Software Design Documentation
for the mobile application
“Habit tracker”

Kamila Khamidullina, Makshe Seytkaliev, Dias Usenov

1. Glossary

- ➔ Goal - some aim that the user wants to achieve.
- ➔ Progress - visualizing steps performing.
- ➔ Reward - rating which user can get for successful step on lose for failing the step.
- ➔ Steps - daily stage in a way to achieve the goal.
- ➔ User - someone who uses the app.

2. Stakeholders

Stakeholder	Roles	Responsibilities
Developer	Front-end developer Back-end developer Designer Database administrator Content provider	Create a design of a mobile app Create the front-end Create the back-end Create the database, fill it with goals samples Test and release the product Writing clear documentation for the project
User	Android app user	Use the application
Owner	Customer of the product	Giving the task, preferences about the project Accepting/rejecting the project

Table 2.1 Stakeholders

3. Concerns

A. Features

Must have	Should have	Could have
Set a new goal to achieve	Authorization using mobile phone or email	Share the progress in social media
Send notifications about daily steps	The ability to check the progress on the goal (previous, current and future steps, failures and successes)	Reward steps in a way to goal achieving
Giving up achieving the goal	The failure of a daily step	

Table 3.1 Features

B. Functional requirements

Use Cases

Feature	Primary Actor	Basic flow	Extensions
Set a new goal to achieve	User	1. The User presses a button "*Set a goal*". 2. The System offers to choose one item from the list of "*Bad habits*" or "*Good habits*" 3. The User chooses the start date.	1. The System does not have a necessary goal in lists. a. The User chooses option "*Other*" b. The System shows a textbox to enter the goal. c. User enters.
Send notificatio ns about daily steps	System	1. The system sends notifications about the user's current progress.	1. The user does not want to get notifications. a. The user can turn off the notifications. 2. The user does not have goals. a. The system does not send notifications.

Authorization using mobile phone or email	User	1. The user wants to sign up using a mobile phone or email. 2. The system shows the sign up form. 3. The user fills the form. 4. The system registers the new user.	1. The user filled in the incorrect mobile phone or email. a. The system shows the error. b. The user tries to fill it again. 2. Entered mobile phone or email are already registered. a. The system shows an error. b. The user tries again.
The ability to check the progress on the goal	User	1. The user wants to check his current progress and his future steps. 2. The user opens the app, chooses the goal. 3. The system shows the progress on this goal.	1. The user does not have goals. a. The system offers to set a new goal.
Share the progress in social media	User	1. The user wants to share the progress with friends. 2. The user chooses social media, messenger or etc. 3. The user chooses for whom to send, or other options. 4. The system sends a message to the friend or creates a post.	1. The user did not log in his account in the chosen social network. a. The system shows an error and asks to log in.
Giving up achieving the goal	User	1. The user wants to remove the goal. 2. The system asks if the user is sure. 3. The user confirms. 4. The system deletes the goal and offers to retry it later.	1. The user does not confirm deleting the goal. a. The system does not delete the goal.

The failure of the daily step	User	1. The user did not cope with the step. 2. The system offers to retry this step on the next day.	1. The user does not want to continue archiving the goal. a. The system offers to delete the goal.
Reward steps in a way to goal achieving	User	1. The user fulfils the step. 2. The system gives him points to his rating.	1. The user fails the step. a. The system takes away points from the user rating.

Table 3.2 Use Cases

User Stories

1. Set a new goal to achieve

   As a user I want to be able to set any goal so that I can create my own goal if there is no necessary predefined one.

2. Send notifications about daily steps

   As a user I want to be able to get notifications from the app so that I can remember about steps to do that day.

3. Authorization using mobile phone or email

   As a developer I want users to be able to register users in the app using phone number or email so that I can simplify the process of changing password for users.

4. The ability to check the progress on the goal

   As a user I want to see both previous and future steps of the goal so that I can visualise the whole progress on the goal.

5. Share the progress in social media

   As a user I want to be able to share my progress in social media so that I can show my achievements to others.

6. Giving up achieving the goal

   As a developer I want to add a possibility to stop achieving the goal so that users can take a break and try it again.

7. The failure of the daily step

   As a user I want to have the ability to repeat the daily step if I could not fulfill it so that I can continue with my goal.

8. Reward steps in a way to goal achieving

   As a developer I want to add a rewarding system, e.g. give points for successes and take back for failures so that I can motivate users.

