Bilkent University

Department of Computer Engineering

Senior Design Project

reporTown

Low Level Design Report

Arda Akça Büyük, Elif Özer, Cemre Biltekin, Oğuz Kaan İmamoğlu, Mustafa Yaşar

Supervisor: Dr. Ayşegül Dündar

Jury Members: Dr. Shervin Arashloo, Dr. Hamdi Dibeklioğlu

Innovation Expert: Mehmet Çakır

Feb 28, 2022

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Table of Contents

1. Introduction

Cities are home to many problems that can victimize and even sometimes endanger the citizens living in them. Road problems, garbage problems, transportation problems can be given as examples of those problems. When citizens encounter such problems, most of the time, they are unable to interfere in those problems individually. Therefore, these problems can only be resolved by institutions like the municipality, governorship, or non-governmental organizations. Citizens who encounter such situations can report these situations to the authorities through various channels. However, these tools cannot provide adequate solutions both during the reporting of the situation and during the follow-up of the situation. In addition, if the relevant problem does not create enough of an agenda, the authorities tend not to do what is necessary about the issue or take it slow. The fact that the addressee of the problem is not known about many problems also undermines the problem-solving process.

In some cases, instead of reaching out to the authorities, people may need help from other people and may want to find a volunteer to fix a problem. The feeding of stray animals can be given as an example of such cases.

It is difficult to report problems in cities, and following this process makes people desensitized about these problems. The lack of platforms where these problems can have a social impact also makes authorities insensitive. The fact that people do not have the opportunity to see the problems in the city and the locations of these problems also makes people's daily lives difficult and wastes their time. All this causes cities to become more and more neglected and more challenging to live in. The need for a project that can offer solutions to all of these problems seems obvious.

reportTown is a mobile application that aims to solve all these problems. Citizens can convey the problems they see around them to the relevant institutions via reportTown. A person who encounters a problem takes a photograph of the problem, uploads it to the application by giving detailed information such as the description and location of the

problem, and tags the institution or municipality that should solve the problem or they can add “volunteering” tag. The reports shared with the volunteering tag express that the problem is suitable to be solved with the help of other citizens. Thanks to machine learning algorithms, object detection and problem classification are made from the photos taken. Also, using machine learning, trending problems are also detected.

Citizens and institutions will be able to see the problems on a map user interface. Depending on the frequency of problems reported in an area, nodes of different colours will appear on the map. Hot colours like red and orange mean that there are many problems reported in that area, and cold colours like blue and green mean that a few problems are reported in the area. By touching these nodes, users and authorities can view related posts. Thanks to this map, authorities can plan their actions wisely, and citizens can plan their days with these problems in their minds.

Also, reports can be seen in the feed interface, people can upvote reports shared by other citizens and comment on these reports. Every citizen and institution has points calculated based on the quality of the reports, the number of reports they solve, and the upvotes they receive. In this way, a sweet competition is created between both citizens and institutions in order to obtain more livable cities.

In this report, we will explain the low level design of reporTown. Object-design trade-offs will be explained in terms of our decisions. Also, explanations of packages in Presentation Layer, Business Layer and Data Layer are given in detail. Classes we implement are covered in section 3 by using the interface documentation guidelines given in section 1.2.

1.1 Object Design Trade-offs

1.1.1 Performance vs Speed

Sending data to servers and waiting for a response from the servers can slow down the speed. Thus, we will use cloud computing servers for the visual data in the application.

Also, since reportTown is powered by Machine Learning and Computer Vision, we can achieve better performance by using complex architectures so that these parts do not block report creation.

1.1.2 Usability vs Functionality

reporTown allows users to use the same functionality in different ways such as following reports using map or feed feature. Recommending a report topic to users is one of the features that facilitates usability. The application is well suited for adding many more features, but adding too many features to reportTown may reduce usability.

1.1.3 Portability vs Functionality

In order to implement all functionalities of reporTown for a promised due date, the application will just run on Android devices. Although all functionalities will be implemented, the fact that the application can only run on android devices reduces portability.

1.2 Interface Documentation Guidelines

Styling for demonstrating classes is as follows:

class ClassName	Definition of class
Attributes	-attributeName: attribute_type
Methods	+method_name(arg: arg_type) : return type

- : private
+ : public

1.3 Engineering Standards

In this report and other reports prepared within the scope of this project, IEEE standards for citations and UML standards for diagrams are followed. For the project management method, we went sprint-based and used the Scrumban project management framework. Also, for code formatting, we followed eslint.

