Final System Document

CEN4010 – Software Engineering
Prepared for Peter Clarke
By Team 5

Teriq Douglas
Yovanni Jones
M. Kian Maroofi
Armando J. Ochoa
Anthony Sanchez-Ayra

December 03, 2019

Abstract

Student Organization System (SOS) is a web-based system meant to provide leaders and administrators of organizations a way to manage members and events. Simultaneously, it allows users to monitor the events and organizations they belong to. Although it is like other organization systems which are hosted and managed by university and colleges, the SOS differs in that it is primarily online and has a primary focus on event hosting and user interaction. The SOS is developed using the Unified Software Development Process (USDP), which is described in the two first sections of this document. The specifications of the system are captured in the form of use cases, which describe the use case model of the USDP and creating UML class and sequence diagrams, which are part of the Analysis model of the USDP. As part of the Design model, this document contains detailed descriptions of the chosen architectures for the SOS (Three-Tier (3TA) and Repository) as well as detailed descriptions of the decomposition of the SOS along with detailed diagrams explaining the functionality of each subsystem. The SOS also uses multiple design patterns (namely, Singleton, Command, Façade, and Builder) to ensure the efficiency and reuse of code in the system. This document also describes the deployment, implementation and test models for the SOS. The deployment model for the system is described in a detailed manner using deployment diagram. The implementation model for SOS is also found to show the interfaces of the Java classes that were made for the backend of the system. Finally, the test model is shown to ensure the validity and the verification of our SOS with the use cases that were implemented throughout the semester.

Table of Contents

1 Introduction

The following chapter introduces the Final Systems Document (FSD) with the goal of explaining how certain qualities of the Student Organization System (SOS) project were implemented, tested and evaluated for launch of the site.

The purpose of the FSD is to give an overview of the analysis, design, deployment, development and test model of the current SOS system. In fact, this document states the functionalities that work within the SOS and others that do not function properly. The analysis model describes the requirements, compiled in the form of use cases of the SOS system, as well as the interactions between potential users and the system. The design model describes the decomposition of the system and object design of the SOS to create both a stable and efficient architecture. The deployment model describes how the SOS system will be deployed and what are the nonfunctional requirements that are required to access our site. The development model describes the decisions made during the development of the SOS and describes bugs and constraints the team faced. Finally, the test model describes the series of test cases performed on our system to validate and verify that the SOS is functional and follows both the SRD and DD requirements and constraints.

The purpose of the SOS is defined below. Following that, the scope of the system is defined in Section 1.2. Section 1.3 contains a list of relevant terms, acronyms, definitions and abbreviations used throughout the system. Finally, Section 1.4 contains a brief outline of this document. Following chapters include a detailed section on proposed software architecture (Section 2) and a detailed section on the design of SOS (Section 3).

1.1 Purpose of the System

The Student Organization System (SOS) is a web-based system meant to provide leaders and administrators of organizations a fast, interactive, and accessible way to manage members and events from a single, centralized place. Simultaneously, the SOS system also allow users to monitor and keep up-to-date information about the events and requirements of the organizations they belong to. Finally, the system also allows organizers to advertising their organizations and recruit new members from the general userbase. In essence, the Student Organization System is meant to aid the interaction between members and organizations.

Although the system is meant primarily for academic settings, with Universities being the main target, organization creation and management is open and could be used in other environments, both academic (High Schools, etc.) and non-academic (Company Campuses, Community Centers, etc.).

1.2 Scope of the System.

The managerial side of the Student Organization System (SOS) allows organizers to administer their organizations by in four core ways:

i. It provides a single point of access for users, members and non-members alike;
ii. It organizes all the members of the club under a single network;
iii. It provides tools to manage organization leaders and their privileges; and
iv. It provides means to create and advertise organization-related events;

The system also allows users to interact with the following ways:

i. By collating all the organizations that they belong (or might interested in) to in a single place.
ii. By presenting them with events in their surrounding hosted by their organizations or any other public events.
iii. By connecting them with new organizations and other members.

The event system specially is at the core of both the managerial and the user side of the system. Events postings are created by organizers to promote their organizations and are attended by users. Events are geo-tagged and presented primarily by their location and time. Moreover, the system also provides attendance tracking and RSVP functionalities, which are integrated into a point-ranking system that organizers might choose to enable for their organizations in order to foster member participation.

The following functionalities, although related to organization management and user communities, are outside of the scope of the SOS and are not covered by this document:

i. Integration with social media sites (e.g., Facebook)
ii. Social media features such as comments and postings.
iii. User-defined (as contrasted to organization-defined) events.
iv. Detailed organization leader management features (e.g., payments, duties, etc.)
v. Detailed organization tasks and projects management systems.
vi. Organization-relation features (e.g., community chapters)

A future version of the SOS might include some of the features.

1.3 Development Methodology

The development of the Student Organization System (SOS) follows the Unified Software Development Process (USDP; Jacobson, Booch, & Rumbaugh, 1999). The USDP can be seen as defined by a set of interconnected models: (a) use case model, (b) analysis model, (c) design model, (d) deployment model, (e) implementation model, and (f) test model. Their relationships can be seen in Figure 1: The relationships between the models in the Unified Software Development Process (USDP).Figure 1.

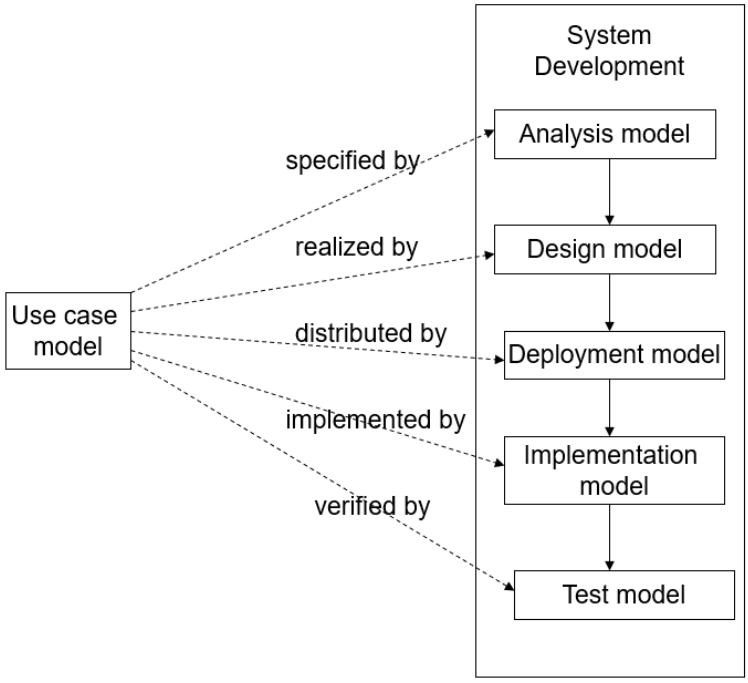

Figure 1: The relationships between the models in the Unified Software Development Process (USDP).

This document contains the all the models seen in Figure 1. The analysis model is described in Section 4.3 an contains a more structured and formal description all the possible interactions between users and the system for a particular piece of functionality. The design model gives a more detailed view of the system in the form of a set of interconnected subsystems, each containing classes and performing a discrete action. Sections 6 contain an overview of these subsystems in the form of a top-level UML Package Diagram. A more descriptive view of the UML Class Diagrams for the subsystem decompostion are found in Appendix D. A simplified version of the implementation model, is also presented in this document, in Section 5, Hardware and Software Mapping, which contains a UML Deployment Diagram of the SOS. The design and deployment models should provide a detailed description of the system structure without relying on implementation details and which could be ported to any desired platform with sufficient functionalities. The implementation model is found in Appendix E and Appendix C. Appendix E is the code for the subsystems that were implemented while Appendix C has the User Interface designs implemented for the SOS system. The test model description is found in Section 7, where all the test cases for the features in the SOS are found as well as the evalutations of said tests.

1.4 Definitions, Acronyms, and Abbreviations

Table 1: Definitions, Acronyms, and Abbreviation, contains a series of terms and acronyms used through this document. A more elaborate glossary can also be found in Section 6 of this document.

Term	***Meaning***
3TA	Three-Tier Architecture
API	Application Programming Interface
DB	Data Base (Data Storage)
DD	Design Document
FIU	Florida International University
FSD	Final Systems Document
N/A	Not Applicable
SOS	Student Organization System
SRD	Software Requirements Document
UML	Unified Modeling Language
USDP	Unified Software Design Process
V&V	Validation & Verification

Table 1: Definitions, Acronyms, and Abbreviation

1.5 Overview of the Document

This document is structured into chapters, each describes a different model of the SOS. Chapter 2 describes the limitations and problems of the SOS. Chapter 3 discusses the overall project plan for the SOS. Chapter 4 describes the requirements of the system including the use case model and the analysis model used in the SOS. Chapter 5 discusses the software architecture concepts used to implement the SOS system. The software architecture involves the decomposition of the system into subsystems, hardware and software mapping, persistent data management, and security management. Chapter 6 wraps up all parts of detailed design including several different UML diagrams (package, class, object, state, and sequence diagrams) for every subsystem derived from the system decomposition. Chapter 7 discusses the test model that were used in the SOS to validate and verify the system requirements and constraints. Chapter 8 goes through the glossary which defines the domain-specific keywords and terms used in this document. Chapter 9 is the approval page of the document which contains all of the SOS team members' signatures. Chapter 10 includes the references used in the document.

Finally, Chapter 11 contains seven different appendixes (A-G). Appendix A includes the project's schedule, Appendix B includes all the use cases that were generated for the SOS, Appendix C conatins all the user interface designs for SOS, Appendix D contains detailed subsystem class diagrams (total of 7), Appendix E describes the Java code for the subsystem classes, Appendix F contains the documented code for the test driver of the SOS, Appendix G contains diaries for every meeting held during this all the deliverable milestones for the semester.

2 Current System

Although the Student Organization System (SOS) shares some functionalities with social media sites such as Facebook (which can create group pages to which individual profiles can subscribe to), it is most like existing student organization systems that are hosted and managed by Universities and Colleges.

An example of this system is Panther Connect (Campus Lab, 2019) which is hosted by Florida International University (FIU). Panther Connect is a website where FIU students can search for clubs and organizations to join. It also allows them to keep track off and attend events happening on the FIU campus and sign up for volunteering activities. Event organizers can keep track of their member's involvement and stay in contact with them, as well as send invitations to new users. The platform also allows to create events that members can RSVP to.

Because of the relationship the system has with FIU, it provides another functionality which the SOS system does not provide: forms for campus and institutional requests. These are accessible through the "Forms" option in the homepage. However, privacy is a problem with this system. Whenever an organization creates a form, it is automatically added to the list where any user can view it. They do not need to have an account or a membership with the organization in order to access it.

Membership management has a couple of functionality issues. This is related to the fact that the platform is event-centered rather management-centered. For management, the platform provides the option to end a member's membership. However, it doesn't provide the functionality of selecting multiple members at once (except for an "End All Memberships" button, but it is not needed most of the time). If a user needs to end a decent number of memberships, they must select each member one by one. It also allows users to invite others through email, but it can only invite 500 people at a time. There is also a way for users to send mass emails to their rosters. Unfortunately, its mail server can be unreliable at times and may sometimes take up to 3 hours to send an email.

Most other communities (non-academic) have organization management systems but these are either offline, with a primary focus on management, or they are manual (e.g., community centers sharing events through newsletters or posters).

3 Project Plan

The following sections describes the how the project is structured, which includes managerial information, constraints on the system, and a schedule of activities. Section 3.1 describes the organization of the project, which includes information about the members, the established communication mechanisms, the schedule of meetings, and the assigned roles. Section 3.2 contains the hardware and software constrains of the system. Finally, Section 3.3 contains the schedule table for the project.

3.1 Roles

This section contains the roles used in the project and who they are assigned too. Each member has several roles assigned to them at the same time. Each member has one or more project roles assigned:

Member Name	Roles
Armando J. Ochoa	System Architect, Minute Taker
Anthony Sanchez-Ayra	Database Administrator, Primary Facilitator
M. Kian Maroofi	Front-End Developer, Team Leader
Yovani Jones	Tester, Time Keeper
Teriq Douglas	Object Designer, Accountant

Table 2: The roles assigned to the team members.

3.2 Hardware and Software Requirements

The hardware and software materials needed to complete the project are captured in the following subsections.

3.2.1 Hardware Requirements

The testing environment is a network-enabled computer system with the following hardware requirements:

- Processor: Intel (R) Core (TM) i7-7700 CPU @ 3.60GHz
- Installed Memory (RAM): 16GB DDR4 SDRAM
- Storage: 512GB
- Network Adapter: Inter (R) Ethernet Connection (2) I219-LM

Each member has its own individual station. The details of these stations are not reported in this document.

3.2.2 Software Requirements

The testing environment has the following software applications:

- MySQL 8.0, which is used for a back-end data store server.
- Java JDK 1.8.0_221-b11, with the following external libraries:
  - *netty-socketio*, a java implementation of *socket.io* used for front-end/back-end communication.
- Node.JS version 10.16.3 LTS, with the following external libraries:
  - *React*, which is used to create the front-end.

- *Redux*, which is used for state management of the front-end.
- *Router*, which is used to handle front-end navigation.
- *Google Maps API*, which is used to obtain and display location information.

3.3 Project Schedule Table

The project is divided into several tasks, which are collected in Section 3.3.1. These tasks build towards the following deliverables:

- Deliverable 1, Software Requirements Document (SRD; this document), which has the following milestones:
  - M1, Section 4, Requirements Elicitation, and Section 5, Requirements Analysis, are completed.
  - M2, Software Requirements Document is completed.
- Deliverable 2, Design Document (DD), which has the following milestones:
  - M3, Section 2, Proposed Software Architecture, and Section 3, Detailed Design, are completed
  - M4, Design Document is completed.
- Deliverable 3, Final Systems Document (FSD), which has the following milestones:
  - M5, System Implementation is Completed.
  - M6, Section 7, Testing Process, and Appendix F, Document code for Test Driver are completed.
  - M7, Final System Document is completed.

3.3.1 Task Schedule

Task	Group	Description	Duration (Days)	Dependencies
T1		Set up Communication Methods: Weekly Meeting, WhatsApp Group, and GitHub.	1	
T2	Requirements Elicitation	Identify Use Cases for the Complete System	15	
T3	Requirements Elicitation	Decide on 10 Implementation Use Cases	1	T2
T4	Requirements Elicitation	Create Use Case Diagrams	6	T2
T5	Requirements Analysis	Create Sequence Diagrams and Class Diagram for 10 Implementation Use Cases	5	T3
T6	Requirements Analysis	Create Scenarios and Object Diagram for Use Cases	5	T3
M1		Write up SRD Sections 4 and 5	1	T1, T4, T5, T6
T7		Write up SRD Sections 1-3, 6-9	22	T1
M2		Compile SRD	2	M1, T7
T8	Development	Set up Project Environments	1	T1
T9	Development	Create React Mock-up	23	T8
M3		SRD Presentation	3	M2, T9
T10	Detailed Design	Decide on the Subsystem, Data Management.	5	M2
T11	Detailed Design	Create Detailed Class Design for the Subsystems.	8	T10
M4		Write up DD Section 2 and 3	12	T11
T12		Write up DD Sections 1, 4-7	25	M2
M5		Compile DD	6	M4, T12
T13	Development	Implement Front-End Subsystems	15	M3, T10
T14	Development	Implement Back-End Logic Subsystems	15	T10
T15	Development	Implement Back-End Datastore Subsystems	15	T10
M6	Development	Integrate the Subsystem. The Implementation is Completed.	3	T13, T14, T5
T16		Update Sections from DD and SRD to their FSD version.	3	M5, M6
T17	Testing	Set up Testing Environment and Formulate Test Cases	5	M6
T18	Testing	Perform Testing Process	5	T17
M7		Write up FSD Section 7 and Appendix F	5	T18
T19	Development	System Verification	3	T18
T20		Write up FSD Sect. 4-6, Appendix A-E, G	5	T19
M8		Complete FSD	4	T16, M7, T20
M9		FSD Presentation	3	M8

Table 3: Task Schedule for the Project

4 Requirements of the System

4.1 Functional and Nonfunctional Requirements

The following subsections define the functional and non-functional requirements of the SOS system.

4.1.1 Functional Requirements

Below is a short description of the functional requirements of the SOS system for each of the implemented Use Cases. The complete use cases for each can be found in Appendix B. The corresponding UI design for the implemented Use Cases can be seen in Appendix C.

- The system shall allow an organizer to create events for their organizations (see Use Case SOS01 in Appendix B).
- The system shall allow the current organizer to add/invite other members of the organization to be granted with the organizer role (see Use Case SOS02 in Appendix B).
- The system shall allow a member of an organization to earn points for attending an event (see User Case SOS03 in Appendix B).
- The system shall allow users to check-in for each event on the platform (see Use Case SOS04 in Appendix B).
- The system shall allow users to have privileged access to an event depending on the status and privileges of a user (see Use Case SOS05 in Appendix B).
- The system shall allow users the have profile privacy (see Use Case SOS06 in Appendix B).
- The system shall allow users to edit their profile data including their email, phone number, date of birth, password, and privacy features (see Use Case SOS07 in Appendix B).
- The system shall allow users to share events or organizations to other members using social media (see Use Case SOS08).
- The system shall allow members to see their rank in a certain organization based on the points they get by attending events (see Use Case SOS09).
- The system shall allow users to find all nearby events based on the user's current location (see Use Case SOS10 in Appendix B).
- The system shall allow organizers to add a certain amount of points to any event that they create (see Use Case SOS11 in Appendix B).
- The system shall allow the users to set up two factor authentication within the (see Use Case SOS12 in Appendix B).
- The system shall allow an organizer to have and/or grant kick privileges to other organizers (see Use Case SOS13 in Appendix B).
- The system shall allow an organizer to create new roles to delegate management of their organization to other members (see Use Case SOS14 in Appendix B).
- The system shall allow a member to enable or disable notifications (see Use Case SOS15 in Appendix B).
- The system shall allow users to create their own organization (see Use Case SOS16 in Appendix B).

- The system shall allow the organizer to cancel the event (see Use Case SOS17 in Appendix B).
- The system shall allow the organizer to create a task for a certain event that they created (see Use Case SOS18 in Appendix B).
- The system shall allow the organizer to request organization details from the SOS (see Use Case SOS19 in Appendix B).
- The system shall allow the organizer to disband organizations they no longer want to lead (see Use Case SOS20 in Appendix B).
- The system shall allow the organizer to avoid time conflicts when scheduling different events (see Use Case SOS21 in Appendix B).
- The system shall allow visitors to register for a new account (see Use Case SOS22 in Appendix B).
- The system shall allow administrators to delete inappropriate events (see Use Case SOS23 in Appendix B).
- The system shall allow administrators to extend their privileges to other members of the SOS (see Use Case SOS24 in Appendix B).
- The system shall allow users to filter events (see Use Case SOS25 in Appendix B).
- The system shall allow organizers to invite users to an organization through a roster (see Use Case SOS26 in Appendix B).
- The system shall allow organizers to remove users from an organization through a roster (see Use Case SOS27 in Appendix B).
- The system shall allow users to RSVP to events (see Use Case SOS28 in Appendix B).
- The system shall allow users to confide in the access management security of the SOS (see Use Case SOS29 in Appendix B).
- The system shall allow users to only create events for organizations that they have been granted privileges for (see Use Case SOS30 in Appendix B).
- The system shall allow users to login to their registered account (see Use Case in SOS31 in appendix B).
- The system shall allow users who are already logged-in to logout from the system (see Use Case SOS32 in appendix B).

4.1.2 Non-Functional Requirements

Below is a summary of the non-functional requirements of the SOS system. The expected requirements for each Use Case have been collated into general system-wide requirements. A more detailed description of the non-functional requirements is in each use case in Appendix B.

4.1.2.1 Usability

In general, no training or special knowledge is required to use any of the implemented functionalities. For each user, a tutorial or help frame should be provided to guide new users. Users should take at most 10 minutes to find and use each of the functionalities provided by SOS.

4.1.2.2 Reliability

In general, a mean time to failure between 1 and 5% monthly is acceptable. Availability is affected by two downtimes, one for login back up, 30 minutes every 24-hour period, and another for maintenance, 1 hour in a 2 weeks period.

4.1.2.3 Performance

Privilege checks should be done within 2 seconds. The system should be able to handle 20 privilege checks in 1 minute. Each individual form and request should be sent, processed, and saved within at most 10 seconds. The system should be able to handle around 20 and 50 requests per minute.

4.1.2.4 Supportability

The whole system is supported by Chrome, Mozilla, and IE desktop and mobile browsers.

4.1.2.5 Implementation

The whole system is implemented using JS React for the front-end and Java-based software for the backend.

4.2 Use Case Diagram

The Use Case diagram describing the whole system is shown in Figure 2 to Figure 9. The actors participating in the system are:

- User – any individual using the website, including ones without a registered account.
- Member – any user with a registered account who belongs to an organization.
- Organizer – any member of an organization with leadership and/or administrative privileges on that organization.
- Admin – a privilege user with system-wide powers and access

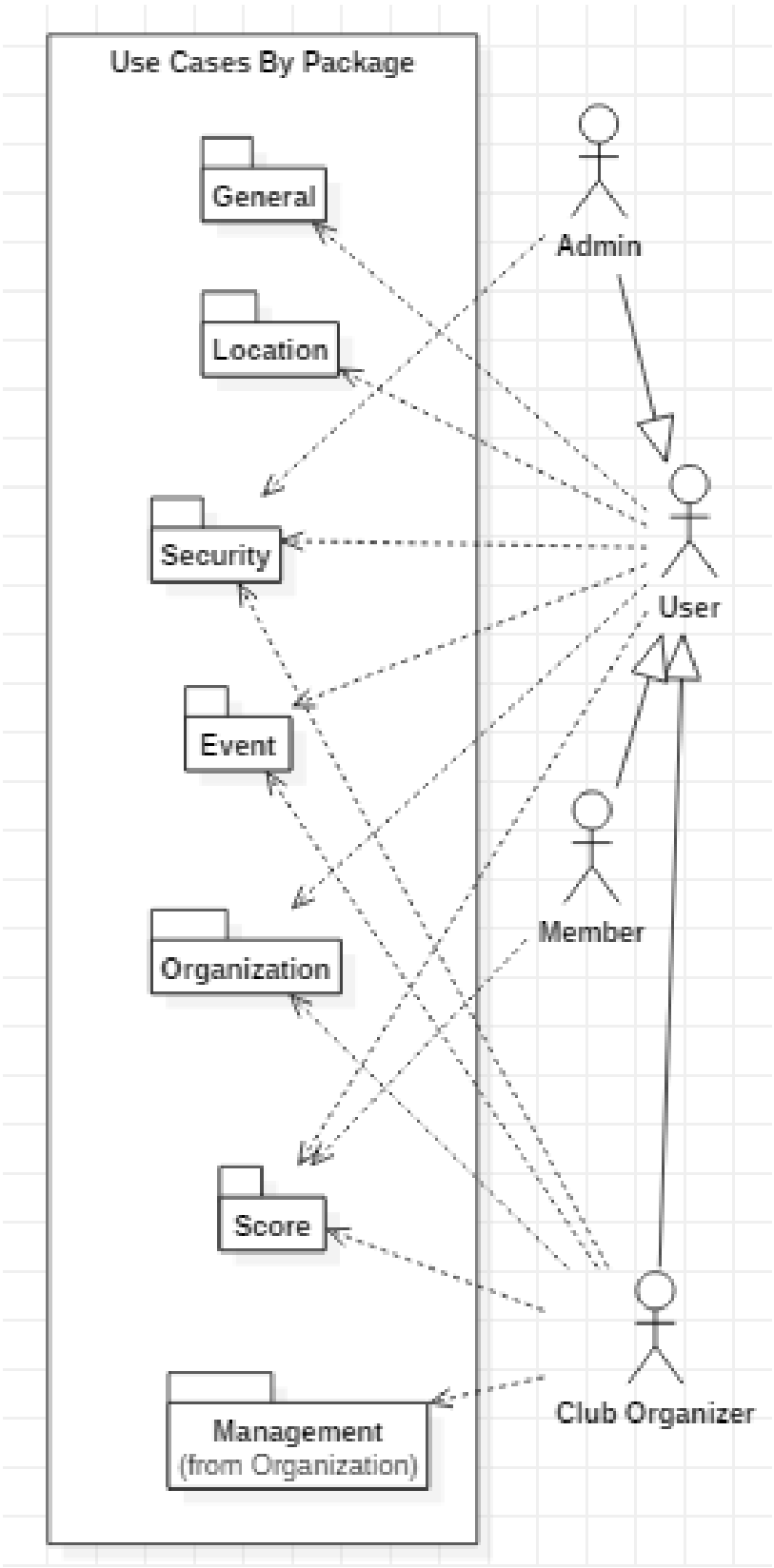

Figure 2: Use Case diagram for the whole system.

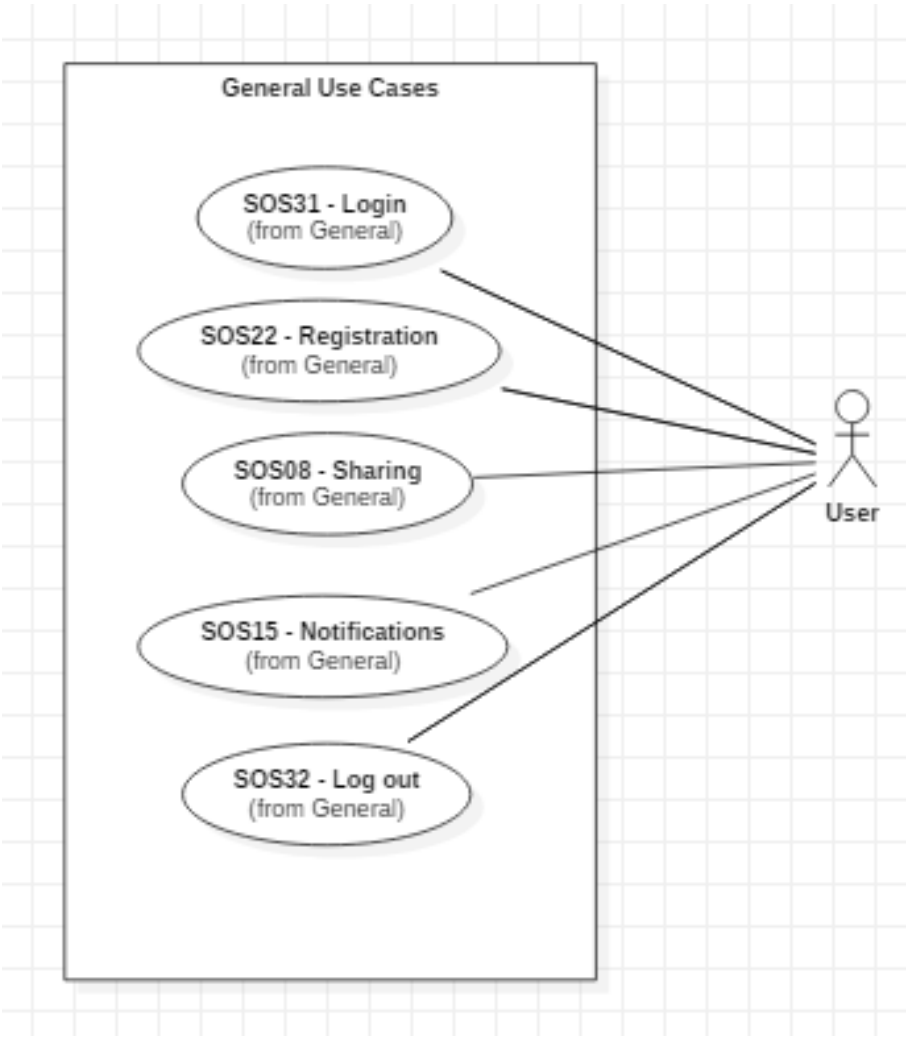

Figure 3: General Use Cases, all of which represent use cases that affects all users of the system.

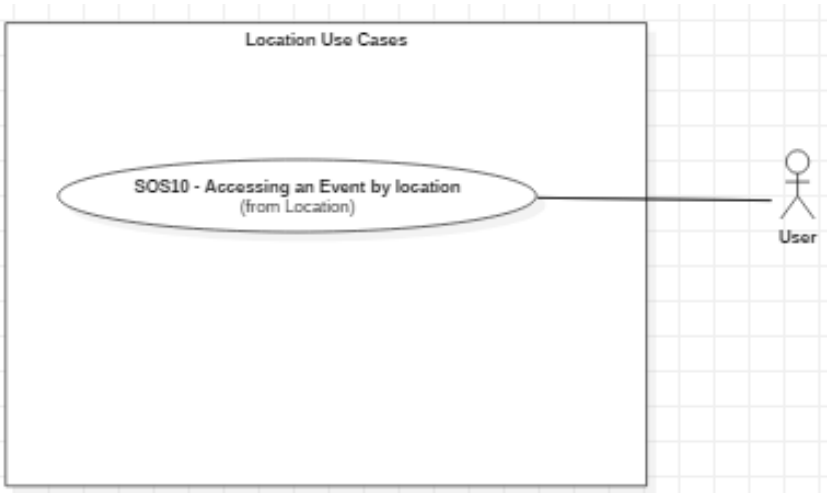

Figure 4: Location Use Case Diagram, which collects the use cases having to do with the external geolocation API.

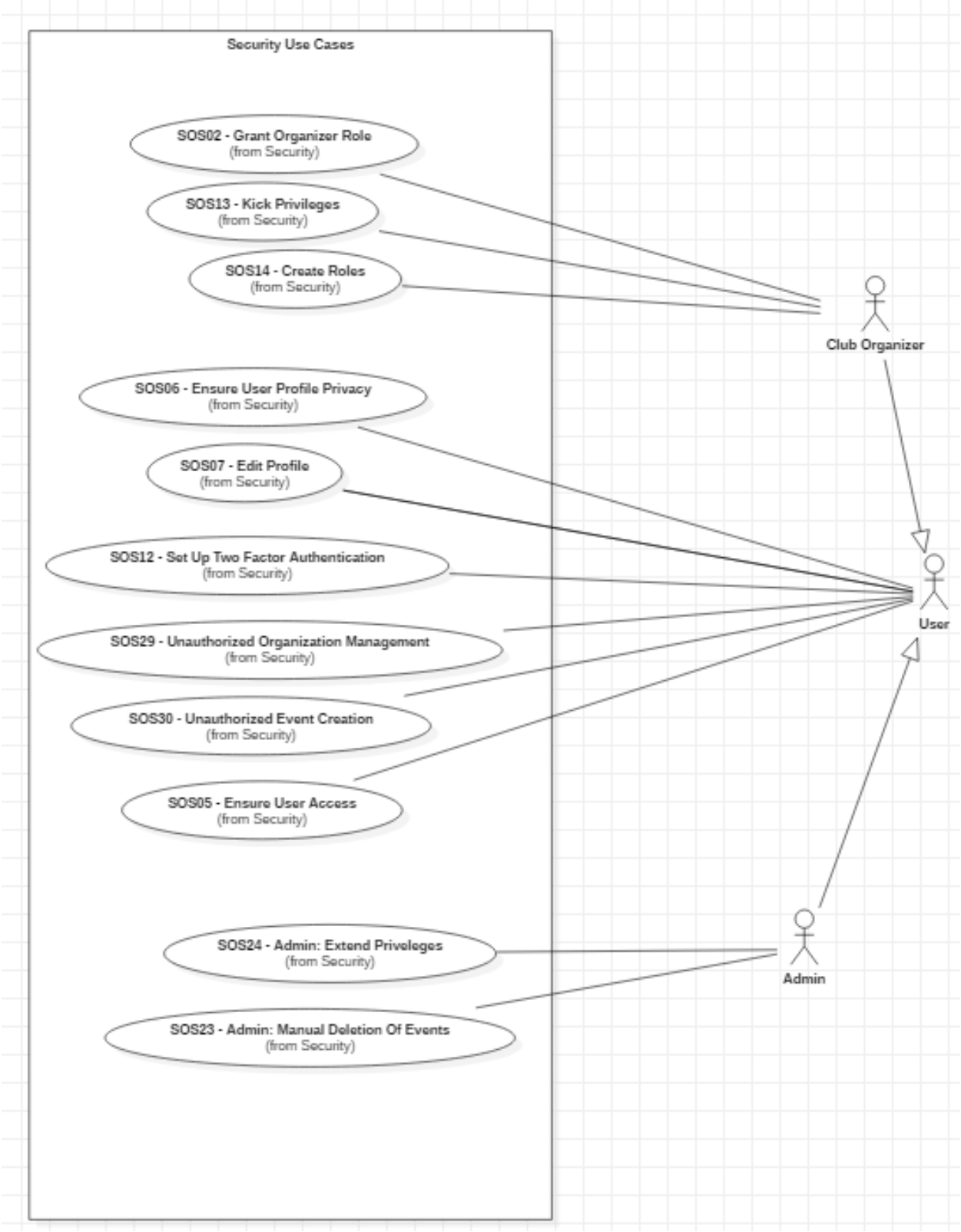

Figure 5: Security Use Cases, which collects all the use cases having to do with security.

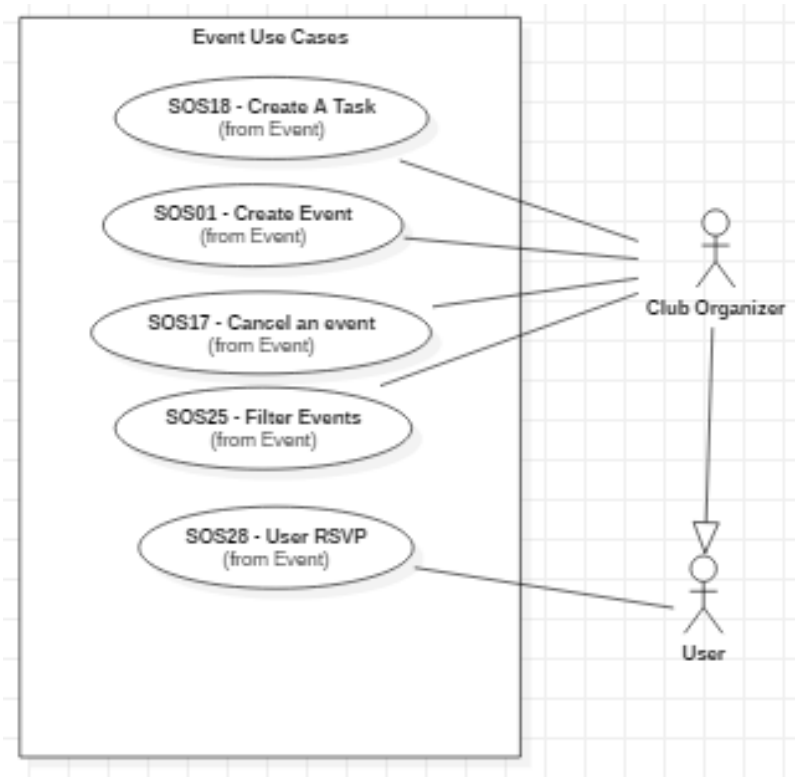

Figure 6: Event Use Case Diagram, which collects all the use cases related to event creation, destruction, etc.

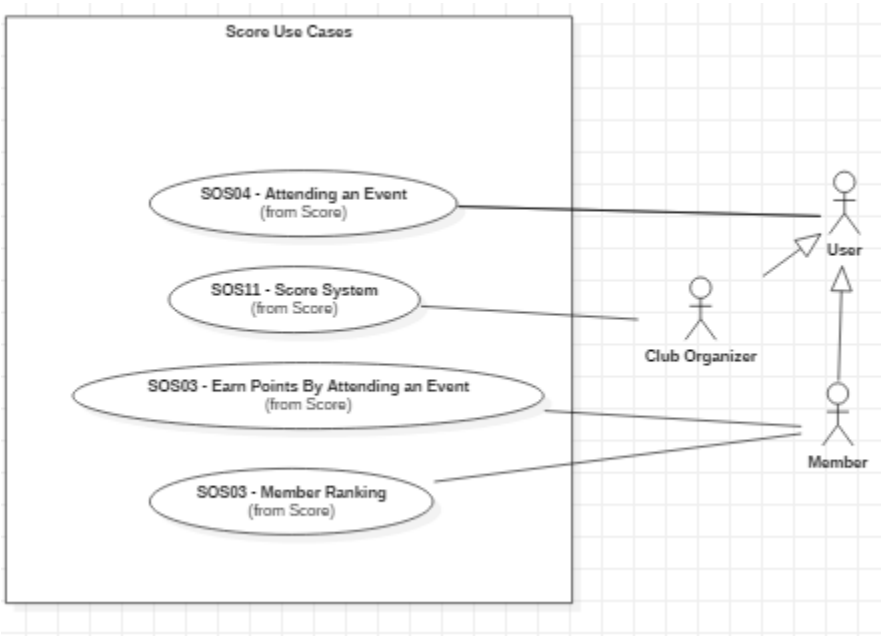

Figure 7: Score Use Case Diagram, which collects all the use cases having to do with the ranking and point system.

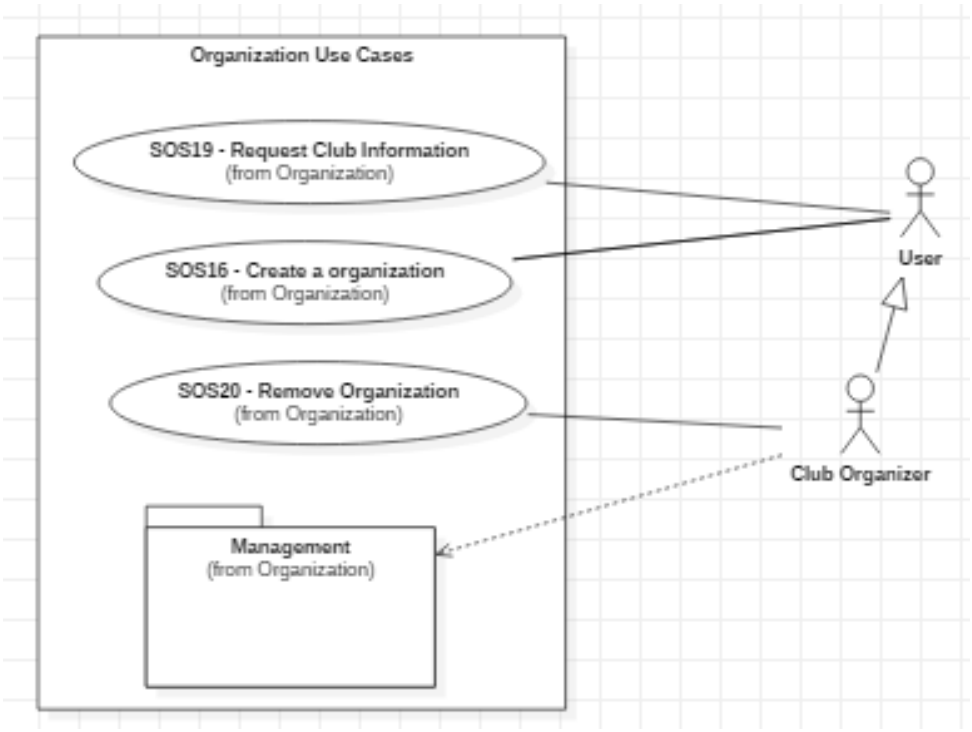

Figure 8: Organization Use Case Diagram, which collects all the use cases having to do with organizations.

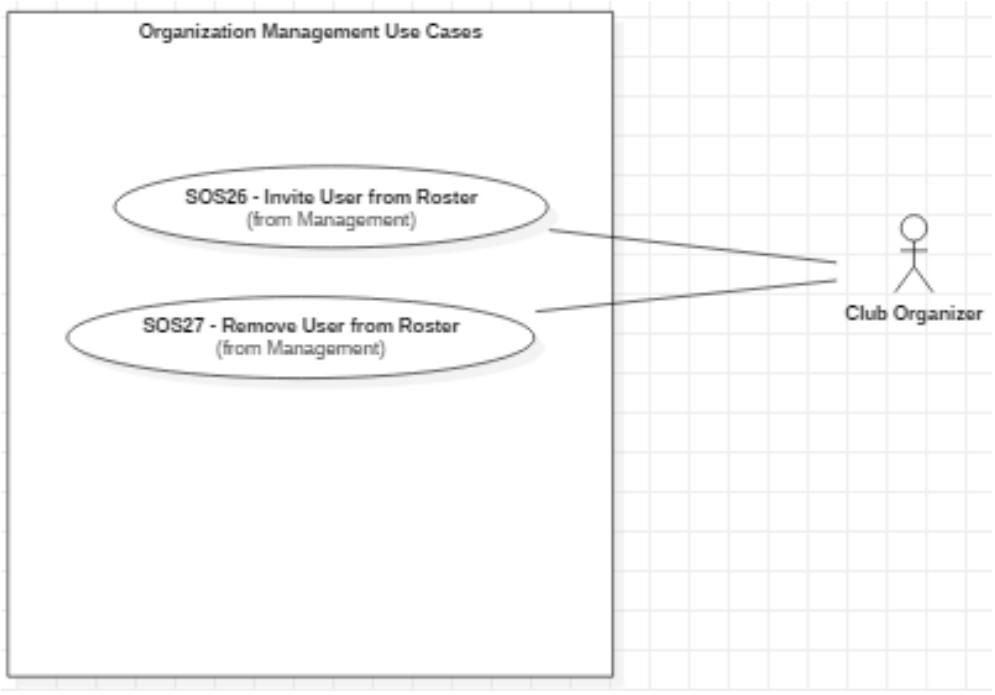

Figure 9: Organization Management Use Cases, which collects all the use cases having to do with organization management and is a subset of the Organization Use Cases.