C. Non-functional requirements

NFR	Sub-Characteristics	How will you achieve it
Response time	Complete page loads under 3 seconds	Ensure that the code is efficient at all levels
Reliability	The app should be available at least 90% of the day	Predict and solve the app failures
Usability	Users should be able to get through different features based on their cognitive knowledge	Use as little data as possible, separate content from navigation
	Users should be able to create goal under three clicks	
Portability	Screen works across all popular devices without loss of data and functionality	Use special Android libraries, APIs
Maintainability	Fixing a bug should take now more than one week	Follow SOLID principles, use standard API formats and clear document interfaces

Table 3.3 Non-functional requirements

4. Languages

Database design

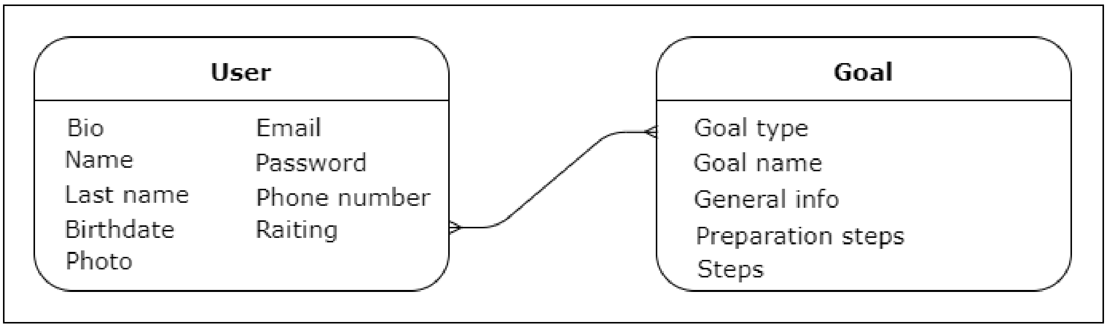


Figure 4.1 Entity relationship diagram

There are two tables: User for storing the information about users and Goal for storing the samples of goals. They have many-to-many relationship because one user can have multiple goals and one goal can be taken by multiple users.