1.4. Definitions, Acronyms and Abbreviations

- **YOLO:** You only look once (YOLO) is a state-of-the-art, real-time object detection system [1].
- **MongoDB:** MongoDB is a document-oriented NoSQL database used for high volume data storage [2].
- **Report:** The report is the cornerstone of reportTown. The problem created by a citizen and uploaded to the application is called a report and can be solved by authorized institutions or volunteers.
- **Category:** Each report has a category. The category expresses what the problem encountered is about.
- **Citizen:** The user type that creates the reports.
- **Volunteer:** Citizens can volunteer and resolve some reports.
- **Institution:** It is the type of user that expresses institutions such as the governorship and municipality, it deals with the resolution of the reports.
- **DAO:** Data Access Object.
- **JWT:** JSON Web Token is an open standard for securely transmitting information between parties as a JSON object [3].

2. Packages

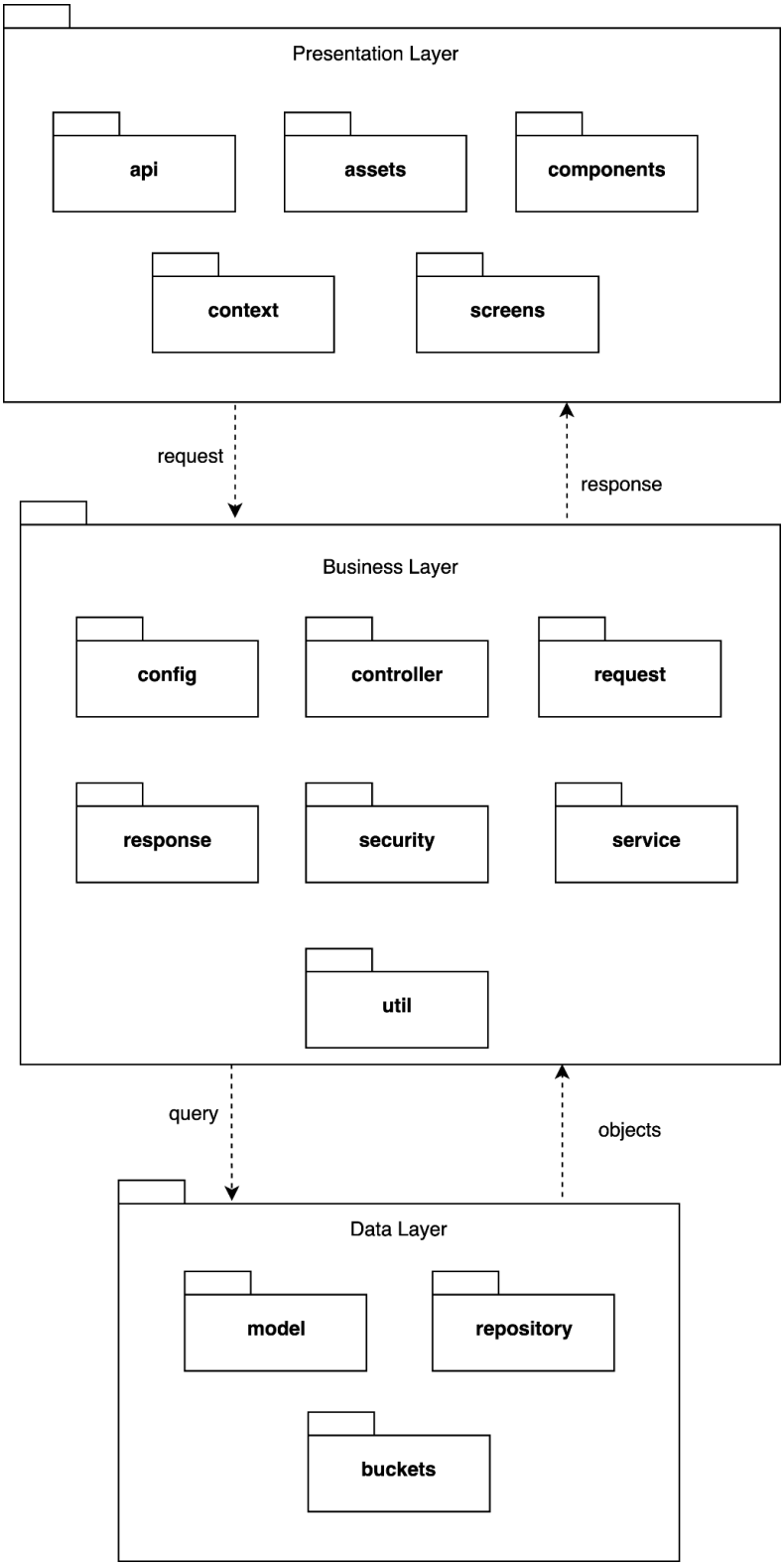

Figure 1. reporTown packages

In reporTown, we utilized a three-layered architecture for more scalability and portability. We have three communicating layers.

