspect is the UI framework that will be used in the project. Since the early days of Android XML layouts were used to define user interfaces and they remain the most popular instrument for this. However, in 2019 Google introduced the early alpha of Jetpack Compose, and the first stable version was released in 2021. (Android Developers, 2023) Since then, the technology has received a lot of updates and improvements and was set on track to become the primary way to create user interfaces in Android. It provides a less complex and more flexible way to create a stateless UI while writing less code and using only Kotlin programming language.

This project will use Jetpack Compose to access all of its benefits and gain more knowledge and experience with the new technology.

3. Problem analysis and design

This section will describe the requirements for the system to comply with and suggest a solution to be implemented.

3.1. Use cases diagram

The goal of the use cases diagram is to define functions that are available in the application for different types of users. (Bittner & Spence, 2002)

The diagram includes one generic user, two different user roles, and an admin which is a type of super user who has access to all the features. All the users have access to features of account management and basic content viewing capabilities provided that the admin has granted them permissions. The author is supposed to be submitting articles and possibly updating them based on received evaluations. Reviewers should be performing evaluations and to avoid being biased by other evaluations should not be able to see them. Admin in its turn doesn’t have any limitations of author or reviewer and additionally has more functions like managing a congress or assigning permissions and user roles.

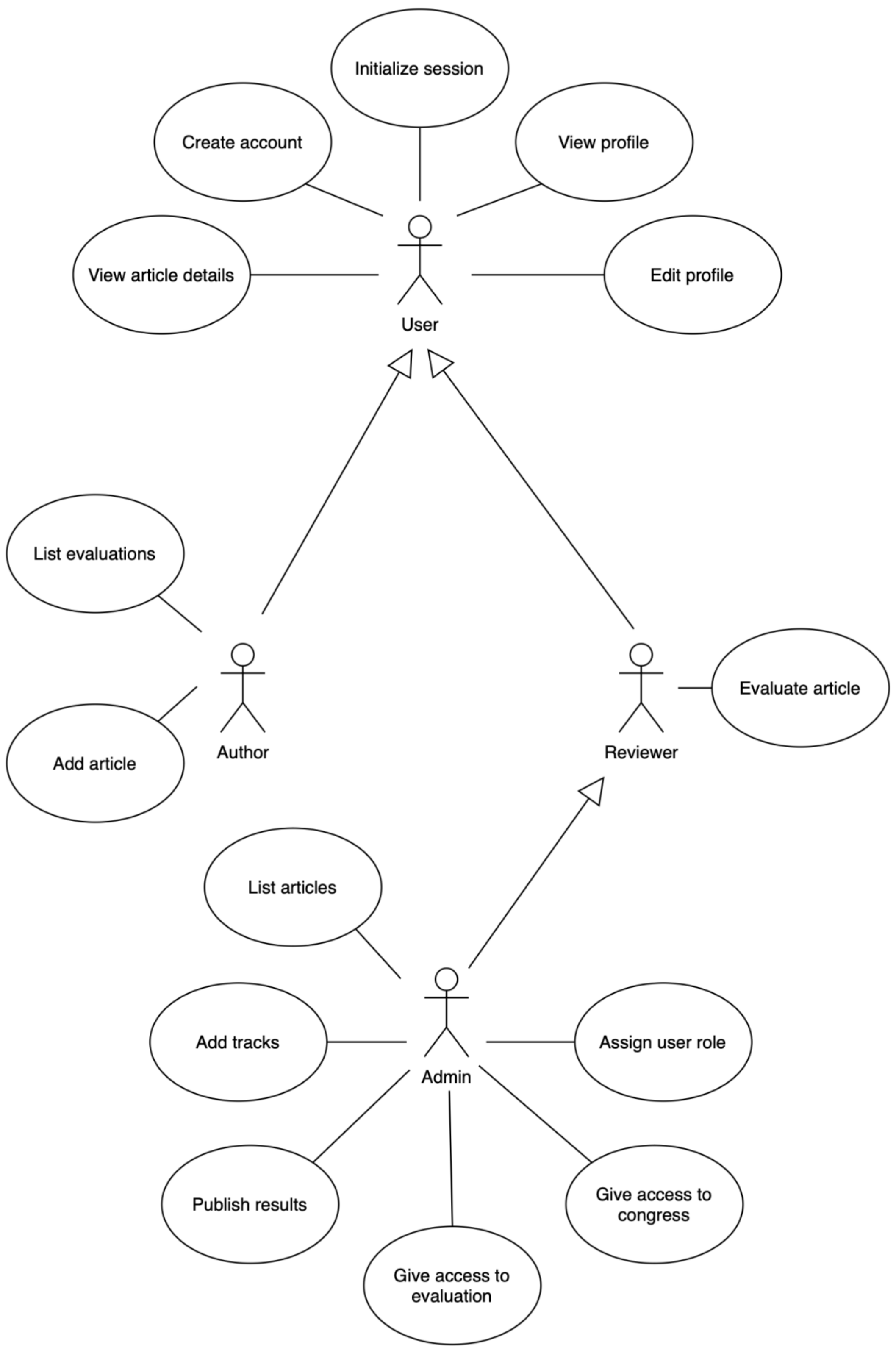


Figure 1 Use cases diagram

3.2. Use cases description

In this section, reader can find more information about the use cases presented in Figure 1.

3.2.1. All roles use cases

Use case	Create account
Actors	User
Goal	Register within the system.
Summary	User creates an account in the system and his data is saved in the database.
Preconditions	No user is authenticated in the application.
Postconditions	New entry for the user is created in the database and his credentials can be used to log in.

User actions		**System actions**	
2	User chooses to create a new account.	**1**	System displays login options.
4	User introduces all the necessary data.	**3**	System displays option to create account with Google, email, or phone number and provides appropriate input fields.
		5	System creates an entry in the database for the new user.
		6	System performs authentication for the newly created user.

Use case	Initialize session
Actors	User
Goal	Access the application.
Summary	User identifies and authenticates using his credentials.
Preconditions	No user is authenticated in the application.
Postconditions	User is authenticated in the application.

User actions		**System actions**	
2	User introduces the credentials.	**1**	System displays login options with credentials input.
		3	System verifies introduced data.
		4	System authenticates user.

Use case	View profile
Actors	User
Goal	Review the information that is presented in user profile.
Summary	User can open the screen to see their name, avatar as well as contact information.
Preconditions	User is authenticated.
Postconditions	-

User actions		**System actions**	
1	User chooses option to view his profile.	**2**	System displays profile information.

Use case	Edit profile
Actors	User
Goal	Update profile information.
Summary	User can edit their profile information, upload new avatar.
Preconditions	User is authenticated.
Postconditions	Profile is updated with new data that was introduced.

User actions		**System actions**	
1	User chooses option to edit his profile information.	**2**	System shows input fields for user data and avatar selection option.
3	User updates the data and saves changes.	**4**	System saves updated information on a server.

		5	If avatar has changed system uploads new avatar to the server.

Use case	View article
Actors	User
Goal	Get an overview of the article.
Summary	User can see details of the given article as well as download the article pdf.
Preconditions	User is authenticated and has access to the article.
Postconditions	-

User actions		**System actions**	
1	User selects an article.	**2**	System shows article details and offers options to preview or download article.

3.2.2. Author use cases

Use case	Add article
Actors	Author
Goal	Submit new article to the congress.
Summary	Author can add new article to the congress by uploading a pdf.
Preconditions	Author is authenticated and has access to the congress. Congress is open to the submission of new articles.
Postconditions	New article entry is created in the database and the pdf file is uploaded to server.

User actions		**System actions**	
1	Author selects a congress.	**2**	System shows a special action button to add an article.
3	User introduces all the necessary data.	**5**	System creates new entry in the database for the article
4	User selects the pdf file to be uploaded.	**6**	System saves the pdf to the server.

Use case	List evaluations
Actors	Author
Goal	Get an overview of provided reviews.
Summary	Author can see the list with all the reviews that were provided for his article.
Preconditions	Author is authenticated and has access to the congress. Author has submitted an article. Article has received reviews.
Postconditions	-

User actions		System actions	
1	Author selects option to view the evaluations.	2	System displays all the available evaluations for the article.

3.2.3. Reviewer use cases

Use case	Evaluate article
Actors	Reviewer
Goal	Submit a review of the article.
Summary	Reviewer can add a review to the article providing its evaluation and option descriptions.
Preconditions	Reviewer is authenticated and has access to the congress and the article. Congress is open to submitting reviews.
Postconditions	New review entry is created in the database.

User actions		System actions	
1	Reviewer selects an article and chooses to add an evaluation.	2	System asks for all the required data for the evaluation.
3	Reviewer introduces the data and saves changes.	4	System creates new review entry in the database.

3.2.4. Admin use cases

Use case	Add tracks
Actors	Admin
Goal	Add tracks to the congress.
Summary	Admin can add tracks to the congress thus defining all the due dates and deadlines for participants.
Preconditions	-
Postconditions	Congress is set up with corresponding tracks.

User actions		**System actions**	
1	Admin selects a congress and chooses to add a new track.	**2**	System displays the date and other input fields.
3	Admin introduces all the necessary data and saves changes.	**4**	System updates the corresponding information in the database.

Use case	Publish results
Actors	Admin
Goal	Make the outcome and list of admitted articles known to participants.
Summary	Admin can select which articles to admit and make the results visible to other users.
Preconditions	Congress is in a finished state, articles have been submitted and evaluated.
Postconditions	List of admitted articles is formed and is visible to the users.

User actions		**System actions**	
1	Admin selects a congress.	**2**	System checks if congress is finished.
4	Admin performs articles selection and saves results.	**3**	System provides an option to select articles for admission.
		5	System updates the database and makes all the admitted articles visible to participants.

Use case	List articles
Actors	Admin
Goal	Get an overview of submitted articles.
Summary	Admin can see a list of articles that were submitted for the congress.
Preconditions	Admin is authenticated.
Postconditions	-

User actions		**System actions**	
1	Admin selects a congress.	**2**	System displays congress information along with a list of submitted articles.

Following three use cases share the same pattern of configuring user accounts by admin so only one will be described in detail.

Use case	Assign user role
Actors	Admin
Goal	Set user role to author or reviewer.
Summary	Admin can assign user role of reviewer.
Preconditions	-
Postconditions	User has his new role defined.

User actions		**System actions**	
1	Admin opens screen with a list of users.	**3**	System displays user information and an option to change user role.
2	Admin selects a user.	**5**	System updates user entry in the database.
4	Admin selects new user role and saves changes.		

Use case	Give access to congress
Actors	Admin
Goal	Let author access congress.
Summary	Admin can select an author and permit them to access a congress.
Preconditions	Selected user is an author.
Postconditions	Author can access congress and submit articles.

Use case	Give access to the evaluation
Actors	Admin
Goal	Let the reviewer evaluate the article.
Summary	Admin can select a reviewer and permit them to evaluate an article that was submitted to the congress.
Preconditions	Selected user is a reviewer. Congress has submitted articles.
Postconditions	Reviewer can perform an evaluation of the article.

3.3. Classes diagram

This section presents the classes diagram which is aimed to explain how different entities interact with each other in the application. Reader can see that congress is composed of different tracks which are controlled by the admin. The tracks will have an arbitrary number of articles assigned. In the article entity, is_admitted parameter is responsible for the admission of the article. Articles themselves are submitted by authors and can have many reviews that evaluate them. Reviews are added by reviewers and will be visible to the authors of the articles.

Also, admin, author, and reviewer entities extend from the user entity which corresponds to the use cases diagram.