UML diagrams

- Class diagram

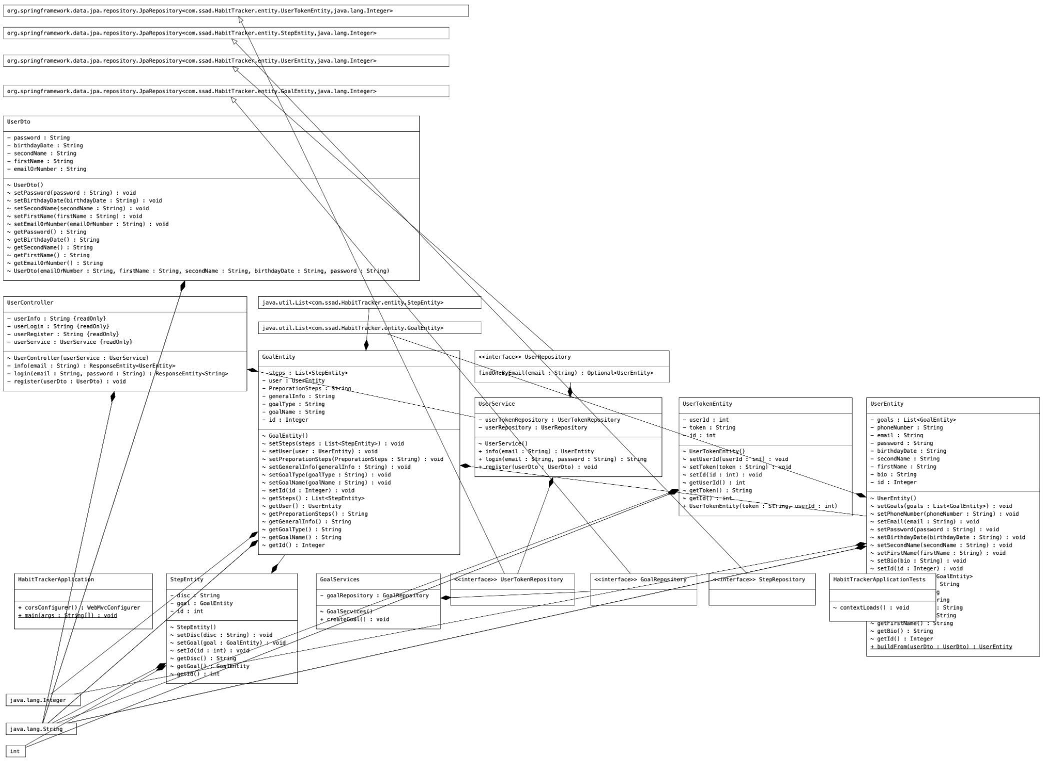

org.springframework.data.jpa.repository.JpaRepository<com.ssad.HabitTracker.entity.UserTokenEntity,java.lang.Integer>
org.springframework.data.jpa.repository.JpaRepository<com.ssad.HabitTracker.entity.StepEntity,java.lang.Integer>
org.springframework.data.jpa.repository.JpaRepository<com.ssad.HabitTracker.entity.UserEntity,java.lang.Integer>
org.springframework.data.jpa.repository.JpaRepository<com.ssad.HabitTracker.entity.GoalEntity,java.lang.Integer>
UserDto
– password : String
– birthdayDate : String
– secondName : String
– firstName : String
– emailOrNumber : String
~ UserDto()
~ setPassword(password : String) : void
~ setBirthdayDate(birthdayDate : String) : void
~ setSecondName(secondName : String) : void
~ setFirstName(firstName : String) : void
~ setEmailOrNumber(emailOrNumber : String) : void
~ getPassword() : String
~ getBirthdayDate() : String
~ getSecondName() : String
~ getFirstName() : String
~ getEmailOrNumber() : String
~ UserDto(emailOrNumber : String, firstName : String, secondName : String, birthdayDate : String, password : String)
UserController
– userInfo : String {readOnly}
– userLogin : String {readOnly}
– userRegister : String {readOnly}
– userService : UserService {readOnly}
~ UserController(userService : UserService)
– info(email : String) : ResponseEntity<UserEntity>
– login(email : String, password : String) : ResponseEntity<String>
– register(userDto : UserDto) : void
java.util.List<com.ssad.HabitTracker.entity.StepEntity>
java.util.List<com.ssad.HabitTracker.entity.GoalEntity>
GoalEntity
– steps : List<StepEntity>
– user : UserEntity
– PreporationSteps : String
– generalInfo : String
– goalType : String
– goalName : String
– id : Integer
~ GoalEntity()
~ setSteps(steps : List<StepEntity>) : void
~ setUser(user : UserEntity) : void
~ setPreporationSteps(PreporationSteps : String) : void
~ setGeneralInfo(generalInfo : String) : void
~ setGoalType(goalType : String) : void
~ setGoalName(goalName : String) : void
~ setId(id : Integer) : void
~ getSteps() : List<StepEntity>
~ getUser() : UserEntity
~ getPreporationSteps() : String
~ getGeneralInfo() : String
~ getGoalType() : String
~ getGoalName() : String
~ getId() : Integer
<<interface>> UserRepository
findOneByEmail(email : String) : Optional<UserEntity>
UserService
– userTokenRepository : UserTokenRepository
– userRepository : UserRepository
~ UserService()
+ info(email : String) : UserEntity
+ login(email : String, password : String) : String
+ register(userDto : UserDto) : void
UserTokenEntity
– userId : int
– token : String
– id : int
~ UserTokenEntity()
~ setUserId(userId : int) : void
~ setToken(token : String) : void
~ setId(id : int) : void
~ getUserId() : int
~ getToken() : String
~ getId() : int
+ UserTokenEntity(token : String, userId : int)
UserEntity
– goals : List<GoalEntity>
– phoneNumber : String
– email : String
– password : String
– birthdayDate : String
– secondName : String
– firstName : String
– bio : String
– id : Integer
~ UserEntity()
~ setGoals(goals : List<GoalEntity>) : void
~ setPhoneNumber(phoneNumber : String) : void
~ setEmail(email : String) : void
~ setPassword(password : String) : void
~ setBirthdayDate(birthdayDate : String) : void
~ setSecondName(secondName : String) : void
~ setFirstName(firstName : String) : void
~ setBio(bio : String) : void
~ setId(id : Integer) : void
GoalEntity>
String
g
ring
: String
String
~ getFirstName() : String
~ getBio() : String
~ getId() : Integer
+ buildFrom(userDto : UserDto) : UserEntity
HabitTrackerApplication
+ corsConfigurer() : WebMvcConfigurer
+ main(args : String[]) : void
StepEntity
– disc : String
– goal : GoalEntity
– id : int
~ StepEntity()
~ setDisc(disc : String) : void
~ setGoal(goal : GoalEntity) : void
~ setId(id : int) : void
~ getDisc() : String
~ getGoal() : GoalEntity
~ getId() : int
GoalServices
– goalRepository : GoalRepository
~ GoalServices()
+ createGoal() : void
<<interface>> UserTokenRepository
<<interface>> GoalRepository
<<interface>> StepRepository
HabitTrackerApplicationTests
~ contextLoads() : void
java.lang.Integer
java.lang.String
int

