Problem Statement 2

Business Analysis 2

Business Use Case Diagram 3

Business Activity Diagram 4

Call for help 4

Announce Call Details 5

Make Report 5

System Context Diagram 6

Domain Model 6

Requirements Analysis 7

Non-Functional Requirements 7

Use Case Diagram 8

Events 8

Actors 8

Software Use Case Diagram 9

System functions 9

Accept Call 9

Forward Call 9

Log in 10

Manage Reports 10

Make Report 10

View Log 10

Software Use Case Description 10

Accept Call 10

Forward Call 11

Log in 11

Manage Reports 12

Make Report 12

View Log 13

Architecture Draft Document Package Diagram 14

System Sequence Diagrams 14

Accept Call 14

Forward Call 15

Make Report 16

Manage Reports 17

View LogsLogin 18

Operation Contract 19

WriteReport 19

UpdateReport 19

Design Model 20

State Diagram for Operation object 20

Component Diagram 21

Responsibilities to classes 21

Interaction Model 22

Design Class Diagram 25

Architecture Package Diagram 27

User Interface Design 28

Accept Call 28

Forward Call 29

Manage Report 30

View Log 31

Implementation 31

Code 31

Installation Manual 31

User Manual 32

You are a dispatcher? 32

You are an emergency Unit? 33

Manage Reports 34

View Logs 35

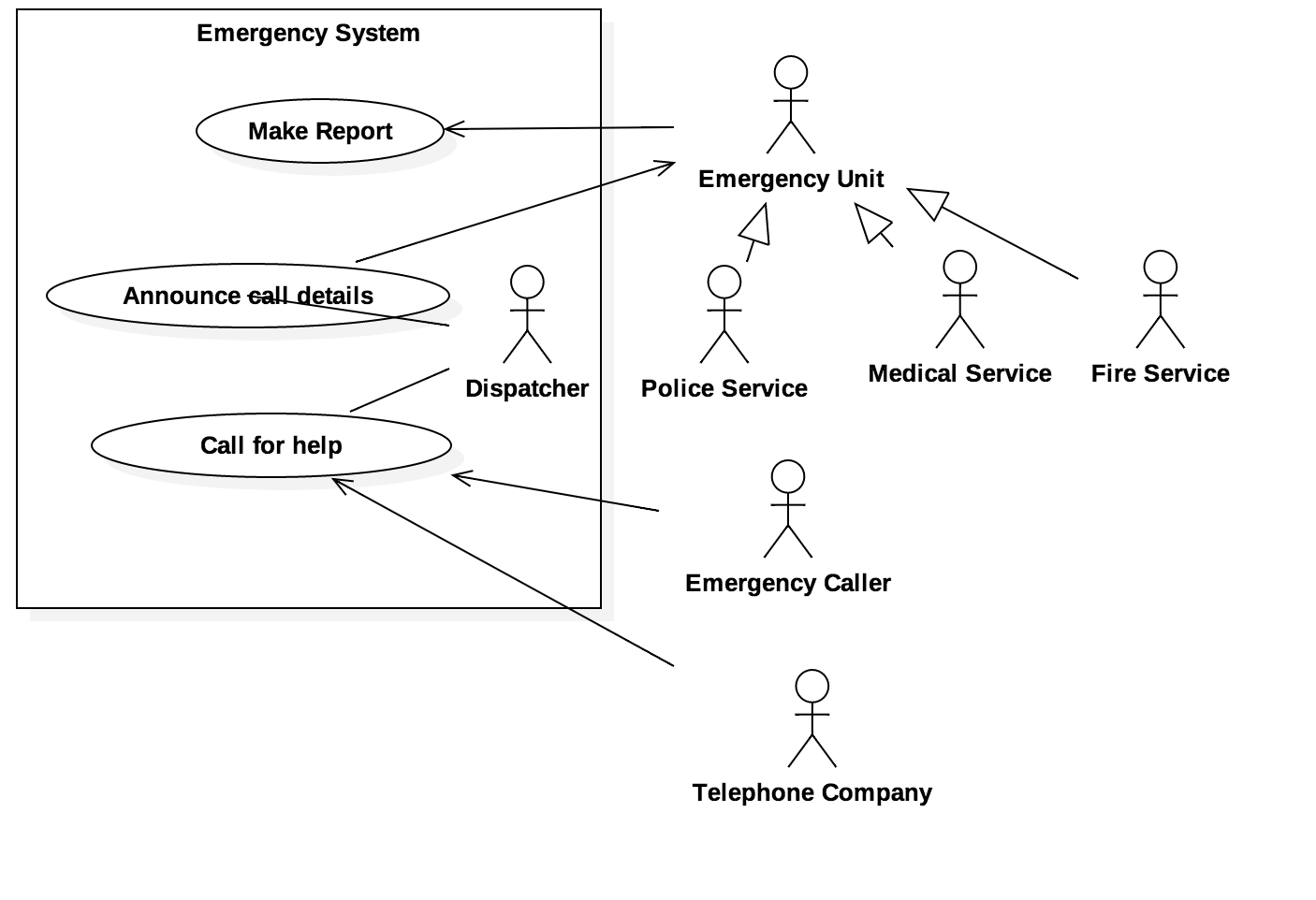
# Problem Statement

„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

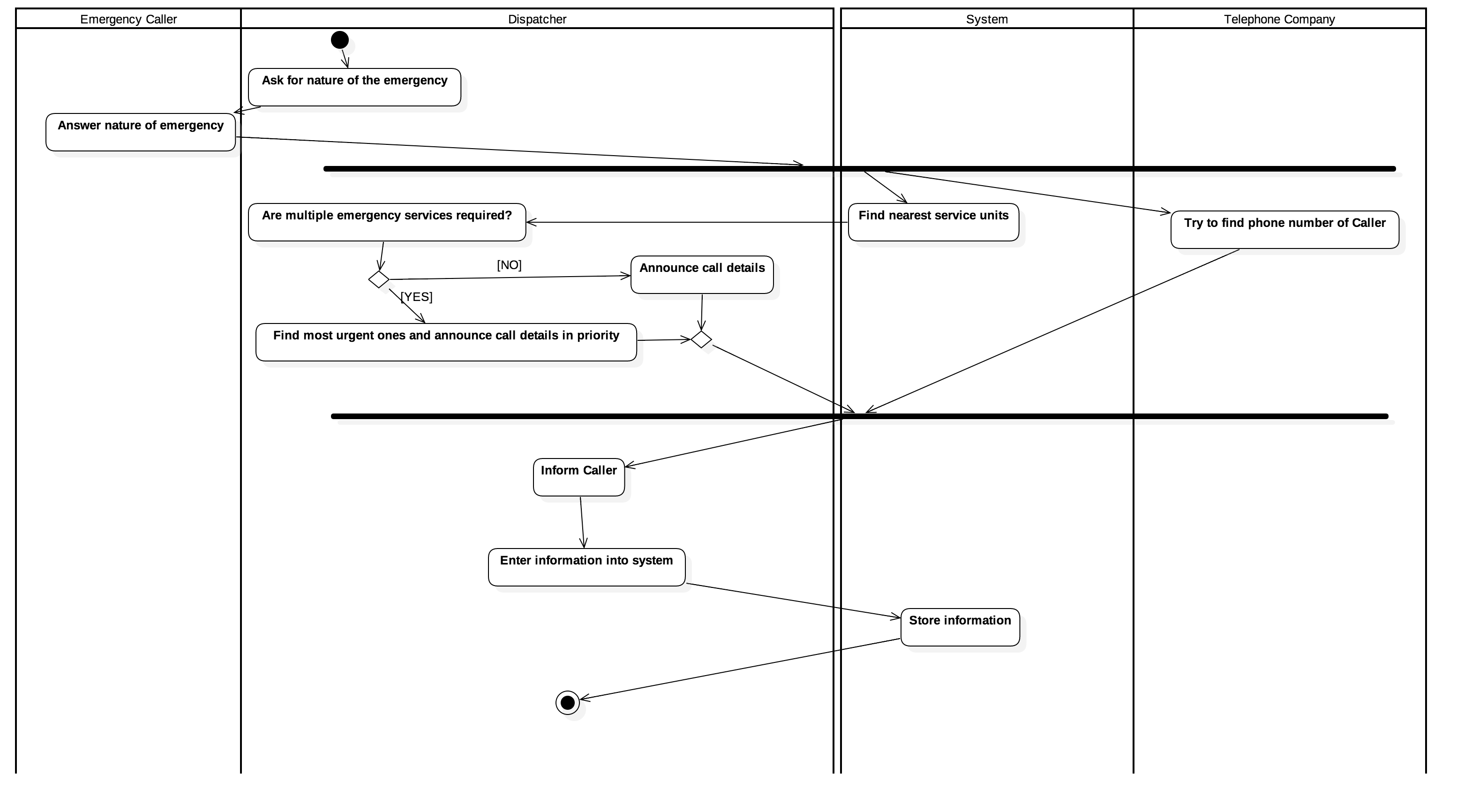
* Business Actors:
  + Emergency Unit
    - Police Service
    - Medical Service
    - Fire Service
  + Emergency Caller
  + Telephone Company
* Business Process:
  + „Manage good inputs and outputs in an emergency dispatching center“
* Business Agents:
  + Dispatcher
  + Emergency unit
* Automated Activities
  + 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