4.3 Requirements Analysis

In the requirement analysis for the SOS the team developed both a static model and dynamic model to realize the use cases that were described in the earlier sections. In the static models we created an SOS class diagram which described the different classes, along with their operations and attributes, needed to realize the requirements for the use case models of our system. These class diagrams are found in the SOS SRD which was turned in earlier in the semester. Another major part of this static model were the object diagrams that we had created by generating scenarios for the SOS. In fact, many of these scenarios were used as the basis for the test cases described later in our test model. In the dynamic model of the analysis model we created sequence diagrams to show the flow of actions, functions and methods of communications needed to complete the use cases in our site. Both the dynamic and static models' charts are found in the STAR UML file and the SRD that was turned in earlier this semester. All in all, it can be said that the analysis model, contains a more structured and formal description of these use cases using Unified Modeling Language (UML) class, sequence, and object diagrams. Between the use case and the analysis model they provide a consice description of the system and its functionalities.

5 Software Architecture

The following sections contain a top-level description of the architecture of the Student Organization System (SOS), including subsystems decomposition, as well as data management and security requirements. Section 5.1 contains a general overview of the system, including a general description of the architectural patterns used. Following that, Section 5.2 contains a subsystem decomposition for the SOS. Section 5.3 contains a UML Deployment Diagram showing the hardware and software mapping expected for the system. Section 5.4 contains the requirements and schema used for persistent data in the system. Finally, Section 5.5 contains the security requirements and schema for the system.

5.1 Overview

The SOS system is implemented using a three-tier architecture (3TA). In a 3TA, systems components are divided along three layers: (a) an interface layer, which includes the objects that interact with the user, in the SOS's case, a front-end Website; (b) an application logic layer, which includes the control and entity objects implementing the system's logic, in the SOS's case, a back-end Java server; and (c) a storage layer, which contains, maintains, and retrieves the persistent objects found in SOS. The 3TA was chosen because it allows the SOS system to be divided into interchangeable layers which can be updated and maintained separately if their separate interfaces are maintained. Moreover, it allows each of the layers to be hosted in different systems, which matches the desired deployment structure of a front-end client, a back-end system, and a separated storage system (see Section 2.3 for a full deployment description). In addition, 3TA has superior performance for medium-to-high volume environment, which matches the expected volume that the SOS system would experience if deployed in its target environment (universities and other similar closed communities). The SOS system subdivides its structure into more than three subsystems, but these are grouped into each of the three layers of a 3TA. This mapping is presented in the following section, Section 2.2.

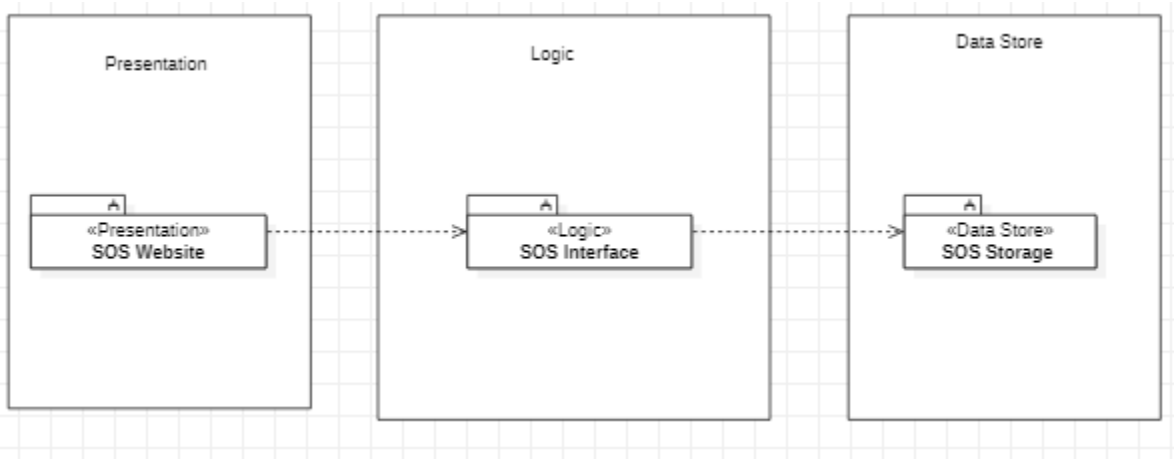

Figure 10: Class Diagram showing the 3TA of the Student Organization System.

Besides the 3TA, the SOS system also implements a repository architecture. In a repository architecture, several subsystems access and modify data from a single data structure (a repository) which mediates their interaction. This architecture is used in the storage layer as shown in Figure 11. Because our primary architecture is 3TA, most of the subsystem interaction is not mediated by the repository, but instead by within-layer connections. However, some subsystems do interact with the repository when calling functions of the storage facade within the storage layer. This architecture was chosen because it serves as an efficient way to store a large amount of data and retrieve it from a single monolithic source. Moreover, it reduces the overheard of a transient data between software components.

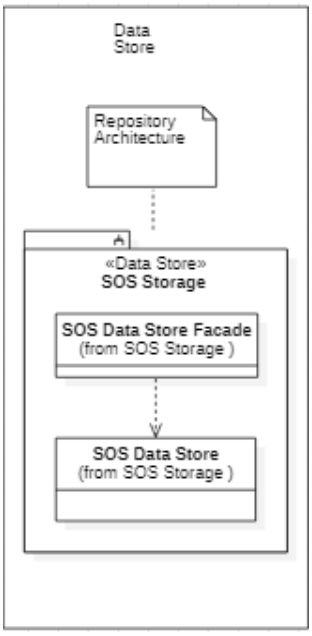

Figure 11: Class Diagram showing the Repository architecture used in the Student Organization System.

The combination of these two architectures was chosen to meet the standards and expectations of the non-functional requirements of both performance and reliability, since both architectures ensure that the system will be responsive and quick to handle requires.

5.2 Subsystem Decomposition.

The following subsystems compose the Study Organization System:

- SOS Storage, which will act as a central node in the repository architecture where persistent data is stored, maintained, and retrieved. It alone is part of the storage layer in the 3TA. All use cases that interact with the data store use this subsystem.
- SOS Website, which represents the interface layer of the 3TA. It contains the objects which will present the SOS site that acts as the user interface. This will be done on each user's browser (front-end). All use cases will by default use this subsystem.
- SOS Interface, which acts as the server of the application which processes requests from the SOS website and create solution objects of the other subsystems that will resolve those requests and interact with the data store. This subsystem is a core part of the application logic layer of the 3TA. All use cases will by default use this subsystem.
- User Management, which contains all the system functions relating to Users, such as Registration (SOS22), Edit Profile (SOS07), and User Roles (SOS02, Grant Organizer Role). This subsystem is part of the application logic layer of the 3TA.
- Event Management, which contains all the system functions relating to events, such as Event Creation (SOS01), Attending Events (SOS04), Accessing Events by Location (SOS10), and Canceling Events (SOS17). This subsystem is part of the application logic layer of the 3TA.
- Organization Management, which contains all the system functions relating to Organizations, such as Granting Organizer Roles (SOS02), and Creating an Organization (SOS16). This subsystem is a part of the application logic layer of the 3TA.
- Security Management, which contains all the security-related functions, which mostly include password management and access control. These functions relate to User Roles (SOS02), Editing Profile Access (SOS07), Registration (SOS22), Login In (SOS31) and Out (SOS32). This subsystem is a part of the application logic layer of the 3TA.
- Google Maps GPS API, which represents an external API responsible for retrieving location coordinates for Events and Users. This is used for Creating Events (SOS01) and Accessing Events by Location (SOS10).
- Utilities, which represent helper functions, constants and enumerations that are used to promote the reuse of code.

5.3 Hardware and Software Mapping

The hardware and software mapping for the SOS system can be seen in the UML Deployment diagram in Figure 12. The system uses three nodes, one web or mobile node for the client (front-end), a dedicated server for the SOS logic-layer (back-end) and a third dedicated server for the SOS data store layer. Alternatively, the two back-end layers could be unified into a single node.

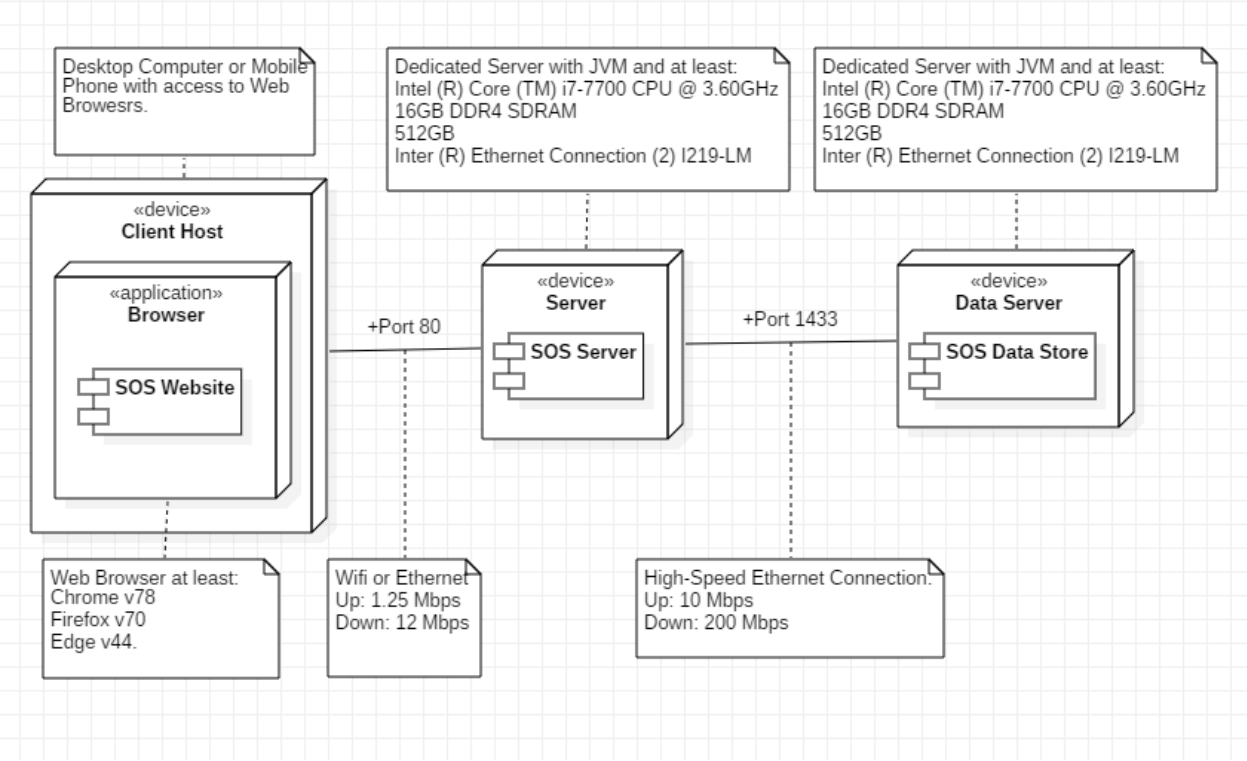

Figure 12: UML Deployment Diagram for the SOS system.

The client should be on a desktop computer or a mobile phone with access to the web browses that SOS can run on which are Chrome version 78, Firefox version 70 and Edge version 44. To access the SOS server the user needs at least 1.25 Mbps upstream and 12 Mbps downstream to ensure fast and reliable connectivity. The SOS server will run on its own computer and it will have JVM installed in it. The specifications of the computer where the server will be hosted in are Intel ® Core ™ i7-7700 CPU @ 3.60 GHz 16 GB DDR4 SDRAM 512 GB Inter ® Ethernet Connection (2) I219-LM. The server will communicate to the data server with a high speed ethernet connection of 10 Mbps upstream data and 200 Mbps downstream data. The data server will run on its own device with similar specifications to the server computer. In addition, the Data Server needs to have MySQL server (version 8.0.18) and a Connector/J (version 8.0.18) to communicate and receive requests from the Java ran back-end.

5.4 Persistent Data Management

The persistent entities for the SOS system, as well as the connections between them, are represented in the UML ER diagram in Figure 13.

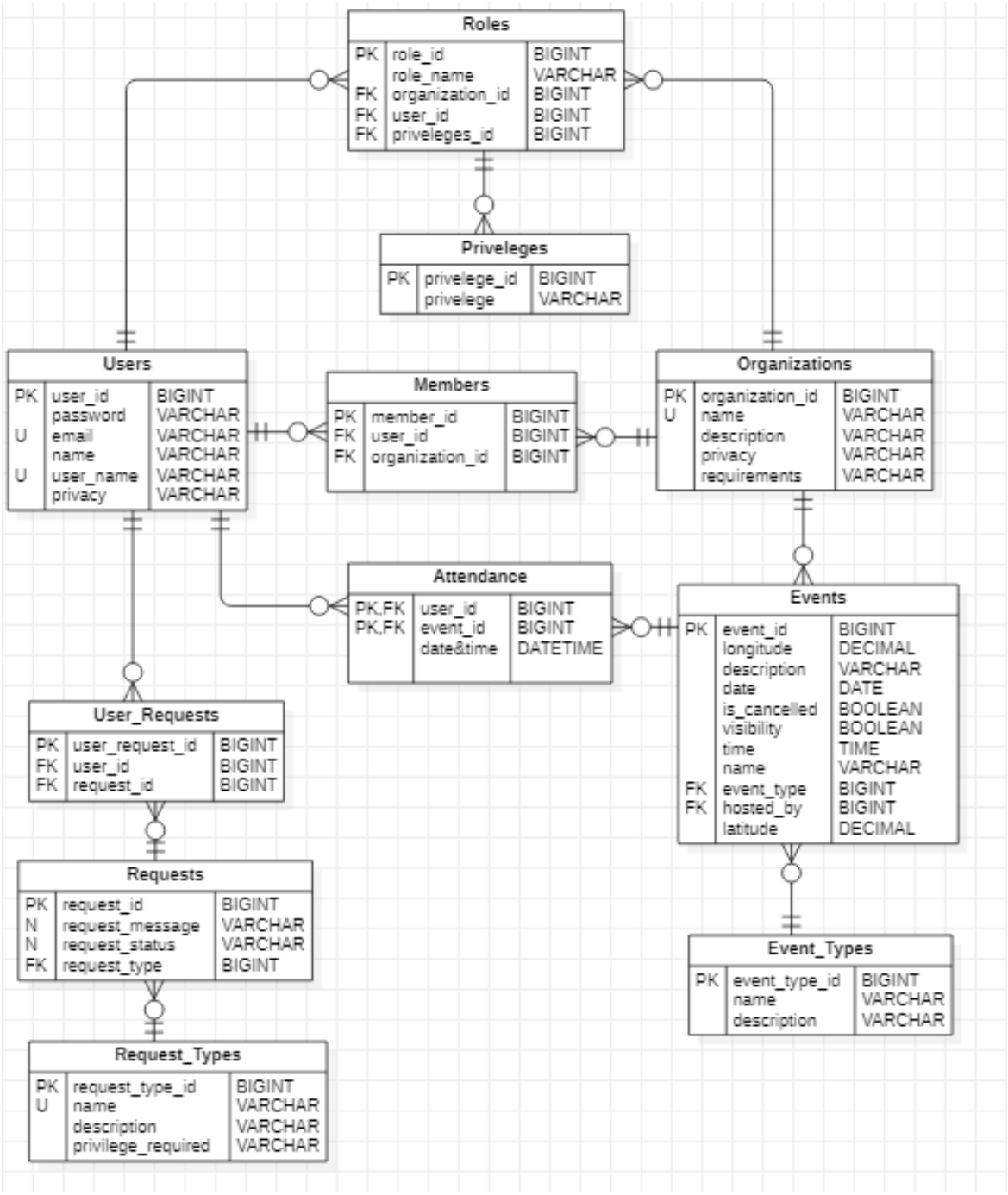

Figure 13: UML ER Diagram for the SOS System.

The diagram observers Third-Normal Form. The SOS system has the following tables:

- *Users*, which represent the user-defined accounts on the system. Users can be Members of Organizations, they can also have Roles (e.g., Organizer) in Organizations, they can attend Events, and they can make Requests on the system.

- *Members*, which is a link between Users and Organizations. A Member (which is an Actor in our system) is a User that belongs to an Organization.
- *Organizations*, which represent groups of Users in the system. Organizations have Members, which are Users that may or may not have privileges, and Organizers, which are Users which have Roles, with privileges. Organizations can host Events.
- *Roles*, which represents a set of privileges a User has in an Organization. A Role defines an Organizer (which is an Actor in our system).
- *Privileges*, which is a right a User might have with respect to an Organization. There are a set number or Privileges, which includes Create Event, Invite Users, Delete Organization.
- *Events*, which represent real-life activities. Events are associated with their hosting Organization and can be Attended by Users. Events also have Types.
- *Event_Types*, which represent types of Events. There is a set number of accepted Types.
- *Attendance*, which is a link between Users and Events.
- *Request*, which is a request on the system by a User. They are kept for housekeeping and maintenance purpose.
- *User_Requests*, which is a link between Users and Requests.
- *Request_Types*, which represent types of Requests. There is a set number of accepted Types.

The descriptions for each field in Figure 13, as well as field size, data type, and format, can be seen in the data dictionary in Table 4.

entity name:	field name	field size	data type	Data Format	Example	Description
user	name	20 chars	string	----------	Rick Sanchez	First name of the user.
	user_name	20 chars	string	----------	TiredgeSnius68	The user's username.
	email	20 chars	string	username@domainname	Szechuan808@hotmail.com	The email address of the user.
	password	20 chars	string	----------	3o9t23bf4180rf87b2387	The encrypted password of the user.
	user_id	------	int	----------	1	A unique ID for the User
	privacy	20 chars	string	PRIVATE PUBLIC	PRIVATE	The privacy setting of the account
member	member_id	------	int	----------	1	A unique ID for the relation.
	user_id	------	Int	----------	1	Identifies a User
	organization_id	------	Int	----------	1	Identifies an Org.

Organi-zation	organization_id	------	int	----------	1	Identifies an Org.
	name	100 chars	string	----------	Club SOS	The name of the Organization
	description	500 chars	string	----------	A club for club-goers	The description of the Org.
	privacy	20 chars	string	PRIVATE PUBLIC	PUBLIC	The privacy setting of the Org.
	requirements	500 chars	string	----------	Be a FIU Student	The requirements for joining.
Roles	role_id	------	Int	----------	1	Identifies a Role.
	role_name	20 chars	string	----------	Owner	A given name for the Role.
	organization_id	------	Int	----------	1	Identifies an Org.
	user_id	------	Int	----------	1	Identifies a User.
	priviledges_id	------	Int	----------	1	Identifies a Priv.
Privi-ledge	priviledge_id	------	Int	----------	1	Identifies a Priv.
	privilege	20 chars	String	CREATE READ UPDATE DESTROY …	CREATE	A defined and immutable set of privileges which organizers might have in their Orgs
event	event_id	------	Int	----------	1	Identifies a Event
	lonigtude	------	Doub le	----------	5.00	A longitude measure to determine the longitude of where an event will occur.
	latitude	------	Doub le	----------	23.44	A latitude measure to determine the longitude of where an event will occur.
	description	500 chars	String	----------	1st Meeting	The description of the Event.
	date	------	Date	MM/DD/YY	10/10/19	Date of the Event
	is_cancelled	------	Bool	----------	FALSE	True if the Event is cancelled.
	visibility	------	Bool	----------	TRUE	True if the Event is visible.
	time	------	Time	HH:MM TM	10:30 AM	Time of the Event

	name	100 chars	String	----------	SOS Meeting	Name of the Event.
	event_type	------	Int	----------	1	Type of the Event.
	hosted_by	------	Int	----------	1	Identifies the hosting Org.
Event Types	event_type_id		Int	----------	1	Identifies the Event Type.
	name	20 chars	String	----------	Meeting	The name of the event type.
	description	100 chars	String	----------	A get-together.	The description of the event type.
Atten-dance	user_id	------	Int	----------	1	Identifies a User
	event_id	------	Int	----------	1	Identifies a Event
	datetime	------	Date-time	----------	10/10/19, 10:30 AM	The date and time of the attendance.
Request	request_id	------	Int	----------	1	Identifies a request.
	request_messa-ge	500 chars	String	----------	Add General Meeting Event.	The message of the request.
	request_status	100 chars	String	----------	Valid	The resolution of the request.
	request_type	------	Int	----------	1	Identifies a Request Type.
User Request	user_request_id	------	Int	----------	1	Identifies a user-request link.
	user_id	------	Int	----------	1	Identifies a User.
	request_id	------	Int	----------	1	Identifies a Request.
Request Types	request_type_id	------	Int	----------	1	Identifies a Request Type.
	name	100 chars	String	----------	Add Event	The name for the type of request.
	description	500 chars	String	----------	Adds and Event from an Org.	The description for this type of request.
	privilege_req	100 chars	String	----------	Event Creation.	The required privilege for this request.

Table 4: Data Dictionary for the SOS Persistent Data.

5.5 Security Management

The SOS uses three core security mechanism, Password Management, which is described in Section 2.5.1, Access Management, which is described in Section 2.5.2, and Transfer Management. In the first and last cases, the relevant classes are implemented in the Security Subsystem (see Section 3.1.7). For, Access Management, the security is implementing ad-hoc on the front-end. Besides these functionalities, other systems are also used such as API Keys for the Google Maps GPS API and Encryption for network sharing of important data.

5.5.1 Password Management.

The goal of the Password Management policies is to ensure authenticity of the Users logged onto the SOS, and to ensure that changes issued by those User's accounts are committed by them and not by third parties who have gained access to their account. In order to do this, accounts must be locked behind passwords which only the real User should know. Hence, these passwords should be protected tightly by the SOS so no third-party gain access to them.

To ensure the safety of the password, the system encrypts it at the client side and shares it through the network encrypted. The encryption method used is public-private key encryption (RSA): when a session starts, the client receives a public key from the system, which it can use to encrypt the password. This ciphertext is then sent over the network to the back-end which decrypts it using its corresponding private key. In order to avoid storing real passwords in the back-end, the front end will hash the plaintext password with a salt value to create a unique hash. The hash and the salt will be sent to the backend and stored in leu of the actual password. To make things simple, the salt value will be the account's username.

5.5.2 Access Management

The goal of the Access Management policies is to ensure authorization of the actions that known Users are doing within the system, i.e., to ensure that Users can only do the actions that they can perform. In the SOS's case, the main actions involve exclusively creating, reading, updating, and destroying persistent data object such as Events, Organizations, and Users (i.e., Accounts). Because of this, a simple view of the access management policy can be represented using an Access Matrix on these objects, as is seen in Table 5.

		Data Types			
		Events	Organizations	Users Accounts	Roles
Actors	Member (Non-Owner)	R	R	R	R
	Organizer (Non-Owner)	R	R	R	R
	Member (Owner)	*Not Applicable*	*Not Applicable*	CRUD	*Not Applicable*
	Organizer (Owner)	CRUD	CRUD	CRUD	CRUD

Table 5: Access Matrix for the SOS System. Uses the CRUD mnemonics: Create, Read, Update, and Destroy. Note that Users cannot own Events, Organizations, or Roles, so the CRUD is not applicable to those relations.

The access policy, especially with regards to Organizations, is based on the notion of Privileges, which are specific permissions which Users have with regards to system functions. For example, a User might have a "Create Event" privilege in a given Organization, which lets them create new Event objects hosted by that Organization. Note that the distinction between our two actors, Members and Organizers is effectuated within our system exclusively by means of privileges: Members are Users linked to Organizations while Organizers are Members which also have Roles assigned to them which give them Privileges on that Organization.

5.5.3 Transfer Management

The goal of the Transfer Management policies is to ensure confidentiality while transferring data between the front-end and the back-end. This is managed by means of an encryption protocol which is shared by the two nodes. The workflow is as follows:

1. When the server first starts, it fetches a pre-set and constant public-private keypair from a Key Store. This keypair was generated using RSA.
2. When a client first connects to the server, it generates a public-private keypair also using RSA. Then, the client and server exchange their public keys in the form of certificates, so that the client has the server's public key and the server has the client's. Each one also stores its own private key.
3. When a message is transferred between the client and server (in any direction), the sender generates a symmetric key using AES. This symmetric key is used to encrypt the actual message that the server and client want to share (i.e., the payload). The encryption mechanism is AES/CBC. This is done because simple RSA can't encrypt plaintexts of size bigger than its own key-size, so transfer size would be limited.
4. The sender then encrypts the key (as well as a initialization value for the CBC algorithm) using the receiver's public key and sends a packet over with the ciphertext, the encrypted key, and the encrypted initialization value.
5. The receiver then users its private key to decrypt the symmetric key and initialization value and uses those to decrypt the actual payload using the same algorithm it was used to encrypt.
6. After that, the plaintext message is fed to the corresponding server function.

Transfer management in this way protects the communication between server and client, especially when it comes to sensible information such as passwords. Although password security and password hashing do protect the password from being cracked and stolen from a leaked database, if the message itself isn't encrypted, then the password (even if it is already hashed) can be stolen in-flight by a misuser.

6 Detailed Design

The following sections contain a detailed description of the Student Organization System (SOS) in the form of UML package, class, state and sequence diagrams. Section 6.1 contains an overview of the system showing the minimal class diagram for each of the subsystem as well as a short description of each class depicted in those diagrams. Following that, Section 6.2 contains a state machine for the SOS in the form of a UML State Chart Diagram. Section 6.3 contains the object interactions for each of the implemented use cases of the SOS. Finally, Section 6.4 contains a detailed description of each class of the implemented subsystems, as well as OCL constrains for the control object on each subsystem.

6.1 Overview

Each of the following sections contains a minimal UML Class Diagram for each of the subsystems of the SOS. The subsystem decomposition of the SOS can be seen in Section 5.2. For each minimal Class Diagram, a complete equivalent diagram with attributes and operations can be found in Appendix D.

Note that each of the minimal class diagrams also contain the non-subsystem packages showing the relationships to classes on other subsystems.

6.1.1 SOS Website

The minimal class diagram for the SOS Website subsystem can be seen in Figure 14. A full equivalent class diagram can be found in Appendix D.

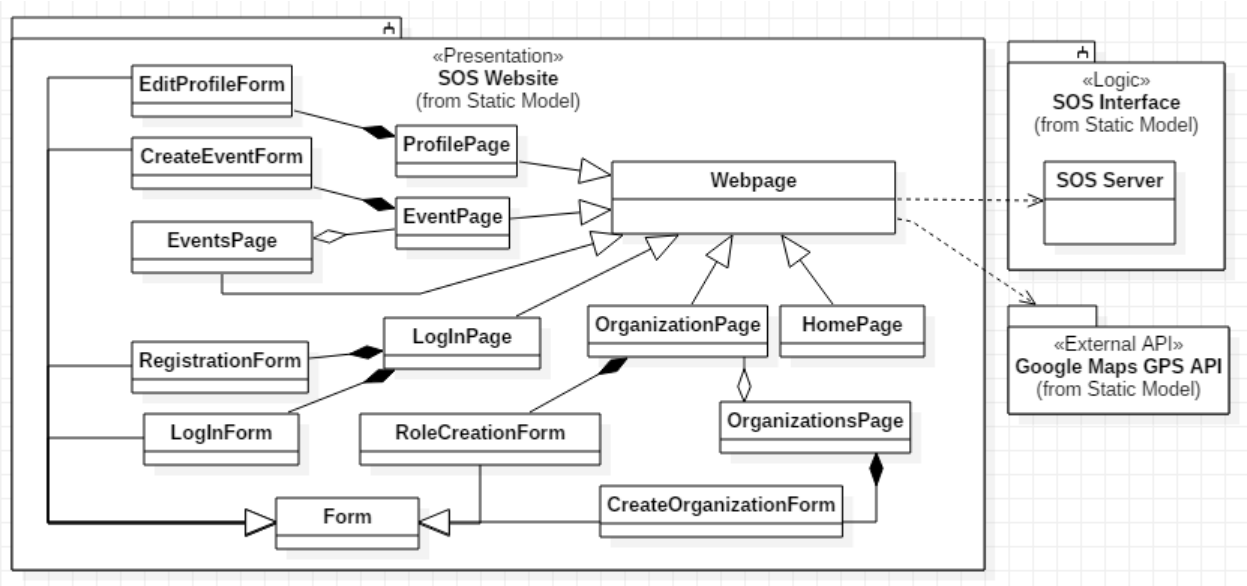

Figure 14: Minimal Class Diagram for SOS Website subsystem.

The following classes belong to this subsystem:

- Webpage, which is the core website class of the system, and the only one that interacts with the backend. Most other classes inherit from this one. The following classes extend Webpage with some specialized data:

- ProfilePage, which contains the User profile.
- EventPage, which contains Event data.
- EventsPage, which contains a list of Events.
- OrganizationPage, which contains Organization Data.
- OrganizationsPage, which contains a list of Organizations.
- HomePage, which contains the home page of the system.

- Form, which is the parent class for a series of input forms in the front end. These are:
  - LogInForm, which is the form for User Login.
  - RegistrationForm, which is the form for new User Registration.
  - CreateEventForm, which is the form for creating an Event.
  - EditProfileForm, which is the form for editing a User profile.
  - CreateOrganizationForm, which is the form for new Organization Creation.
  - RoleCreationForm, which is the form for new Role Creation.

6.1.2 SOS Interface

The minimal class diagram for the SOS Controller subsystem can be seen in Figure 15. A full equivalent class diagram can be found in Appendix D.

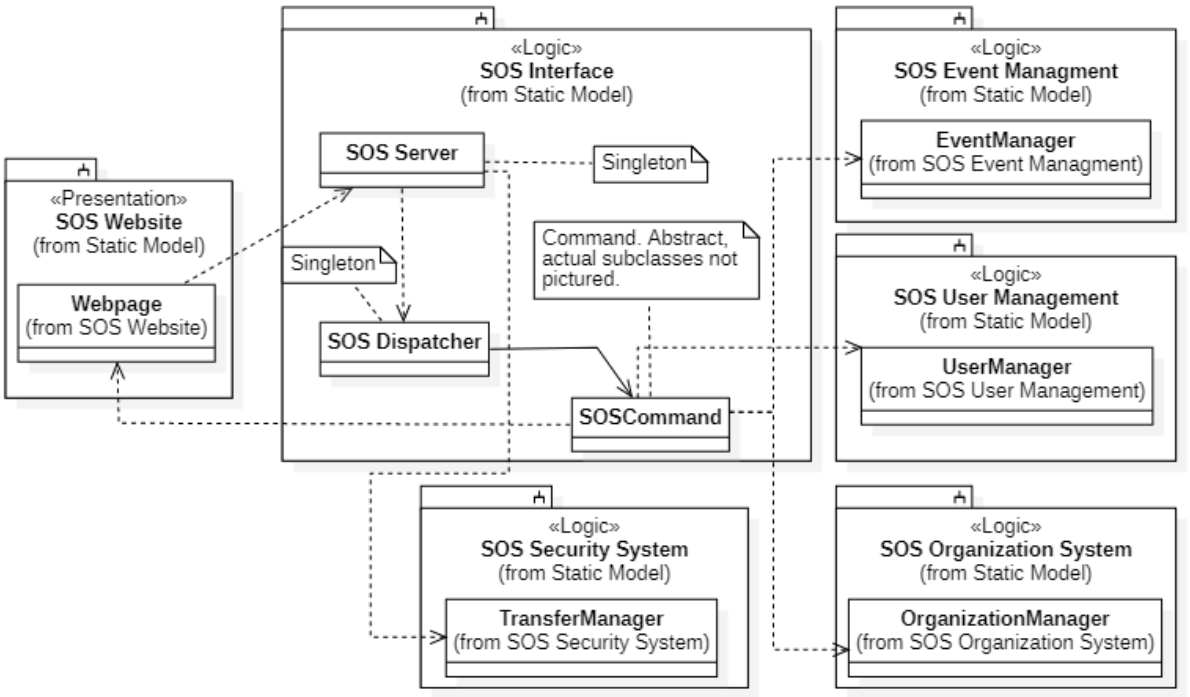

Figure 15: Minimal Class Diagram for SOS Interface.

- The following classes belong to this subsystem:

- SOS Server, which the main server instance of the system.
- SOS Dispatcher, which propagates front-end request to different commands found on the back-end.
- SOS Command, an abstract class that delegates tasks to the different target managers in the back-end of the SOS.

6.1.3 User Management

The minimal class diagram for the User Management subsystem can be seen in Figure 16. A full equivalent class diagram can be found in Appendix D.

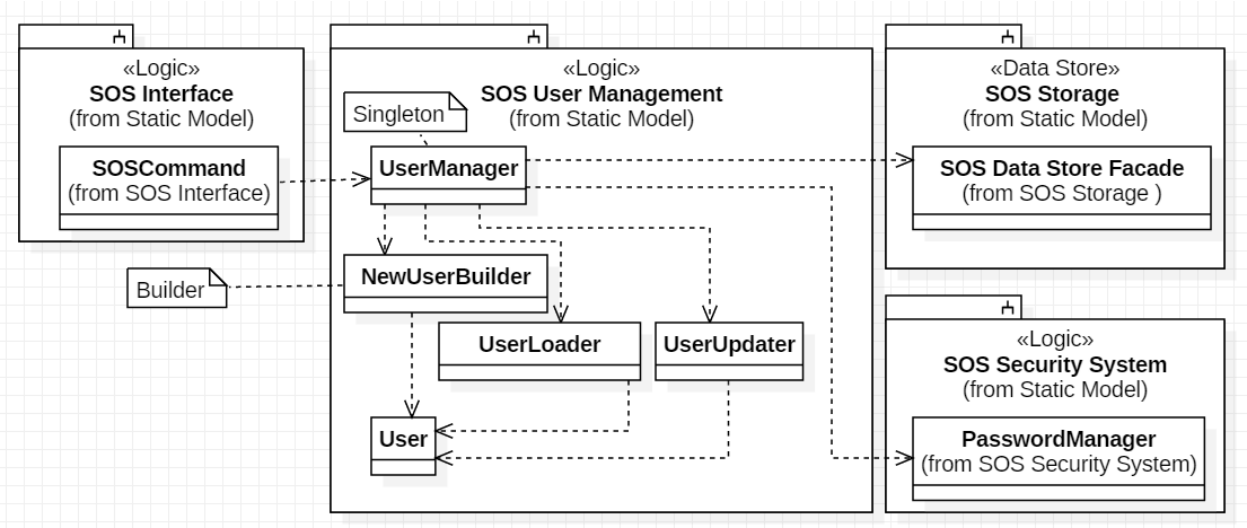

Figure 16: Minimal Class Diagram for SOS User Management.

The following classes belong to this subsystem:

- UserManager, which is a Singleton which manages all the User functions.
- NewUserBuilder, which is a Builder which creates new User objects.
- UserLoader, which is a class which creates a User object from a User database object.
- UserUpdater, which is a class which deal with User modifications.
- User, which is a run-time representation of a User persistent object.

6.1.4 Event Management

The minimal class diagram for the Event Management subsystem can be seen in Figure 17. A full equivalent class diagram can be found in Appendix D.

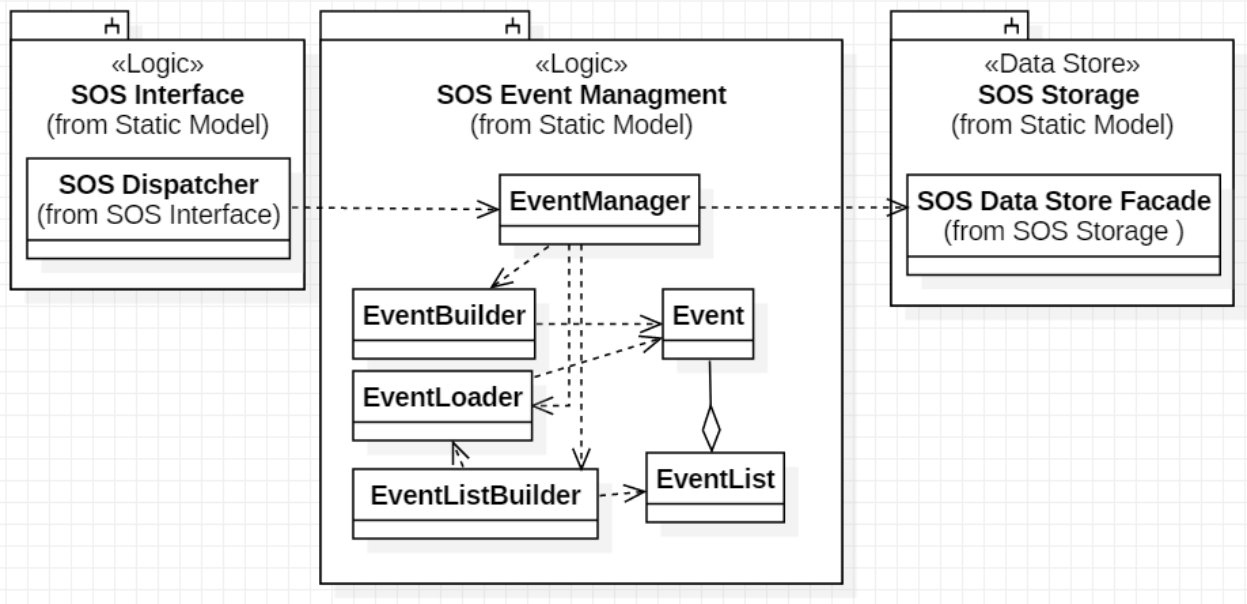

Figure 17: Minimal Class Diagram for SOS Event Management

The following classes belong to this subsystem:

- EventManager, which is a Singleton which manages all the Event functions.
- EventBuilder, which is a Builder which creates new Event objects.
- EventLoader, which is a class which creates an Event object from an Event database object.
- EventListBuilder, which is a Builder which creates new EventList objects.
- Event, which is a run-time representation of an Event database object.
- EventList, which is a class that aggregates Events.

6.1.5 Organization Management

The minimal class diagram for the Organization Management subsystem can be seen in Figure 18. A full equivalent class diagram can be found in Appendix D.

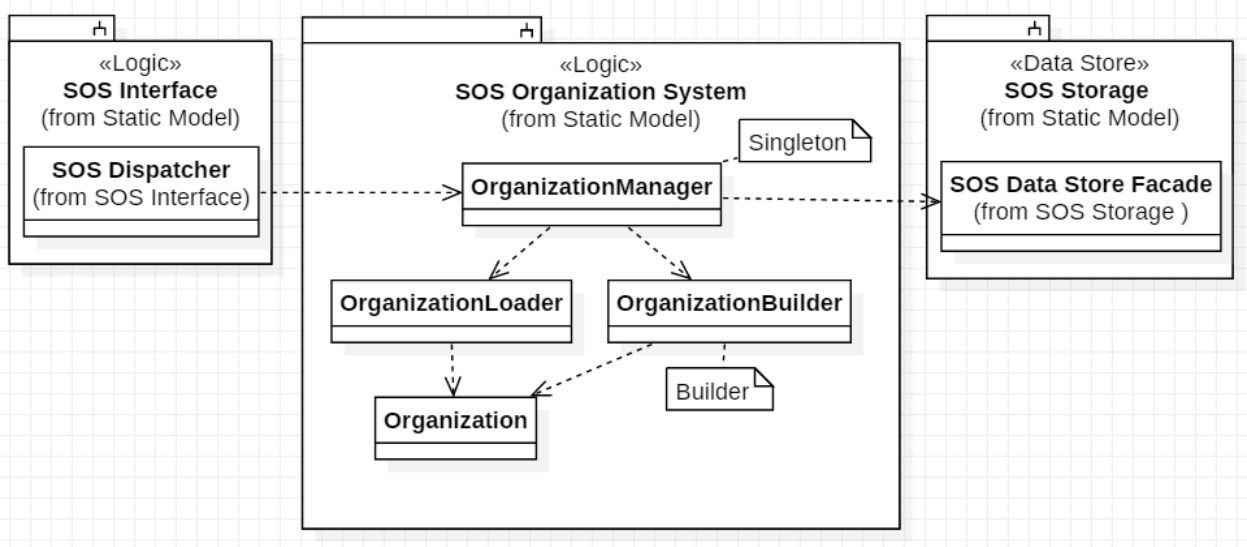

Figure 18: Minimal Class Diagram for SOS Organization Management

The following classes belong to this subsystem:

- OrganizationManager, which is a Singleton which manages all the Organization functions.
- OrganizationBuilder, which is a Builder which creates new Organization objects.
- OrganizationLoader, which is a class which creates an Organization object from an Organization database object.
- Organization, which is a class which deal with Organization modifications.