Figure 4.2 UML class diagram

- Component diagram

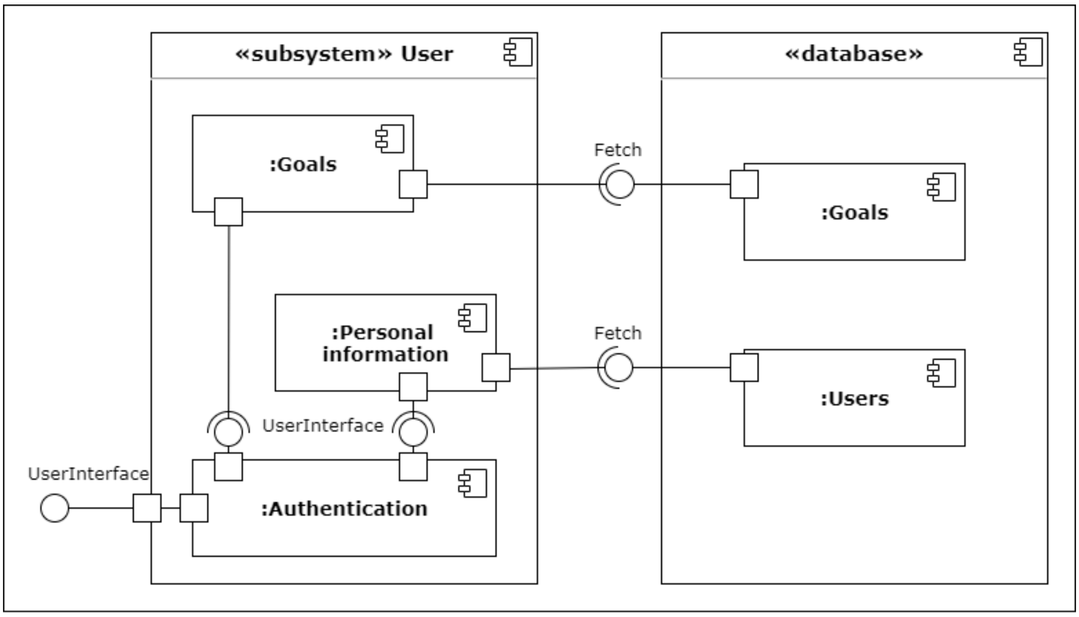


Figure 4.3 UML component diagram

There is one subsystem in the project - User. It has personal information and goals to achieve. Both personal information and goals are taken from the database. There are no other subsystems because there is no need to control users actions or offer them something.

- Deployment diagram

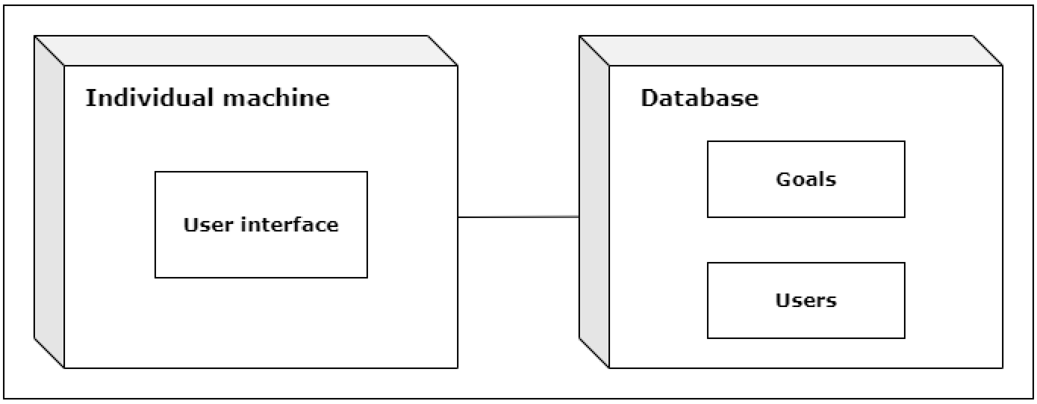


Figure 4.4 UML deployment diagram

There are two main nodes: Individual machine of the user (android mobile phone) and the Database from where all information is stored. Individual machine has User Interface. Database contains information about goals and users.