The presentation layer is the frontend application of reporTown. It has the interface role between the end-user and the server.

The business layer is the package that the application logic is contained in. This package deals with the computational and the logical load of the application.

The data layer is the interface between the MongoDB database and the AWS buckets. It provides the necessary objects to the business layer from these storages.

Both the business and data layer is maintained in the backend application of reporTown.

2.1 Presentation Layer

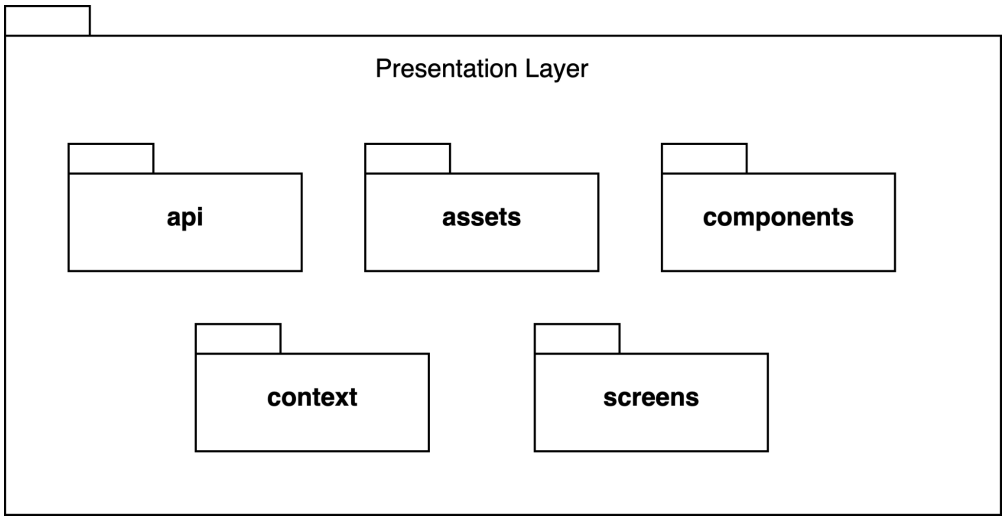

Figure 2. Presentation Layer Package

This is the presentation layer of the application, where the user activities are sent to the server (business layer), and where the responses are presented to the user. This layer behaves as the interface between the end-user and the server.

API package is responsible for requests that are sent to the database and provides the relevant obtained data to the related screens.

Assets package is responsible for containing the assets that are used throughout the application.

Components package contains the common components that are presented to the users within screens. For instance, report components are components which are used multiple times in feed screens. This separation operation makes the code more readable, understandable, and robust because of the fact that they are not implemented multiple times in different screens.

Context package is used to handle operations in the presentation layer such as registration, logging in, and storing user information such as authentication token that is used throughout the application. This package easily stores and transfers such critical data that are needed by different screens.

Screens package has multiple folders for different types of users such as citizens, institutions, and volunteers. This package is responsible for providing different screens with those different types of users. Since each user type has different kinds of operations, handling such operations has been separated into another package.

