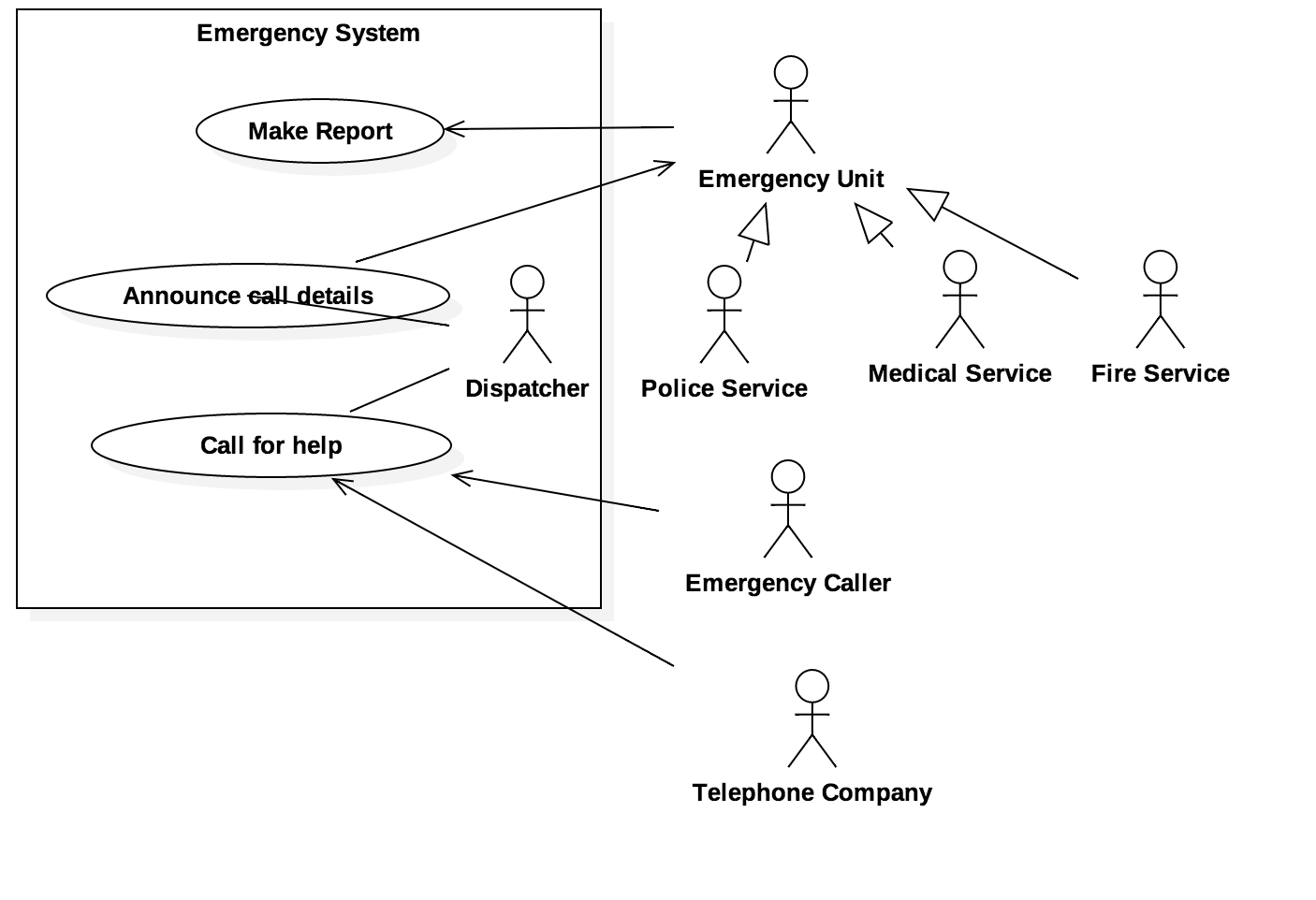
# Problem Statement

„Develop an application that is responsible for the organization and management of the process of an emergency call at an emergency dispatching center with their field units”

# Business Analysis

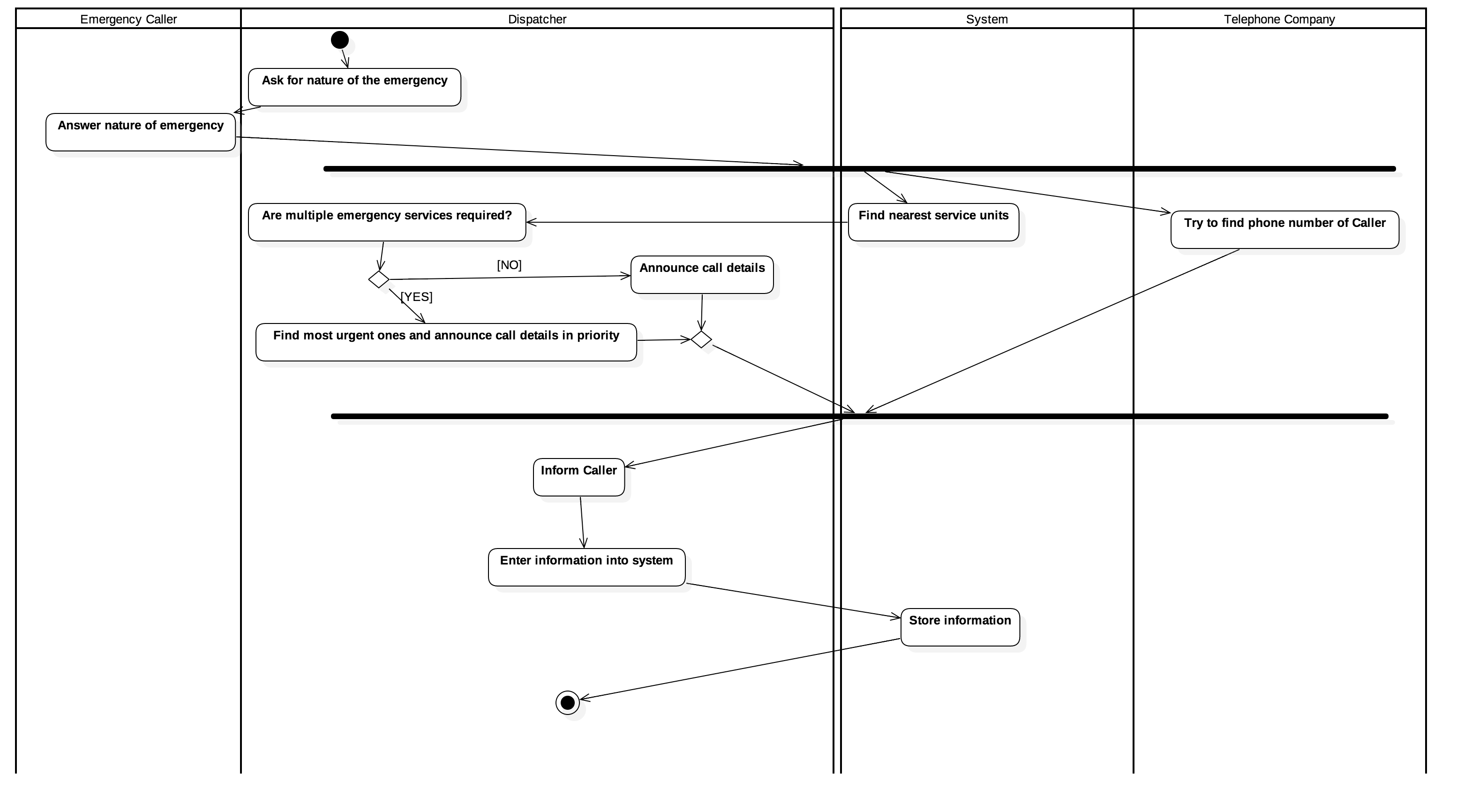
* Business Actors:
  + Emergency Unit
    - Police Service
    - Medical Service
    - Fire Service
  + Emergency Caller
  + Telephone Company
* Business Process:
  + „Manage good inputs and outputs in an emergency dispatching center“
* Business Agents:
  + Dispatcher
  + Emergency unit
* Automated Activities
  + Make Report
    - Complete Operation: No
    - Save complete time: Yes
    - Write report: Partly (a template)
    - Save report: Yes
  + Announce call details
    - Call appropiate service: Yes
    - Acknowledge: Yes
    - Announce necessary details: Partly (a template)
    - Acknowledge: No
  + Call for help
    - Ask for nature of the emergency: May be automated, but it’s better not to do that
    - Answer nature of emergency: No
    - Announce call details: Yes can be automated, partly
    - Find most urgent ones and announce call details in priority: Can be automated, partly (finding the urgent ones not, but calling the service)
    - Find nearest service units: Yes
    - Try to find phone number of Caller: Yes
    - Inform Caller: No
    - Enter information into system: Partly (template)
    - Store information: Yes