- Sequence diagram

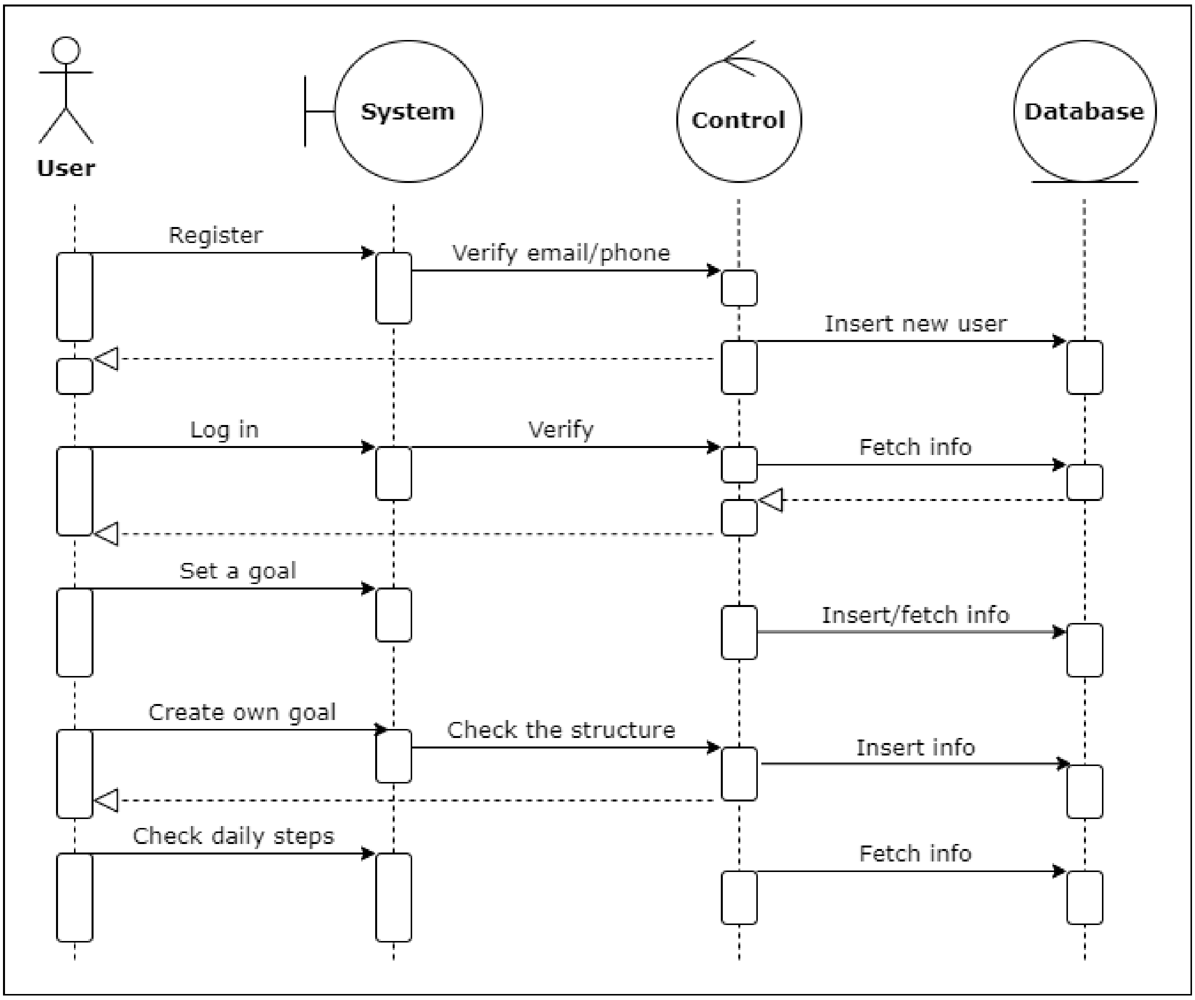


Figure 4.5 UML sequence diagram

Above it is shown how the main processes operate. When a user registers, the information is verified and inserted to the database.
After logging in the login information is verified and necessary staff is fetched from the database.
If a user wants to set a goal, the sample is fetched from the database and the information about the user is updated.
if a user creates a new goal, its structure is checked and the information is inserted into the database.
If a user wants to check his daily steps, the necessary information is fetched from the database.

SOLID principles usage

In order to make code more maintainable and less buggy it is reasonable to follow SOLID principles. Let us show their use in the project.

S - Single Responsibility Principle

Each class handles only one operation (has only one responsibility). In the UML class diagram (fig. 4.2) it is shown that classes are responsible for single operations of one type.

O - Open/Closed Principle

Classes are closed for modifications. In the UML class diagram (fig. 4.2) it is shown that classes' attributes cannot be modified from other classes. Attributes and methods are final.

L - Liskov's Substitution Principle

Parent classes should be easily substituted with child classes. From the UML class diagram (fig. 4.2) it is clear that there is no inheritance in this project. So, this principle cannot be checked.

I - Interface Segregation Principle

Many specific interfaces are better than one general. From the UML class diagram (fig. 4.2) it is seen that interfaces in the project are specific.

D - Dependency Inversion Principle

Communicate with classes through interfaces. From the UML class diagram (fig. 4.2) one can notice that each class has an interface for communication.