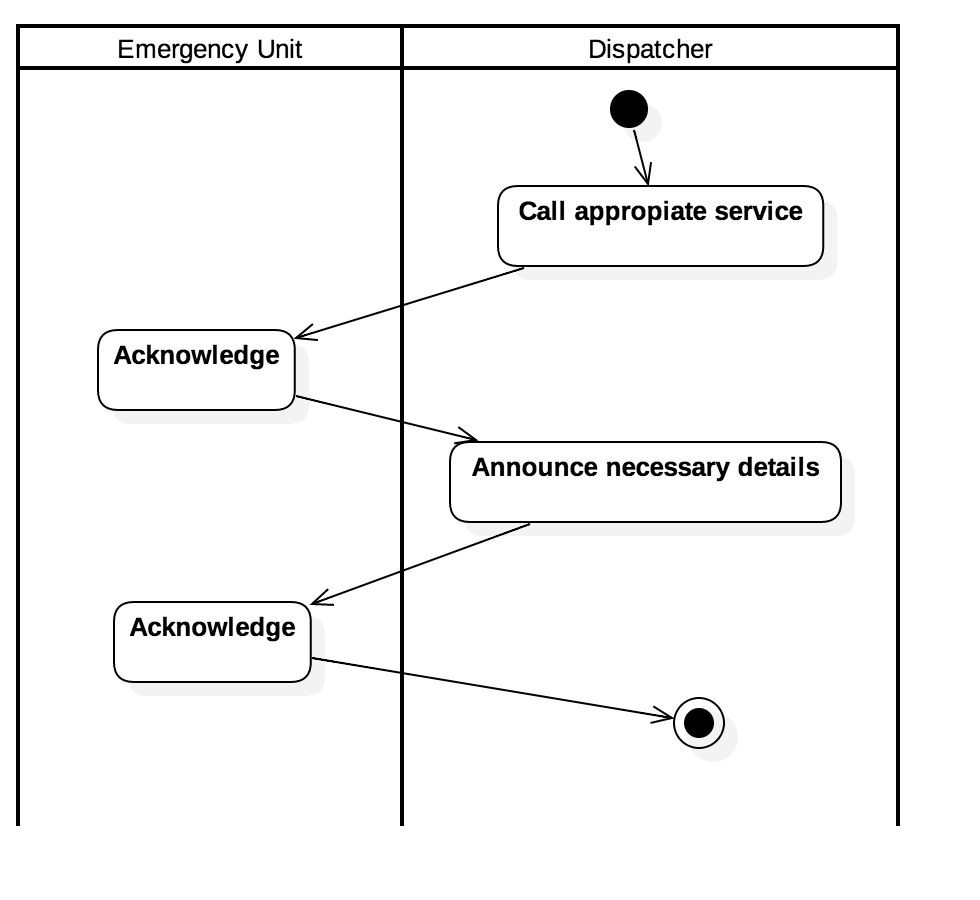


## Business Activity Diagram

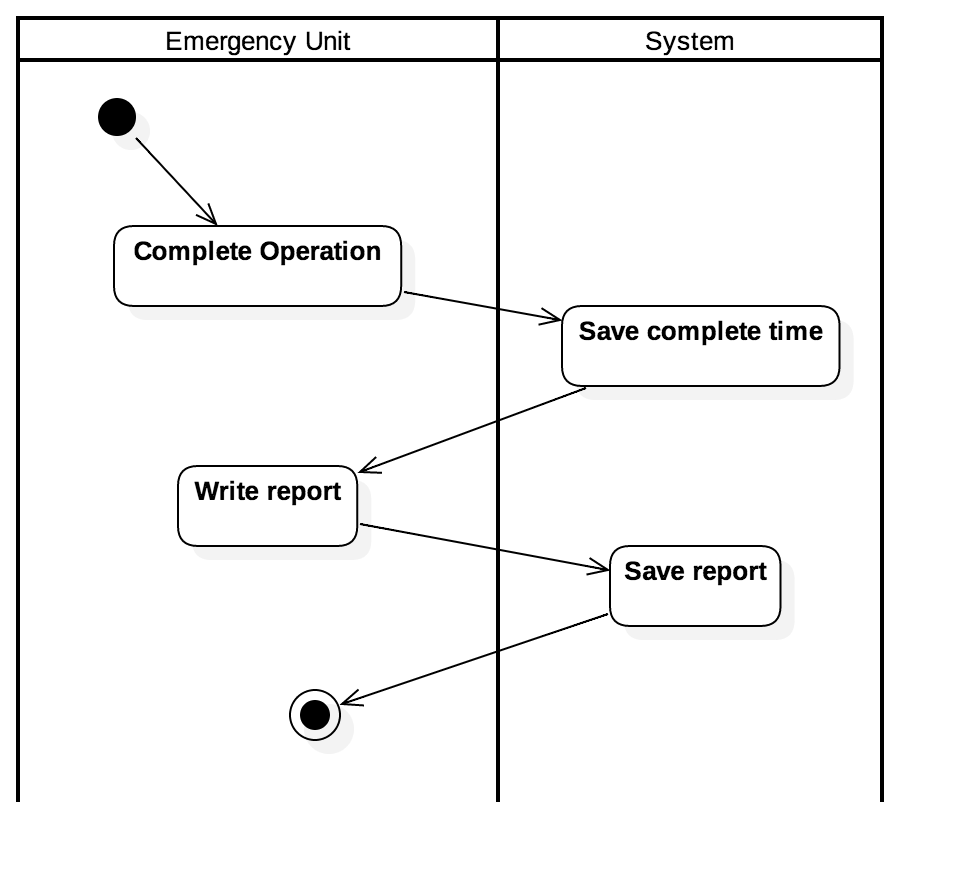
### Call for help



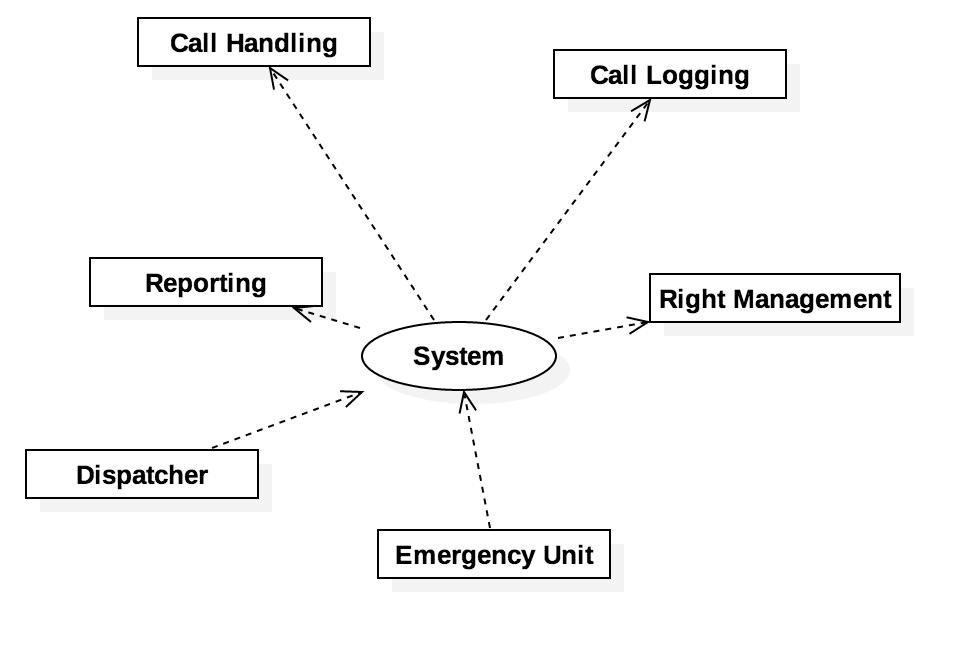
### Announce Call Details



### Make Report



## System Context Diagram



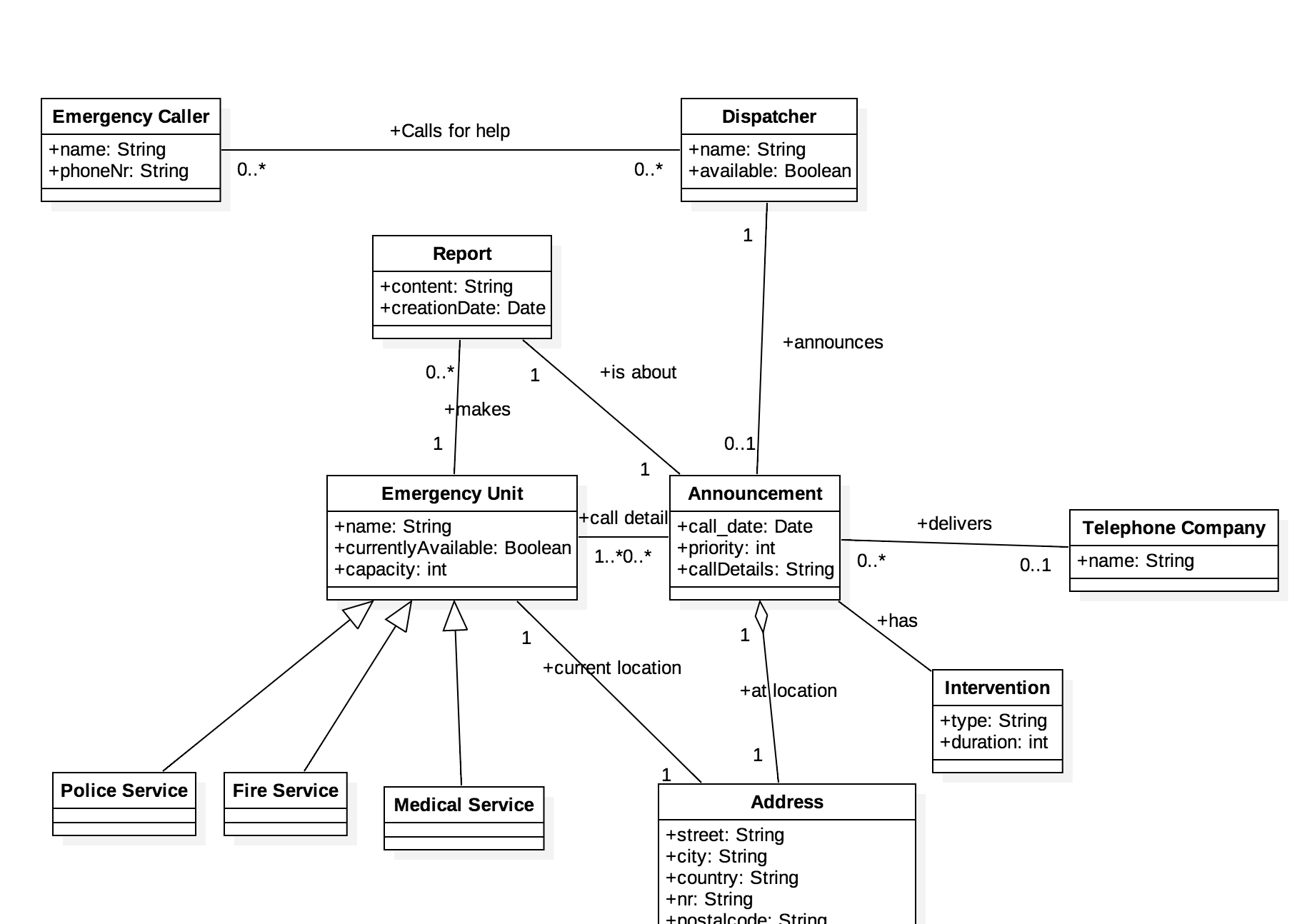
## Domain Model

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



# 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.
* 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 type of non functional requirement does not need an extra consideration and should be implemented what suits best the developer.

## Use Case Diagram

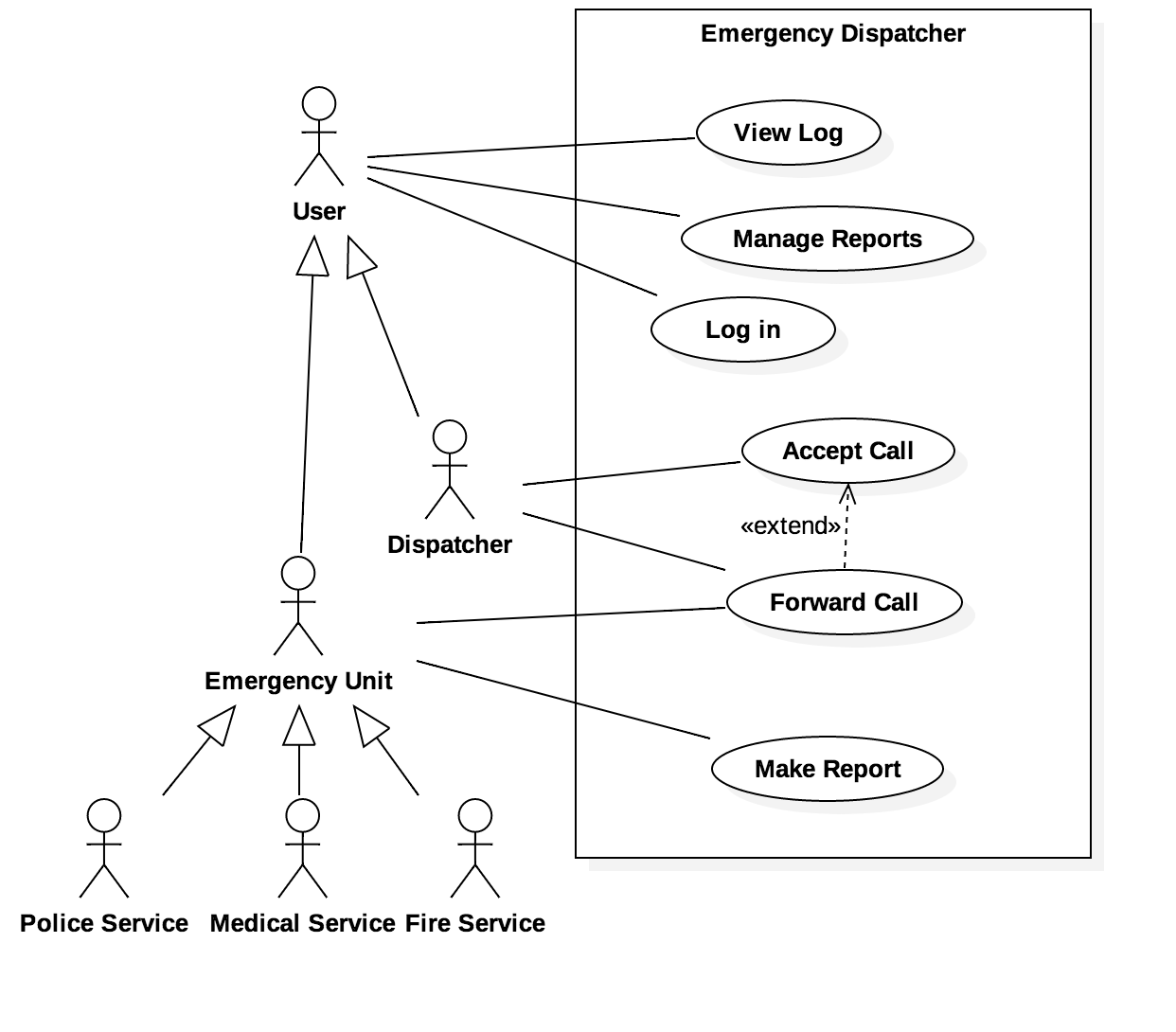
### Events

* 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



## System functions

### Accept Call

|  |  |  |
| --- | --- | --- |
| Function | Description | Importance |
| F1 | System accepts call | 10 |
| F2 | System asks for data (where, who, what happened) | 10 |
| F3 | The system validates input | 4 |
| F4 | The system stores the information | 8 |
| F5 | System forwards call to Emergency Unit | 10 |

### Forward Call

|  |  |  |
| --- | --- | --- |
| Function | Description | Importance |
| F1 | System lists available operations to emergency unit | 10 |
| F2 | Emergency Unit can choose operation and “take” responsibility for them | 10 |
| F3 | Emergency Unit can finish operation. | 10 |

### Log in

|  |  |  |
| --- | --- | --- |
| Function | Description | Importance |
| F1 | User can enter username and password and submit it | 10 |
| F2 | System checks if username exists. Password is being encrypted | 10 |
| F3 | Show available actions in the menu: Dispatcher: Accept Call, View Logs; For Emergency Unit: Forward Call; Make Report | 10 |