6.1.6 Security Management

The minimal class diagram for the Security Management subsystem can be seen in Figure 19. A full equivalent class diagram can be found in Appendix D.

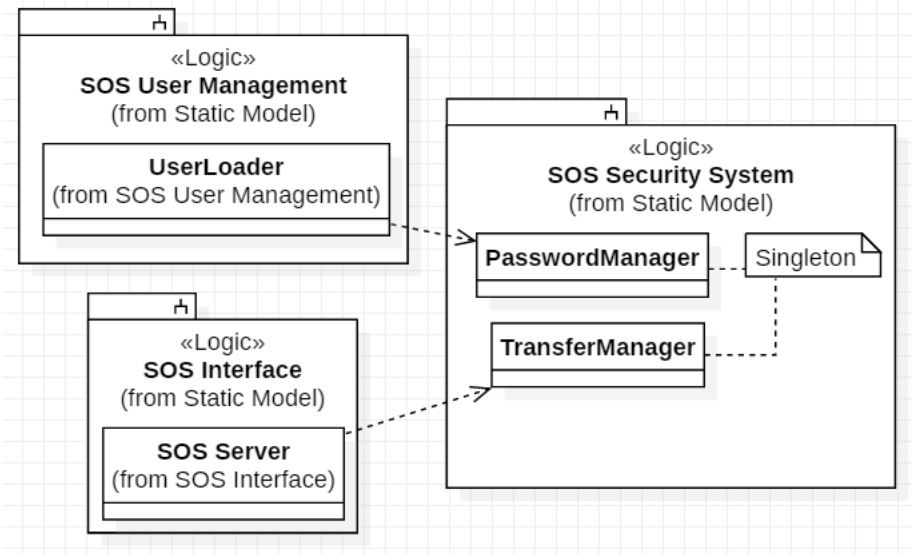

Figure 19: Minimal Class Diagram for SOS Security.

The following classes belong to this subsystem:

- PasswordManager, which is a Singleton dealing with password control actions.
- TransferManager, which is a Singleton dealing with encrypting/decrypting the JSON messages transferred from the front end to the back end and vice versa. It uses an AES/RSA hybrid method.

6.1.7 SOS Storage

The minimal class diagram for the SOS Storage subsystem can be seen in Figure 20. A full equivalent class diagram can be found in Appendix D.

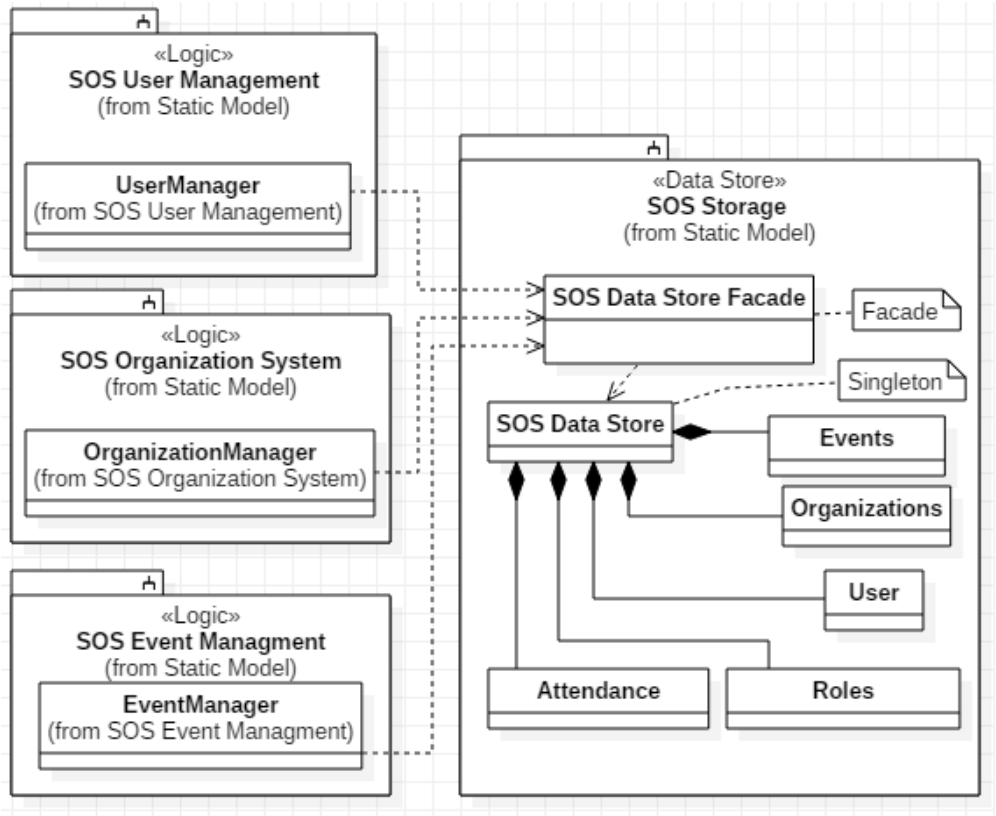

Figure 20: Minimal Class Diagram for Data Store subsystem.

The following classes belong to this subsystem:

- SOS Data Store Façade, which is the interface for the SOS Storage subsystem.
- SOS Data Store, which is the actual database implementation of the SOS system.
- Events, which is the Events table (see Section 2.4).
- Organizations, which is the Organizations table (see Section 2.4).
- User, which is the Users table (see Section 2.4).
- Roles, which is the Roles table (see Section 2.4).
- Attendance, which is the Attendance table (see Section 2.4).

6.1.8 Google Maps GPS API

The Google Maps GPS API does not have a class diagram as external it is just a software module imported into the front-end website code.

6.2 State Machine

The state machine for the SOS is shown in Figure 21. It provides a top-level, general description of the sequence of actions that our system takes, starting with user interactions through the Webpage, to processing and dispatching requests by the SOS Server, to executing those requests in the individual control objects of each subsystem and their interaction with the SOS Storage.

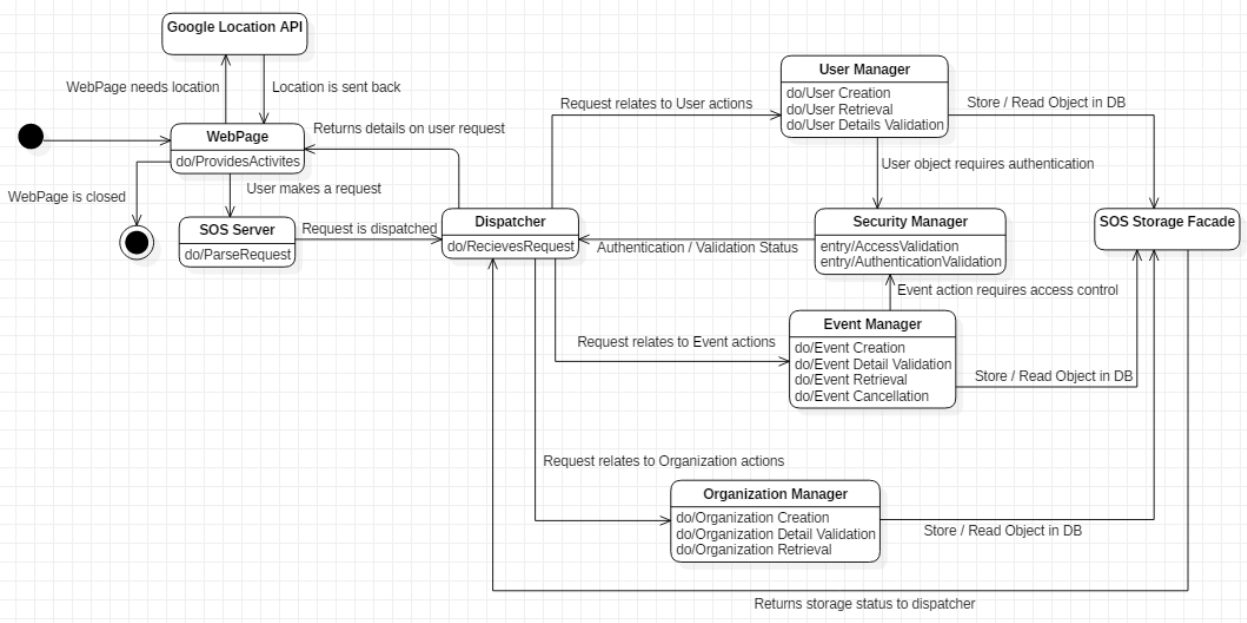

Figure 21: State Machine for the Student Organization System.

6.3 Object Interaction

Each of the following sections contain a sequence diagram detailing the object interactions for each of the implemented Use Cases for the SOS system. Each sequence diagram contains the interactions between the Actors, the core control objects, and key solution objects of the relevant subsystems which implement the functionality of the Use Case.

6.3.1 Sequence Diagram for SOS01 – Create an Event

The sequence diagram in Figure 22 corresponds to the Use Case in Appendix B, Section 11.2.1.

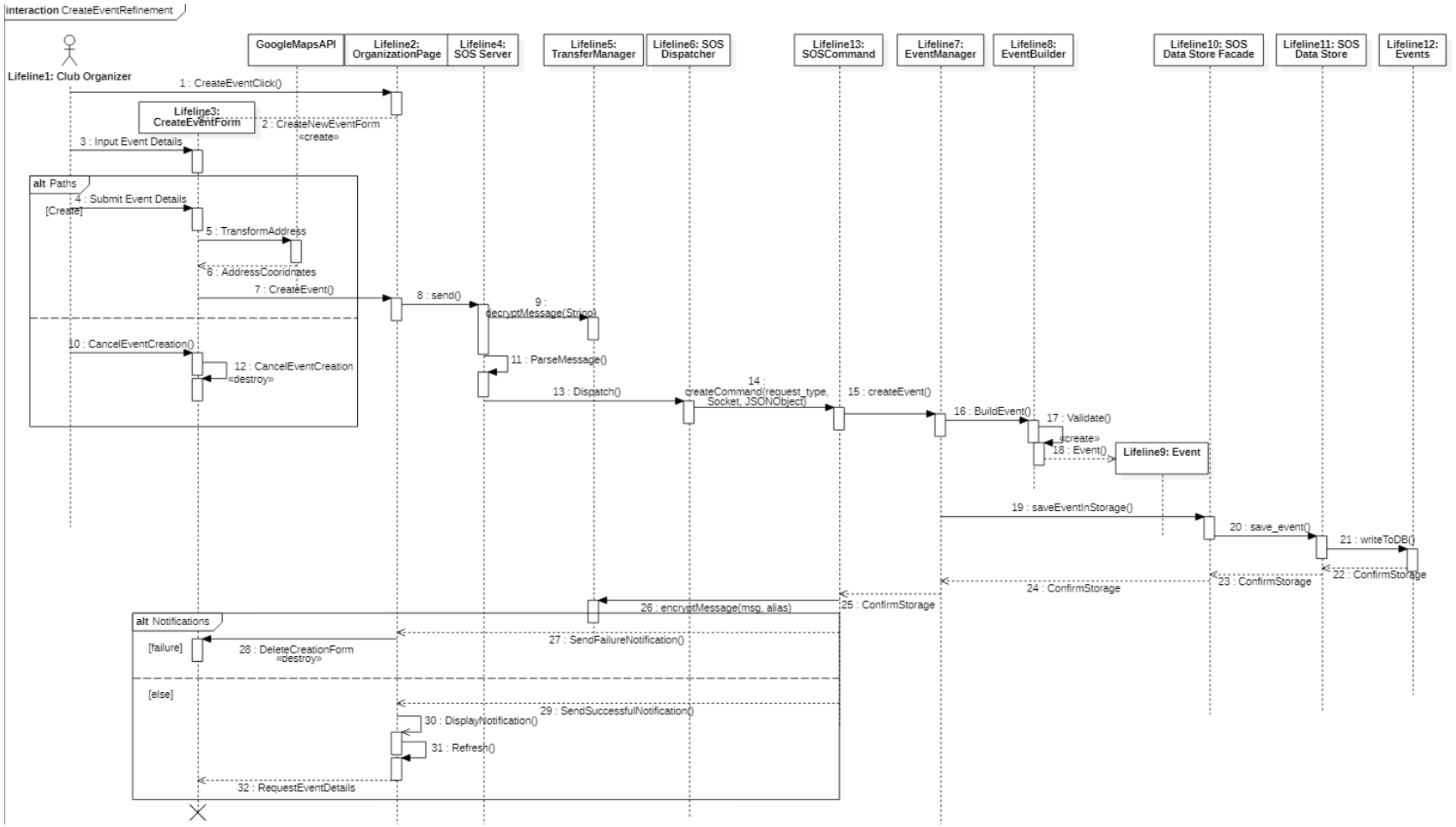

Figure 22: Sequence Diagram for SOS01 - Create an Event

Transfer manager allows the back-end to decrypt the JSON message received from the front-end. Transfer manager also allows the back-end to encrypt the JSON message that they want to respond with to the front-end of the SOS. The Event Manager is a solution object that is responsible for tasks that have to do with event management. The Event Builder allows us to validate the fields of the new event that a user is trying to build. The Event solution object ensures that event creation is also compliant with SOS standards and translates mySQL result sets into JSON object to be sent back to the front-end. Events is the representation of all the events stored in persistent data and it allows us to add new events to the storage of SOS.

6.3.2 Sequence Diagram for SOS04 – Attending an Event.

The sequence diagram in Figure 23 corresponds to the Use Case in Appendix B, Section 11.2.4.

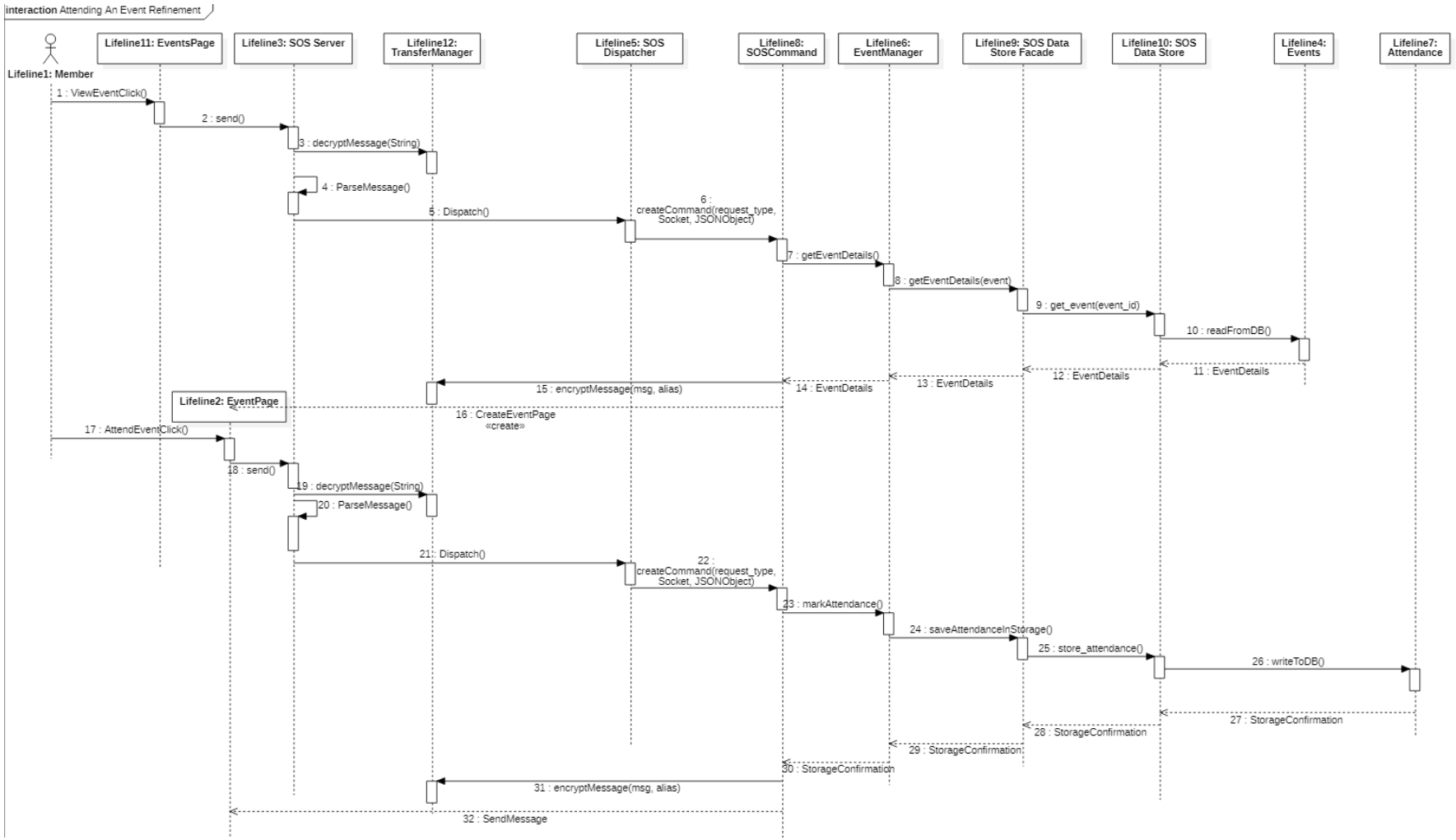

Figure 23: Sequence Diagram for SOS04 – Attending an Event.

Transfer manager allows the back-end to decrypt the JSON message received from the front-end. Transfer manager also allows the back-end to encrypt the JSON message that they want to respond with to the front-end of the SOS. The Event Manager is a solution object that is responsible for tasks that have to do with event management. The Event Builder allows us to validate the fields of the new event that a user is trying to build. The Event solution object ensures that event creation is also compliant with SOS standards and translates mySQL result sets into JSON object to be sent back to the front-end. Events is the representation of all events stored in persistent data. Attendance is the representation of all the events that users have attended in using SOS and this is used to mark and store the physical attendance in the storage of SOS.

6.3.3 Sequence Diagram for SOS02 – Grant Organizer Role

The sequence diagram in Figure 24 corresponds to the Use Case in Appendix B, Section 11.2.2.

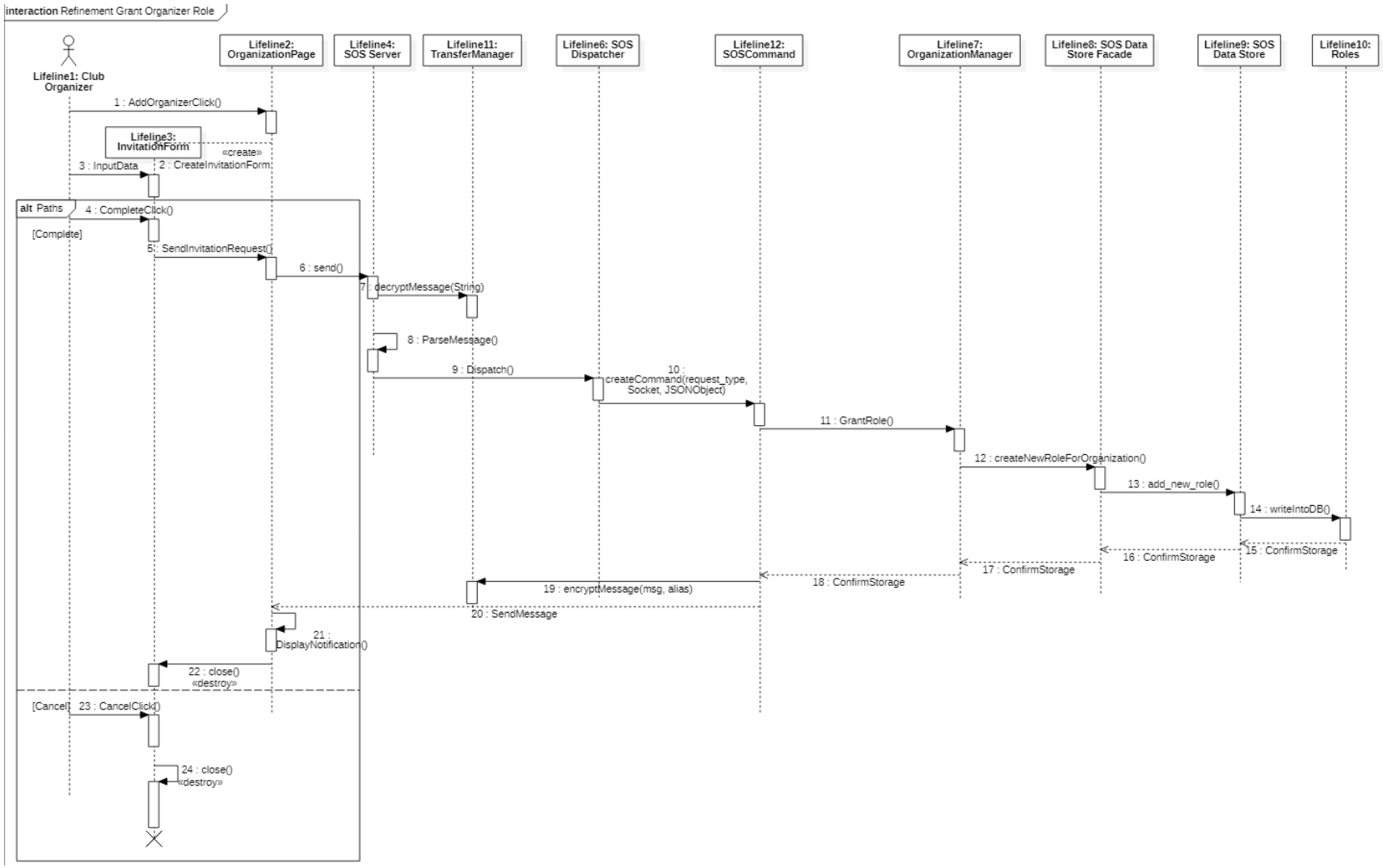

Figure 24: Sequence Diagram for SOS02 – Grant Organizer Role.

Transfer manager allows the back-end to decrypt the JSON message received from the front-end. Transfer manager also allows the back-end to encrypt the JSON message that they want to respond with to the front-end of the SOS. Organization manager is a solution object responsible for tasks that have to do with organizations. Roles represents the current roles stored in persistent data for an organization that exists in SOS and allows for the current roles in the club as well as the addition of new roles for the organization.

6.3.4 Sequence Diagram for SOS07 – Edit Profile

The sequence diagram in Figure 25 25 corresponds to the Use Case in Appendix B, Section 11.2.7.

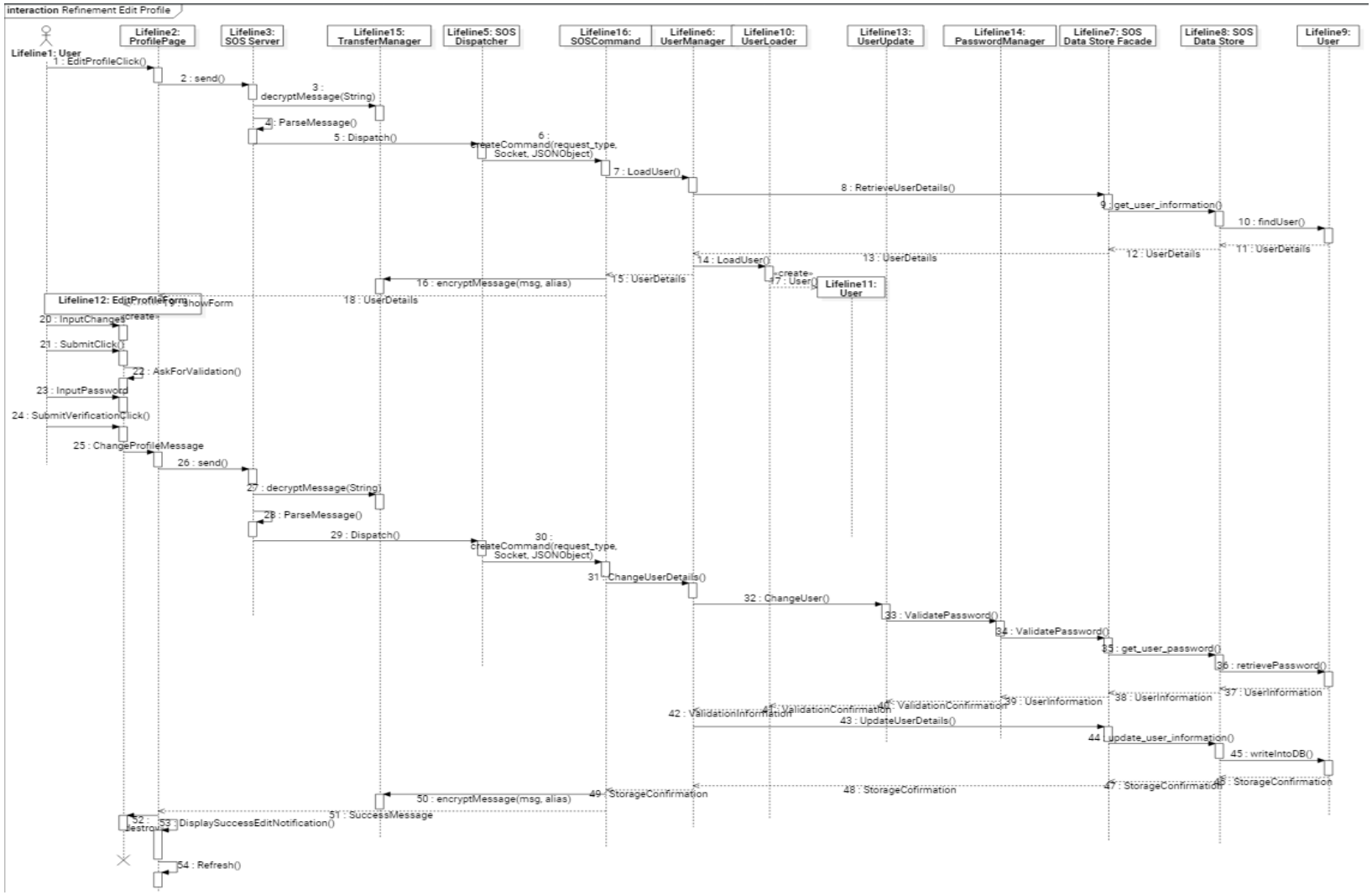

Figure 25: Sequence Diagram for SOS07 – Edit Profile.

Transfer manager allows the back-end to decrypt the JSON message received from the front-end. Transfer manager also allows the back-end to encrypt the JSON message that they want to respond with to the front-end of the SOS. User Manager is a solution object responsible for tasks that have to do with user-oriented actions. User Update is a solution object that allows the verification and validation of updating a user's new fields. User loader is a solution object that allows us to load the user to verify that it was retrieved successfully from the SOS storage. Password manager is responsible for verifying that the password sent through the front-end matches the one stored in the database. Users represents all the persistent data stored on users currently on the SOS system and allows access to the user's profile information.

6.3.5 Sequence Diagram for SOS16 – Create Organization

The sequence diagram in Figure 26 corresponds to the Use Case in Appendix B, Section 11.2.16.

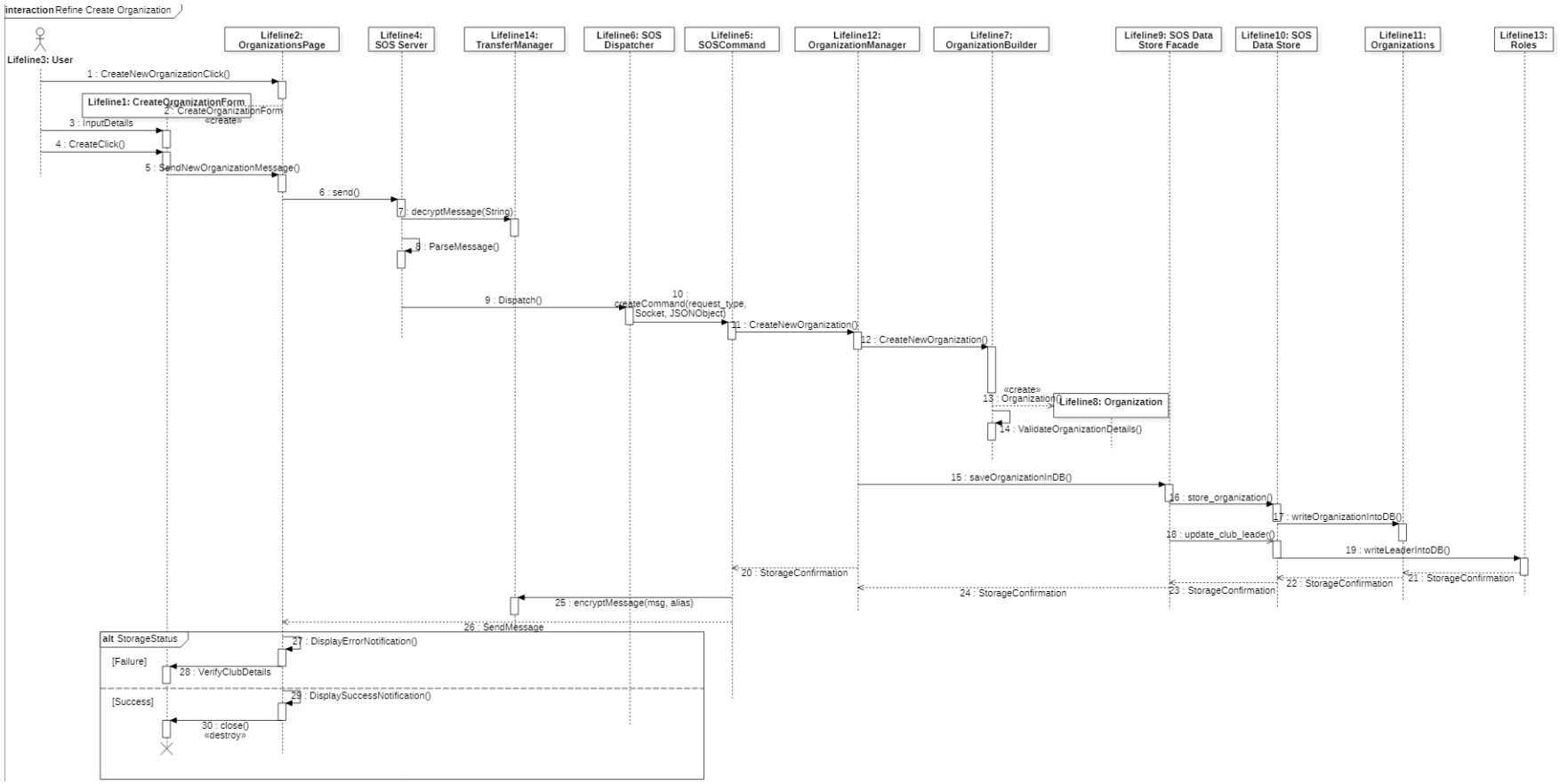

Figure 26: Sequence Diagram for SOS16 – Create Organization.

Transfer manager allows the back-end to decrypt the JSON message received from the front-end. Transfer manager also allows the back-end to encrypt the JSON message that they want to respond with to the front-end of the SOS. Organization Manager is a solution object responsible for tasks that have to do with organization-oriented actions. Organization Builder is a solution object that allows the validation of the fields of the new organization being created. Organization is a solution object that reinforces the validity of the fields that were given by the user and allows as a mechanism to translate mySQL result sets to JSON. Organizations represents persistent data that correlates to the organizations stored in SOS and this allows us to add a new organization to the storage. Roles represents all the roles that are stored in the database for an organization. Roles lets us define a leader for the newly created organization.

6.3.6 Sequence Diagram for SOS17 – Cancel an Event

The sequence diagram in Figure 277 corresponds to the Use Case in Appendix B, Section 11.2.17.

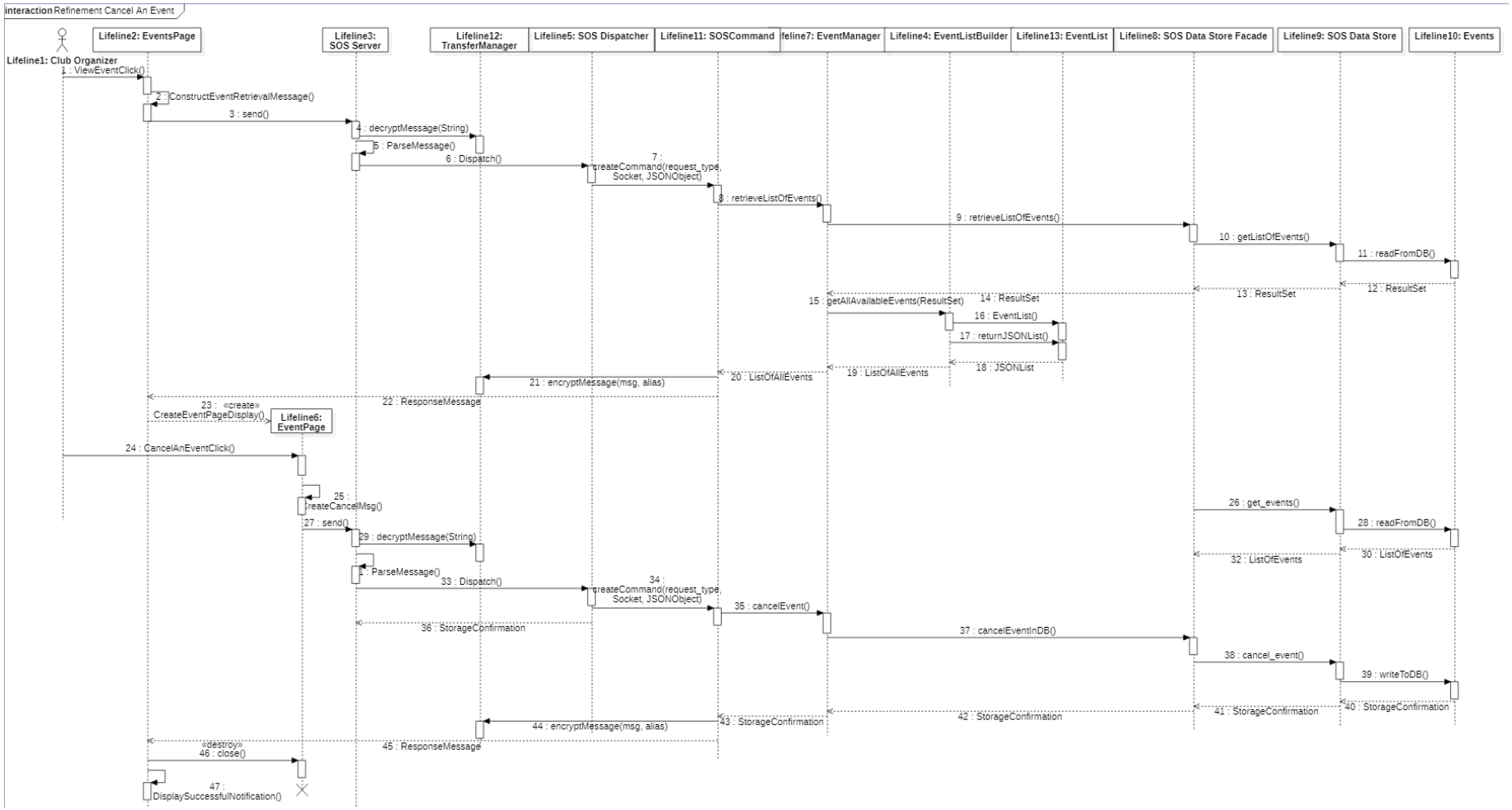

Figure 27: Sequence Diagram for SOS17 – Cancel an Event.

Transfer manager allows the back-end to decrypt the JSON message received from the front-end. Transfer manager also allows the back-end to encrypt the JSON message that they want to respond with to the front-end of the SOS. The Event Manager is a solution object that is responsible for tasks that have to do with event management. The Event List Builder is a solution object that builds a list for the given result set. The Event List is the actual list object that parses through the result set entries and translates them into a JSONList to return to the front end. The Event List Builder and Event List solution objects are mainly used for data retrieval for the front-end. Events is the persistent data set of events that are stored in the SOS. Events allows us to access and read these events as well as create and cancel events (in this case).

6.3.7 Sequence Diagram for SOS22 – Registration

The sequence diagram in Figure 28 corresponds to the Use Case in Appendix B, Section 11.2.22.

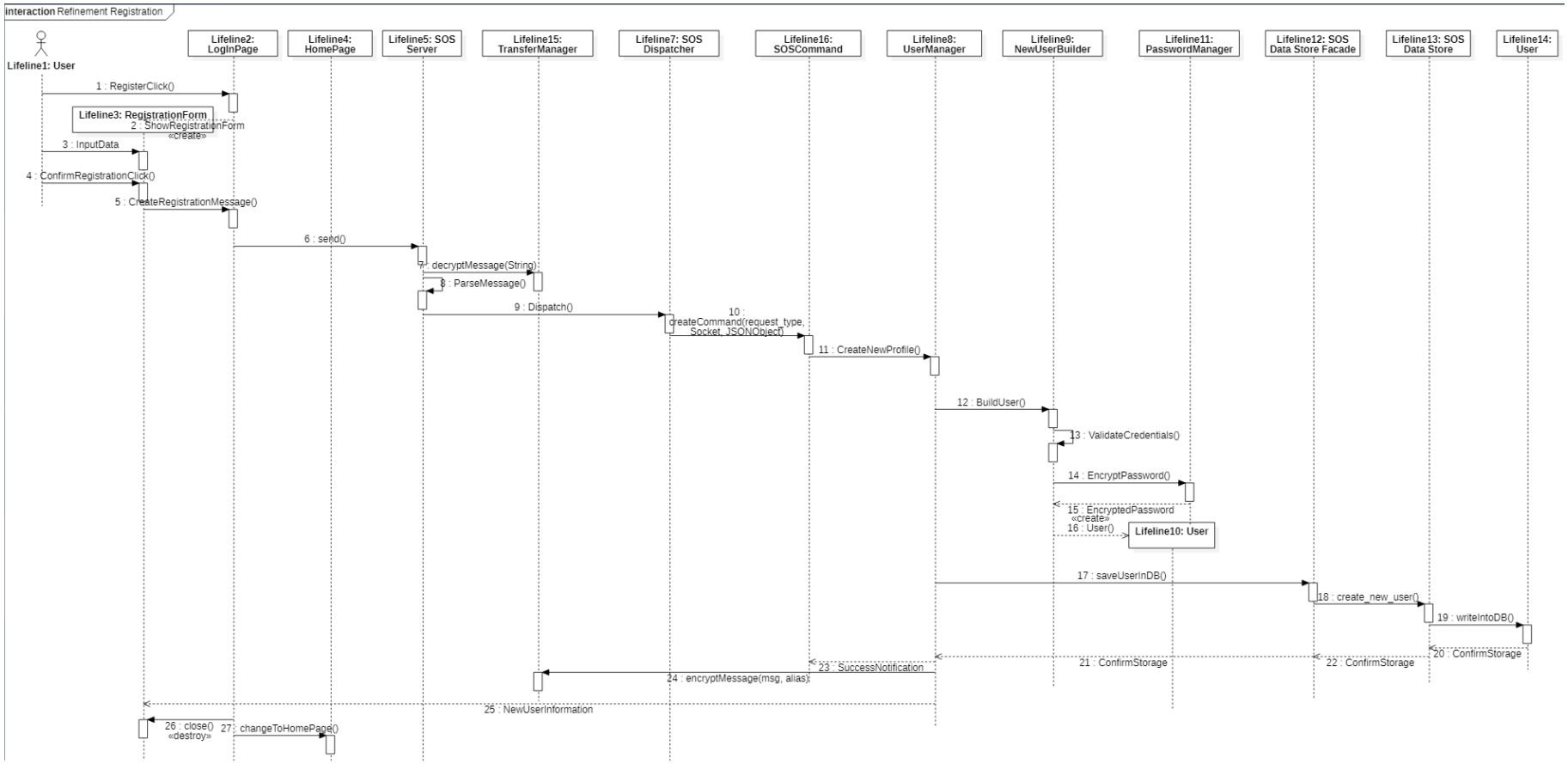

Figure 28: Sequence Diagram for SOS22 – Registration.

