Application Requirements and Design

Real-time Geospatial Data Processor and Visualiser

Client: Werner Raath

COEUS

Members:
Nsovo Baloyi 12163262
Maluleki Nyuswa 13040686
Keletso Molefe 14222583
Kamogelo Tswene 12163555

9 September 2016
v0.3

Contents

Document History **2**

1 Introduction **3**

2 Vision **3**

3 Background **3**

4 Functional Requirements and Application Design **4**

4.1 System Overview 4

4.2 Use Cases 4

4.3 Use Case Prioritization 5

4.4 Modular System 5

4.4.1 Disaster Management Module 5

4.4.2 Disaster Management Module 6

4.4.3 Authentication Module 6

4.5 Open Issues 6

5 References **6**

Document History

Version	Date	Changed By	Summary
v0.1	27 May 2016	Nsovo Baloyi Maluleki Nyuswa Keletso Molefe Kamogelo Tswene	First Draft
v0.2	29 July 2016	Maluleki Nyuswa Kamogelo Tswene	Second Draft
v0.3	9 September 2016	Nsovo Baloyi Maluleki Nyuswa Keletso Molefe Kamogelo Tswene	Third Draft

1 Introduction

The purpose of this document is to outline the core functionality of the Real-time Geospatial Data Processor and Visualiser system in a technology neutral design specification. The design is not necessarily a depiction of what the final system will look like as during the development of the system it is possible that new functionalities are added, or old ones removed.

In this document we will thoroughly discuss and layout the project's functional requirements to provide a clear overview of the system as a whole. An agile approach is being followed which involves an interactive and incremental method of managing and designing the system as described by the scrum methodology.

2 Vision

The aim of this project is to provide a fully functional system to facilitate the processing and visualisation of geospatial data in real-time[1], including but not limited to climate and fire disasters. The system should be maintainable, with detailed supporting documentation for the Real-time Geospatial Data Processor and Visualiser system. When implemented, the system should be able to store, process and visualise large amounts of data in real-time.

3 Background

In the business of disaster management, being able to monitor a disaster's spread and change in near-real time is essential for assessing damages e.g. wildfire burned area, flooded area etc. and risk to human lives and assets.
There are numerous mature open-source technology for processing the voluminous real-time data, but systems to visualise this data is woefully incomplete and its development remains at an infancy stage. Thus we need to contribute to this quest for knowledge. The need for systems such as the one suggested is very high. The system can save lives and serve as a tool to do research amongst other things.

[1]real-time

4 Functional Requirements and Application Design

The purpose of this section is to outline the functional requirements and application design. The application requirements and design specify the the core functional requirements, i.e. the user's requirements around use cases and not secondary functional requirements arising from realizing non-functional requirements.

4.1 System Overview

Figure 1 shows a high-level overview of the system.

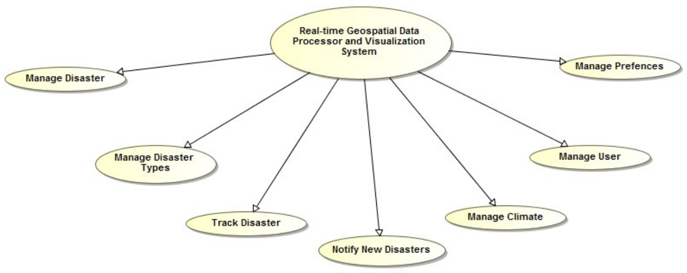


Figure 1: Overview of the System

4.2 Use Cases

Below is a list of all the use cases we have identified thus far:

- **Manage Disaster**
  Manage disaster is the use case that allows the user to query a specific disaster by type and time.
- **Track Disaster**
  Track disaster use case allows the user to track an active disaster. they will be able to get any new updates on the currently active disasters.
- **Manage Disaster Types**
  Manage Disaster Types allows an Admin user to add disaster types to the system.

- **Notify New Disaster**
  Notify New Disaster notifies a currently online user if there are any disasters that have just became active.
- **Manage Climate**
  This use case is similar to the manage disaster use case except that the user can only view climate details.
- **Manage User**
  This use case allows users to register their details to the system, as well as login and log out ogf the system.
- **Manage Prefences**
  This use case allows a logged in user to add, edit or view their weather or disaster prefences.

4.3 Use Case Prioritization

The system use cases are characterised as follows:

- **Manage Disaster - Critical**
  This is a critical since the main functionality of the system requires disaster management.
- **Track Disaster - Important**
  Users need to be able to track a disaster that has occurred for feedback purposes.
- **Manage Disaster Types - Important**
  An authorized user should be able to add a new or remove a redundant disaster type.
- **Notify New Disaster - Important**
  A user should be able to get instantaneous notification of a newly occurring disaster.
- **Manage Climate - Important**
  Normal weather climate should be properly managed by an authorized user for continuous display.
- **Manage User - Important**
  A user needs to be able to register with the system because as registered users, once they are logged into the system they are able to manage their weather and disaster prefences as well as log out of the system

once done. An authorized user (admin) once logged in is able to access and/or modify relevant system data.

- **Manage Prefences - Important**
  A registered user should be able to add, edit or view their weather and disaster prefences.

5 Modular System

5.1 Disaster Management Module

5.1.1 Scope

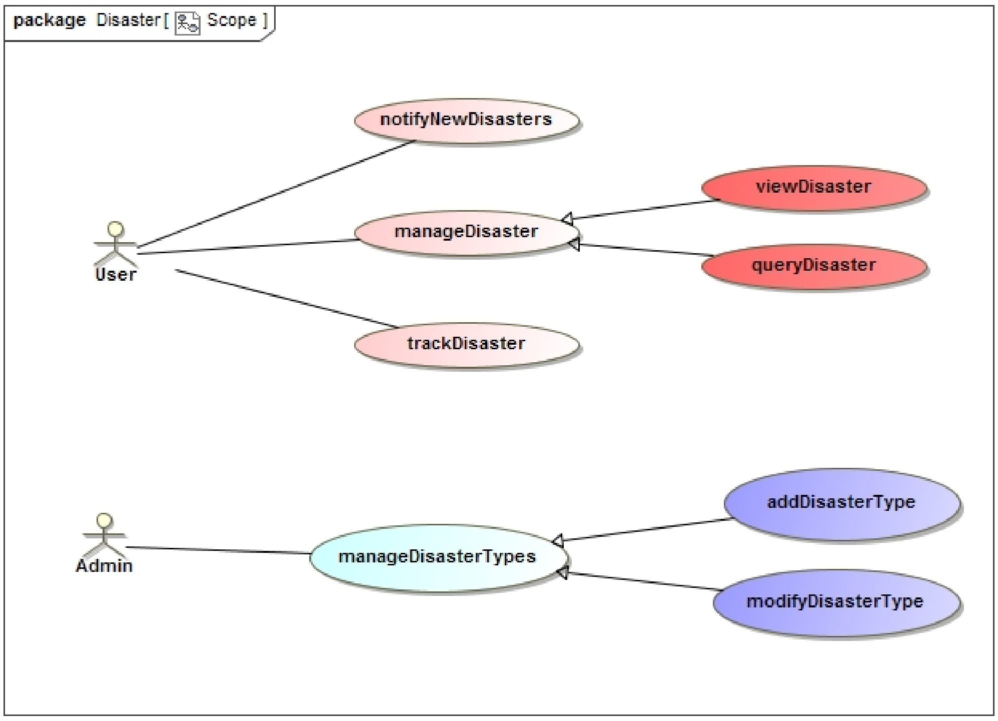


Figure 2: The scope of functionality required from the disaster module

Admin can add and/or modify the different kinds of disaster types that are required whilst users are able to track and manage disasters as well as get notifications for new disasters.

5.1.2 viewDisaster

A user can view disasters by going on to the disasters tab on the system and being able to view all the active disasters that are occurring throughout the world. Below are the service contract, activity diagram and functional requirements diagram for viewDisaster.