## Business Use Case Diagram

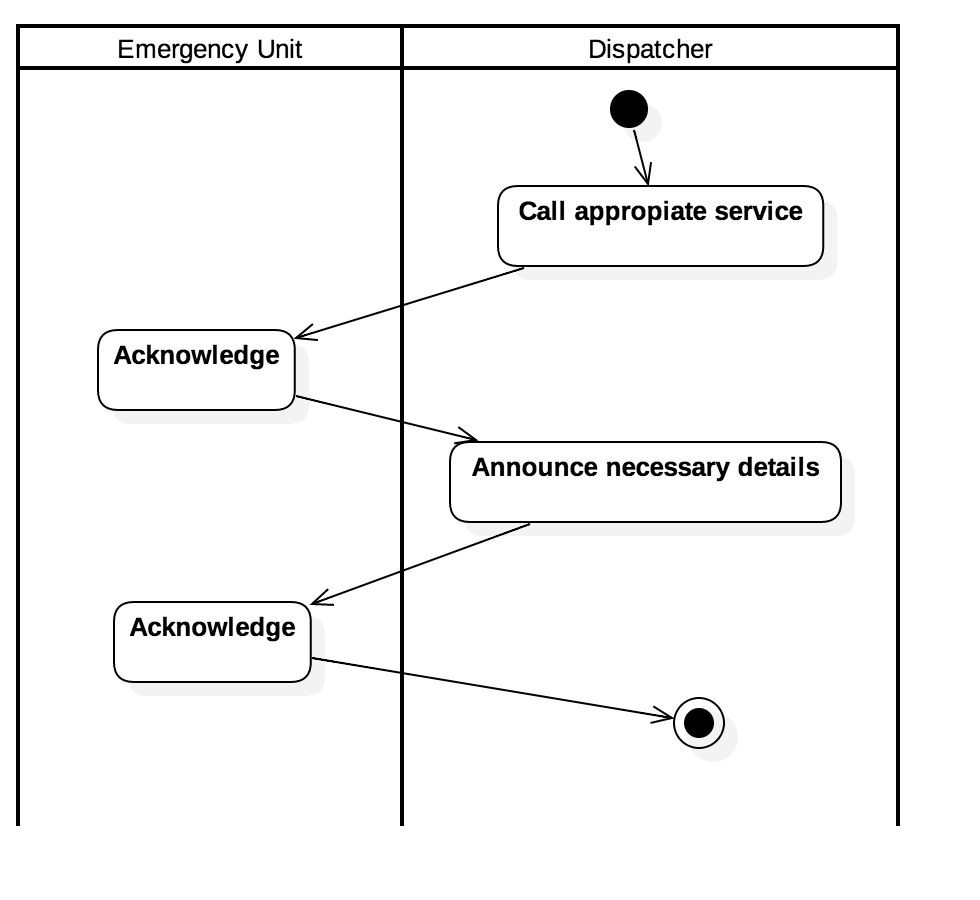


## Business Activity Diagram

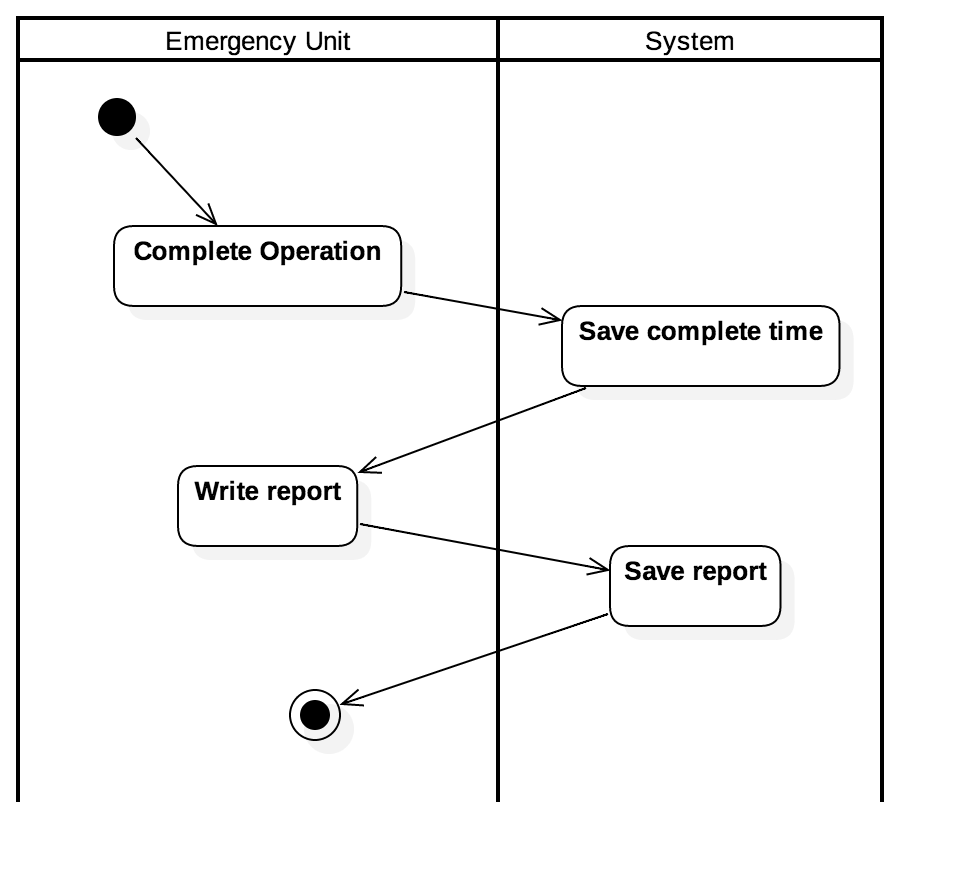
### Call for help



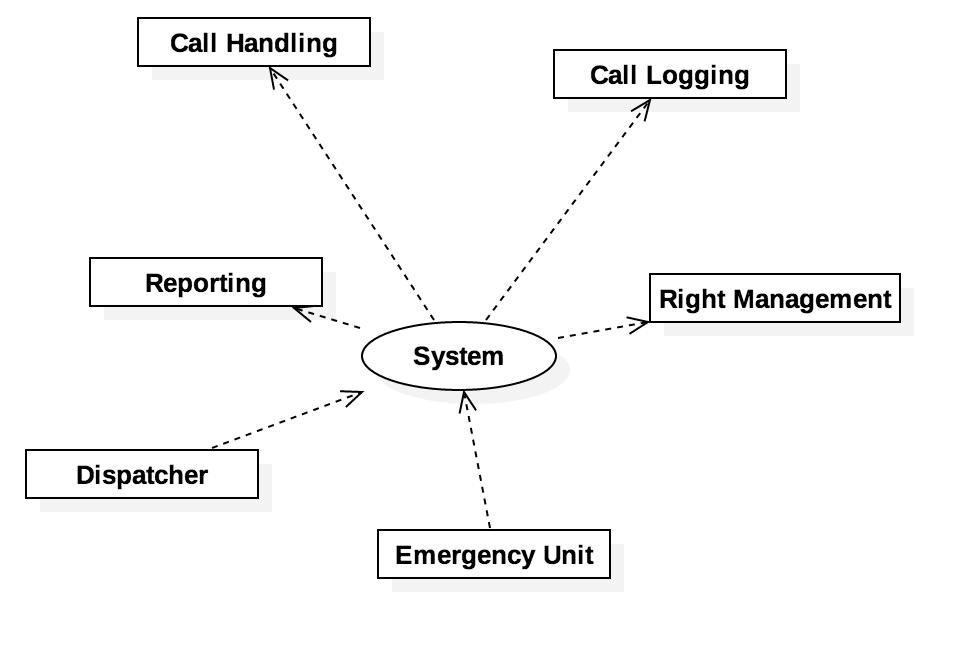
### Announce Call Details



### Make Report



## System Context Diagram



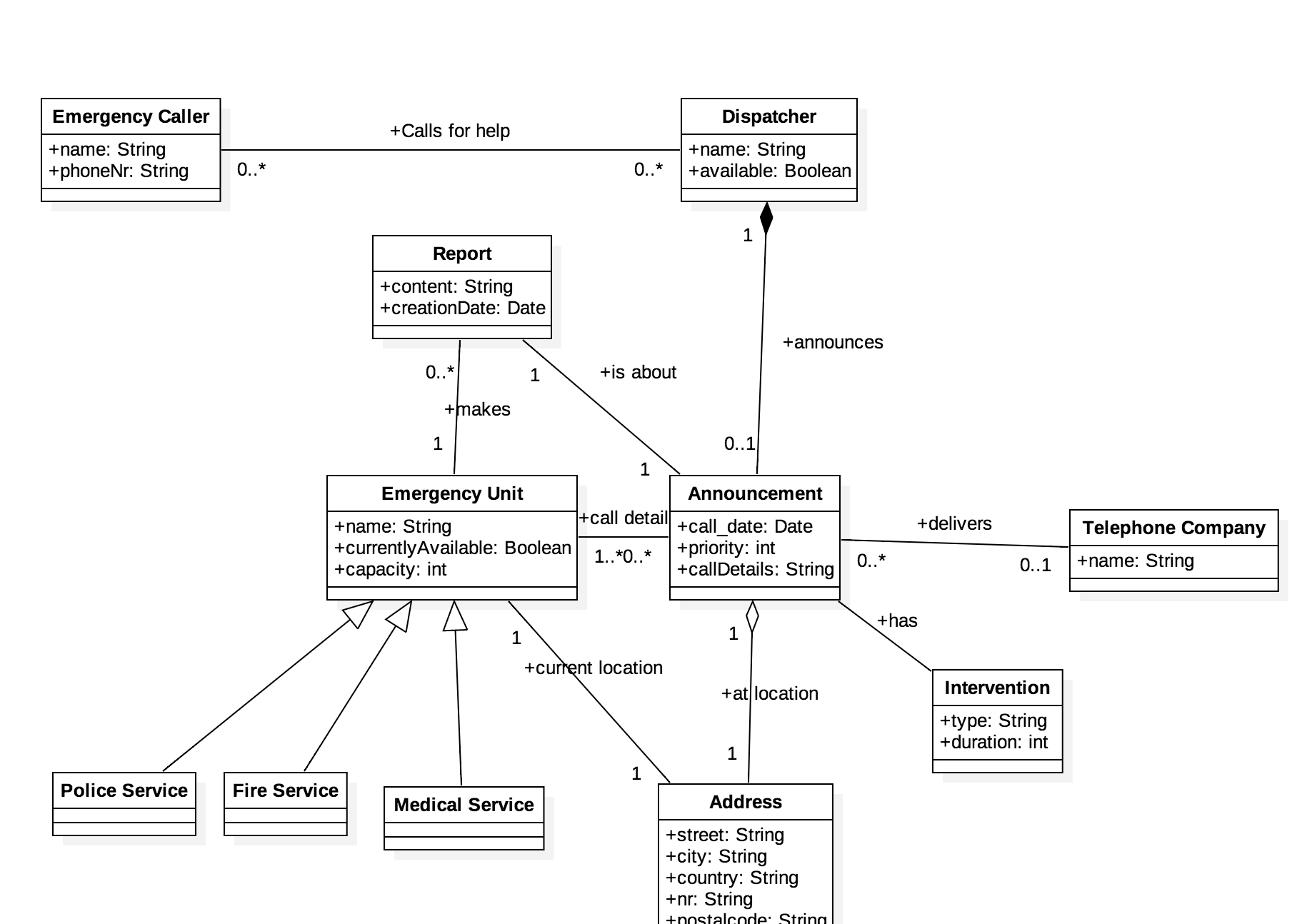
## Domain Model

Domain Concepts

* Emergency Unit
* Police Service
* Medical Service
* Fire Service
* Emergency Caller
* Telephone Company
* Dispatcher
* Report
* Call Detail
* Help
* ~~System~~
* Call
* Announcement
* Help

Conceptual Connections

* Emergency Unit – System
* Emergency Unit – Announce
* Emergency Unit – Report
* Dispatcher – Call
* Dispatcher – Announce
* Dispatcher – Call Details
* Telephone Company – Call
* Call Details – Emergency Unit



TODO: remove strong asso at dispatcher

# Requirements Analysis

## Non-Functional Requirements

* Availability/Reliability: Because emergency calls are crucial a high availability is required. A Availability of 99.999% is expected.  
  The MTBF (Mean Time Between Failures) for severe failures should exceed one year.  
  The MTTR (Mean Time To Recover) should never exceed one day.
* Security: Measures against attacks like SQL Injections and DDoS should be applied and encryption is needed.
* Portability: Mac OS X should be the target machine.
* Fault tolerance: Try to make the system as modular as possible and define clear interfaces.
* Usability: The Dispatchers should have a fast workflow, they will get a detailed introduction (1 hour with high school degree) to the system.
* Backup: Reports and operation data are important and should be backupped regularly. A weekly backup is required.
* Performance: Quick intervention is required so reponse time is very important. No user interaction should have a delay more than 250ms, with the average being 100ms (except logging in, this may take longer). And the time between detail announcements to the emergency unit should never exceed more than five seconds.
* Documentation: User documentation is important. System documentation should be available to a degree that any IT professional is able to maintain the software.
  + Architecture Diagram
  + Software Use Case Diagram
  + Software Class diagram
  + Code Comments and JavaDoc where applicable

Every other non functional requirement does not need an extra consideration and should be implemented what suits best the developer.

## Use Case Diagram

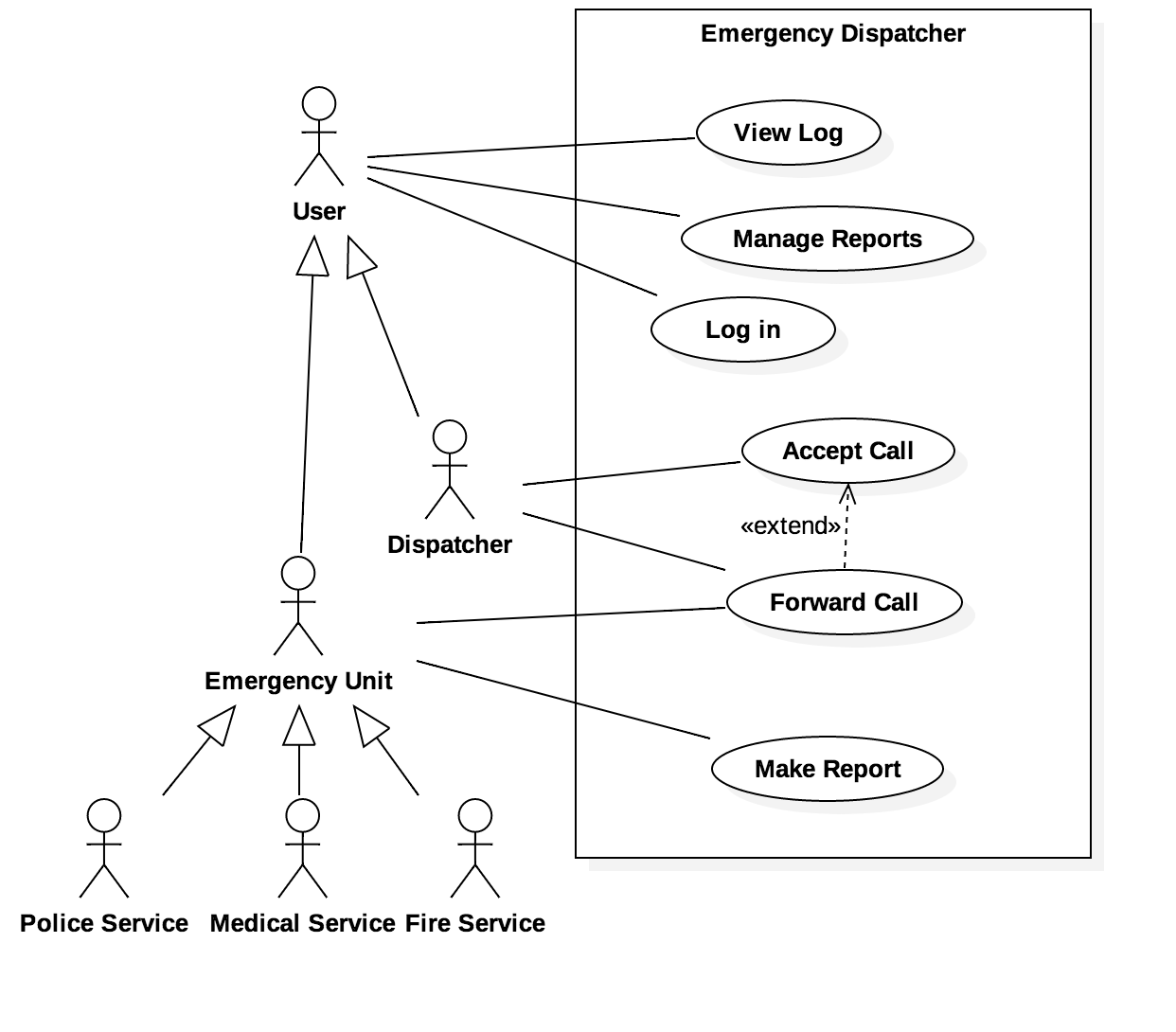
### Events

* User calls => Accept Call
* Data is entered => Forward Call
* Operation finished => Make Report
* Log needs to be checked => View Log
* Reports need to be managed => Manage Log
* Logging in is required => Log In