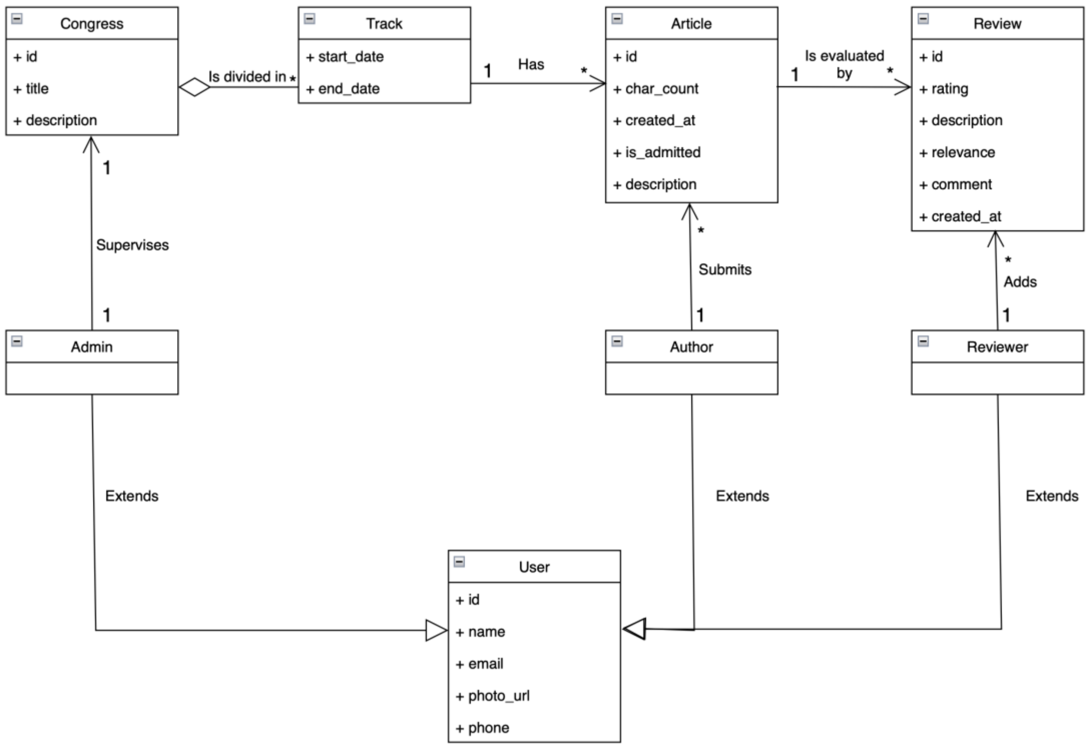
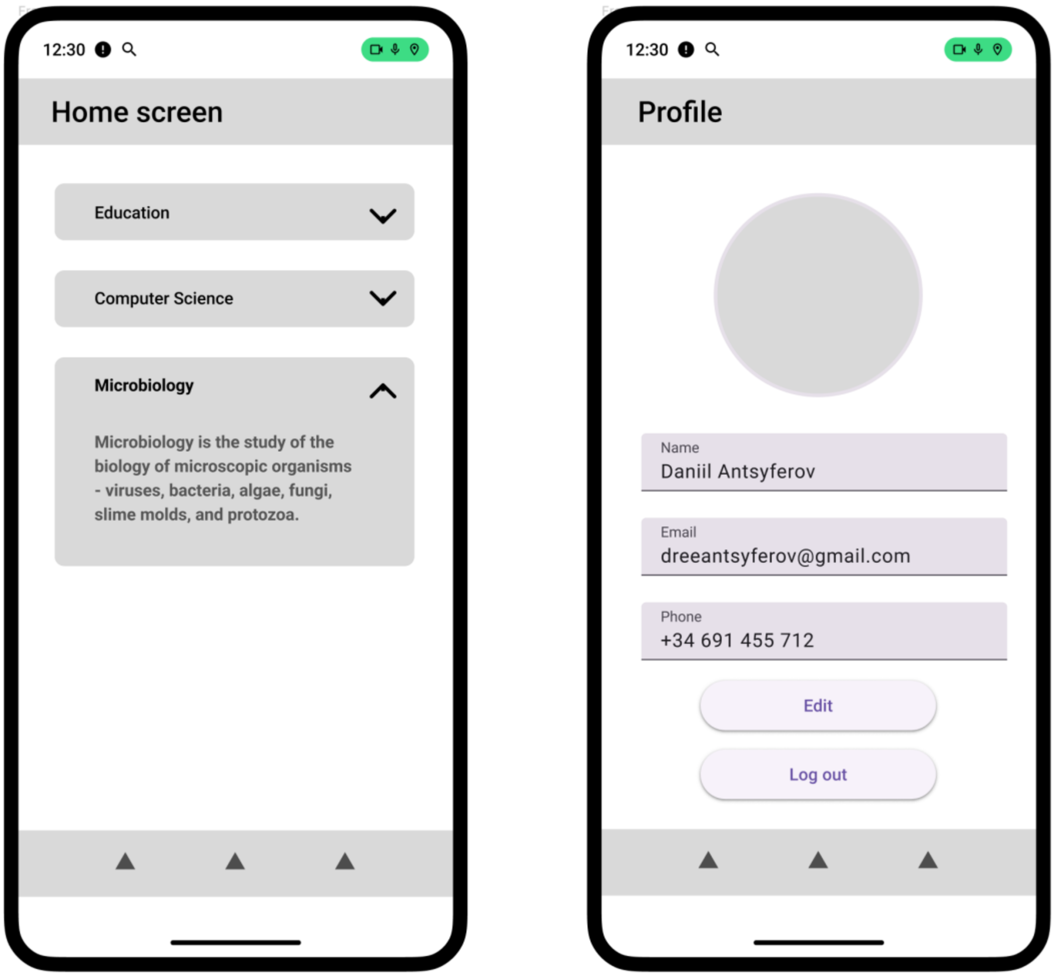


Figure 2 Classes diagram

Note that description of used UML elements is available in Annex B.

3.4. User interface prototypes

In continuation, some of the initial prototypes are presented. They were used to visualize ideas for the user interface in the early stages of development and make decisions on the usage of key interface elements and how they are going to help users interact with the application.



Microbiology is the study of the biology of microscopic organisms - viruses, bacteria, algae, fungi, slime molds, and protozoa.

Name
Daniil Antsyferov

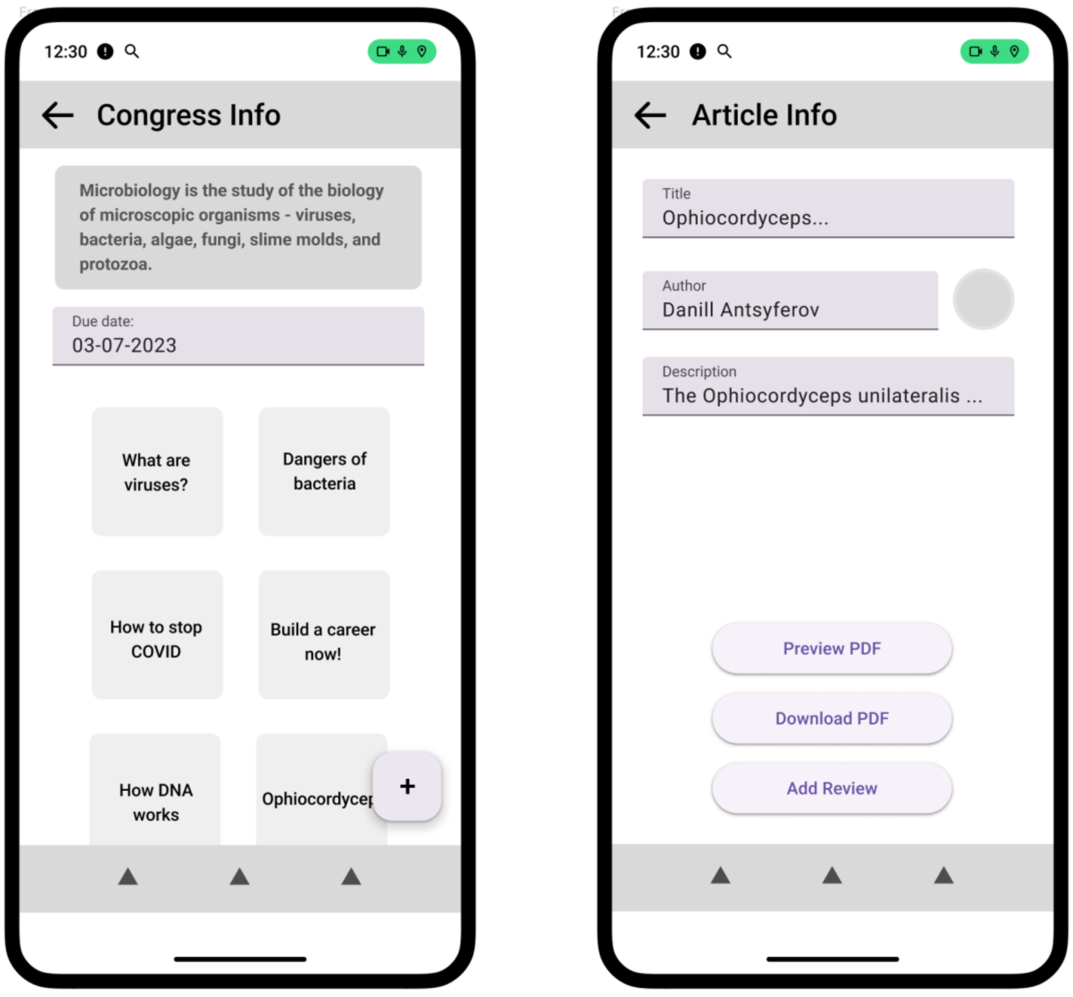
Email
dreeantsyferov@gmail.com

Phone
+34 691 455 712

Figure 3 Home and profile screens prototypes

In Figure 3 reader can observe prototypes of home and profile screens. One of the main decisions was to implement bottom navigation to provide users with familiar tools for navigation and switching between different use cases. An argument could be made for using a side drawer with a menu for navigation, but this would overcomplicate the layout and possibly be confusing with some navigation gestures. However, this is still a possibility to consider in the future in case additional features will be added to the app.

More prototypes are presented in Figure 4. They follow the same navigation scheme and give more insight into how UI could be implemented to cover some of the defined use cases of viewing articles.



Microbiology is the study of the biology of microscopic organisms - viruses, bacteria, algae, fungi, slime molds, and protozoa.

Due date:
03-07-2023

Title
Ophiocordyceps...

Author
Danill Antsyferov

Description
The Ophiocordyceps unilateralis ...

Figure 4 Prototypes of article viewing screens

Overall application has maintained key aspects defined in these prototypes however some changes have been made during the development phase to adjust the prototypes for better user experience. The final layout of the application will be presented in the following sections.

3.5. Color palettes

This section presents chosen color palettes for the UI. Android framework allows to easily implement dark and light themes but switching sets of colors that ap is going to use so both options have been implemented in the application. Color palettes have been obtained through the use of online resources that ensure chosen colors are balanced and follow material guidelines.

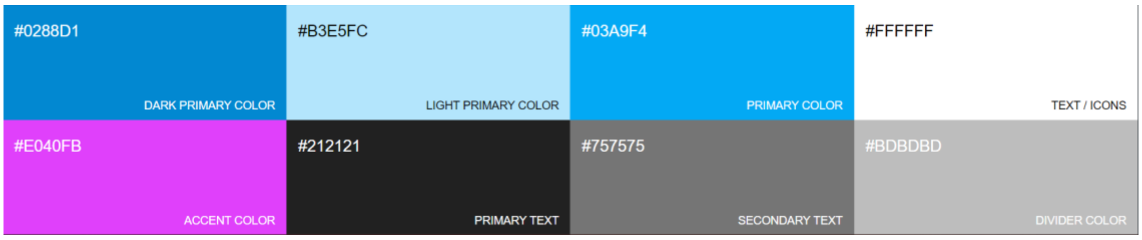


Figure 5 Light color palette

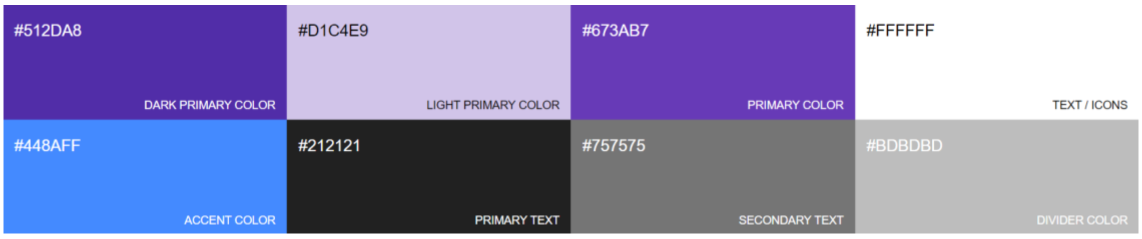


Figure 6 Dark color palette

In figures 5 and 6 reader can observe selected color palettes.

3.6. Ethical Analysis

The project has some ethical considerations to be taken into account. Namely, the process of article evaluation should guarantee the fairness of the review by not influencing reviewers' opinions and not showing data that could lead to a biased evaluation. This is achieved by concealing all other article evaluations from the reviewer so that a fair opinion can be formed.

It is also possible for the reviewer to perform more than one evaluation thus updating the previous version of the review. However previous versions of the review are not deleted and can be accessed by the author and admin. This provides the ability to correct a review but doesn't create an opportunity for any unethical behavior of changing reviews dramatically under some external influence.

Another aspect of this process is that the reviewer's name is never shown to the author thus not allowing him to contact the reviewer outside of the system and again preventing any potential unwanted influence.

4. Implementation of the solution

This section describes the architecture of the system and is essential for understanding how the application works and how it achieves its objectives of providing services to a potentially elevated number of users as well as being flexible enough to accommodate any immediate needs for changes in a dynamic environment.

4.1. System architecture

In most general words this is an example of a server-client application where the server is responsible for storing data and communicating it to the client on request. Client in this case the Android application and role of the server is performed by Firebase. The components of the application are represented in Figure 7 to visualize the information.

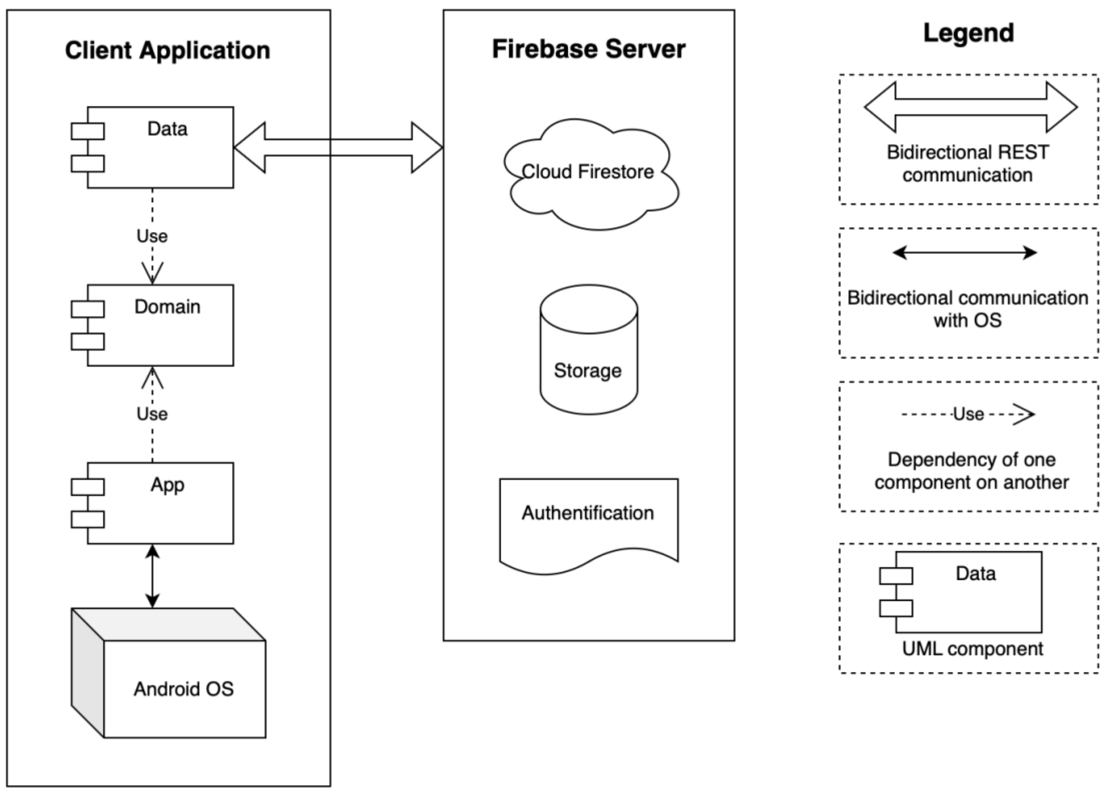


Figure 7 Diagram of project architecture

I will start by describing how the server is implemented. Since Firebase is not just a singular service but rather a collection of services it is divided into different parts such as Cloud Firestore, Authentication, Storage, and others such as Cloud Messaging or Crashlytics.
In the scope of this project, only the first three are used but additional services may be configured if the need arises.