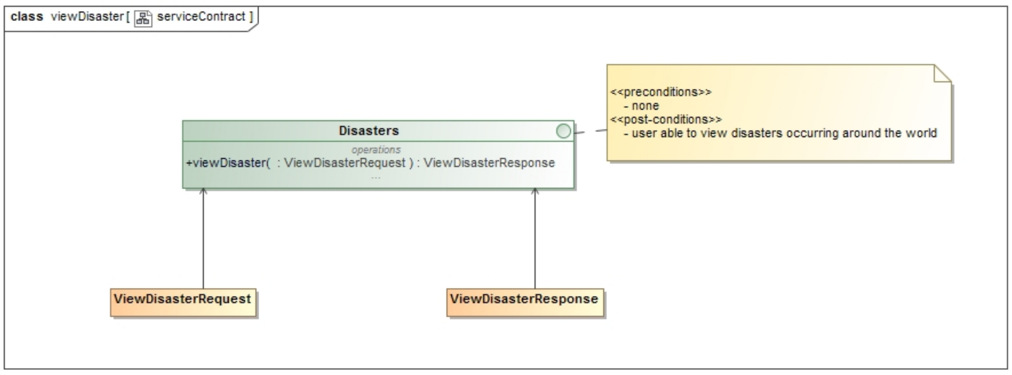

class viewDisaster [serviceContract]
Disasters
operations
+viewDisaster(: ViewDisasterRequest) : ViewDisasterResponse
...
<<preconditions>>
- none
<<post-conditions>>
- user able to view disasters occurring around the world
ViewDisasterRequest
ViewDisasterResponse

Figure 3: The service contract for viewDisaster

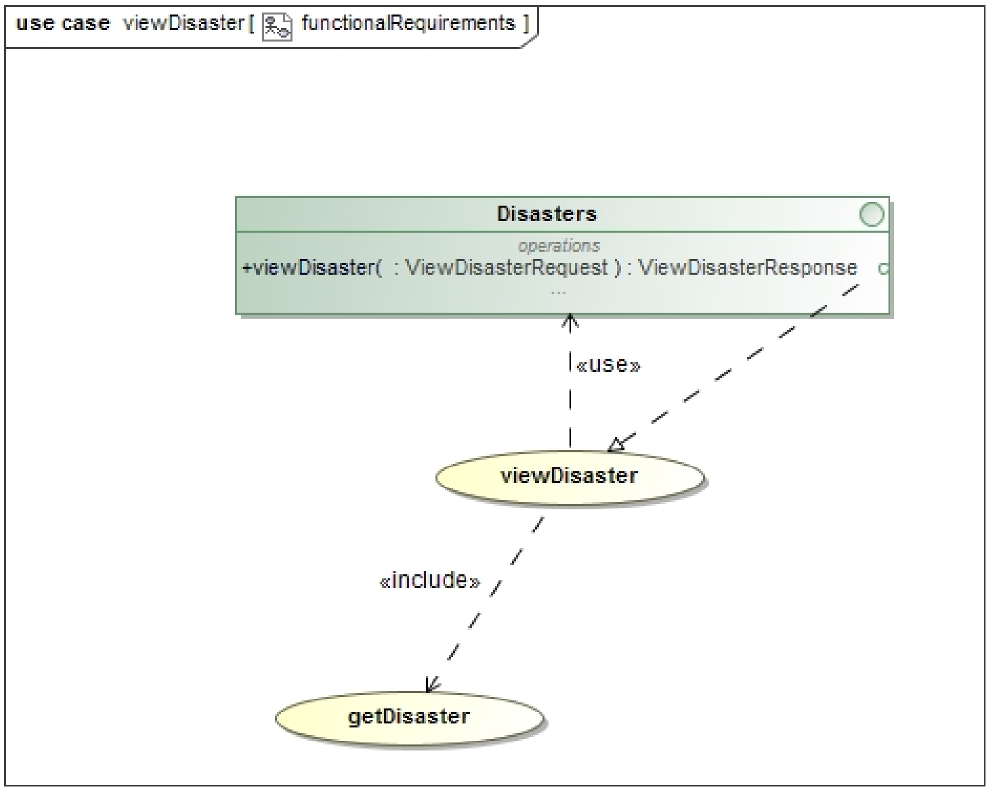

use case viewDisaster [functionalRequirements]
Disasters
operations
+viewDisaster(: ViewDisasterRequest) : ViewDisasterResponse
...
«use»
viewDisaster
«include»
getDisaster

Figure 4: The functional requirements diagram for viewDisaster

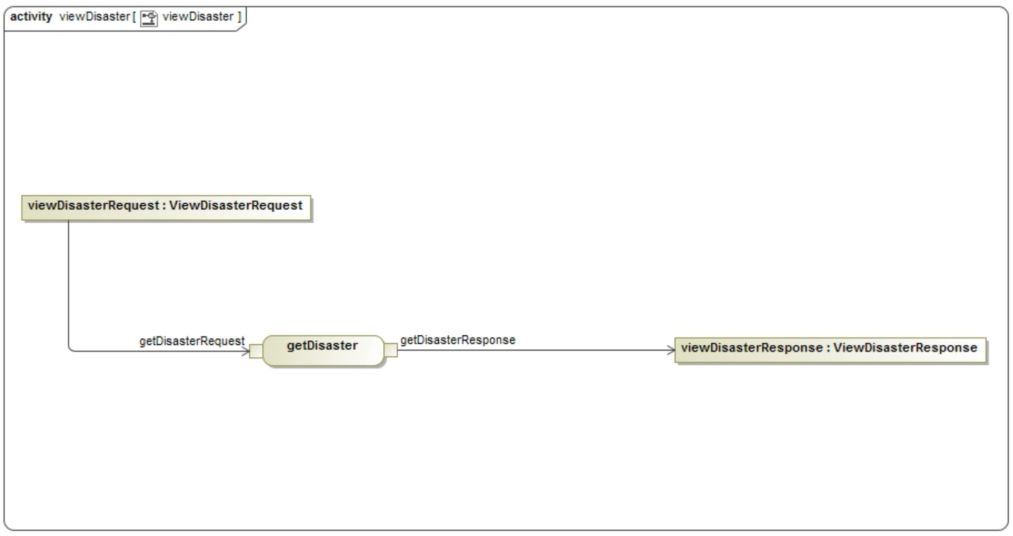

activity viewDisaster [viewDisaster]
viewDisasterRequest : ViewDisasterRequest
getDisasterRequest
getDisaster
getDisasterResponse
viewDisasterResponse : ViewDisasterResponse

Figure 5: The activity diagram for viewDisaster

5.1.3 queryDisaster

The user will be able to query the system for specific disasters, by entering search criteria, such as a start date and/or a end date, so that only disasters matching the specified criteria are returned to the user. Below are the service contract, activity diagram and functional requirements diagram for queryDisaster.

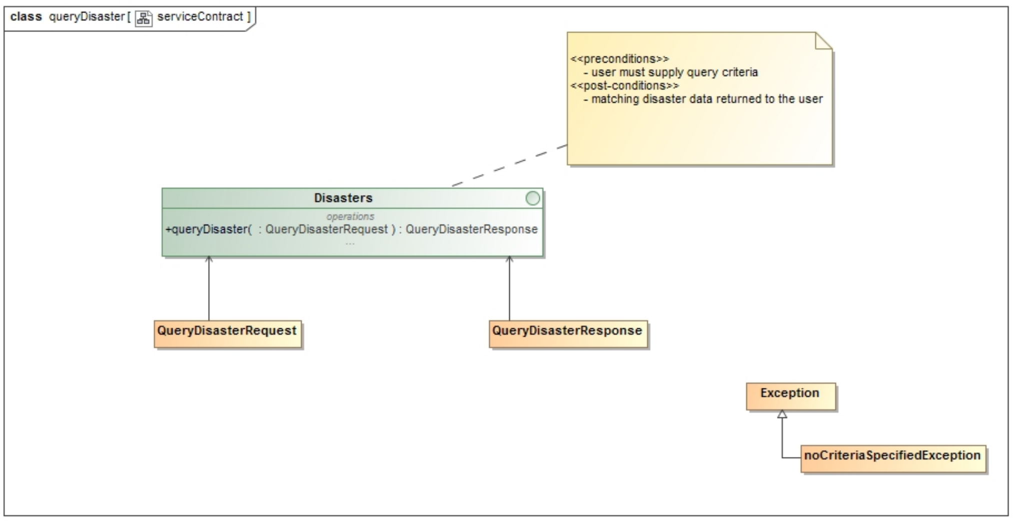

class queryDisaster [serviceContract]
<<preconditions>>
- user must supply query criteria
<<post-conditions>>
- matching disaster data returned to the user
Disasters
operations
+queryDisaster(: QueryDisasterRequest) : QueryDisasterResponse
...
QueryDisasterRequest
QueryDisasterResponse
Exception
noCriteriaSpecifiedException

Figure 6: The service contract for queryDisaster

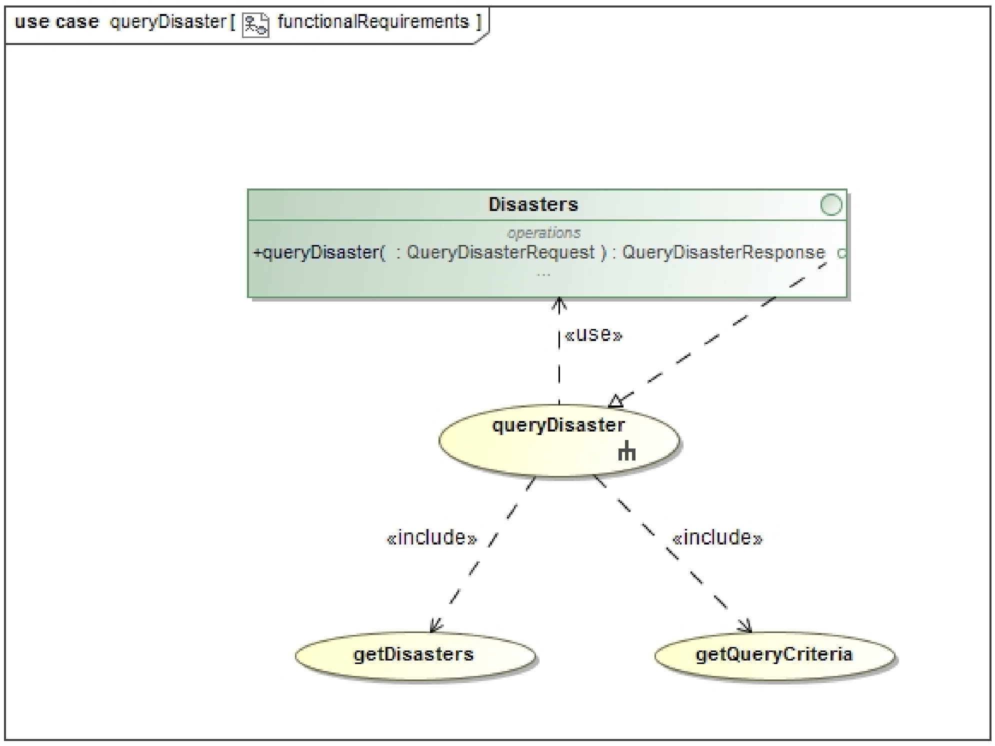

use case queryDisaster [functionalRequirements]
Disasters
operations
+queryDisaster(: QueryDisasterRequest) : QueryDisasterResponse
...
«use»
queryDisaster
«include»
«include»
getDisasters
getQueryCriteria