Transfer manager allows the back-end to decrypt the JSON message received from the front-end. Transfer manager also allows the back-end to encrypt the JSON message that they want to respond with to the front-end of the SOS. User Manager is a solution object responsible for tasks that have to do with user-oriented actions. User Builder is a solution object that allows the validation of creating a new user with the given fields from the front-end. Password manager is responsible for verifying that the password sent through the front-end meets SOS criteria before it is stored in the database SOS data store. Users represents all the persistent data stored on users currently on the SOS system and allows creation and storage of new Users.

6.3.8 Sequence Diagram for SOS10 – Access an Event by Location

The sequence diagram in Figure 29 corresponds to the Use Case in Appendix B, Section 11.2.10.

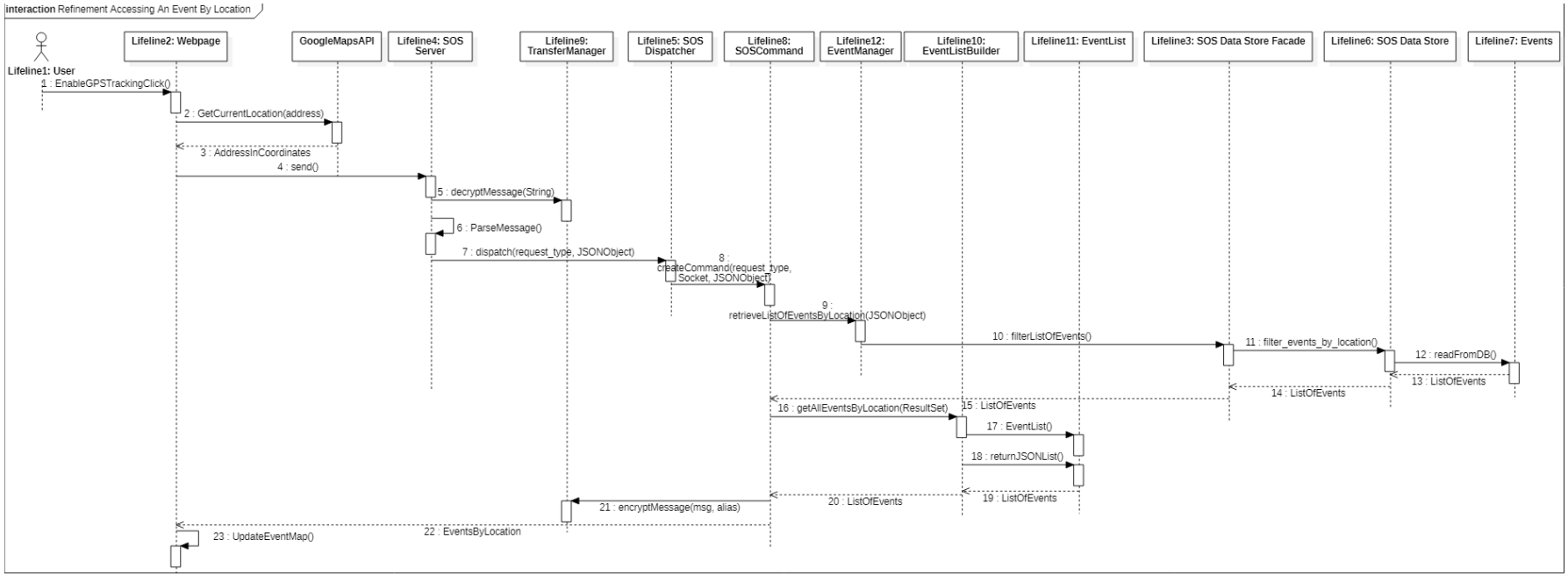

Figure 29: Sequence Diagram for SOS10 – Access an Event by Location.

Transfer manager allows the back-end to decrypt the JSON message received from the front-end. Transfer manager also allows the back-end to encrypt the JSON message that they want to respond with to the front-end of the SOS. The Event Manager is a solution object that is responsible for tasks that have to do with event management. The Event List Builder is a solution object that builds a list for the given result set. The Event List is the actual list object that parses through the result set entries and translates them into a JSONList to return to the front end. The Event List Builder and Event List solution objects are mainly used for data retrieval for the front-end. Events is the persistent data set of events that are stored in the SOS that are stored with data coordinates which are used to compare to the locality of the user when returning a list of events.

6.3.9 Sequence Diagram for SOS31 – Login

The sequence diagram in Figure 30 corresponds to the Use Case in Appendix B, Section 11.2.31.

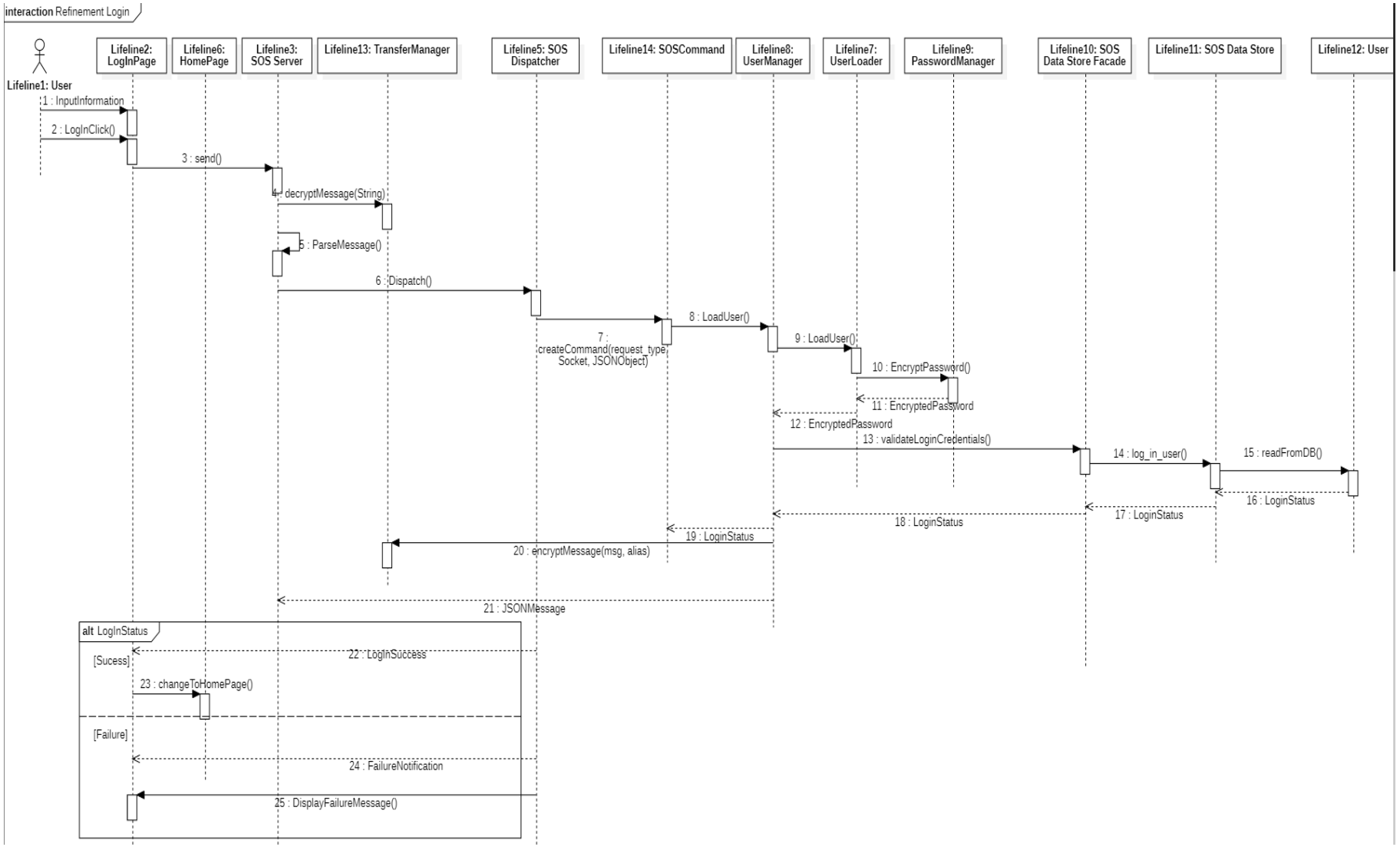

Figure 30: Sequence Diagram for SOS31 – Login.

Transfer manager allows the back-end to decrypt the JSON message received from the front-end. Transfer manager also allows the back-end to encrypt the JSON message that they want to respond with to the front-end of the SOS. User Manager is a solution object responsible for tasks that have to do with user-oriented actions. User Loader is a solution object that uses the Password Manager object to validate the input given by the user attempting to log into the site. Password manager is used to hash the password on the back-end and then used the hash value to compare it to the one stored in the SOS storage. User is the set of persistent data objects available in the SOS storage and in this case, it allows us to retrieve information about the User that is attempting to log-in.

6.3.10 Sequence Diagram for SOS32 – Logout

The sequence diagram in Figure 31 corresponds to the Use Case in Appendix B, Section 11.2.32.

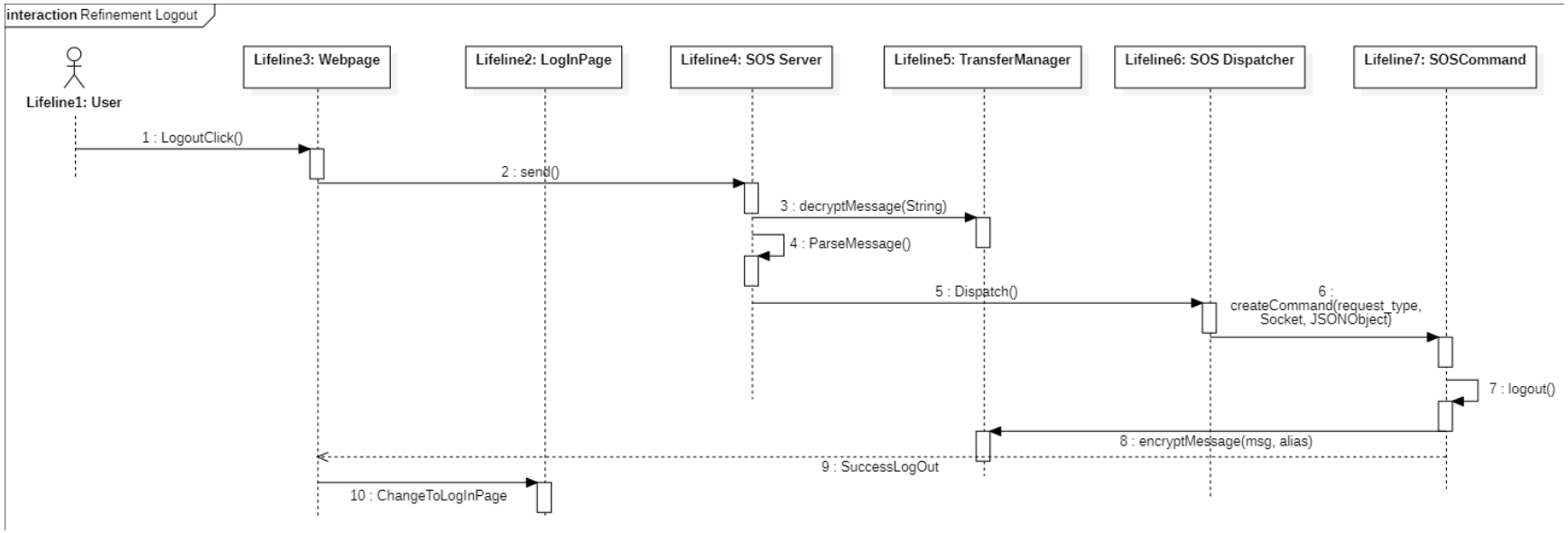

Figure 31: Sequence Diagram for SOS32 – Logout

Transfer manager allows the back-end to decrypt the JSON message received from the front-end. Transfer manager also allows the back-end to encrypt the JSON message that they want to respond with to the front-end of the SOS.

6.4 Detailed Class Design

This section contains the detailed class design of each class in each subsystem of the SOS system. Section 6.4.1 contains a detailed description for each class in the system, while Section 6.4.2 contains the Object Constraint Language (OCL) constraints for each control objects of each subsection.

6.4.1 Class Description

Each one of the following subsections contains a detailed class description for each of the classes in the subsystems of the SOS. In each case, a minimal class diagram can be found in Section 3.1, and a complete class diagram can be found in Appendix D.

6.4.1.1 SOS Website

The complete class diagram for the SOS Website, which contains operations and attributes where applicable, can be found in Figure 52 in Appendix D, Section 11.4.1. The Java Code interface for these classes are in Appendix E, Section 11.5. The following classes are part of the SOS Website:

- Webpage, which is the core website class of the system, and the only one that interacts with the backend. Most other classes inherit from this one, namely all the ones with "Page" in their name. This means that all pages inherit the functionalities form the Webpage, including simple web-app functions like refresh, and more complex ones such as interfacing with the Google Maps GPS API.
- Session, which is a data wrapper class which contains client-specific information about the current session, such as what user is logged in, what their privileges are, current events, and so on. Most of the page information will be stored in this class and accessed rather than directly in pages, and all pages have access to it as they inherit it from Webpage.
- ProfilePage, which contains the User profile. This page presents a view of the User which is different depending on whether the User is seeing their own page, or someone else's page. If a User is logged in an seeing their own page, they have access to editing their profile and changing their privacy settings.
- EventPage, which contains the Event data. This page presents a view of an Event created by an Organization. It should also provide the attendance functionality, but only if the set time for the event and the current time of the system are the same. Event owners (i.e., organizers) also have access to canceling and editing event details.
- EventsPage, which contains a set or list of Events. This class represents a groping of Event classes, each of which can be accessed independently. On top of wrapping a list of events, this class also provides other functionalities such as filtering (namely by GPS location).
- OrganizationPage, which contains the Organization Data. This page presents a view of the Organization which is different depending on whether the User who is seeing it has privileges on the Organization (i.e., is an Organizer). For non-privilege Members, the Organization page just displays relevant information such as the about and description, but for privilege Organizers, it allows Event Creation, Event Cancellation, Member Invites, etc.

- OrganizationsPage, which contains a set or list of Organizations. This class represents a grouping of Organization classes, each of which can be accessed independently. On top of wrapping a list of organizations, this class also provides other functionalities such as filtering, and creating new organizations.
- HomePage, which contains the home page of the system. This is the first view of the system that users see and provides them with navigation links to all other pages.
- Form, which is the parent class for a series of input forms which are used throughout the other pages. As such, it includes several general functions which are inherited by other forms and that implement important features like submitting. Forms also have their own attributes as they don't use the same Session object as the pages.
- LoginForm, which is the form for user login. It stores the username and password and communicates them to the SOS Interface. It resides in the login page.
- RegistrationForm, which is the form for user registration. It stores username, password, confirm password and email and communicates it to the SOS Interface. It resides in the LoginPage.
- CreateEventForm, which is the form for creating an Event. It stores an event name, date, time, duration, type, visibility, and location and communicates it to the SOS Interface. It resides in the EventPage.
- EditProfileForm, which is the form for editing a User profile. It stores an email, privacy setting, date of birth, phone number, and password. Note that when open, these values are preset form the known current value of these attributes in the datastore. It resides inside the ProfilePage.
- CreateOrganizationForm, which is the form for creating a new Organization. It stores an organization name, description, privacy setting, and join requirement and communicates it to the SOS Interface. It resides inside the OrganizationPage.
- RoleCreationForm, which is the form for creating a new Role. It stores a role name, privileges, and security requirements and communicates those to the SOS Interface. It resides inside the OrganizationPage.

6.4.1.2 SOS Interface

The complete class diagram for the SOS Interface, which contains operations and attributes where applicable, can be found in Figure 53 in Appendix D, Section 11.4.2. The Java Code interface for these classes are in Appendix E, Section 11.5. The following classes are part of the SOS Interface:

- SOS Server, which is the main server instance of the system. It implements the socket connections which receive network messages (Requests) from the remote front-end and preprocess them to prepare a dispatched request through SOS Dispatcher. Note that this class is a Singleton, as only one SOS Server is necessary for the entire system.
- SOS Dispatcher, which is in charge of creating a new SOS Command containing the requested action and calling it (execute) so that the functionality is executed. This class is a singleton.
- SOS Command, which is an abstract class representing an action within the system which a client can call. It is extended by actual commands which implement real functionalities

(via the abstract execute method). Note that these subclasses are created dynamically so they are not detailed in this document.

6.4.1.3 User Management

The complete class diagram for the User Management, which contains operations and attributes where applicable, can be found in Figure 54 in Appendix D, Section 11.4.3. The Java Code interface for these classes are in Appendix E, Section 11.5. The following classes are part of the User Management:

- UserManager, which is a Singleton class which managers all the User functions. This class receives dispatched actions from the SOS Dispatcher and completes that action using objects internal to its subsystem. It also is in charge of interacting with the SOS Data Store Façade directly. Part of the role of this class is to parse front-end format user data (e.g., JSON-String defining a new User) and calling the appropriate functions on the other classes according to that data. It also tasked with encoding a User object into database format objects (e.g., SQL Table entry for User).
- NewUserBuilder, which is a Builder which creates new User objects. It is used to decouple the parts of the process of creating a new User from the actual User class, which is intended to only be a data wrapper class which can be easily parsed into the database format. Namely, this class implements the checks and validations necessary to create a valid User and will reject invalid ones. As part of this validation, it must interact with the SOS Security System classes that implement the password and access policies.
- UserLoader, which is a class which creates a User object from a database-format User object (e.g., a SQL Table entry for User). This class decouples the parsing from the database to the system logic from the UserManager class and can be extended to include internal checks for data integrity purposes.
- UserUpdater, which is a class which deals with User modifications. User modifications are done on the system logic-level User object first and are only finalized once they are stored to the database. The UserUpdater decouples these modifications from the UserManager class and from the User class itself and implements checks and validations in the same way that NewUserBuilder does. It also ensures that every modification to the User class is saved to the SOS Data Store.
- User, which is a run-time representation of a User persistent object. This class is used as an intermediary for creation, retrieval, and modification of User data within the Java code (and the JVM). It is encodable (or serializable) to a database format (e.g., SQL Entry).

6.4.1.4 Event Management

The complete class diagram for the Event Management, which contains operations and attributes where applicable, can be found in Figure 55 in Appendix D, Section 11.4.4. The Java Code interface for these classes are in Appendix E, Section 11.5. The following classes are part of the Event Management:

- EventManager, which is a Singleton which manages all the Event functions. This class receives dispatched actions from the SOS Dispatcher and completes that action using

objects internal to its subsystem. It also is in charge of interacting with the SOS Data Store Façade directly. Part of the role of this class is to parse front-end format data (e.g., JSON-String description of new Events) and calling the appropriate functions on other classes according to that data. It is also in charge of encoding Event objects into database-format (e.g., SQL Table entries). Another role is to create EventLists based on filter requests through the EventListBuilder.

- EventBuilder, which is a Builder class which creates new Events. It is used to decouple the parts of the process of creating a new Event from the actual Event class, which is intended to only be a data wrapper class which can be easily parsed into the database format. Namely, this class implements the checks and validations necessary to create a valid Event and will reject invalid ones.
- EventLoader, which is a class which creates an Event object from an Event database object. This class decouples the parsing from the database to the system logic from the EventManager class and can be extended to include internal checks for data integrity purposes.
- EventListBuilder, which is a Builder which creates new EventList objects. As other builders, it is used to decouple the process of creating a new EventList from the actual EventList class, and also provides functions implementing attribute-base filtering (e.g., filter by location, or by hosting organization, etc.)
- EventList, which is a class that aggregates Events.
- Event, which is a run-time representation of an Event persistent Object. This class is used as an intermediary for creation, retrieval, and modification of Event data within the Java code (and the JVM). It is encodable (or serializable) to a database format (e.g., SQL Entry).

6.4.1.5 Organization Management

The complete class diagram for the Organization Management, which contains operations and attributes where applicable, can be found in Figure 56 in Appendix D, Section 11.4.5. The Java Code interface for these classes are in Appendix E, Section 11.5. The following classes are part of the Organization Management:

- OrganizationManager, which is a Singleton which manages all the Organization functions. This class receives dispatched actions from the SOS Dispatcher and completes that action using objects internal to its subsystem. It also is in charge of interacting with the SOS Data Store Façade directly. Part of the role of this class is to parse front-end format data (e.g., JSON-String description of new Organization) and calling the appropriate functions on other classes according to that data. Another job of this class is to manage Role creation and assignment, as well as mediate the modification of data in an Organization, and of Event hosting.
- OrganizationBuilder, which is a Builder which creates new Organization objects. It is used to decouple the process, including validations and checks, of creating an Organization from the actual Organization class itself.
- OrganizationLoader, which is a class which creates an Organization object from an Organization database object. This class decouples the parsing from the database to the

system logic from the OrganizationManager class and can be extended to include internal checks for data integrity purposes.

- Organization, which is a run-time representation of an Organization persistent object. This class is used as an intermediary for creation, retrieval, and modification of Organization data within the Java code (and the JVM). It is encodable (or serializable) to a database format (e.g., SQL Entry).

6.4.1.6 Security Management

The complete class diagram for the Security Management, which contains operations and attributes where applicable, can be found in Figure 57 in Appendix D, Section 11.4.6. The Java Code interface for these classes are in Appendix E, Section 11.5. The following classes are part of the Security Management:

- PasswordManager, which is a Singleton which deals with password control actions. It implements most of the back-end side of the password policy defined in Section 2.5.1, including resolving passwords and checking the input password against the database.
- TransferManager, which is a Singleton which deals with message encryption between the front-end and the backend. It implements the back-end side of the transfer policy defined in Section 2.5.3, including creating keys, storing client certificates, encrypting and decrypting messages, etc. The front-end also has an analogous set of functions.

6.4.1.7 SOS Storage

The complete class diagram for the SOS Storage, which contains operations and attributes where applicable, can be found in Figure 58 in Appendix C, Section 11.4.7. The Java code interface for these classes is in Appendix E, Section 11.5. The following classes are part of the SOS Storage:

- SOS Data Store Façade, which is a Façade object that acts as the interface for the SOS Storage subsystem. The other subsystems interact with the database through a preset set of actions defined in the SOS Data Store Façade. A façade is a structural design pattern which is used to provide a unified interface to a set of objects within a subsystem. Even though our data store is a single object, a façade is still warranted because the SOS Data Store is implemented using a database component (SQL) and through the SOS Data Store Façade we can decouple the details of the database component (such as the SQL language) from the rest of the system.
- SOS Data Store, which is the actual database implementation for the SOS Storage Subsystem. The other subsystems interact with it through the SOS Data Store Façade. This class implements the system data storage and is effectively the repository (or repository interface) in the Repository architecture of our system. The database component it links to is a relational (SQL) database file which implements the data management policy described in Section 2.4.
- User, which represents the User table on the database.
- Organizations, which represents the Organizations table on the database.
- Roles, which represents the Roles table on the database.
- Attendance, which represents the Attendance table on the database.

6.4.2 Control Objects Description

This section contains the Object Constraint Language (OCL) specification of the interfaces of each of the major control objects in the SOS subsystems. The OCL defines interfaces in terms of three types of constraints on classes: (a) an invariant, which is a predicate that is always true for a class; (b) a precondition, which is a predicate that must be true before the class is used; and (c) a postcondition, which is a predicate that must be true after the class is used.

6.4.2.1 SOS Server OCL

The OCL specification for the SOS Server class is defined in Figure 32.

```
context SOS Server
inv: self.sessionManager <> null

context SOS Server :: ParseMessage(JSONString)
pre:    json.ofCorrectFormat(JSONString)
        and JSONString.contains(action)
        and JSONString.contains(payload)

context SOS Server :: send (action, payload)
pre:    SOS Dispatcher.acceptedActions.includes(action)
        and action.ofCorrectFormat(payload)
```

Figure 32: OCL specification for SOS Sever.

6.4.2.2 User Manager OCL

The OCL specification for the User Manager class is defined in Figure 33.

```
context User Manager :: LoadUser (SQLEntry)
pre:    SQLEntry.contains(name)
        and SQLEntry.contains(user_name)
        and SQLEntry.contains(email)
        and SQLEntry.contains(password)
        and SQLEntry.contains(user_id)
        and SQLEntry.contains(privacy)

context User Manager :: changeUserDetails (User, change)
pre:    change.contains(name)
        or change.contains(user_name)
        or change.contains(email)
        or change.contains(privacy)
post:   User.name == change.name
        or User.user_name == change.user_name
        or User.email == change.email
        or User.privacy == change.privacy

context User Manager :: CreateNewProfile (JSONString)
pre:    JSONString.contains(name)
        and JSONString.contains(user_name)
        and JSONString.contains(email)
        and JSONString.contains(password)
```

Figure 33: OCL specification for User Manager.

6.4.2.3 Organization Manager OCL

The OCL specification for the Organization Manager class is defined in Figure 34: OCL

```
context Organization Manager :: grantRole(user_id, organization_id, privilege_ids)
pre:    SOS Database.User.contains(user_id)
        and privilege_ids->any(SOS Database.Privileges.contains(self))
post:   SOS Database.Roles.contains((user_id, organization_id))
```

specification for Organization Manager..

Figure 34: OCL specification for Organization Manager.

6.4.2.4 Event Manager OCL

The OCL specification for the Event Manager class is defined in Figure 35.

```
context Event Manager  :: cancelEvent (event_id)
pre:    SOS Database.Event.contains(event_id)
        and SOS Database.Event.get(event_id).cancelled == False
post:   SOS Database.Event.get(event_id).cancelled == True

context Event Manager  :: createEvent (JSONString)
pre:    JSONString.contains(location)
        and JSONString.contains(description)
        and JSONString.contains(date)
        and JSONString.contains(time)
        and JSONString.contains(event_type)
        and JSONString.contains(hosted_by)
        and JSONString.contains(visibility)

context Event Manager :: retrieveListOfEvents (event_ids)
pre:    event_ids.size() > 0 and event_ids->any(SOS Database.Event.contains(self))

context Event Manager :: markAttendance (user_id, event_id)
pre:    SOS Database.Event.contains(event_id)
        and SOS Database.User.contains(user_id)
post:   SOS Database.Attendance.contains((user_id, event_id))

context Event Manager :: getEventDetails (event_id)
pre:    SOS Database.Event.contains(event_id)
```

Figure 35: OCL specification for Event Manager.

6.4.2.5 PasswordManager OCL

The OCL specification for the PasswordManager class is defined in Figure 36.

```
context PasswordManager
inv:    self.passwordPolicy <> null
```

Figure 36: OCL specification for PasswordManager.

7 Testing Process

The following section contains the relevant information regarding the testing done for the Student Organization System (SOS). Section 7.1 contains the Unit Tests implemented for the manager classes of the system, each unit test representing a test on a function of that system. Section 7.2 contains the system tests, representing the tests done on the system as a whole, with the front-end as an interface. Finally, Section 7.3 contains an evaluation of the tests on the two previous sections.

7.1 Unit Tests

This section contains the unit tests for the SOS system, grouped into subsections each representing their container classes. Each unit test is a test done on a relevant function on a single class of the system. Testing was done using the JUnit plugin for Eclipse. The relevant code for each unit test can be see in the relevant subsection of Appendix F.

7.1.1 TransferManager

The unit tests in this section represent tests on functions for the TransferManager class. For the code for the tests for this class, check Appendix F, Section 11.6.1.1. It contains the following tests:

1. SOS-UnitTest-001, which tests the instance() function.
2. SOS-UnitTest-002, which tests the encryptMessage() function.
3. SOS-UnitTest-003, which tests the decryptMessage() function.
4. SOS-UnitTest-004, which tests the getSharableCertificate() function.
5. SOS-UnitTest-005, which tests the setCertificateEntry() function.

7.1.2 OrganizationManager

The unit tests in this section represent tests on functions for the OrganizationManager class. For the code for the tests for this class, check Appendix F, Section 11.6.1.2. It contains the following tests:

1. SOS-UnitTest-006, which tests the instance() function.
2. SOS-UnitTest-007, which test the loadOrganizationDetails() function.

7.1.3 EventManager

The unit tests in this section represent tests on functions for the EventManager class. For the code for the tests for this class, check Appendix F, Section 11.6.1.3. It contains the following tests:

1. SOS-UnitTest-008, which tests the instance() function.
2. SOS-UnitTest-009, which tests the loadEvent() function.

7.1.4 UserManager

The unit tests in this section represent tests on functions for the UserManager class. For the code for the tests for this class, check Appendix F, Section 11.6.1.4 It contains the following tests:

1. SOS-UnitTest-010, which tests the instance() function.

2. SOS-UnitTest-011, which tests the loadUser() function.
3. SOS-UnitTest-012, which tests the login() function.
4. SOS-UnitTest-013, which tests the UserManager() constructor.
5. SOS-UnitTest-014, which tests the changeUserDetails() function.
6. SOS-UnitTest-015, which tests the getMembersOfOrganization() function.

7.1.5 SOSServerTest

The unit tests in this section represent tests on functions for the SOSServer class. For the code for the tests for this class, check Appendix F, Section 11.6.1.5. It contains the following test:

1. SOS-UnitTest-016, which tests the instance() function.

7.2 System Tests

This section contains the system tests for the SOS system, grouped into subsections each representing their relevant use case. Two types of use cases are provided, sunny-day, which represent non-exceptional sequence of actions, and rainy-day, which represent exceptional ones. These use cases where tested using Selenium. The code for these test cases is provided in another document.

7.2.1 Logout

The following system test cases belong to the Logout Use Case (SOS32).

7.2.1.1 SOS32-System-001-Sunny-01

Name: SOS32-System-001-Sunny-01
Purpose: Investigate normal execution of the Use Case with inputs which are similar to those expected by the system.
Test set up: The SOS system is set up and working. The User is using Chrome as their browser. The User is already logged in. This User wants to log out of their account on the system. The logout button is accessible and visible to such User on the top right corner of the main navigation bar.
Input: The following sequence is done:

1. User clicks on Logout button.

Expected Output: The system completes the request without exceptions or errors. The User will be logged out of their account successfully and will go back to the system's login page.

7.2.1.2 SOS32-System-002-Sunny-02

Name: SOS32-System-002-Sunny-02
Purpose: Investigate normal execution of the Use Case with inputs which are similar to those expected by the system.
Test set up: The SOS system is set up and working. The User is using Safari as their browser. The User is already logged in and on the SOS home page. This User wants to log out of their account on the system. The logout button is accessible and visible to such User on the top right corner of the main navigation bar.
Input: The following sequence is done:

1. User clicks on account name on the navigation bar.
2. User then clicks on the Logout button.

Expected Output: The system completes the request without exceptions or errors. The User will be logged out of their account successfully and will go back to the system's login page.

7.2.1.3 SOS32-System-003-Rainy-01

Name: SOS32-System-003-Rainy-01
Purpose: Investigate the execution of the Use Case with inputs which are expected to create exceptions or errors on the system.
Test set up: The SOS system is set up and working. The User is using Safari as their browser. The User is already logged in and on the SOS home page. This User wants to log out of their account on the system. The logout button is accessible and visible to such User on the top right corner of the main navigation bar. Note that internet (network) connection is having connectivity issues and working slowly.
Input: The following sequence is done:

1. User clicks on Logout button.

Expected Output: The system cannot complete the request without exceptions or errors. The user will receive a time-out error after 10 seconds and will be redirected to the SOS webpage.

7.2.1.4 SOS32-System-004-Rainy-02

Names: SOS32-System-004-Rainy-02
Purpose: Investigate the execution of the Use Case with inputs which are expected to create exceptions or errors on the system.
Test set up: The SOS system is set up and working. The User is using Safari as their browser. The User is already logged in and on the SOS home page. This User wants to log out of their account on the system. The logout button is accessible and visible to such User on the top right corner of the main navigation bar. Note that internet (network) connection is having connectivity issues and working slowly.
Input: The following sequence is done:

1. User clicks on account name on the navigation bar.
2. User then clicks on the Logout button.

Expected Output: The system cannot complete the request without exceptions or errors. The user will receive a time-out error after 10 seconds and will be redirected to the SOS webpage.

7.2.2 Access Events by Location

The following system test cases belong to the Access Events by Location Use Case (SOS10).

7.2.2.1 SOS10-System-005-Sunny-03

Name: SOS10-System-005-Sunny-03
Purpose: Investigate normal execution of the Use Case with inputs which are similar to those expected by the system.
Test set up: The SOS system is set up and working. The User is using Chrome as their browser. The User is already logged in. The User has their GPS tracking feature enabled on their device. Then the User want to locate the nearby events on the system.
Input: The following sequence is done:

1. User goes to Events page or Home page on the website.
2. User then clicks accept using location by system in the given prompt.

Expected Output: The system completes the request without exceptions or errors. The User will be shown an Event map which their current location is the center and also includes the nearby

events. In addition the Event feed is also reordered to prioritize the Events within the range of the User's location.

7.2.2.2 SOS10-System-006-Sunny-04

Name: SOS10-System-006-Sunny-04
Purpose: Investigate normal execution of the Use Case with inputs which are similar to those expected by the system.
Test set up: The SOS system is set up and working. The User is using Firefox as their browser. The User is already logged in. The User has their GPS tracking feature enabled on their device. Then the User want to locate the nearby events on the system. Note that the User is way out of the range from all of the Events happening in the certain geological testing environment.
Input: The following sequence is done:

1. User goes to Events page or Home page on the website.
2. User then clicks accept using location by system in the given prompt.

Expected Output: The system completes the request without exceptions or errors. The User will be shown a map which their current location is the center and also empty of nearby Events. In this case the Event feed is not reordered due to the far distance of the User from all events happening in the geological environment.

7.2.2.3 SOS10-System-007-Rainy-03

Name: SOS10-System-007-Rainy-03
Purpose: Investigate the execution of the Use Case with inputs which are expected to create exceptions or errors on the system.
Test set up: The SOS system is set up and working. The User is using Chrome as their browser. The User is already logged in. The User do not have their GPS tracking feature enabled on their device.
Input: The following sequence is done:

1. User goes to Events page or Home page on the website.
2. User then clicks accept using location by system in the given prompt.

Expected Output: The system cannot complete the request without exceptions or errors. System will not be able to locate the current User's coordinates and hence cannot determine the nearby Events. In this case an alternate course of action will be applied. Furthermore, the Event map and Event feed will show all of the related Events without any prioritization, and also map will be centered to a predefined center point coordinates.

7.2.2.4 SOS10-System-008-Rainy-04

Name: SOS10-System-008-Rainy-04
Purpose: Investigate the execution of the Use Case with inputs which are expected to create exceptions or errors on the system.
Test set up: The SOS system is set up and working. The User is using Chrome as their browser. The User is already logged in. The User has their GPS tracking feature enabled on their device. Then the User want to locate the nearby events on the system.
Input: The following sequence is done:

1. User goes to Events page or Home page on the website.
2. User then clicks deny using location by system in the given prompt.

Expected Output: The system cannot complete the request without exceptions or errors. System will not be able to locate the current User's coordinates and hence cannot determine the nearby Events. In this case an alternate course of action will be applied. Furthermore, the Event map and Event feed will show all of the related Events without any prioritization, and also map will be centered to a predefined center point coordinates.

7.2.3 Grant Organizer Role

The following system test cases belong to the Grant Organizer Role Use Case (SOS02).

7.2.3.1 SOS02-System-009-Sunny-05

Name: SOS02-System-009-Sunny-05
Purpose: Investigate normal execution of the Use Case with inputs which are similar to those expected by the system.
Test set up: The SOS system is set up and working. The Organizer is using Chrome as their browser. An Organizer is logged in and belongs to the chosen Organization and has rights to give privileges to other Members in the organization. This other Member exists in the system with ID "Johns01". There is a Privilege in the system called "Event Manager" which can be assigned.
Input: The following sequence is done:

1. Organizer clicks on Add Organizer.
2. Organizer adds the following data:
    a. Member ID = "Johns01"
    b. Organizer Title = "Event Organizer"
    c. Powers and Privileges = "Event Manager"
3. Organizer clicks on Complete.

Expected Output: The system completes the request without exceptions or errors. A new Role object is saved to the database reflecting the relation between the Member and the Privilege. A new Request object is saved to the database reflecting the whole transaction. The Member is able to create events on the Organization.

7.2.3.2 SOS02-System-010-Sunny-06

Name: SOS02-System-010-Sunny-06
Purpose: Investigate normal execution of the Use Case with inputs which are similar to those expected by the system.
Test set up: The SOS system is set up and working. The Organizer is using Firefox as their browser. An Organizer is logged in, is the current Owner of the chosen Organization and has rights to give privileges to other Members in the organization. This other Member exists in the system with ID "Jdoe001". There is a Privilege in the system called "Owner" which can be assigned.
Input: The following sequence is done:

1. Organizer clicks on Add Organizer.
2. Organizer adds the following data:
    a. Member ID = "Jdoe001"
    b. Organizer Title = "President"
    c. Powers and Privileges = "Owner"

3. Organizer clicks on Complete.

Expected Output: The system completes the request without exceptions or errors. A new Role object is saved to the database reflecting the relation between the Member and the Privilege. A new Request object is saved to the database reflecting the whole transaction. The Member is now the owner of the Organization. The previous owner is no longer the owner of the Organization.

7.2.3.3 SOS02-System-011-Rainy-05

Name: SOS02-System-011-Rainy-05
Purpose: Investigate the execution of the Use Case with inputs which are expected to break the system.
Test set up: The SOS system is set up and working. The Organizer is using Firefox as their browser. An Organizer is logged in and belongs to the chosen Organization and has rights to give privileges to other Members in the Organization. The target Member does not exist in the system. There is a Privilege in the system called "Event Manager" which can be assigned.
Input: The following sequence is done:

1. Organizer clicks on Add Organizer.
2. Organizer adds the following data:
    a. Member ID = "Jdoe001"
    b. Organizer Title = "Event Organizer"
    c. Powers and Privileges = "Event Manager"
3. Organizer clicks on Complete.

Expected Output: The system does not complete the request as it finds that the member ID cannot be found. It displays an error message explaining precisely that. A new Request object is saved to the database reflecting the whole transaction, noting the erroneous result.

7.2.3.4 SOS02-System-012-Rainy-06

Name: SOS02-System-012-Rainy-06
Purpose: Investigate the execution of the Use Case with inputs which are expected to break the system.
Test set up: The SOS system is set up and working. The Organizer is using Firefox as their browser. An Organizer is logged in and belongs to the chosen but is missing the rights to give other Members roles. The target Member exists in the System and has the Member ID "Jdoe001". There is a Privilege in the system called "Event Manager" which can be assigned.
Input: The following sequence is done:

1. Organizer clicks on Add Organizer.
2. Organizer adds the following data:
    a. Member ID = "Jdoe001"
    b. Organizer Title = "Event Organizer"
    c. Powers and Privileges = "Event Manager"
3. Organizer clicks on Complete.

Expected Output: The system does not complete the request as it finds that the Organizer does not have the rights to give other privileges. It displays an error message explaining precisely that. A new Request object is saved to the database reflecting the whole transaction, noting the erroneous result.

7.2.4 Attending an Event

The following system test cases belong to the Attending an Event Use Case (SOS04).

7.2.4.1 SOS04-System-013-Sunny-07

Name: SOS04-System-013-Sunny-07
Purpose: Investigate normal execution of the Use Case with inputs which are similar to those expected by the system.
Test set up: The SOS system is set up and working. The Organizer is using Chrome as their browser. A Member, "jdoe001", is logged in. This Member belongs to an Organization which is hosting an Event, "General Meeting", and is currently at the physical location of that Event.
Input: The following sequence is done:

1. Member clicks on Events.
2. Member clicks on the "General Meeting" Event.
3. Member clicks on the "I'm Here!" button on the Event page.

Expected Output: The system completes the request without exceptions or errors. A new Attendance object is created and added to the Database reflecting the relationship between the User and the Event. It includes the correct real-time of attendance. A Request object is created reflecting the while transaction.

7.2.4.2 SOS04-System-014-Sunny-08

Name: SOS04-System-014-Sunny-08
Purpose: Investigate normal execution of the Use Case with inputs which are similar to those expected by the system.
Test set up: The SOS system is set up and working. The Organizer is using Firefox as their browser. A Member, "johns01", is logged in. This Member belongs to an Organization which is hosting an Event, "General Meeting", and is currently at the physical location of that Event.
Input: The following sequence is done:

1. Member clicks on Events.
2. Member clicks on the "Fishing for Dummies" Event.
3. Member clicks on the "I'm Here!" button on the Event page.

Expected Output: The system completes the request without exceptions or errors. A new Attendance object is created and added to the Database reflecting the relationship between the User and the Event. It includes the correct real-time of attendance. A Request object is created reflecting the while transaction.

