TED UNIVERSITY

CMPE 492 SENIOR PROJECT II

SEARCH AND RESCUE OPERATION PORTAL (SAROP)

Low Level Design Report

Project Name: SAROP (Search and Rescue Operation Portal)

Project Url: sarop.tech

Team Members:

• Arda Gök

• Saliha Nursu Baltacı

• Ceren Özdoğan

• Mert Çıkla

Name of the supervisor: Emin Kuğu

Names of the juries: Tolga Kurtuluş Çapın - Venera Adanova

1.Introduction

Every solid software system begins with a well-planned design in which decisions are carefully considered and technical standards are strictly adhered to. Think of this report as a guidebook, unveiling the inner workings of the Search and Rescue Operations Platform (SAROP). SAROP is a digital assist designed to suit the special needs of search and rescue teams in difficult conditions, and we'll go over the details in this Low-Level Design Report. In the next sections, we'll look at how SAROP was designed and what it can achieve.

1.1 Object design trade-offs

SAROP faced difficult decisions into a full open-source operation platform designed specifically for search and rescue teams. These trade-offs are all about finding the right balance between enhancing system functionality and solving the specific issues that search and rescue teams encounter.

1.1.1 Efficiency vs. Precision in Operations:

Efficiently processing map data during search and rescue operations is a problem compared to precision. The goal here is to develop an algorithmically efficient system for quickly analyzing and manipulating maps, allowing teams to operate effectively in the field. This emphasis on efficiency over perfect precision is chosen to meet the real-time needs of catastrophe response scenarios.

1.1.2User-Friendly Interface vs. Data Privacy:

One of SAROP's design objectives is to make the application user-friendly while protecting users' sensitive maps and operations data and their personal data. Implementing a strong authentication system and using hashing methods to store data are the trade-offs. In this way, SAROP provides a user-friendly experience while protecting the privacy and security of sensitive information.

1.1.3 Offline Accessibility vs. Dependency on Internet:

SAROP is aware of the difficulties of working in places where internet connection is limited. In order to solve this problem, the project is designed to be used without the Internet, allowing search and rescue teams to complete their tasks without any problem in the operation. This method requires trade-offs in terms of real-time data updates but prioritizes the practicality of working in various operational contexts.

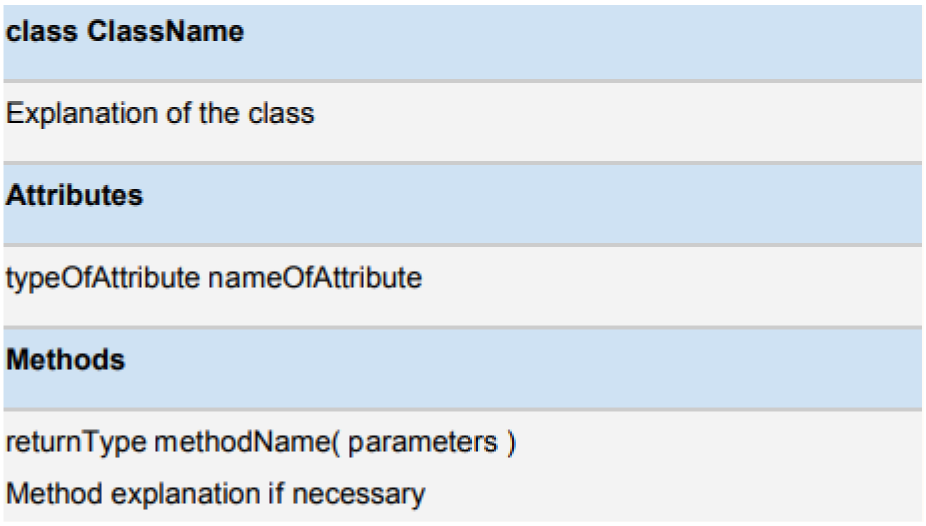
1.1.4 Integration with Third-Party Devices vs. System Independence:

Recognizing the importance of location information, SAROP establishes communication with GPS devices. This trade-off involves integrating with third-party devices for real-time location data while ensuring the overall system's independence and adaptability to various devices commonly used by search and rescue teams.

By carefully balancing these trade-offs, SAROP expects to offer a powerful operation portal that improves the efficiency of search and rescue teams while addressing unique difficulties.