### Manage Reports

|  |  |  |
| --- | --- | --- |
| Function | Description | Importance |
| F1 | Fetch Reports for logged in user | 10 |
| F2 | Display table of available reports | 10 |
| F3 | User can choose a report | 6 |
| F4 | System displays report content | 10 |
| F5 | User may change report and system updates last modification date | 7 |

### Make Report

|  |  |  |
| --- | --- | --- |
| Function | Description | Importance |
| F1 | Find open operation that the unit needs to write a report for | 10 |
| F2 | User can type report into system | 10 |
| F3 | System saves additional data (last modification date, who edited it) | 7 |
| F4 | System validates data | 5 |
| F5 | System saves data and marks operation as being reported | 8 |

### View Log

|  |  |  |
| --- | --- | --- |
| Function | Description | Importance |
| F1 | Fetch data from database | 10 |
| F2 | Build statistics based on data: amount of operations per type, workload of units, operations total | 8 |
| F3 | Display data in a graphical user interface | 10 |

## Software Use Case Description

### Accept Call

* Name: Accept Call
* Description: When a person calls because of an emergency the dispatcher accepts the call and enters information into the system
* Stakeholders: Dispatcher (main), Emergency Caller
* Trigger Event: Call/SMS/…
* Precondition: A call is on the way; The Dispatcher is logged in
* Postcondition: The Call has been logged; The emergency unit is notified

|  |  |
| --- | --- |
| **Actors** | **System** |
| 1. Person calls dispatcher |  |
| 2. Dispatcher accepts call |  |
|  | 3. Request user to enter information (where, who, what, …) |
|  | 4. Create log and store in database. |
| 5. Dispatcher enters information by asking the callee. |  |
|  | 6. The system validates input |
|  | 7. The system stores the information |
|  | 8. System forwards call to Emergency Unit |

## Forward Call

* Name: Forward Call
* Description: When an Emergency Unit is ready for an operation the system forwards the call to the unit. The unit executes it and the system logs necessary information.
* Stakeholders: Emergency Unit (main)
* Trigger Event: Emergency Unit is ready
* Precondition: The call has been accepted
* Postcondition: Emergency unit has accepted the call and is executing the operation
* Exceptions: User does not finish operation

|  |  |
| --- | --- |
| **Actors** | **System** |
| 1. Emergency Unit wants to see available operations |  |
|  | 2. System fetches available operations for unit and displays them with available information. |
| 3. Emergency Unit selects most appropriate emergency |  |
|  | 4. System updates database and assigns operation to unit |
| 5. Emergency unit finishes operation |  |
|  | 6. System updates database and sets finish time. System requests user to write a report |
| 7. Emergency unit writes a report |  |
|  | 8. System saves report to database and marks the operation as finished. |

### Log in

* Name: Log In
* Description: User logs into the system and has to verify himself.
* Stakeholders: User (Emergency Unit/Dispatcher)
* Trigger Event: User wants to use the system
* Precondition: User is NOT logged in
* Postcondition: User IS logged in if data is right
* Exceptions: Invalid username/password

|  |  |
| --- | --- |
| **Actors** | **System** |
| 1. User wants to log into the system. |  |
|  | 2. System displays a form where the user may enter his credentials. |
| 3. User enters username and password and submits |  |
|  | 4. System checks if username exists. Password is being encrypted |
|  | 5. Show available actions in the menu: Dispatcher: Accept Call, View Logs; For Emergency Unit: Forward Call; Make Report |

### Manage Reports

* Name: Manage Reports
* Description: A user can manage reports by either view or edit them.
* Stakeholders: User (Emergency Unit/Dispatcher)
* Trigger Event: User wants to view/edit 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

|  |  |
| --- | --- |
| **Actors** | **System** |
| 1. User requests reports available for him or her. |  |
|  | 2. Fetch Reports available for user from database. Dispatcher can view all reports. Emergency Units can only view reports of emergencies they were involved in. |
|  | 3. Display table of available reports. |
| 4. User clicks by choosing report |  |
|  | 5. System displays report content and it may be editable if he has sufficient rights (if he is an Emergency Unit) |
| 6. User may edit report |  |
|  | 7. If user changed report it is saved to the database. Additionally the system updates “modification date” and who edited it. |

## Make Report

* Name: Make Report
* Description: When an operation is finished the unit has to make a report about the operation.
* Stakeholders: Emergency Unit (main)
* Trigger Event: Emergency Unit finished an operation
* Precondition: Emergency Unit finishes operation; Emergency Unit is logged in
* Postcondition: A proper report is in the database

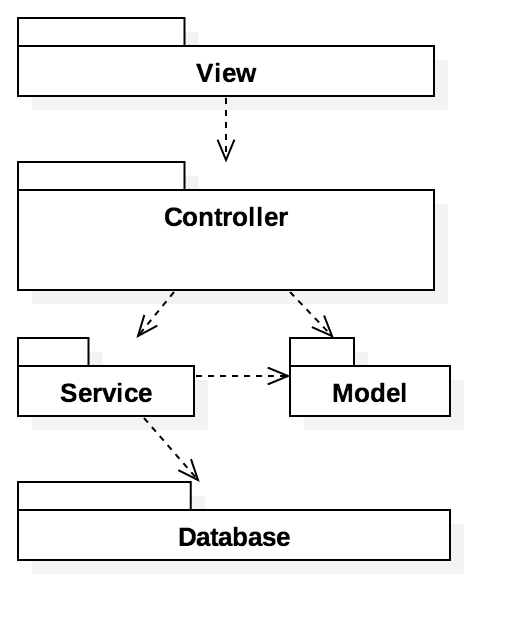
|  |  |
| --- | --- |
| **Actors** | **System** |
| 1. Emergency unit finished an operation and needs to write a report for. |  |
|  | 2. Find open operation that the unit needs to write a report for and display form. |
| 3. User types report into system and submits it. |  |
|  | 4. System saves additional data (last modification date, who edited it) |
|  | 5. System saves data to database and marks operation as being reported |

## View Log

* Name: View Log
* Description: Displays a comprehensive log of all calls and operations that were ever done. Additionally some useful statistics are also shown.
* Stakeholders: User (Dispatcher, Emergency unit)
* Trigger Event: User wants to view logs
* 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

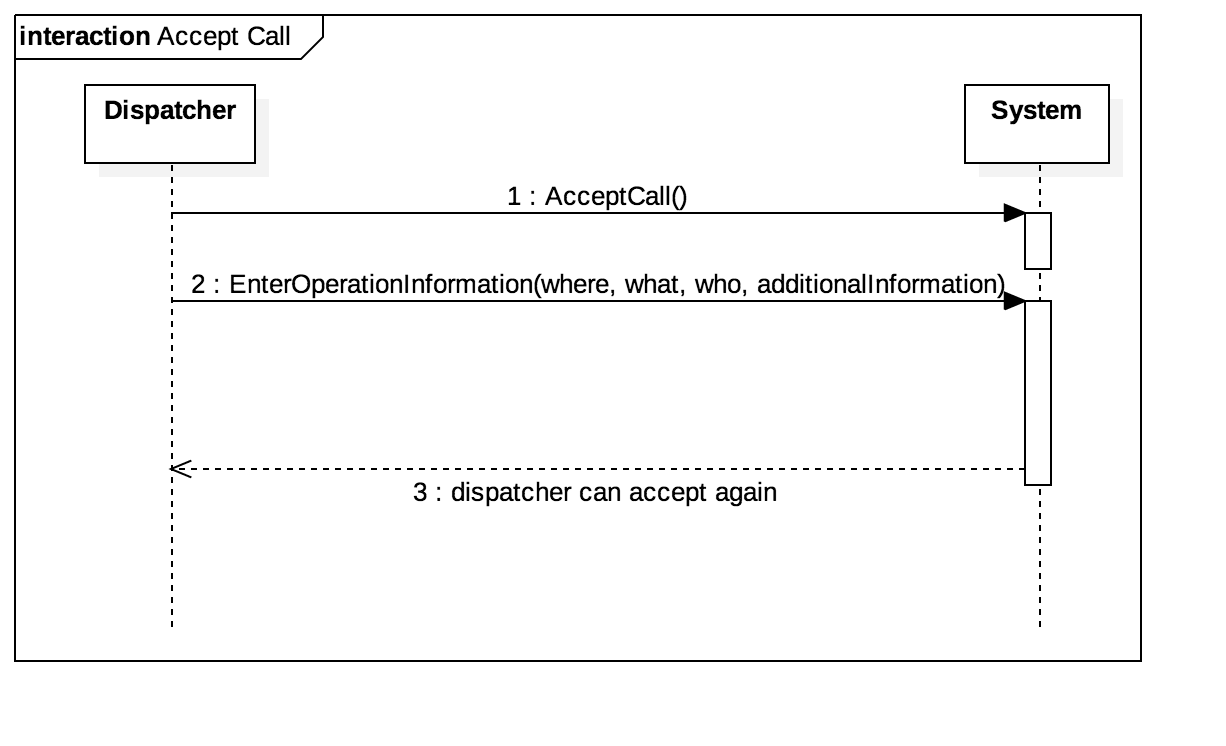
|  |  |
| --- | --- |
| **Actors** | **System** |
| User wants to view logs |  |
|  | Fetches data from database. Dispatcher can view all logs, Emergency Unit can only view their logs |
|  | Build statistics based on data: amount of operations per type, workload of units, operations total |
|  | Display data in a tables. |
| User may view report of operation. |  |
|  | If user clicked on report of operation user is forwarded to “Manage Report” |