Figure 7: The functional requirements diagram for queryDisaster

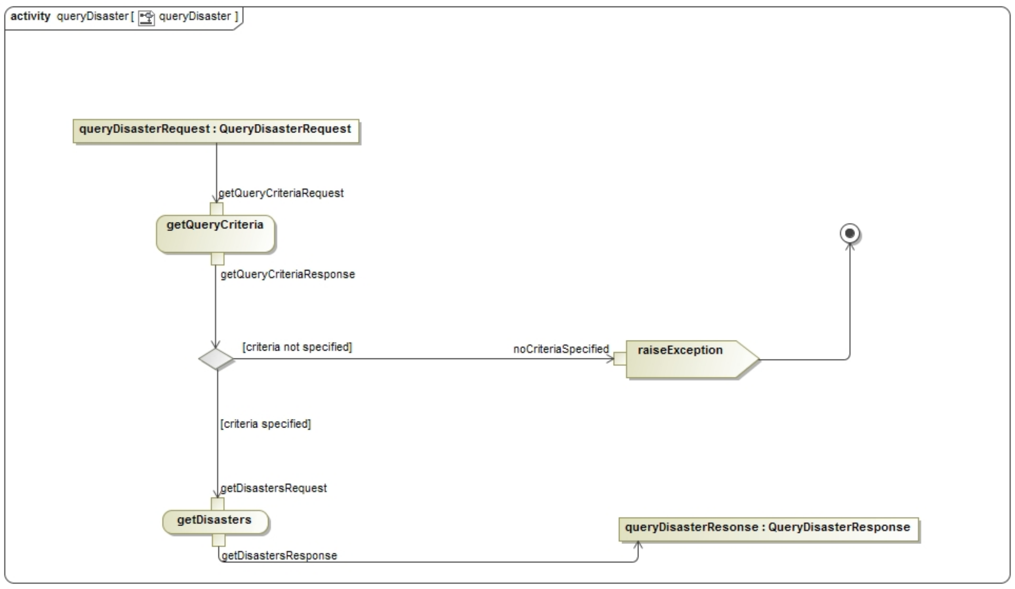


Figure 8: The activity diagram for queryDisaster

5.1.4 notifyNewDisaster

A user will be able to be notified of any new disasters that are occuring as of the time they are using the system. Below are the service contract, activity diagram and functional requirements diagram for notifyNewDisaster.

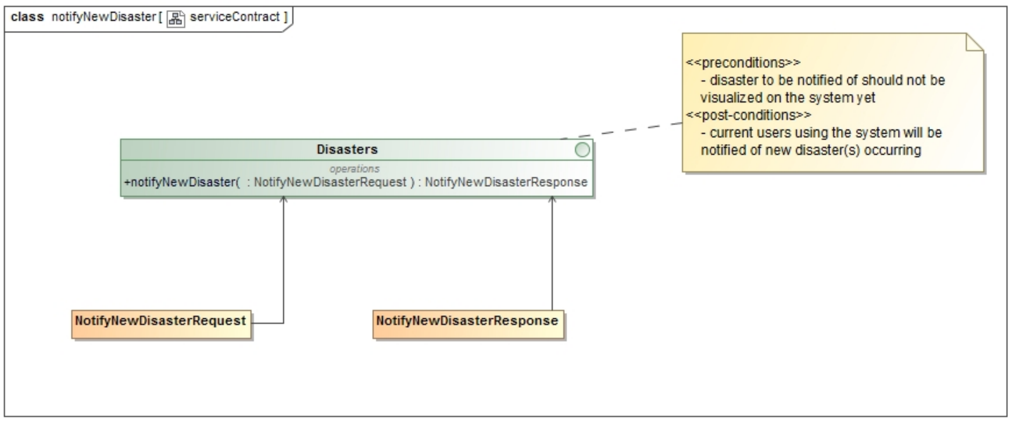

class notifyNewDisaster [serviceContract]
<<preconditions>>
- disaster to be notified of should not be visualized on the system yet
<<post-conditions>>
- current users using the system will be notified of new disaster(s) occurring
Disasters
operations
+notifyNewDisaster(: NotifyNewDisasterRequest) : NotifyNewDisasterResponse
NotifyNewDisasterRequest
NotifyNewDisasterResponse

Figure 9: The service contract for notifyNewDisaster

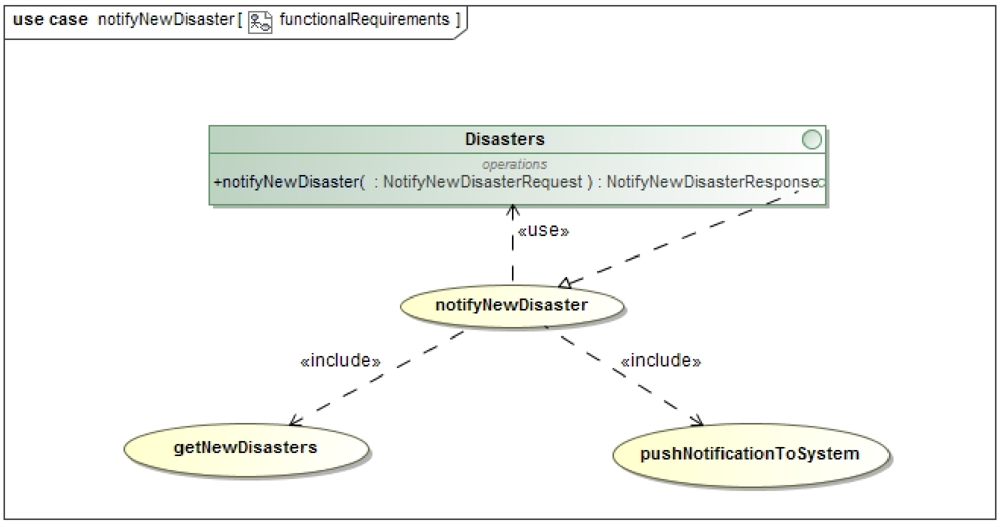

use case notifyNewDisaster [functionalRequirements]
Disasters
operations
+notifyNewDisaster(: NotifyNewDisasterRequest) : NotifyNewDisasterResponse
«use»
notifyNewDisaster
«include»
«include»
getNewDisasters
pushNotificationToSystem

Figure 10: The functional requirements diagram for notifyNewDisaster

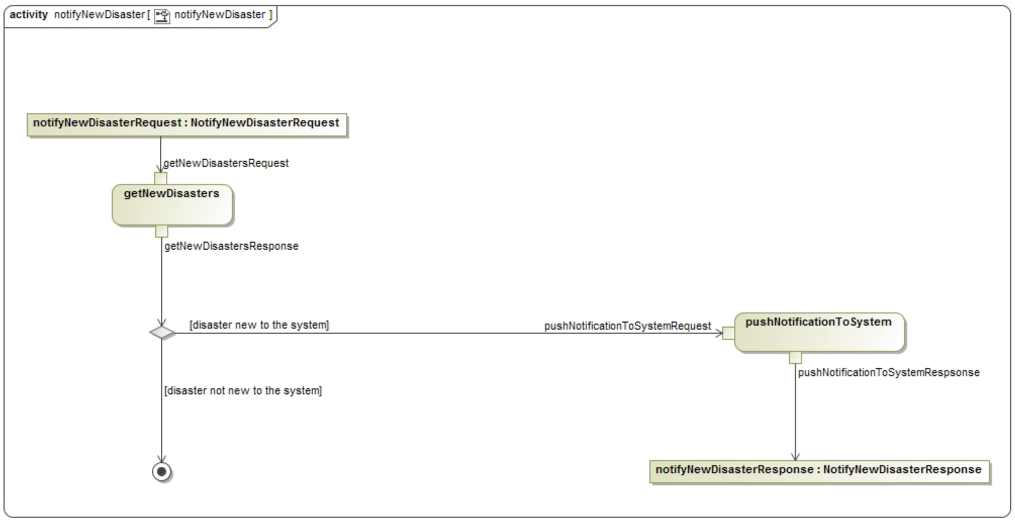

activity notifyNewDisaster [notifyNewDisaster]
notifyNewDisasterRequest : NotifyNewDisasterRequest
getNewDisastersRequest
getNewDisasters
getNewDisastersResponse
[disaster new to the system]
pushNotificationToSystemRequest
pushNotificationToSystem
pushNotificationToSystemRespsonse
[disaster not new to the system]
notifyNewDisasterResponse : NotifyNewDisasterResponse

Figure 11: The activity diagram for notifyNewDisaster

5.1.5 addDisasterType

Adding a disaster type requires that the user has admin rights and that there should not be a disaster type of the same name as the one being added that exists. If the user is admin and the disaster type being added is unique then a new disaster type will be added to the system. Below are the service contract, activity diagram and functional requirements diagram for addDisasterType.