Cloud Firestore performs the role of the main database in this project. It’s a NoSQL database that provides useful features like real-time synchronization or offline support. It stores data in what can be called a highly modified json which is split into collections and can be queried in an SQL-like manner. These queries are more limited compared to their SQL counterparts and do not provide strong type safety but with correct handling on the client side I was able to achieve a very flexible system where changes to the data models can be introduced without any need for extensive migration and older models can be updated what is called on the fly.

Firebase Storage in its turn is a go-to solution to save files on the server. It functions in a similar way to a file system on any PC. Files are saved to some path in the system and can be retrieved by accessing the same path. Also, they can be deleted or overwritten by another file on the same path which is also an equivalent of modification. On top of this, it is possible to generate URLs so that upload files can be downloaded from any device with network access.

Firebase Authentication provides a service for managing users and as well creating new accounts with different providers like Google, Facebook, Twitter, or just email or phone number for identification.

As for the Android application, it is using a multi-module approach which is an industry standard for developing mobile applications. (Robert, 2018) There are a couple of ways to implement it but in this case, the application is divided into the domain, data, and app module. Each of them is responsible for a certain layer of functionality and can potentially be substituted for another implementation.

Domain module is the core part of the application which defines how other modules are implemented and provides most of the functionality to the UI layer by making appropriate queries to the data layer. It also provides data models to other parts of the app thus controlling the scope of information they can work with.

The data module is an implementation of a communication layer between the application and Firebase. It is specifically implemented to be swapped out for any other API implementation if the need ever arises. So potentially any transition from Firebase to another service provider can be done without affecting most of the client-side code.
It is worth noting that since Firebase provides offline support out of the box, substituting it for another API would create a need to manually implement this feature within the potential new data module.

The app module represents the UI layer of the application and connects it to the Android framework performing all the needed communications with the system. Naturally, it is responsible for managing any features that require user or hardware interactions since they should be performed through the use of Android OS to comply with security policies.
On top of that, this module serves as an entry point for the application which will be launched and managed by the Android system.

4.2. Detailed system design

This section is aimed to explain system architecture in more detail and to provide information on the implementation of already mentioned components.

4.2.1. Database structure

As was already mentioned in the previous section Cloud Firestore is used as a primary solution for the Firebase but before explaining how it is implemented it would be useful to formalize the intended database structure to provide needed context.

The logic scheme for the database can be represented as the following:

Please note that PK is a primary key, and FK is a foreign key. Also, the notation Entity(parameters…) means that a table in the database has columns that correspond to parameters.

Publication (id, title, description, review_date, final_submit_date, completion_date)
PK: id

For the reader's clarity, the publication is an entity that was designed to combine properties of congress and a track represented earlier. The idea is to simplify the system and represent all the needed information in one table so that admin can interact with it much easier. The publication is very similar to the track it has its title and deadlines as well as multiple articles will be linked to it but it allows admin to manipulate its fields with simple queries eliminating the need for complex SQL queries which will be very helpful given the details of actual implementation that are explained in the following sections.

Article (id, publication_id, author_id, title, description, character_count, is_selected, created_at)
PK: id
FK: publication_id → Publication (id)
FK: author_id → User (id)

Review (id, publication_id, article_id, review_author_id, article_author_id, rating, relevance, comment, description, created_at)
PK: id
FK: publication_id → Publication (id)
FK: article_id → Article (id)
FK: review_author_id → User (id)
FK: article_author_id → User (id)

User (id, role, name, email, phone, photo_url)
PK: id

AuthorOf (user_id, article_id, publication_id)
PK: user_id, article_id, publication_id
FK: user_id → User (id)
FK: article_id → Article (id)
FK: publication_id → Publication (id)

HasPermission (user_id, publication_id, article_id)
PK: user_id, article_id, publication_id
FK: user_id → User (id)
FK: article_id → Article (id)
FK: publication_id → Publication (id)

In this scheme, reader can observe N:1, 1:N, and N: M types of relationships between entities. In SQL database 1:N relationship can be represented with the inclusion of a foreign key, for example, Article has a publication_id key which links many Article entities to one Publication.

On the other hand, 1:N will need a new table such as AuthorOf which represents that one User entity is an author for many Article entities. Also, a new table is required for N: M relationships which in this case is a HasPermission table which is needed to link many users to many articles or publications. This relation indicates that the user can add new articles to the selected publication or add reviews to the selected articles, this choice will depend on the user role that is assigned to the given entity.

Now as reader understands the intended database structure an explanation is needed on how this is implemented in the NoSQL environment. In Cloud Firestore database consists of collections that can be seen as tables and each collection contains documents that reader can think of as entries in the table. (Firebase, 2023) There is no built-in foreign key functionality for representing N:1 relation but this can be simply imitated by ensuring the supply of correct values by the client application.

As for 1:N and N: M it would be possible to create additional collections to store these relations but with limited querying capabilities extracting data would become a problem because there is no support for complex queries. There are two possible solutions to this problem which I'm going to explain.

The first is as simple as storing an array of values directly in the entity. It is not a problem because the database supports arrays of arbitrary length, so we don't have to worry about exciding memory limits which make arrays in the SQL environment unusable.
This has proven to be a convenient way of replacing AuthorOf and HasPermission tables since in the client app firebase can directly parse those values into Kotlin data classes. This on the

client side simplifies work with the database thus avoiding writing complex and error-prone code.

The second option is creating a subcollection within a collection of documents. For example, in the case of this project, every document in the publications collection has a subcollection of articles. This structure is illustrated in figure 8.

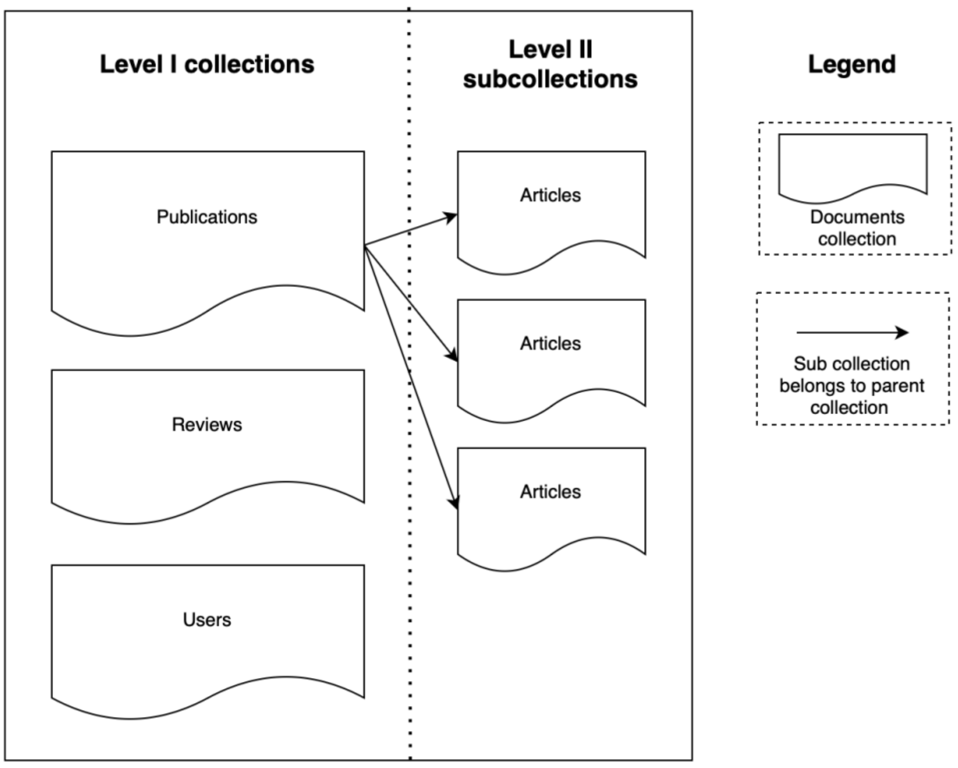


Figure 8 Structure of subcollections

To properly understand this concept reader should know that documents are queried by their path in the database. This means that to query a specific article the client application will refer to the database with the following path:

“publications/{publication_id}/articles/{article_id}”

This can be seen as equivalent to SQL query:

SELECT * FROM articles WHERE id={article_id};

The advantage of this approach is that instead of looking through the whole collection of articles server only needs to search the specific subcollection linked to the parent collection thus greatly reducing the size of the problem. Actual speedups will depend on the specific distribution of data in the collections but in general, we can assume that every publication has an equal number of articles thus problem size is reduced by N times where N is the number of publications.

However, the disadvantage is the fact that the client must know publication_id parameter to carry out the request. Therefore, it is included as a foreign key in the Article entity in the formal representation of the database. Without this key querying a specific article from the database would be more complicated than in SQL since the client would have to check every possible subcollection of every publication. But this limitation has not become a problem in the development since it naturally fits into the interface where user is first required to select a publication and only then a corresponding article.

4.2.2. Client application structure

This section describes the internal composition and implementation of the client app.

4.2.2.1. Application Modularization

As it was already mentioned before application is divided into three modules: app, domain, and data. Before explaining why this structure was chosen reader should first understand what the benefits of modularization are and what are we goals looking to achieve by implementing this architecture. (Android Developers, 2022)

In general code qualities such as scalability and readability tend to reduce as the code base grows. Therefore, developers should act when such effects occur and implement some sort of appropriate architecture to ensure that the growing codebase remains manageable and maintainable.

One way of achieving this is splitting the app into different modules. A module can be thought of as a highly independent package that can be replaced or reused on demand. Here is the list of the benefits that can be provided by modular architecture:

- Scalability – in a big application even a small change can trigger a lot of changes to seemingly unrelated code. Therefore, splitting code into modules that follow the principle of separation of concepts can limit the potential scope of changes that will be produced as a result of code alteration.
- Encapsulation – modules help to ensure that each part of code possesses minimal knowledge of other parts. Code inside modules can be marked as internal or private thus remaining invisible outside of the module.
- Reusability – modules can be seen as building bricks of the application and can be exported and reused in other applications as well as enabled or disabled by demand providing a convenient way of supplying demo versions of an application to the users.

There are more potential benefits to the modularization such as marking ownership, enabling easy testability, or reducing build times, but those were not primary concerns when designing this application.

However, none of these benefits are guaranteed and to achieve them modularization has to be carried out in an appropriate way for each given application. One of the concepts that can help with understanding if a given modularization approach is beneficial is granularity.

Granularity intends to describe to which extent your codebase is composed of modules. To understand the level of granularity developer should consider the size of the codebase and the number of modules it consists of. Too fine-grained code will be very complex in maintaining and can even negatively affect some of the earlier described qualities. This is because modularization brings a certain amount of overhead and boilerplate code that also needs to be supported. On the other hand, too coarse-grained modules might now provide enough benefits due to each of them being in fact another monolith structure.

Another concept that can help to evaluate if modularity is implemented correctly is coupling. Coupling is a way to measure the degree to which modules are codependent or in other words how many dependencies exist between two given modules. In general, developers should aim to achieve low coupling.

One more thing to consider when modularizing the app is cohesion. It represents how parts of the given module are functionally related to each other. It is a good idea to aim for high cohesion which will make the application logically structured and code clearly readable.

Knowing these principles, it would also be beneficial to discuss two main strategies when modularizing the application and how they can be combined to achieve maximum benefits. Those strategies would be modularized by layers or by features.

The first one is what is implemented in the application it has 3 modules each responsible for a different layer of functionality. The data module is responsible for communicating with the server, domain module contains business logic for example it determines what users can do with their level of permissions in the system or how what state can be provided to the app layer. The app layer in its turn serves as the entry point for Android application and is responsible for displaying the UI state of the system.

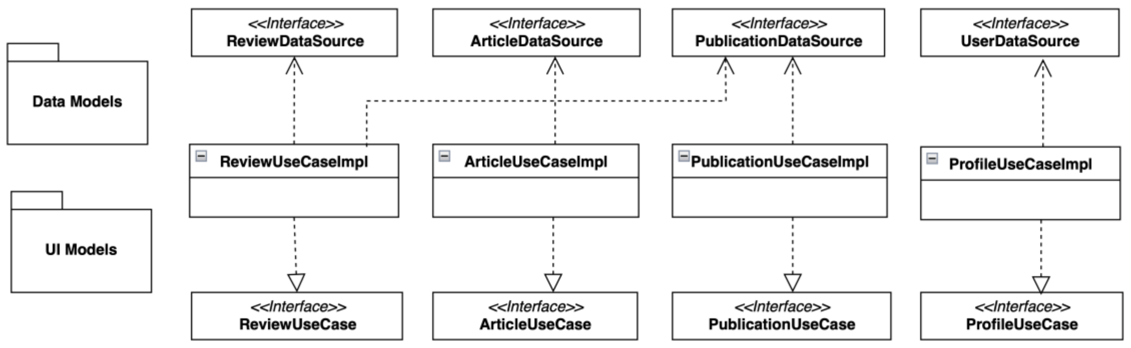
The second option would be modularizing by features. For example, there could be a feature module containing all the code necessary to query, create and display articles and then other feature modules split into coherent blocks such as managing users or publications. The downside of this approach is the high level of repeated boilerplate code because it is a frequent situation when one feature module would need to reuse some code from another feature module, and it would be unable to do so without violating its independent structure.

The third option is derived from the previous two and consists in combining both approaches. The idea would be to split each feature module into layer modules and possibly supply some core modules for access to the database or network resources. That would enable feature modules to reuse the code of other modules in the same layer without causing too high coupling.

In the application, I have implemented the first option which suits it well by providing a medium level of granularity, low coupling, and medium coherence. In the event of growing the codebase, I would suggest switching to the third way of combining modularization by layers and by features but in the scope of this project it would produce a huge number of boilerplate code resulting in a system that is way too hard to maintain for its size. Therefore, I consider modularization only by layers the best choice for the given scenario.

4.2.2.2. Domain classes organization

In continuation, reader will be presented with a more detailed organization of classes in the application. The general structure of the domain module can be seen in Figure 9. Note that description of used UML elements is available in Annex B.