1.2 Interface Documentation Guidlenes

Considering that SAROP is an open-source project, emphasizing the significance of comprehensibility in the interface documentation becomes even more crucial. Clear and well-documented interfaces play a pivotal role in ensuring that developers, whether they are contributors or users of the project, can readily understand how to interact with different components. The following class interface description outlines the best practices for documenting interfaces:



class ClassName
Explanation of the class
Attributes
typeOfAttribute nameOfAttribute
Methods
returnType methodName(parameters) Method explanation if necessary

1.3 Engineering standards

In creating this report, we followed the UML (Unified Modeling Language) principles for depicting class interfaces, diagrams, scenarios, and subsystem compositions.

UML, as a widely recognized standard, offers a user-friendly and effective method of creating these graphical representations.

The use of UML ensures clarity and consistency when depicting the structure and relationships of our project. Its simplicity and understandability make it an excellent alternative for communicating complicated software architecture and design concepts.

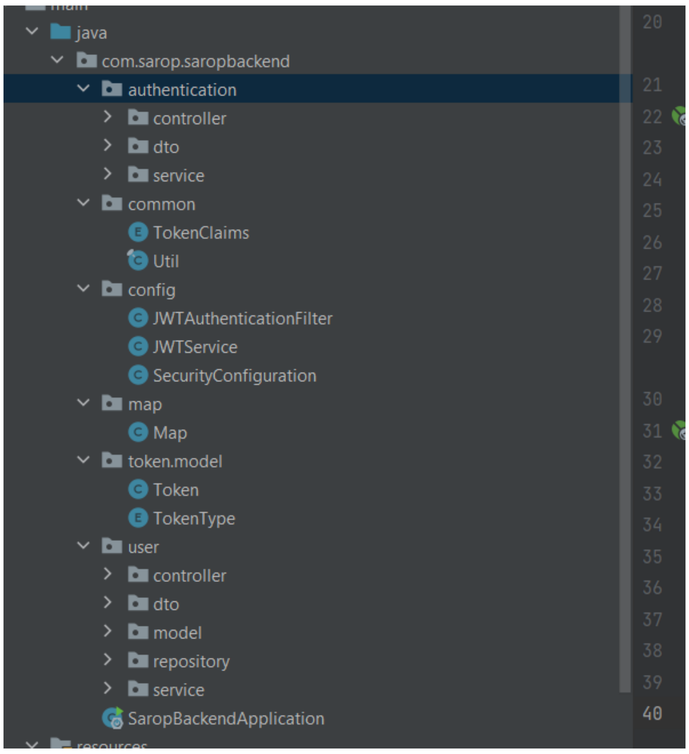
Furthermore, to ensure intellectual rigor and integrity, this study carefully follows the Institute of Electrical and Electronics Engineers' (IEEE) citation requirements. Following IEEE citation requirements improves the authenticity and professionalism of our work by properly attributing sources of information and ideas.

This dedication to industry-standard methods not only improves the quality of our documentation, but it also aligns our work with established conventions, allowing for greater understanding and collaboration among the professional and academic communities.

1.4 Definitions, acronyms, and abbreviations

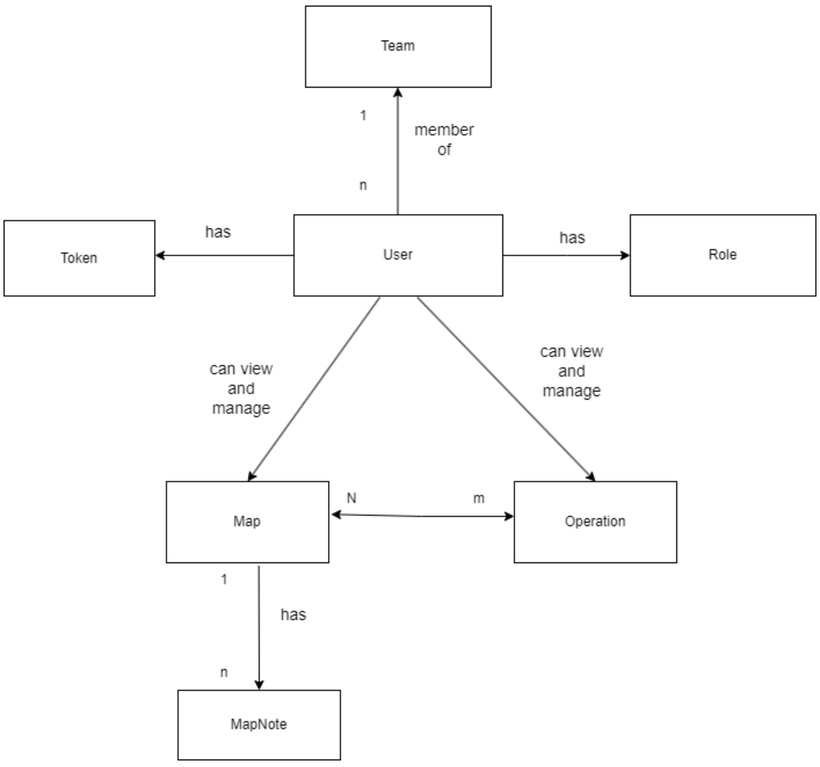
SAROP: Search and Rescue Operations Platform
UML: Unified Modeling Language
IEEE: Institute of Electrical and Electronics Engineers