7.2.4.3 SOS04-System-015-Rainy-07

Name: SOS04-System-015- Rainy-07
Purpose: Investigate the execution of the Use Case with inputs which are expected to create exceptions on the system.
Test set up: The SOS system is set up and working. The Organizer is using Chrome as their browser. A Member, "johns01", is logged in. This Member belongs to an Organization which is hosting an Event, "General Meeting" but isn't currently at the physical location specified by the Event.
Input: The following sequence is done:

1. Member clicks on Events.
2. Member clicks on the "Fishing for Dummies" Event.

3. Member clicks on the "I'm Here!" button on the Event page.

Expected Output: The system does not complete the request because it detects that the Member is not at the location specified by the Event. A message is communicated to the Member explaining precisely this. A Request object is created reflecting the while transaction, including its erroneous return.

7.2.4.4 SOS04-System-016-Rainy-08

Name: SOS04-System-016-Rainy-08
Purpose: Investigate the execution of the Use Case with inputs which are expected to create exceptions on the system.
Test set up: The SOS system is set up and working. The Organizer is using Chrome as their browser. A Member, "johns01", is logged in. This Member belongs to an Organization which is hosting an Event, "General Meeting" but their currently location is not available to the system.
Input: The following sequence is done:

1. Member clicks on Events.
2. Member clicks on the "Fishing for Dummies" Event.
3. Member clicks on the "I'm Here!" button on the Event page.

Expected Output: The system does not complete the request because the Member's current location is not defined. A message is communicated to the Member explaining precisely this. A Request object is created reflecting the while transaction, including its erroneous return.

7.2.5 Edit Profile

The following system test cases belong to the Edit Profile Use Case (SOS07).

7.2.5.1 SOS07-Security-017-Sunny-09

Name: SOS07-Security-017-Sunny-09
Purpose: Test to ensure that a Profile that has been previously created is able to have its email address changed.
Test set-up: There is already a user who has made a profile within the system. She wishes to edit her email address since she recently made a new one that is more secure.
Input:

1. So from profile page she will click on the 'edit profile' button.
2. A form will pop up that has information she had already inserted before.
3. She inserts the change into the email field, erasing "style@hotmail.com" and typing in "fab@gmail.com"
4. She will click the "submit" button.
5. She will enter her password.

Expected Output: The system will show a confirmation message to the User on the screen, also, the field for email address in the profile has been changed to fab@gmail.com

7.2.5.2 SOS07-Security-018-Sunny-10

Name: SOS07-Security-018-Sunny-10
Purpose: Test to ensure that a Profile that has been previously created is able to have the privacy of the account changed.

Test set-up: There is already a user who has made a profile within the system. This user wishes to change the privacy on his account because he does not want just anyone to look at his activities. He is paranoid and does not want unfamiliar people knowing about his business.
Input:

1. From profile page user clicks on 'edit profile' button
2. Form pops up to shown information that has been inserted before.
3. He clicks on the dropdown box labeled "privacy"
4. Changes the option from "public" to "private"
5. He will click on the "submit" button
6. He will enter his password

Expected Output: The system will show a confirmation message to the User on the screen, also, the field for privacy has been changed to private.

7.2.5.3 SOS07-Security-019-Rainy-09

Name: SOS07-Security-019-Rainy-09
Purpose: Test how the system will react to user attempting to edit their profile but instead cancels from the form.
Test set-up: User already has a profile on the account. He is thinking about making a change to his phone number because he is worried that he entered the wrong digits. He goes to the Profile Page.
Input:

1. Clicks on "edit profile" button.
2. Form pops up to shown information that has been inserted before.
3. User realizes that there is actually no mistake and he did enter his number correctly.
4. He presses the "x" button at the upper right corner of the form to close it.

Expected Output: No changes have occurred to profile information. User is back on profile page.

7.2.5.4 SOS07-Security-020-Rainy-010

Name: SOS07-Security-020-Rainy-010
Purpose: Test under assumption that user account has been hacked and unable to edit profile.
Test set-up: User has logged into her account, she wants to check her profile to see if any information needs to be updated. She is unaware that her account has been tampered with. She goes to the profile page.
Input:

1. Clicks on "edit profile" button.
2. She notices that the privacy is set to public and that is not right.
3. She clicks the dropdown box but nothing happens.
4. Drop down box is grayed out, she is unable to change her settings.
5. He presses the "x" button at the upper right corner of the form to close it.

Expected Output: Unable to do anything to correct her account the user will log out and call for technical support.

7.2.6 Create Organization

The following system test cases belong to the Create Organization Use Case (SOS16).

7.2.6.1 SOS16-System-021-Sunny-11

Name: SOS16-System-021-Sunny-11
Purpose: Ensure that a user can successfully create an organization on the website
Test set-up: User has logged into their account on the webpage. He is eager to create a new organization for those who have a hobby in playing music, He goes onto the Profile page
Input:

1. Clicks on the "Create Organization" button
2. A form will pop up in front of the screen to enter information
3. User enters the data into respective fields
    a. Organization Name = "XYZ"
    b. Description = "A club for those who have a passion for music"
    c. Request for Joining = "Be a current student in University"
    d. Privacy = "public"
4. User clicks on the "submit" button

Expected Output: The request is submitted and a pop up about the Organization will appear before the user. The user will gain ownership over the Organization and their account will be changed to that of an Organizer status.

7.2.6.2 SOS16-System-022-Sunny-12

Name: SOS16-System-022-Sunny-12
Purpose: Ensure that a user can successfully create an organization on the website
Test set-up: User is logged into her account, she wants to make an organization that is about having study groups. She goes onto her profile page.
Input:

1) User clicks on "Create Organization" button
2) A form will pop up in front of the screen to enter information
3) User enters data in respective fields
    a) Organization Name = "QRT"
    b) Description = "Lets study"
    c) Requirements for Joining = none
    d) Privacy = "public"
4) She then clicks on the "submit" button

Expected Output: The request is submitted and a pop up about the Organization will appear before the user. The user will gain ownership over the Organization and their account will be changed to that of an Organizer status.

7.2.6.3 SOS16-System-023-Rainy-11

Name: SOS16-System-023-Rainy-11
Purpose: Test to ensure that if user wants to make an Organization she must enter all required details.
Test set-up: User wants to create an Organization, she is new to the system and does not know how to use technology very well. She is logged into her account and tries to create the organization on the profile page.
Input:

1) She clicks on the "Create Organization" button
2) A form pops up to fill out required information

3) She enters the following data
    a) Organization Name = “My special club”
4) Then she clicks the “submit” button

Expected Output: A pop up message will appear in front of the user saying that some of the required fields have not been filled out. She will not be able to create the Organization so she must close the message and try again.

7.2.6.4 SOS16-System-024-Rainy-12

Name: SOS16-System-024-Rainy-12
Purpose: To test that a user cannot create an Organization if those same specifications already exist.
Test set-up: User wants to create a club named “Hot Singles”. He is interested in finding a girlfriend and wants to have an Organization where the men and women can meet and maybe set up some dates. He goes onto the profile page.
Input:

1) He clicks on “Create Organization” button
2) A form pops up in front of the screen to enter information
3) He enters the following data
    a) Organization Name = “Hot Singles”
    b) Description = “ Looking for a girlfriend/boyfriend”
    c) Requirement for joining = “you have to be hot!!!!!”
    d) Privacy = “public”
4) He clicks on “submit” button

Expected Output: The system will not create this organization and will send a message to user. There is an organization that already exists named “Hot Singles” so the name cannot be used. The user is left on Create Organization page.

7.2.7 Create Event

The following system test cases belong to the Create Event Use Case (SOS01).

7.2.7.1 SOS01-System-025-Sunny-13

Name: SOS01-System-025-Sunny-13
Purpose: To verify that the create event feature is working properly.
Test set-up: The user is logged into the system, belongs to an organization, and has event creation privileges. The user provides the information needed to create an event.
Input:

1. The user clicks on the “create event” button.
2. The user provides the following information.
    a. Event Name: Movie Night
    b. Event Date and Time: December 11th, 2019 8pm
    c. Event Location: GC 150, FIU MMC
    d. Event Description (Optional):
    e. Event Type: Normal
    f. Event Visibility: Visible
3. The user clicks the “submit” button.

Expected Output: The system will display a message stating that the event creation was successful.

7.2.7.2 SOS01-System-026-Sunny-14

Name: SOS01-System-026-Sunny-14
Purpose: To verify that the create event feature is working properly.
Test set-up: The user is logged into the system, belongs to an organization, and has event creation privileges. The user provides the information needed to schedule the event creation for a later date.
Input:

1. The user clicks on the "create event" button.
2. The user provides the following information.
    a. Event Name: Movie Night
    b. Event Date and Time: December 11th, 2019 8pm
    c. Event Location: GC 150, FIU MMC
    d. Event Description (Optional):
    e. Event Type: Normal
    f. Event Visibility: Visible
    g. Create On: 12/1/19
3. The user clicks the "submit" button.

Expected Output: The system will display a message stating that the event will be created on the date specified.

7.2.7.3 SOS01-System-027-Rainy-13

Name: SOS01-System-027-Rainy-13
Purpose: To verify that that unexpected actions are handled.
Test set-up: The user is logged into the system, belongs to an organization, and has event creation privileges. The user provides blank responses.
Input:

1. The user clicks on the "create event" button.
2. The user does not provide any information.
3. The user clicks the "submit" button.

Expected Output: The system will display a message prompting the user to enter information in the appropriate fields.

7.2.7.4 SOS01-System-028-Rainy-14

Name: SOS01-System-028-Rainy-14
Purpose: To verify that that unexpected actions are handled.
Test set-up: The user is logged into the system, belongs to an organization, and has event creation privileges. The user cancels the event creation.
Input:

1. The user clicks on the "create event" button.
2. The user may or may not provide any information.
3. The user clicks the "cancel" button.

Expected Output: The system will display a message stating that the operation was cancelled.

7.2.8 Login

The following system test cases belong to the Login Use Case (SOS21).

7.2.8.1 SOS31-System-029-Sunny-15

Name: SOS31-System-029-Sunny-15
Purpose: To verify that the login feature is working properly.
Test set-up: The user has an account and provides their login information.
Input:

1. The user clicks on the "login" button.
2. The user provides the following information.
    a. Email: panther567@fiu.edu
    b. Password: paws891
3. The user clicks the "submit" button.

Expected Output: The system will display a message stating that the user was logged in successfully.

7.2.8.2 SOS31-System-030-Sunny-16

Name: SOS31-System-030-Sunny-16
Purpose: To verify that the login feature is working properly.
Test set-up: The user has an account and the browser provides the user's saved login information.
Input:

1. The user clicks on the "login" button.
2. The browser generates the following information from a previous session.
    a. Email: panther567@fiu.edu
    b. Password: paws891
3. The user clicks the "submit" button.

Expected Output: The system will display a message stating that the user was logged in successfully.

7.2.8.3 SOS31-System-031-Rainy-15

Name: SOS31-System-031-Rainy-15
Purpose: To verify that that unexpected actions are handled.
Test set-up: The user has an account and provides the incorrect login information.
Input:

1. The user clicks on the "login" button.
2. The user provides the following information.
    a. Email: panther567@fiu.edu
    b. Password: paw45
3. The user clicks the "submit" button.

Expected Output: The system will display a message stating that the user's email and password combination did not match.

7.2.8.4 SOS31-System-032-Rainy-16

Name: SOS31-System-032-Rainy-16
Purpose: To verify that that unexpected actions are handled.

Test set-up: The user does not have an account and provides non-existent login information.
Input:

1. The user clicks on the “login” button.
2. The user provides the following information.
    a. Email: spartan547@outlook.com
    b. Password: bluenorange97
3. The user clicks the “submit” button.

Expected Output: The system will display a message stating that the user’s email and password combination does not exist.

7.2.9 Registration

The following system test cases belong to the Registration Use Case (SOS22).

7.2.9.1 SOS22-System-033-Sunny-17

Test Case ID: SOS22-System-033-Sunny-17
Purpose: Investigate normal execution of the Use Case with inputs which are similar to those expected by the system.
Test set up: The SOS system is set up and working. The User is using Chrome as their browser. The User has not logged in since they are not registered yet. This User wants to register for an account on the system. They are also on the login/registration page and register button is visible on the top of that page.
Input: The following sequence is done:

1. User clicks on Register button.
2. User enters their username.
3. User enters email.
4. User enters password.
5. User confirms password by retyping it.
6. User then clicks on the register/ok button.

Expected Output: The system completes the request without exceptions or errors. The system confirms the successful registration for the new User. Finally user is logged in automatically to their new account they just registered for on the system and they’ll be redirected to the SOS homepage.

7.2.9.2 SOS22-System-034-Sunny-18

Test Case ID: SOS22-System-034-Sunny-18
Purpose: Investigate normal execution of the Use Case with inputs which are similar to those expected by the system.
Test set up: The SOS system is set up and working. The User is using Safari as their browser. The User has not logged in since they are not registered yet. This User wants to register for an account on the system. They are also on the login/registration page and register button is visible on the top of that page.
Input: The following sequence is done:

1. User clicks on Register button.

2. User enters their email.
3. User enters username.
4. User enters password.
5. User confirms password by retyping it.
6. User then clicks on the register/ok button.

Expected Output: The system completes the request without exceptions or errors. The system confirms the successful registration for the new User. Finally user is logged in automatically to their new account they just registered for on the system and they'll be redirected to the SOS homepage.

7.2.9.3 SOS22-System-035-Rainy-17

Test Case ID: SOS22-System-035-Rainy-17
Purpose: Investigate the execution of the Use Case with inputs which are expected to create exceptions or errors on the system.
Test set up: The SOS system is set up and working. The User is using Chrome as their browser. The User has not logged in since they are not registered yet. This User wants to register for an account on the system. They are also on the login/registration page and register button is visible on the top of that page.
Input: The following sequence is done:

1. User clicks on Register button.
2. User enters their email.
3. User enters username.
4. User enters password.
5. User confirms password by retyping it (incorrect password in this case).
6. User then clicks on the register/ok button.

Expected Output: The system cannot complete the request without exceptions or errors. System will not be able to register the new User, since they did not enter the same password in the confirmation field. Hence system will respond with a message saying that proper credentials should be entered.

7.2.9.4 SOS22-System-036-Rainy-18

Test Case ID: SOS22-System-036-Rainy-18
Purpose: Investigate the execution of the Use Case with inputs which are expected to create exceptions or errors on the system.
Test set up: The SOS system is set up and working. The User is using Chrome as their browser. The User has not logged in since they are not registered yet. This User wants to register for an account on the system. They are also on the login/registration page and register button is visible on the top of that page.
Input: The following sequence is done:

1. User clicks on Register button.
2. User enters their email.
3. User enters password.
4. User confirms password by retyping it.

5. User then clicks on the register/ok button.

Expected Output: The system cannot complete the request without exceptions or errors. System will not be able to register the new User, since they left the user name field in the registration form blank. Hence system will respond with a message saying that proper credentials should be entered.

7.2.10 Cancel an Event

The following system test cases belong to the Cancel an Event Use Case (SOS17).

7.2.10.1 SOS17-System-037-Sunny-19

Test Case ID: SOS17-System-037-Sunny-19
Purpose: Investigate normal execution of the Use Case with inputs which are similar to those expected by the system.
Test set up: The SOS system is set up and working. The User is using Chrome as their browser. The Organizer is already logged in the SOS. The Organizer belongs to an organization on the SOS. There exists at least one on going event on the system for the organization which organizer belongs too, and then the organizer wants to cancel an event.
Input: The following sequence is done:

1. Organizer clicks on the event they are willing to cancel.
2. Organizer clicks on cancel event button in the Event Description view.
3. Organizer clicks on confirm button in the validation message.

Expected Output: The system completes the request without exceptions or errors. The Organizer will be shown a message saying the cancellation of the event was successful. Finally the event is removed from being viewed. All of the subscribed users to that event will be notified of the cancellation.

7.2.10.2 SOS17-System-038-Sunny-20

Test Case ID: SOS17-System-038-Sunny-20
Purpose: Investigate normal execution of the Use Case with inputs which are similar to those expected by the system.
Test set up: The SOS system is set up and working. The User is using Safari as their browser. The Organizer is already logged in the SOS. The Organizer belongs to an organization on the SOS. There are multiple ongoing events for the organization.
Input: The following sequence is done:

1. Organizer clicks on the event they are willing to cancel.
2. Organizer clicks on cancel event button in the Event Description view.
3. Organizer clicks on confirm button in the validation message.

Expected Output: The system completes the request without exceptions or errors. The Organizer will be shown a message saying the cancellation of the event was successful. Finally the event is removed from being viewed. All of the subscribed users to that event will be notified of the cancellation.

7.2.10.3 SOS17-System-039-Rainy-19

Test Case ID: SOS17-System-039-Rainy-19
Purpose: Investigate the execution of the Use Case with inputs which are expected to create exceptions or errors on the system.
Test set up: The SOS system is set up and working. The User is using Safari as their browser. The Organizer is already logged in the SOS. The Organizer belongs to an organization on the SOS. There have been an old session which a certain event exists on the Event Description view, however it is deleted in a newer session but the old session is not refreshed yet. Following input is applied in the older unrefreshed session.
Input: The following sequence is done:

1. Organizer clicks on cancel event button in the Event Description view.
2. Organizer clicks on confirm button in the validation message.

Expected Output: The system cannot complete the request without exceptions or errors. System will not be able to cancel the event because it has been already canceled before and could not be found in active events anymore. Then they will be redirected to the Events list page.

7.2.10.4 SOS17-System-040-Rainy-20

Test Case ID: SOS17-System-040-Rainy-20
Purpose: Investigate the execution of the Use Case with inputs which are expected to create exceptions or errors on the system.
Test set up: The SOS system is set up and working. The User is using Firefox as their browser. The Organizer is already logged in the SOS. The Organizer belongs to an organization on the SOS. There are multiple ongoing events for the organization.
Input: The following sequence is done:

1. Organizer clicks on the event they are willing to cancel.
2. Organizer clicks on cancel event button in the Event Description view.
3. Organizer clicks on reject button in the validation message.

Expected Output: The system cannot complete the request without exceptions or errors. System will not be able to cancel the event because the Organizer rejected the validation message.

7.3 Evaluation of Tests

This section contains the results of all tests (unit tests & system tests) in a table format, as well as possible soultions and imporvements regarding each test case. Code coverage for unit tests are also included in the second part of this section.

7.3.1 Test Results:

Table 6 shows the results of unit and system tests and possible fixes and improvements:

Test Case ID	Pass/Fail	Fix
SOS-UnitTest-001	Pass	N/A
SOS-UnitTest-002	Pass	N/A

SOS-UnitTest-003	Pass	N/A
SOS-UnitTest-004	Pass	N/A
SOS-UnitTest-005	Fail	N/A
SOS-UnitTest-006	Pass	N/A
SOS-UnitTest-007	Fail	N/A
SOS-UnitTest-008	Pass	N/A
SOS-UnitTest-009	Pass	N/A
SOS-UnitTest-010	Pass	N/A
SOS-UnitTest-011	Pass	N/A
SOS-UnitTest-012	Pass	N/A
SOS-UnitTest-013	Pass	N/A
SOS-UnitTest-014	Pass	N/A
SOS-UnitTest-015	Pass	N/A
SOS-UnitTest-016	Pass	N/A
SOS32-System-001-Sunny-01	Pass	N/A
SOS32-System-002-Sunny-02	Pass	N/A
SOS32-System-003-Rainy-01	Pass	N/A
SOS32-System-004-Rainy-02	Pass	N/A
SOS07-Security-017-Sunny-09	Fail	Submit form data has to be fixed
SOS07-Security-018-Sunny-10	Fail	Submit form data has to be fixed
SOS07-Security-019-Rainy-09	Pass	N/A
SOS07-Security-020-Rainy-10	Pass	N/A
SOS16-System-021-Sunny-11	Pass	N/A
SOS16-System-022-Sunny-12	Pass	N/A
SOS16-System-023-Rainy-11	Fail	Input validation for the create organization form
SOS16-System-024-Rainy-12	Pass	Showing a message about duplicate organization*

SOS01-System-025-Sunny-13	Fail	Time picker message format should be modified to match DB entries
SOS01-System-026-Sunny-14	Fail	Time picker message format should be modified to match DB entries
SOS01-System-027-Rainy-13	Pass	Showing a message about blank fields*
SOS01-System-028-Rainy-14	Pass	N/A
SOS31-System-029-Sunny-15	Pass	N/A
SOS31-System-030-Sunny-16	Pass	N/A
SOS31-System-031-Rainy-15	Pass	Showing an error message*
SOS31-System-032-Rainy-16	Pass	Showing an error message*
SOS22-System-033-Sunny-17	Pass	Does not automatically login right after registration*
SOS22-System-034-Sunny-18	Pass	Does not automatically login right after registration*
SOS22-System-035-Rainy-17	Pass	Showing an error message*
SOS22-System-036-Rainy-18	Pass	Showing an error message*
SOS17-System-037-Sunny-19	Pass	N/A
SOS17-System-038-Sunny-20	Pass	N/A
SOS17-System-039-Rainy-19	Pass	N/A
SOS17-System-040-Rainy-20	Pass	N/A
SOS02-System-009-Sunny-05	Pass	N/A
SOS02-System-010-Sunny-06	Pass	N/A
SOS02-System-011-Rainy-05	Pass	N/A
SOS02-System-012-Rainy-06	Pass	N/A

SOS04-System-013-Sunny-07	Pass	N/A
SOS04-System-014-Sunny-08	Pass	N/A
SOS04-System-015-Rainy-07	Pass	N/A
SOS04-System-016-Rainy-08	Pass	N/A
SOS10-System-005-Sunny-03	Pass	N/A
SOS10-System-006-Sunny-04	Pass	N/A
SOS10-System-007-Rainy-03	Pass	N/A
SOS10-System-008-Rainy-04	Pass	N/A

Table 6: Test Cases Results

7.3.2 Code Coverage.

Table 7 shows the code coverage for the unit and system tests.

Type of Testing	# of Tests Cases	# of Bugs	Code Coverage
Unit Testing	16	2	28.6%
System Testing	40	5	N/A

Table 7: Code Coverage for the whole SOS System.

The code coverage results for the Unit Tests (obtained using Eclipse's code coverage tools) can be seen in Figure 37.

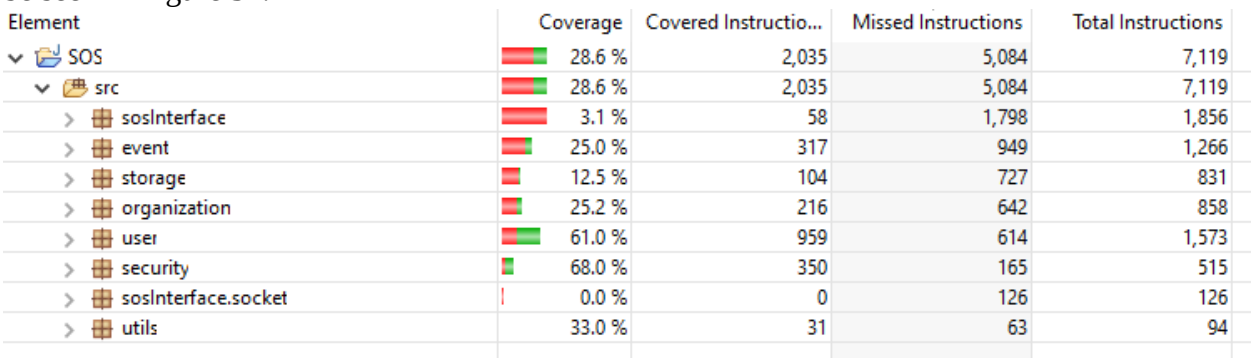

Figure 37: Code Coverage for SOS Backend.

8 Glossary

- **Scenario**, a scene that illustrates some interactions of the proposed system.
- **Static Model**, a model which does not depend on elements of time.
- **Dynamic Model**, a model which depends on or contains elements of time, especially allowing interactions between entities over time.
- **Gantt Chart,** a bar chart where the x-axis is time and the y-axis is the different tasks, and the duration of each task is represented by the length of a bar.
- **Unified Software Development Model**, …
- **Sequence Diagram,** an interaction diagram which focus on the time-ordering of messages and interactions.
- **Use Case Diagram,** a diagram that shows a set of use cases and actors; and their relations.
- **SOS,** Student Organization System.
- **Object Diagram,** a diagram that models the instances of things contained in a class diagram, i.e., a set of objects and their relationships at a point in time.
- **Class Diagram,** a UML diagram containing a representation
- **Attribute,** a variable on a UML class.
- **Operation,** a function on a UML class indicating an action.
- **Role,** a set of technical and managerial tasks that are expected from a participant or a team.
- **Activity,** a set of tasks performed towards a specific purpose.
- **Task,** an atomic unit of work that can be managed and that consumes resources.
- **Milestone,** end-point of a software process activity.
- **Deliverable,** a work product for the client.
- **Notation,** a graphical or textual set of rules representing a model.
- **Method,** a repeatable technique for solving a specific problem.
- **Methodology,** a collection of methods for solving a class of problems.
- **Use Case,** a sequence of events describing all possible actions between actors and the system for a given piece of functionality.
- **Actors,** the roles interacting with the system such as end-users and other computer systems.

9 Approval Page:

Approval Page of System Requirements Document of

Student Organization System

Member Signatures

Armando J. Ochoa	12/03/2019
Member Signature	Date
Yovanni Jones	12/03/2019
Member Signature	Date
M.Kian Maroofi	12/03/2019
Member Signature	Date
Teriq Douglas	12/03/2019
Member Signature	Date
Anthony Sanchez-Ayra	12/03/2019
Member Signature	Date

10 References

Campus Lab. (2019). *Panther Connect*. Retrieved from Panther Connect: https://fiu.campuslabs.com/engage/

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The Unified Software Development Process.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

11 Appendices

11.1 Appendix A – Project Schedule

The scheduled tasks are contained in Section 3.3. The Gantt chart of the schedule is below, in Figure 38

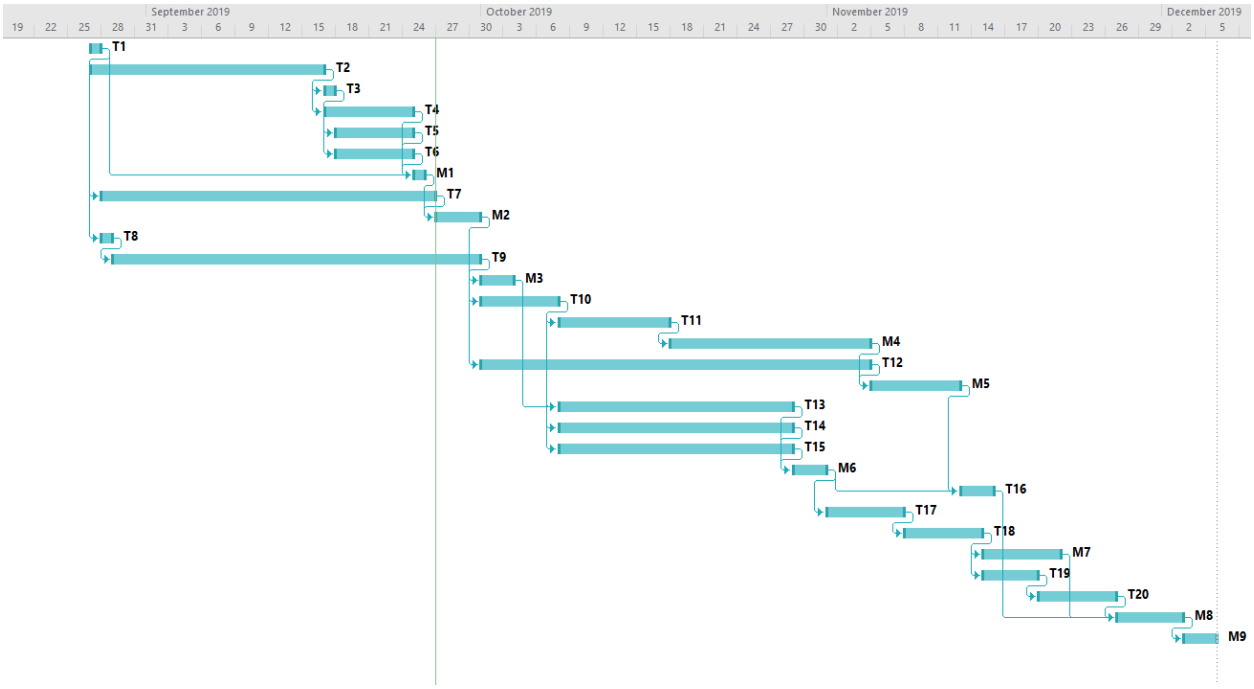

Figure 38: Gantt chart for the full project schedule.

11.2 Appendix B – Use Cases

Each of the following subsections presents a Use Case describing a feature of the SOS system. These refer to the actors involved (see Section 4.2) and describe a step-by-step interaction between these actors and the system. They also include support information as well as usability, reliability, performance, supportability, and implementation constraints.

11.2.1 Create Event

Use Case ID: SOS01 - Create Event

Use Case Level: User Goal

Details:

- **Actor:** Organizer
- **Pre-conditions:**
    1. Organizer has successfully logged onto the system.
    2. Organizer is assigned to an Organization.
    3. Organizer has Event Creation privileges
- **Description:**
    1. Use case begins when Organizer clicks on **Create Event** on the administration page of their organization.
    2. The system shall prompt the Organizer with an Event Creation form, which shall present them with a template for data entry.
    3. The Organizer shall enter the following data:
        - **Event Name**
        - **Event Date and Time**
        - **Event Location**
        - **Event Description** (Optional)
        - **Event Type** (Defaults to Normal Event)
        - **Event Visibility** (Defaults to Visible)
    4. The Organizer shall complete the Event Creation by selecting the **publish** button.
    5. The system shall notify the Organizer that the event was published correctly.
    6. Use case ends when the system receives the Event specifications, generates a **unique event id** and publishes the Event according to the given specifications.
- **Relevant requirements:**

    None

- **Post-conditions:**

    1. An event has been published by the Organizer representing the Organization according to the specifications given.

Alternative Courses of Action

1. In step D.4, the Organizer has the option to **cancel** the Event Creation.
2. In step D.4, the Organizer has the option to **schedule** the Event Creation for a future date.
3. In step D.4, the Organizer has the option to **save without publishing** the Event Creation to complete at a later date.
4. In step D.5, if any of the required fields are blank, the system shall notify the Organizer and request an entry to the appropriate fields.

Extensions:

1. SOS21 – Avoid Time Conflicting Events

Exceptions:

1. The event database is not active.
2. The event creation view is not active.

Concurrent Uses:

None

Related Use Cases:

None

Decision Support

Frequency: On average 3 Events are created per Organization weekly.

Criticality: High. The most basic and central activity of the whole system is Event Creation.

Risk: Medium. Implementation does not require any complex specialized knowledge.

Constraints:

- Usability
    a) No previous training or knowledge.
    b) Tutorial or Help frame should be provided.
    c) Organizer should take less than 10 minutes to create an event.

- Reliability
    a) Mean Time to Failure – 5% failure monthly is acceptable.
    b) Availability
        - Downtime for Login Back-up – 30 minutes in a 24-hour period.
        - Downtime for Maintenance – 1 hour in a 2 weeks period.
- Performance
    a) The form should be sent and saved within 10 seconds.
    b) The system should be able to handle 50 requests in 1 minute.
- Supportability
    a) The Event Creation should be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Armando J. Ochoa

Initiation date: 09/01/2019

Date last modified: 09/15/2019

11.2.2 Grant Organizer Role

Use Case ID: SOS2 – Grant Organizer Role

Use Case Level: User Goal

Details:

- **Actor:** Organizer
- **Pre-conditions:**
    1. Target Member belongs to the current organization.
    2. Target Member does not have Organizer status on the current organization.
    3. Organizer has power to give other people Organizer status.
- **Description:**
    1. Use case begins when the Organizer clicks on the **Add Organizer** tab on the organization management view.

2. The system shall prompt the Organizer with an **Invitation Menu**, which shall present them with a template for data entry.
3. The Organizer shall enter the following data:
    - **Member ID** (Either a name, or selectable from a drop-down menu with the list of organization members).
    - **Organizer Title** (Optional)
    - **Powers and Privileges** (From a list of pre-set privileges).
4. The Organizer shall finish adding an organizer by selecting the **complete** button.
5. The system shall notify the Organizer that the Member's privilege and status has been changed correctly.
6. Use case ends when the system changes the Member's status in its database and the Member has been notified.

- **Post-conditions:**
    1. The status of the target Member has been changed, and he or she has received new privileges on the given organization.
    2. The list of Organizers in the Organization has been updated.
    3. The Member has been notified of the update.

Alternative Courses of Action

1. In step D.3, if the Organizer attempts to set a privilege that they themselves do not have, then the system shall notify them that they lack the required privileges (e.g., an Organizer without Event Creation privileges cannot invite another Organizer with Event Creation privileges).
2. In step D.4, the Organizer has the option to **cancel** the invitation.
3. In step D.5, if any of the required fields are blank, the system shall notify the Organizer and request an entry to the appropriate fields.

Extensions:

None

Exceptions:

1. Incorrect input in step D.3 (such as a non-existent Member ID) shall cause an exception and trigger a notification to the Organizer.

Concurrent Uses:

None

Related Use Cases:

None

Decision Support

Frequency: On average, 2 or 3 times per month per organization.

Criticality: High. This is basic element of the system and is required for good usability.

Risk: Medium. Implementation does not require any complex specialized knowledge.

Constraints:

- Usability
    a) No previous training or knowledge.
    b) Tutorial or Help frame should be provided.
    c) Organizer should take less than 10 minutes to complete the invitation.
- Reliability
    a) Mean Time to Failure – 1% failure yearly is acceptable.
    b) Availability – 30 minutes in a 24-hour period for backup and maintenance.
- Performance
    a) Privilege Checks should be done within 2 seconds.
    b) The system should handle 20 privilege checks in 1 minute.
- Supportability
    a) Should be supported by all browsers.
- Implementation
    a) Using Java-based software for back-end.

Modification History

Owner: Armando J. Ochoa

Initiation date: 09/01/2019

Date last modified: 09/15/2019

11.2.3 Earn Points by Attending an Event

Use Case ID: SOS3 – Earn Points by Attending an Event

Use Case Level: User Goal

Details:

- **Actor:** Member
- **Pre-conditions:**
    1. Member has successfully logged onto the system.
    2. Member belongs to an organization.
    3. Member is participating in the organization's points ranking.
- **Description:**
    1. Use case begins when the Member is marked as attending an Event.
    2. The system shall check the Event log to see if the Member is already marked as having attended in this Event.
    3. The system shall note the Member's participation on the Event log.
    4. The system shall note the Member's participation on the Member's page.
    5. The system shall award the Member a certain amount of points, as defined by the Event specifications.
    6. The system shall update the Organization's ranking to reflect the new points.
    7. The case ends once the system notifies the Member that his or her point ranking has changed, by how much, and what his or her new ranking on the Organization is.
- **Relevant requirements:**

    None
- **Post-conditions:**
    1. The Event log has been updated with the Member's participation.
    2. The Member's points towards the organization has been updated.
    3. The Organization ranking has been updated with the Member's new points.
- **Alternative Courses of Action:**
    1. In steps D.2, if the Member's participation is already in the Event log, then, the following steps are ignored. The Member is notified that he or she has already participated in the Event.

Extensions:

None.

Exceptions:

1. The Event log, Organization, and Ranking are not accessible or active. In which case the Member shall be notified of the error and told his or her points will not be counted.

Concurrent Uses:

None

Related Use Cases:

SOS4 – Attending an Event
SOS9 – Member Ranking

Decision Support

Frequency: On average, 15-30 participants per Event, with an average of 3 Events per Organization created weekly.

Criticality: Medium. The point and ranking systems are an optional functionality that not everybody will use, and that is subordinate to other systems.

Risk: Medium. Implementation does not require any complex specialized knowledge.

Constraints:

- Usability
    a) No previous training or knowledge. The system should respond without user interaction after the attendance is completed.
- Reliability
    a) Meant Time to Failure: 5% failure monthly is acceptable.
- Performance
    a) The system should be able to handle 20 requests in 1 minute.
    b) The system should update the Event, Member, and Organization logs within 2 seconds.
- Supportability
    a) Point earning should be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end, as well as SQL for database management.

Modification History

Owner: Armando J. Ochoa

Initiation date: 09/01/2019

Date last modified: 09/15/2019

11.2.4 Attending an Event

Use Case ID: SOS04 - Attending an Event

Use Case Level: User Goal

Details:

- **Actor:** Member
- **Pre-conditions:**
    1. Member has an account in our application.
    2. Member is successfully logged into the application.
    3. Member is part of an organization and is attending an event hosted by said organization.
- **Description:**

  Trigger:
    1. Use case begins when member clicks on the events tab.
    2. The system shall provide the member with a sorted list of events that the user has signed up for.
    3. The member will click on the event that they are currently attending.
    4. The system shall provide the member with a description of the event as well as a button that says, “I’m here!”
    5. The user shall click on the “I’m here” button.
    6. The system shall process the request for the click.
    7. Use case ends when the system notifies the user that their attendance at the event was noted.
- **Relevant requirements:**

  None
- **Post-conditions:**
    1. The attendance request is saved in the system, along with arrival time.
    2. The member is awarded a certain amount of points for attending the event.

Alternative Courses of Action:

1. In step D.10 the "I'm here" button will only appear if the user is at the location where the event is occurring.
2. In step D.8 the sorted list provided by to the user can be sorted by date the event will take place on or by organization name.

Exceptions:

1. If the member tries to click the I'm here button 15 minutes before the event is ending, they will not get credit for attending the event.

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average 100 attendance requests are made weekly by the organization leader.

Criticality: High. Allows the member to notify their organization that they are active in their organization.

Risk: High. Implementing this use case requires web-based technology and GPS tracking.

Constraints:

- Usability:
  a) No previous training required.
  b) On average the user should take 2 minutes to complete the notification request to the system.
- Reliability
  a) Mean time to failure – 5% failures for every month of operation is acceptable.
  b) Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
- Performance
  a) Request should be sent and saved within 6 seconds.
  b) System should be able to handle 1000 request in 1 minute.
- Supportability
  a) The Event Creation should be supported by Chrome, Mozilla, and IE.

- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Anthony Sanchez-Ayra

Initiation date: 09/04/2019

Date last modified: 09/15/2019

11.2.5 Ensure User Access

Use Case ID: SOS05 – Ensure User Access

Use Case Level: Security.

Details:

- **Actor:** User
- **Pre-conditions:**
    1. User has privileged access to an Event, Organization, or Member Profile page.
    2. User is logged in.
- **Description:**
    1. Use case begins when the User clicks on an Event, Organization or Member Profile page.
    2. The system requests the User status and privileges.
    3. The system checks that status and privileges against the set requirements to see the Event, Organization, or Member Profile.
    4. The case ends when the privileged Event, Organization, or Member Profile view is presented to the User.
- **Relevant requirements:**

    None
- **Post-conditions:**
    1. The User's view has been changed to the appropriate Event, Organization, or Member Profile view. Privileged view might include editing, deleting, or seeing privileged information.
- **Alternative Courses of Action:**

1. In step D.3, if the User status and privileges are not adequate to view the Event, Organization, or Member Profile page, then they are denied access or presented with a non-privileged view.

Extensions:

None.

Exceptions:

None.

Concurrent Uses:

None

Related Use Cases:

None

Decision Support

Frequency: On average, 20 attempts per day.

Criticality: High. The system should ensure correct access and privileges.

Risk: Medium. This is a standard security measure that does not require a lot of work to implement.

Constraints:

- Usability
    a) User must be aware of their privileges and what actions those privileges permit.
- Reliability
    a) Mean Time to Failure – 1% failure yearly is acceptable.
    b) Availability – 30 minutes in a 24-hour period for backup and maintenance.
- Performance
    a) Privilege Checks should be done within 2 seconds.
    b) The system should handle 20 privilege checks in 1 minute.
- Supportability
    a) Should be supported by all browsers.
- Implementation
    a) Using Java-based software for back-end.

Modification History

Owner: Armando J. Ochoa

Initiation date: 09/01/2019

Date last modified: 09/15/2019

11.2.6 Ensure User Profile Privacy

Use Case ID: SOS6 – Ensure User Profile Privacy

Use Case Level: Security.

Details:

- **Actor:** User
- **Pre-conditions:**
    1. The target User has profile information set to private or with restricted access.
- **Description:**
    1. Use case begins when the User attempts to view the private information belonging to the target User (e.g., a private feed, or a private membership, or ranking).
    2. The system shall check the target User's privacy settings.
    3. The system shall check the User's privileges.
    4. The system shall check the User against the target User's whitelist.
    5. The case ends when the system rejects the User and present him or her with a standard page indicating that the page is private.
- **Relevant requirements:**

    None
- **Post-conditions:**
    1. The system has presented the User with an adequate view of the profile.
    2. The system has logged the Misuser's attempt to see the target Member's data.
- **Alternative Courses of Action:**
    1. In step D.2, if the privacy settings are not **private**, then system shall provide access.
    2. In step D.3., if the User privileges allow it, then the system shall give access (i.e., the User is an **admin** or has similar privileges).
    3. In step D.4, if the User is in the target User's whitelist, then the system shall provide them access.