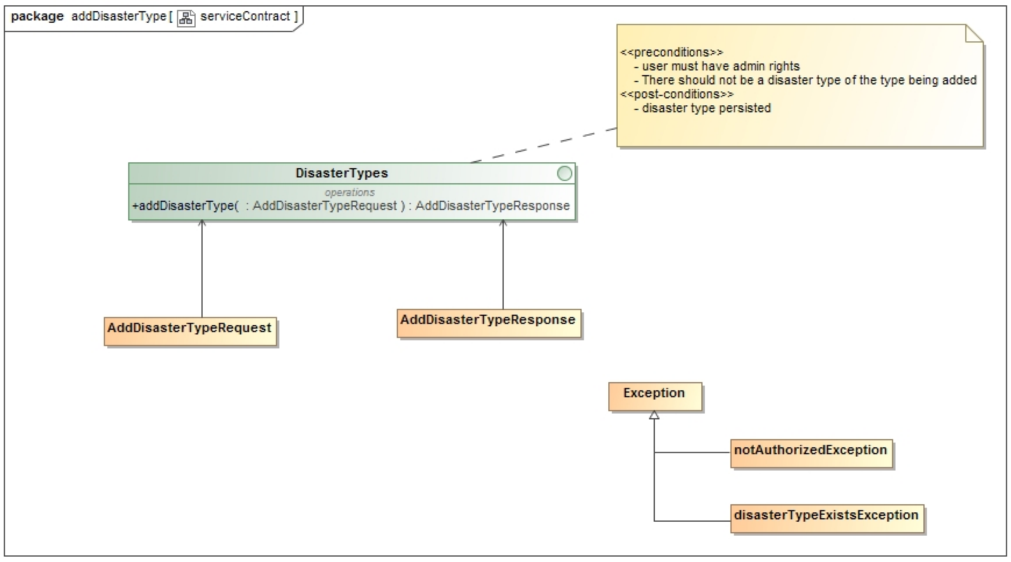

package addDisasterType [serviceContract]
<<preconditions>>
- user must have admin rights
- There should not be a disaster type of the type being added
<<post-conditions>>
- disaster type persisted
DisasterTypes
operations
+addDisasterType(: AddDisasterTypeRequest) : AddDisasterTypeResponse
AddDisasterTypeRequest
AddDisasterTypeResponse
Exception
notAuthorizedException
disasterTypeExistsException

Figure 12: The service contract for addDisasterType

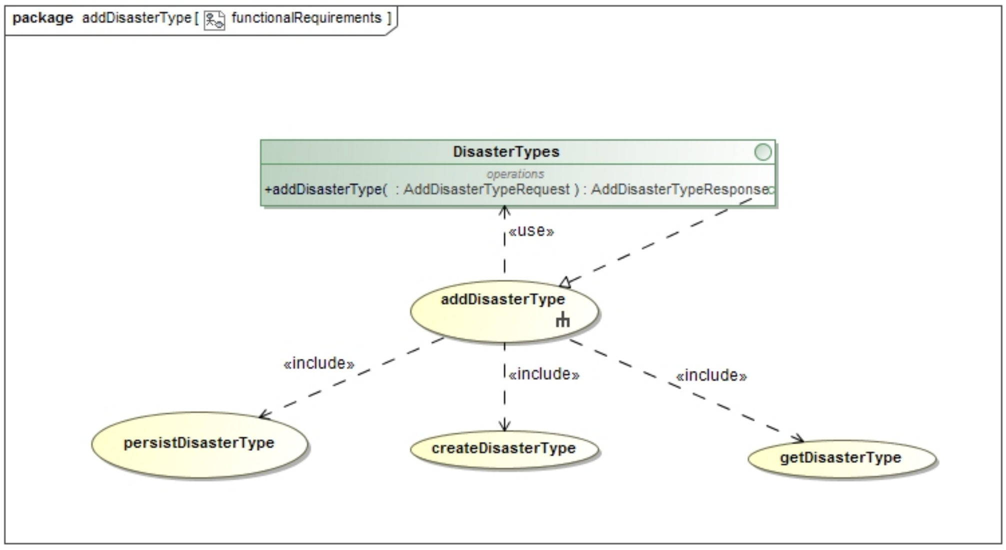


Figure 13: The functional requirements diagram for addDisasterType

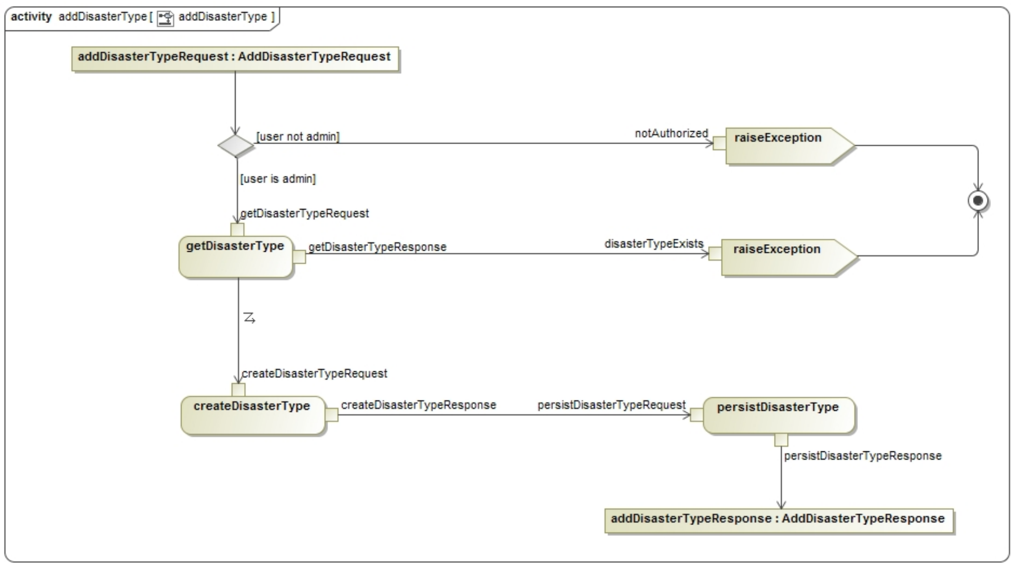
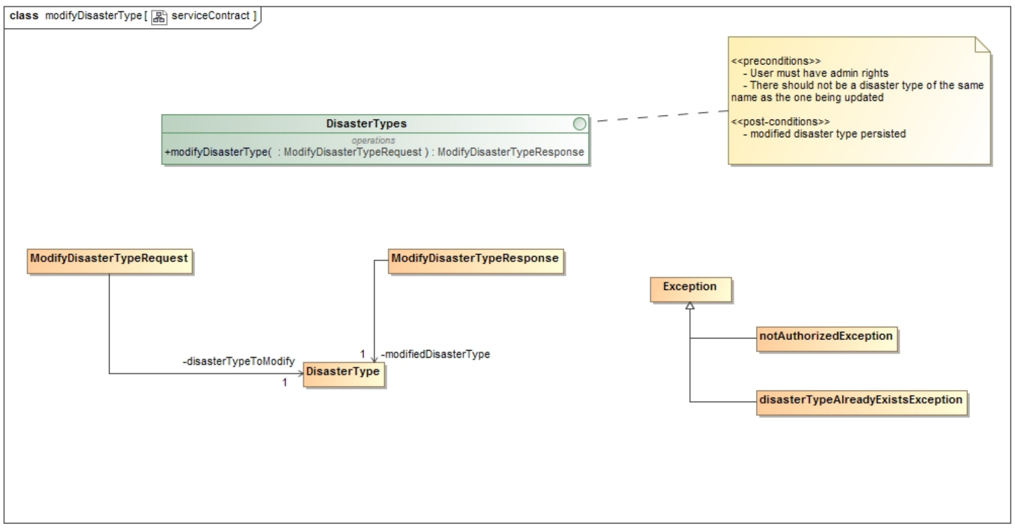

activity addDisasterType [addDisasterType]
addDisasterTypeRequest : AddDisasterTypeRequest
[user not admin]
notAuthorized
raiseException
[user is admin]
getDisasterTypeRequest
getDisasterType
getDisasterTypeResponse
disasterTypeExists
raiseException
createDisasterTypeRequest
createDisasterType
createDisasterTypeResponse
persistDisasterTypeRequest
persistDisasterType
persistDisasterTypeResponse
addDisasterTypeResponse : AddDisasterTypeResponse