Data Models
UI Models
<<Interface>> ReviewDataSource
<<Interface>> ArticleDataSource
<<Interface>> PublicationDataSource
<<Interface>> UserDataSource
ReviewUseCaseImpl
ArticleUseCaseImpl
PublicationUseCaseImpl
ProfileUseCaseImpl
<<Interface>> ReviewUseCase
<<Interface>> ArticleUseCase
<<Interface>> PublicationUseCase
<<Interface>> ProfileUseCase

Figure 9 Domain module structure

Domain module exposes interfaces for data and app modules to work with. Data modules depend on the DataSource interfaces and implement them. Thus, the domain module can receive the data from it and process the correct data to display for the app module. The app module in its turn depends on UseCase interfaces and consumes data from them to display UI information to the user or initiate system events.

Using interfaces with the DataSource name is a common pattern for decoupling high-level business logic from low-level storage and networking implementation. This enables the domain module to initiate I/O operations without considering how it is implemented in the system. This means that as long as the domain module is supplied with some valid implementation of required interfaces the application can function normally. For example, Cloud Firestore implementation in the data module could be easily swapped out for any other database implementation without affecting any application code at all.

UseCase interfaces follow the same logic with the difference that they are implemented in the domain module and are used by the app module. This enabled a very similar advantage for the app module as described in the previous paragraph for the domain module, the app module can function independently of the implementation of business logic.

As reader could have noticed Figure 9 also contains blocks for data models and UI models. In Figure 10 reader can observe these classes in detail. In continuation, it is also explained why there are two sets of models and what is their purpose.

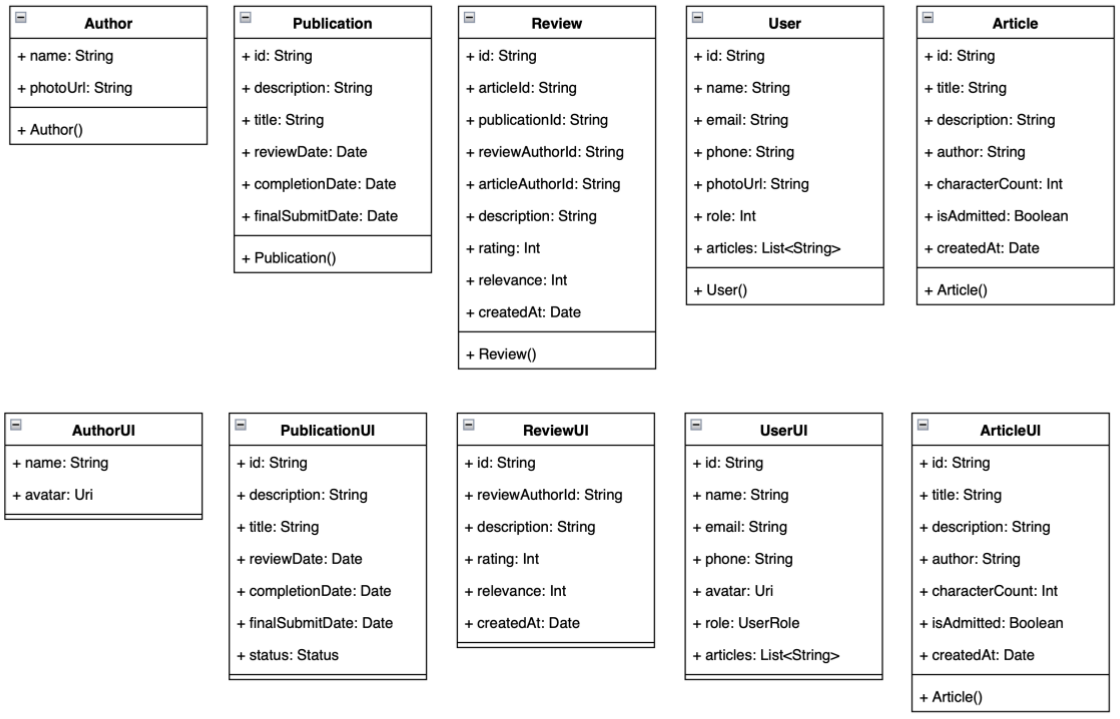


Figure 10 Model classes

These classes follow the same relations that were described in the database structure explanation. The important piece of information here is that there are duplicates of these classes for ones that are stored in the database and others that are presented on UI. This structure helps to further decouple UI from database implementation and ensures app module and data module do not share any models and can function completely independently and unaware of each other.

For example, it is possible to add a field in the database implementation of a model without affecting the user interface layer at all. It can be useful if say some business requirements come in and it is needed to store additional data for the given model that only affects some business logic, but the UI is the same. This implementation would allow to limit the scope of such a change to only data and domain modules without affecting any UI layer thus making it easier to develop and maintain the codebase.

However, it is also worth mentioning that adding duplicate or very similar model classes creates a certain overhead in writing and maintaining the codebase since any global changes to the system need to be reflected in data and UI-level classes. Therefore, it is important to consider actual benefits before implementing such a structure.

In the scenario of this project, I find it appropriate since it provides all the described benefits, and the overhead is negligible.

4.2.2.3. Data module structure

Due to the use of the Firebase library for communication with the database data module only contains four classes which are respective implementations of their DataSource interfaces. They are responsible for making queries and performing transactions. They also provide local caching implementations which is one of the features of the Firebase library.

If another database implementation was chosen it would be needed to split these classes into ones that are responsible for local storage on the device and others that are communicating with the remote database. Since this is not the case the data module remains quite simple in its implementation which is an important benefit of the chosen database.

4.2.2.4. App module functionality

Due to the chosen modularization strategy app module serves as an entry point for the Android system and provides the implementation of the user interface. Its implementations can be very distinct base on selected technologies so in this section I will only mention its general structure and feature and describe more low-level decisions in later sections.

The module has one Activity which is a class that will be instantiated by Android to launch the application and all the UI and navigation are rooted in this Activity. Single activity type of application is a common choice for small to medium-sized apps since it simplifies management of the code.

Also, the module has a navigation component that is responsible for the selection of the user interface to be drawn on the screen. Classes that are responsible for drawing UI and implemented separately as would be expected from any modern application.

Finally, the module is responsible for requesting and receiving data from the domain layer as well as starting events based on user interactions. This is performed mainly with the use of ViewModels. ViewModel classes are part of the Android framework and provide an abstraction layer over the user interface. This is a necessary part of architecture because UI can be destroyed and/or recreated and any time thus erasing all the present state and active subscriptions as part of the observer pattern. (Gamma, et al., 1995) These changes are called lifecycle in Android and ViewModel is responsible for correctly handling these events, saving the state when necessary, and clearing it after Activity was destroyed.

For the reader to properly understand the importance of this functionality first concept of memory leak needs to be introduced. The memory leak is a type of error that occurs when the JVM garbage collector fails to release unused objects from memory.

Java memory model is divided into the stack where primitive values are stored in LIFO order and the heap where all the allocated objects are assigned memory dynamically. (Gamage, 2022) While clearing the stack is not a problem and can be done with the usual management of method scopes, clearing the heap requires a more complex algorithm which will be clearing all the unused objects. Garbage collector implements such an algorithm. It considers an object unused if it can’t be referenced from the stack or application root. During its execution garbage collector clears such objects from memory to free up space. Failure in this step will result in frequent app-not-responding and out-of-memory errors which will significantly impact user experience.

Now when reader knows what a memory leak is and why it is important to prevent such errors, I will explain what can potentially cause it in the Android environment. One of the more frequent errors is passing Activity context to an entity that lives longer than the Activity, this could be a singleton class or new thread that takes some time to execute. While this entity holds Activity context and is considered to be in use, Activity can’t be released from memory. This means that every time Activity is recreated new instance is added to the memory, but previous instances are not cleared. This is a simple example of a memory leak that over time will accumulate and cause the app to crash.

ViewModel was introduced into the Android framework to prevent such types of errors. It holds UI state, provides it to the Activity, and survives Activity being recreated on device rotation for example but it does not hold Activity context. This set of characteristics makes it a perfect tool for avoiding memory leaks in Android and practically the only modern standard.

It is worth noting that before the introduction of ViewModels developers have been using other architectural approaches and some third-party libraries but due to the lack of standardization and different implementations they often failed to prevent memory leaks and complicated understanding of the code in some situations when developers have chosen to implement their own solutions. ViewModels have largely reduced these problems and nowadays all modern Android applications make use of them.

4.3. Technologies

This section is aimed to describe technologies there were used in the project, why they were selected, and how they are implemented.

4.3.1. Kotlin

Kotlin is a programming language that is widely used in Android development. It is a recommended language by Google and receives support for all the modern features and functionalities. (JetBrains, 2023)

It is designed to be interoperable with Java and it is compiled into bytecode that runs on the same JVM. This means Java classes can be referenced from Kotlin classes and vice versa. This is especially useful if the developer is intending to use some older libraries that were written in Java or even implement some of the project classes using Java language because Kotlin allows to do this without any added overhead.

As for the language characteristics, it is very similar to Java in key aspects such as strong typization and compilation into bytecode but offers a more concise code style as well as some new features that are missing in Java.

One of those being null safety. By default, in Kotlin all the variables are non-nullable which means that they can’t have null value assigned to them and to assign null to a variable it needs to be marked as nullable. This greatly helps programmers to avoid NPEs in their applications, it doesn’t solve all the problems but is very useful.

In further sections, I will explain other features of the Kotlin language that are important in the context of this project. Only features that are used in the application code will be listed and for the full list of available functionalities, reader can refer to the official web resources.

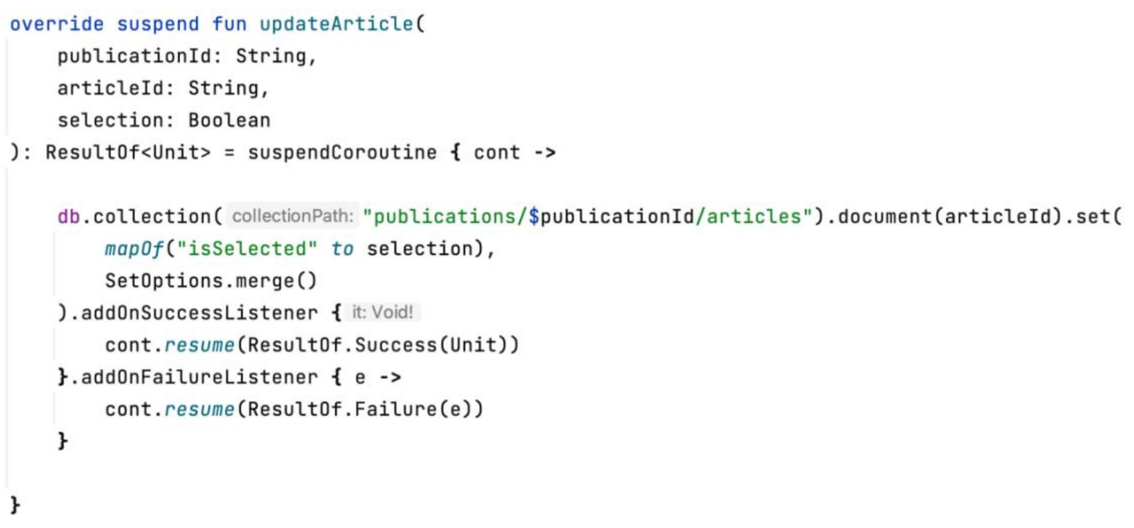
4.3.1.1. Kotlin Coroutines

Coroutines are essentially a Kotlin solution to writing asynchronous code that can run in parallel. Their implementation is vastly different from Java threads although internally it would compile to the same mechanisms. Coroutines are often presented as lightweight threads with the difference that the program can launch thousands of those without significant overhead. This is achieved by internally using executor pools.

A coroutine can be launched with the use of launch() or async() functions that accept lambda with user code. Inside this lambda developer can write code in a synchronous manner not worrying about run conditions. It is possible to configure this code to run on different executor pools using the withContext() function inside a coroutine. This will take pressure off the Main thread which is constantly occupied by drawing UI and cannot afford to pause and wait for I/O operations to complete.

An essential part of the coroutines is suspend functions, which are functions that can be paused and then resumed by passing around the Continuation object. They also provide ways to convert old-fashioned callback-based code to suspend function with special tools like suspendCoroutine() which accepts lambda with Continuation parameter.

In Figure 11 reader can observe an example of how a callback can be converted into suspend function. The updateArticle() function itself is marked with suspend keyword and its return type is the ResultOf object. Then it uses the wrapper function suspendCoroutine() which provides a Continuation object. This structure means that updateArticle() will only return when resume() is called on Continuation. Therefore, we can now setup callback listeners on the request to the database with addOnSuccessListener() and addOnFailureListener(), and once some result from those is generated it is passed to the Continuation object which concludes the execution of the function.



```kotlin
override suspend fun updateArticle(
    publicationId: String,
    articleId: String,
    selection: Boolean
): ResultOf<Unit> = suspendCoroutine { cont ->

    db.collection( collectionPath: "publications/$publicationId/articles").document(articleId).set(
        mapOf("isSelected" to selection),
        SetOptions.merge()
    ).addOnSuccessListener { it: Void!
        cont.resume(ResultOf.Success(Unit))
    }.addOnFailureListener { e ->
        cont.resume(ResultOf.Failure(e))
    }

}
```

Figure 11 Example of callback conversion to suspend function

One more aspect of Coroutines that wasn’t covered so far is that they need to be launched with CoroutineScope. Ensuring that the coroutine is tied to the lifecycle of ViewModel or Activity prevents memory leaks because the coroutine started from viewModelScope, for example, will be canceled on ViewModel destruction and its resources will be released. The activity also provides the instance of lifecycleScope to launch coroutines. It is as well possible to launch a coroutine from GlobalScope although it is not recommended to do so because the coroutine is tied to the lifecycle of the entire application and may occupy memory resources for longer than it is intended by design.

Overall coroutines provide very extensive coverage of all the needed functionality and are extensively used throughout the project.

4.3.1.2. Kotlin Flows

In the project flows are used to supply a sequence of values from the database to UI instead of just one value like suspend functions do. This is necessary to implement real-time updates, whenever a value in the database gets modified, a new value will be emitted into the flow and UI can collect it.