### Actors

* User: A general user, capable of doing basic things
* Dispatcher: A user who takes calls
* Emergency Unit (Police, Medical, Fire): An unit which executes calls and comes for help

### Software Use Case Diagram



## Use Case Description + System Operation Contracts

### Accept Call

* Precondition: A call is on the way; The Dispatcher is logged in
* Postcondition: The Call has been logged; The emergency unit is notified

|  |  |  |
| --- | --- | --- |
| Function | Description | Importance |
| F1. | User calls the system and the system shows an alert | 10 |
| F2. | Dispatcher accepts call | 10 |
| F3 | Dispatcher asks for data (where, who, what happened) and puts them into the system | 10 |
| F4 | The system validates input | 4 |
| F5 | The system stores the information | 8 |
| F6 | System forwards call to Emergency Unit | 10 |

## Forward Call

* Precondition: The call has been accepted
* Postcondition: Emergency unit has accepted the call and is executing the operation
* Exceptions: User does not finish operation

|  |  |  |
| --- | --- | --- |
| Function | Description | Importance |
| F1 | The system calls the first available emergency unit appropiate for the emergency | 10 |
| F2 | After max 10 seconds the emergency unit has to answer by pressing the accept button, if not find next available emergency unit | 10 |
| F3 | The system checks if the operation is finished | 10 |

### Log in

* Precondition: User is NOT logged in
* Postcondition: User IS logged in if data is right
* Exceptions: Invalid username/password

|  |  |  |
| --- | --- | --- |
| Function | Description | Importance |
| F1 | User enters username and password and submits | 10 |
| F2 | System checks if username exists. Password is being encrypted | 10 |
| F3 | Show available operations: Dispatcher: Accept Call, View Logs; For Emergency Unit: Forward Call; Make Report | 10 |

## Manage Reports

* Precondition: User is logged in and has appropriate (Dispatcher can manage all reports Emergency Unit only their own)
* Postcondition: User can modify and view reports

|  |  |  |
| --- | --- | --- |
| Function | Description | Importance |
| F0 | Check if logged in user has appropriate rights | 8 |
| F1 | Fetch Reports for user | 10 |
| F2 | Display table of available reports | 10 |
| F3 | User clicks by choosing report | 10 |
| F4 | System displays report content | 10 |
| F5 | User may change report and system updates last modification date | 7 |

## Make Report

* Precondition: Emergency Unit finishes operation; Emergency Unit is logged in
* Postcondition: A proper report is in the database

|  |  |  |
| --- | --- | --- |
| Function | Description | Importance |
| F1 | Check if logged in and user has appropiate rights | 8 |
| F2 | Find open operation that the unit needs to write a report for | 10 |
| F3 | User types report into system | 10 |
| F4 | System saves additional data (creation date, last modification date) | 7 |
| F5 | System validates data | 5 |
| F6 | System saves data and marks operation as being reported | 8 |

## View Log

* Precondition: User is logged in and has appropriate rights (Dispatcher can view all logs; Emergency unit only their own)  
  Postcondition: User gets information about logs

|  |  |  |
| --- | --- | --- |
| Function | Description | Importance |
| F1 | Check if logged in and user has appropriate rights: Dispatcher can view all logs, Emergency Unit can only view their logs | 8 |
| F2 | Fetch data from database | 10 |
| F3 | Display data in a graphical user interface | 10 |

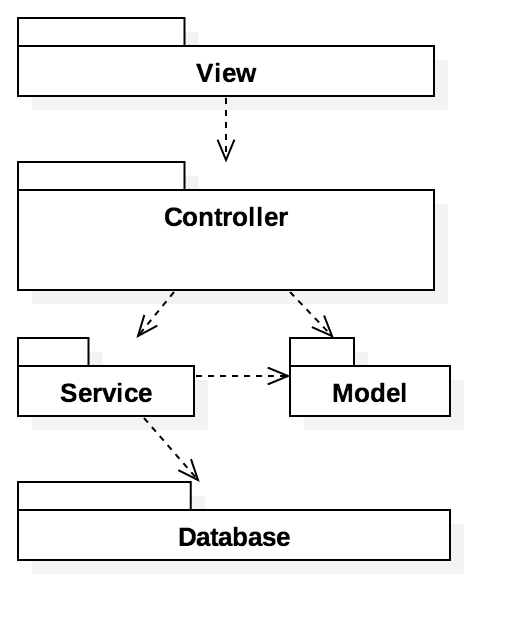
TODO: Look at guidelines for proper layout

TODO: Operation Contract for single Operation (getListOfAvailableOperations e.g.)

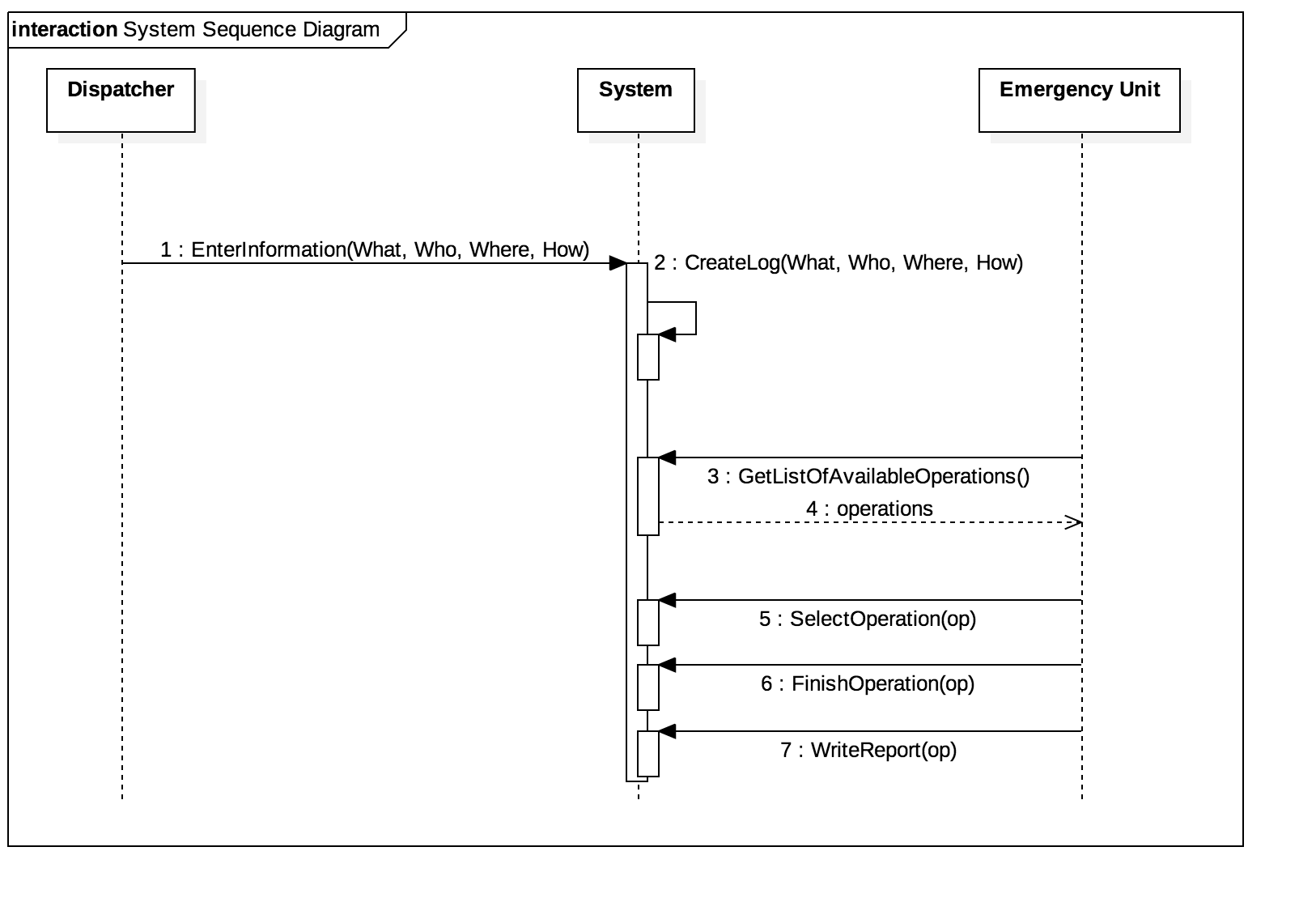
### Textual Use Case Descriptions

* View Log:
  + Any User may View the log. The log is the history of calls made by people. It is shown in a table, when the call was made, which telephone number and the corresponding Report (if it exists) or if the duty is still going on.
  + The log may not be changed by any user, only by the system if a new call is registered.
* Manage Reports:
  + Any User may manage his/her own reports (only view and modify) additionally any Dispatcher can view all reports of every user.
  + Manage Reports displays all available reports and by clicking on a report the system fetches the corresponding report and displays details, content, date, creator and last modification
* Log In:
  + Any User registered can log in with a username and password. There is no “Forgot password”. Based on the type of user the available actions will be displayed. The usernames are stored in cleartext in the database and the password are SHA1 encrypted.
* Accept Call:
  + Any Dispatcher may accept a call, but only one at a time.
  + When an Emergency happens the “Accept Call” function is available. By pressing the button the Dispatcher gets a Step-by-step guide of questions he has to ask and type into the system: Where is it? What happened? Who is involved? How critical? Who should come?
  + After the appropriate information are entered and it is really an emergency the call my be forwarded.
* Forward Call:
  + The system checks the type of the emergency and forwards the call to the emergency unit. The Emergency Units get an alert with all the necessary information and they accept or deny. If an Emergency Unit accepts the emergency is in progress, if not the next available emergency unit is asked.
* Make Report:
  + After an emergency all emergency units that were involved have write a report. The content, creator and date is saved into the database.
  + Reports may only contain text and no additional formatting.