Figure 14: The activity diagram for addDisasterType

5.1.6 modifyDisasterType

Modifying a disaster type requires that the user has admin rights and that there should not be a disaster type of the same name as the modified one that exists. If the user is admin and the modified disaster type is unique then the modified disaster type will be persisted to the database. Below are the service contract, activity diagram and functional requirements diagram for modifiyDisasterType.


class modifyDisasterType [serviceContract]
DisasterTypes
operations
+modifyDisasterType(: ModifyDisasterTypeRequest) : ModifyDisasterTypeResponse
ModifyDisasterTypeRequest
ModifyDisasterTypeResponse
-disasterTypeToModify
1
DisasterType
1
-modifiedDisasterType
Exception
notAuthorizedException
disasterTypeAlreadyExistsException

<<preconditions>>
- User must have admin rights
- There should not be a disaster type of the same name as the one being updated

<<post-conditions>>
- modified disaster type persisted

Figure 15: The service contract for modifyDisasterType

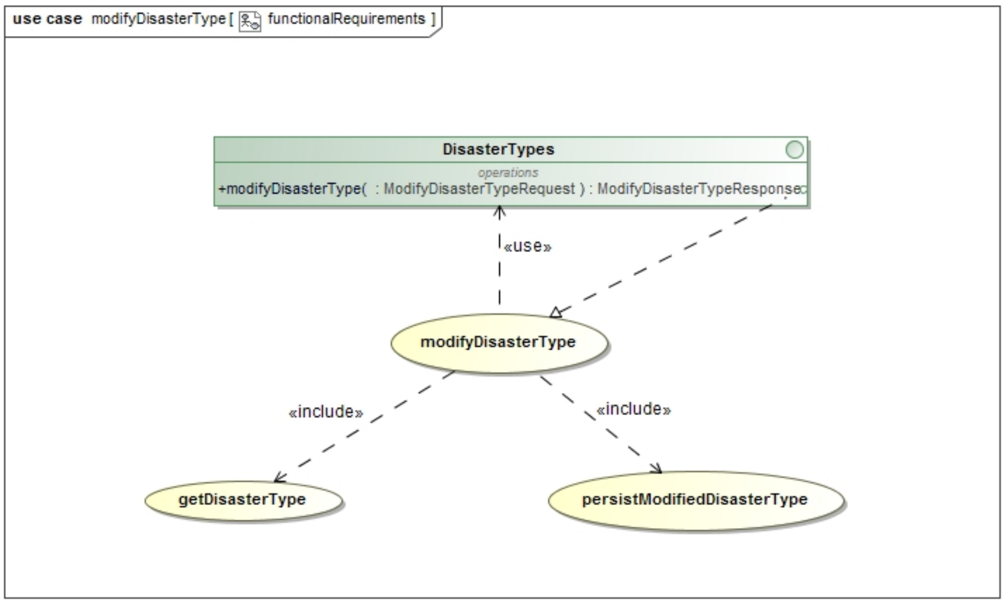

use case modifyDisasterType [functionalRequirements]
DisasterTypes
operations
+modifyDisasterType(: ModifyDisasterTypeRequest) : ModifyDisasterTypeResponse
«use»
modifyDisasterType
«include»
«include»
getDisasterType
persistModifiedDisasterType

Figure 16: The functional requirements diagram for modifyDisasterType

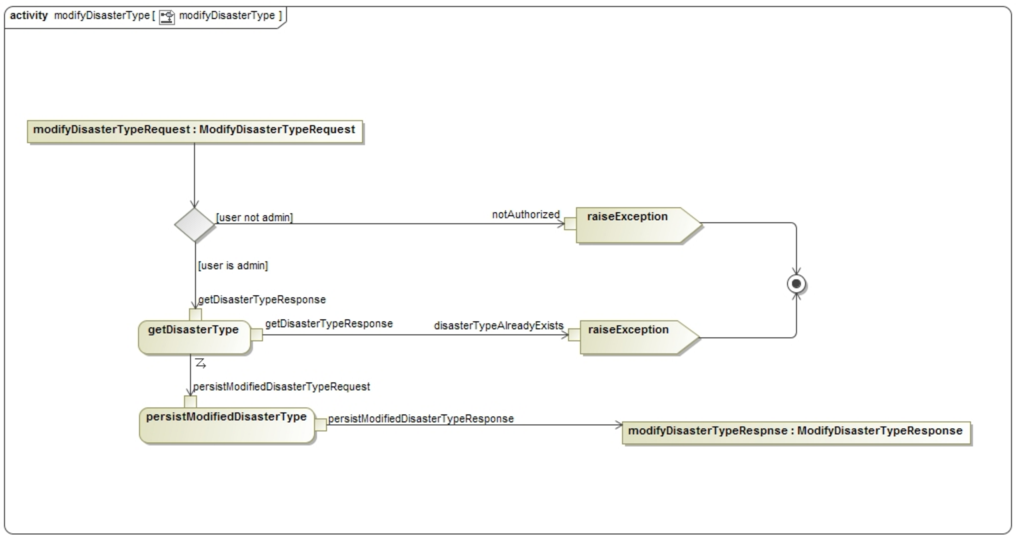

activity modifyDisasterType [modifyDisasterType]
modifyDisasterTypeRequest : ModifyDisasterTypeRequest
[user not admin]
notAuthorized
raiseException
[user is admin]
getDisasterTypeResponse
getDisasterType
getDisasterTypeResponse
disasterTypeAlreadyExists
raiseException
persistModifiedDisasterTypeRequest
persistModifiedDisasterType
persistModifiedDisasterTypeResponse
modifyDisasterTypeRespnse : ModifyDisasterTypeResponse

Figure 17: The activity diagram for modifyDisasterType

5.1.7 trackDisaster

5.2 Climate Management Module

5.2.1 Scope

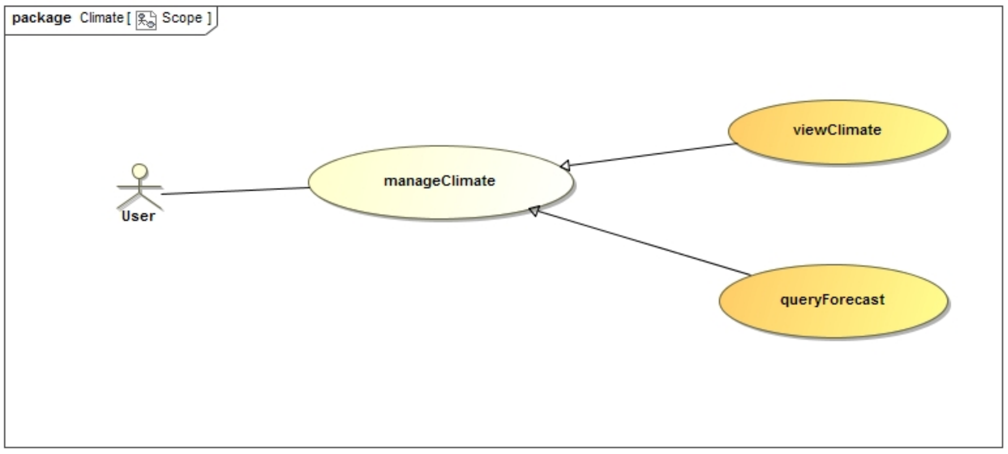


Figure 18: The scope of functionality required from the climate module

Users are able to view weathers and query forecasts.

5.2.2 viewWeather

A user can view weather immediately as they go into the system or by going on to the weather tab on the system. The user is able to see weather for the entire world or for a specific place. Below are the service contract, activity diagram and functional requirements diagram for viewWeather.

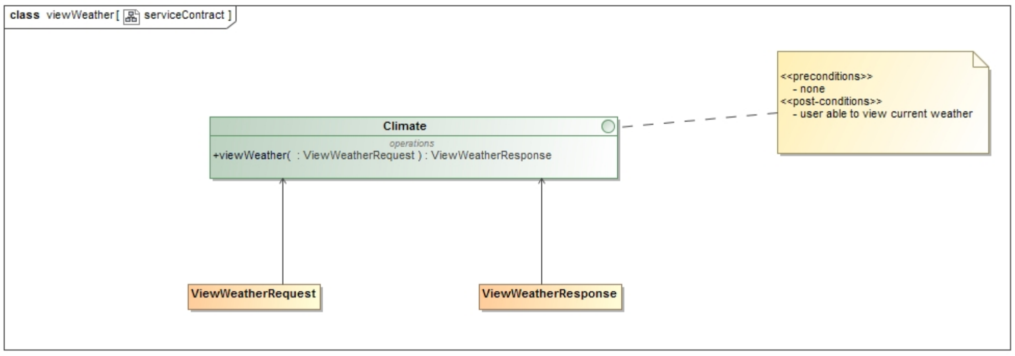

class viewWeather [serviceContract]
Climate
operations
+viewWeather(: ViewWeatherRequest) : ViewWeatherResponse
<<preconditions>>
- none
<<post-conditions>>
- user able to view current weather
ViewWeatherRequest
ViewWeatherResponse