The operation of converting callbacks to the flow is somewhat similar to what happened with coroutines but instead of suspendCoroutine() we need to call the flow() function and inside of it new values can be provided using emit() function. As well as coroutine flows can be executed on different executor pools to lift pressure from the Main thread.

Then flow needs to be collected for the UI to update its state accordingly. Collect() is a suspend function so this operation must happen inside a coroutine. Assuming that the coroutine is launched correctly with Activity or ViewModel scope this mechanism provides built-in protection from memory leaks since the observer will unsubscribe from the flow once the coroutine is canceled.

Also, values in flows can be updated in the middle of the communication sequence, meaning that value from one or several flows from the database can get mapped to another value which the UI layer expects. This is a common strategy to generate a UI state based on observed values from the database.

In the project, most of the values that are received from the database are passed to UI using flows that enable real-time updates. Coroutines are more widely used for some singular operations like saving or updating some data when only a one-time response is expected.

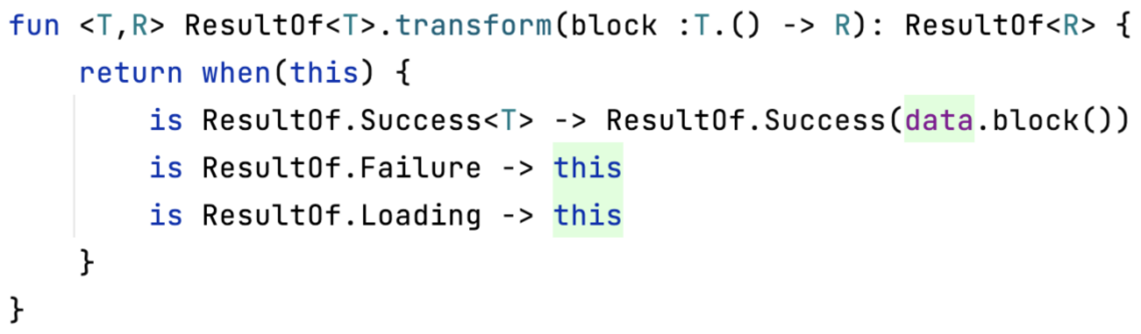
4.3.1.3. Data class type

It is worth highlighting that Kotlin also has several special class types, the one that is used a lot in the project is data class. It is different from the normal class because Kotlin will automatically generate getters and setters for all properties declared in the primary constructor. On top of this, it generates:

- equals() and hashCode() pair
- toString() in for format of “Article(title=MyTitle, charCount=1000, …)”
- copy() function that can be used to create a new instance of a given entity but with supplied parameters that should be different, the rest of the parameters are the same.
- componentN() which are used for destructuring declarations in Kotlin

4.3.1.4. Scope functions

Scope functions are special functions in Kotlin that allow changing the scope of **this** or **its** keywords. Their primary objective is to assist developers in writing concise and readable code. There are five of them that are available out of the box: **let**, **run**, **with**, **apply**, and **also**. They are useful for configuring objects or shortening some otherwise heavy language expressions, but the interesting part is that they do not use any special keywords or features and are written with the use of normal Kotlin expressions that are available for all developers. This allows for the creation of custom scope functions like reader can observe in Figure 12.



```
fun <T,R> ResultOf<T>.transform(block :T.() -> R): ResultOf<R> {
    return when(this) {
        is ResultOf.Success<T> -> ResultOf.Success(data.block())
        is ResultOf.Failure -> this
        is ResultOf.Loading -> this
    }
}
```

Figure 12 Example of custom scope function

In this example transform() function is declared on the custom wrapper class ResultOf which is used for communicating the state of I/O operations. Reader can see that the ResultOf class has a generic type and that the transform() works with two different generic types. The aim of the functions is to change the type of ResultOf class from **T** to **R** by applying some operation that is passed in the **block** parameter. The key part in this example is that the declaration of this operation
T.() ->R indicates that this operation will apply to type **T** and return type **R**. In the function, the body reader can see that this **block** operation is called on the property of ResultOf class with the use of **data.block()** syntax.

This allows the use of very concise syntax when mapping data from one application layer to another by calling the **result.transform { dataBaseModel** -> **UIModel(…) }** instead of writing repeatable chunks of code everywhere. Therefore, scope functions are quite useful tools and need to be considered when developing in Kotlin language.

4.3.2. Jetpack Compose

Jetpack Compose is a relatively new Android library developed by Google which is a replacement for the older XML view system. The main idea behind it is removing the need to set up UI through multiple files of XML and Kotlin code and declaring all the UI elements using only Kotlin code.

In practice, this greatly simplifies development since this new approach allows to omit a previous problematic need for binding Kotlin to XML. This allows to carry out complex initializations of lists in one file with minimal code when with XML system one simple list element could be scattered across multiple files making debugging and maintaining overly complicated.

In the application use of this new library allowed to create complex interfaces with animations and reuse multiple components by maintaining a set of independent UI elements that are ready to be used at any point.

Implementation of this library also requires following some code-style rules to ensure the best performance gains. For example, all the functions responsible for UI elements need to be free of side effects or idempotent to maintain a uniform state. In turn, there are special mechanisms for marking some variables to the UI system to be observed, when these variables change value application will redraw the screen with corresponding updates.

The process of redrawing is called recomposition. It is a complex algorithm that will try to redraw the minimal amount of UI elements that depend on the variable that was changed to maximize the performance. This also encourages developers to avoid any I/O or long-running operation in the scope of recomposition since it needs to be run frequently and thus be very efficient to deliver the best performance.

Overall, the use of Compose library ensures that the developed solution is future-proof because it is now the recommended way of developing Android applications and will be receiving further extensive support and improvements. Also, as a summary of this section, it can be said that Compose library surpasses XML views in many ways and has helped a lot in writing readable and reusable code that doesn’t create any unwanted side effects and thus can be easily maintained.

4.3.3. Navigation component

Navigation has always been an important topic in Android development and while there have been some proprietary solutions that add abstraction layers between application navigation system and giving Android exact orders for navigations, those solutions have not been widely adopted. The introduction of an official navigation library has solved a lot of the pressing issues and introduced a single standard to be implemented in all the applications.

This navigation library has also been adopted for the use with Compose UI toolkit. It is a primary solution for navigation in the app and while work is still going on to introduce some more advanced features it already covers all the necessary functionality.

The core element of the library is NavHostController. It can be seen as the navigation entry point and all the operations of changing screens pass through this class. For it to know how to process some requested destinations NavHostController has to be supplied with a navigation graph where all the destinations are registered and entry points for drawing corresponding screens are provided.

In its previous XML implementation developers has access to the interactive navigation graph builder but at the moment it is removed from Compose version and the graph needs to be constructed using Kotlin language. Knowing the issue that it might grow unmanageable in case the application has a lot of screens, there are convenient solutions available for splitting up a single graph into multiple subgraphs divided by some logical factors. This makes navigation code maintainable while also keeping the benefit of the single-entry point since all the subgraphs should get registered in the main graph to be properly set up.

The use of NavHostController has enabled efficient navigation in the application which is easy to adjust to the business needs without working directly with orders to the Android system and avoiding possible drawbacks or complications.

4.3.4. Hilt

The hilt is used as a library for dependency injection which is further referred to as DI. This is a programming approach to supply classes in the app hierarchy with their respective dependencies. (Android Developers, 2022)

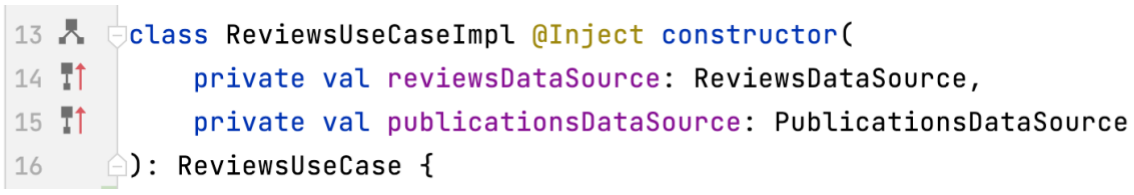
We can consider an example of how classes of use cases implementation need actual instances of data sources to work with although we only want them to be aware of interfaces of these data sources and be independent of actual implementations. If there was no DI implemented in the project use cases would need to create objects of data sources themselves and thus depend on those classes. This is undesirable behavior therefore these dependencies are supplied in the form of parameters in the constructor and use cases can only know about the interface they are using but not the implementations.

Now when the need for DI is justified there are a couple of ways of implementing it. One possibility is doing it manually, but the downside is that developer should create and manage all the code responsible for supplying instances of classes in the project himself, and in a growing codebase this can become very problematic since dependency graphs keep growing and getting unmanageable.

Another approach that is implemented in Android is just to request the needed dependencies from other classes for example Context class has a special function getSystemService() that can provide various system services to the application. This is convenient in terms of working with Android APIs but in terms of manual implementation would still pose the same complexity issues as the first solution.

The third approach is using a special library for DI in this case Hilt was selected as the latest and recommended one. It also has integrations with Android Studio that help to trace dependencies in the user interface of the IDE.

To enable Hilt constructors in the same example of use cases classes need to be marked with special **@Inject** annotation and respective dependencies need to be bound to the Hilt graph with the use of **@Binds** or **@Provides** annotations. An example of such a setup can be observed in Figure 13.



```
13 class ReviewsUseCaseImpl @Inject constructor(
14     private val reviewsDataSource: ReviewsDataSource,
15     private val publicationsDataSource: PublicationsDataSource
16 ): ReviewsUseCase {
```

Figure 13 Example of DI using Hilt

As reader can see in this example ReviewUseCaseImpl receives its dependencies in the constructor only rereferring to the interfaces and any implementation can be supplied. Also, the user can click the icon on line 13 to navigate to where ReviewUseCaseImpl itself is used, or click icons on lines 14 and 15 to navigate to from where these dependencies are supplied.

Overall Hilt library provides simple tools for managing otherwise complex problems of DI and is an essential instrument for developing modern Android applications.

4.3.5. Firebase

Firebase functionality was already mentioned and explained to the reader in previous sections. One thing to point out is that Firebase consists of many components and only three of them are used in the project: Cloud Firestore, Firebase Storage, and Firebase Authentication.

4.3.6. Other libraries

As for other third-party libraries project includes Coil which is used for loading images from URL resources into UI. Internally it provides caching to optimize performance and it also was developed for Compose so that it fits perfectly into the chosen architecture.

Another library is PDF Viewer which is used to display pdf documents to the user. There is no native solution for this through Compose or XML views, so usage of external libraries is required to avoid using low-level tools for this task.

This covers all the optional libraries that were used. In project has more dependencies developed but they are a part of the standard Android toolkit and are assumed by default in any project.

4.3.7. Android Studio

IDE that was used for the development of the solution is Android Studio it is based on IntelliJ Idea but it is distributed free of charge and provides an advanced set of features specifically for Android development.

It supports both Java and Kotlin languages and allows for traducing Java code to Kotlin. It has integrations with VCS and can be modified with many available plugins.

One of the main features that help the developer is the emulator that can be configured to represent a large variety of physical devices as well as any version of Android OS. Apart from this Android Studio offers a set of special device inspectors that allows one to view database contents, network requests, or resource usage to analyze the performance of the application and identify any potential bottlenecks or errors.

Android Studio also comes with Gradle which is responsible for managing dependencies and building APKs. It can also perform all the necessary signing steps for an APK to be submitted to Google Play Market.

Overall, this IDE is a crucial piece of software for Android development and covers all the needed functionality to be able to efficiently develop applications.

4.3.8. GitHub

GitHub was chosen as a version control system. It allows to create of public repositories for free thus allowing for storing code and maintaining version history. If needed automated scripts can be configured to run Gradle for APK assembly, running tests, or distributing application versions.

4.3.9. Android

Android is an operative that was designed by Google for use in mobile devices such as phones, tablets, TVs, wearables, etc. (Android Developers, 2023) It is based on Linux thus it is distributed for free. Many manufacturers prefer to supply devices with slightly modified versions of Android which they can develop without any problems due to the policy of Android OS distribution. However, it is a disputed topic since it creates security issues such as receiving crucial updates later than the official version of Android or various issues that tend to reproduce only on specific versions of Android of some manufacturers.

This introduces an inconvenience for developers since the range of possible devices and their OS versions is much wider than on IOS for example. Therefore, to properly test some device-specific features it is recommended to run the same test on a set of different devices to ensure correct functionality across the whole specter of options.

In terms of Android architecture, it provides an abstraction layer over device hardware and allows for smooth communication with the device. Also, OS ensures that the application has the respective permissions to access the hardware. Android comes with a set of native C++ libraries and Davlink Virtual Machine (DVM) which is an Android version of JVM optimized for running on mobile devices.

Overall Android operating system remains one of the most accessible and secure systems for users to work with as long as they keep their versions of the OS up-to-date and follow basic security guidelines. Thus, choosing to develop an application for this system enables the maximum number of users to download and utilize it.

5. Jetpack Compose research

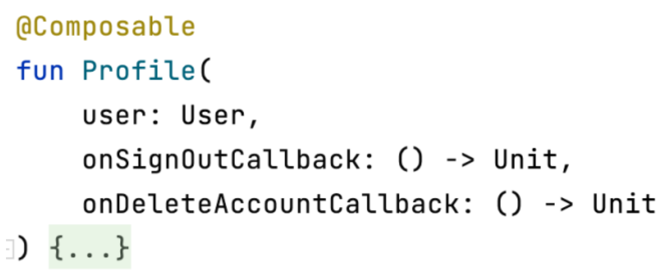
This section is intended to cover some of the knowledge about Jetpack Compose that was gained during the development process. Since the topic of this work largely relies on research on Compose, reader will be able to learn more about it and how some of the functionalities are implemented as well as the benefits it provides over XML technology.

First of all, it is important to understand that all the UI elements are declared by use of functions annotated with **@Composable,** further composables. Composables can be nested within each other and reused. These allow to create complex interface elements and reuse them throughout the app. As was already mentioned earlier composables should be idempotent and ideally only receive state as a parameter, avoiding performing any other calls on objects like ViewModel for example. This allows developers to easily reuse the elements and also write tests for them without the need to mock complex objects. (Android Developers, 2022)

Obviously, UI should not only display information but also react to user input, to achieve such behavior Android uses a system of click listeners which accept callbacks and activate when a user performs the corresponding action. (Android Develoeprs, 2023) It is recommended to pass the callbacks to lower lever composables to avoid their direct interaction and dependency on higher-level objects.