2.2 Business Layer

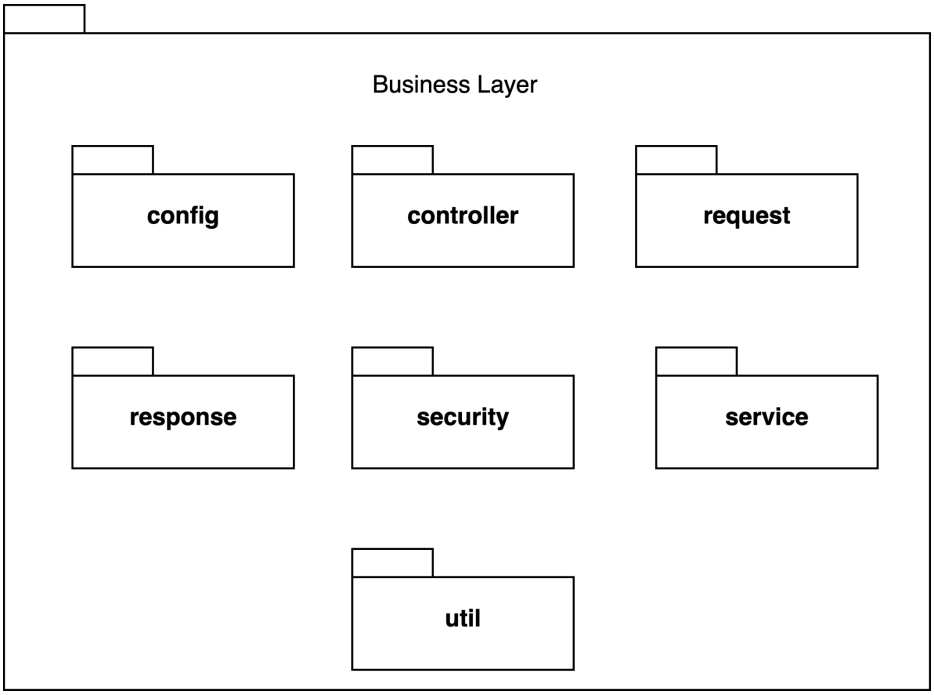

Figure 3. Business Layer Package

This is the business layer of the application. This layer takes the request from the presentation layer, does the needed computations/queries, and if necessary, sends the response (data/objects) back to the presentation layer.

The config package is responsible for the AWS configurations for the buckets that the application will utilize for storing images.

The controller package contains the controller that takes the request from the presentation layer and calls certain services that contain the application logic.

The request package contains the objectified requests with certain attributes that the application receives from the end-user. These request objects come in handy since they do not require JSON string parsing on the server-side.

The response package contains the objectified responses with certain attributes that the application sends back to the end-user. Again, these responses can be directly sent to the presentation layer without parsing to JSON.

The security package deals with the authorization and authentication of the application. The application uses Spring Boot Security, which is a highly reliable and easy-to-implement Spring Boot framework.

The service package contains the relevant services and application logic that the complex computations and data queries are done. These services are called as the controllers receive requests from the presentation layer.

The util package contains the utilities that the services sometimes need (i.e. EmailValidator, which validates the format of an email while registering or logging in).

2.3 Data Layer

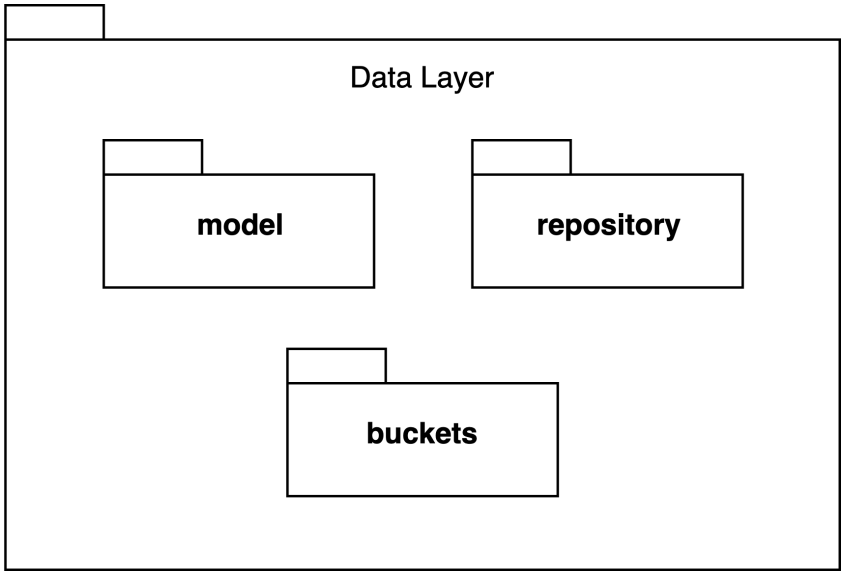

Figure 4. Data Layer Package

This is the data layer of the application, where the data query interfaces and the data models are contained.

The model package contains all the MongoDB database objects that the application persists. The classes inside this package basically act as Object Relation Mappings.

The repository package contains the classes which are MongoDB query interfaces that contain custom MongoDB queries.

The buckets package is responsible for connecting to AWS buckets and providing the content inside them (mainly images) to the business layer of the application.

3. Class Interfaces

The classes may have getter and setter methods for their attributes. Since these methods are self-explanatory and will lengthen the representations, they are not displayed explicitly in the documentation.

3.1 Presentation Layer

The presentation layer is mostly made up of UI components like various screens, feeds and also API calls in JavaScript, so there are no classes to document in the report.

3.2 Business Layer

3.2.1 config

3.2.1.1 AmazonConfig

class AmazonConfig	This class is the service client object for Amazon Web Services S3 storage.
Attributes	No attributes.
Methods	+s3() : AmazonS3

- s3(): Obtains an instance of the client builder and modifies client's properties.