Figure 19: The service contract for viewWeather

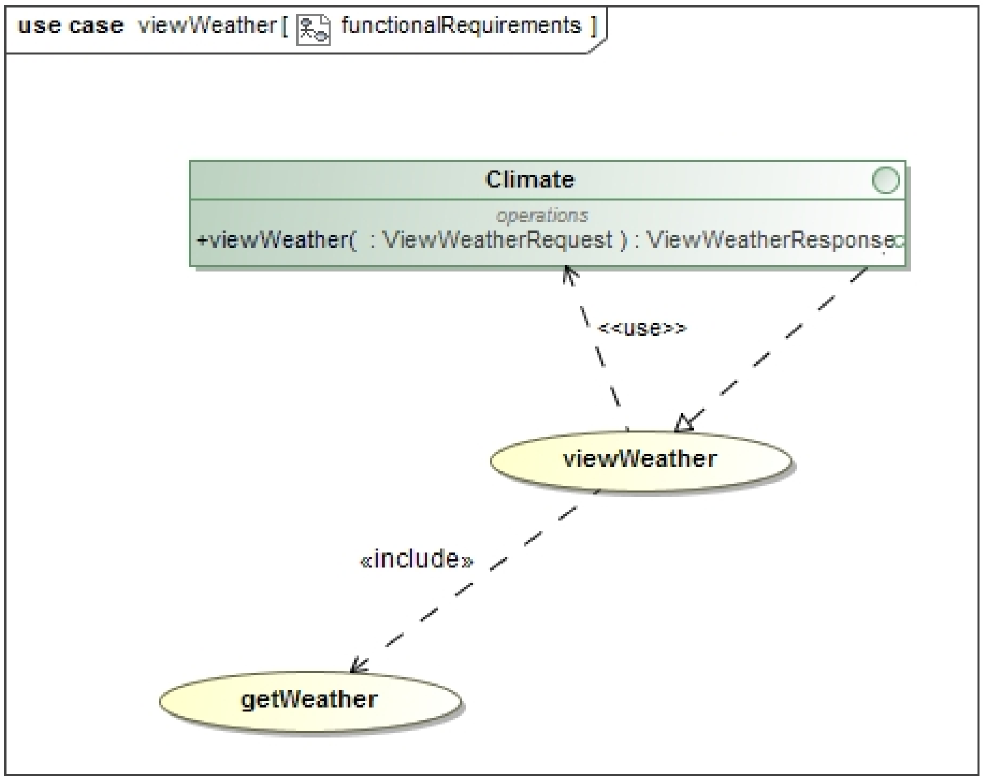

use case viewWeather [functionalRequirements]
Climate
operations
+viewWeather(: ViewWeatherRequest) : ViewWeatherResponse
<<use>>
viewWeather
«include»
getWeather

Figure 20: The functional requirements diagram for viewWeather

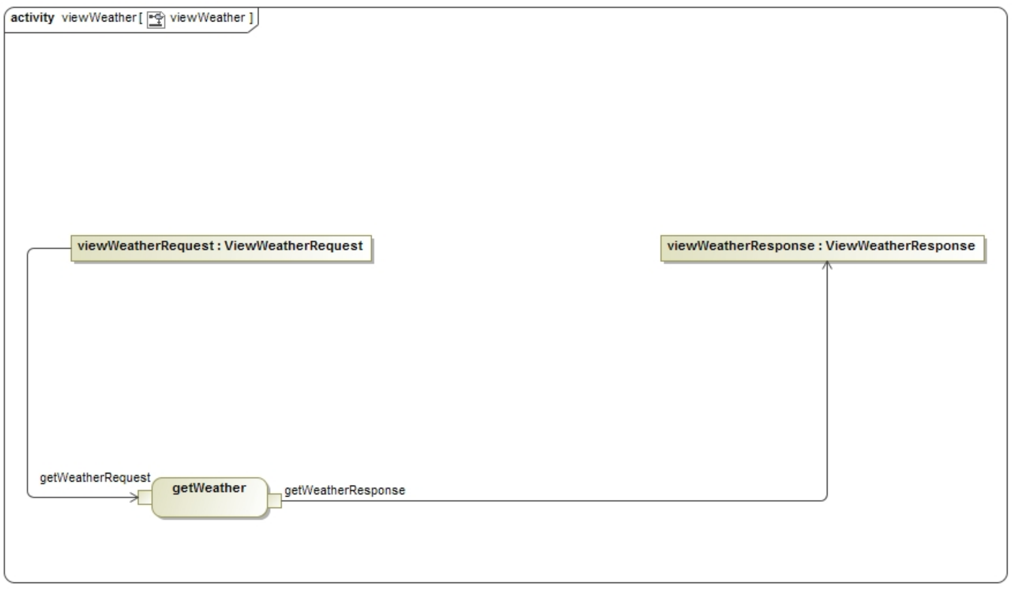

activity viewWeather [viewWeather]
viewWeatherRequest : ViewWeatherRequest
viewWeatherResponse : ViewWeatherResponse
getWeatherRequest
getWeather
getWeatherResponse

Figure 21: The activity diagram for viewWeather

5.2.3 queryForecast

5.3 User Module

5.3.1 Scope

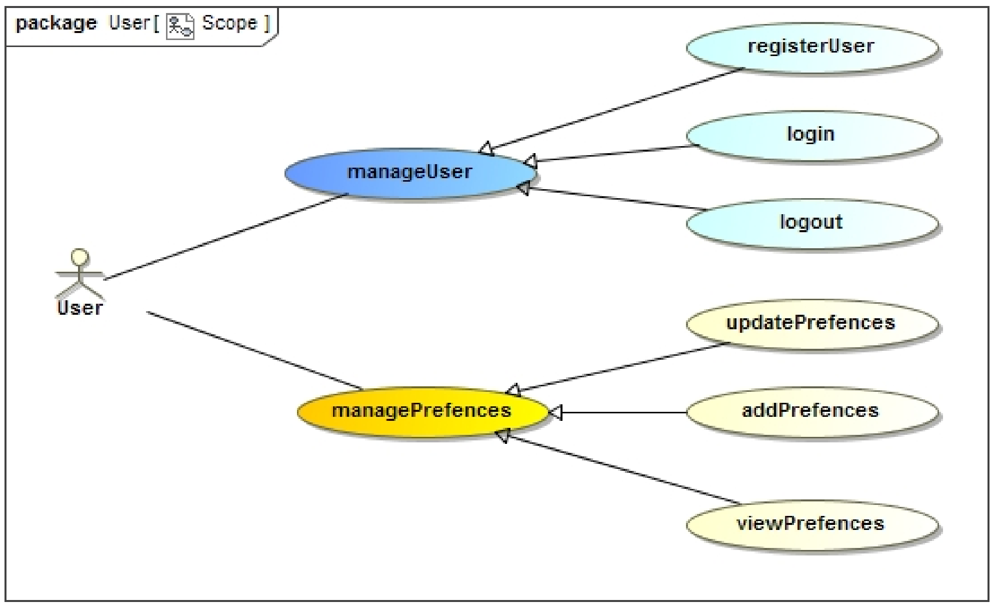


Figure 22: The scope of functionality required from the user module

A user can register and login in to the system to get access to functionality such as adding,viewing and editing prefences of weather or disaster data that they would like to be at their disposal. They are also ablo to logout of the system once done.

5.3.2 registerUser

A user can register with the system as long as they are not already registered. Below are the service contract, activity diagram and functional requirements diagram for registerUser.

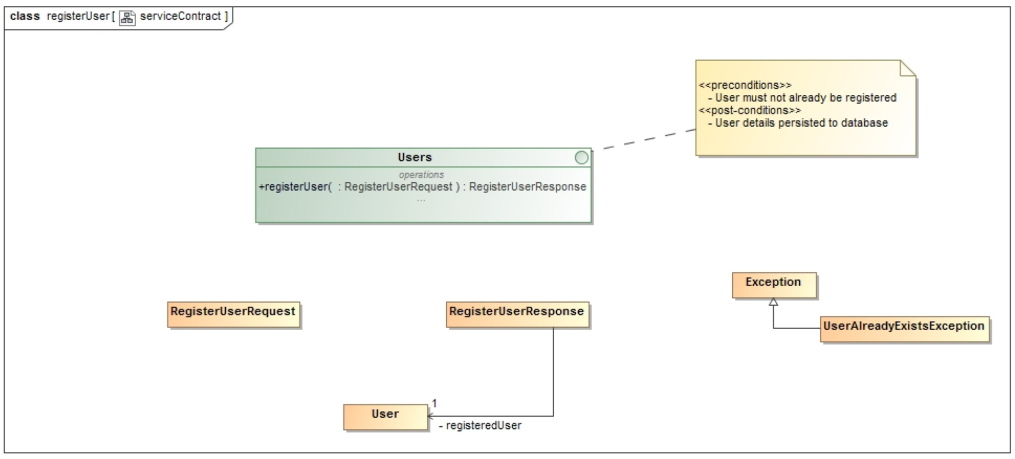

class registerUser [serviceContract]
<<preconditions>>
- User must not already be registered
<<post-conditions>>
- User details persisted to database
Users
operations
+registerUser(: RegisterUserRequest) : RegisterUserResponse
...
Exception
RegisterUserRequest
RegisterUserResponse
UserAlreadyExistsException
User
1
- registeredUser