5. Viewpoint / Architecture (Static, Dynamic view)

Static view

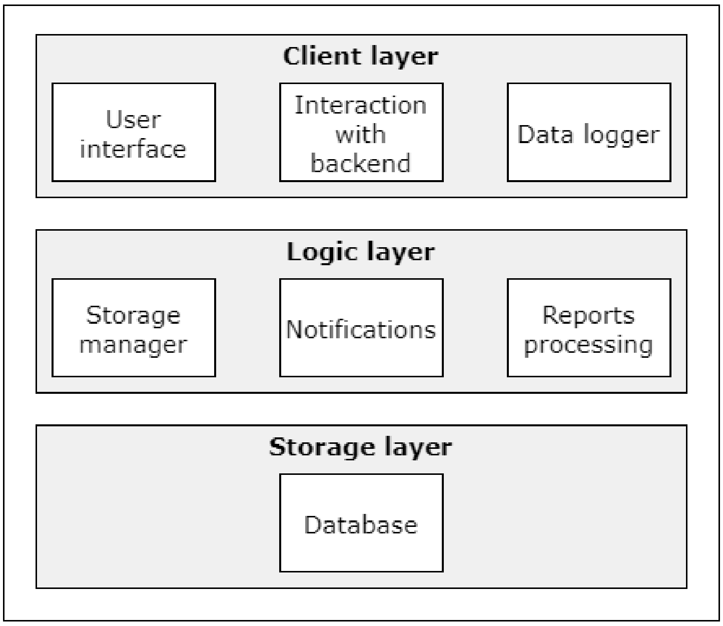


Figure 5.1 Static view

There are three layers in the app. Client layer is responsible for User interface, Interaction with backend and working with data. Logic layer - for Reports processing, sending Notifications and working with storage. And the Storage layer is responsible for the database.

Dynamic view

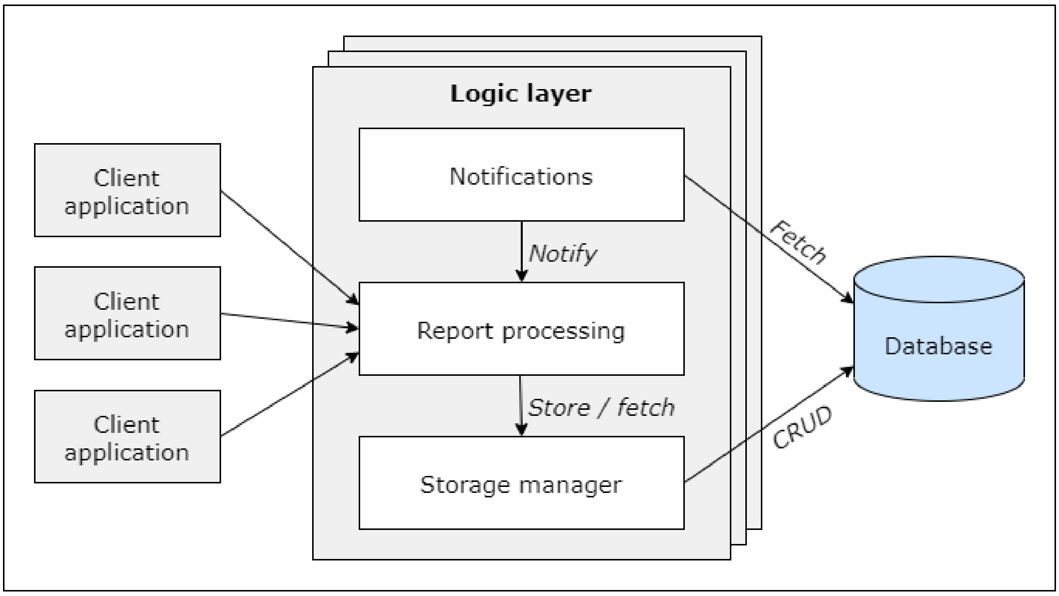
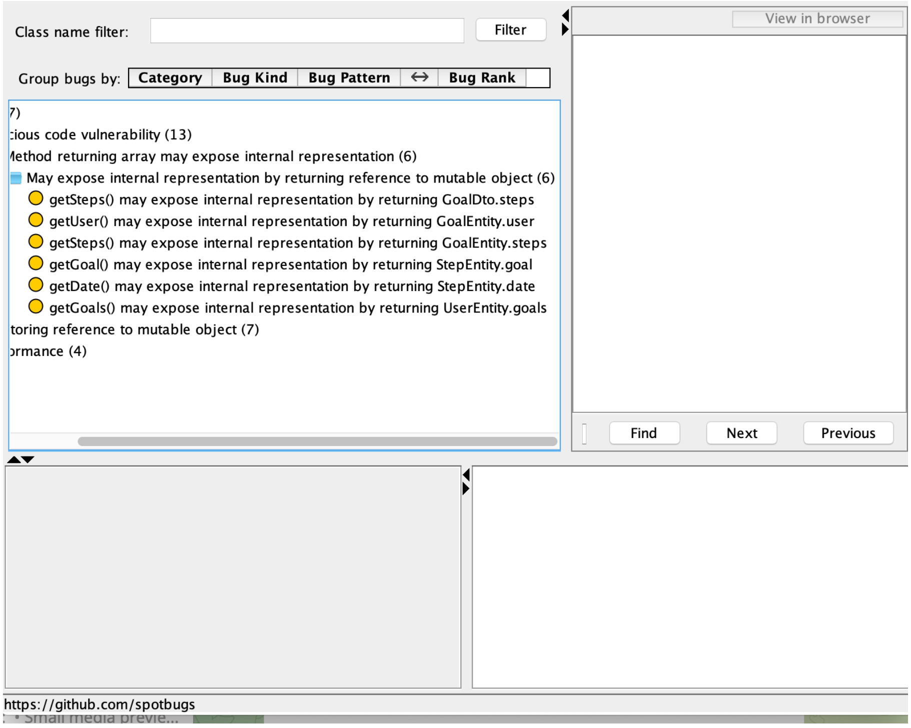