2.Packages



```
java
  com.sarop.saropbackend
    authentication
      controller
      dto
      service
    common
      TokenClaims
      Util
    config
      JWTAuthenticationFilter
      JWTService
      SecurityConfiguration
    map
      Map
    token.model
      Token
      TokenType
    user
      controller
      dto
      model
      repository
      service
    SaropBackendApplication
```

The package structure of the program is above. This package structrure is designed based on features and it is in a modular monolith design. It allows us to find related code and parts easily. It also makes application to debug in a simpler way.



User: This class covers the basic properties of the users in the system. It has basic attributes such as id, first name, last name, email, password, role in the system and team. User has two role which are admin and user. User has also token to access the functionalities of system. User can be member of one of teams of search and rescue platform. Lastly, user can manage operations and maps based on the authorities that its role allow.

Role: There are two roles which are “Admin” and “User”. Admin can upload, delete and edit a map while users can only view and export a map. They can also add notes for related maps. Admin can create, update, view and delete an operation. Admin can also assign people and set a due date for the operation and attach some maps to operations as a relation. User can view, update, and tag people for the operation.

Token: Token will be used to access the functionalities in the system. There are two tokens such as access token and register token. Access tokens allow to access the capabilities of the system while refresh token allow the system to create new access tokens once access token is expired.

Team: Users are member of different teams inside system. Each team can be related with some operations and maps. This class have an id, name, team leader and a set of operations.

Operation: This class covers the basic properties such as id, name, location, team, map, assigned users and due date for a operation.