Figure 23: The service contract for registerUser

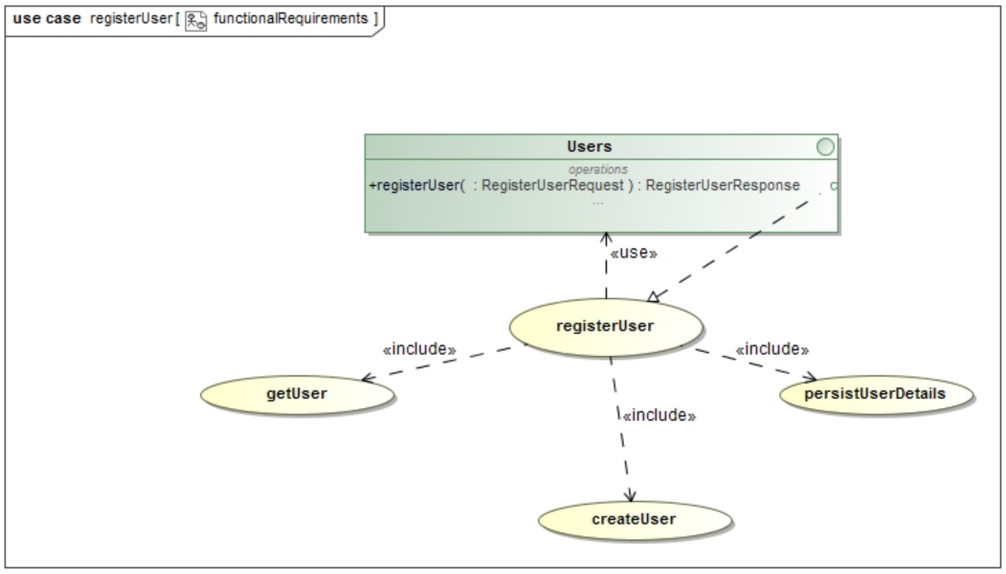

use case registerUser [functionalRequirements]
Users
operations
+registerUser(: RegisterUserRequest) : RegisterUserResponse
...
«use»
registerUser
«include»
getUser
«include»
persistUserDetails
«include»
createUser

Figure 24: The functional requirements diagram for registerUser

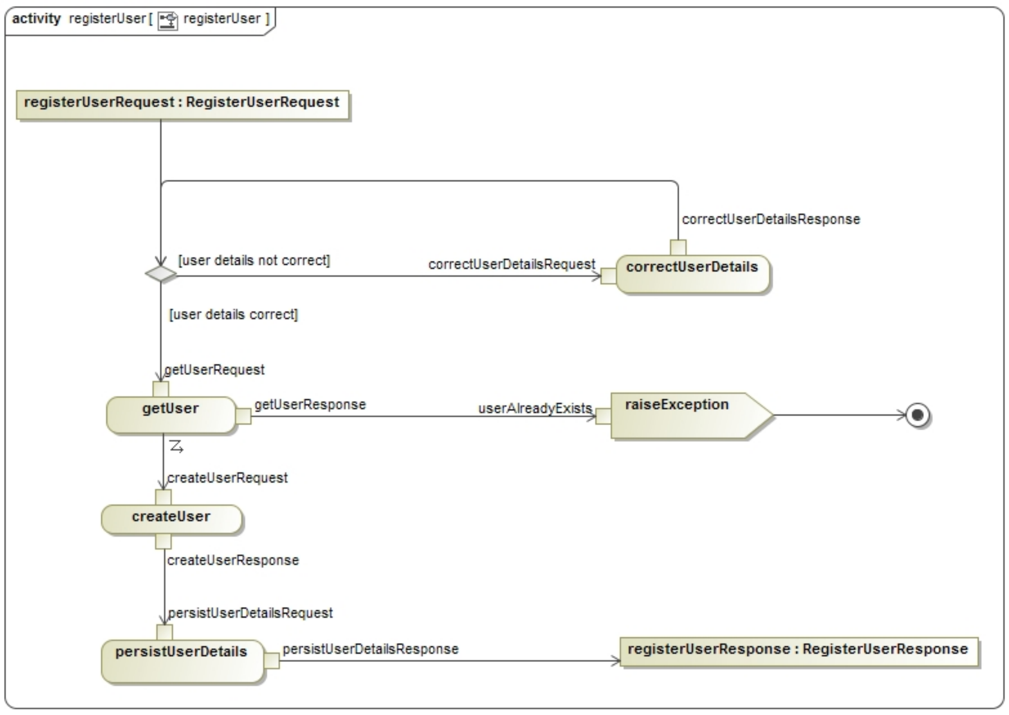

activity registerUser [registerUser]
registerUserRequest : RegisterUserRequest
correctUserDetailsResponse
[user details not correct]
correctUserDetailsRequest
correctUserDetails
[user details correct]
getUserRequest
getUser
getUserResponse
userAlreadyExists
raiseException
createUserRequest
createUser
createUserResponse
persistUserDetailsRequest
persistUserDetails
persistUserDetailsResponse
registerUserResponse : RegisterUserResponse

Figure 25: The activity diagram for registerUser

5.3.3 login

A registered user, given that they provide correct details to authenicate them, is able to access functionality such as adding and editing weather and disaster prefences once logged in. Below are the service contract, activity diagram and functional requirements diagram for login.

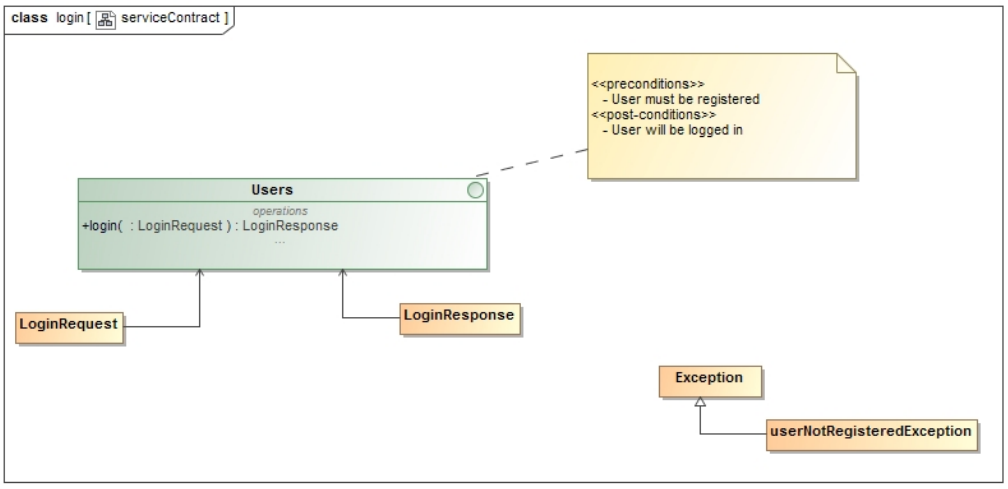

class login [serviceContract]
<<preconditions>>
- User must be registered
<<post-conditions>>
- User will be logged in
Users
operations
+login(: LoginRequest) : LoginResponse
...
LoginRequest
LoginResponse
Exception
userNotRegisteredException

Figure 26: The service contract for login

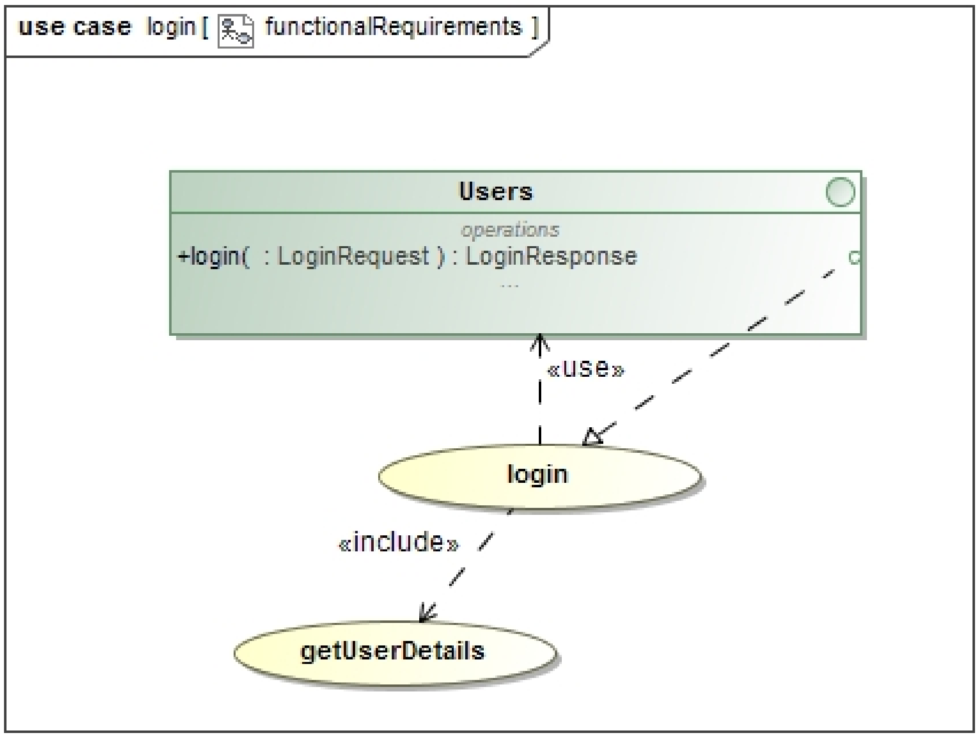

use case login [functionalRequirements]
Users
operations
+login(: LoginRequest) : LoginResponse
...
«use»
login
«include»
getUserDetails