Figure 5.2 Dynamic view

Client applications communicate with the Logic layer. Firstly, the layer processes the report, then goes to the Storage manager which accesses the Database. Also The Logic layer is responsible for Notifications, they fetch the information from the Database and send it to the Report processing.

6. Code

Static analyzer (lint) result

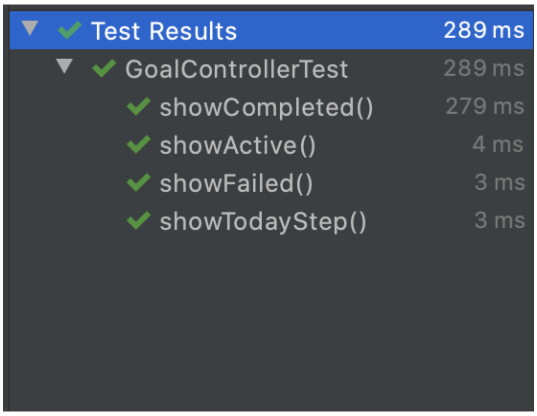


```
7)
:ious code vulnerability (13)
/lethod returning array may expose internal representation (6)
 May expose internal representation by returning reference to mutable object (6)
   getSteps() may expose internal representation by returning GoalDto.steps
   getUser() may expose internal representation by returning GoalEntity.user
   getSteps() may expose internal representation by returning GoalEntity.steps
   getGoal() may expose internal representation by returning StepEntity.goal
   getDate() may expose internal representation by returning StepEntity.date
   getGoals() may expose internal representation by returning UserEntity.goals
toring reference to mutable object (7)
ormance (4)
```

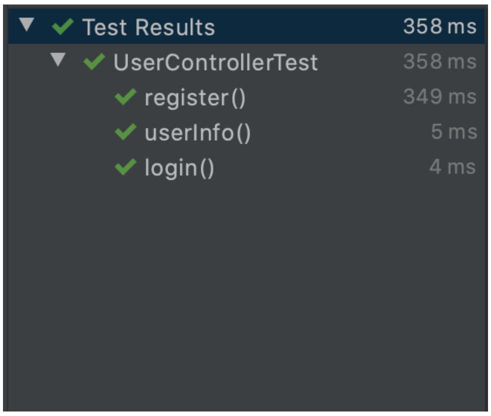
https://github.com/spotbugs

Figure 6.1 Lint result

Test coverage



```
▼ ✓ Test Results                289 ms
   ▼ ✓ GoalControllerTest       289 ms
        ✓ showCompleted()       279 ms
        ✓ showActive()            4 ms
        ✓ showFailed()            3 ms
        ✓ showTodayStep()         3 ms
```



```
▼ ✓ Test Results                358 ms
   ▼ ✓ UserControllerTest       358 ms
        ✓ register()            349 ms
        ✓ userInfo()              5 ms
        ✓ login()                 4 ms
```

Figure 6.2 First test

Figure 6.3 Second test

7. Rationale

a. Front-end

For the front-end development Kotlin was chosen. Let us compare it with one of the most known programming languages for Android development - Java.