3.2.2 controller

3.2.2.1 LoginController

class LoginController	This class is the controller class for log in (authentication).
Attributes	-authenticationManager: AuthenticationManager -jwtUtils: JwtUtils
Methods	+authenticateUser(loginRequest: LoginRequest) : ResponseEntity<>

- authenticateUser(loginRequest: LoginRequest): Handles authentication of the user given the username and the password.

3.2.2.2 RegistrationController

class RegistrationController	The class is the controller class for registration of new users.
Attributes	No attributes.
Methods	+register(request: RegistrationRequest) : User

- register(request: RegistrationRequest): Handles registration of a new user given appropriate registration information according to the user type.

3.2.2.3 ReportController

class ReportController	This is the controller class for all report related actions.
Attributes	-reportService: ReportsService -notificationService: NotificationService -mlService: MLService
Methods	+postReport(report: Report) : ResponseEntity<?> +upvote(user: User, report: Report) : ResponseEntity<?> +comment(comment: Comment): ResponseEntity<?> +getReport(id: long, user: User): Report +getAllReports(): ArrayList<Report> +getReportsFiltered(user: User, filter: enum): ArrayList<Report> +getResolvedReports(user: User, filter: enum): ArrayList<Report> +getUnresolvedReports(user: User, filter: enum): ArrayList<Report> +getUpvotes(report: Report): ArrayList<Citizen> +getComments(report: Report): ArrayList<Comment>

	+reportAsSpam(report: Report): ResponseEntity<?> +volunteer(report: Report, user: User): ResponseEntity<?> +updateVolunteer(report: Report, user: User): ResponseEntity<?> +deleteReport(report: Report, user: User): ResponseEntity<?> +updateUpvote(user: User, report: Report) : Report +assignOfficialToReport(official: Official, report: Report): ResponseEntity<?>

- postReport(report: Report): Given report properties specified by the citizen, creates a new report and saves it to the database.
- upvote(user: User, report: Report): Increases the upvote count of a report should a citizen upvotes a report.
- comment(comment: Comment): Creates and posts a comment on a published report by a citizen.
- getReport(id: long, user: User): Fetches the report specified the report id and the creator citizen.
- getAllReports(): Fetches all reports in the database.
- getReportsFiltered(user: User, filter: enum): Fetches all reports appropriate to the selected filter like get reports by user etc.
- getResolvedReports(user: User, filter: enum): Fetches resolved reports appropriate to the selected filter like get resolved reports by user etc.
- getUnresolvedReports(user: User, filter: enum): Fetches unresolved reports appropriate to the selected filter like get unresolved reports by user etc.
- getUpvotes(report: Report): Fetches upvote count of a specified report.
- getComments(report: Report): Fetches comments of a specified report.
- reportAsSpam(report: Report): Marks a published report as spam to report to admins for removal.
- volunteer(report: Report, user: User): Claims the current user as a volunteer for the published report.
- updateVolunteer(report: Report, user: User): Undoes volunteering action for the citizen should the citizen changes his/her mind about volunteering.
- deleteReport(report: Report, user: User): Deletes the report from the system permanently issued by the current user.
- updateUpvote(user: User, report: Report): Undoes upvote action for a citizen should the citizen changes his/her mind.
- assignOfficialToReport(official: Official, report: Report, user: User): Allows an institution (that is the current user) to assign an official to handle the report.

3.2.2.4 SolutionController

class SolutionController	This class is the controller class for all actions related to a solution for a report.

Attributes	-reportService: ReportService -solutionService: SolutionService -notificationService: NotificationService
Methods	+postSolution(solution: Solution, report: Report, user: User): ResponseEntity<?> +approveSolution(solution: Solution, user: User): ResponseEntity<?> +rejectSolution(solution: Solution, user: User): ResponseEntity<?>

- postSolution(solution: Solution, report: Report, user: User): Creates a solution by an official from an institution and saves it in the database.
- approveSolution(solution: Solution, user: User): Allows a citizen to accept the solution should it be appropriate.
- rejectSolution(solution: Solution, user: User): Allows a citizen to reject the solution, should it be inappropriate.

3.2.2.5 UserController

class UserController	This class is the controller class for all user/profile related actions.
Attributes	-userService: UserService -reportService: reportService
Methods	+getUser(userId: ObjectId): ResponseEntity<ProfileResponse> +updateUserInfo(user: User, userNew: User): User

- getUser(userId: ObjectId): Fetches the user from appropriate user type repository by unique user id.
- updateUserInfo(user: User, userNew: User): Updates user information with new entered information for user fields.