## Architecture Draft Document Package Diagram



## System Sequence Diagram



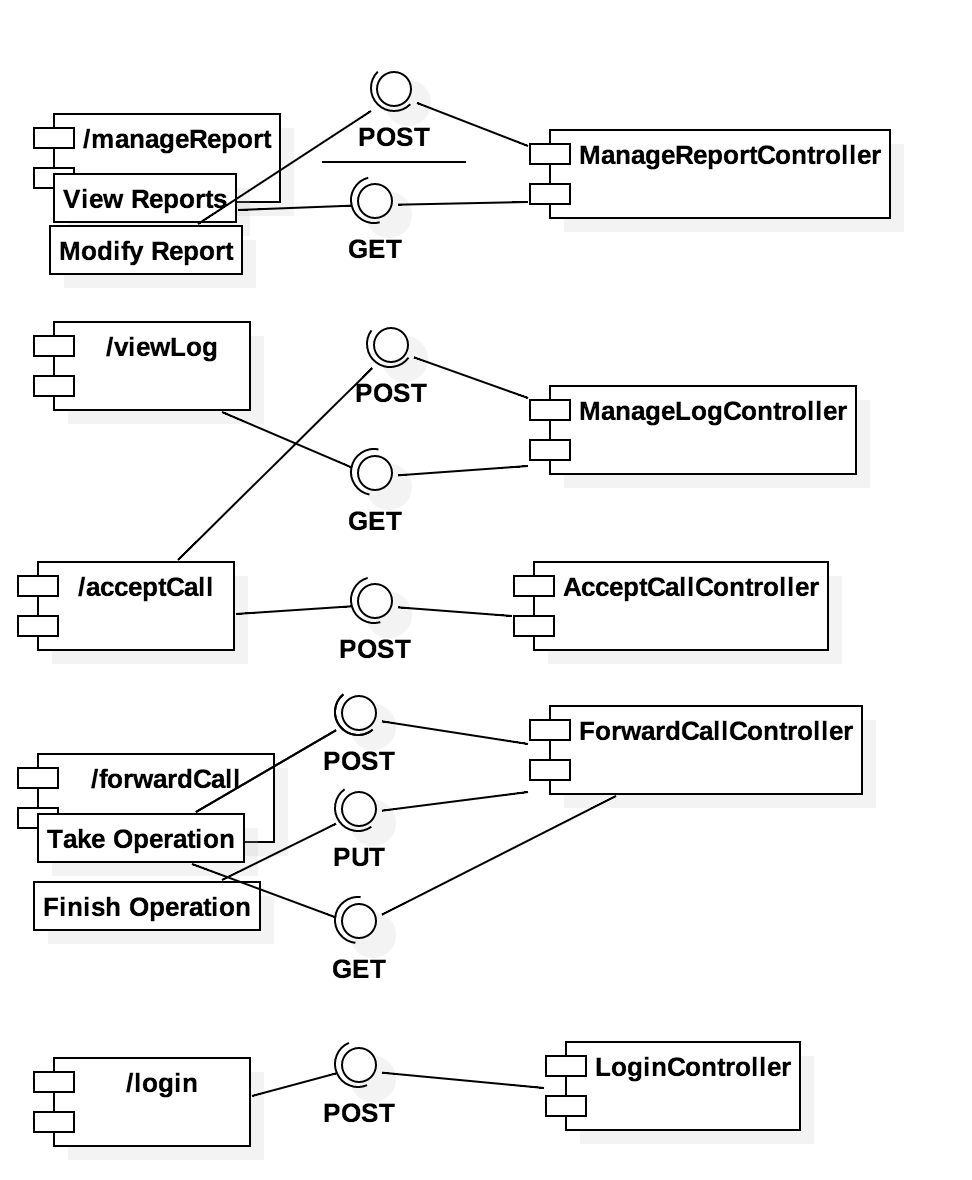
TODO: 1 per use case

# Design Model

## State Diagram for Operation objectInternet Explorer:Users:rfischer:Dropbox:Studium_Sem4:SDM:FinalMissingDocs:StateOfOperation.png

* initialized... all necessary attributes are set
* forwarded... Emergency Unit accepted operation (setAssignedUnit(unit))
* operation finished... end date has been set (setEndDate(date)) and empty report assigned setReport(report)
* report submitted... report has been added successfully (report.setReport(text))