Extensions:

None.

Exceptions:

None.

Concurrent Uses:

None

Related Use Cases:

SOS7 – Edit Profile

Decision Support

Frequency: On average, 20 attempts per day.

Criticality: Medium. The system should not allow Misusers to easily access non-privileged pages, but implementing private Member, Organization, and Event pages is a secondary objective to the main functionality of the system.

Risk: Medium. This is a standard security measure that does not require a lot of work to implement.

Constraints:

- Usability
    a) User must be aware of their privileges and what actions those privileges permit.
- Reliability
    a) Mean Time to Failure – 1% failure yearly is acceptable.
    b) Availability – 30 minutes in a 24-hour period for backup and maintenance.
- Performance
    a) Privilege Checks should be done within 2 seconds.
    b) The system should handle 20 privilege checks in 1 minute.
- Supportability
    a) Should be supported by all browsers.
- Implementation
    a) Using Java-based software for back-end.

Modification History

Owner: Armando J. Ochoa

Initiation date: 09/01/2019

Date last modified: 09/15/2019

11.2.7 Edit Profile

Use Case ID: SOS7 – Edit Profile

Use Case Level: Security

Details:

- **Actor:** User
- **Pre-conditions:**
    1. User have already signed up.
    2. User is currently at their profile page.
- **Description:**
    1. Use case begins when user clicks on the edit profile button.
    2. The system then will retrieve current user data by contacting the data storage and send the data back to the front-end.
    3. The page shall display the retrieved data in an input form which will allow the user to modify the data in the edit profile form:
        - Email
        - Phone number
        - Privacy
        - Date Of Birth
    4. The user inputs the modified data and clicks on the submit button.
    5. The system shall ask the user for their password.
    6. The user inputs their password and clicks confirm.
    7. The system shall transmit the modified data to the data storage.
    8. The case ends when there is a confirmation message.
- **Relevant requirements:**

  None.
- **Post-conditions:**
    1. User information in the datastore has updated values.
    2. Profile page has been updated with the updated values.

Alternative Courses of Action:

1. In step D.4, it is possible that the user closes the input form without clicking the submit button. In that case system shall not change the current user information.

Extensions:

None.

Exceptions:

None.

Concurrent Uses:

None

Related Use Cases:

SOS6 – Ensure User Profile Privacy

Decision Support

Frequency: On average, 20 Users will change their privacy settings on a given week.

Criticality: Low. User-set privacy is a secondary feature of the system.

Risk: Medium. This does not require any complex background knowledge except for some basic knowledge about access control.

Constraints:

- Usability
    a) User will take about 20 seconds to find and use this piece of functionality.
- Reliability
    a) Mean Time to Failure – 5% failure monthly is acceptable.
    b) Availability
        - Downtime for Login Back-up – 30 minutes in a 24-hour period.
        - Downtime for Maintenance – 1 hour in a 2 weeks period.
- Performance
    a) Privilege Checks should be done within 2 seconds.
    b) The system should handle 20 privilege checks in 1 minute.
- Supportability
    a) Should be supported by all browsers.
- Implementation
    a) Using Java-based software for back-end.

Modification History

Owner: Kian Maroofi

Initiation date: 09/10/2019

Date last modified: 09/27/2019

11.2.8 Sharing

Use Case ID: SOS08 - Sharing

Use Case Level: User Goal

Details:

- **Actor:** Member
- **Pre-conditions:**
    1. Member has successfully logged onto the system.
- **Description:**
    1. Use case begins when clicks on the **Share** link on an Event or Organization.
    2. The system shall prompt a menu with several sharing options, including:
        - Share with Other Member
        - Share with Facebook
        - Share with Twitter
        - Share with Email
        - Copy URL to Clipboard
    3. The user can decide how to share the Event or Organization by clicking on the corresponding choice.
    4. The system shares the Event or Organization.
    5. The case ends once the system notifies the Member that it has shared the Event or Organization according to his or her choice.
- **Relevant requirements:**

    None
- **Post-conditions:**

    None
- **Alternative Courses of Action:**
    1. In step D.3, the Member can click on **cancel** or outside of the menu to cancel the sharing.
    2. In step D.3, if the Member choose to Share with Other Member, then the system shall prompt another menu asking for the recipient User's username.

Extensions:

None.

Exceptions:

None.

Concurrent Uses:

None

Related Use Cases:

None

Decision Support

Frequency: On average, events will be shared 20 to 30 times per week.

Criticality: Low. Not an important feature.

Risk: Low. Facebook, Twitter, and Email sharing are easy to implement using ready-made widgets.

Constraints:

- Usability
    a) No previous training or knowledge.
- Reliability
    a) Meant Time to Failure: 5% failure monthly is acceptable.
- Performance
    a) The system should be able to handle 20 requests in 1 minute.
    b) Sharing should happen instantly.
- Supportability
    a) Point earning should be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end, as well as SQL for database management.

Modification History

Owner: Armando J. Ochoa

Initiation date: 09/01/2019

Date last modified: 09/01/2019

11.2.9 Member Ranking

Use Case ID: SOS9 – Member Ranking

Use Case Level: User Goal

Details:

- **Actor:** Member
- **Pre-conditions:**
    1. Member belongs to at least one Organization.
    2. Member enabled access to their current location via GPS.
    3. They have attended Events that gives them scores.
- **Description:**
    1. Use case begins whenever the Member is marked as attending an Event and earns points because of it.
    2. The system shall store the Member's point total in a database, together with the Member's information.
    3. The system shall rank the Member and all other members of his Organization based on their point score. This rank and the point total of all the members of an Organization shall be linked to in the Organization's page.
    4. The case ends when the rankings are updated and redisplayed in the Organization's page.
- **Relevant requirements:**

  None
- **Post-conditions:**

  None.

Alternative Courses of Action:

None

Extensions:

None.

Exceptions:

None.

Concurrent Uses:

None

Related Use Cases:

SOS3 – Earn Points by Attending and Event

Decision Support

Frequency: On Average, 30 members per Organization will be reporting attendance to Events

Criticality: Medium. The point and ranking systems are an optional functionality that not everybody will use, and that is subordinate to other systems.

Risk: Medium. Implementation requires specialized knowledge, but GPS and Geolocation Services are available in most web browsers (Desktop and Mobile).

Constraints:

- Usability
    a) User must be aware of their privileges and what actions those privileges permit.
- Reliability
    a) Mean Time to Failure – 1% failure yearly is acceptable.
    b) Availability – 30 minutes in a 24-hour period for backup and maintenance.
- Performance
    a) Privilege Checks should be done within 2 seconds.
    b) The system should handle 20 privilege checks in 1 minute.
- Supportability
    a) Should be supported by all browsers.
- Implementation
    a) Using Java-based software for back-end.

Modification History

Owner: Kian Maroofi

Initiation date: 09/10/2019

Date last modified: 09/15/2019

11.2.10Access Events by Location
Use Case ID: SOS10 – Access Events by Location

Use Case Level: User Goal

Details:

- **Actor:** User
- **Pre-conditions:**
    1. User is logged into the system.
- **Description:**
    1. Use case begins when the User goes to the Events page or the Home page on the website.
    2. The webpage shall ask for accessing to the current location of the User by GPS.
    3. The system shall verify that User gave access to their location.
    4. The system shall find events within a defined proximity range of the User's location.
    5. The system shall update the Event map component to center on the User's location.
    6. The case ends when the system modifies the Event feed to prioritize Events within range of the User's location, and when the Event map component is updated to the User's location.
- **Relevant requirements:**

    None
- **Post-conditions:**
    1. The User's location is tracked on the system, and several Events are marked as within range.
    2. The Map component is updated to center on the User's location.
- **Alternative Courses of Action:**
    1. In step D.2, if the User has agreed to share location before, or if it has a permanent flag to share location in his or her profile, then it this step is ignored, and the system jumps directly to D.4
    2. In step D.3, if the User declines access, then the system shall ignore User location when presenting the Events.
    3. In step D.4, if location is not enabled, the system shall present all Events of the Organization.
    4. In step D.5, if location is not enabled, the system shall center on a system-wide default position.

Extensions:

None.

Exceptions:

None.

Concurrent Uses:

None

Related Use Cases:

None

Decision Support

Frequency: On average, users access the Home and Event pages 5 to 10 times daily.

Criticality: Medium, geolocation of events is an optional functionality that not everybody will use, and that is subordinate to other systems.

Risk: Medium. Medium. Implementation requires specialized knowledge, but GPS and Geolocation Services are available in most web browsers (Desktop and Mobile).

Constraints:

- Usability
    a) User must be aware of their privileges and what actions those privileges permit.
- Reliability
    a) Mean Time to Failure – 1% failure yearly is acceptable.
    b) Availability – 30 minutes in a 24-hour period for backup and maintenance.
- Performance
    a) Privilege Checks should be done within 2 seconds.
    b) The system should handle 20 privilege checks in 1 minute.
- Supportability
    a) Should be supported by all browsers.
- Implementation
    a) Using Java-based software for back-end.

Modification History

Owner: Kian Maroofi

Initiation date: 09/10/2019

Date last modified: 09/15/2019

11.2.11Score System

Use Case ID: SOS22 – Score System

Use Case Level: User Goal

Details:

- **Actor:** Organizer
- **Pre-conditions:**
    1. Organizer have already posted an event.
    2. Users (Members & Guests) enabled access to their current location via GPS.
- **Description:**
    1. Use case begins when the Organizer has posted a new event on the platform.
    2. The system shall provide an input feature such that provides the organizer a score definition system.
    3. The score that each attendee earns shall able to be defined by the organizer at the time of creating the event.
    4. The score must be selected from the following set which depends on the importance of the event which organizer defines. Score set is 5, 10, 15, 20, 25.
    5. The case ends when the event is created by the organizer.
- **Relevant requirements:**

    GPS and Geolocation Services available in most web browsers (Desktop and Mobile).
- **Post-conditions:** None.
- **Alternative Courses of Action:**

Extensions:

None.

Exceptions:

None.

Concurrent Uses:

None

Related Use Cases:

SOS4 – Attending an Event

SOS9 – Member Ranking

Decision Support

Frequency: On Average, 30 members per Organization will be reporting attendance to Events

Criticality: Medium. The point and ranking systems are an optional functionality that not everybody will use, and that is subordinate to other systems.

Risk: Medium. Does not require specialized knowledge.

Constraints:

- Usability
    a) User must be aware of their privileges and what actions those privileges permit.
- Reliability
    a) Mean Time to Failure – 1% failure yearly is acceptable.
    b) Availability – 30 minutes in a 24-hour period for backup and maintenance.
- Performance
    a) Privilege Checks should be done within 2 seconds.
    b) The system should handle 20 privilege checks in 1 minute.
- Supportability
    a) Should be supported by all browsers.
- Implementation
    a) Using Java-based software for back-end.

Modification History

Owner: Kian Maroofi

Initiation date: 09/10/2019

Date last modified: 09/22/2019

11.2.12Set Up Two Factor Authentication (2FA)
Use Case ID: SOS12 – Set Up Two Factor Authentication (2FA)

Use Case Level: Security.

Details:

- **Actor:** User
- **Pre-conditions:**
    1. User have made an account on the web app already, and is not logged in.
- **Description:**
    1. Use case begins when the User clicks on **Enable 2 Factor Authentication** in their profile, under the security tab.
    2. The system shall generate a 2FA seed and save it to its database.

3. The system shall ask the User to connect either Google Authenticator or such services using the generated seed.
4. The system checks that authenticator service is successfully connected to their account on the website by asking for a generated 2FA code on the authenticator service.
5. The case ends when the system confirms the link to the authentication service and notifies the User that 2FA has been enabled.

- **Relevant requirements:**

  None

- **Post-conditions:**

  1. The User needs to provide 2FA generated codes every time they are trying to log in to their account on the website.
  2. The 2FA Seed for the User is stored in the system's database.
  3. 2FA Authentication is marked as Enabled in the User's profile.

- **Alternative Courses of Action:**

  None

Extensions:

None.

Exceptions:

None.

Concurrent Uses:

None

Related Use Cases:

None

Decision Support

Frequency: On average, 5 attempts per day.

Criticality: High. The system should ensure correct access and privileges.

Risk: High. This is a standard security measure that does not require a lot of work to implement, including integration of authenticator applications such as Duo, Google Authenticator or SMS.

Constraints:

- Usability
    a) User must be aware of their privileges and what actions those privileges permit.
- Reliability
    a) Mean Time to Failure – 1% failure yearly is acceptable.
    b) Availability – 30 minutes in a 24-hour period for backup and maintenance.
- Performance
    a) Privilege Checks should be done within 2 seconds.
    b) The system should handle 20 privilege checks in 1 minute.
- Supportability
    a) Should be supported by all browsers.
- Implementation
    a) Using Java-based software for back-end.

Modification History

Owner: Kian Maroofi

Initiation date: 09/10/2019

Date last modified: 09/15/2019

11.2.13Kick Privileges
Use Case ID: SOS13 – Kick Priveleges

Use Case Level: Privileges

Details:

- **Actor:** Organizer.
- **Pre-conditions:**
    1. Organizer has successfully logged onto the system.
    2. The application is open.
    3. There is at least one member part of the organization.
- **Description:**
    1. Use case begins when Organizer clicks on the member management tab.
    2. The system shall provide the Organizer with a list of members that are sorted.
    3. The Organizer will click on the member that they want to kick out.

4. The Organizer will then click on the kick button in the member description.
5. The Organizer will provide a short description to the member why they are being kicked from their organization.
6. The Organizer will send the request by selecting the send button.
7. The system shall notify Organizer if the request was submitted correctly.
8. Use case ends when the system will remove the member from the organization.

- **Relevant requirements:**

  None

- **Post-conditions:**

  1. The request to kick the member is saved by the system.
  2. When the kicked member logs in they will receive a message notifying why they have been kicked from said organization.

Alternative Courses of Action

1. In step D.6 (step 6 of Description section) the user has the option to cancel the kick request.
2. In step D.5 if the description is left blank the system will provide the user with a message to give a short reason why the member is being kicked.
3. In step D.2 the list of users can be sorted alphabetically or by ranking.

Exceptions:

1. There are no members in the organization to kick.

Related Use Cases: None.

Decision Support

Frequency: On average 50 kick requests are made monthly by Organizer.

Criticality: High. Allows the Organizer to kick inactive members to make space for other people that will contribute to their organization.

Risk: Medium. Implementing this use case requires web-based technology.

Constraints:

- Usability:

  a) No previous training required.

b) On average the user should take 2 minutes to complete the kick request to the system.

- Reliability
    a) Mean time to failure – 5% failures for every month of operation is acceptable.
    b) Availability – Down time for Login Back-up 30 minutes in a 24 hour period.
- Performance
    a) Request should be sent and saved within 6 seconds.
    b) System should be able to handle 100 requests in 1 minute.
- Supportability
    a) The Event Creation should be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Anthony Sanchez-Ayra

Initiation date: 09/03/2019

Date last modified: 09/15/2019

11.2.14Create Roles
Use Case ID: SOS14 – Create Roles

Use Case Level: User Privileges

Details:

- **Actor:** Organizer.
- **Pre-conditions:**
    1. Organizer has successfully logged onto the system.
    2. Organizer has Manage Roles Privileges.
    3. The application is open.
- **Description:**
    1. Use case begins when Organizer clicks on the **Organization Roles** tab within the **Organization Management** view.

2. The system shall display a view with a description of the current organization roles along with options to **Edit** the roles and **Create New Role**.
3. The Organizer clicks on the **Create New Role** button.
4. The system shall prompt the Organizer with a Role Creation form, which shall present them with a template for data entry.
5. The Organizer shall enter the following data:
    - **Role Name**
    - **Privileges of the Role**, which come from a set list of privileges including:
        i. Kick
        ii. Invite
        iii. Promote
        iv. Manage Event
        v. Manage Roles
    - **Security Requirement**, which come from a set list including:
        i. 2-Factor Authentication
        ii. Organization-Defined Password
6. The Organizer shall complete the Role Creation by selecting the **Submit** button.
7. The System shall notify the Organizer that the Role was added correctly.
8. Use Case ends when the system adds the new role to the Organization.

- **Relevant requirements:**

  None

- **Post-conditions:**
    1. The request to create a new role is saved by the system
    2. The new role appears as an option when assigning roles to Organizers.

Alternative Courses of Action

1. In step D.2, the list of roles can be sorted alphabetically or by privileges.
2. In step D.5, if any of the fields are left empty the system will require the user to fill in those requirements.
3. In step D.6, the Organizer has the option to **Cancel** the new role creation.

Exceptions:

1. The organization administrator attempts to make a role that already exists.

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average 5 role creation requests are made every 3 months by organizer.

Criticality: High. Allows the Organizer to give different privileges to users to ensure that organization management runs smoothly.

Risk: Medium. Implementing this use case requires web-based technology.

Constraints:

- Usability:
    a) No previous training required.
    b) On average the user should take 2 minutes to complete the promotion request to the system.
- Reliability
    a) Mean time to failure – 5% failures for every month of operation is acceptable.
    b) Availability – Down time for Login Back-up 30 minutes in a 24 hour period.
- Performance
    a) Request should be sent and saved within 6 seconds.
    b) System should be able to handle 100 requests in 1 minute.
- Supportability
    a) Shall be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Anthony Sanchez-Ayra

Initiation date: 09/03/2019

Date last modified: 09/22/2019

11.2.15Notifications

Use Case ID: SOS15 - Notifications

Use Case Level: High-Level

Details:

- **Actor:** Member.
- **Pre-conditions:**
    1. Member has an account in the system.
    2. Member is part of at least one organization and is subscribed to events.
- **Description:**
    1. Use case begins when member clicks on the organizations tab.
    2. The system shall provide the member with a set of cards that represent the organizations that they are a part of.
    3. The member will click on the organization that they want to obtain notifications for.
    4. The member will click on get event news button on the organization description page.
    5. The system shall notify the member that the request was submitted correctly.
    6. Use case ends when the system allows the user to receive notifications for events of the organization.
- **Relevant requirements:**

    None
- **Post-conditions:**
    1. The request to receive notifications from the organization is saved in the system.

Alternative Courses of Action:

None.

Exceptions:

None.

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average 30 notification requests are made daily by the member.

Criticality: High. Allows the member to know when the organization that they are a part of is conducting events.

Risk: Medium. Implementing this use case requires web-based technology.

Constraints:

- Usability:
    a) No previous training required.
    b) On average the user should take 2 minutes to complete the notification request to the system.
- Reliability
    a) Mean time to failure – 5% failures for every month of operation is acceptable.
    b) Availability – Down time for Login Back-up 30 minutes in a 24 hour period.
- Performance
    a) Request should be sent and saved within 6 seconds.
    b) System should be able to handle 100 requests in 1 minute.
- Supportability
    a) The Event Creation should be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Anthony Sanchez-Ayra

Initiation date: 09/03/2019

Date last modified: 09/15/2019

11.2.16Create Organization
Use Case ID: SOS16 – Create Organization

Use Case Level: High-Level

Details:

- **Actor:** User
- **Pre-conditions:**
    1. User has an account in our application.
    2. User is successfully logged into the application.
- **Description:**
    1. Use case begins when User clicks on the Organization tab in their current page (home page for example) and the homepage refreshes and provides the Organizer with the Organization page.
    2. The organization page shall provide the User with a set of cards that represent the organizations that they are a part of and a Create Organization option.
    3. The User will click on the Create Organization option.
    4. The organization page shall provide the User with a form to fill out, asking for the following details:
        - **Organization Name**
        - **Organization Description**
        - **Requirements for Joining**
        - **Privacy of the Organization** (whether it's open to others or not).
    5. The system shall notify the User that the request was submitted correctly by showing a notification in the Organization page.
    6. Use case ends when the organization page the displays the new organization that the User has created a new organization.
- **Relevant requirements:**

    None
- **Post-conditions:**
    1. The request to create an organization is stored in the system.
    2. The organization is shown to members depending on its privacy settings.
    3. The User has gained owner status with respect to the created organization.

Alternative Courses of Action:

1. In step D.4 the user has the option to cancel the creation of their organization.
2. In step D.5 if any of the fields are left blank the system will provide the user with a message to fill in all the fields.

3. In step D.5 the system shall ask the user to confirm if they would like to create an organization.

Exceptions:

1. If the User tries to make an organization that already exists, then they will get an error message.

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average 20 organization creation requests are made monthly by the User.

Criticality: High. Allows the User to create an organization which allows new communities to grow around campus.

Risk: Medium. Implementing this use case requires web-based technology.

Constraints:

- Usability:
    a) No previous training required.
    b) On average the user should take 2 minutes to complete the notification request to the system.
- Reliability
    a) Mean time to failure – 5% failures for every month of operation is acceptable.
    b) Availability – Down time for Login Back-up 30 minutes in a 24 hour period.
- Performance
    a) Request should be sent and saved within 6 seconds.
    b) System should be able to handle 200 requests in 1 minute.
- Supportability
    a) The Event Creation should be supported by Chrome, Mozilla, and IE.
- Implementation

a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Anthony Sanchez-Ayra

Initiation date: 09/04/2019

Date last modified: 09/15/2019

11.2.17Cancel an Event
Use Case ID: SOS17 - Cancel an Event

Use Case Level: User Goal

Details:

- **Actor:** Organizer
- **Pre-conditions:**
    1. Organizer has an account in our application.
    2. Organizer is successfully logged into the application.
    3. Organizer is part of a organization.
- **Description:**
    1. Use case begins when organizer clicks on the event that they want to cancel.
    2. The system shall redirect the organizer to the Event Description view, which shall present them with a button labeled cancel event.
    3. The organizer will click on the cancel event button.
    4. The organizer will click yes on the validation message displayed by the system.
    5. The system shall notify the organizer that the event was cancelled.
    6. End case ends when the system removes the event from being viewed.
- **Relevant requirements:**

    None
- **Post-conditions:**
    1. The system notifies all users that subscribed to the event that it has been cancelled.

Alternative Courses of Action:

1. In step D.3 the system will prompt the organizer with a validation message to confirm that they actually want to cancel the event.

Exceptions:

1. The database is not active.
2. The Event Description view is not active.
3. The validation message is not active.

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average 5 cancellation requests are made weekly by the organizer.

Criticality: High. Allows the organizer to cancel an event whenever necessary.

Risk: High. Implementing this use case requires web-based technology.

Constraints:

- Usability:
    a) No previous training required.
    b) On average the user should take 2 minutes to complete the notification request to the system.
- Reliability
    a) Mean time to failure – 5% failures for every month of operation is acceptable.
    b) Availability – Down time for Login Back-up 30 minutes in a 24 hour period.
- Performance
    a) Request should be sent and saved within 6 seconds.
    b) System should be able to handle 100 requests in 1 minute.
- Supportability
    a) Shall should be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Anthony Sanchez-Ayra

Initiation date: 09/04/2019

Date last modified: 09/15/2019

11.2.18Create Task
Use Case ID: SOS18 – Create Task

Use Case Level: User Goal

Details:

- **Actor:** Organizer
- **Pre-conditions:**
    1. An Event has been created by an Organization
    2. Organizer has privileges on the given Organization, and is logged in.
- **Description:**
    1. Use case begins when the Organizer clicks on the Add **Task** in the edit view of an Event page.
    2. The system shall prompt the Organizer with an **Add Task** form, which shall present them with a template for data entry.
    3. The Organizer shall input the following data in the template:
        - **Task Name**
        - **Task Description**
        - **Expected Number of Participants**
    4. The Organizer shall finish adding the task by selecting the **Complete** button.
    5. The page shall notify the Organizer that the task was added to the Event.
    7. Use case ends when the system updates the Event with the task according to the specification.
- **Relevant requirements:**

    None
- **Post-conditions:**
    1. The Event has been updated so that it shows the details pertaining to the task in the backing database, and this change is reflected in the Event's page.

Alternative Courses of Action:

1. In step D.4, the Organizer has the option to **Cancel** the task creation.

Exceptions:

None

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average, 20 tasks are added to events a week.

Criticality: Medium. Not all events require tasks to be complete, so not all users will use this functionality.

Risk: Medium. Implementation does not require any complex specialized knowledge besides a database system.

Constraints:

- Usability:
    a) Requires minimal training.
    b) One or two help frames on the Help page shall be provided explaining how to add tasks.
    c) On average the user should less than 5 minutes to complete the notification request to the system.
- Reliability
    a) Mean time to failure – 5% failures for every 24 hours of operation is acceptable.
    b) Availability
        - Downtime for Login Back-up – 30 minutes in a 24-hour period.
        - Downtime for Maintenance – 1 hour in a 2 weeks period.
- Performance
    a) Request should be sent and saved within 6 seconds.
    b) System should be able to handle 100 requests in 1 minute.
- Supportability
    a) Shall be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Yovanni Jones

Initiation date: 09/02/2019

Date last modified: 09/22/2019

11.2.19Request Organization Information
Use Case ID: SOS19 – Request Organization Information

Use Case Level: Access Organization Page

Details:

- **Actor:** Organizer
- **Pre-conditions:**
    1. User is logged into the system.
- **Description:**
    1. Use case begins when the User opens the Sidebar and clicks on the **Organization** tab.
    2. The system shall change the view to the Organizations view, listing all the available organizations.
    3. The User selects an Organization by clicking on it.
    4. Use Case ends when the system changes the view to the Organization's page, which shall contain a description of the Organization and Event information.
- **Relevant requirements:**

  None
- **Post-conditions:**
    1. The view of the User has changed to the Organization's page.

Alternative Courses of Action:

1. In step D.4, if the User has privileges over the chosen Organization, a privileged view providing access to the Event Creation, Task Creation, and other Organization management tabs will be displayed instead.

Exceptions:

1. The page for the Organization cannot be found or has been deleted.

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average, asking for a description of what the organization is could happen 1000 times a day.

Criticality: High. This a core functionality of the system.

Risk: Low. Requires no specialized knowledge.

Constraints:

- Usability:
    a) No previous training required.
    b) Should take under 5 minutes to acquire info on organization
- Reliability
    a) Mean time to failure – 5% failures for every 24 hours of operation is acceptable.
    b) Availability
        - Downtime for Login Back-up – 30 minutes in a 24-hour period.
        - Downtime for Maintenance – 1 hour in a 2 weeks period.
- Performance
    a) System should be able to handle 100 requests in 1 minute.
- Supportability
    a) Shall be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Yovanni Jones

Initiation date: 09/02/2019

Date last modified: 09/22/2019

11.2.20Remove Organization
Use Case ID: SOS20 – Remove Organization

Use Case Level: User Goal

Details:

- **Actor:** Organizer
- **Pre-conditions:**
    1. Organizer is the owner of the target Organization.
- **Description:**
    1. Use case begins when the Organizer clicks on the **Remove Organization** button on the Organization's Settings page.
    2. The system shall prompt the Organizer with a form, requesting for the Organization's unique ID number.
    3. The Organizer shall enter the unique ID number.
    4. The Organizer shall complete the deletion by selecting the **Confirm** button.
    5. The system shall remove all the future Events by the Organization from the Event views and delete their records. Past Events shall be kept and displayed on the User's page.
    6. The system shall revoke the Member status from Users who were members of the Organization. Same thing for Organizers.
    7. Use case ends when the system has notified the relevant users and saved a record of the deletion.
- **Relevant requirements:**

    None
- **Post-conditions:**
    1. The Organization has been deleted from the system and it will no longer appear on the Organization tab.
    2. The future Events by the Organization have been deleted.
    3. Users with Member or Organizer status on the Organization have been stripped of these status.
    4. A record has been saved of the deletion request.

Alternative Courses of Action:

1. In step D.2, the Organizer has the option to **Cancel**.

Exceptions:

The Organizer is missing the required permissions for deletion (is not the owner).

The Organization has special privileges preventing it from being deleted.

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average, 3 organizations removed per week.

Criticality: High. Deletions and disbandment should be handled correctly and the information on the website should be kept up-to-date.

Risk: Medium. Implementation does not require any complex specialized knowledge, but a secure implementation is required to make sure no unauthorized person is able to delete an Organization.

Constraints:

- Usability:
    a) Requires minimal training.
    b) One or two help frames on the Help page shall be provided.
    c) On average the user should less than 5 minutes to complete the notification request to the system.
- Reliability
    a) Mean time to failure – 5% failures for every 24 hours of operation is acceptable.
    b) Availability
        - Downtime for Login Back-up – 30 minutes in a 24-hour period.
        - Downtime for Maintenance – 1 hour in a 2 weeks period.
- Performance
    a) Request should be sent and saved within 6 seconds.
    b) System should be able to handle 100 requests in 1 minute.

- Supportability
    a) Shall be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Yovanni Jones

Initiation date: 09/02/2019

Date last modified: 09/22/2019

11.2.21Avoid Time Conflicting Events
Use Case ID: SOS21 – Avoid Time Conflicting Events

Use Case Level: User Goal

Details:

- **Actor:** Organizer
- **Pre-conditions:**
    1. Organizer has privileges on the given Organization, and is logged in.
- **Description:**
    1. Use case begins when the Organizer clicks on the **Create Event** on the administration page of their Organization.
    2. The system shall prompt the Organizer with an Event Creation form, which shall present them with a template for data entry.
    3. The Organizer shall enter the required data (see SOS1 – Create Event).
    4. The system shall check the **Event Date and Time** against the other Events of the Organization.
    5. If a conflicting Event is found, the system shall notify the Organizer of this conflict and present the Organizer with a new form.
    6. The Organizer shall enter the following data:
        - **New Event Date and Time** which will be preset with the conflicting date.
    7. The Organizer complete the Event Creation by selecting the **publish** button.
    8. The system shall notify the Organizer that the event was published correctly.

9. Use case ends when the system receives the Event specification, generates a unique event id and publishes the Event according to the specifications.

- **Relevant requirements:**

  None

- **Post-conditions:**

  1. An event has been published by the Organizer representing the Organization according to the specifications given.

Alternative Courses of Action:

1. In step D.6., the Organizer has the option of publishing the event at the original conflicting date by clicking **publish** without changing the default conflicting date.

Exceptions:

1. The event database is not active.
2. The event creation view is not active.

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average 3 Events are created per Organization weekly.

Criticality: High. The most basic and central activity of the whole system is Event Creation.

Risk: Medium. Implementation does not require any complex specialized knowledge.

Constraints:

- Usability
  a) No previous training or knowledge.
  b) Tutorial or Help frame should be provided.
  c) Organizer should take less than 10 minutes to create an event.
- Reliability
  a) Mean Time to Failure – 5% failure monthly is acceptable.

b) Availability

- Downtime for Login Back-up – 30 minutes in a 24-hour period.
- Downtime for Maintenance – 1 hour in a 2 weeks period.

- Performance
  a) The form should be sent and saved within 10 seconds.
  b) The system should be able to handle 50 requests in 1 minute.
- Supportability
  a) The Event Creation should be supported by Chrome, Mozilla, and IE.
- Implementation
  a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Yovanni Jones

Initiation date: 09/02/2019

Date last modified: 09/22/2019

11.2.22Registration
Use Case ID: SOS22 – Registration

Use Case Level: User Goal

Details:

- **Actor:** User
- **Pre-conditions:**
  1. The User does not have an account on the site.
- **Description:**
  1. Use case begins when the User presses the **Register** button on the log-in/register page.
  2. The system shall prompt the User with a **Registration** form, which shall present them with a template for data entry.
  3. The Organizer shall input the following data in the template:
     - **User Name**
     - **Email**

- **Password**
- **Confirm Password**

4. The User shall complete the registration by selecting the **Ok** button.
5. The system shall confirm that the registration was successful.
6. Use case ends when the User is automatically logged into the system and the view is moved to home.

- **Relevant requirements:**

  None

- **Post-conditions:**

  None

Alternative Courses of Action:

1. In step D.3, If any of the fields have incorrect information or are left blank system will respond with a message saying that proper credentials should be entered.

Exceptions:

None

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average, 20 tasks are added to events a week.

Criticality: Medium. Not all events require tasks to be complete, so not all users will use this functionality.

Risk: Medium. Implementation does not require any complex specialized knowledge besides a database system.

Constraints:

- Usability:
  a) Requires minimal training.

b) One or two help frames on the Help page shall be provided explaining how to add tasks.

c) On average the user should less than 5 minutes to complete the notification request to the system.

- Reliability
    a) Mean time to failure – 5% failures for every 24 hours of operation is acceptable.
    b) Availability
        - Downtime for Login Back-up – 30 minutes in a 24-hour period.
        - Downtime for Maintenance – 1 hour in a 2 weeks period.
- Performance
    a) Request should be sent and saved within 6 seconds.
    b) System should be able to handle 100 requests in 1 minute.
- Supportability
    a) Shall be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Yovanni Jones

Initiation date: 09/02/2019

Date last modified: 09/22/2019

11.2.23Admin: Manual Deletion of Events
Use Case ID: SOS23 – Admin: Manual Deletions of Events

Use Case Level: Administrator Role

Details:

- **Actor:** Administrator
- **Pre-conditions:**
    1. Administrator is logged into the system.
    2. A User has created an Event which violates privacy agreements, terms of use, promotes violence, or is otherwise deemed inadmissible.

- **Description:**

    1. Use case begins when the Event is presented to the Administrator to be reviewed, either because it has been reported by Users, or because it has been found inadmissible by the Administrator.
    2. The Administrator reviews the event with a system that checks the spelling for any misconduct. Nouns, Verbs, Adjectives, etc. that may imply some sort of malicious intention.
    3. The Administrator clicks on **Quarantine Event** to initiate a removal process, giving a reason as to why this measure was taken.
    4. The system shall delete the Event from the Events and Organization page.
    5. The system shall notify that the Event will be deleted, citing the reason given by the Administrator. A standard warning about misconduct shall be issued to the User.
    6. The Use Case ends when the system records the request for deletion, as well as record the infringement under the User's information for the Administrator to see in the future.

- **Relevant requirements:**

    None

- **Post-conditions:**

    1. The User who created the account will had been warned about the action. If continued infringements occur, he or she will be barred from creating more events or event banned from the system.
    2. The Event in question will had been deleted from public view.

Alternative Courses of Action:

1. In step D.3, the Administrator has an option to request more information by clicking **Inquire**, which will open an investigation to the Event and contact the Organization and the User

Exceptions:

None

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average, the system will have to do 5-10 checks daily.

Criticality: High. Users will be making a lot of posts so making sure they are not dangerous is crucial.

Risk: Medium. Implementation does not require any complex specialized knowledge besides a database system.

Constraints:

- Usability:
    a) Will require training for Administrator to deal with and recognize threats, but the system itself should be easy to use.
    b) One or two help frames explaining the Quarantine and Inquire process should be provided.
- Reliability
    a) Mean time to failure – 5% failures for every 24 hours of operation is acceptable.
    b) Availability
        - Downtime for Login Back-up – 30 minutes in a 24-hour period.
        - Downtime for Maintenance – 1 hour in a 2 weeks period.
- Performance
    a) Request should be sent and saved within 6 seconds.
    b) System should be able to handle 100 requests in 1 minute.
- Supportability
    a) Shall be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Yovanni Jones

Initiation date: 09/02/2019

Date last modified: 09/22/20

11.2.24Admin: Extended Privileges
Use Case ID: SOS24 – Admin: Extend Priveleges

Use Case Level: Administrator Role

Details:

- **Actor:** Administrator
- **Pre-conditions:**
    1. Administrator is logged into the system.
- **Description:**
    1. Use case begins when an Administrator accesses a User Profile, Organization Page, or Event Page.
    2. The system shall present the Administrator with privilege views over those pages, giving a more flexible control on each Event, Organization, and enabling monitoring and observing normal Users (Members, Organizers) for them.
    3. The Use Case ends when these pages are presented to the Administrator.
- **Relevant requirements:**

  None
- **Post-conditions:**

  None

Alternative Courses of Action:

None

Exceptions:

None

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average, 25 attempts per day.

Criticality: High. The system should ensure correct access and privileges.

Risk: High. This is a standard security measure that does not require a lot of work to implement.

Constraints:

- Usability:
    a) User must be aware of their privileges and what actions those privileges permit.
    b) Some training about privileges is required.
    c) One or two help frames explaining the extent of Administrator Privileges, Roles, and Expectations shall be provided.
- Reliability
    a) Mean time to failure – 5% failures for every 24 hours of operation is acceptable.
    b) Availability
        - Downtime for Login Back-up – 30 minutes in a 24-hour period.
        - Downtime for Maintenance – 1 hour in a 2 weeks period.
- Performance
    a) Privilege Checks should be done within 2 seconds.
    b) The system should handle 20 privilege checks in 1 minute.
- Supportability
    a) Shall be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Kian Maroofi

Initiation date: 09/02/2019

Date last modified: 09/22/20

11.2.25Filter Events
Use Case ID: SOS25 – Filter Events

Use Case Level: User Goal

Details:

- **Actor:** User
- **Pre-conditions:**
    1. User is logged into the site.
- **Description:**
    1. Use case begins when the user clicks on the “events” tab.
    2. The user clicks on “find events”.
    3. The system displays a list of tags (potlucks, volunteering, social events, etc.).
    4. The user selects one or more of their desired tags.
    5. Use case ends when the system automatically updates the page with a list of events relevant to the selected tags.
- **Relevant requirements:**

  None
- **Post-conditions:**
    1. The relevant events are made viewable.

Alternative Courses of Action:

1. In step D.5, the user has the option to unselect and reselect tags.

Exceptions:

1. The find event button is not active.
2. The user does not select any tags.

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average 100 requests are made daily.

Criticality: High. Allows the user to find events they may be interested in.

Risk: Low. Implementing this use case doesn’t require specialized knowledge nor using it requires any sensitive information from the user.

Constraints:

- Usability:
    a) No previous training time, no explicit instructions required.
    b) Should take about 30 seconds for the average user to complete the use case.
- Reliability
    a) Mean time to failure – 5% failures for every month of operation is acceptable.
    b) Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
- Performance
    a) The page should be updated in real time as the user clicks on each tag.
- Supportability
    a) The Event Creation should be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Teriq Douglas

Initiation date: 09/06/2019

Date last modified: 09/16/2019

11.2.26Invite User from Roster
Use Case ID: SOS26 – Invite User from Roster

Use Case Level: User Goal

Details:

- **Actor:** Organizer
- **Pre-conditions:**
    1. The Organizer is logged into site.
    2. The Organizer has the adequate privileges within the Organization.
- **Description:**
    1. Use case begins when the organizer clicks "*My Organization*".
    2. Then organizer clicks "*View Roster*".
    3. The system shall show a list of current members registered on the site.

4. The organizer clicks "*Invite Member*".
5. The system shall ask for the organizer to input the member's email.
6. The organizer clicks "*Submit*".
7. The system shall ask the Organizer for confirmation.
8. The organizer clicks "*Confirm*".
9. The system shall send an invitation email to the member.
10. Use case ends when the system displays the message "*Invitation Sent*".

- **Relevant requirements:**

  None

- **Post-conditions:**

  1. The relevant events are made viewable.

Alternative Courses of Action:

1. In step D.8, the organizer clicks "*Cancel*", cancelling the request.

Exceptions:

1. Incorrect email.
2. The submit and/or remove button is not active.

Concurrent Use Cases:

None.

Related Use Cases:

SOS27 – Removing User from Roster

Decision Support

Frequency: About 5000 roster changes are made daily.

Criticality: High. Allows the organizer to have a stable view of their roster.

Risk: Low. Implementing this use case doesn't require any complex knowledge.

Constraints:

- Usability:
  a) Might require light training.
  b) One help frame on the Help page provided.

c) On average the user should take 1 minute to update their roster.

- Reliability
    a) Mean time to failure – 1% failures for every month of operation is acceptable.
    b) Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
- Performance
    a) System should be able to handle 100 requests in 1 minute.
- Supportability
    a) The Event Creation should be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Teriq Douglas

Initiation date: 09/06/2019

Date last modified: 09/16/2019

11.2.27Remove User from Roster
Use Case ID: SOS27 – Remove User From Roster

Use Case Level: User Goal

Details:

- **Actor:** Organizer
- **Pre-conditions:**
    1. The Organizer is logged into site.
    2. The Organizer has the adequate privileges within the Organization.
- **Description:**
    1. Use case begins when the organizer clicks "*My Organization*".
    2. Then Organizer clicks "*View Roster*".
    3. The system shall show a list of current members registered on the site.
    4. The Organizer clicks "*Remove Member*".
    5. The system shall ask the Organizer for a member's name or email.

6. The Organizer clicks "*Submit*".
7. The system shall ask the Organizer for confirmation.
8. The Organizer clicks "*Confirm*".
9. The system shall remove the member from the organization.
10. Use case ends when the system displays the message "*Invitation Sent*".