Figure 27: The functional requirements diagram for login

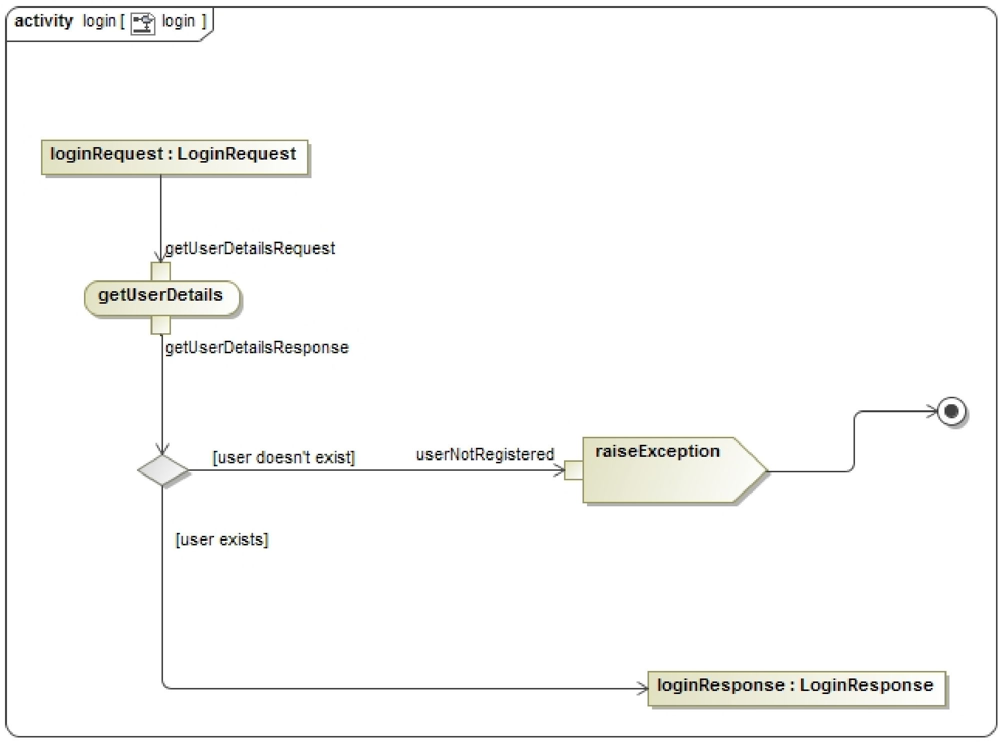


Figure 28: The activity diagram for login

5.3.4 logout

A user, given that they are logged in to the system, is able to logout of the system once done. Below are the service contract, activity diagram and functional requirements diagram for logout.

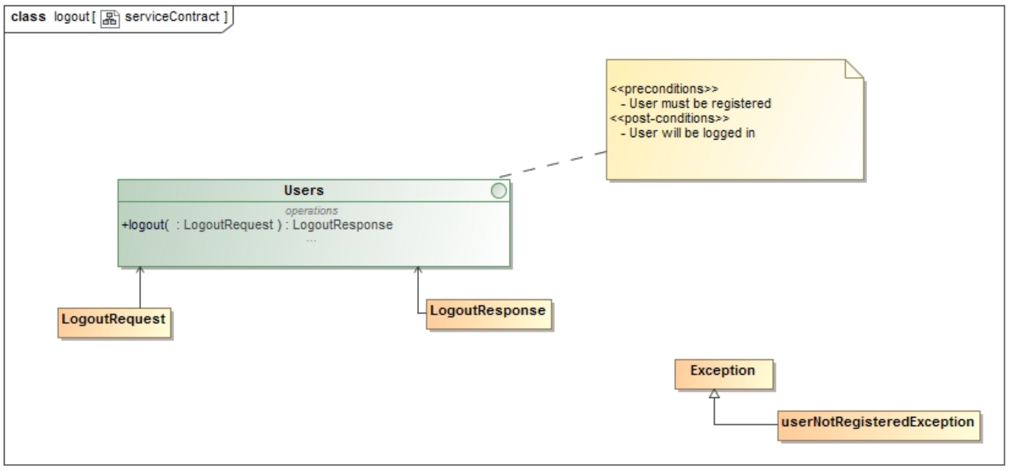

class logout [serviceContract]
<<preconditions>>
- User must be registered
<<post-conditions>>
- User will be logged in
Users
operations
+logout(: LogoutRequest) : LogoutResponse
...
LogoutRequest
LogoutResponse
Exception
userNotRegisteredException

Figure 29: The service contract for logout

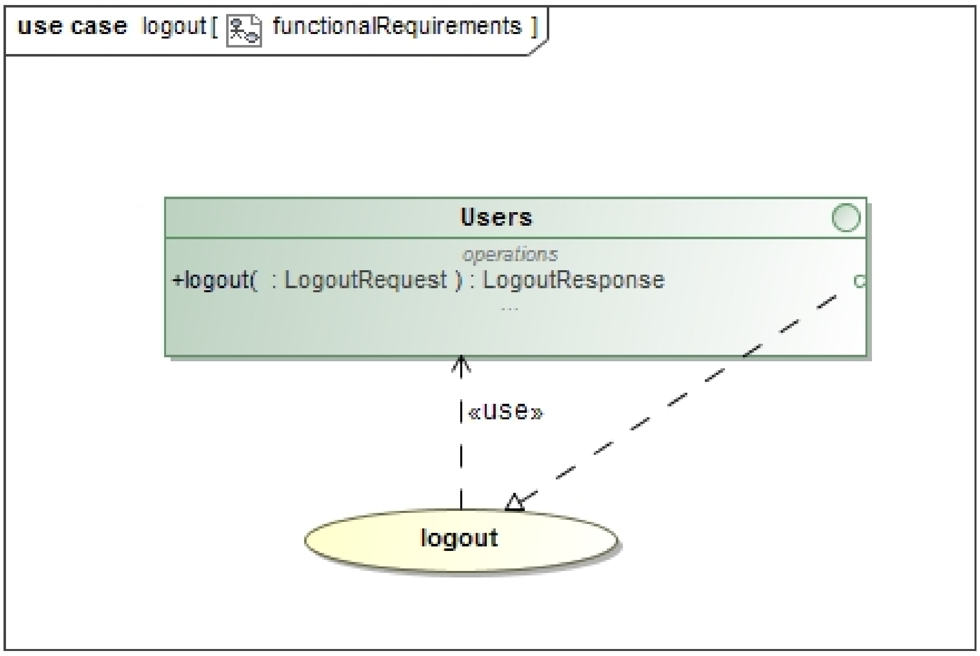

use case logout [functionalRequirements]
Users
operations
+logout(: LogoutRequest) : LogoutResponse
...
«use»
logout

Figure 30: The functional requirements diagram for logout

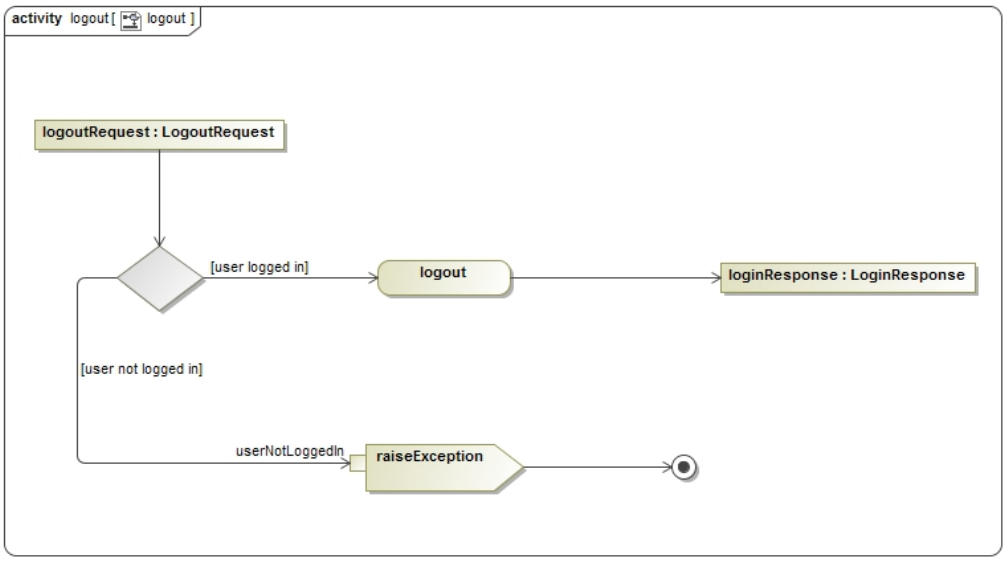

activity logout[logout]
logoutRequest : LogoutRequest
[user logged in]
logout
loginResponse : LoginResponse
[user not logged in]
userNotLoggedIn
raiseException

Figure 31: The activity diagram for logout

5.3.5 addPrefences

5.3.6 editPrefences

5.3.7 viewPrefences

6 Open Issues

7 References