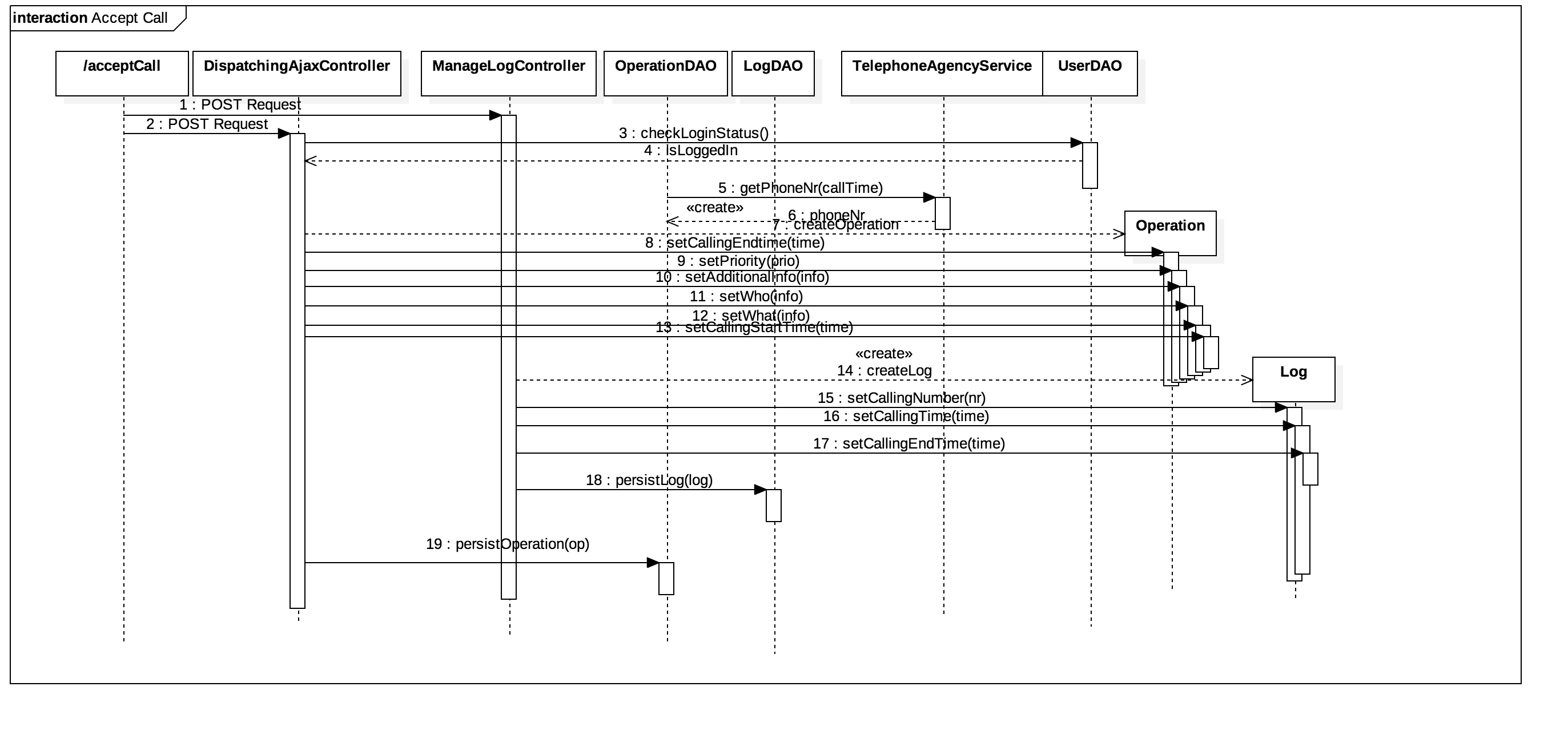
## Component Diagram

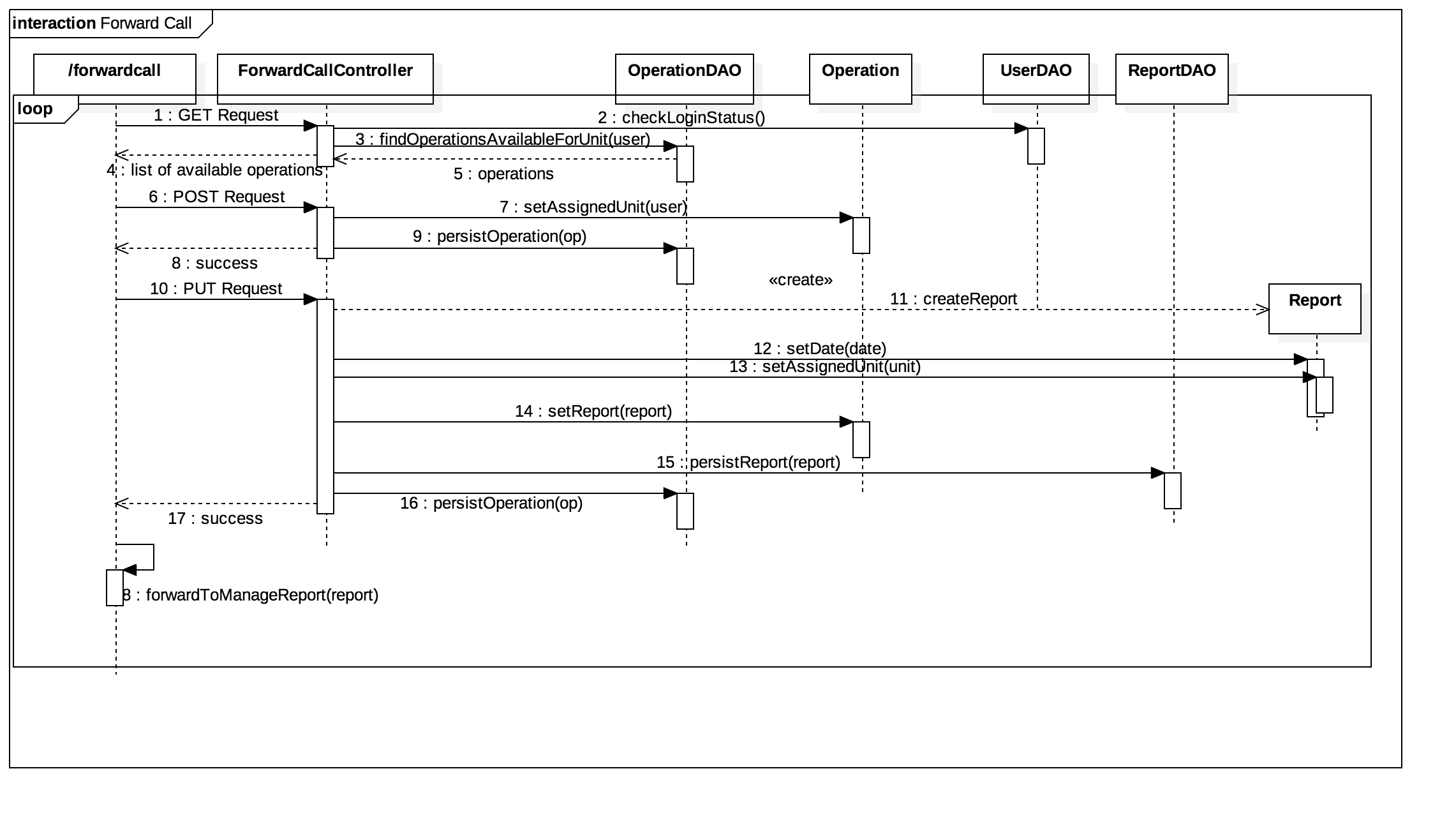


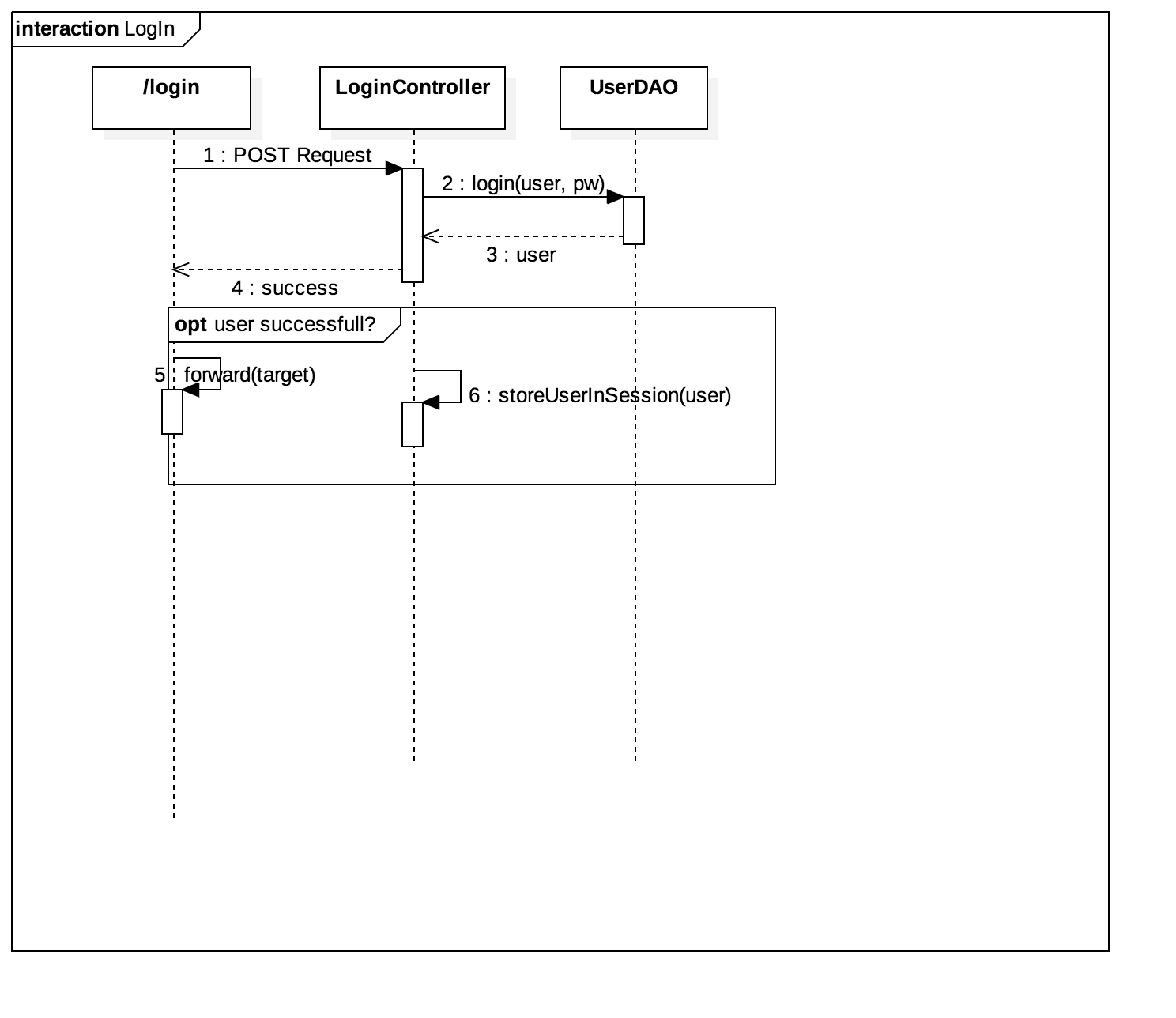
### Responsibilities to classes

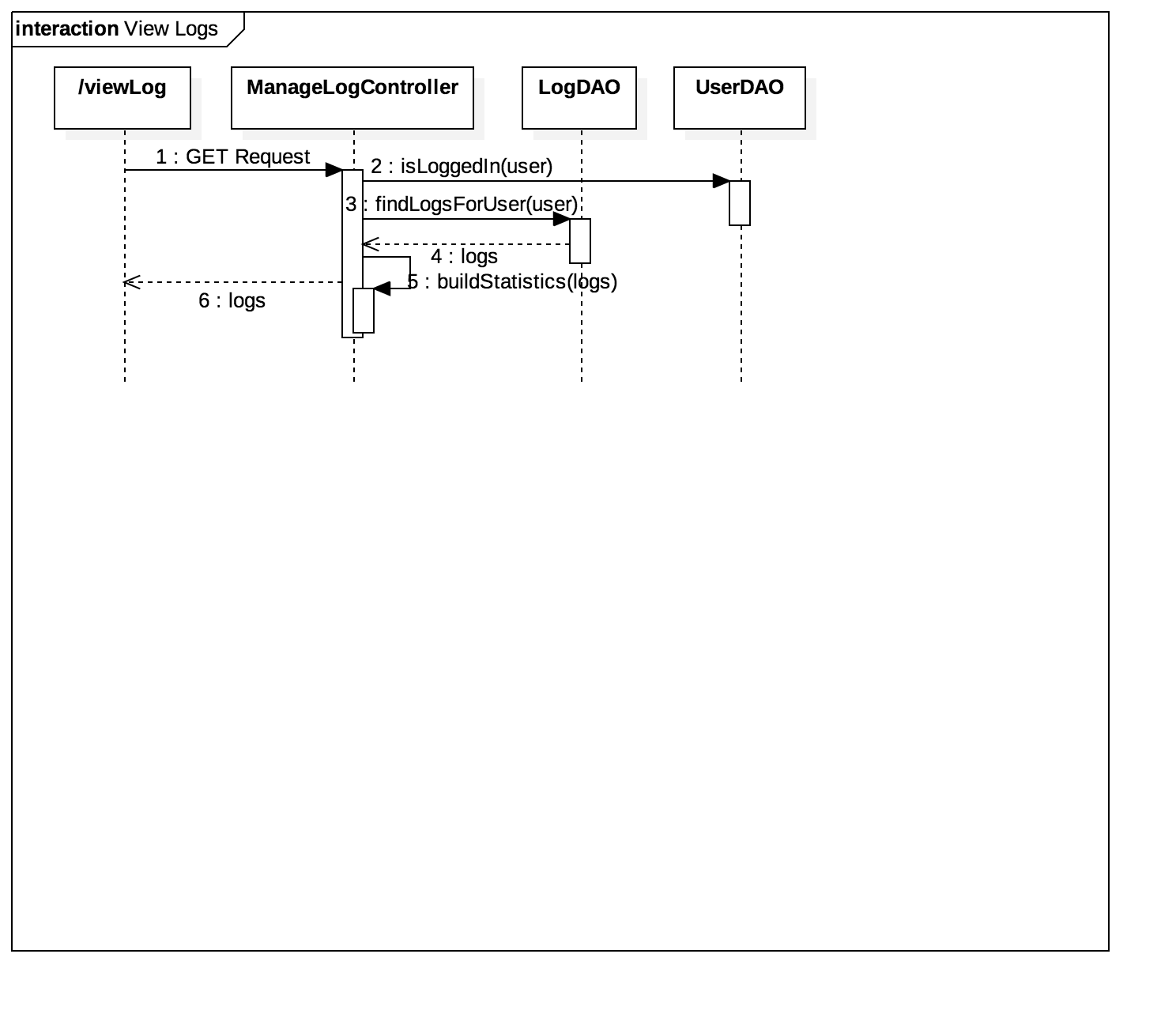
* View Log
  + UserDAO, LogDAO, ManageLogController, Log
* Manage Report
  + UserDAO, ReportDAO, ManageReportController, Report
* Make Report
  + UserDAO, ReportDAO, ManageReportController, Report
* Accept Call
  + UserDAO, DispatcherController, ManageLogController, OperationDAO, LogDAO, TelephoneAgencyService
* Forward Call
  + UserDAO, ForwardCallController, OperationDAO, Operation, ReportDAO, Report
* Login
  + UserDAO, LoginController