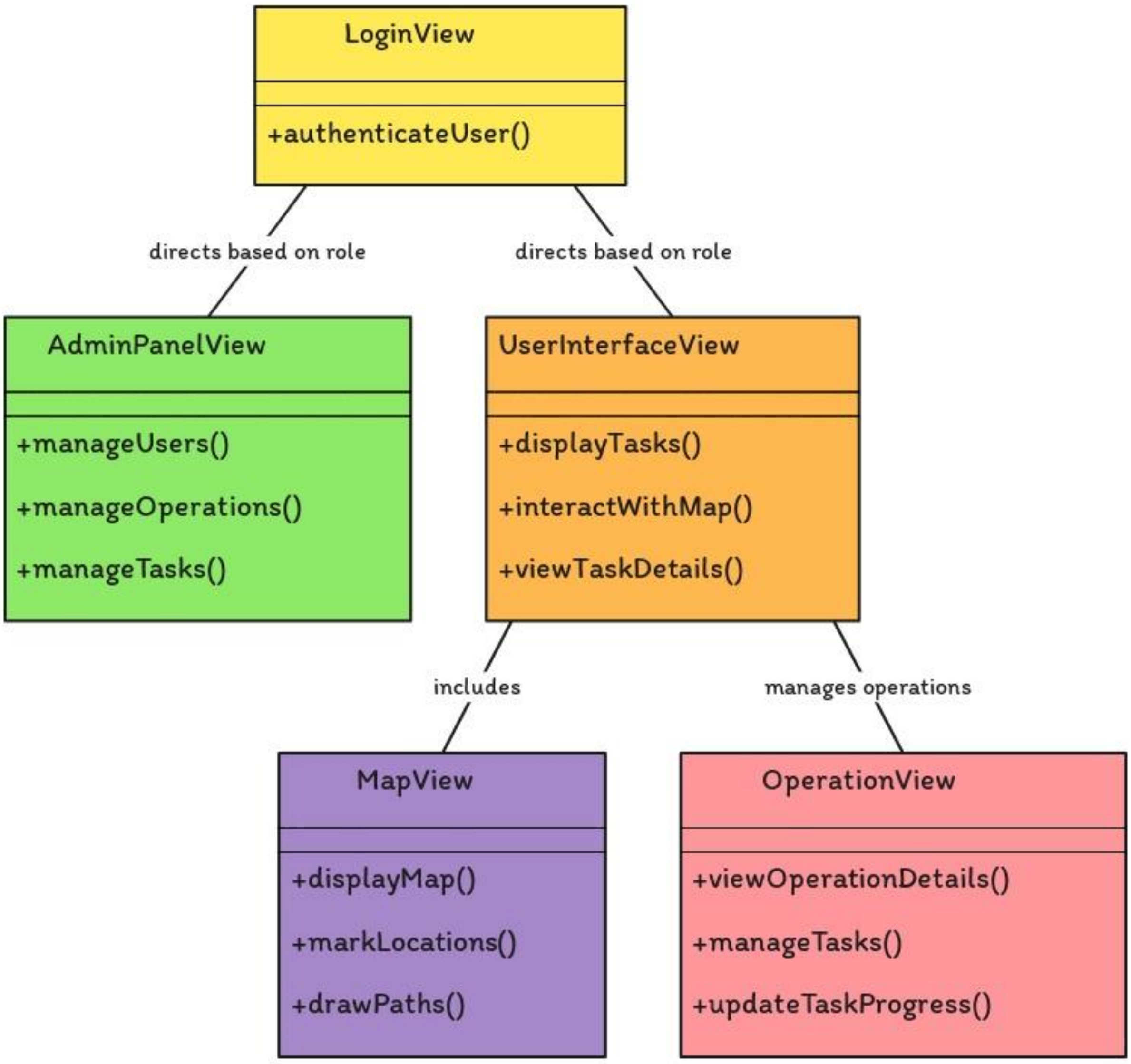
Map: This class covers the basic properties such as id, name, workspaceName, file url, operations and notes.

MapNote: This class operates as the notes which can be attached to a specific location on a map. It includes basic properties such as id, name, message, Map and location information.

2.1 Controller

- **AuthController:** This controller covers the operations such as register, login, logout and creation of a new access token from refresh token.
- **UserController:** This controller covers the operations like viewing all the users and making filters on the user list. It also allows users to change their passwords.
- **MapController:** This controller covers the related CRUD(Create-Read-Update-Delete) operations related to map. It also allows us to add notes on map.
- **OperationController:** This controller covers the related CRUD operations and some functios such as assigning person, setting due date, attaching map.

2.2 View



2.2.1 Login View:

- This view presents a form where users can input their username and password to authenticate and log into the system.
- It may include validation to ensure that the provided credentials are correct before allowing access.
- Upon successful login, users are redirected to either the Admin Panel View or the User Interface View based on their role.

2.2.2 Admin Panel View:

- This view provides a dashboard specifically designed for administrators.
- Admins can manage users, operations, and tasks from this view.
- It includes features such as:
    - **User management:** Add, edit, and delete users.
    - **Operation management:** Add, edit, and delete operations. Assign tasks to users.
    - **Task management:** View tasks associated with operations, edit task details, mark task progress, etc.
- Admins may have access to additional features or settings compared to regular users.

2.2.3 User Interface View:

This view serves as the main dashboard for regular users.

- Users can view tasks assigned to them, mark their current location on the map, add notes to tasks, etc.
- It includes features such as:
    - **Task overview:** Displays a list of tasks assigned to the user, along with relevant details such as status and deadlines.
    - **Map integration:** Displays the map view component where users can interact with maps, mark locations, draw paths, etc.
    - **Task details:** Allows users to view detailed information about each task, add notes, edit task details, etc.

- Users may have limited access compared to admins, focusing mainly on their assigned tasks and map interactions.

2.2.4 Map View:

- This view is dedicated to displaying maps and facilitating user interactions with them.
- It includes features such as:
    - **Map display:** Shows various maps relevant to search and rescue operations, including foundation maps and public maps.
    - **Location marking:** Allows users to mark their current location on the map and add notes or annotations.

        - **Path drawing:** Enables users to draw paths or routes on the map, useful for planning search and rescue operations.
- The map view may also support functionalities like selecting different map layers, zooming in/out, and panning.