3.2.2.6 AdminController

class AdminController	This class is the controller class for all admin related actions.
Attributes	-adminService: adminService -userService: UserService -reportService: reportService
Methods	+banUser(user: User): ResponseEntity<?>

	+deleteReport(report: Report, user: Report): ResponseEntity<?> +approveInstitutionDocument(document: File, institution: Institution): ResponseEntity<?>

- banUser(user: User): Allows admin to ban (delete) a user after examining spam reports.
- deleteReport(report: Report, user: Report): Allows admin to delete a report after examining it as spam.
- approveInstitutionDocument(document: File, institutionInfo: Institution): Allows admin to approve official institution registration document and save institution into the database.

3.2.3 request

3.2.3.1 LoginRequest

class LoginRequest	This class is the request class for Login operation
Attributes	-username: String -password: String
Methods	+getUserName() : String +getPassword() : String +setUserName(username: String) +setPassword(password: String)

3.2.3.2 RegistrationRequest

class RegistrationRequest	This class is responsible for registration request operation of user
Attributes	-email: String -username: String -password: String -role: String -firstName: String -lastName: String -position: String -institutionName: String -approvalDocument: String -positions: String

3.2.3.3 ReportRequest

class RegistrationRequest	This class is the request class for Report object.
Attributes	-institutionId: ObjectId -description: String -category: String -comments: ArrayList<ObjectId> -upvotes: ArrayList<ObjectId> -location: JsonObject -report_image_link: String -file: MultipartFile

3.2.4 response

3.2.4.1 JwtResponse

class JwtResponse	This class is the response class for the JWT authentication after login.
Attributes	-token: String -type: String -id: ObjectId -username: String -email: String -roles: List<String>

3.2.4.2 ProfileResponse

class ProfileResponse	This class is the response class for the profile page.
Attributes	-success: boolean -user: UserDetails -reports: List<Report>

3.2.4.3 ReportResponse

class ReportResponse	This class is the response class for the report page.
Attributes	-report: Report

3.2.5 security

3.2.5.1 SecurityConfiguration

class SecurityConfiguration	This class is responsible for making security configurations in processes like authentication and session management.
Attributes	-passwordEncoder: PasswordEncoder -userService: UserService -unauthorizedHandler: AuthEntryPointJwt
Methods	+authenticationJwtTokenFilter(): AuthTokenFilter +configure(http: HttpSecurity): void +daoAuthenticationProvider: DaoAuthenticationProvider

- authenticationJwtTokenFilter(): Implements a JWT filter for extracting JWT token from headers for authentication.
- configure(http: HttpSecurity): Enables HTTP Security in Spring and makes sure requests are authenticated properly.
- daoAuthenticationProvider: Enables retrieval from relational database by providing authentication.

3.2.5.2 PasswordConfiguration

class PasswordConfiguration	This class is responsible for implementing a password encoder for password protection of users using BCrypt.
Attributes	No attributes.
Methods	+passwordEncoder() : PasswordEncoder

- passwordEncoder(): Provides password encoding with BCrypt.

3.2.6 service

3.2.6.1 UserService

class UserService	This class is the service class for the user object.
Attributes	-passwordEncoder: PasswordEncoder -userRepository: UserRepository -institutionRepository: InstitutionRepository -officialRepository: OfficialRepository -citizenRepository: CitizenRepository

	-logger: Logger
Methods	+getUser(userId: ObjectId): RegisteredUser +updateProfilePicture(picture: File, id: long): User +updateBio(bio: String, id: long): User +changePassword(id: long, password: String): User +getPassword(id: long): String +changeEmail(id: long): User +getEmail(id: long): String +changeUsername(username: String): User +getUsername(id: long): String +changeFirstName(id: long): User +changeLastName(id: long): User +getFullName(id: long): String +saveUser(user: User): void +getByUsername(username: String): User +getInstitutionOfficials(institutionId: long): ArrayList<Official>

- getUser(userId: ObjectId): Fetches user from user repository by its id.
- updateProfilePicture(picture: File, id: long): Updates profile picture of a user with a new picture and updates it on the repository.
- updateBio(bio: String, id: long): Updates bio of a user with a new bio string and updates it on the repository.
- changePassword(id: long, password: String): Updates password of a user with the given new password.
- getPassword(id: long): Fetches password of a specific user by id.
- changeEmail(id: long): Updates email of a user with the new email specified by the user.
- getEmail(id: long): Fetches email of a specific user by id.
- changeUsername(username: String): Updates username of a user with the new username specified by the user, checking the uniqueness of the username from the repository.
- getUsername(id: long): Fetches username of a specific user by id from repository.
- changeFirstName(id: long): Updates first name of a user on the database.
- changeLastName(id: long): Updates last name of a user on the database.
- getFullName(id: long): Fetches full name of a user by combining first and last name of a user.
- saveUser(user: User): Saves user to the user repository given user information.
- getByUsername(username: String): Fetches user by its username from the repository.
- getInstitutionOfficials(institutionId: long): Fetches officials of an institution given the institution id from institution repository.