- **Relevant requirements:**

  None

- **Post-conditions:**

  1. The relevant events are made viewable.

Alternative Courses of Action:

1. In step D.8, the organizer clicks "*Cancel*", cancelling the request.

Exceptions:

1. Incorrect email.
2. The submit and/or remove button is not active.

Concurrent Use Cases:

None.

Related Use Cases:

SOS26 – Invite User from Roster

SOS13 – Kick Privileges

Decision Support

Frequency: About 5000 roster changes are made daily.

Criticality: High. Allows the organizer to have a stable view of their roster.

Risk: Low. Implementing this use case doesn't require any complex knowledge.

Constraints:

- Usability:
  a) Might require light training.
  b) One help frame on the Help page provided.
  c) On average the user should take 1 minute to update their roster.

- Reliability
    a) Mean time to failure – 1% failures for every month of operation is acceptable.
    b) Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
- Performance
    a) System should be able to handle 100 requests in 1 minute.
- Supportability
    a) The Event Creation should be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Teriq Douglas

Initiation date: 09/06/2019

Date last modified: 09/16/2019

11.2.28User RSVP
Use Case ID: SOS28 - RSVP

Use Case Level: User Goal

Details:

- **Actor:** User
- **Pre-conditions:**
    1. The Organizer is logged into site.
- **Description:**
    1. Use case begins when the User finds clicks on "*RSVP*" on an Event.
    2. The system shall display a description of the Event which includes the date, time, location, and a list of rules.
    3. The User must click on "*Confirm*" to confirm the RSVP.
    4. Use case ends when the system shows a success message.
- **Relevant requirements:**

    None

- **Post-conditions:**
    1. The system adds the user to the guest list.
    2. The system adds the event to the user's list of attending events.

Alternative Courses of Action:

1. In step D.3, the User can cancel the RSVP by clicking on "*Cancel*".

Exceptions:

1. Max number of guests reached.

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average 500 RSVPs are made daily.

Criticality: Medium. Allows the user to formally attend events created on campus if they agree with the terms set by the hosts.

Risk: Low. Implementing this use case doesn't require any complex knowledge.

Constraints:

- Usability:
    a) Requires no training.
    b) On average the user should take 20 seconds to perform an RSVP.
- Reliability
    a) Mean time to failure – 1% failures for every month of operation is acceptable.
    b) Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
- Performance
    a) RSVP requests should be processed within 5 seconds.
    b) The system shall be consistent when handling RSVP requests.
- Supportability
    a) The Event Creation should be supported by Chrome, Mozilla, and IE.

- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Teriq Douglas

Initiation date: 09/06/2019

Date last modified: 09/16/2019

11.2.29Unauthorized Organization Management
Use Case ID: SOS29 – Unauthorized Organization Management

Use Case Level: User Goal

Details:

- **Actor:** User
- **Pre-conditions:**
    1. The User is logged into site.
    2. The User does not have privileges (or Organizer status) on the target Organization.
- **Description:**
    1. Use case begins when the User access target Organization's page.
    2. The system shall check for the User's privileges on that Organization.
    3. Use case ends when the system displays the Organization profile, which includes a description and contact information and excludes "*View Roster*" as well as other privileged views of the Organization.
- **Relevant requirements:**

    None
- **Post-conditions:**
    1. The view of the website has changed to the target Organization's page.

Alternative Courses of Action:

None

Exceptions:

None

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: The act of viewing an organization's profile will occur on average 1000 times daily.

Criticality: High. Prevents unauthorized changes in an organization's roster.

Risk: Medium. Implementing this use case doesn't requires some specialized knowledge about privilege control.

Constraints:

- Usability:
    a) Requires no training.
    b) On average the user should take less than 5 seconds to locate and click on the Organization page. It should also not take longer than 1 minutes to realize that the view is different when not logged as an Organizer.
- Reliability
    a) Mean time to failure – 1% failures for every month of operation is acceptable.
    b) Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
- Performance
    a) Should be able to produce results within 3 seconds.
- Supportability
    a) The Event Creation should be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Teriq Douglas

Initiation date: 09/06/2019

Date last modified: 09/16/2019

11.2.30Unauthorized Event Creation

Use Case ID: SOS30 – Unauthorized Event Creation

Use Case Level: User Goal

Details:

- **Actor:** User
- **Pre-conditions:**
    1. The User is logged into the site.
    2. The User does not have privileges (or Organizer status) on the target organization.
- **Description:**
    1. Use case begins when the user clicks "*My Organizations*" assuming the user belongs to an organization.
    2. The system shall display a list of organizations the user belongs to.
    3. The user selects their desired organization.
    4. The system shall check the privileges of the User relating to the chosen administration.
    5. Use case ends when system displays the profile page omitting the "*Schedule*" button and other managerial views.
- **Relevant requirements:**

    None
- **Post-conditions:**
    1. The view of the website has changed to the target organization's page.

Alternative Courses of Action:

1. In step D.2, if they user does not belong to any organization, when they click on "my organization" the system will display a message saying that they do not belong to one.

Exceptions:

None

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average, up to 3000 requests daily.

Criticality: High. Prevents unauthorized event creation.

Risk: Medium. Implementing this use case doesn't requires some specialized knowledge about privilege control.

Constraints:

- Usability:
    a) Requires no training.
    b) On average the user should take less than 5 seconds to locate and click on the Organization. It should also not take longer than 1 minutes to realize that the view is different when not logged as an Organizer.
- Reliability
    a) Mean time to failure – 1% failures for every month of operation is acceptable.
    b) Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
- Performance
    a) Should be able to produce results within 3 seconds.
- Supportability
    a) The Event Creation should be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Teriq Douglas

Initiation date: 09/06/2019

Date last modified: 09/16/2019

11.2.31Log in
Use Case ID: SOS31 – Log in

Use Case Level: User Goal

Details:

- **Actor:** User

- **Pre-conditions:**
    1. The User has an account on the SOS site.
- **Description:**
    1. Use case begins when the user is in the **Log-In** page of the site.
    2. The login page shall provide an input form with to following parameters:
        - **Email address**
        - **Password**
    3. The user inputs their email and password and then clicks on login.
    4. The system shall verify if the email and password match.
    5. Use case ends when system allows the user to login.
- **Relevant requirements:**

  None
- **Post-conditions:**
    1. the user is redirected to the **Home** page.

Alternative Courses of Action:

1. In step D.4, if the user types an invalid password or email then the system will notify them that their "email and password do not match."

Exceptions:

None

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average, up to 10000 requests daily.

Criticality: High. Allows the user to log-in to view their organizations and nearby events.

Risk: Low. Implementing this use case doesn't requires specified knowledge.

Constraints:

- Usability:
    a) Requires no training.
    b) On average the user should take less than 10 seconds to type their information and attempt to log in.
- Reliability
    a) Mean time to failure – 1% failures for every month of operation is acceptable.
    b) Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
- Performance
    a) Should be able to produce results within 3 seconds.
- Supportability
    a) The Event Creation should be supported by Chrome, Mozilla, and IE.
- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Anthony Sanchez-Ayra

Initiation date: 09/06/2019

Date last modified: 09/16/2019

11.2.32Log Out
Use Case ID: SOS32 – Log out

Use Case Level: User Goal

Details:

- **Actor:** User
- **Pre-conditions:**
    1. The User is currently logged into the SOS page.
- **Description:**
    1. Use case begins when the user clicks on the **Sign Out** button.
    2. The current page the user is in will call a system call to log the user out.
    3. The system will then attempt to log the user out of the webpage.
    4. Use case ends when website redirects the user to the **Login** page.

- **Relevant requirements:**

  None

- **Post-conditions:**

  None.

Alternative Courses of Action:

None.

Exceptions:

None.

Concurrent Use Cases:

None.

Related Use Cases:

None.

Decision Support

Frequency: On average, up to 10000 requests daily.

Criticality: High. Allows the user to log-out to make sure that no other user can tamper with their account if they were to access the site from the same computer.

Risk: Low. Implementing this use case doesn't requires specialized knowledge.

Constraints:

- Usability:
  a) Requires no training.
  b) On average the user should take less than 5 seconds to find the sign out button and click on it.
- Reliability
  a) Mean time to failure – 1% failures for every month of operation is acceptable.
  b) Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
- Performance
  a) Should be able to produce results within 3 seconds.
- Supportability
  a) The Event Creation should be supported by Chrome, Mozilla, and IE.

- Implementation
    a) The implementation shall use JS React for front-end, and Java-based software for back-end.

Modification History

Owner: Anthony Sanchez-Ayra

Initiation date: 09/06/2019

Date last modified: 09/16/2019

11.3 Appendix C – User Interface Designs

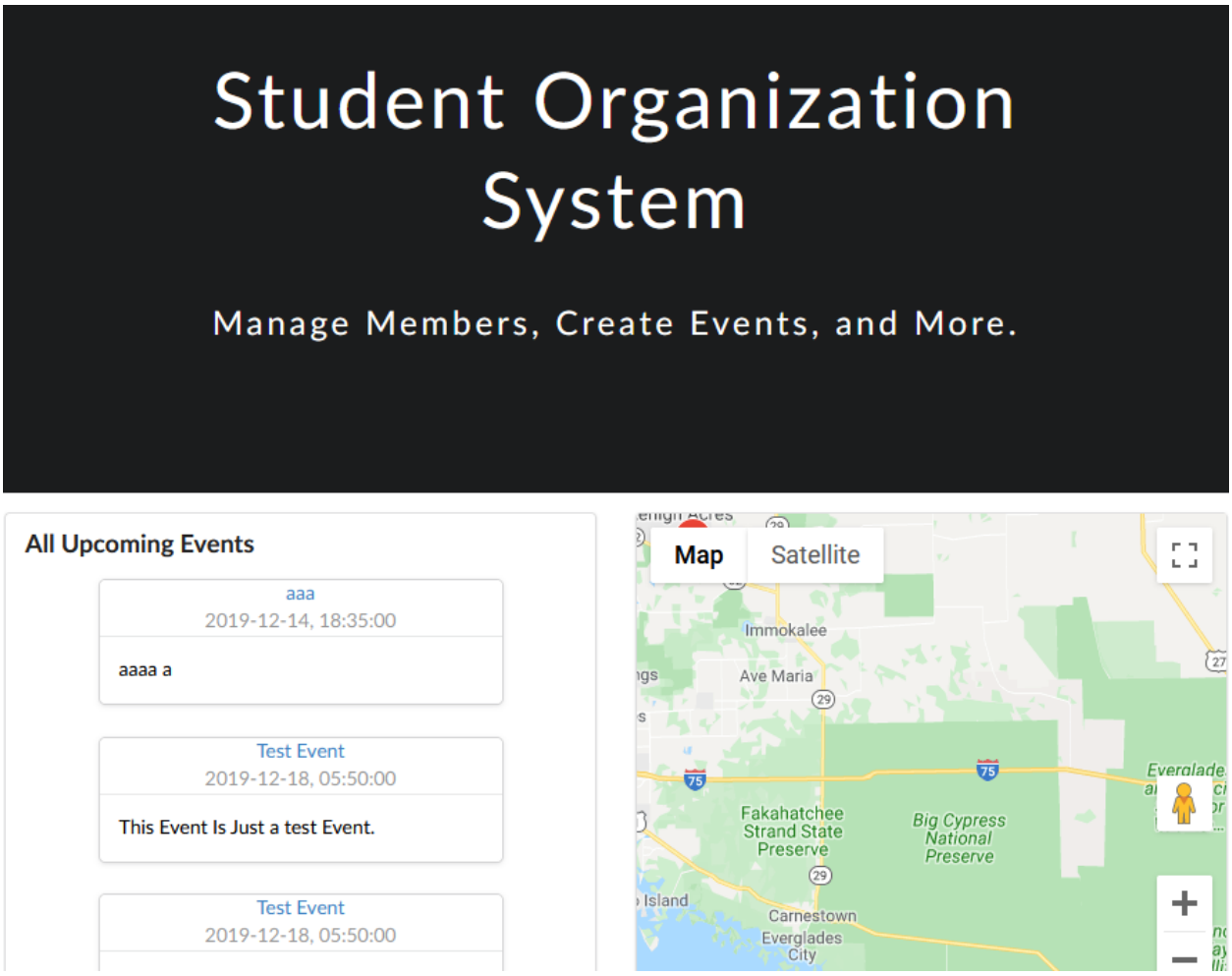

Figure 39: UI Layout for the Homepage. It includes the top-bar, which is a static element of the while system which contains a button to open the navigation menu, and another one to open a log-in pop-up.

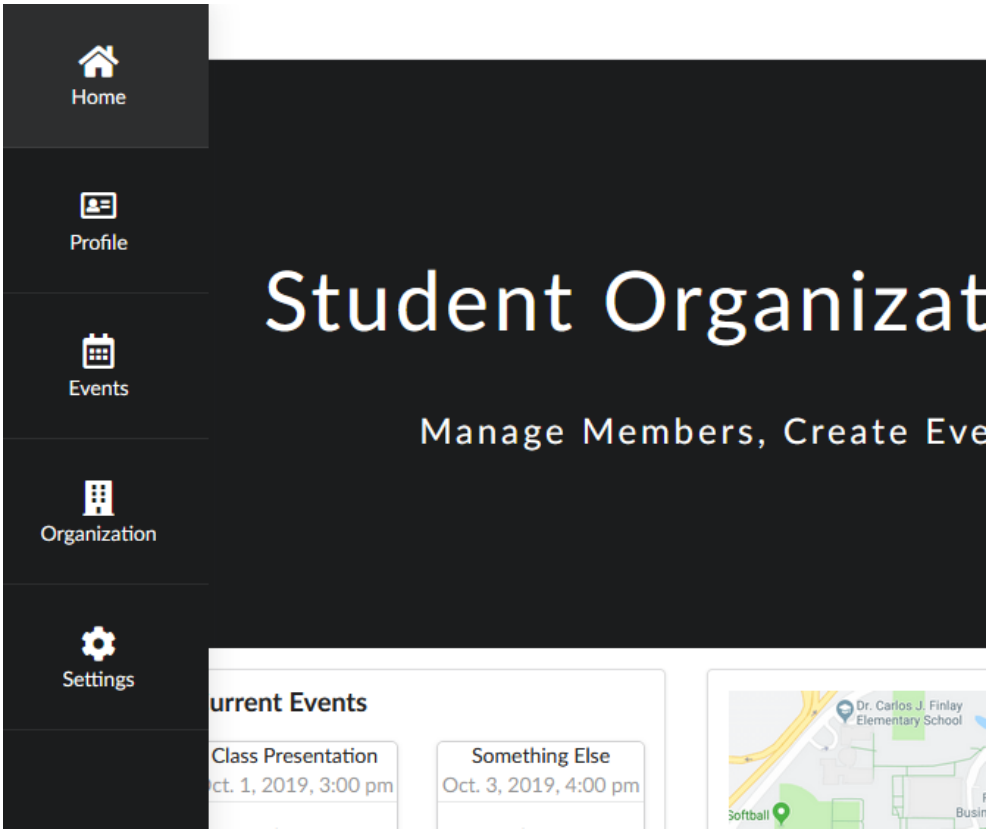

Figure 40: UI Layout for the menu sidebar (with the Homepage on the background). Each button redirects to a specific section on the website that the user can access.

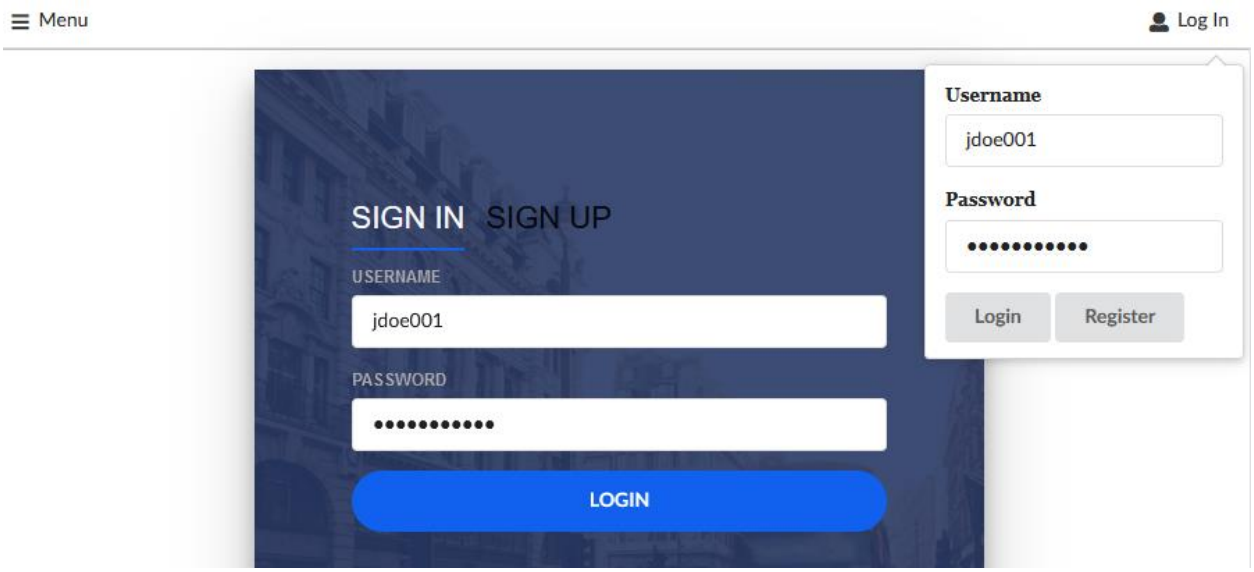

Figure 41: UI Layout for the Log In pop-up (with the Login/Resgiteration page on the background).

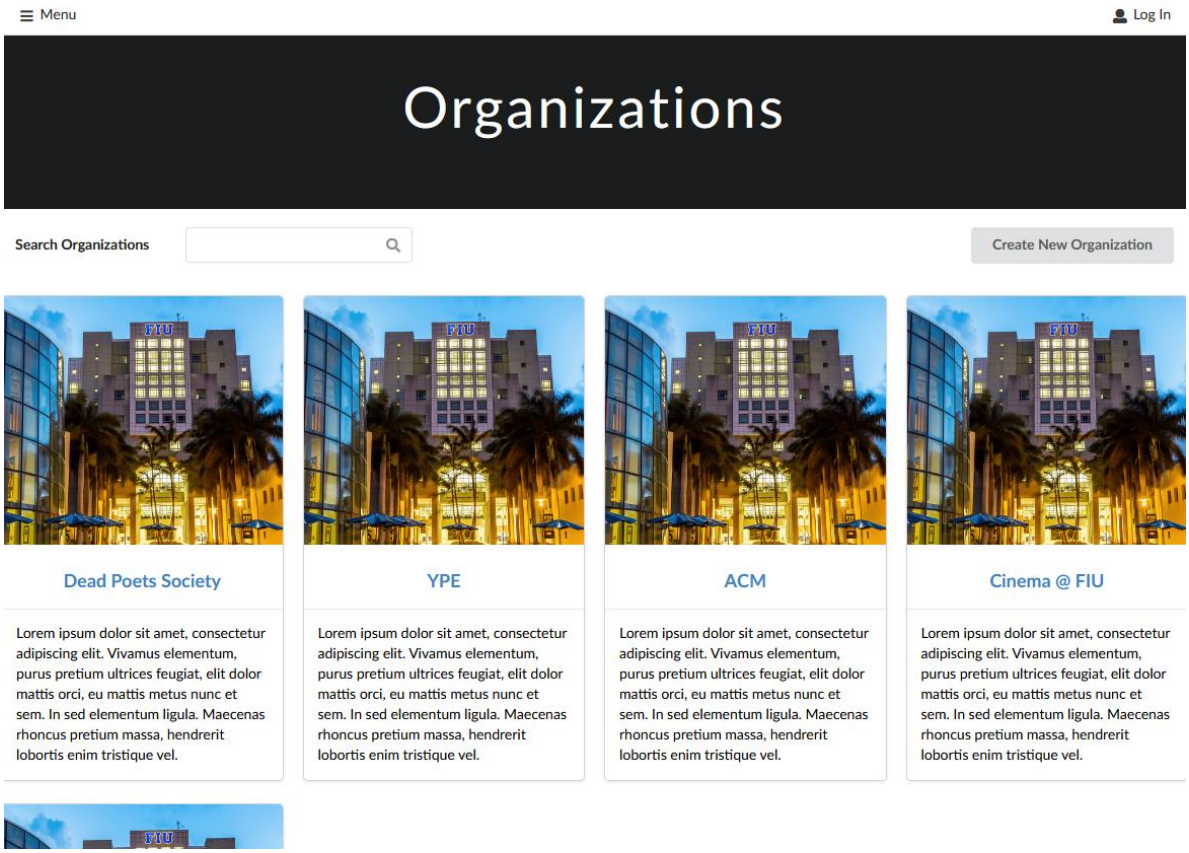

Figure 42: UI layout for the Organizations page. Each organization in the system that is displayed to the user is displayed as at-a-glance card. The create organization button triggers a modal/form (see Figure 36) to define a new Organization.

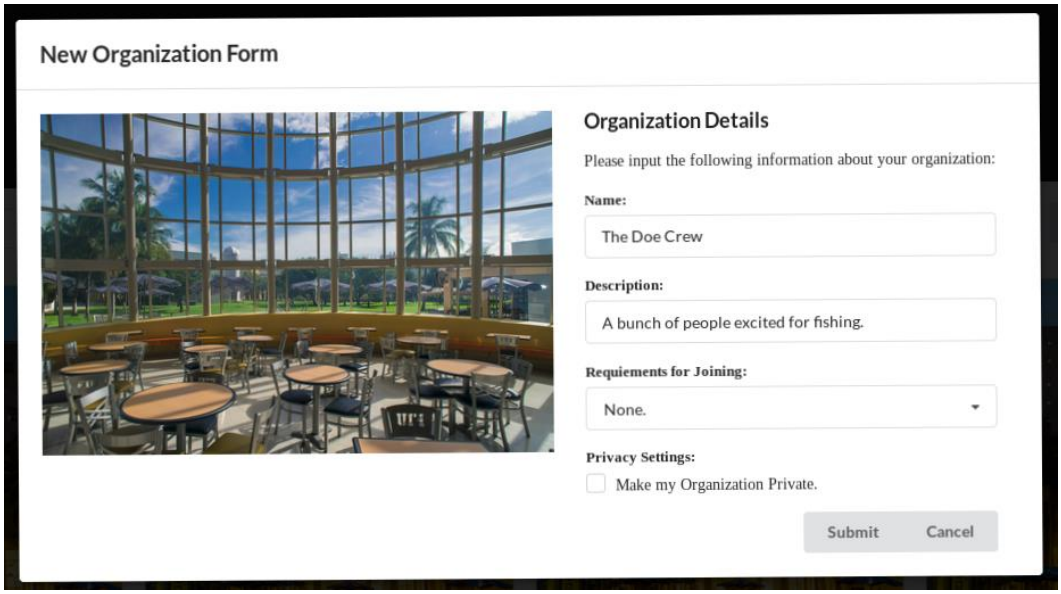

Figure 43: UI Layout for the New Organization Form modal.

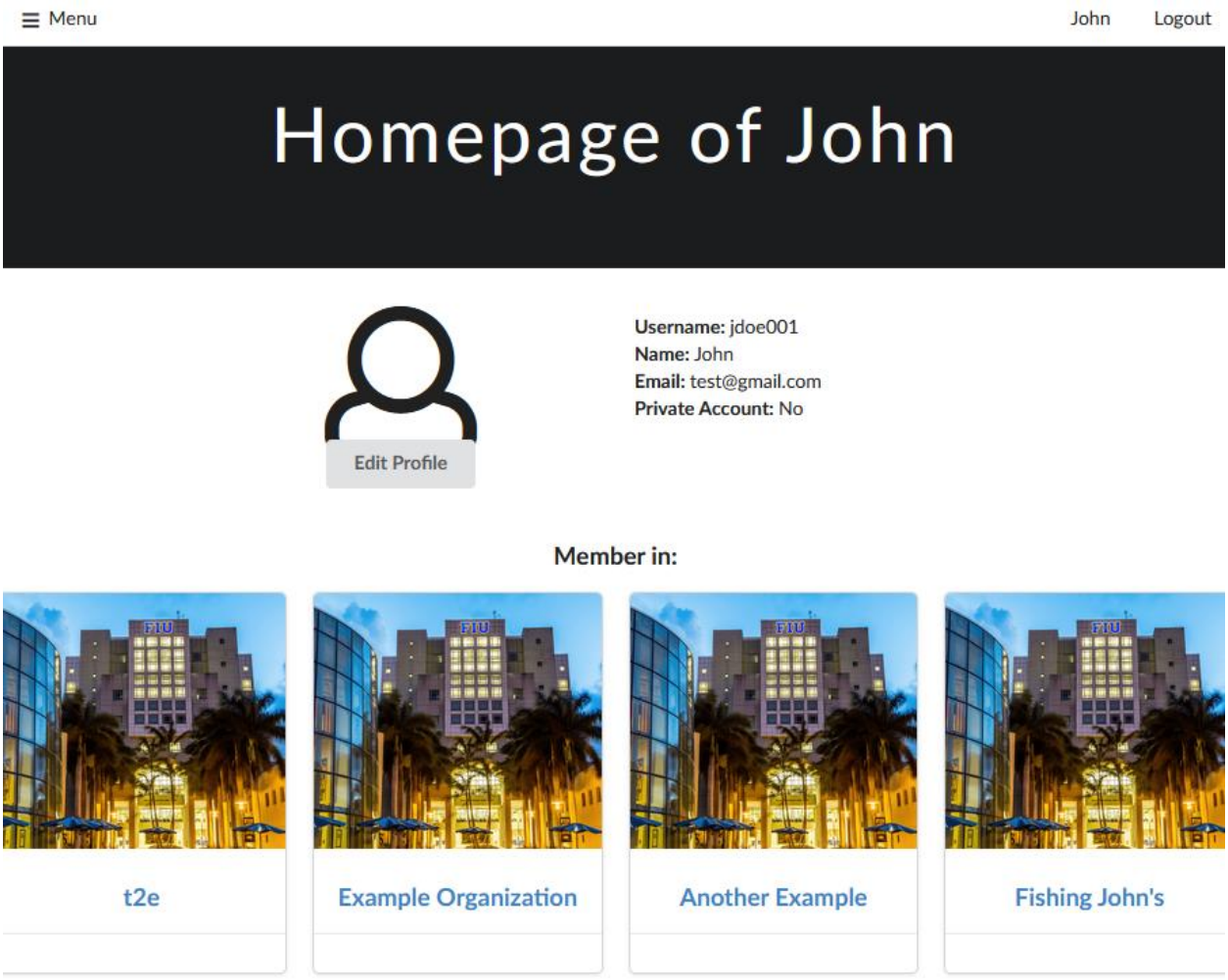

Figure 44: UI layout for the Profile page of a given user, in this case, Jane Doe. When the user is logged in, a edit profile option is given to the user, which triggers a Edit Profile Form modal (see Figure 38). The placeholder section shall include information about the user's clubs and / or events.

Figure 44: UI layout for the Organization page when logged in, in this case, Example Organization. Inlcudes a menu bar (opens top-down) to manage Organization, Events, and Users.

Figure 45: UI Layout for the Edit Profile Form modal

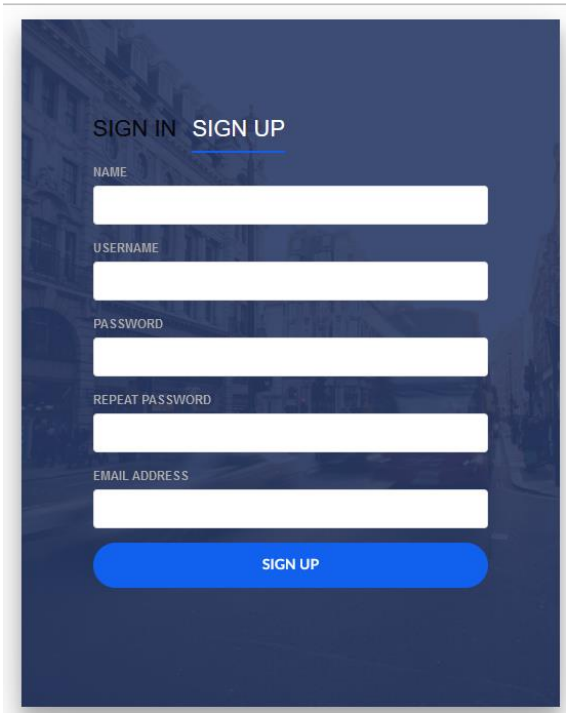

Figure 46: UI Layout for the Signup Form modal

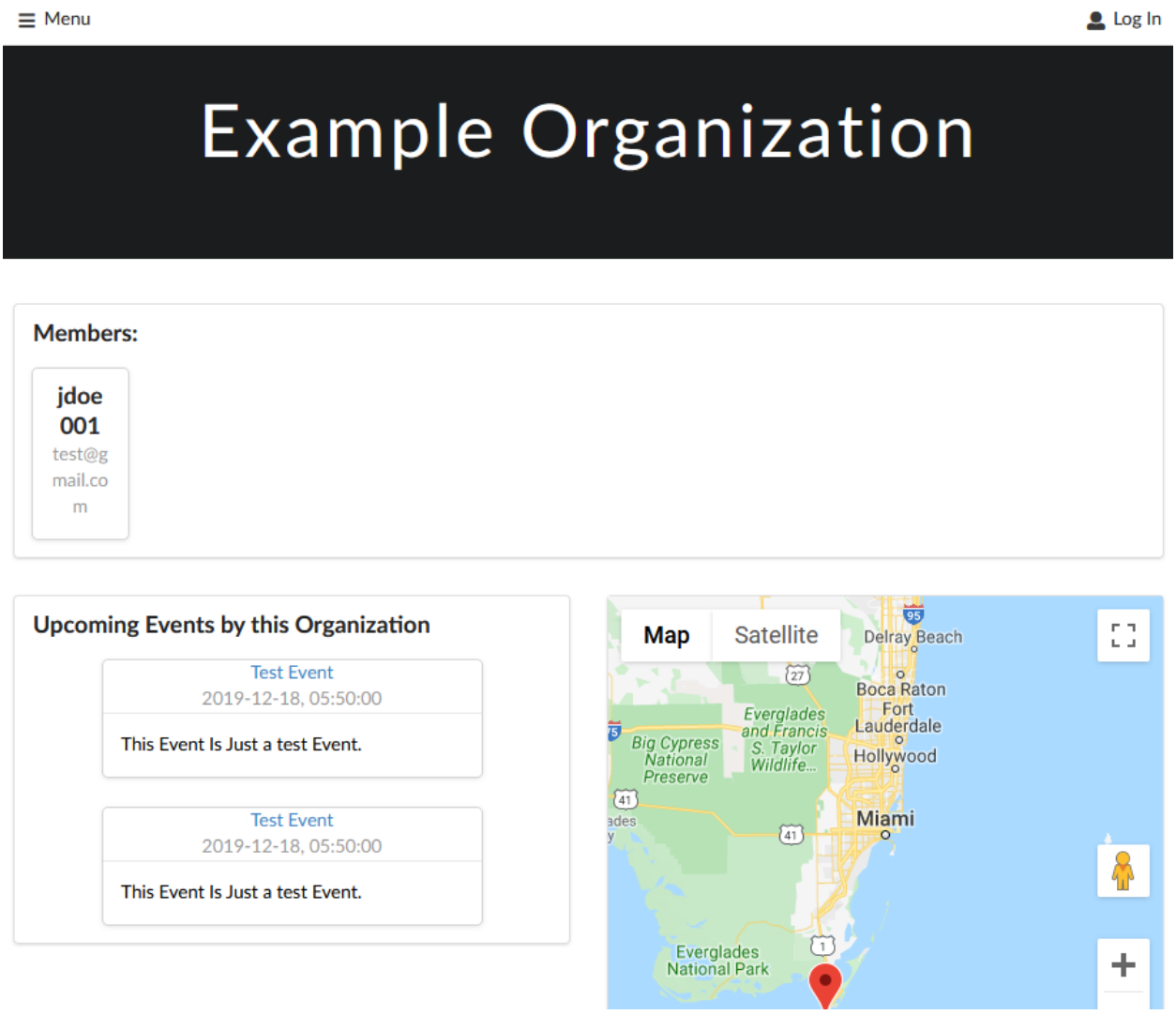

Figure 47: UI Layout for the Organization page when Organizer is not logged in.

Figure 48: UI Layout for the Cancel Event Form modal.

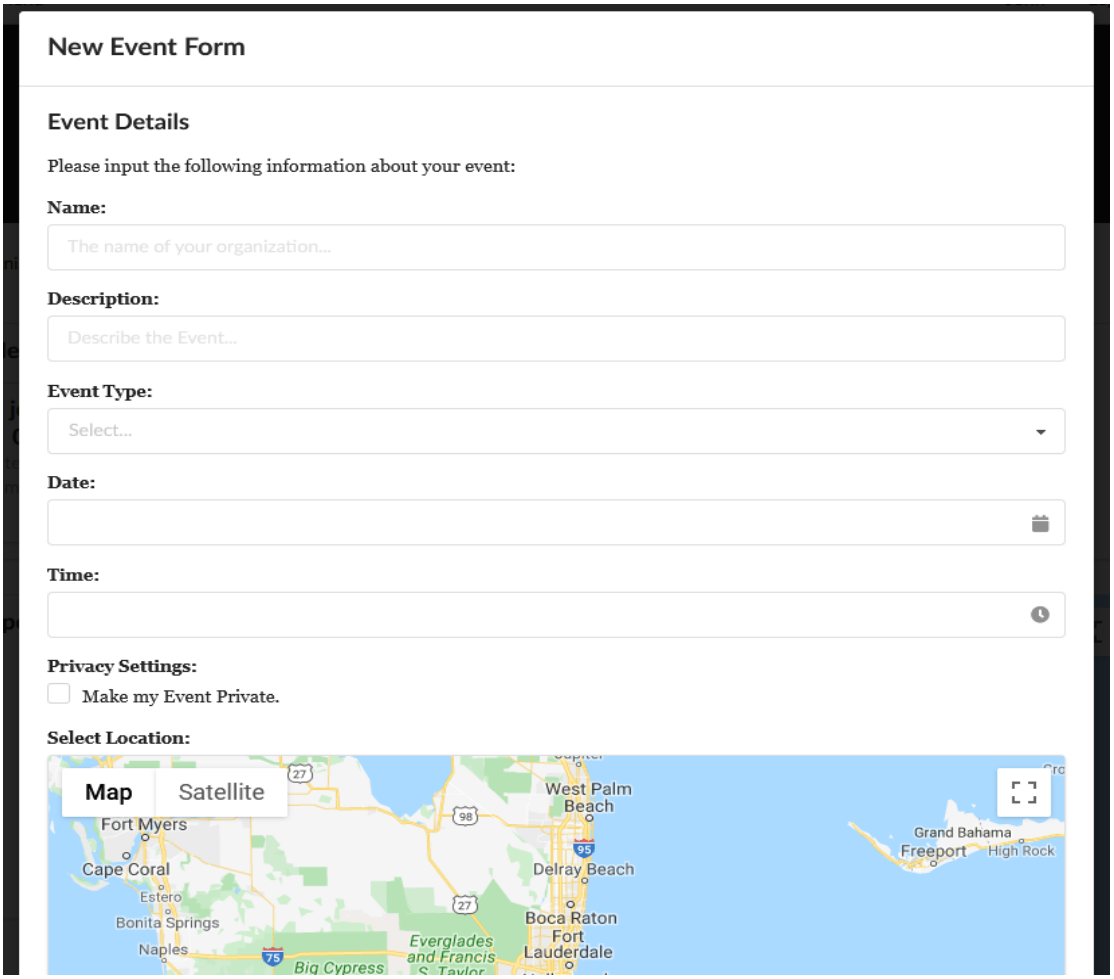

Figure 49: UI Layout for the Create New Event Form Modal.

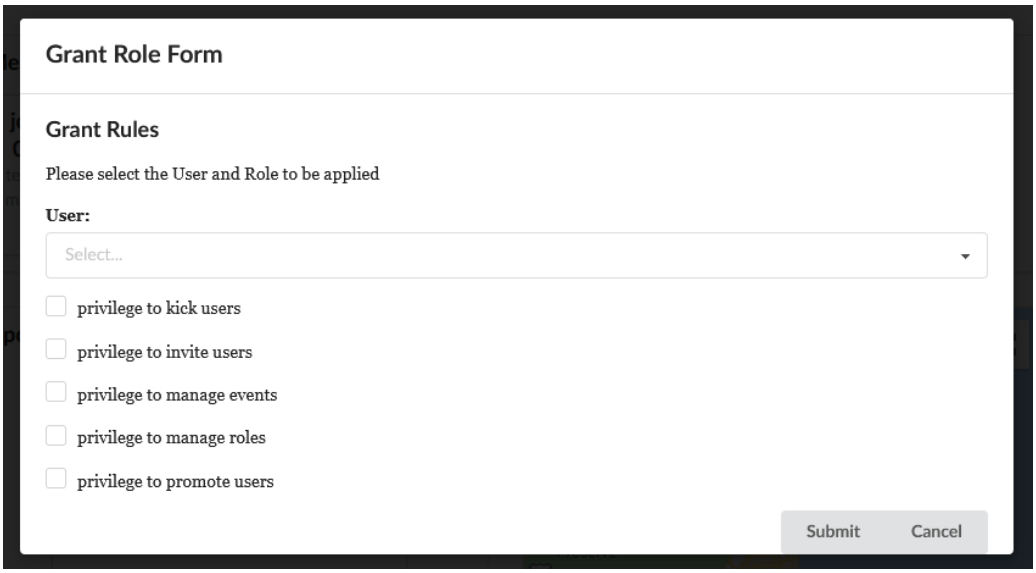

Figure 50: UI Layout for the Grant Role Form Modal.

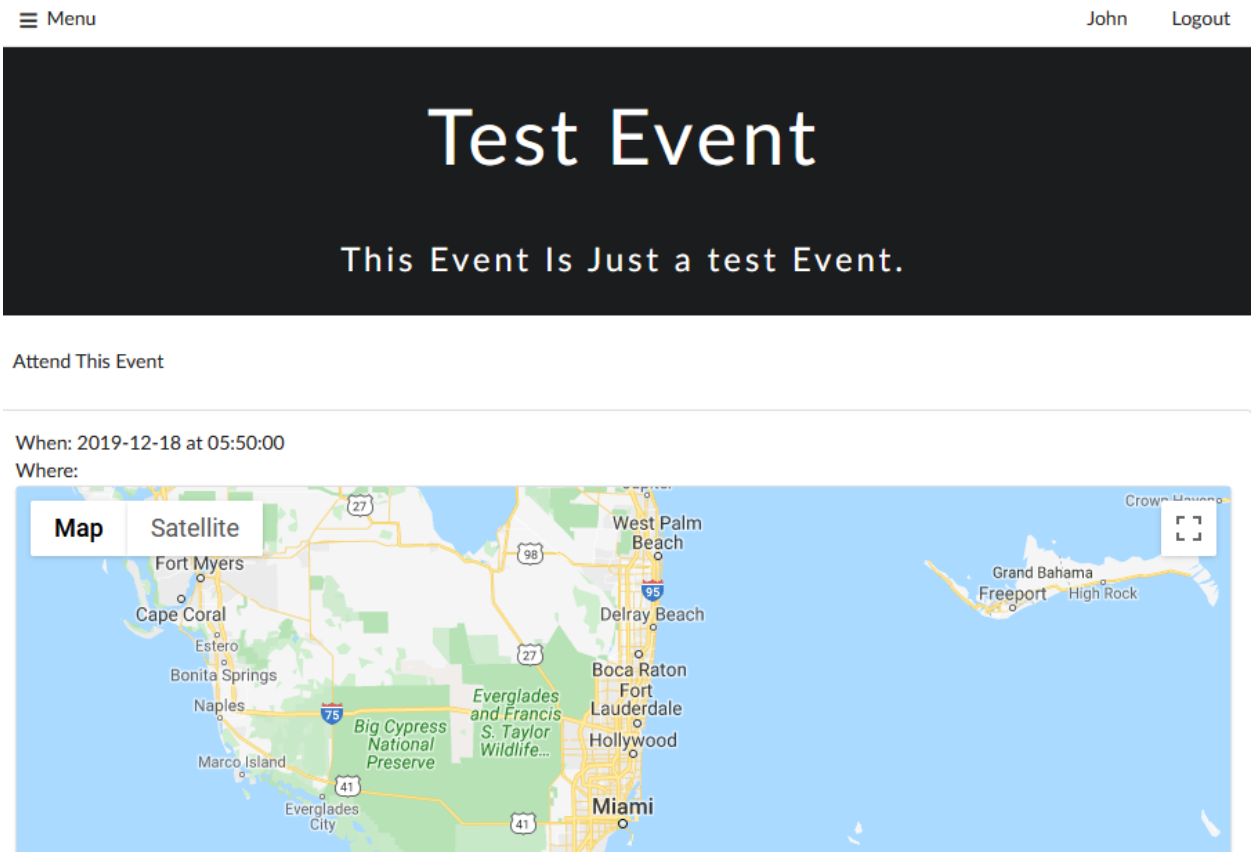

Figure 51: UI Layout for the Event View page.