Another detail about composable elements is that they can receive a special configuration object called Modifier. It can perform various functions from declaring the size of the element to enabling animations or setting click listener.

To sum up, this explanation of composable functions reader is presented with an example of such a composable that receives a simple state value, the callback for user interaction, and a modifier so that it can be adjusted for reuse. An example can be observed in figure 14. Profile composable is responsible for displaying user profile information that it receives in **user** parameter. Also, it receives two callbacks to be hooked up to corresponding buttons on the profile screen. The function body is omitted since it would provide too much irrelevant information to the reader, but the general idea is that lower-level composable **Text** will be displayed with the corresponding user parameters supplied to them.



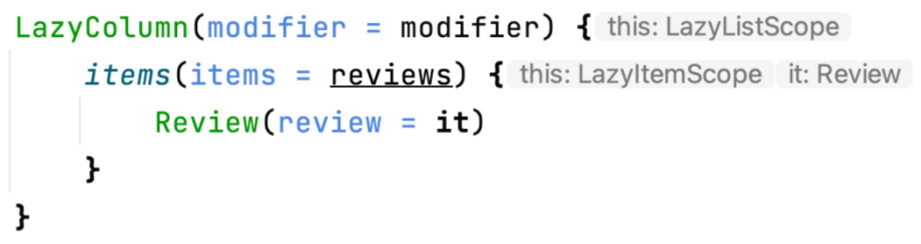
```
@Composable
fun Profile(
    user: User,
    onSignOutCallback: () -> Unit,
    onDeleteAccountCallback: () -> Unit
) {...}
```

Figure 14 Example of composable function

At this point, reader can understand the basic concepts of Compose but some of the benefits remain unclear. To provide a more practical example reader will be presented with an explanation of how to display a list of items in composable. To add some context, a list of items is a very common element in mobile development since it is the best way to display large chunks of data to the user while keeping all the individual data of each item available. In XML developer would need to create at least two classes and maintain at least three separate files like it is demonstrated in official examples. (Android Developers, 2023)

Partially this problem is caused by performance complications. Processing all the data at once and doing calculations for items that are not on the screen would result in huge performance hits, so all this infrastructure is aimed to optimize the process and where possible reuse drawn items.

Compose offers a much simpler way of handling the issue with the use of Lazy(Grid/List/Column) composables. These are supplied in the form of single functions accepting several configuration parameters and a trailing lambda where the developer is expected to introduce a list of items that are to be displayed. An example can be seen in figure 15.



```
LazyColumn(modifier = modifier) { this: LazyListScope
    items(items = reviews) { this: LazyItemScope  it: Review
        Review(review = it)
    }
}
```

Figure 15 Example of LazyColumn usage

On figure 15 reader can see that LazyColumn composable receives a modifier from some outside configuration. Then, in the following lambda, it calls the **items()** function which receives a list of data, and also another trailing lambda where it expects instructions on how to draw each single item. This effectively reduces all the boilerplate code from XML to a couple of lines while taking care of performance issues as well.

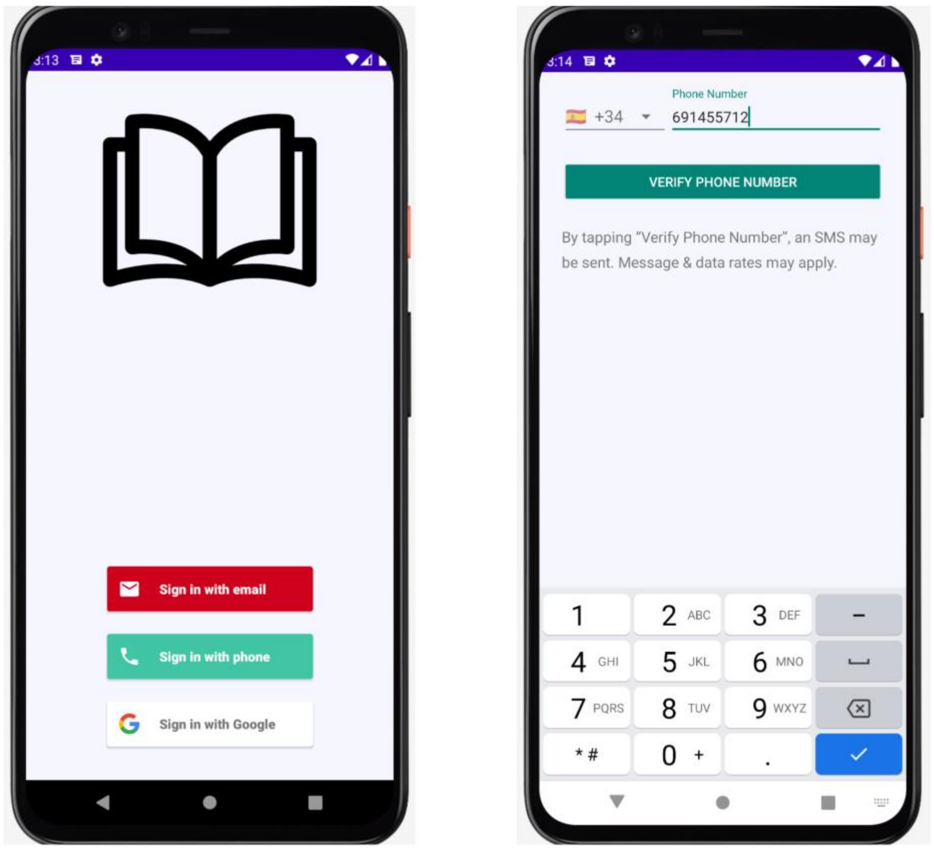
It can now be understood that lists in Compose significantly improve code quality and allow one to do the same amount of work much faster than in XML as well as implement new features or introduce complex changes easier. Since lists are a very common UI element, the speedup tends to accumulate with the project growth which results in a huge benefit of Compose over XML in practically any Android application.

6. User manual

This section is supposed to present a user manual and provide an understanding of how the application is supposed to be used and how defined use cases have been implemented. The content will follow the general structure of use cases that were presented in section three.

6.1. Common features manual

First of all, it will be presented how user can authorize the application both using an existing account or creating a new one.

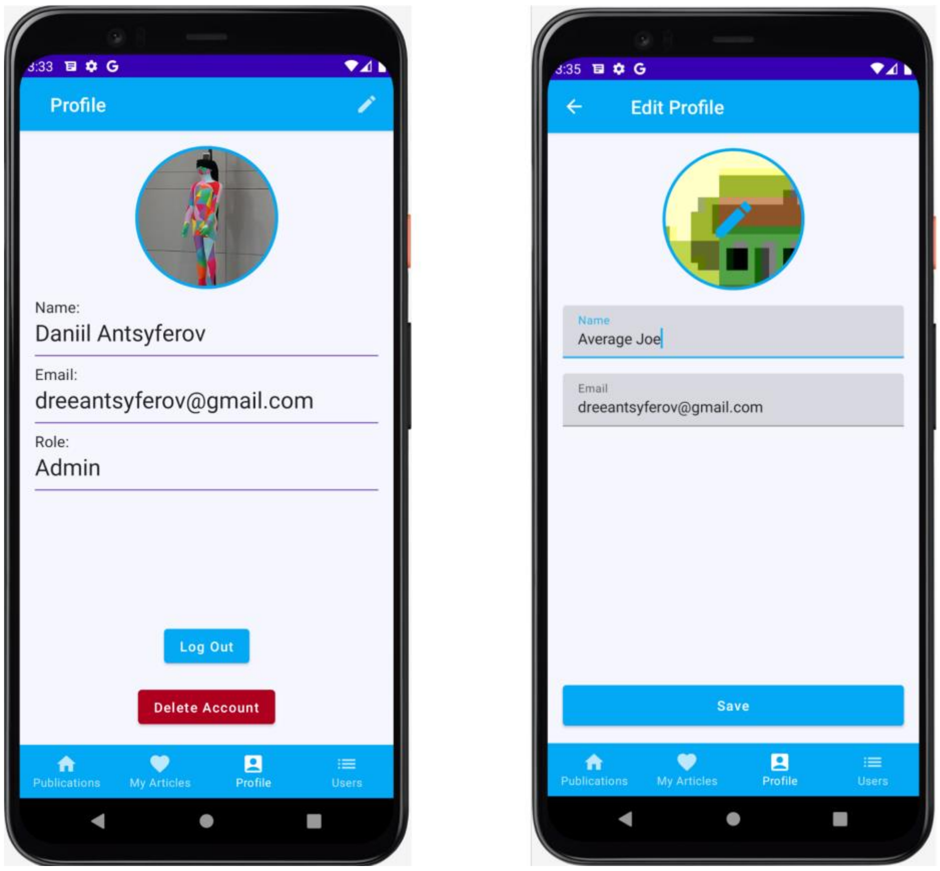


By tapping "Verify Phone Number", an SMS may be sent. Message & data rates may apply.

Figure 16 User authorization screens

Reader can see in figure 16 that user is presented with a login screen when the application is opened, and the system detects there is no active session. User can choose one of three authorization methods: email, phone, or Google account. By clicking on one of them user will navigate to the corresponding input field where he will be asked for login data and if it is not registered in the system, new account will be created. The screenshot on the right corresponds to the phone input where user is supposed to enter his mobile number.

In continuation, it is presented how users can manage their profile information.



Name:
Daniil Antsyferov
Email:
dreeantsyferov@gmail.com
Role:
Admin

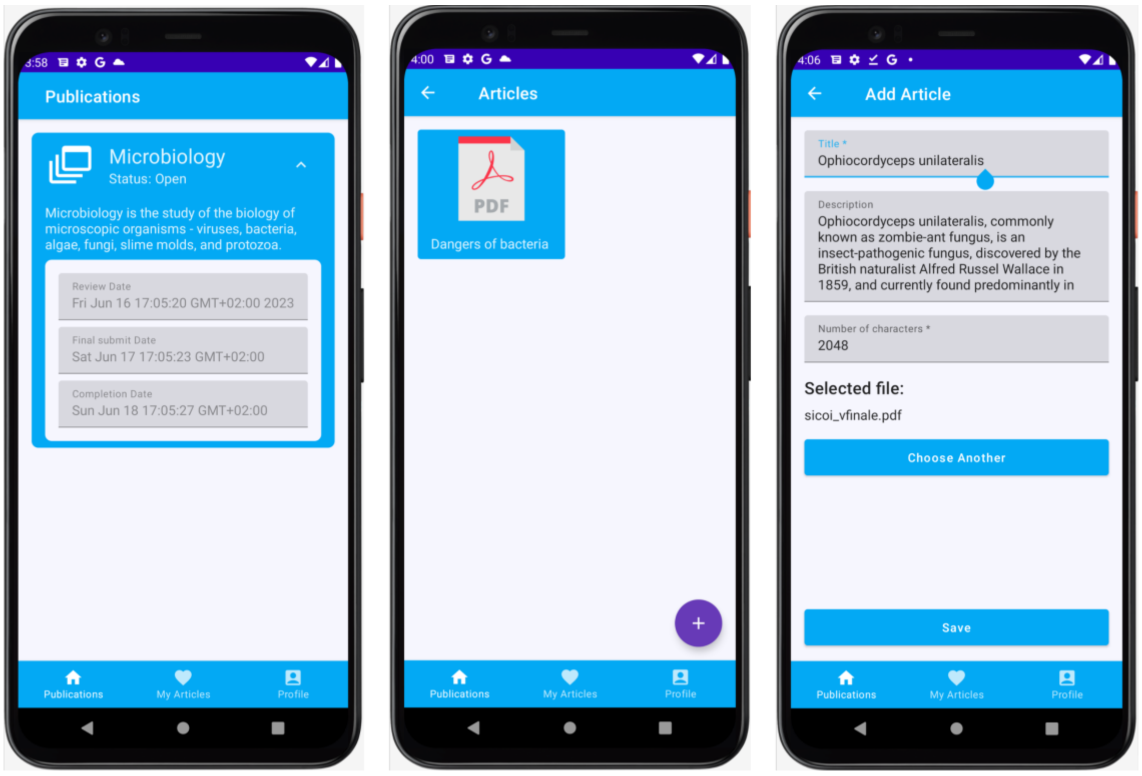
Name
Average Joe
Email
dreeantsyferov@gmail.com

Figure 17 Profile screens

On figure 17 two profile screens are presented. The one on the left displays available profile data and can be accessed via the corresponding bottom navigation tab. The screen also has “Log Out” and “Delete Account” buttons which perform corresponding actions. To edit profile information user can click the icon in the top right which will bring them to the screen on the right. User can change their name and update their associated email address. Also, the user can upload a new avatar image by clicking the edit icon in the middle of the current avatar image circle which will open a system application that allows for image selection. To save newly made changes user should click the “Save” button which will as well navigate them to the left screen and display updates.

6.2. Author manual

As for author functionality at first, it will be presented how an author can submit an article.



Microbiology
Status: Open

Microbiology is the study of the biology of microscopic organisms - viruses, bacteria, algae, fungi, slime molds, and protozoa.