3.2.6.2 AdminService

class AdminService	This class is the service class for the admin object.
Attributes	-userRepository: UserRepository -adminRepository: AdminRepository -institutionRepository: InstitutionRepository -officialRepository: OfficialRepository -citizenRepository: CitizenRepository
Methods	+banUser(user: User): void +deleteReport(report: Report, user: Report): void +approveInstitutionDocument(document: File, institution: Institution): void

- banUser(user: User): Allows admin to ban (delete) a user after examining spam reports.
- deleteReport(report: Report, user: Report): Allows admin to delete a report after examining it as spam.
- approveInstitutionDocument(document: File, institutionInfo: Institution): Allows admin to approve official institution registration document and save institution into the database.

3.2.6.3 ReportService

class ReportService	This class is the service class for the report object.
Attributes	-commentRepository: CommentRepository -reportRepository: ReportRepository -solutionRepository: SolutionRepository -mapRepository: MapRepository
Methods	+postReport(report: Report) : boolean +uploadImage(image: File[5], id: long) : void +upvote(id: long) : Report +comment(comment: Comment, id: long): void +getReport(id: long): Report +findAll(): ArrayList<Report> +getReportsByCategory() : ArrayList<Report> +getReportsByLocation(): ArrayList<Report> +getReportsByUser(): ArrayList<Report> +getReportsByInstiution(): ArrayList<Report> +getResolvedReportsByUser(): ArrayList<Report> +getResolvedReportsByInstiution(): ArrayList<Report> +getUnresolvedReportsByUser(): ArrayList<Report>

	+getUnresolvedReportsByInstiution(): ArrayList<Report> +getUpvotes(report: Report): ArrayList<Citizen> +getComments(report: Report): ArrayList<Comment> +joinVolunteering(id: long): void +updateVolunteer(id: long): void +getReportsByTrend(): ArrayList<Report> +deleteReport(id: long): void +updateUpvote(id: long) : Report +assignOfficialToReport(official: Official, report: Report): void

- postReport(report: Report): Saves created report to the report repository.
- uploadImage(image: File[5], id: long) : Attaches uploaded images to the report.
- upvote(id: long) : Increases upvote count of a report in the repository.
- comment(comment: Comment, id: long): Adds a comment on a report.
- getReport(id: long): Fetches report by its unique id.
- findAll(): Fetches all reports in the repository.
- getReportsByCategory() : Fetches reports from a specific category.
- getReportsByLocation(): Fetches reports from a specific region.
- getReportsByUser(): Fetches reports by creator user.
- getReportsByInstiution(): Fetches reports of a tagged institution.
- getResolvedReportsByUser(): Fetches resolved reports by creator user.
- getResolvedReportsByInstiution(): Fetches resolved reports by tagged institution.
- getUnresolvedReportsByUser(): Fetches unresolved reports by creator user.
- getUnresolvedReportsByInstiution(): Fetches unresolved reports by tagged institution.
- getUpvotes(report: Report): Fetches upvote count of a report from report repository.
- getComments(report: Report): Fetches comments of a report from report repository.
- joinVolunteering(id: long): Associates user with a report for volunteering in the database.
- updateVolunteer(id: long): Updates volunteer situation of a user.
- getReportsByTrend(): Fetches trending reports from machine learning service.
- deleteReport(id: long): Deletes a report from the repository.
- updateUpvote(id: long): Updates upvote situation.
- assignOfficialToReport(official: Official, report: Report): Associates an official with a report in the database.

3.2.6.4 NotificationService

class NotificationService	This class is the service class for the notification object.

Attributes	-notificationRepository: NotificationRepository
Methods	+sendNotification(notif: Notification, id: long): void +getNotifications(id: long): ArrayList<Notification>

- sendNotification(notif: Notification, id: long): Sends notification to the specified user given the notification and saves it into notification repository.
- getNotifications(id: long): Get all notifications of a specified user by user id.

3.2.6.5 MLService

class MLService	This class is the service class for the machine learning services.
Attributes	-reportRepository: ReportRepository
Methods	+checkSpam(): boolean +getTrends(): boolean +classifyCategory(): enum Category +recommendInstiution(): ArrayList<Institution>

- checkSpam(): Checks whether the report is a spam or not using Bag of Words machine learning model and if it is spam, mark the report as spam for admins.
- getTrends(): Checks trending reports from report repository by doing a trend analysis.
- classifyCategory(): Classifies a report in the report making process using object detection from uploaded images to assist user in categorizing the reports.
- recommendInstiution(): Recommends institutions for the report in the report creating process for assisting the user in addressing the issue.