## Interaction Model

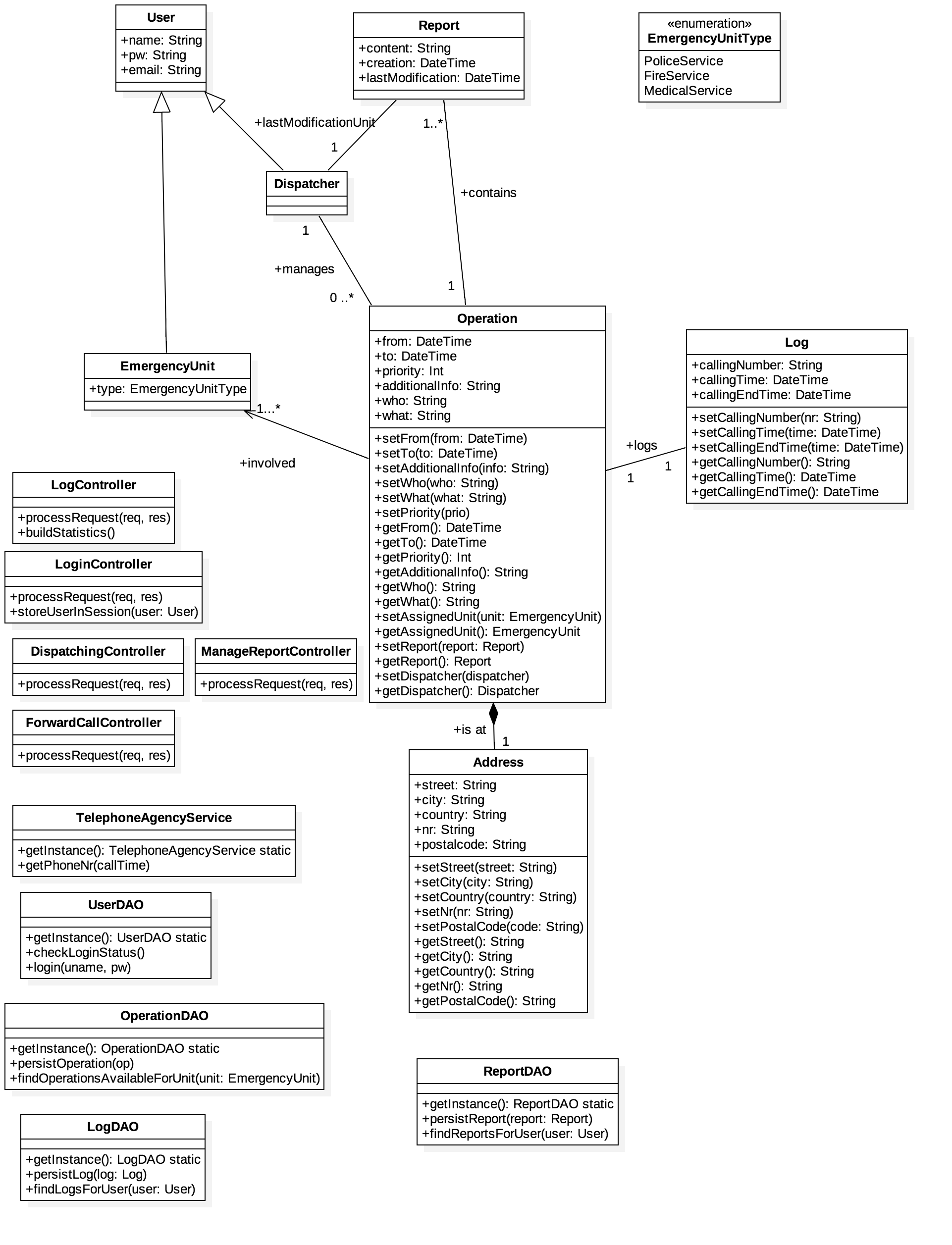
TODO: Dieses Interaction Model updaten



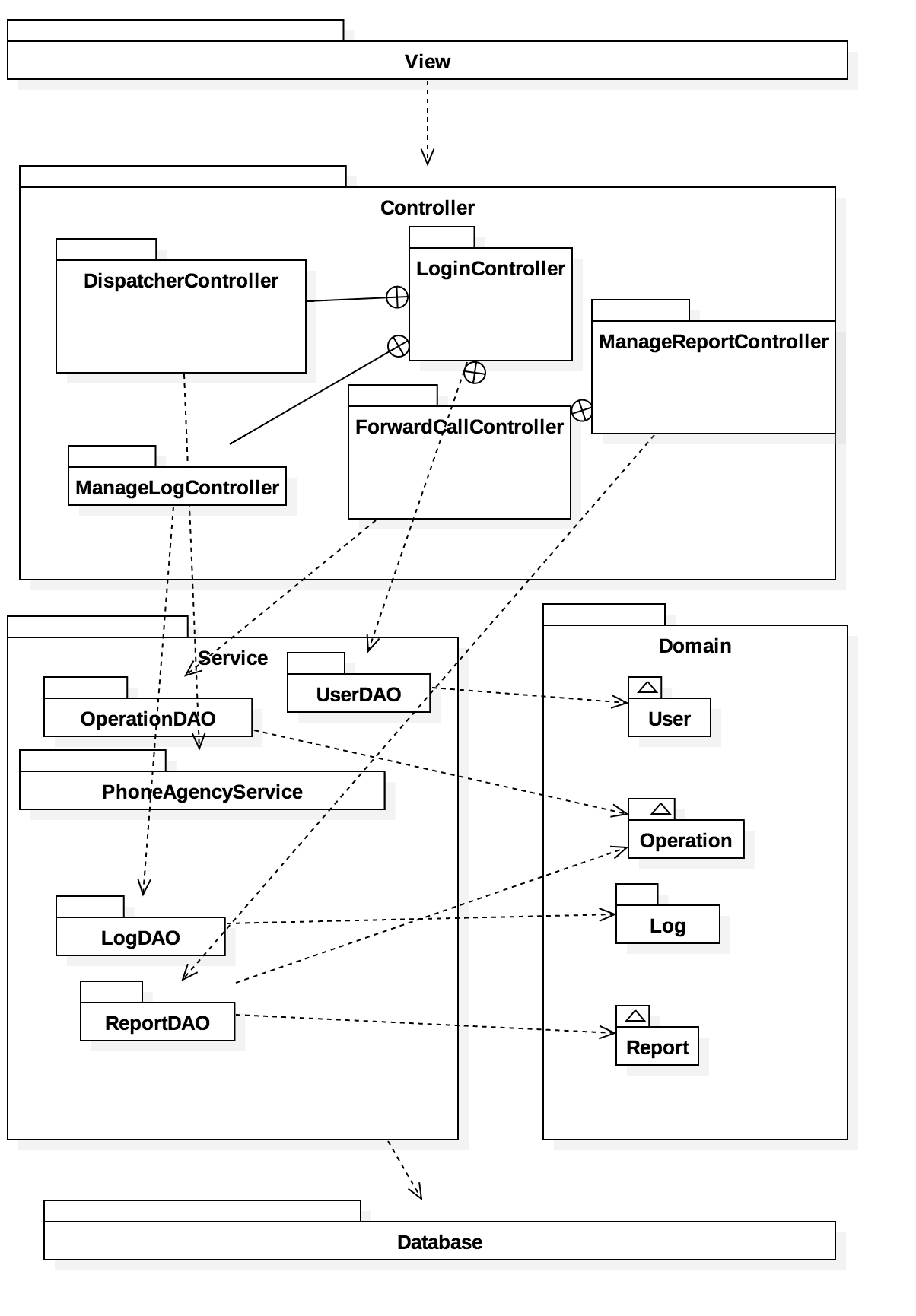




## Design Class Diagram



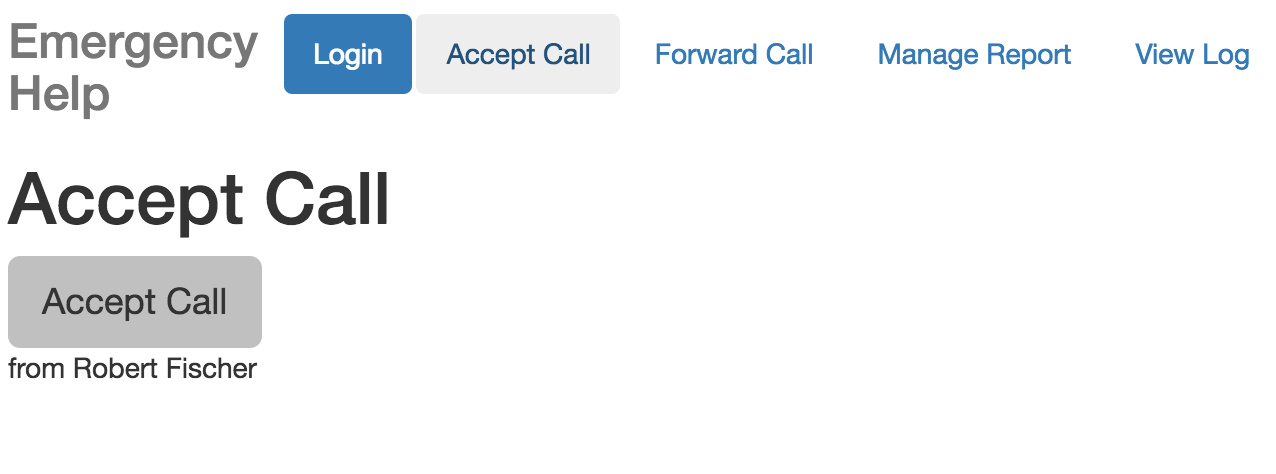
## Architecture Package Diagram



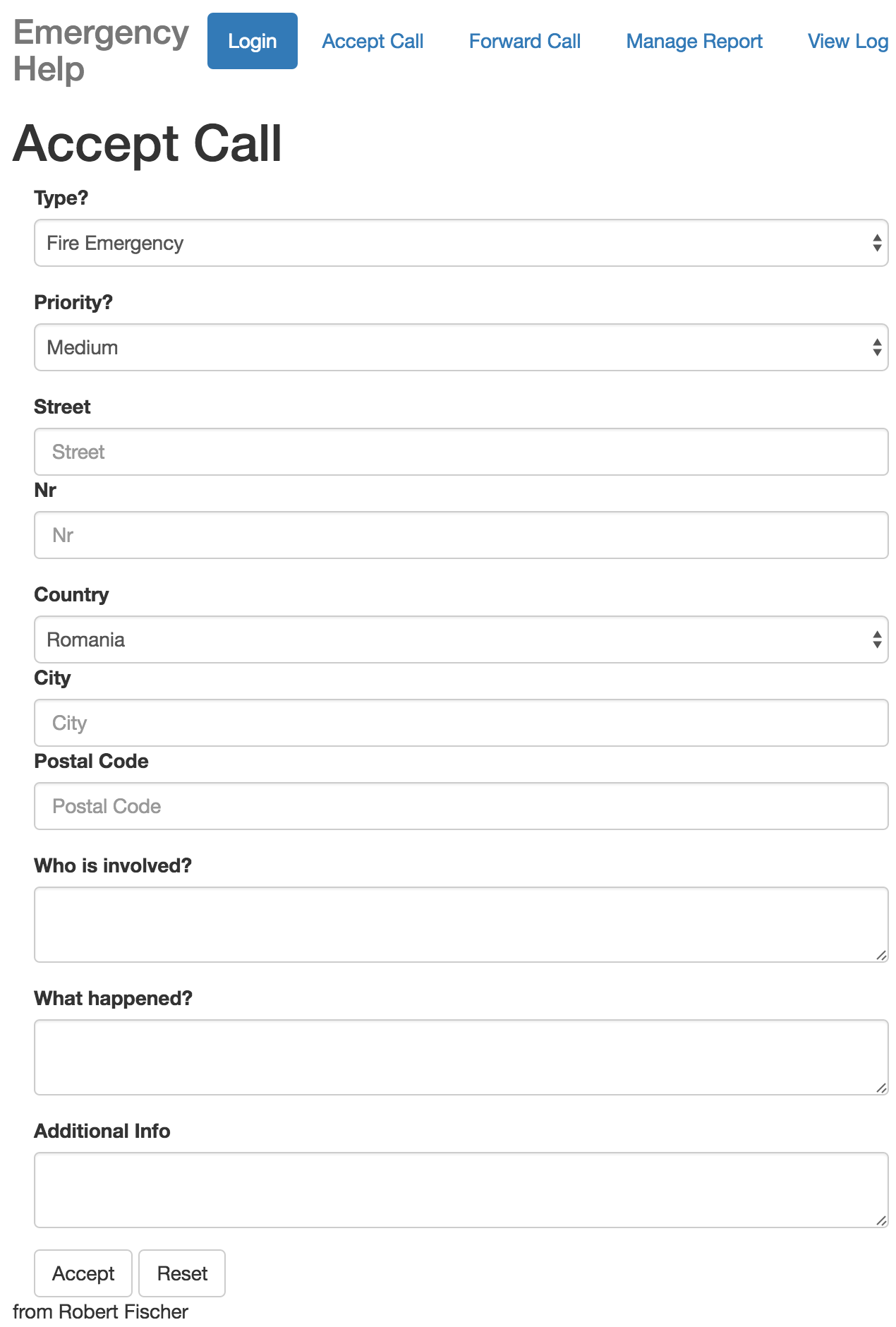
## User Interface Design

### Accept Call

* Wait for call and accept on incoming

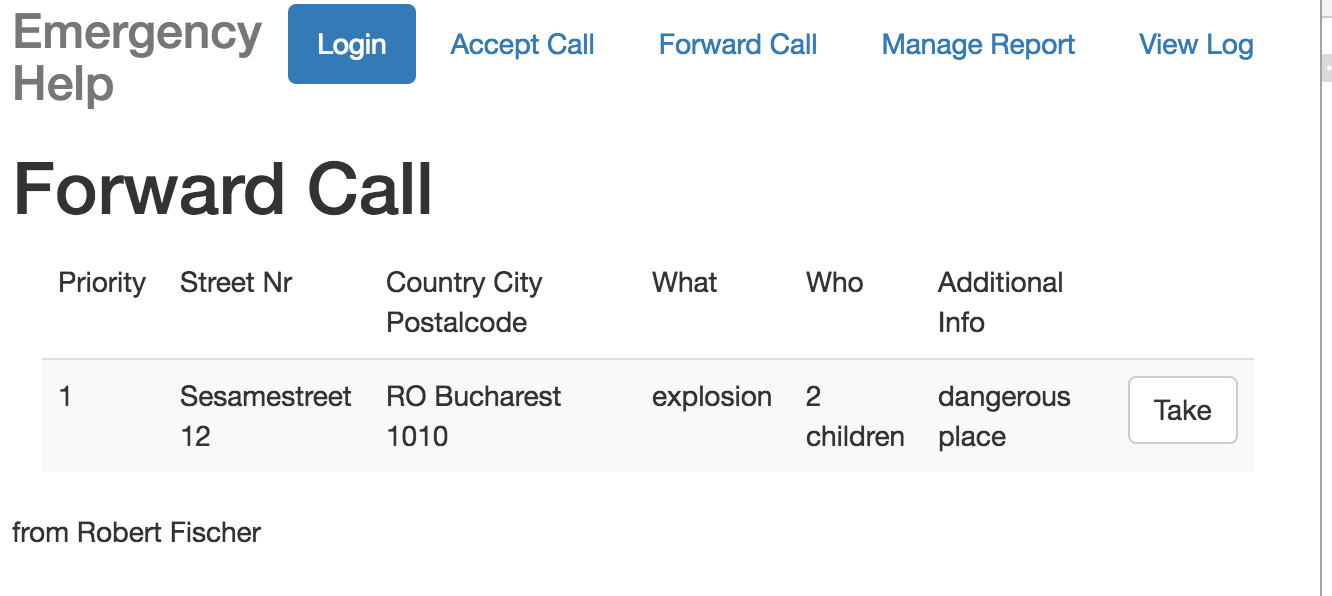


* Enter call information for Emergency Unit

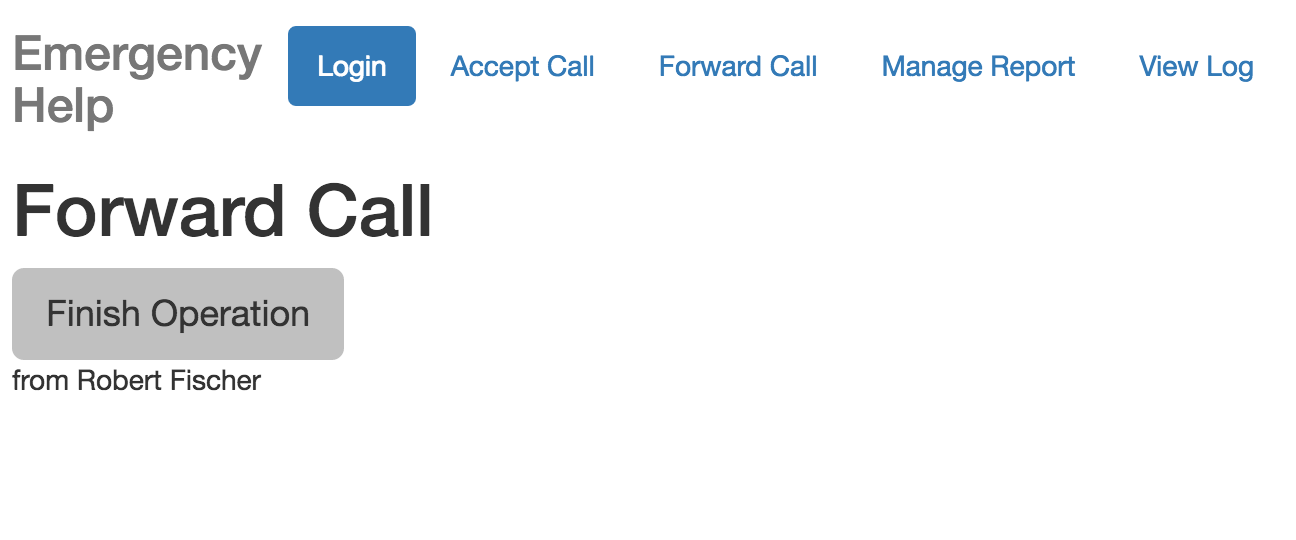


### Forward Call

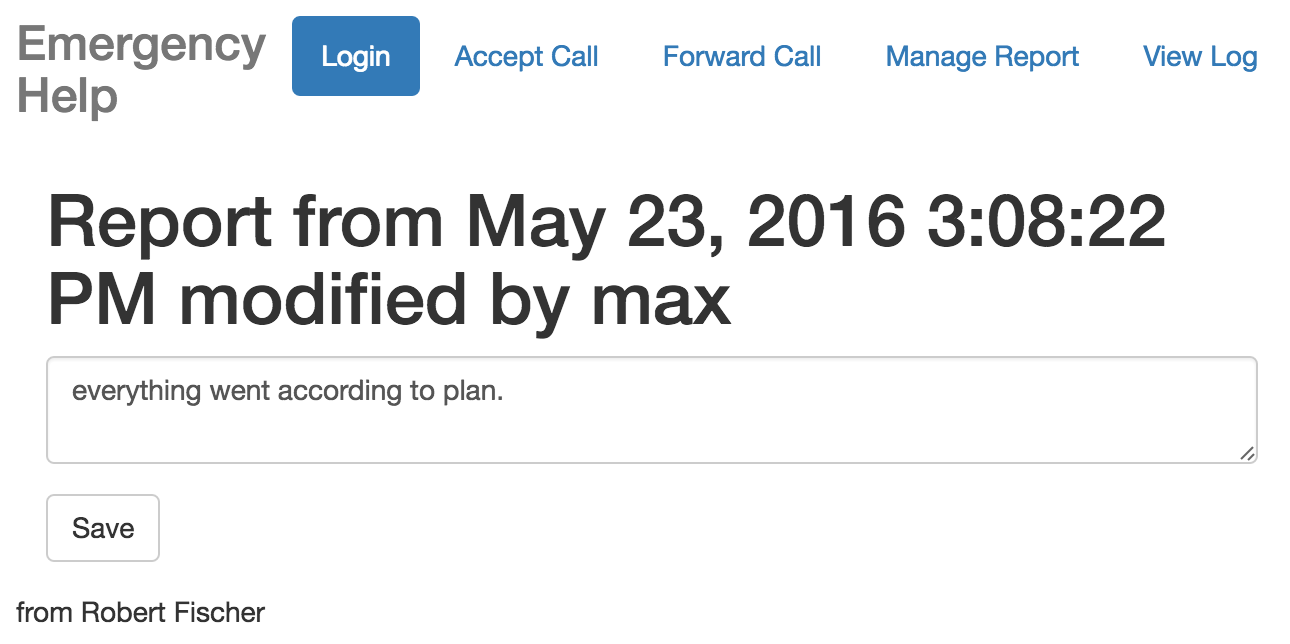
* Ready to forward calls



* Operation in progress

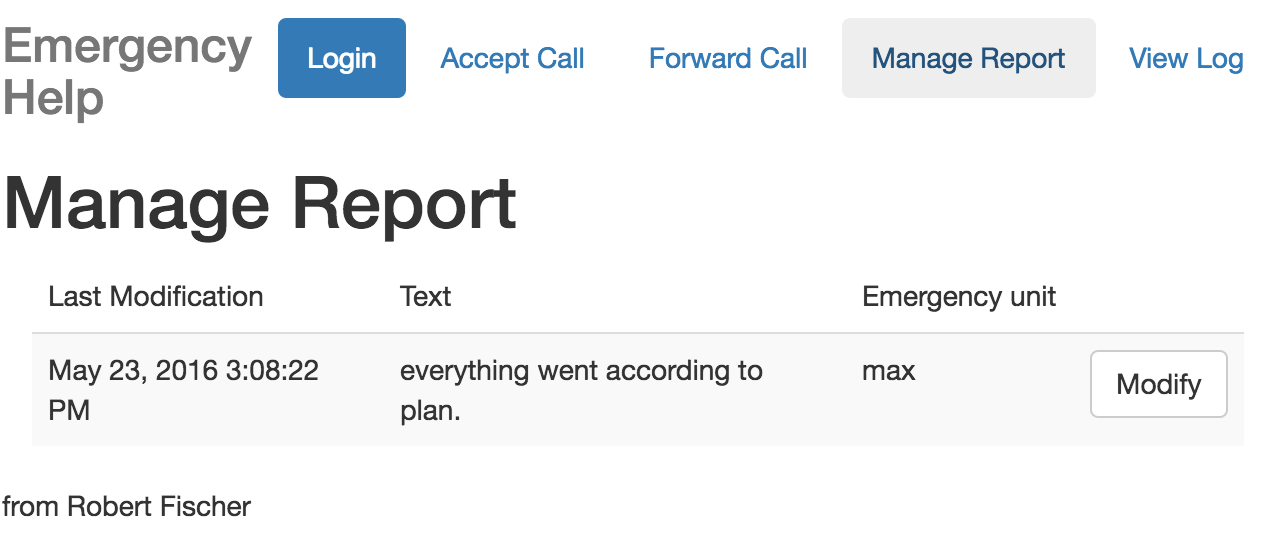


* Write report after operation finished



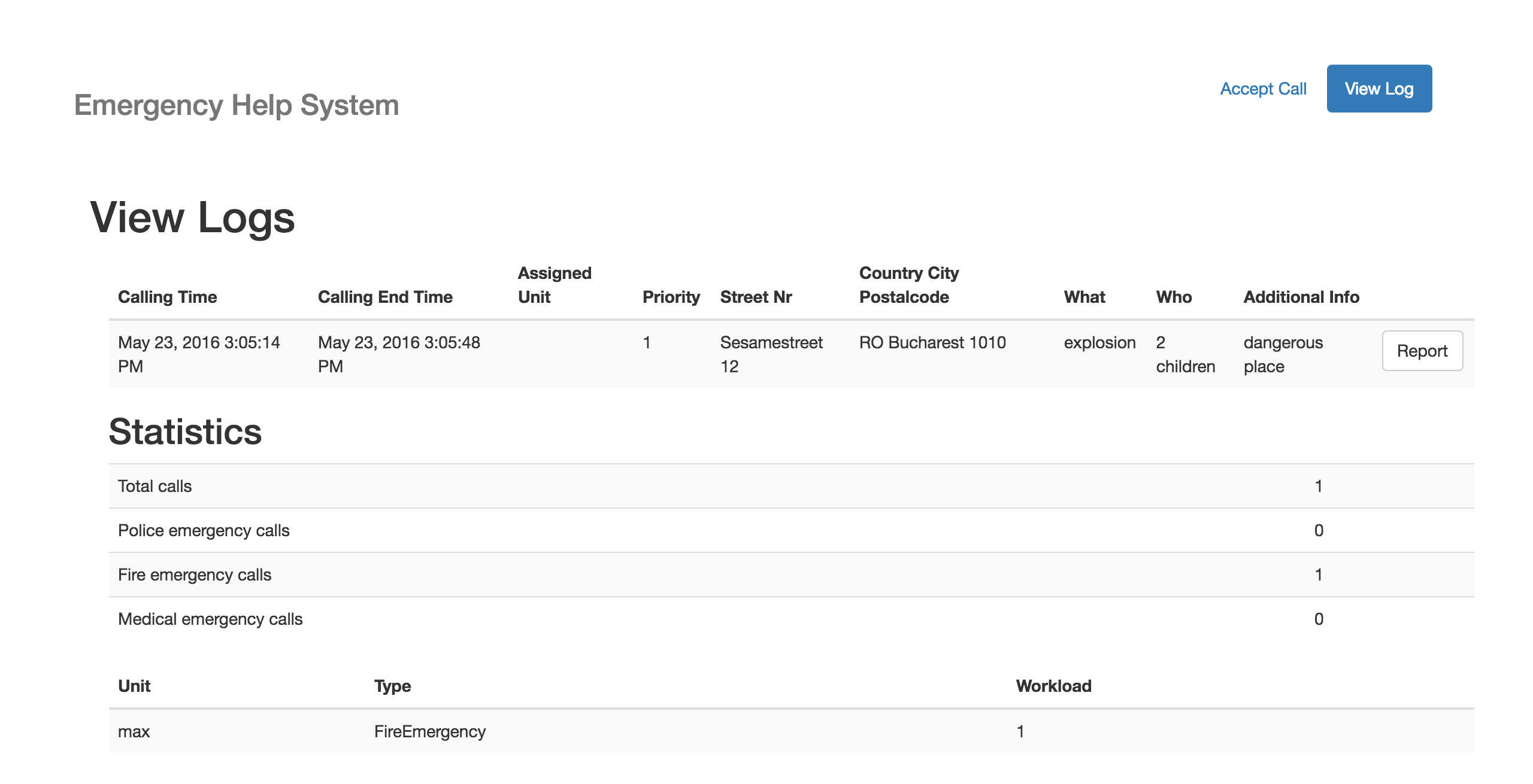
### Manage Report

* List of reports available to the current user



### View Log

* List of logs available to the current user