## Architecture Draft Document Package Diagram

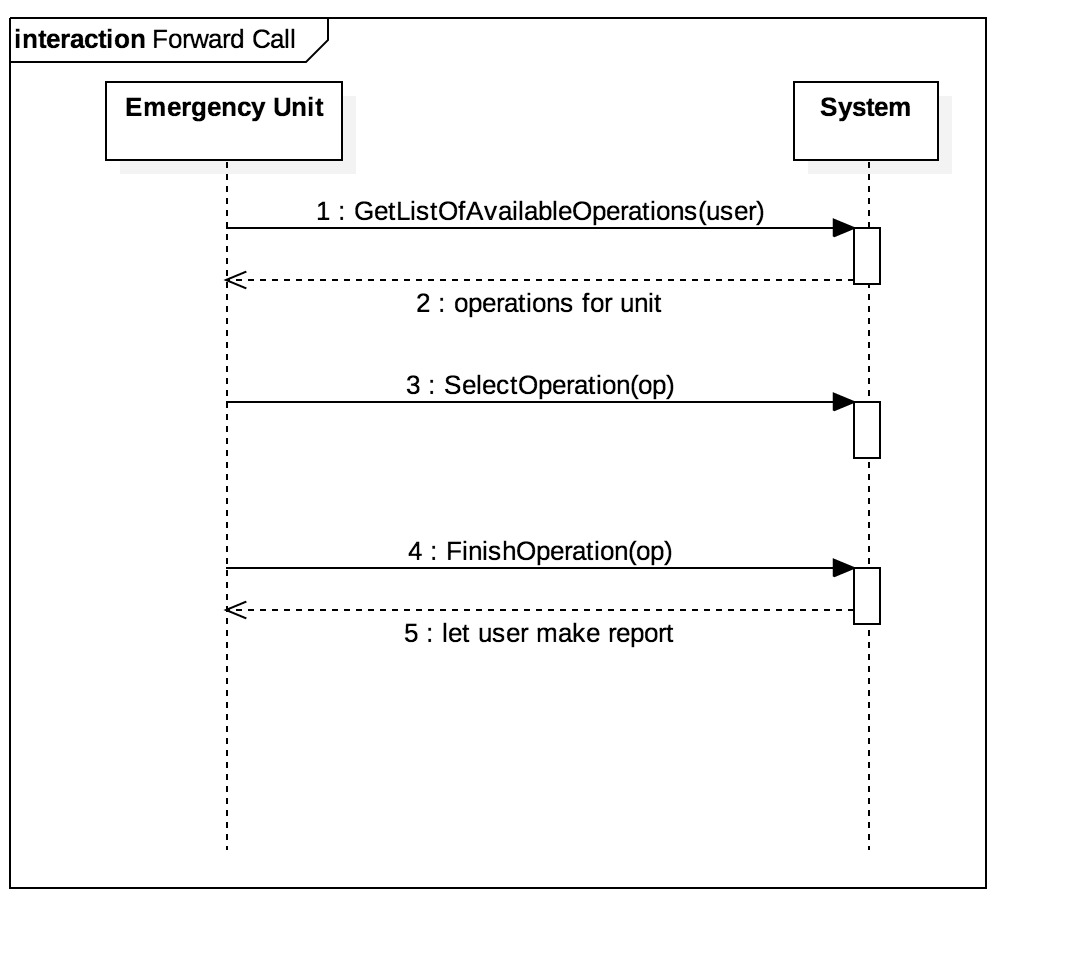


## System Sequence Diagrams

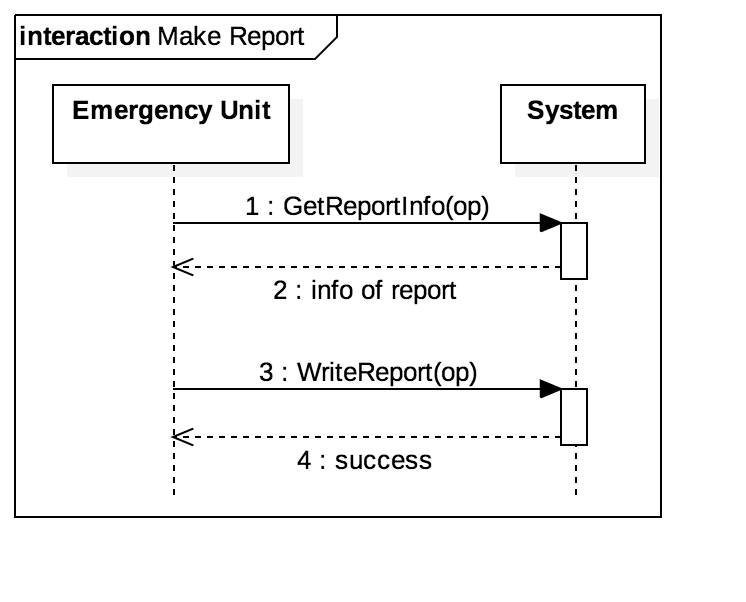
### Accept Call



### Forward Call



### Make Report



### Manage ReportsInternet Explorer:Users:rfischer:Dropbox:Studium_Sem4:SDM:FinalMissingDocs:Manage Reports Sequence.png

### View LogsInternet Explorer:Users:rfischer:Dropbox:Studium_Sem4:SDM:FinalMissingDocs:View Logs Sequence.pngLoginInternet Explorer:Users:rfischer:Dropbox:Studium_Sem4:SDM:FinalMissingDocs:Log In Sequence.png

## Operation Contract

### WriteReport

Name: WriteReport

Responsibilities: Writes a new report to the database and updates the operation object  
Exceptions: None  
Preconditions: Operation exists; Report does not exist in database; Operation has an assigned unit; Operation report attribute is null  
Postconditions: Operation “report” attribute set; Report has a valid “modification date” and a valid “modification unit” (the unit which edited it last)

### UpdateReport

Name: UpdateReport

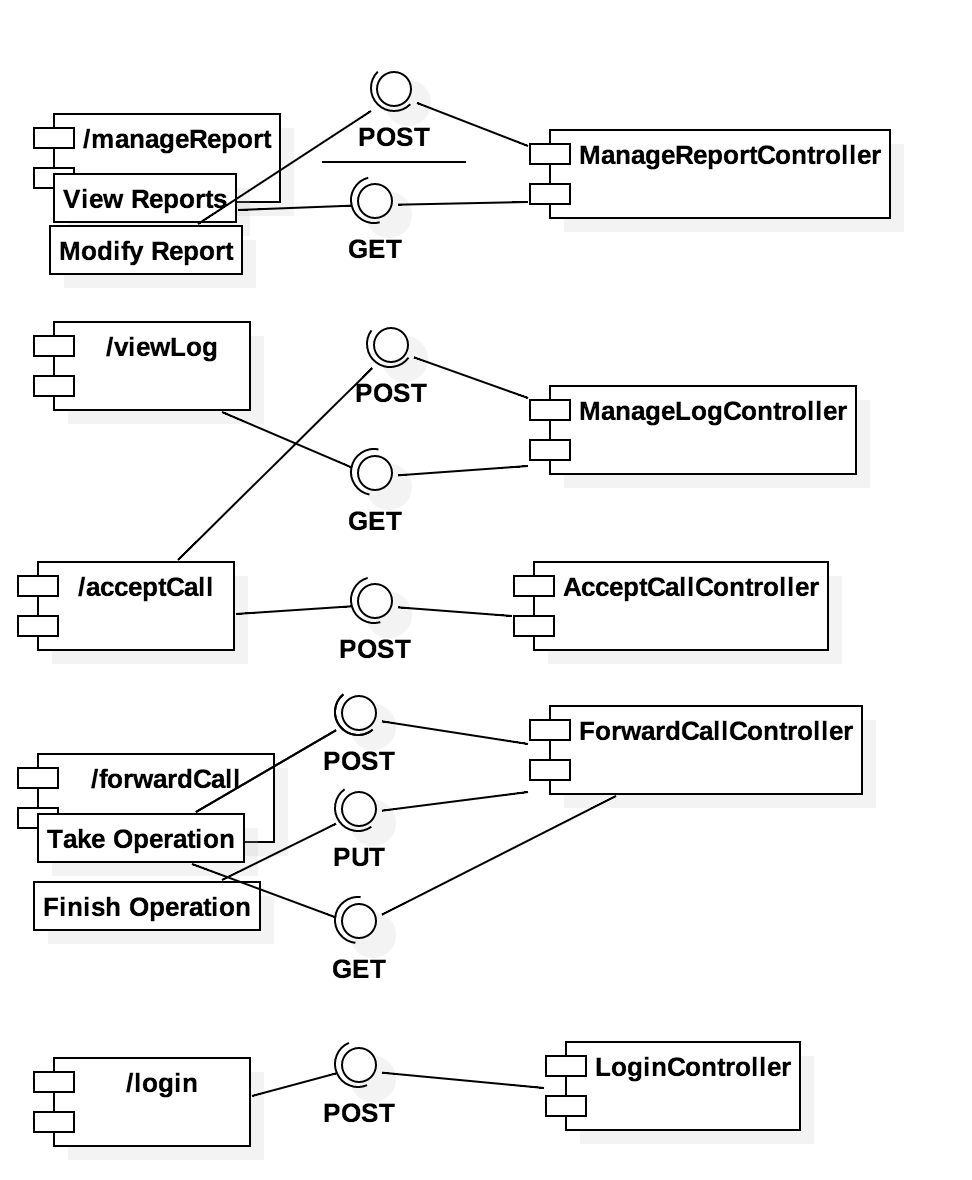
Responsibilities: Updates an existing report to the database and updates the operation object  
Exceptions: None  
Preconditions: Operation exists; Report is exists in database; Operation has an assigned unit  
Postconditions: Operation “report” attribute set and it did not change; Report has a valid “modification date” and a valid “modification unit” (the unit which edited it last)