Review Date
Fri Jun 16 17:05:20 GMT+02:00 2023

Final submit Date
Sat Jun 17 17:05:23 GMT+02:00

Completion Date
Sun Jun 18 17:05:27 GMT+02:00

Dangers of bacteria

Title *
Ophiocordyceps unilateralis

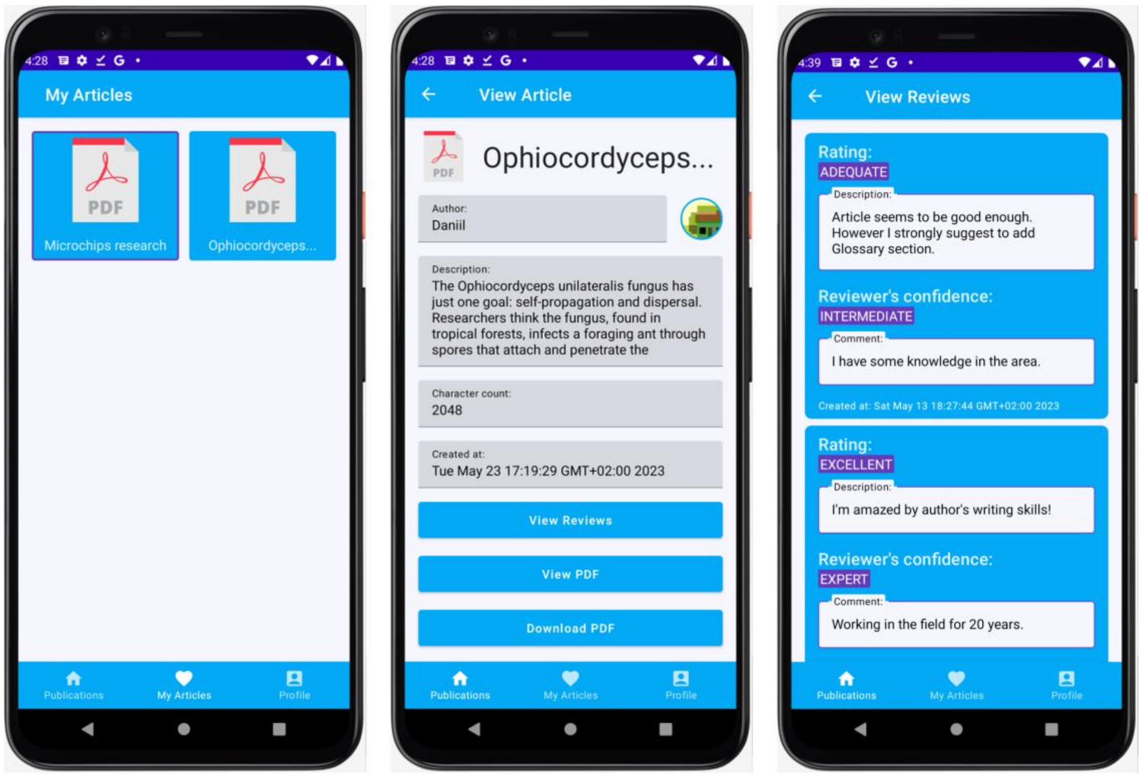
Description
Ophiocordyceps unilateralis, commonly known as zombie-ant fungus, is an insect-pathogenic fungus, discovered by the British naturalist Alfred Russel Wallace in 1859, and currently found predominantly in

Number of characters *
2048

Selected file:
sicoi_vfinale.pdf

Figure 18 Article submission screens

Figure 18 features three screens. The first one is where author can select an available track and consult a general description of the topic alongside deadlines that participants are expected to meet. After selecting the track author is navigated to the second screen with the list of articles that they have already submitted for the given track. The second screen also has a “+” button at the bottom which will lead the author to the third screen. The third screen allows to submit a new article. It has input fields available for entering article information and allows author to select a pdf file from his device that will be uploaded to the server. Clicking the “Save” button navigates the user to the previous screen and creates a new article that will be displayed accordingly.



Ophiocordyceps...

Author:
Daniil

Description:
The Ophiocordyceps unilateralis fungus has just one goal: self-propagation and dispersal. Researchers think the fungus, found in tropical forests, infects a foraging ant through spores that attach and penetrate the

Character count:
2048

Created at:
Tue May 23 17:19:29 GMT+02:00 2023

Rating:
ADEQUATE
Description:
Article seems to be good enough. However I strongly suggest to add Glossary section.
Reviewer's confidence:
INTERMEDIATE
Comment:
I have some knowledge in the area.
Created at: Sat May 13 18:27:44 GMT+02:00 2023

Rating:
EXCELLENT
Description:
I'm amazed by author's writing skills!
Reviewer's confidence:
EXPERT
Comment:
Working in the field for 20 years.

Figure 19 Evaluation list screens

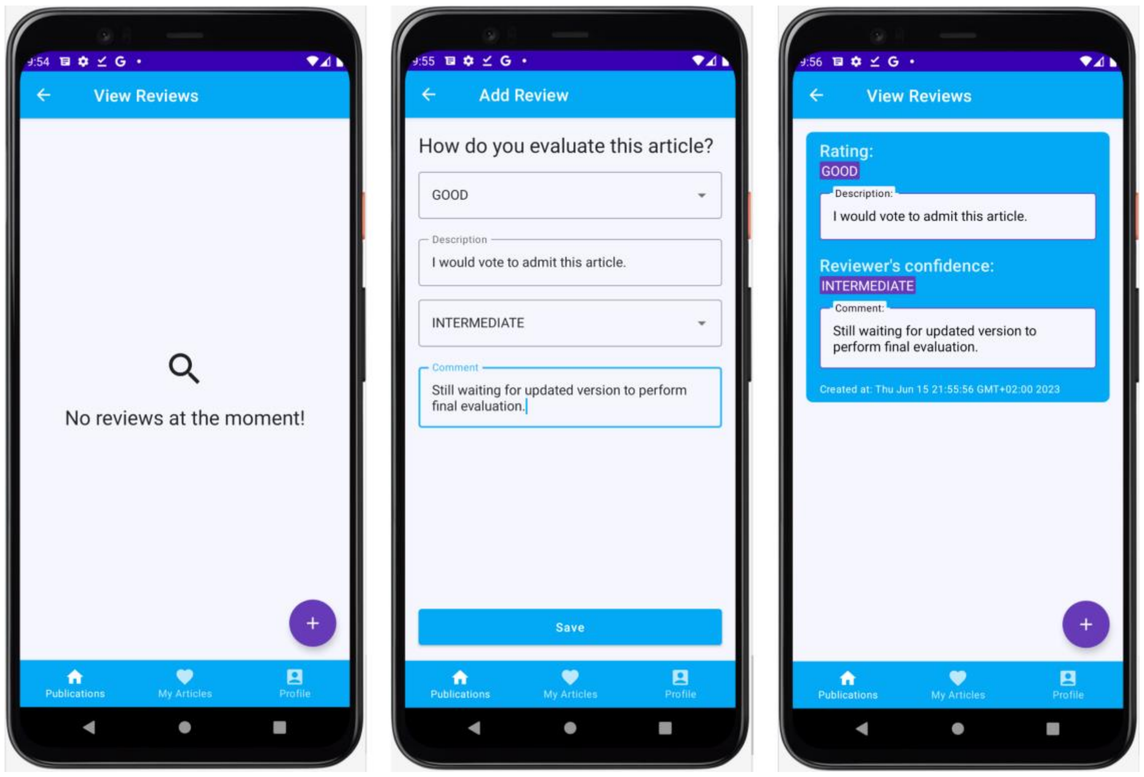
Figure 19 presents three more screens to the reader. The first one contains a list of all the articles an author has submitted. Reader can notice a purple border around the first article on the screen, which means that the article has been admitted by the administrator of the congress.

The second screen presents a detailed view of an article and allows to preview PDF or download it to the device. It also provides navigation to the third screen.

On the third screen, the author can access a list of reviews that were provided for his article. Apart from the evaluation information author can see the date when the review was carried out but cannot see the author of the review as it was defined in the requirements.

6.3. Reviewer manual

This section describes how a reviewer can add an evaluation.



No reviews at the moment!

How do you evaluate this article?

GOOD

Description: I would vote to admit this article.

INTERMEDIATE

Comment: Still waiting for updated version to perform final evaluation.

Rating: GOOD

Description: I would vote to admit this article.

Reviewer's confidence: INTERMEDIATE

Comment: Still waiting for updated version to perform final evaluation.

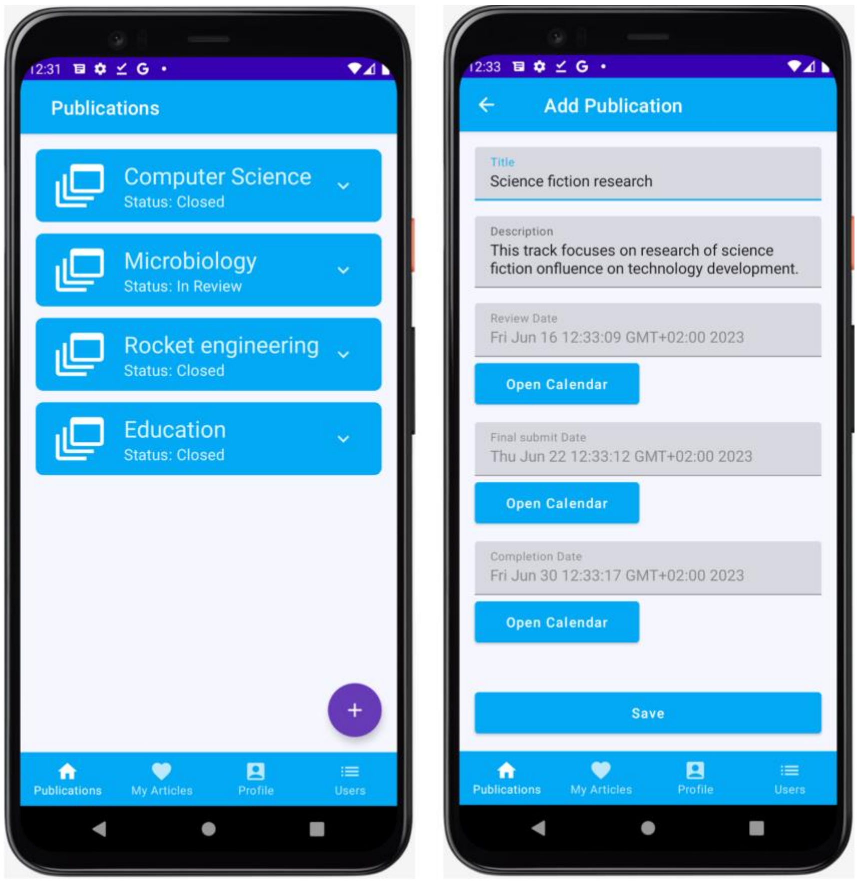
Created at: Thu Jun 15 21:55:56 GMT+02:00 2023

Figure 20 Add review screens

On figure 20 reader can once again observe three screens. The path to the first one would be the same as for an author, however, reviewers will only be able to see evaluations they added themselves and not ones added by others. To proceed to the second screen reviewer would click the "+" button in the bottom left corner. On the second screen user can fill in all the fields to formalize his evaluation and click "Save" to publish the results. It will also automatically navigate the user to the previous screen but this time displaying his newly added review. Reviewer can add as many evaluations as he likes but they all are going to be recorded and the corresponding date saved.

6.4. Admin manual

This section describes administrator functionalities.



Computer Science
Status: Closed

Microbiology
Status: In Review

Rocket engineering
Status: Closed

Education
Status: Closed

Title
Science fiction research

Description
This track focuses on research of science fiction onfluence on technology development.

Review Date
Fri Jun 16 12:33:09 GMT+02:00 2023

Final submit Date
Thu Jun 22 12:33:12 GMT+02:00 2023

Completion Date
Fri Jun 30 12:33:17 GMT+02:00 2023

Figure 21 Add track screens

Figure 21 presents the administrator’s overview of all available tracks and a screen for creating a new track. To create a new track admin should first click on the “+” button, then fill out all the necessary information and finally click “Save”.

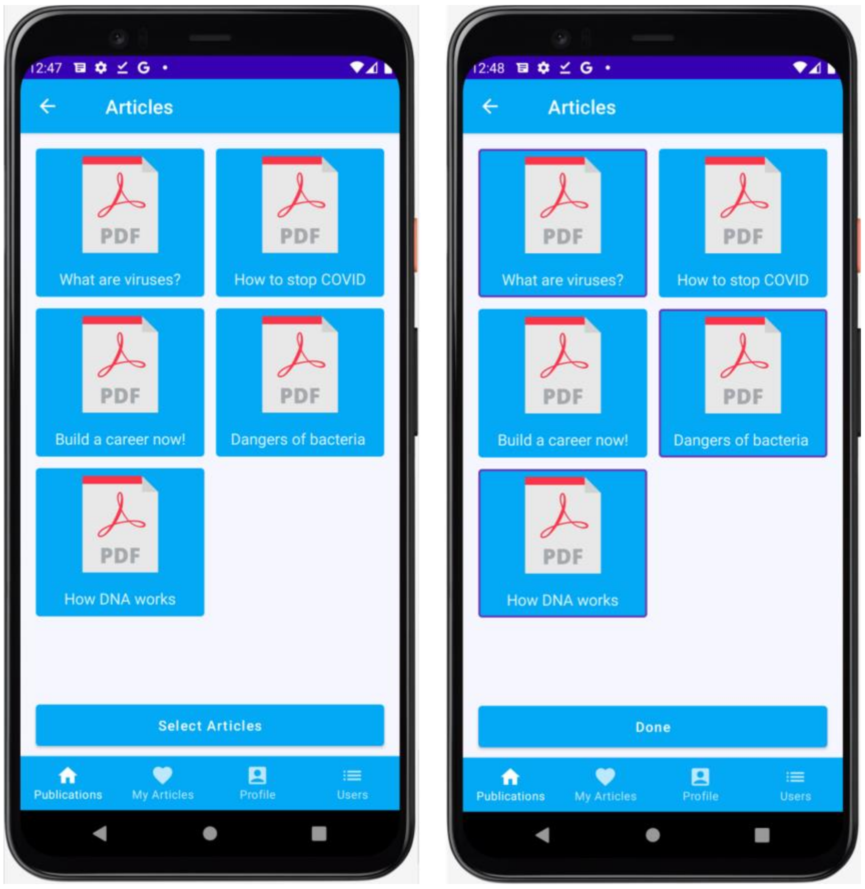


Figure 22 Article selection screens

In Figure 22 reader can observe the admin's view on all the articles in the track and a "Select Articles" button. By clicking it admin can activate a special selection mode on this screen which allows to mark articles that are admitted. This mode is available after the last deadline assigned to the given track has passed. Admitted articles are marked with a purple border. To save the results administrator can click the "Done" button which will push updates and make the results available to authors.