# Implementation

## Code

<https://github.com/metzzo/sdm_project>

## Installation Manual

No installation required: Just open the webpage of the project. The test installation is reachable via <http://localhost:9000/> and needs the grunt serve command to be successfully run on the command line. The backend has to be also successfully started.

To run the test installation issue the following commands (example on OS X) for the frontend:

npm install -g bower grunt-cli

* Install Dependencies

npm install bower update

* Install Ruby + Compass (if not already installed)

gem install compass

* Run grunt

grunt serve

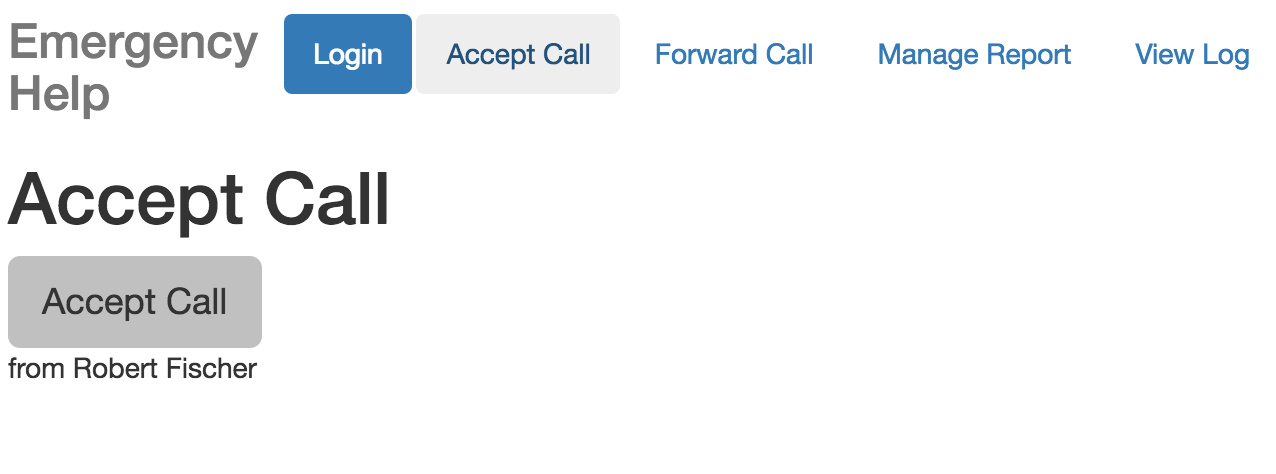
For the backend you need a proper JDBC datasource called “myDatasource” to be set up and a working Glassfish 4.x installation. Additionally there needs to be a accessible MySQL 5.x database.