Kotlin	Java
More concise, that increases readability and maintainability	Needs more lines of code to implement the same thing
New design patterns, cleaner syntax, ideas from functional programming	Lack of support for functional programming features
Scripting capabilities increase the portability (does not need to be compiled)	Needs to be compiled
Semicolons are optional, that makes code lighter	Semicolons are obligatory, more possibility for bugs occurrences
Null safety implemented	NullPointerException errors cause crashes
Has only primary and secondary constructors	Constructors can be used to take parameters, initialize attributes
No static members	Allows static members in classes

Table 7.1 Kotlin vs Java

All in all, Java and Kotlin have a lot in common, however Kotlin is a new programming language. It took the best from its predecessors, corrected mistakes, introduced new ideas. Thus, for Android development Kotlin is a better choice. [1][3]

b. Back-end

For the back-end Java in Spring Boot framework was chosen. Firstly, let us discuss the language selection. One of the most famous languages for the back-end Android development is Python, so here is the comparison of Java and Python languages.

Java	Python
The most supported language by Google	Android does not support native Python development, need to use converting tools
Complicated language for a beginner and SDK increases the complexity	More readable and easy syntax, more concise
Faster than Python	Slower than Java
Object-oriented	Cand mix object-oriented and imperative programming
Statically typed language	Dynamically typed language
Better characterized as low-level implementation language	Better as “glue” language

Table 7.2 Java vs Python

We can see that each of these programming languages has pros and cons. While Java is faster, Python is much easier to learn, it is more concise. On the other side, Java fits Android development more.[8][9]

Moreover, the complexity of the language can be reduced using frameworks. So, the second question is the Java framework. For this project the Spring Boot was chosen. Let us compare it with one of the most popular development frameworks of Java - Spring.

Spring Boot	Spring
Developers need to write more code than in Spring Boot	Reduces boilerplate code
Internally takes care of downloading the dependencies	Developers manually define dependencies for the Spring project
Lack of control (creates dependencies itself)	Eliminates the need to independently create factory and singleton classes

Table 7.1 Spring Boot vs Spring

From this we see that Spring Boot reduces the complexity of Java more than Spring, however it is not suitable for big projects. Thus, as our project is not very large, the Spring Boot is a better choice. [4][8]

c. Database

For this project to work with data, data definition language of the database is needed. So, the most known and usable language - SQL was taken. And MySQL was chosen as a relational database management system. Here arises the question about the DBMS. Why exactly that choice was made. Let us provide a comparison of MySQL with PostreSQL - one of the most popular DBMS for SQL.

MySQL	PostgreSQL
Simpler database that's relatively easy to set up and manage, fast, reliable, and well-understood	Feature-rich database that can handle complex queries and massive databases
SQL-standard types	Support many advanced types such as array, hstore, and user-defined type.
Each new connection is an OS thread	Each new connection is an OS process
performs well in OLAP & OLTP systems.	Performs well when executing complex queries

Table 7.1 MySQL vs PostgreSQL

From this comparison it is obvious that PosgreSQL is more suitable for complex databases with complex queries, it has more features. However, this project has a very simple database and simple queries, that is why MySQL is a better choice.[6][7]

8. References:

1. Berga, M. (2021, September 10). *Kotlin vs Java: The 12 differences you should know*. Blog | Imaginary Cloud. Online source: https://www.imaginarycloud.com/blog/kotlin-vs-java/.

2. Bolton, D. (2020, July 1). *Kotlin in 2020: 'A better java' continues to mature*. Dice Insights. Online source: https://insights.dice.com/2020/03/04/kotlin-2020-better-java-continues-mature/.

3. Gill, N. S. (2020, October 20). *Kotlin vs Java: Which is better for Android app development?* XenonStack. Online source: https://www.xenonstack.com/blog/kotlin-andriod#:~:text=Kotlin%20Application%20Deployment%20is%20faster,be%20executed%20in%20the%20JVM.

4. Kushnir, A. W. at B. (2021, October 2). *Pros and cons of using Spring Boot*. Insights. Online source: https://bambooagile.eu/insights/pros-and-cons-of-using-spring-boot/.

5. Peterson, R. (2021, October 7). *PostgreSQL vs MySQL: What is the difference?* Guru99. Online source: https://www.guru99.com/postgresql-vs-mysql-difference.html.

6. *PostgreSQL vs. mysql*. PostgreSQL Tutorial. (n.d.). Online source: https://www.postgresqltutorial.com/postgresql-vs-mysql/.

7. Smallcombe, M. (2019, June 14). *PostgreSQL vs MySQL: The critical differences*. Xplenty. Online source: https://www.xplenty.com/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/.

8. *Spring vs Spring Boot vs Spring mvc - javatpoint*. www.javatpoint.com. (n.d.). Online source: https://www.javatpoint.com/spring-vs-spring-boot-vs-spring-mvc.

9. System, O. S. (2021, August 5). *Best backend languages for Android App*. OS. Online source: https://os-system.com/blog/best-backend-languages-for-android-app/.