# 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))

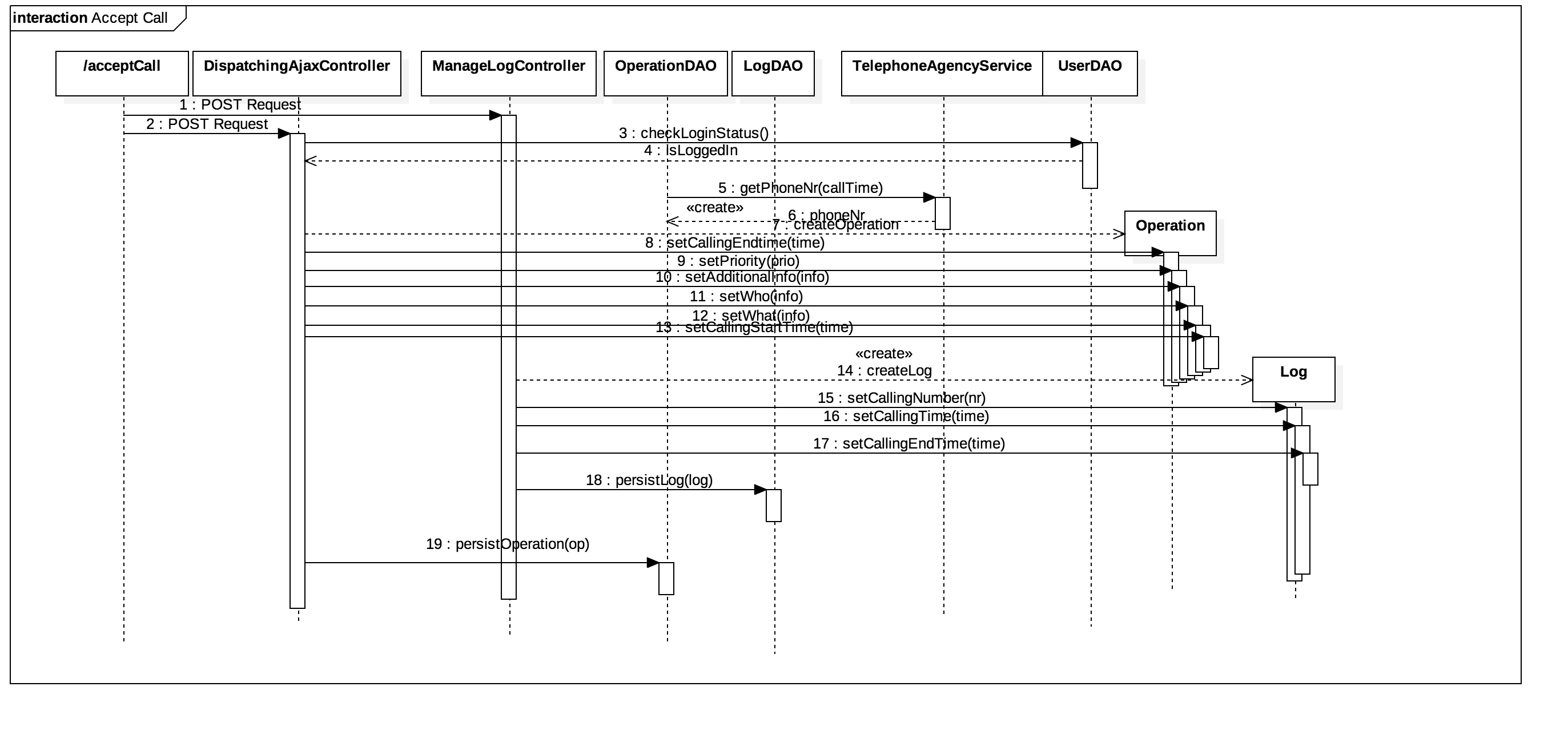
## Component Diagram

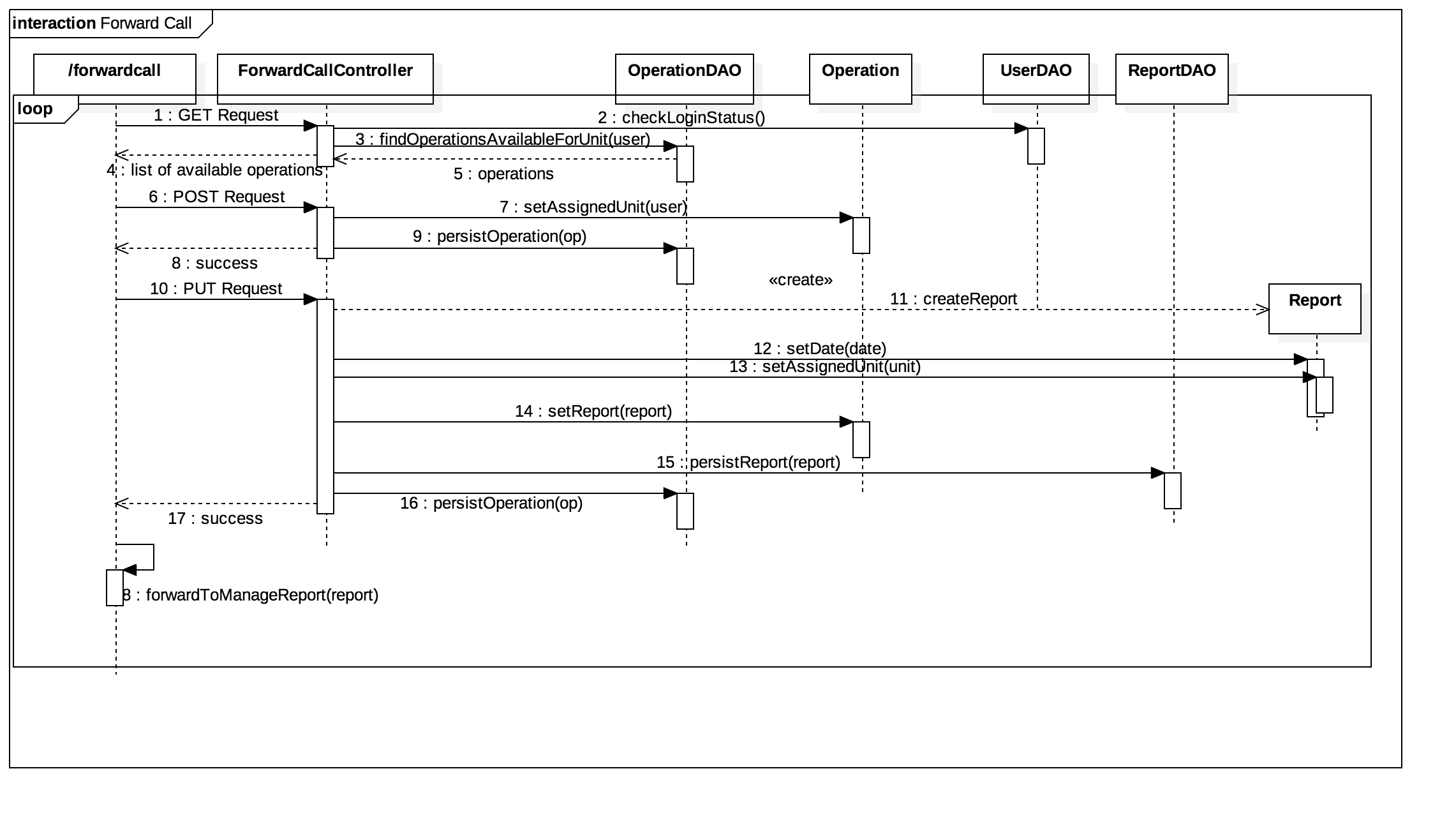


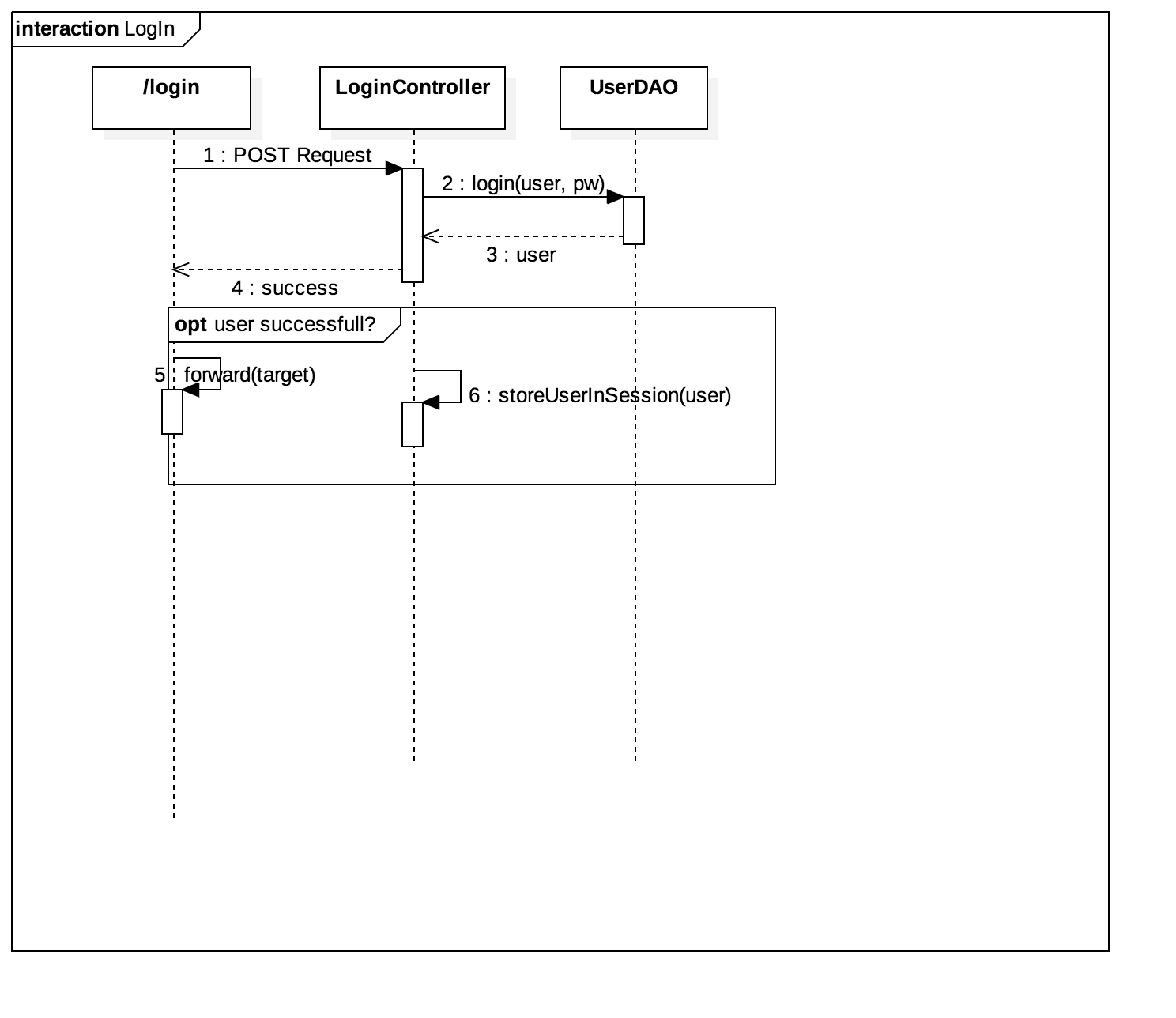
### Responsibilities to classes

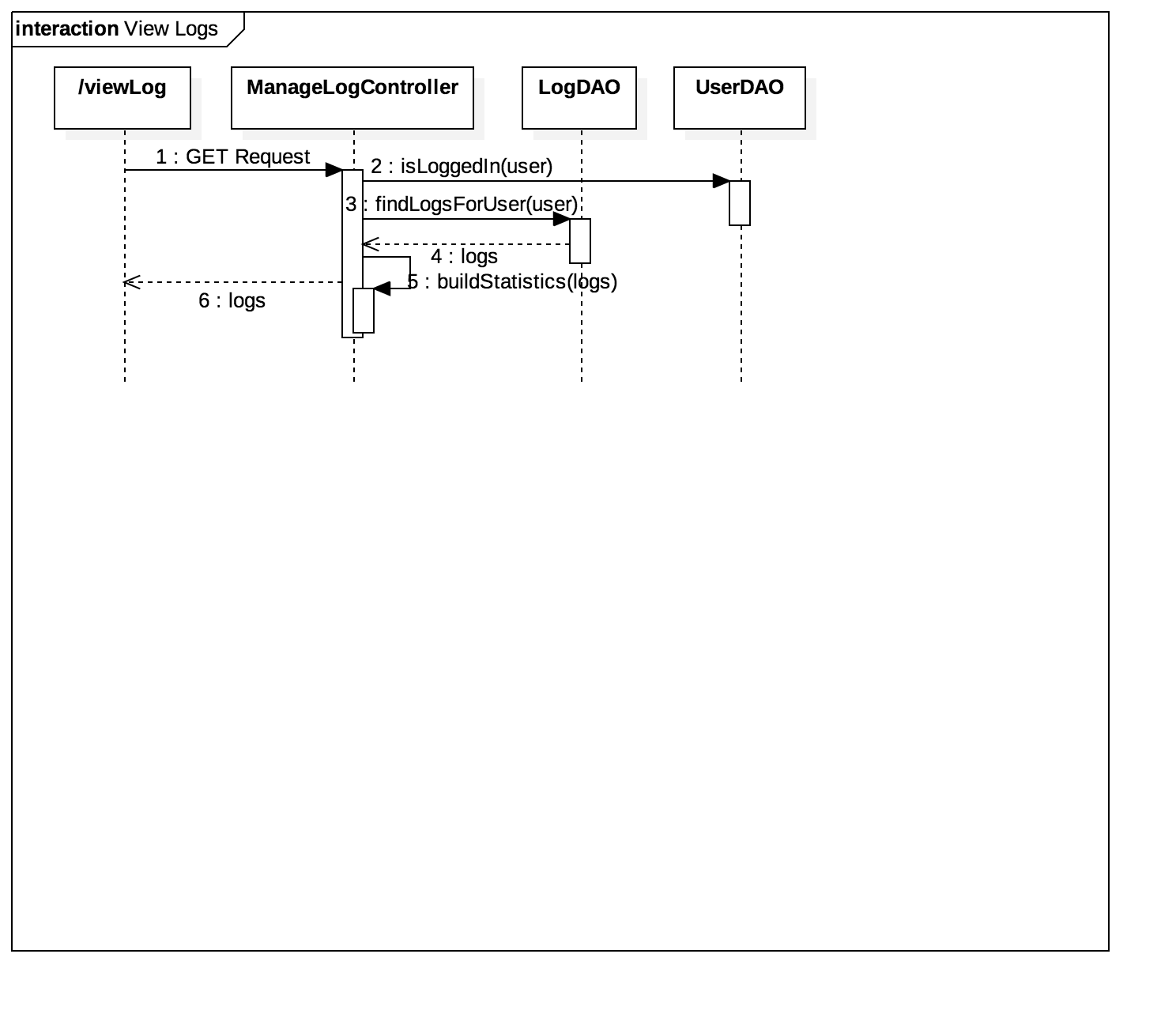
* View Log
  + UserDAO, LogDAO, ManageLogController, Log
* Manage Report
  + UserDAO, ReportDAO, ManageReportController, Report
* Make Report
  + UserDAO, ReportDAO, ManageReportController, Report
* Accept Call
  + UserDAO, DispatcherController, ManageLogController, OperationDAO, LogDAO, TelephoneAgencyService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