The test users are:

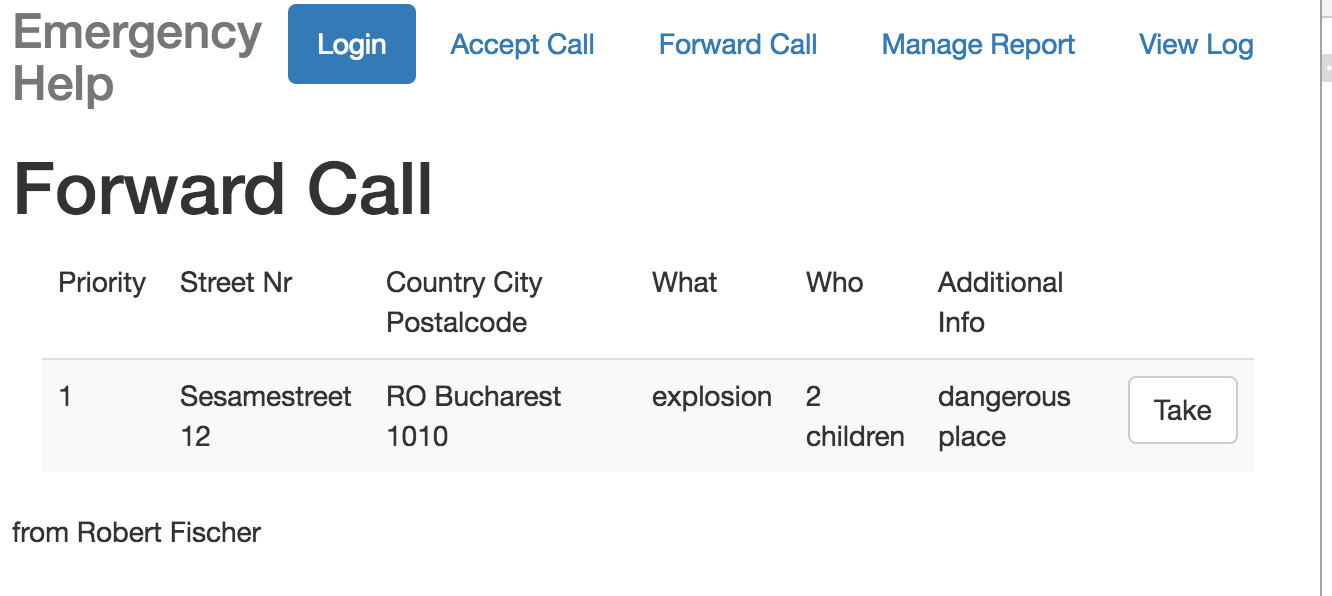
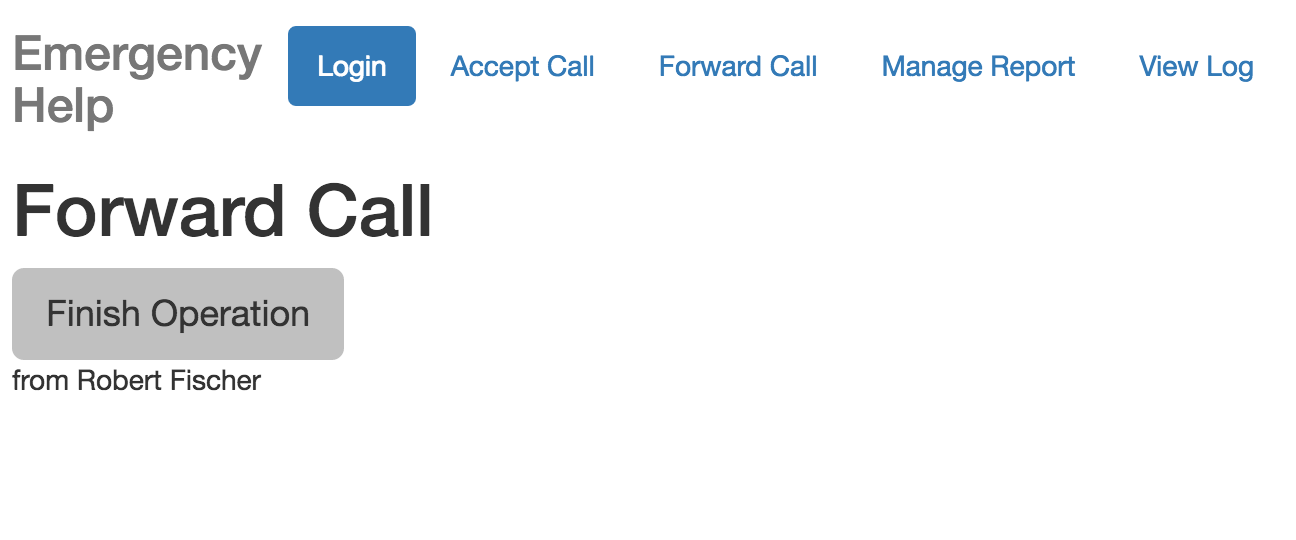
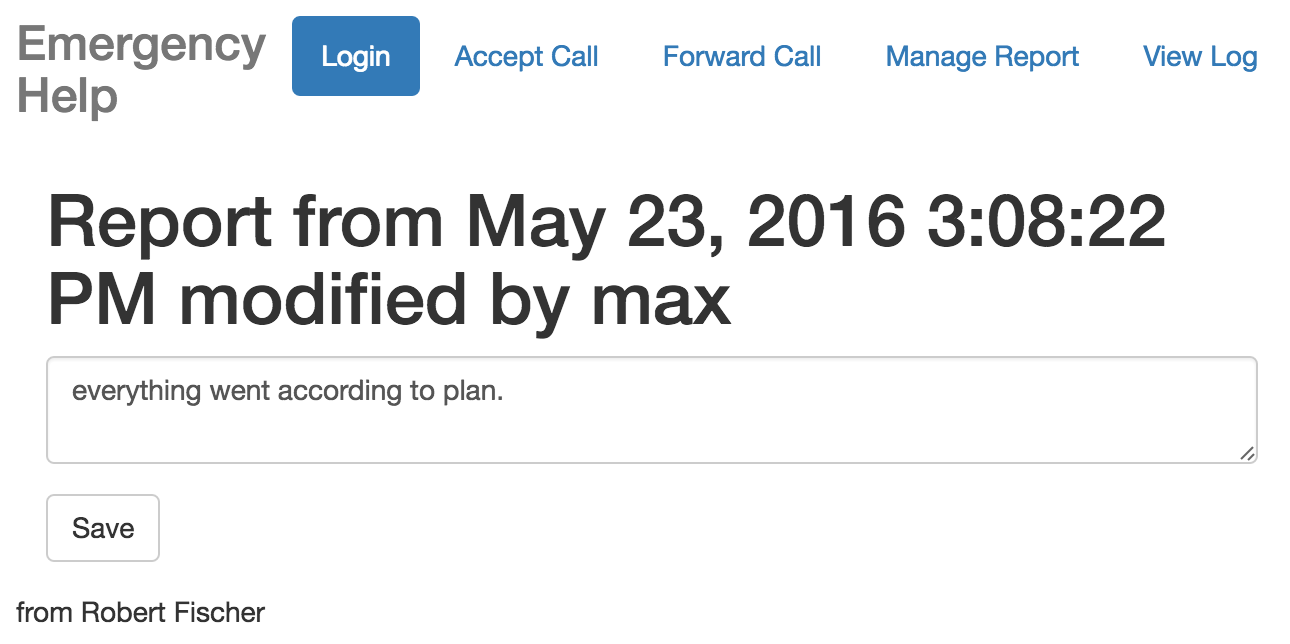
* username: robert password: a … Dispatcher
* username: max password: a … Fire Emergency
* username: alex password: a … Medical Emergency
* username: dorina password: a … Police Emergency

## User Manual

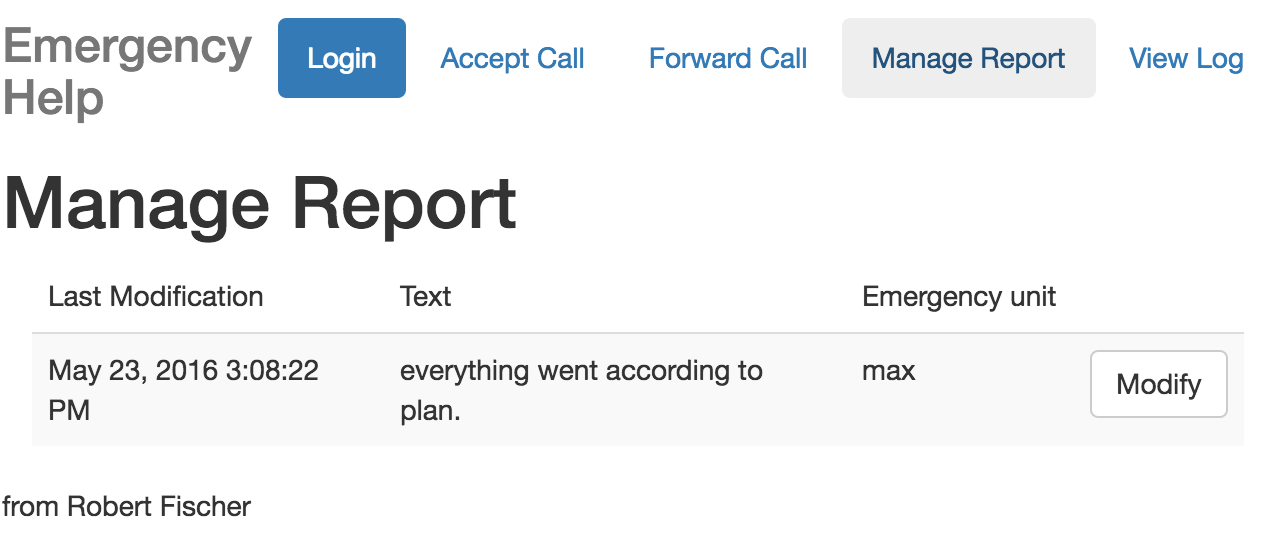
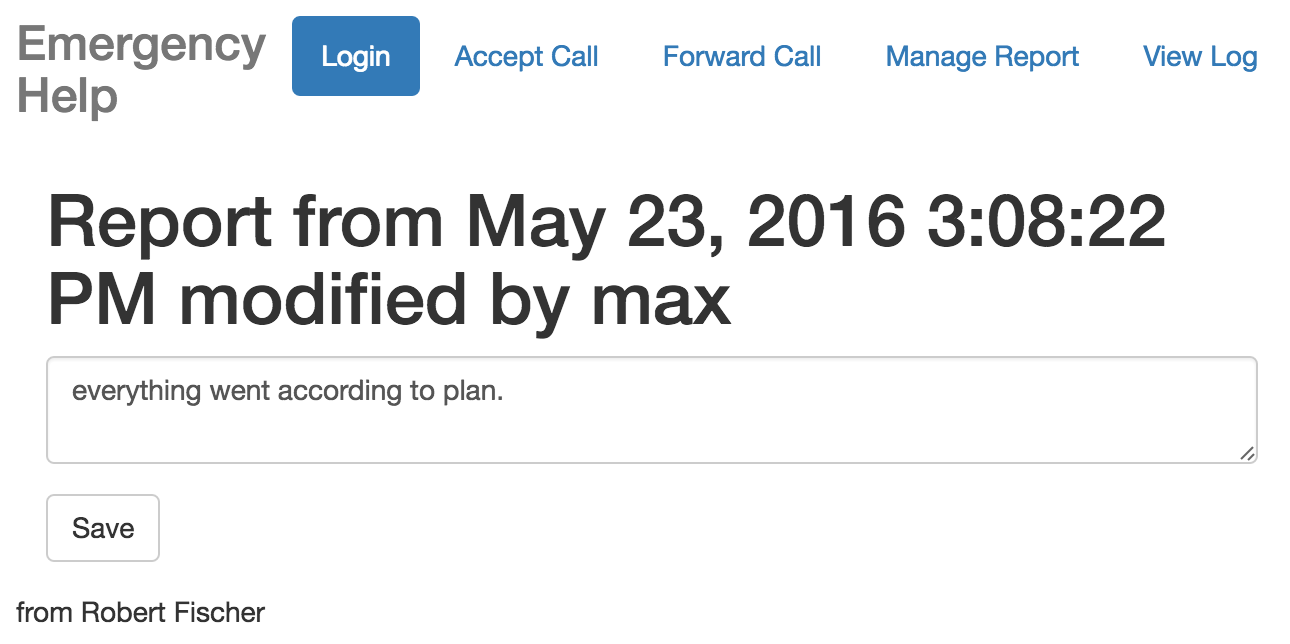
### You are a dispatcher?

1. Open the website
2. Login with your username and credentials
3. If you receive a call click on „Accept Call“:  
   
4. The system will display now input boxes you need to fill out by asking the callee. Please ask accordingly.
5. Click on submit and let a proper emergency unit take the emergency

### You are an emergency Unit?

1. Open the website
2. Login with your username and credentials
3. The system will display a list of available emergencies, take them if you are near them by clicking on „Take“  
   
4. When you take an emergency it wont be available to other units
5. Click on „Finish Operation“ when the operation has finished  
   
6. Write a report about what happened and click on “Save” when ready.  
   

### Manage Reports

* To manage reports click on „Manage Report“ (note: you can only do this if you have appropiate rights)
* You will see a list of reports  
  
* By clicking on „Report“ you can edit the report (if you have appropiate rights)  
  

### View Logs

* To see what operations you’ve been involved or all operations available (if you are a dispatcher) click on „View Logs“
* You can also see information regarding the operations and statistics
* By clicking on „Report“ you can see the report (if available)