2.2.5 Operation View:

- This view focuses specifically on managing search and rescue operations.
- It includes features such as:
    - **Operation overview:** Displays a list of ongoing operations, including basic details like name, status, and deadlines.
    - **Operation details:** Allows users to view detailed information about each operation, edit operation details, add tasks, etc.
    - **Task management:** Within each operation, users can manage tasks associated with it, assign tasks to team members, update task progress, etc.
- Admins have full control over operations, while regular users may have restricted access based on their assigned tasks.

These views collectively provide the necessary interface for users to interact with the SAROP system effectively, facilitating search and rescue operations and enhancing operational efficiency.

3. Class Interfaces

class Map
This class covers the basic properties such as id, name, workspaceName, file url, operations and notes.
Attributes

private int id;
private String name;
private String workspaceName;
private String fileUrl;
private List<Operation> operations;
private List<MapNote> notes;

class MapNote
This class operates as the notes which can be attached to a specific location on a map. It includes basic properties such as id, name, message, Map and location information.
Attributes

private int id;
private String name;
private String message
private Map map;
private String location;

class Team
Users are member of different teams inside system. Each team can be related with some operations and maps. This class have an id, name, team leader and a set of operations.
Attributes

private int id;
private String name;

private User teamLeader;
private List<User> users;
private List<Operation> operation;

class Token
Token will be used to access the functionalities in the system. There are two tokens such as access token and register token. Access tokens allow to access the capabilities of the system while refresh token allow the system to create new access tokens once access token is expired.
Attributes

private int id;
private String token;
private TokenType tokenType = TokenType.Bearer;
private boolean revoked;
private boolean expired;
private User user;

enum Role
There are two roles which are “Admin” and “User”. Admin can upload, delete and edit a map while users can only view and export a map. They can also add notes for related maps. Admin can create, update, view and delete an operation. Admin can also assign people and set a due date for the operation and attach some maps to operations as a relation. User can view, update, and tag people for the operation.
Attributes
USER, ADMIN

class AuthenticationController
This controller covers the operations such as register, login, logout and creation of a new access token from refresh token.
Attributes
private final AuthenticationService authenticationService;
Methods
public ResponseEntity<AuthenticationResponse> register(@RequestBody RegisterRequest request) : Register the user and returns access and refresh token
public ResponseEntity<AuthenticationResponse> login(@RequestBody LoginRequest request): Login the user and returns access and refresh token

class MapController
This controller covers the related CRUD(Create-Read-Update-Delete) operations related to map. It also allows us to add notes on map.
Attributes
private final MapService mapService;
Methods
public ResponseEntity<?> createWorkspace(@RequestBody WorkspaceRequest request);
public ResponseEntity<?> deleteWorkspace(@PathVariable int id);
public ResponseEntity<?> updateWorkspace(@RequestBody WorkspaceUpdateRequest request);
public ResponseEntity<?> getWorkspaces();
public ResponseEntity<?> createLayer(@RequestBody LayerRequest request);
public ResponseEntity<?> deleteLayer(@PathVariable int id);
public ResponseEntity<?> updateLayer(@RequestBody LayerUpdateRequest request);
public ResponseEntity<?> getLayers();

```
public ResponseEntity<?> addNoteToLayer(@RequestBody NoteToMapRequest request);
```

class User
This class covers the basic properties that user can have such as id, first name, last name, email, password, role (can be admin and typical user) and tokens.
Attributes

private String id;
private String firstName;
private String lastName;
private String email;
private String password;
private Role role;
private List<Token> tokens;

class Operation
This class covers the basic properties that operation can have such as id, operation name, operation's status, related responsibles, related maps, the due date for operation task and the description of operation.
Attributes

private String id;
private String opName;
private OperationStatus status;
private List<User> responsibles;
private List<Map> maps;
private String operationDescription;
private Date dueDate;

enum OperationStatus
This enum covers the status of the operation. There may be three different situations: they may not have started, they may be ongoing with operation, and they may be completed.
Attributes
TAMAMLANDI, DEVAM_EDİYOR, BAŞLANMADI

class UserController
This controller covers the operations such as change password and finding the all users.
Attributes
private final UserService service;
Methods
public ResponseEntity<?> changePassword(@RequestBody ChangePasswordRequest request, Principal connectedUser)
public ResponseEntity<?> findAllUsers(@RequestParam(required = false) Optional<String> email,@RequestParam(required = false) Optional<String> id)