3.2.6.6 RegistrationService

class RegistrationService	This class is the service class for the registration purpose.
Attributes	-userService: UserService -emailValidator: EmailValidator
Methods	+register(request: RegistrationRequest): ApplicationUser

- register(request: RegistrationRequest): Registers a user if its credentials are valid for the user type it registered for and saves into the database.

3.2.6.7 SolutionService

class SolutionService	This class is the service class for the solution object.
Attributes	-solutionRepository: SolutionRepository -officialRepository: OfficialRepository -reportRepository: ReportRepository
Methods	+saveSolution(solution: Solution): void +approveSolution(id: long): void +rejectSolution(id: long): void

- saveSolution(solution: Solution): Saves solution to the database given the solution object.
- approveSolution(id: long): Marks solution as approved and report as resolved.
- rejectSolution(id: long): Marks solution as rejected and report as unresolved.

3.3 Data Layer

3.3.1 model

3.3.1.1 User

class User	This class represents the general user with common attributes for all user types.
Attributes	-userId: ObjectId -username: String -email: String -password: String -role: UserRole -isAccountNonExpired: boolean -isAccountNonLocked: boolean -isCredentialsNonExpired: boolean -isEnabled: boolean
Methods	Only getter and setter methods.

3.3.1.2 RegisteredUser

class RegisteredUser	This class represents the registered user with common attributes for user types except admins.

Attributes	-profilePicture: String -bio: String -score: float
Methods	Only getter and setter methods.

3.3.1.3 Citizen

class Citizen	This class represents the citizen user and its special attributes apart from a general registered user.
Attributes	-firstName: String -lastName: String -reports: ArrayList<ObjectId>
Methods	Only getter and setter methods.

3.3.1.4 Official

class Official	This class represents the official user and its special attributes apart from a general registered user.
Attributes	-institutionId: ObjectId -firstName: String -lastName: String -position: String
Methods	Only getter and setter methods.

3.3.1.5 Institution

class Institution	This class represents the institution user and its special attributes apart from a general registered user.
Attributes	-institutionName: String -approvalDocument: File -reports: ArrayList<ObjectId> -officials: ArrayList<ObjectId> -positions: ArrayList<String>
Methods	Only getter and setter methods.

3.3.1.6 Report

class Report	This class is the object for created reports for city problems.
Attributes	-reportId: ObjectId -userId: ObjectId -solutionId: ObjectId -comments: ArrayList<ObjectId> -description: String -images: File[5] -upvotes: ArrayList<ObjectId> -upvoteCount: int -commentCount: int -location: JSON -category: enum -tags: ArrayList<Institution>
Methods	Only getter and setter methods.

3.3.1.7 Admin

class Admin	This class represents the admin user.
Attributes	-adminId: ObjectId
Methods	Only getter and setter methods.

3.3.1.8 Comment

class Comment	This class is the comment object.
Attributes	-commentId: ObjectId -userId: ObjectId -text: String -replies: ArrayList<ObjectId>
Methods	Only getter and setter methods.

3.3.1.9 Notification

class Notification	This class is the notification object.

Attributes	-notificationId: ObjectId -userId: ObjectId -reportId: ObjectId -type: enum -message: String
Methods	Only getter and setter methods.

3.3.1.10 Solution

class Solution	This class is the solution object for created reports.
Attributes	-solutionId: ObjectId -reportId: ObjectId -institutionId: ObjectId -description: String -images: File[5]
Methods	Only getter and setter methods.

3.3.1.11 Map

class Map	This class is map object to represent regions as cities.
Attributes	-reports: HashMap<JSON, ArrayList<ObjectId>>
Methods	Only getter and setter methods.

4. Glossary

MongoDB: an open source NoSQL database for large data

AWS: cloud computing platform for database storage

S3: AWS based object storage service

JSON: human readable format for structure of the data/file

5. References

[1] J. Redmon, "YOLO: Real-Time Object Detection", *Pjreddie.com*, 2022. [Online]. Available: https://pjreddie.com/darknet/yolo/. [Accessed: 27- Feb- 2022].

[2] "What is MongoDB? Introduction, Architecture, Features & Example", *Guru99*, 2022. [Online]. Available: https://www.guru99.com/what-is-mongodb.html. [Accessed: 27- Feb- 2022].

[3] "JWT.IO - JSON Web Tokens Introduction", *Jwt.io*, 2022. [Online]. Available: https://jwt.io/introduction. [Accessed: 27- Feb- 2022].