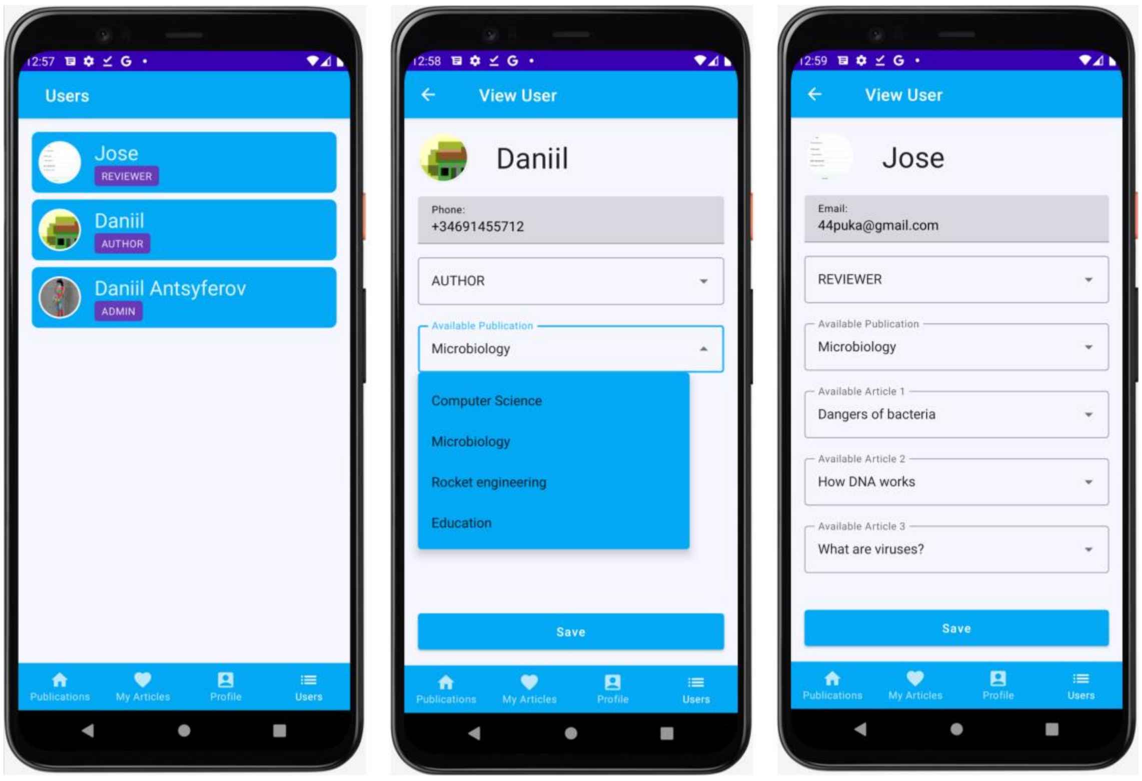


Figure 23 User configuration screens

The administrator also has access to a special users' configuration tab in the bottom navigation that is featured in figure 23. The first screen presents a list of users and their roles, by clicking on a user, the admin can navigate to their detailed information screen. On this screen, the administrator can select a user role and give access to submitting articles to the given track for authors and adding evaluations for selected articles for reviewers. This is achieved by interacting with presented dropdown menus. Finally, by clicking "Save" the administrator will push the updates.

To sum up this section, the application has a lot of different screens, but they tend to reuse a lot of the same elements which helps users to interact with the application. When user intends to perform a new action, they have already interacted with most of the elements and are not getting confused by the interface thus not wasting time on understanding the whole layout from zero.

Also, the application uses a lot of list elements. This is a convenient way to display a lot of information on the screen and will be useful with more realistic examples when the number of articles can be much higher than presented on the screenshot.

Finally, the decision to intergrade the bottom navigation tab has helped to solve the problem of complex navigation and give users some indication of where they currently are in the app navigation system as well as enabling users to quickly switch between contexts of different tasks and uses cases.

7. Future work

This section explains possible future modifications and extensions that could be implemented in the project. The context of the current system is that it is in the early stages of its lifecycle and in case the concept proves itself successful possibilities of further work will be explored.

The first and most obvious possibility would be expanding the project to IOS and/or web platforms. This would considerably increase the possible user base and allow more authors to take part in congress. However, from the professional point of view such expansion wouldn't contribute much to the work since the system design would remain the same and it would only increase the scope of work and amount of code to maintain in the future.

The second option that could be explored is replacing Firebase with a relational SQL database. In the future, this would allow us to implement more complex designs and better support existing functionalities. Retrospectively Firebase has worked quite well in this project, but one can see how the complexity of the solution and lack of complex SQL queries could handicap the project in the future. Thus, looking into SQL databases would be a valid suggestion if more project advancements are planned.

The third suggestion that could be made is to take the project into the direction of personal assistant which would mean adding some optimization algorithms so that application could provide suggestions on which articles to admit. For example, if administrator were to evaluate submitted articles based on a larger number of parameters, the final decision could be difficult to make. Application in its turn could do suggestions on which articles to choose or even try to find an optimal solution. However, before this idea is accepted potential value of such modification should be considered. In the current state of affairs, the size of the optimization problem would be way too small, and it wouldn't make sense to implement any complex solutions. Also, it would be hard to perform relatable studies of obtained benefits over brute force or simplest algorithms. Therefore, this idea sounds promising but requires specific conditions to be valuable enough in the scope of the project.

Finally, as the reader can see there are a lot of possibilities to explore when deciding where to take this work in the future. In my opinion, the decision should be made based on real pressing needs which can be identified by giving users to try out the application and analyzing the feedback. After more data is available it will be easier to make an informed decision. As for now, it is also useful to understand what adjustments are possible and where possible make some effort to put in the foundation for those future expansions.

8. Conclusions

The main objective of this project was to create a system design and implement the actual software that would allow users to manage and participate in the conference. All the functional requirements have been delivered.

Apart from functionality, there were additional requirements set such as scalability and flexibility of the software. These have been also achieved based on implemented database capabilities. In case the number of users outgrows Firebase resources, the app accounts for easy migration to any other API. This approach of covering the immediate needs and preparing for future extensions is a balanced solution for the project in the initial stages of business development.

During this work, I have relied on knowledge and skills acquired from my university studies as well as on previous experience with Android development which I learned independently. I have also developed new skills by learning new technological approaches and successfully incorporating them into the application.

Apart from the above, I have been able to establish an agile work process using techniques like Kanban and available tools like Trello. The project was delivered according to the established timeline with the tutor and regular reports on the progress have been provided. This ensured a balanced rate of development and effort that was put in and overall resulted in better application than it would be with a less organized approach.

To sum up, all the initial goals of the project have been achieved, adaptation of new technologies went smoothly, and development processes have been set up adequately. I have learned how to use new tools, mastered things I already knew, and acquired the necessary skills for further professional work.

9. Bibliography

Android Developers, 2023. *Create dynamic lists with RecyclerView.* [Online]
Available at: https://developer.android.com/develop/ui/views/layout/recyclerview
[Accessed 20 May 2023].

Android Developers, 2022. *Build local unit tests.* [Online]
Available at: https://developer.android.com/training/testing/local-tests
[Accessed 13 June 2023].

Android Developers, 2022. *Dependency injection in Android.* [Online]
Available at: https://developer.android.com/training/dependency-injection
[Accessed 5 February 2023].

Firebase, 2023. *Cloud Firestore Data model.* [Online]
Available at: https://firebase.google.com/docs/firestore/data-model#kotlin+ktx_3
[Accessed 10 February 2023].

Gamage, T. A., 2022. *Understanding Java Memory Model.* [Online]
Available at: https://medium.com/platform-engineer/understanding-java-memory-model-1d0863f6d973
[Accessed 10 June 2023].

Android Developers, 2022. *Guide to Android app modularization.* [Online]
Available at: https://developer.android.com/topic/modularization
[Accessed 4 February 2023].

Android Develoeprs, 2023. *Input events overview.* [Online]
Available at: https://developer.android.com/develop/ui/views/touch-and-input/input-events
[Accessed 15 June 2023].

JetBrains, 2023. *Kotlin docs.* [Online]
Available at: https://kotlinlang.org/docs/home.html
[Accessed 12 February 2023].

Android Developers, 2023. *Platform architecture.* [Online]
Available at: https://developer.android.com/guide/platform
[Accessed 25 May 2023].

Android Developers, 2023. *Why adopt Compose.* [Online]
Available at: https://developer.android.com/jetpack/compose/why-adopt
[Accessed 25 January 2023].

Robert, M. C., 2018. *Clean Architecture: A Craftsman's Guide to Software Structure and Design.* s.l.:Prentice Hall.

Gamma, E., Vlissides, J., Helm, R. & Johnson, R., 1995. *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l.:Addison-Wesley Publishing Company.

Bittner, K. & Spence, I., 2002. *Use Case Modelling.* s.l.:Addison-Weasley Publishing Company.

Rehkopf, M., 2023. *What is a kanban board?.* [Online]
Available at: https://www.atlassian.com/agile/kanban/boards
[Accessed 5 March 2023].

Annex A

Sustainable development goals

Sustainable development goals	High	Medium	Low	Not relevant
ODS 1. **No poverty.**				**X**
ODS 2. **Zero hunger.**				**X**
ODS 3. **Good health and well-being.**				**X**
ODS 4. **Quality education.**		**X**		
ODS 5. **Gender equality.**			**X**	
ODS 6. **Clean water and sanitation**				**X**
ODS 7. **Affordable and clean energy.**				**X**
ODS 8. **Decent work and economic growth.**				**X**
ODS 9. **Industry, innovation and infrastructure.**				**X**
ODS 10. **Reduced inequalities**.		**X**		
ODS 11. **Sustainable cities and communities .**				**X**
ODS 12. **Responsible consumption and production.**			**X**	
ODS 13. **Climate action.**				**X**
ODS 14. **Life below water.**				**X**
ODS 15. **Life on land.**				**X**
ODS 16. **Piece justice and strong institution.**				**X**
ODS 17. **Partnerships.**				**X**

Among the goals mentioned above I believe this project is correlated with the following:

- **Reduced inequalities:** The developed solution allows any author to take part in the congress and to receive a fair evaluation of the submitted articles. In my opinion, this promotes the principles of equality given that any social or economic differences are obscured during the process of evaluating and selecting articles. This puts all the participants in an equal position and ensures that the best articles are admitted regardless of the position of their author.

- **Quality education:** The project has a connection to this goal based on the promotion of the selection of the best works. This provides tools to hold a fair congress and as a result, get the most suitable articles which can provide useful information to the students. Also, it is worth noting that in case congress is held in the context of competition between students, the system would also encourage submitting and evaluating each other's articles allowing to keep students engaged and committed to the objectives of their learning program.

- **Gender equality:** In the context of this work this goal is very similar to the goal of reducing inequalities. People participating in the congress are evaluated based on their submitted work and gender is not indicated and not regarded as a parameter for evaluation.

- **Responsible consumption and production:** This goal is pursued by the fact that a lot of paperwork routine that can be often associated with such types of congresses is substituted by mobile applications. Neither administrators nor participants do not need any additional resources to participate. Any need to print out every article and maintain a complex flow of paper documents is redundant in the scope of this project thus encouraging responsible consumption of paper and other materials.

Annex B

UML notation

This annex provides information to the reader on elements of UML notation that were used in the diagrams throughout this work. This is not a complete catalog of all possible notations, but a quick overview aimed to provide context and clear any questions that reader might have regarding other figures featuring UML.

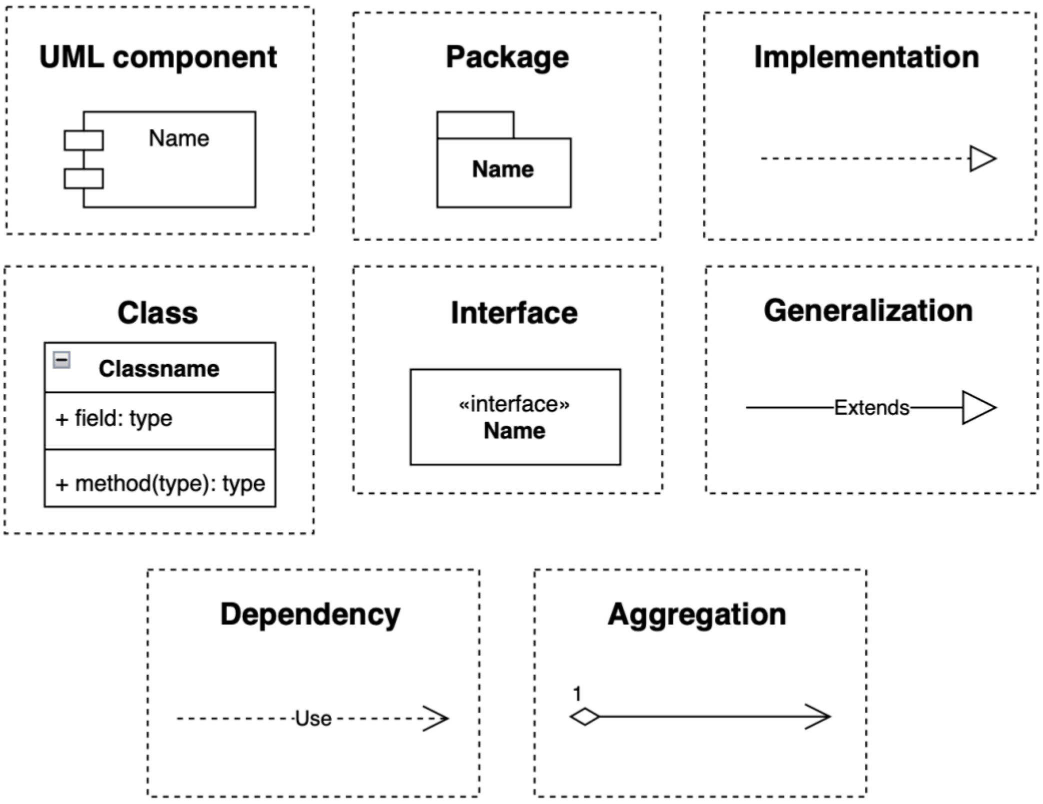


Figure 24 UML elements

Figure 24 presents all the UML elements that were used in other diagrams. In continuation a brief description of each one is provided:

- UML component: a modular self-contained unit that usually encapsulates some specific functionality or feature.
- Package: an organizational unit aimed to group classes in a cohesive manner to better structure the code.
- Class: representation of an object with its properties and methods.

- Interface: a contract that defines methods to be implemented by a class.

- Generalization: a relationship between two classes where one child class inherits its properties from the parent class.

- Implementation: a relationship between a class and an interface where the class provides a realization for the behavior defined in the contract

- Dependency: a relationship between two classes where the one class may be affected by the changes in seconds class.

- Aggregation: a relationship between two classes where one class is composed of or contains an instance of another class, however, another class can exist on its own.