## Interaction Model

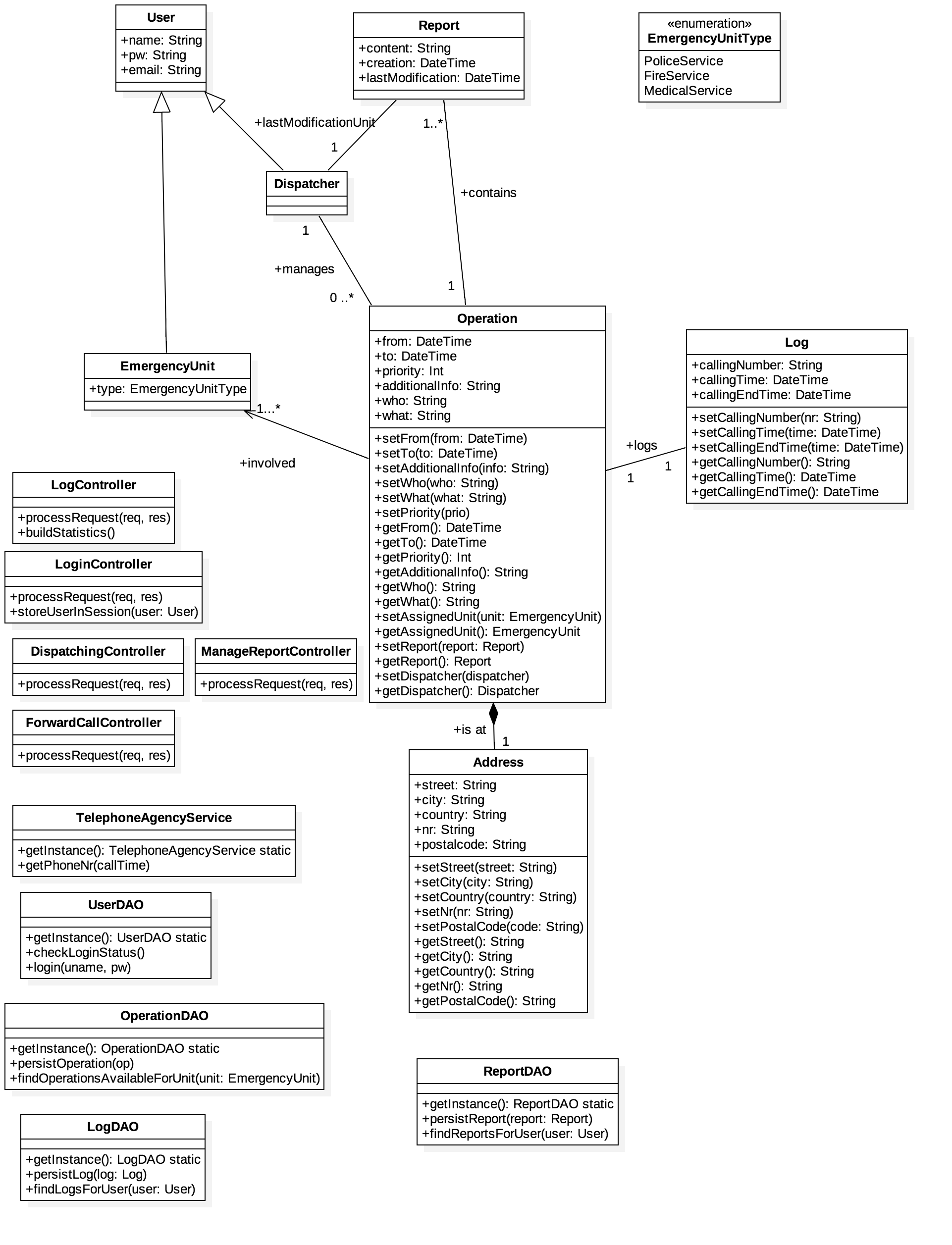




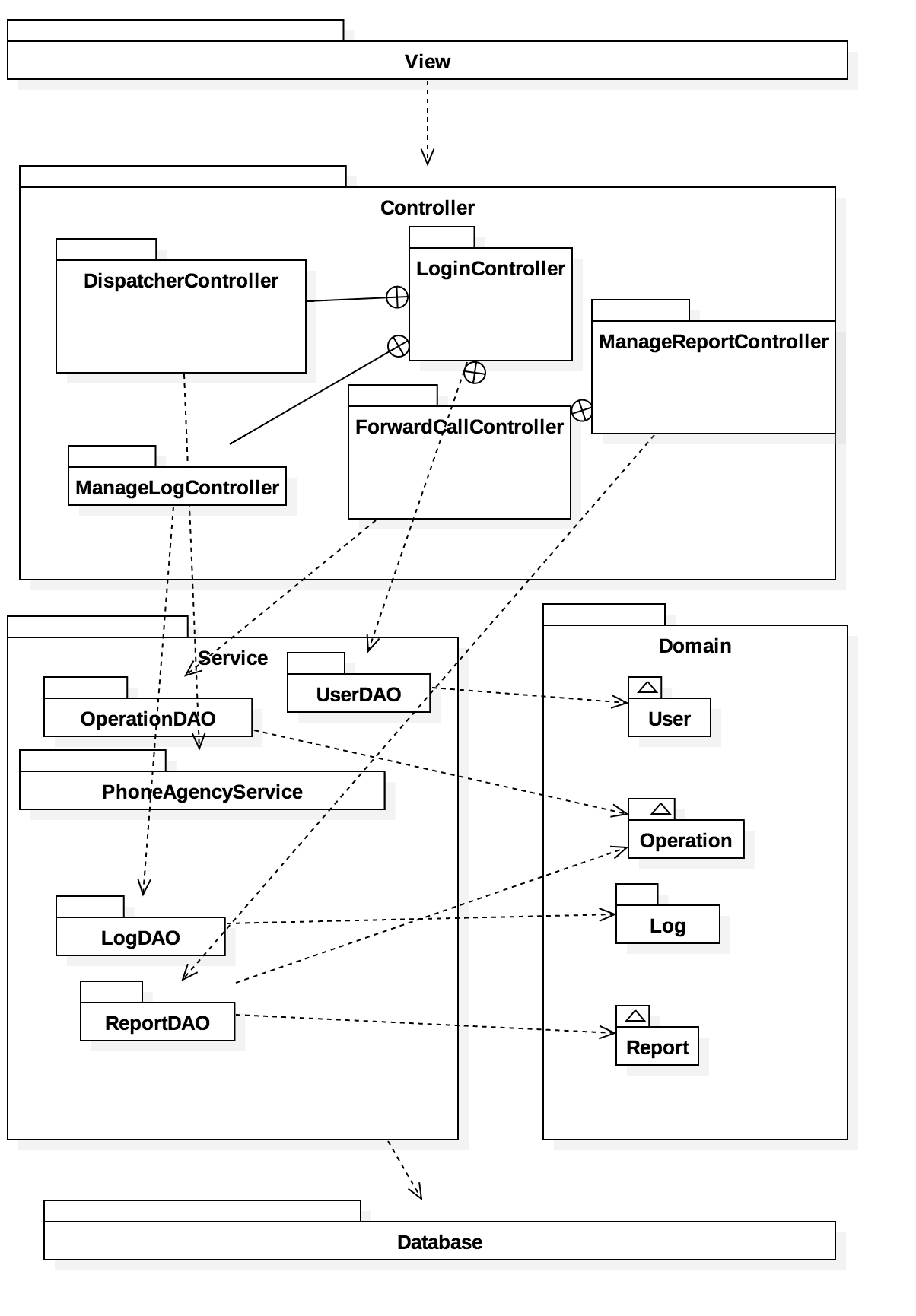




## Design Class Diagram



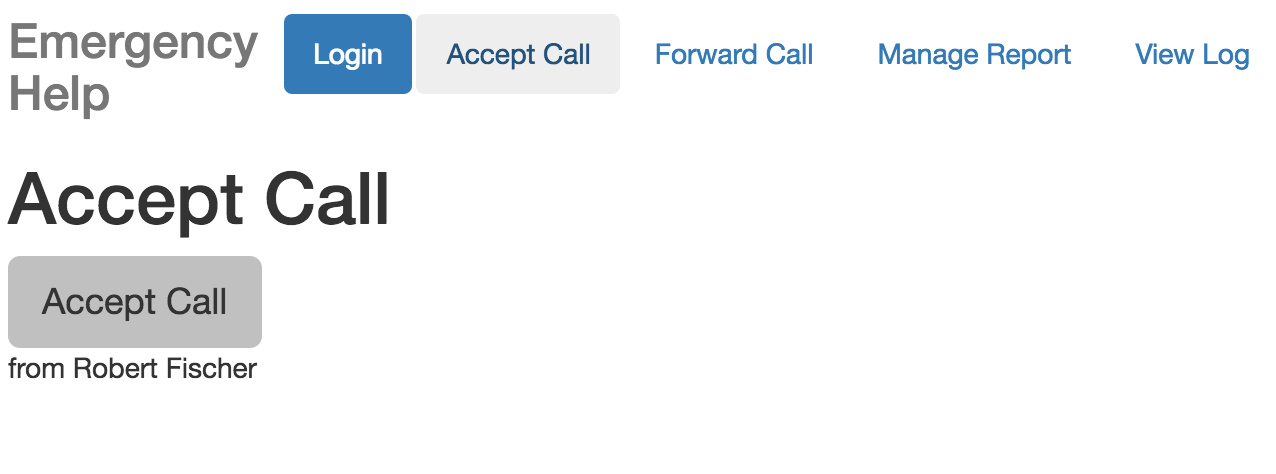
## Architecture Package Diagram



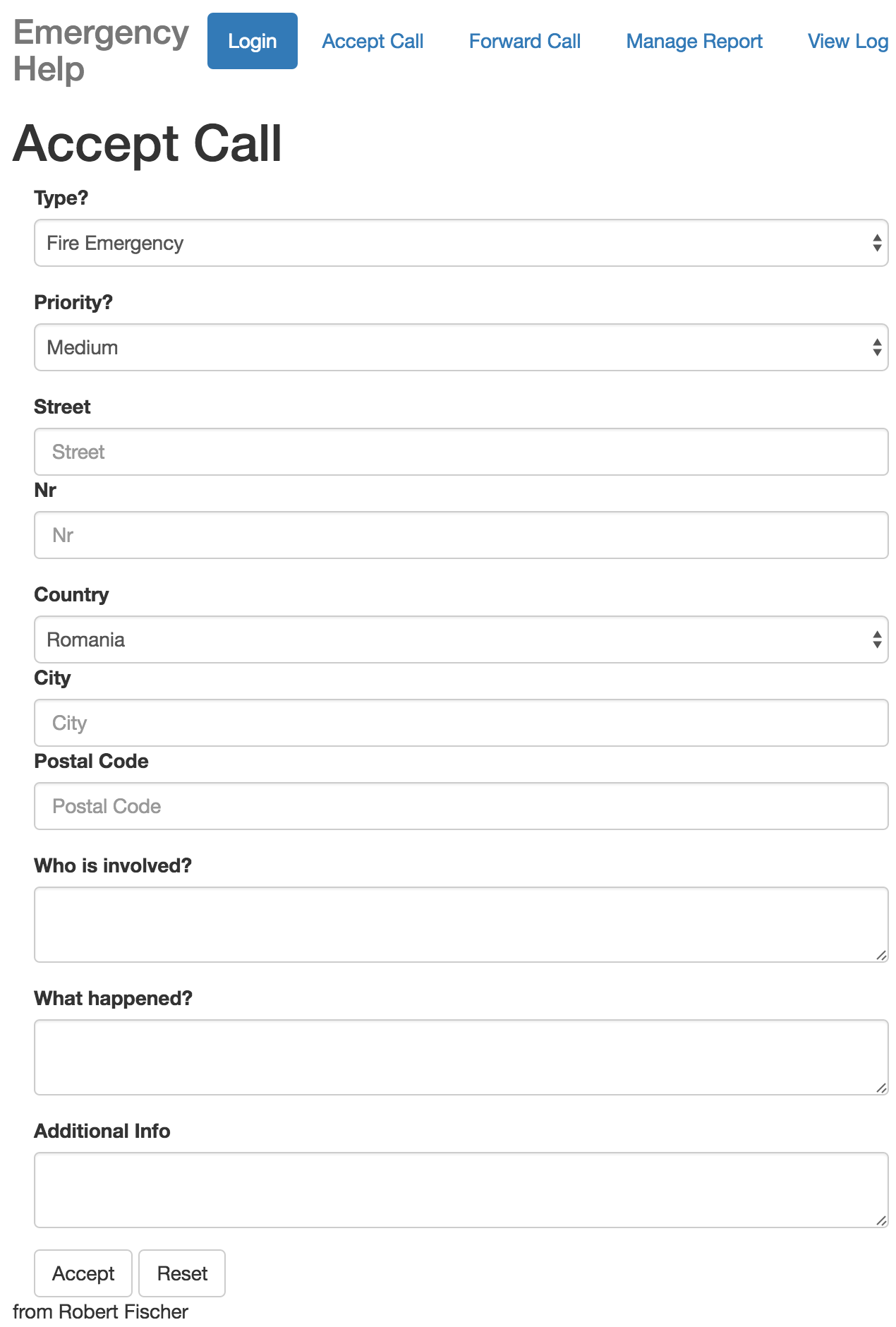
## User Interface Design

### Accept Call

* Wait for call and accept on incoming

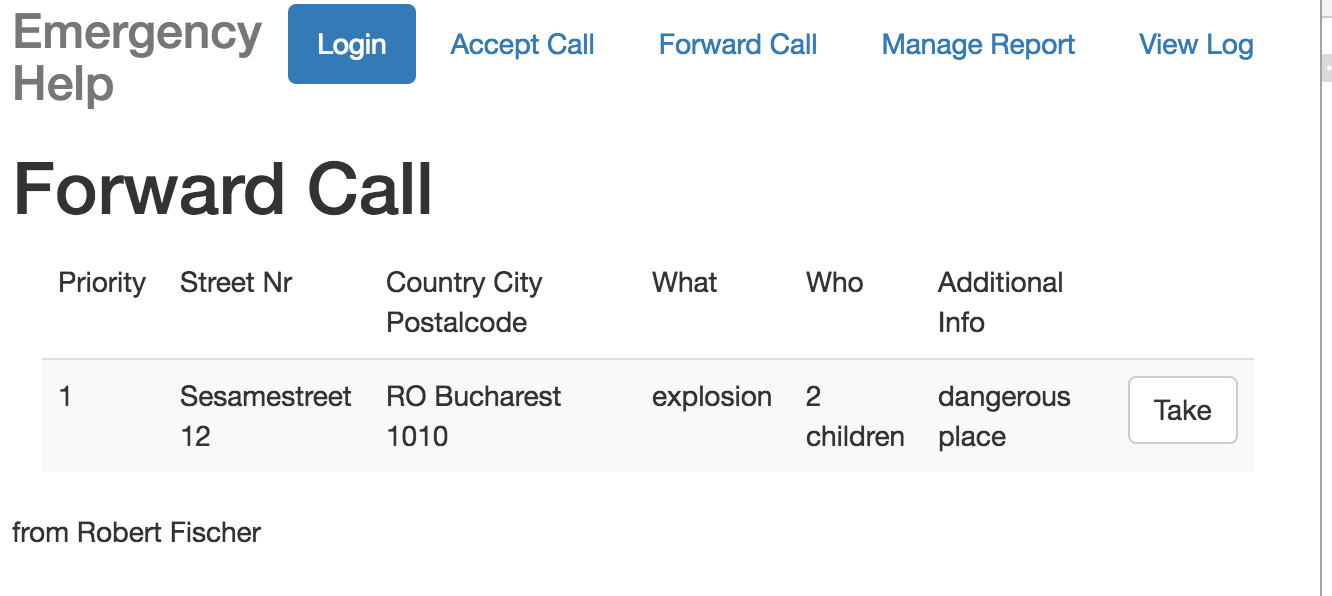


* Enter call information for Emergency Unit

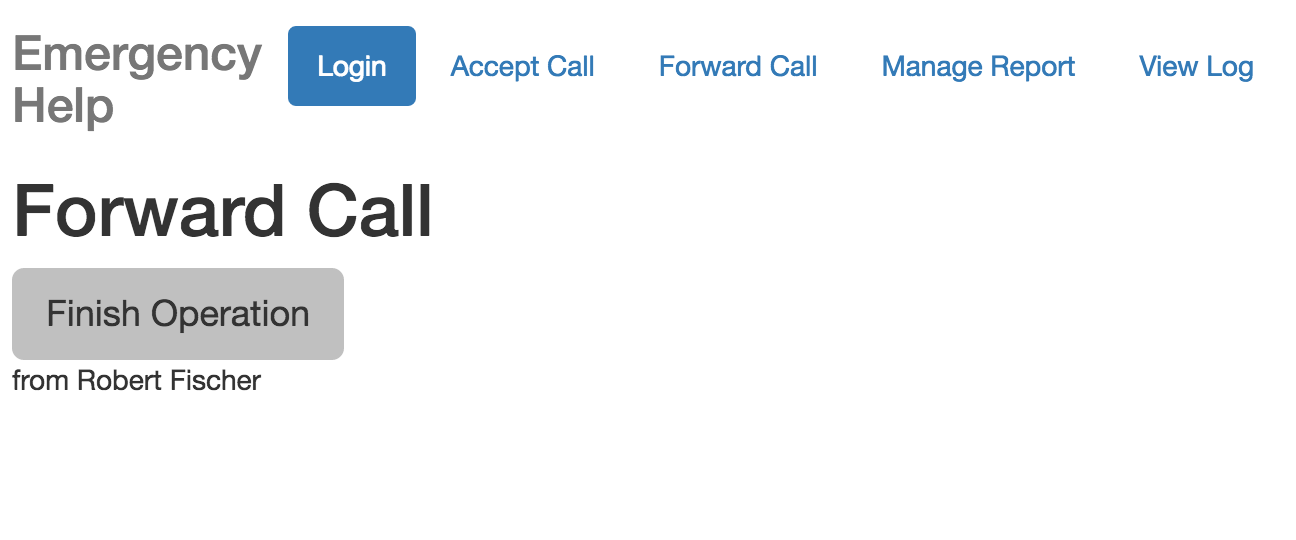


### Forward Call

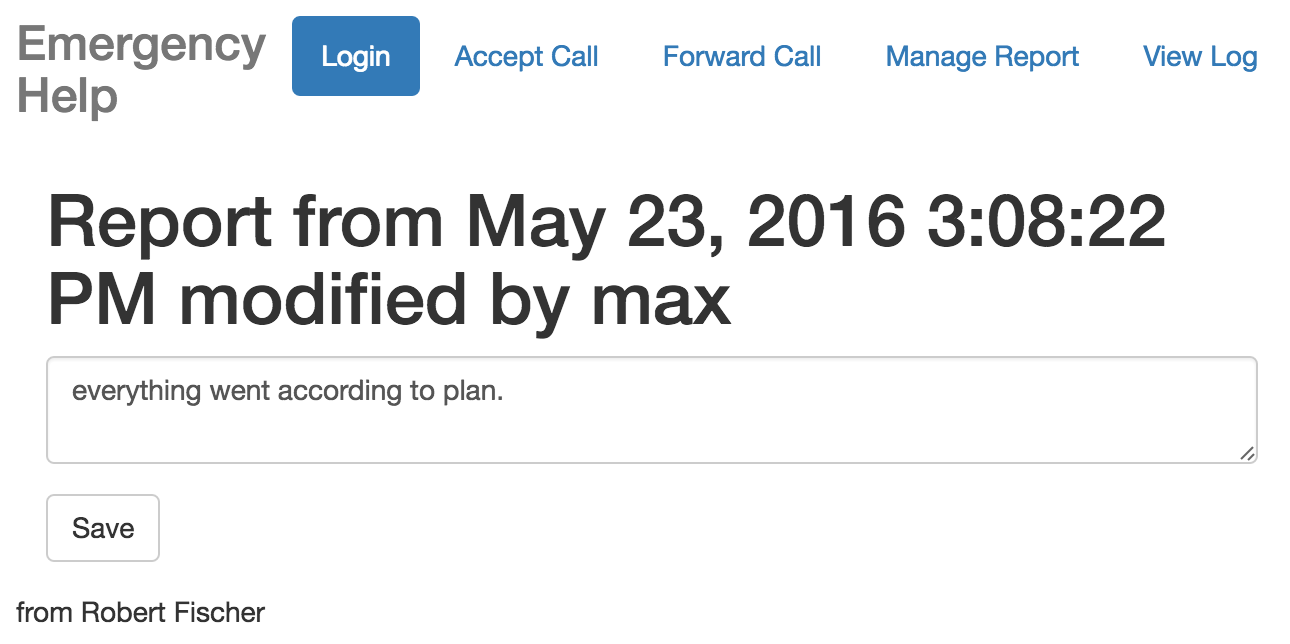