class AdminController
This controller covers the related CRUD(Create-Read-Update-Delete) operations related to user. It also allow us to manage the whole user system by an admin.
Attributes
private final AdminService adminService;
Methods

public ResponseEntity<?> saveUser(@RequestBody UserSaveRequest userSaveRequest)
public ResponseEntity<?> updateUser(@PathVariable String id,@RequestBody UserUpdateRequest userUpdateRequest)
public ResponseEntity deleteUser(@PathVariable String id)

class OperationController
This controller covers the related CRUD (Create-Read-Update-Delete) operations related to operation.
Attributes
private final OperationServiceImpl operationService;
Methods
public ResponseEntity<?> findAllOperations()
public ResponseEntity<?> findById(@PathVariable String id)
public ResponseEntity<?> findByUserId(@PathVariable String userId)
public ResponseEntity<?> saveOperation(@RequestBody OperationSaveRequest operationSaveRequest)
public ResponseEntity<?> updateOperation(@PathVariable String id,@RequestBody OperationUpdateRequest operationUpdateRequest)
public ResponseEntity deleteOperation(@PathVariable String id)
public ResponseEntity<?> getOperationDescription(@PathVariable String id)
public ResponseEntity<?> getOperationDueDate(@PathVariable String id)

class LoginView
This class handles user authentication. The authenticate() method validates the provided username and password. If the credentials are correct, redirect_user() determines whether to redirect the user to the Admin Panel View or the User Interface View based on their role.
Attributes
private int id
String username;
String password;
Methods
Public boolean authenticate(): Validates credentials.
Public void redirect_user(): Redirects user based on

class AdminPanelView
This class provides an interface for administrators to manage users, operations, and tasks. It includes methods for adding, editing, and deleting users and operations, assigning tasks to users, and managing task details and progress.
Attributes
private List<User> user_list
private List<Operation> operation_list

private List<Task> task_list
Methods
public void add_user(User user)

public void edit_user(User user)
public void add_operation(Operation operation)
public void edit_operation(Operation operation)
public void delete_operation(int operationId)
public void assign_task(int taskId, int userId)
public List<Task> view_task()
public void edit_task_details(Task task)
public void mark_task_progress(int taskId, String progress)

class UserInterfaceView
This class serves as the main dashboard for regular users, allowing them to view tasks assigned to them, mark their current location on the map, add notes to tasks, and view/edit task details.
Attributes
private List<Task> assigned_tasks
private Location current_location

Method
public List<Task> view_tasks()
public void mark_location(Location location)
public void add_notes_to_task(int taskId, String note)
public Task view_task_details(int taskId)
public void edit_task_details(int taskId, Task taskDetails)

class MapView
Dedicated to displaying maps and facilitating user interactions, this class includes methods for marking locations, drawing paths, selecting map layers, and navigating the map.
Attributes

private Location current_location
private List<Path> path
Method
public void display_map()
public void mark_location(Location location)
public void draw_path(Path path)
public void select_map_layer(String layer)
public void zoom(int level)
public void pan(Direction direction) // Assuming Direction is an enum or class representing navigation directions.

class OperationView
Focuses on managing search and rescue operations. It allows users (primarily admins) to view and edit details of operations, manage tasks within those operations, including assigning tasks and updating task progress.
Attributes
private List<Operation> operation_list
private Operation selected_operation

Methods
public List<Operation> view_operations()
public Operation view_operation_details(int operationId)

public void edit_operation_details(int operationId, Operation operationDetails)
public void add_task(Task task)
public void assign_task(int taskId, int userId)
public void update_task_progress(int taskId, String progress)

These classes collectively model the interface and functionalities required for the SAROP system, enabling effective management and execution of search and rescue operations. The attributes provide the data each class needs to manage, while the methods define the actions that can be performed within each view.

4. References

https://medium.com/@chikim79/package-structure-for-modular-monolith-and-microservices-8526ad30b6a6

IBM, "UML - Basics," June 2003. [Online]. Available:

http://www.ibm.com/developerworks/rational/library/769.html. [Accessed 17-Feb-2019].

IEEE, "IEEE Citation Reference," September 2009. [Online]. Available:

https://m.ieee.org/documents/ieeecitationref.pdf. [Accessed 17-Feb-2019].