* Ready to forward calls



* Operation in progress

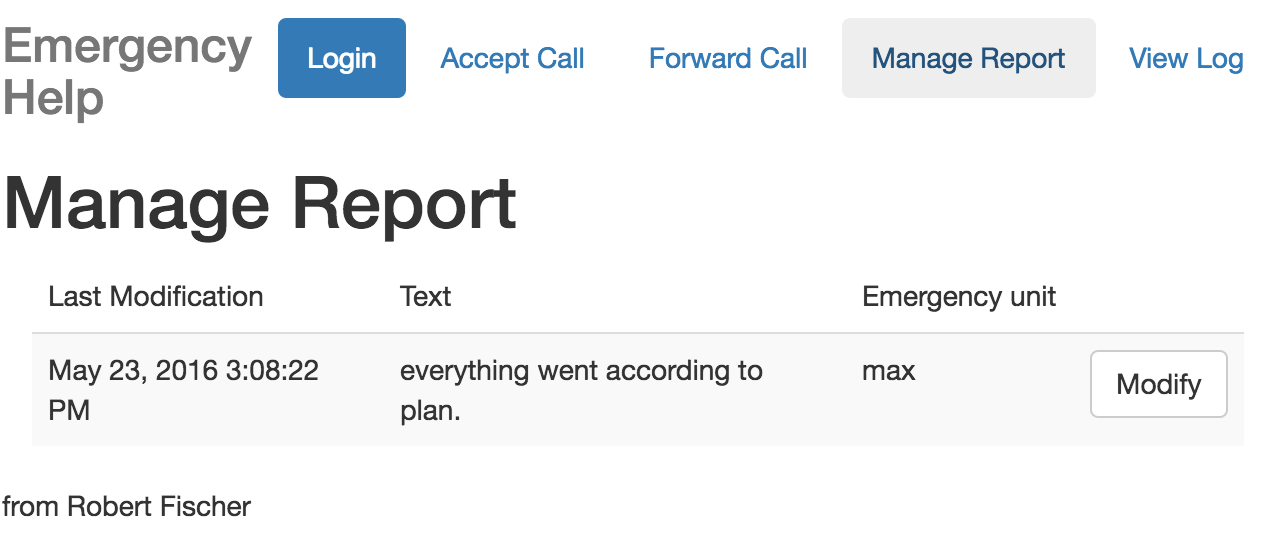


* Write report after operation finished



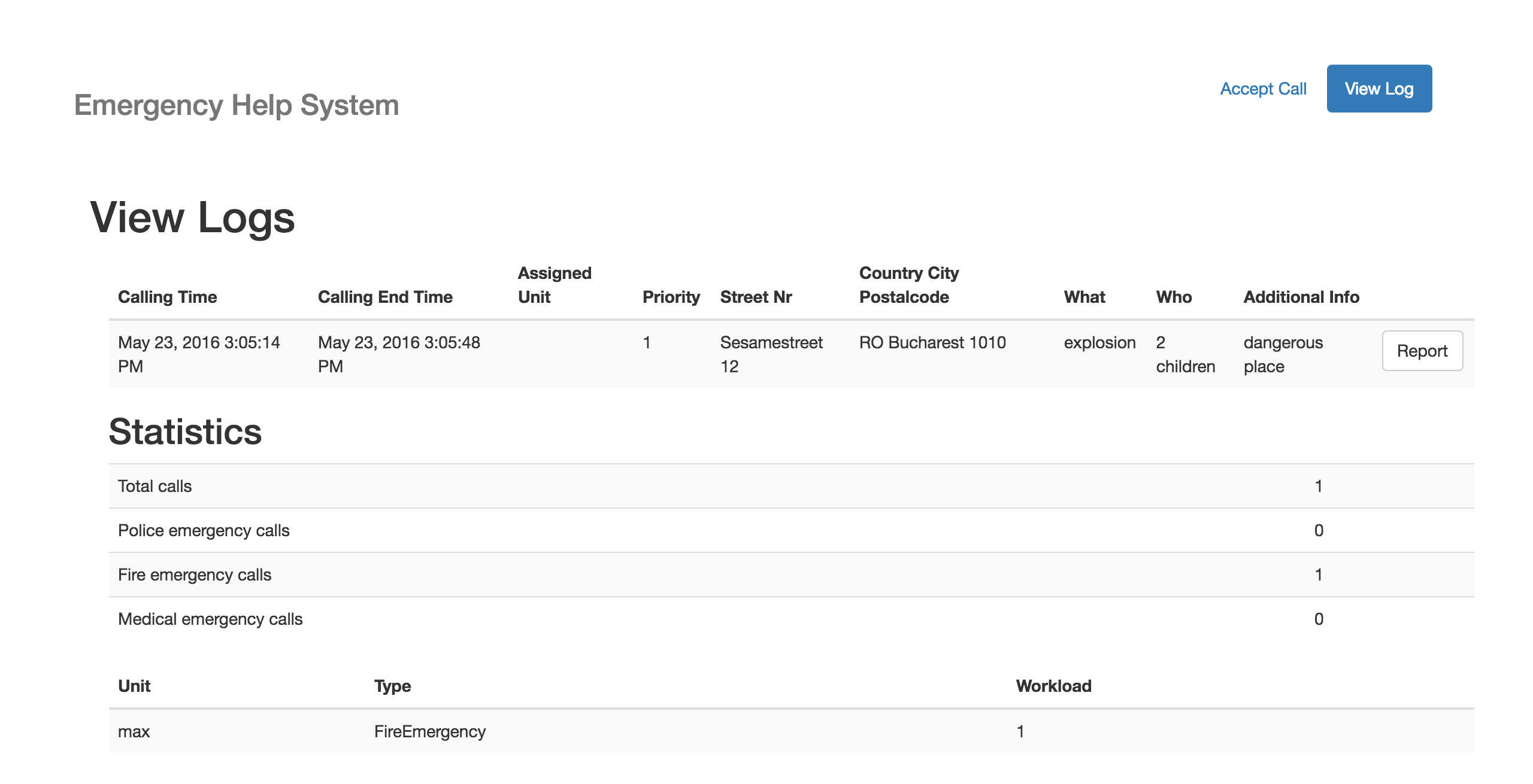
### Manage Report

* 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

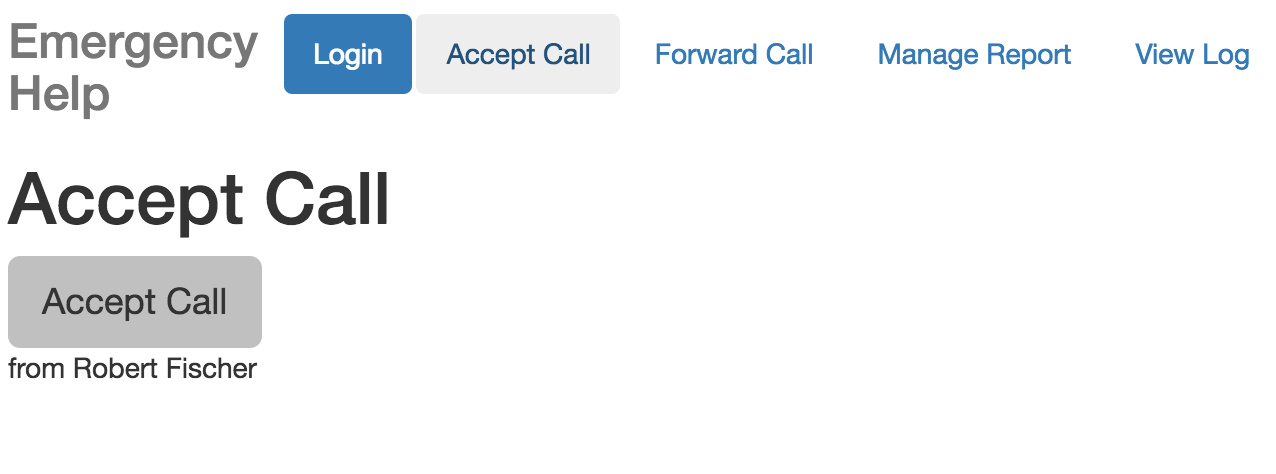
For the backend you need a proper JDBC datasource called “myDatasource” to be set up and a working Glassfish 4.x installation. Additionally there needs to be a accessible MySQL 5.x database.

The test users are:

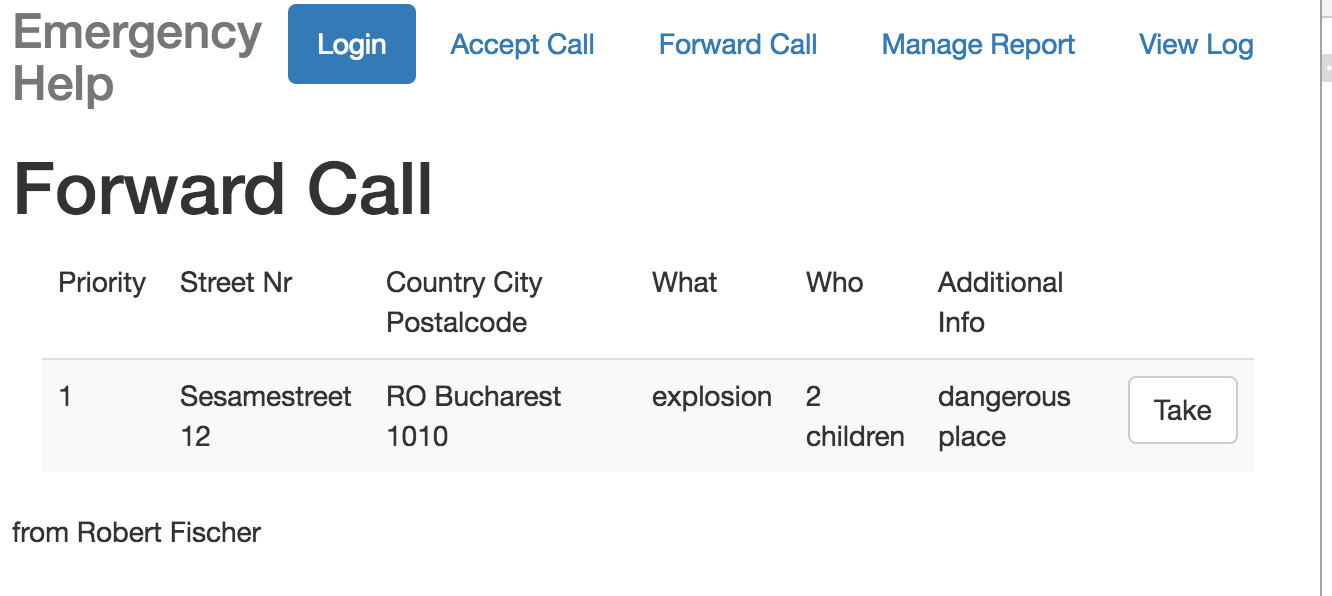
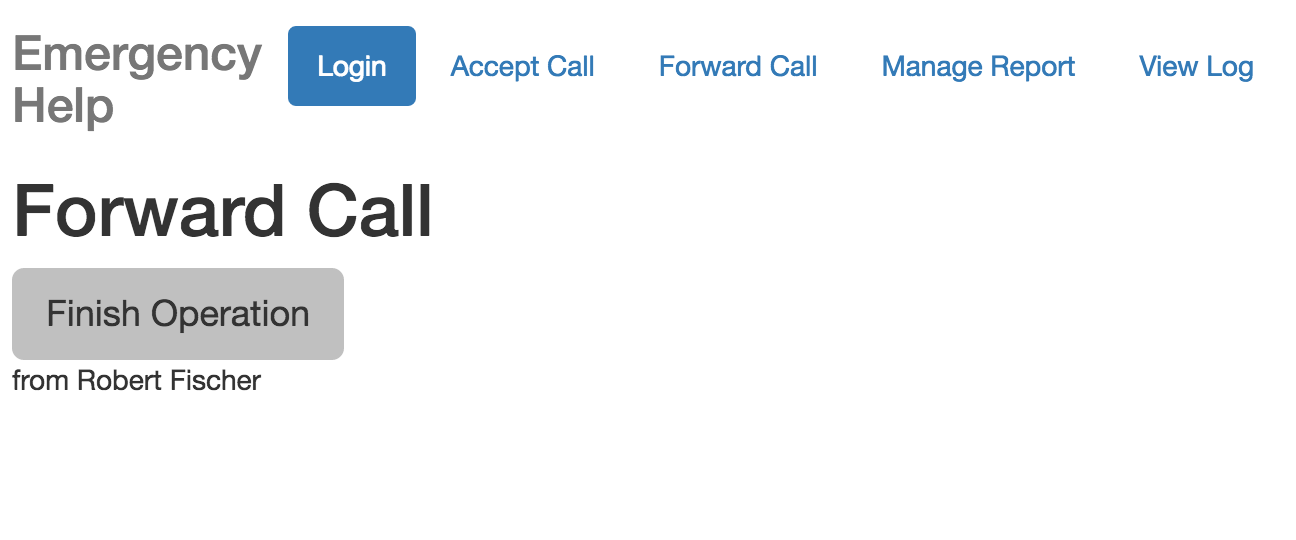
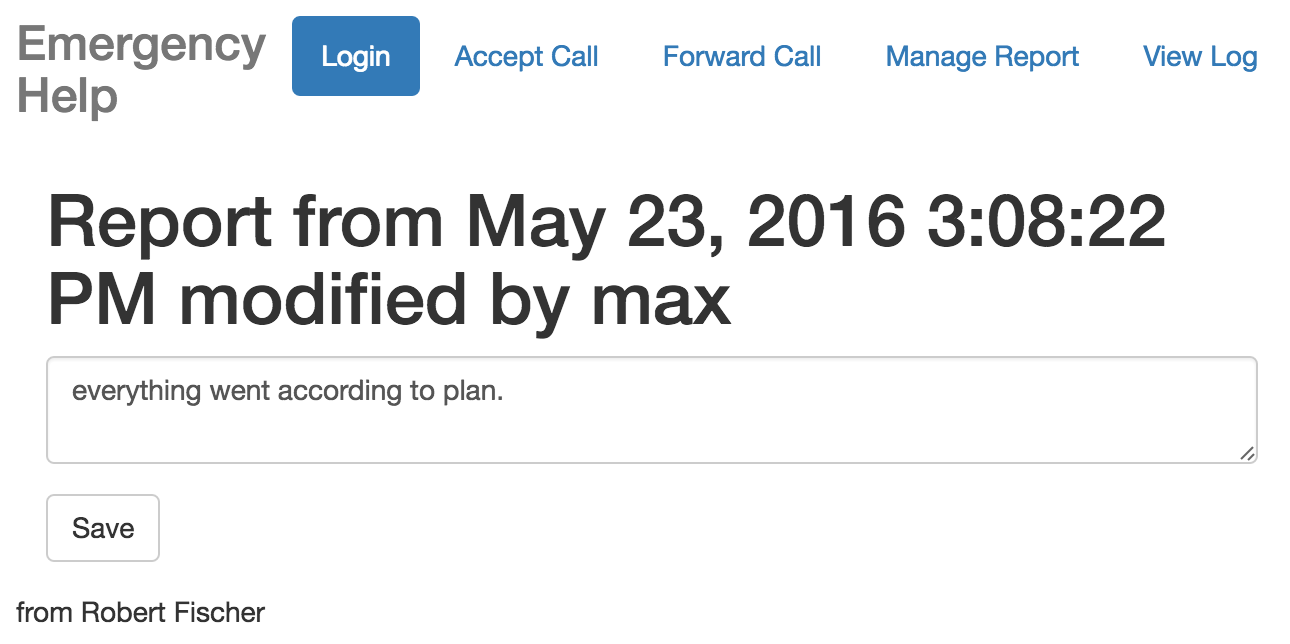
* username: robert password: a … Dispatcher
* username: max password: a … Fire Emergency
* username: alex password: a … Medical Emergency
* username: dorina password: a … Police Emergency

## User Manual

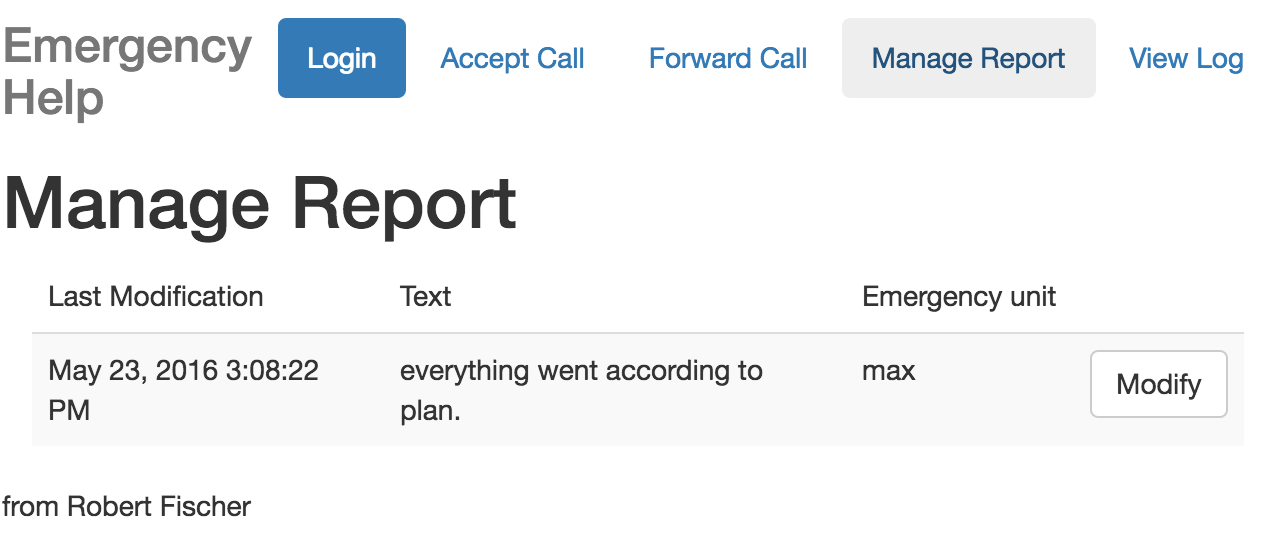
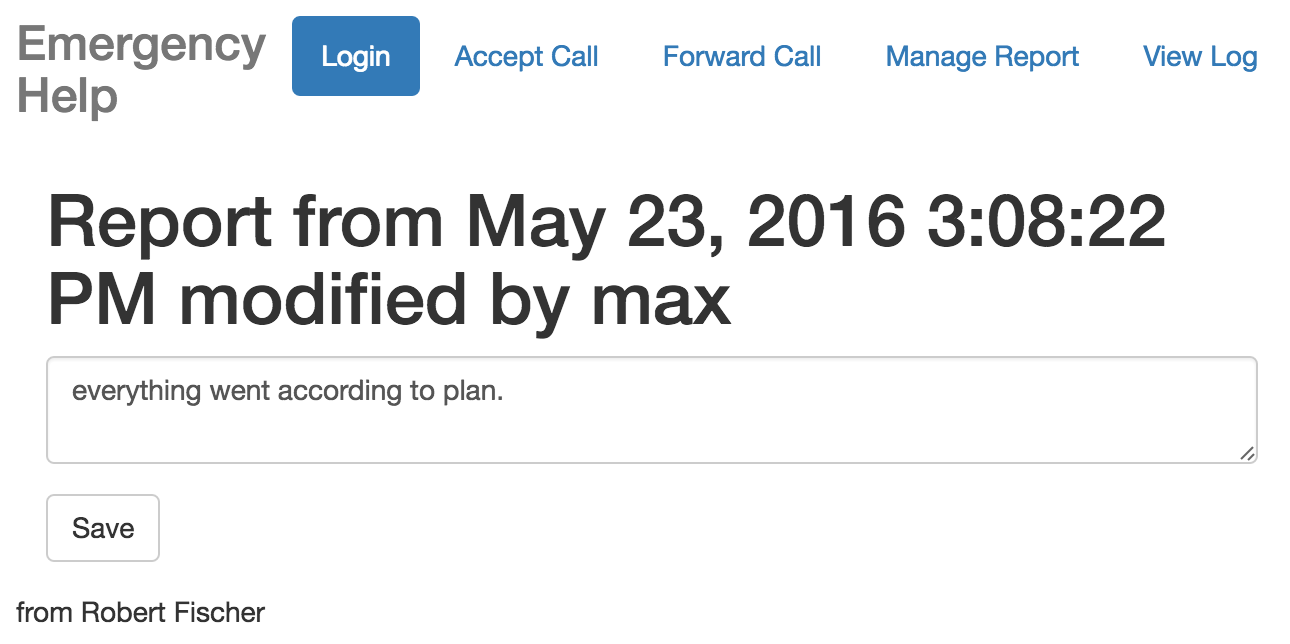
### You are a dispatcher?

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)