11.4 Appendix D – Detailed Subsystem Class Diagrams

11.4.1 SOS Website

The full class diagram can be seen in Figure 52.

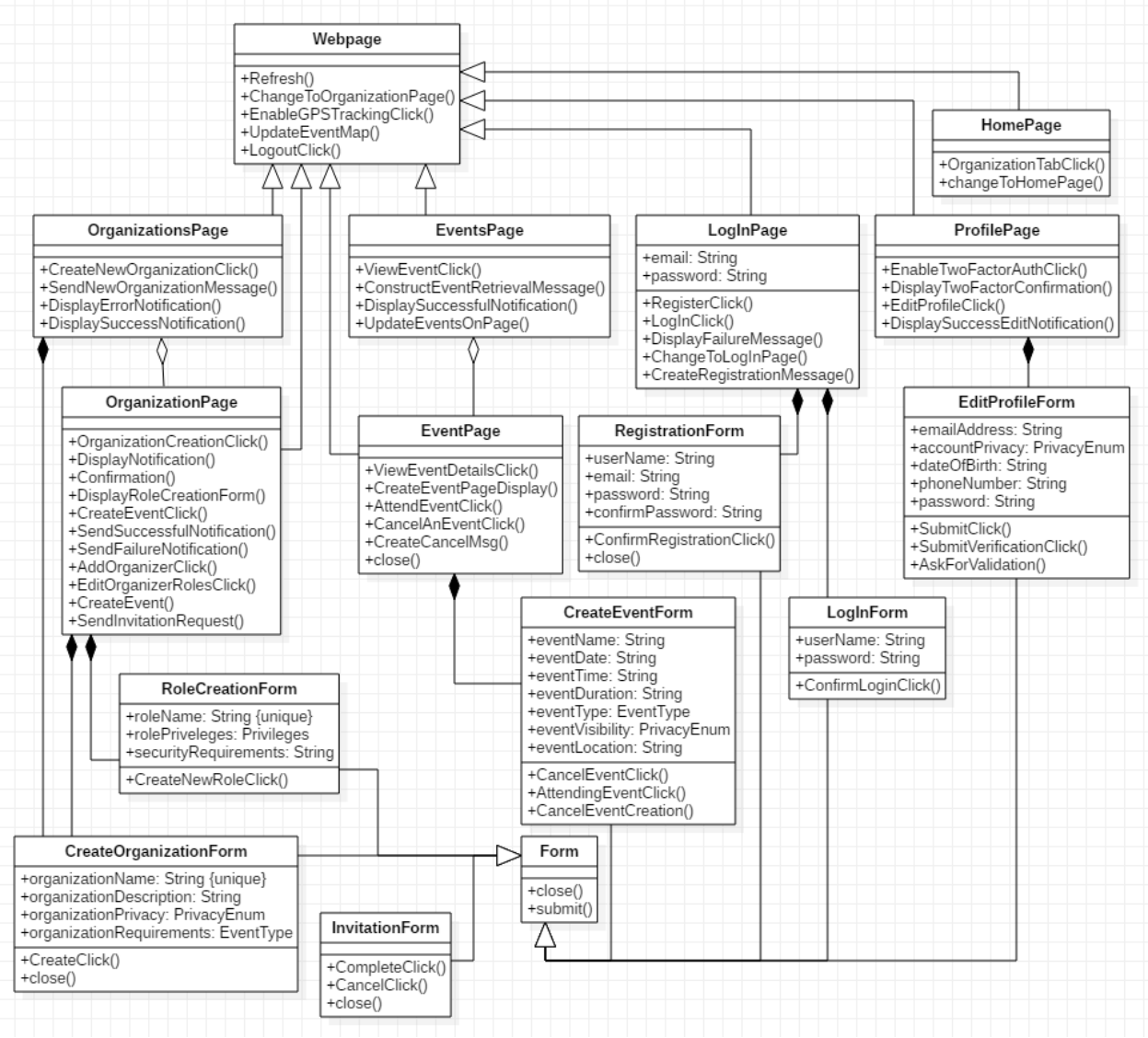

Figure 52: Full Class Diagram for the SOS Website subsystem.

11.4.2 SOS Interface

The full class diagram can be seen in Figure 53.

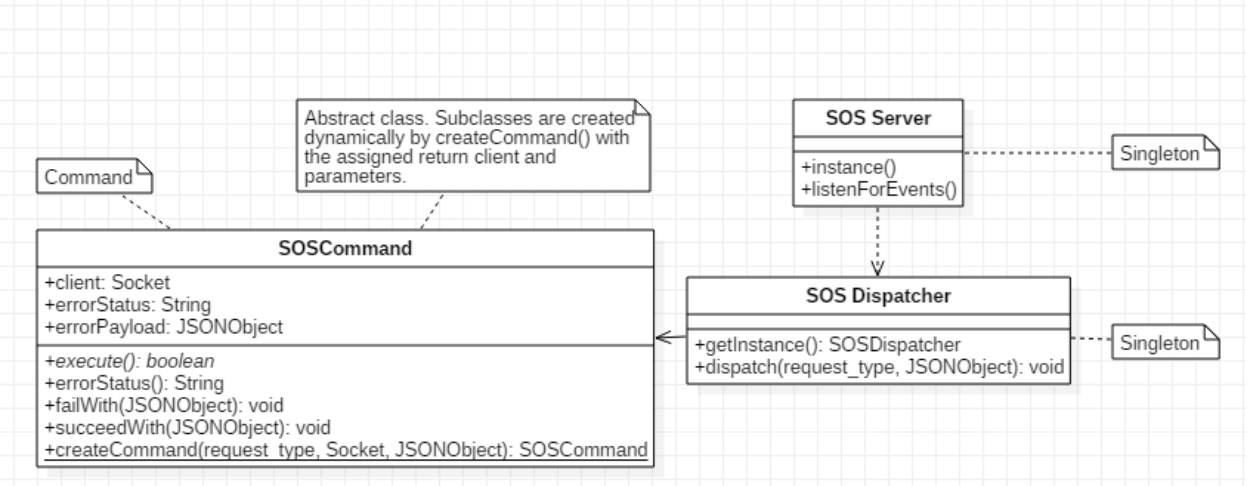

Figure 53: Full Class Diagram for the SOS Interface.

11.4.3 User Management

The full class diagram can be seen in Figure 54.

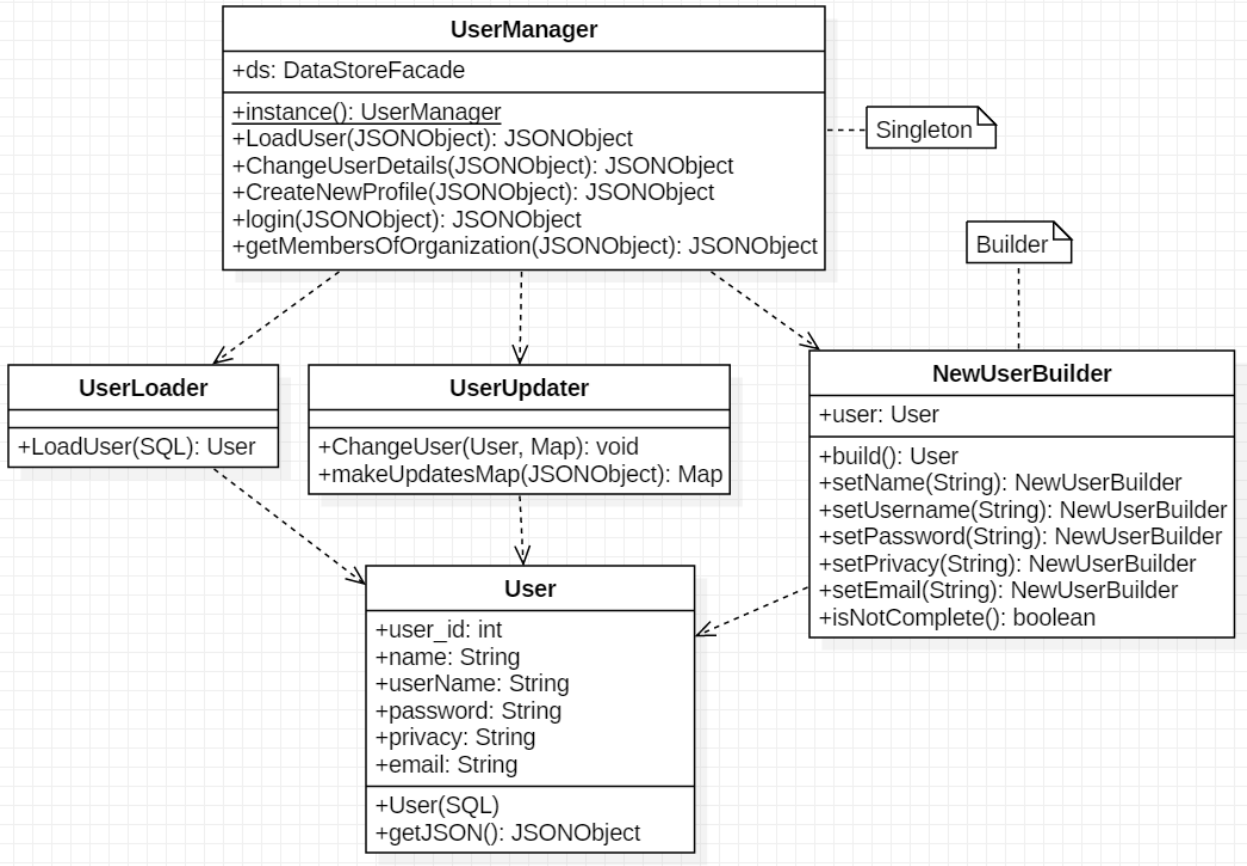

Figure 54: Full Class Diagram for the User Management.

11.4.4 Event Management

The full class diagram can be seen in Figure 55.

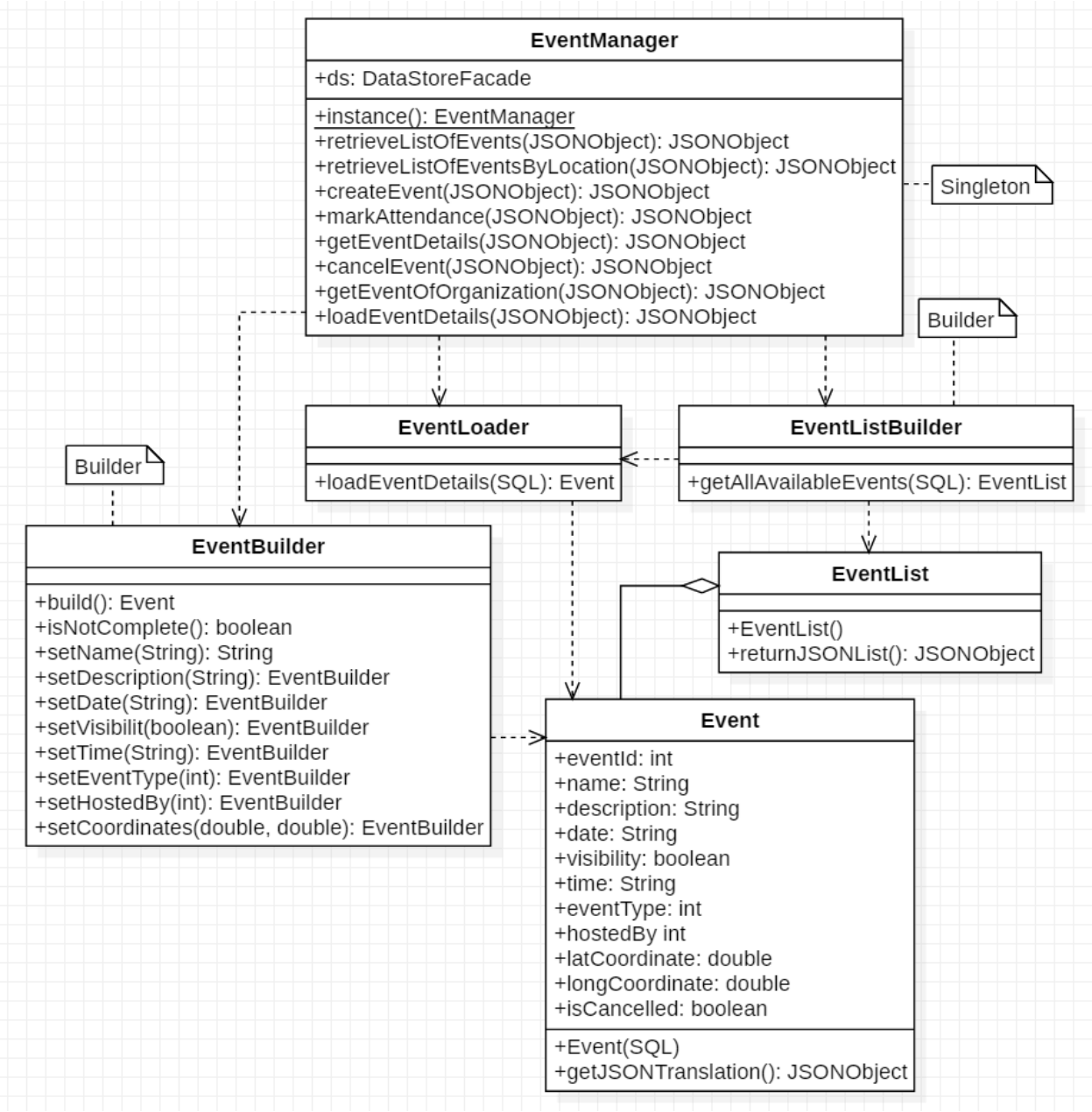

Figure 55: Full Class Diagram for the Event Management.

11.4.5 Organization Management

The full class diagram can be seen in Figure 56.

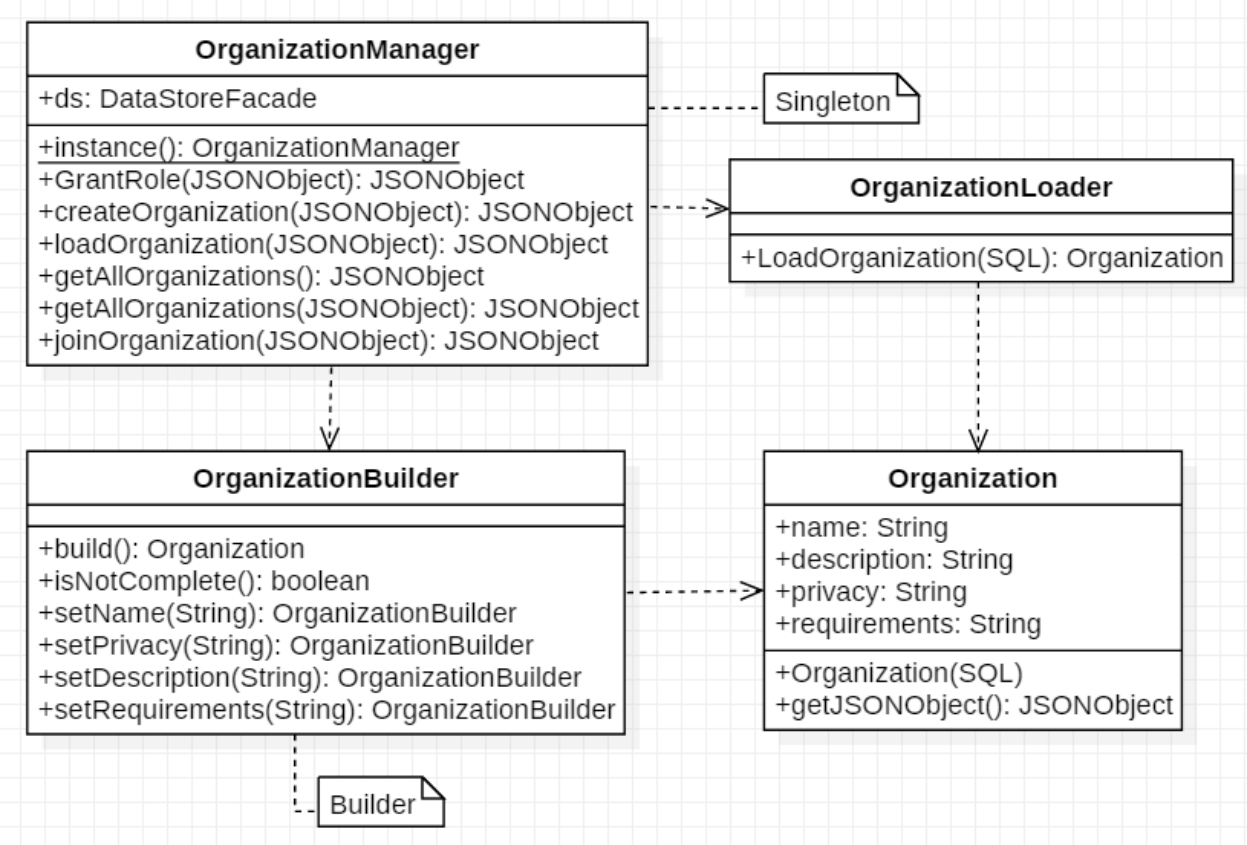

Figure 56: Full Class Diagram for the Organization System.

11.4.6 Security Management

The full class diagram can be seen in Figure 57.

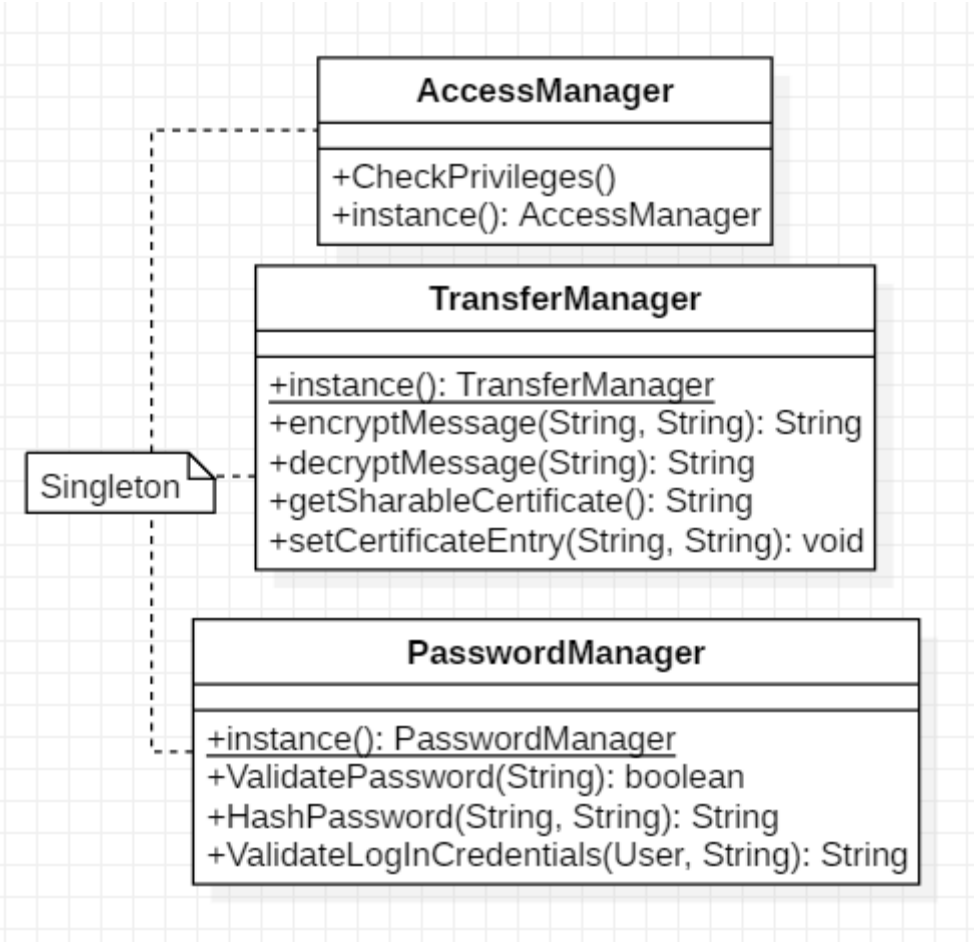

Figure 57: Full Class Diagram for the Security System.

11.4.7 SOS Storage

The full class diagram can be seen in Figure 58.

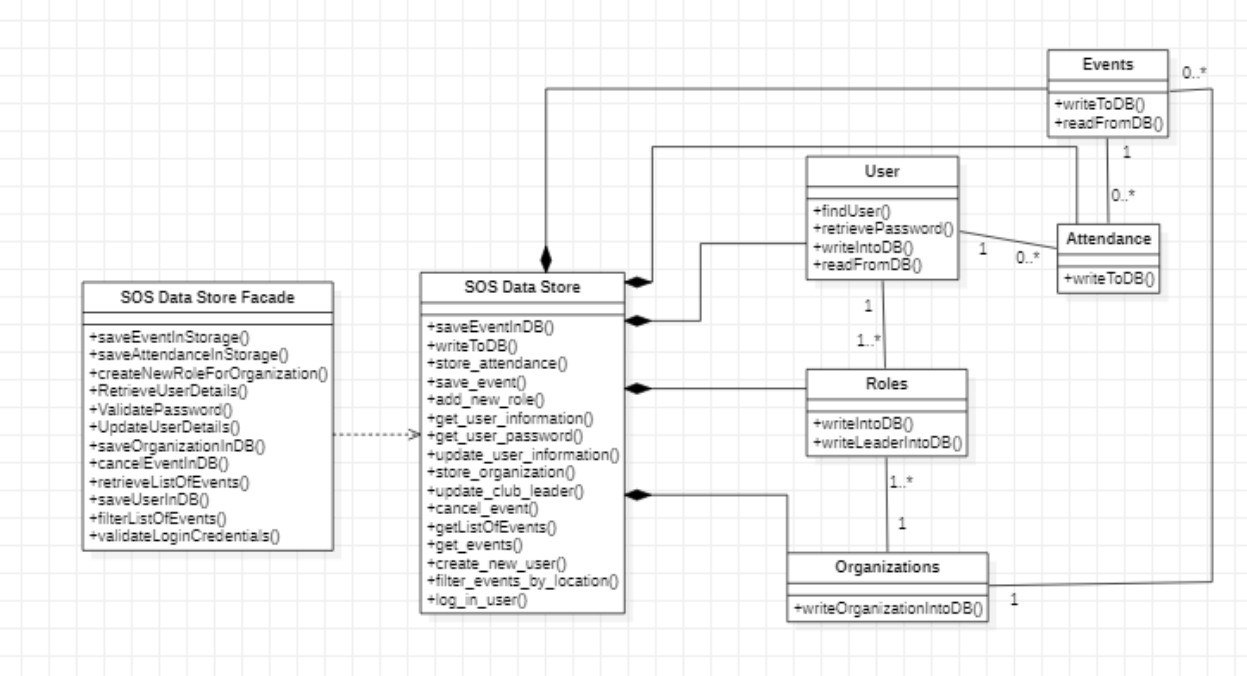

Figure 58: Full Class Diagram for the SOS Storage.

11.5 Appendix E - Class Interfaces

Package event

Class Summary

Event

A run-time representation of an Event persistent Object.

EventBuilder

A Builder class which creates new Events.

EventList

A class that aggregates Events.

EventListBuilder

A Builder which creates new EventList objects.

EventLoader

The Class EventLoader.

EventManager

EventManager, which is a Singleton which manages all the Event functions.

---

event

Class Event

```
java.lang.Object
|
+--event.Event
```

---

public class **Event**
extends java.lang.Object

A run-time representation of an Event persistent Object. This class is used as an intermediary for creation, retrieval, and modification of Event data within the Java code (and the JVM). It is encodable (or serializable) to a database format (e.g., SQL Entry).

Fields

longCoordinate

```
protected double longCoordinate
```
The long coordinate.

---

name

```
protected java.lang.String name
```
The name.

---

time

```
protected java.lang.String time
```
The time.

---

visibility

```
protected boolean visibility
```
The visibility.

Constructors

Event

```
protected      Event()
```

Constructs a new Event class. Called through the EventBuilder class. Attribute assignations are done through protected scope.

Event

```
protected      Event(java.sql.ResultSet results)
throws java.lang.Exception
```

Creates an Event from a given ResultSet.

Parameters:

results - the input ResultSet

Throws:

java.lang.Exception - Thrown if the ResultSet is missing any event-defining variable.

Methods

getDate

`public java.lang.String **getDate**()`

Gets the date.

Returns:

the date

---

getDescription

`public java.lang.String **getDescription**()`

Gets the description.

Returns:

the description

---

getEventID

`protected int **getEventID**()`

Gets the event ID.

Returns:

the event id

getEventType

`public int **getEventType**()`

Gets the event type.

Returns:

the eventType

---

getHostedBy

`public int **getHostedBy**()`

Gets the hosted by.

Returns:

the hostedBy

---

getJSON

`protected org.json.JSONObject` **`getJSON`**`()`

Gets the json.

Returns:

the json

---

getJsonTranslation

`public org.json.JSONObject` **`getJsonTranslation`**`()`

Gets the json translation.

Returns:

the jsonTranslation

---

getLatCoordinate

`public double` **`getLatCoordinate`**`()`

Gets the lat coordinate.

Returns:

the latCoordinate

---

getLongCoordinate

`public double` **`getLongCoordinate`**`()`

Gets the long coordinate.

Returns:

the longCoordinate

---

getName

`public java.lang.String` **`getName`**`()`

Gets the name.

Returns:

the name

---

getTime

```
public java.lang.String getTime()
```

Gets the time.

Returns:

the time

---

isVisibility

```
public boolean isVisibility()
```

Checks if is visibility.

Returns:

the visibility

---

event

Class EventBuilder

```
java.lang.Object
|
+--event.EventBuilder
```

---

public class **EventBuilder**
extends java.lang.Object

A Builder class which creates new Events. It is used to decouple the parts of the process of creating a new Event from the actual Event class, which is intended to only be a data wrapper class. Namely, this class implements the checks and validations necessary to create a valid Event and will reject invalid ones.

Constructors

EventBuilder

```
public      EventBuilder()
```

Creates a new EventBuilder to instantiate the new Event.

Methods

build

```
public event.Event build()
```

Builds the event.

Returns:

the event if the build is complete, throws an error otherwise.

---

isNotComplete

```
public boolean isNotComplete()
```

Checks if is not complete.

Returns:

true if the event is not complete, false otherwise.

---

setCoordinates

```
public event.EventBuilder setCoordinates(double lat, double
logn)
```

Sets the coordinates of the event location.

Parameters:

lat - the latitute of the location
logn - the longitude of the location

Returns:

the event builder.

---

setDate

```
public event.EventBuilder setDate(java.lang.String date)
```

Sets the date of the event.

Parameters:

date - the date of the event.

Returns:

the EventBuilder.

---

setDescription

```
public event.EventBuilder
setDescription(java.lang.String description)
```

Sets the description of the event.

Parameters:

description - the description of the event.

Returns:

the EventBuilder.

---

setEventType

```
public event.EventBuilder setEventType(int eventType)
```

Sets the type of the event.

Parameters:

eventType - the type of the event.

Returns:

the EventBuilder.

---

setHostedBy

```
public event.EventBuilder setHostedBy(int
organization_id)
```

Sets the id of the host.

Parameters:

organization_id - the id of the host.

Returns:

the EventBuilder.

---

setName

```
public event.EventBuilder setName(java.lang.String name)
```

Sets the name of the event.

Parameters:

name - the name of the event.

Returns:

the EventBuilder.

---

setTime

```
public event.EventBuilder setTime(java.lang.String time)
```

Sets the time of the event.

Parameters:

time - the time of the event.

Returns:

the EventBuilder.

---

setVisibility

```
public event.EventBuilder setVisibility(boolean
visibility)
```

Sets the visibility of the event.

Parameters:

visibility - the visibility of the event.

Returns:

the EventBuilder.

---

event

Class EventList

```
java.lang.Object
|
+--event.EventList
```

---

public class **EventList**
extends java.lang.Object

A class that aggregates Events.

Constructors

EventList

```
protected      EventList()
```

Constructs a new EventList class. Called through the EventListBuilder class. Attribute assignations are done through protected scope.

EventList

```
protected  EventList(java.sql.ResultSet results)
           throws java.lang.Exception
```

Contrusts a new EventList from the contents of the ResultSet.

Parameters:

results - the ResultSet containing a collection of Event SQL entries.

Throws:

java.lang.Exception - thrown if errors occur while parsing the ResultSet

Methods

returnJSONList

```
public org.json.JSONArray returnJSONList()
```

Return JSON list.

Returns:

the JSONArray containing the list of Events

---

event

Class EventListBuilder

```
java.lang.Object
|
+--event.EventListBuilder
```

---

public class **EventListBuilder**
extends java.lang.Object

A Builder which creates new EventList objects. As other builders, it is used to decouple the process of creating a new EventList from the actual EventList class, and also provides functions implementing attribute-base filtering (e.g., filter by location, or by hosting organization, etc.)

Constructors

EventListBuilder

```
public      EventListBuilder()
```

Creates a new EventListBuilder to instantiate the new EventList.

Methods

getAllAvailableEvents

```
public event.EventList
getAllAvailableEvents(java.sql.ResultSet set) throws
java.lang.Exception
```

Creates an EventList with all the events in the given result set.

Parameters:

set - the result set

Returns:

the EventList

Throws:

java.lang.Exception - the exception

---

event

Class EventLoader

```
java.lang.Object
|
+--event.EventLoader
```

---

public class **EventLoader**
extends java.lang.Object

The Class EventLoader.

Constructors

EventLoader

```
public      EventLoader()
```

Methods

loadEventDetails

```
public event.Event loadEventDetails(java.sql.ResultSet
results)
throws java.lang.Exception
```

Loads an Event from the given ResultSet.

Parameters:

results - the ResultSet

Returns:

the Event

Throws:

java.lang.Exception - Happens when the ResultSet

---

event

Class EventManager

```
java.lang.Object
|
+--event.EventManager
```

---

public class **EventManager**
extends java.lang.Object

EventManager, which is a Singleton which manages all the Event functions. This class receives dispatched actions from the SOS Dispatcher and completes that action using objects internal to its subsystem. It also is in charge of interacting with the SOS Data Store Façade directly. Part of the role of this class is to parse front-end format data (e.g., JSON-String description of new Events) and calling the appropriate functions on other classes according to that data. It is also in charge of encoding Event objects into database-format (e.g., SQL Table entries). Another role is to create EventLists based on filter requests through the EventListBuilder.

Constructors

EventManager

```
protected      EventManager()
```

A protected or private constructor ensures that no other class has access to the Singleton.

Methods

cancelEvent

```
public org.json.JSONObject
cancelEvent(org.json.JSONObject payload)
```

Cancels the given event.

Parameters:

payload - The event that is going to be cancelled.

Returns:

the JSON object

---

createEvent

```
public org.json.JSONObject
createEvent(org.json.JSONObject json)
```

Creates a new Event from a json Event description. Done by calling the EventBuilder class.

Parameters:

json - the json

Returns:

the JSON object

---

getEventOfOrganization

```
public org.json.JSONObject
getEventOfOrganization(org.json.JSONObject payload)
```

Returns all the Events hosted by a given Organization.

Parameters:

payload - A JSONObject which containts the following keys: organization a JSONObject with the following keys: organization_id which is the id of the target organization.

Returns:

All events hosted by the organization.

---

instance

```
public static event.EventManager instance()
```

Instance.

Returns:

The unique instance of this class.

---

loadEventDetails

```
public org.json.JSONObject
loadEventDetails(org.json.JSONObject payload)
```

Gets event information based on the ID that is provided.

Parameters:

payload - The ID of the event that details are being requested.

Returns:

A JSON object with the event details.

---

markAttendance

```
public org.json.JSONObject
markAttendance(org.json.JSONObject payload)
```

Marks a User as attending an Event by creating an entry on the Attendance table.

Parameters:

payload - the payload

Returns:

the JSON object

---

retrieveListOfEvents

```
public org.json.JSONObject
retrieveListOfEvents(org.json.JSONObject payload)
```

Retrieves a list of events that are stored in the database.

Parameters:

payload - the payload

Returns:

A JSON array of events.

---

retrieveListOfEventsByLocation

```
public org.json.JSONObject
retrieveListOfEventsByLocation(org.json.JSONObject
payload)
```

Retrieves a list of Event given a location.

Parameters:

payload - A JSONObject with the following keys: lantitude the latitute of the search center longitude the longitude of the search center

Returns:

All events within 0.5 latitude/longitude range from the given center.

Package organization

Class Summary

Organization

A run-time representation of an Organization persistent object.

OrganizationBuilder

A Builder which creates new Organization objects.

OrganizationLoader

A class which creates an Organization object from an Organization database object.

OrganizationManager

A Singleton which manages all the Organization functions.

---

organization

Class Organization

```
java.lang.Object
|
+--organization.Organization
```

---

public class **Organization**
extends java.lang.Object

A run-time representation of an Organization persistent object. This class is used as an intermediary for creation, retrieval, and modification of Organization data within the Java code (and the JVM). It is encodable (or serializable) to a database format (e.g., SQL Entry)

Fields

description

`protected java.lang.String` **`description`**
The description.

---

name

`protected java.lang.String` **`name`**
The name.

---

privacy

`protected java.lang.String` **`privacy`** The
privacy.

---

requirements

`protected java.lang.String` **`requirements`** The
requirements.

Constructors

Organization

`protected` **`Organization`**`()`

Constructs a new Organization class. Called through the OrganizationBuilder class. Attribute assignations are done through protected scope.

Organization

`protected` **`Organization`**`(java.sql.ResultSet results) throws java.lang.Exception`

Creates an Organization from the target ResultSet.

Parameters:

results - the result set.

Throws:

java.lang.Exception - thrown if there's an error, like using the incorrect entry.

Methods

getDescription

```
public java.lang.String getDescription()
```

Gets the description.

Returns:

the description

---

getJSONObject

```
protected org.json.JSONObject getJSONObject()
```

Gets the JSON object.

Returns:

the json translation of this Organization.

---

getJsonTranslation

```
public org.json.JSONObject getJsonTranslation()
```

Gets the json translation.

Returns:

the jsonTranslation

---

getName

```
public java.lang.String getName()
```

Gets the name.

Returns:

the name

---

getPrivacy

```
public java.lang.String getPrivacy()
```

Gets the privacy.

Returns:

the privacy

---

getRequirements

```
public java.lang.String getRequirements()
```

Gets the requirements.

Returns:

the requirements

---

organization

Class OrganizationBuilder

```
java.lang.Object
|
+--organization.OrganizationBuilder
```

---

public class **OrganizationBuilder**
extends java.lang.Object

A Builder which creates new Organization objects. It is used to decouple the process, including validations and checks, of creating an Organization from the actual Organization class itself.

Fields

PRIVACY_PRIVATE

```
public static final java.lang.String PRIVACY_PRIVATE
```

The Constant PRIVACY_PRIVATE.

PRIVACY_PUBLIC

```
public static final java.lang.String PRIVACY_PUBLIC
```
The Constant PRIVACY_PUBLIC.

Constructors

OrganizationBuilder

```
public      OrganizationBuilder()
```

Creates a new OrganizationBuilder to instantiate the new Event.

Methods

build

```
public organization.Organization build()
throws
java.lang.IllegalArgumentException
```

Builds the Organization.

Returns:

an Organization object, or throws an error.

Throws:

java.lang.IllegalArgumentException - thrown if the organization being built is not complete.

---

isNotComplete

```
public boolean isNotComplete()
```

Checks if is not complete.

Returns:

true if the organization is not complete false otherwise.

---

setDescription

```
public organization.OrganizationBuilder
setDescription(java.lang.String description)
```

Sets the description.

Parameters:

description - the description.

Returns:

the OrganizationBuilder

---

setName

```
public organization.OrganizationBuilder
setName(java.lang.String name)
```

Sets the name.

Parameters:

name - the name.

Returns:

the OrganizationBuilder

---

setPrivacy

```
public organization.OrganizationBuilder
setPrivacy(java.lang.String privacy)
throws
java.lang.IllegalArgumentException
```

Sets the privacy.

Parameters:

privacy - the privacy.

Returns:

the OrganizationBuilder

Throws:

java.lang.IllegalArgumentException - the illegal argument exception

---

setRequirements

```
public organization.OrganizationBuilder
setRequirements(java.lang.String requirements)
```

Sets the requirements.

Parameters:

requirements - the requirements.

Returns:

the OrganizationBuilder

---

organization

Class OrganizationLoader

```
java.lang.Object
|
+--organization.OrganizationLoader
```

---

public class **OrganizationLoader**
extends java.lang.Object

A class which creates an Organization object from an Organization database object. This class decouples the parsing from the database to the system logic from the OrganizationManager class and can be extended to include internal checks for data integrity purposes.

Constructors

OrganizationLoader

```
public      OrganizationLoader()
```

Methods

LoadOrganization

```
public organization.Organization
LoadOrganization(java.sql.ResultSet set)
throws
java.lang.Exception
```

Creates a Organization from a database-format entry.

Parameters:

set - the set

Returns:

a Organization object with the given attributes.

Throws:

java.lang.Exception - the exception

---

organization

Class OrganizationManager

```
java.lang.Object
|
+--organization.OrganizationManager
```

---

public class **OrganizationManager**
extends java.lang.Object

A Singleton which manages all the Organization functions. This class receives dispatched actions from the SOS Dispatcher and completes that action using objects internal to its subsystem. It also is in charge of interacting with the SOS Data Store Façade directly. Part of the role of this class is to parse front-end format data (e.g., JSON-String description of new Organization) and calling the appropriate functions on other classes according to that data. Another job of this class is to manage Role creation and assignment, as well as mediate the modification of data in an Organization, and of Event hosting.

Constructors

OrganizationManager

```
protected      OrganizationManager()
```

A protected or private constructor ensures that no other class has access to the Singleton.

Methods

createOrganization

```
public org.json.JSONObject
createOrganization(org.json.JSONObject json)
```

Creates an organization in the SOS System.

Parameters:

json - the json

Returns:

the JSON object

---

getAllOrganizations

```
public org.json.JSONObject getAllOrganizations()
```

Gets all the public organizations in the SOS.

Returns:

A JSONArray with all the public organizations stored in the SOS.

---

getAllOrganizations

```
public org.json.JSONObject
getAllOrganizations(org.json.JSONObject payload)
```

Gets all the organizations which a user currently belongs to.

Parameters:

payload - the payload

Returns:

A JSONObject with all the organizations the user is a part of.

---

grantRole

```
public org.json.JSONObject grantRole(org.json.JSONObject
payload)
```

Grants a number of privileges to a User for a given Organization.

Parameters:

payload - the payload

Returns:

the JSON object

---

instance

```
public static organization.OrganizationManager
instance()
```

Instance.

Returns:

The unique instance of this class.

---

joinOrganization

```
public org.json.JSONObject
joinOrganization(org.json.JSONObject payload)
```

Allows the user to join an organization that is part of SOS.

Parameters:

payload - the payload

Returns:

the JSON object

---

loadOrganizationDetails

```
public org.json.JSONObject
loadOrganizationDetails(org.json.JSONObject payload)
```

Loads details for the requested organization.

Parameters:

payload - the payload

Returns:

the JSON object

Package security

Class Summary

AccessManager

A Singleton dealing with access control actions.

PasswordManager

A Singleton which deals with password control actions.

TransferManager

A Singleton that handles secure data exchange between the front end and the back end.

security

Class AccessManager

```
java.lang.Object
|
+--security.AccessManager
```

public class **AccessManager**
extends java.lang.Object

A Singleton dealing with access control actions. It implements most of the back-end side of the access policy for SOS and host the relevant Enumerations for access permissions and other privileges. It also must be called to do checks on the relevant actions, such as creating events, deleting profiles, etc.

Constructors

AccessManager

`protected` **`AccessManager`**`()`

A protected or private constructor ensures that no other class has access to the Singleton.

Methods

CheckPrivileges

```
public boolean CheckPrivileges()
```

Check privileges.

Returns:

The result of privilege check for the current user class.

---

instance

```
public static security.AccessManager instance()
```

Instance.

Returns:

The unique instance of this class.

---

security

Class PasswordManager

```
java.lang.Object
|
+--security.PasswordManager
```

---

public class **PasswordManager**
extends java.lang.Object

A Singleton which deals with password control actions. It implements most of the back-end side of the password policy for SOS, including resolving passwords and checking the input password against the database.

Constructors

PasswordManager

```
protected      PasswordManager()
```

The constructor could be made private to prevent others from instantiating this class. But this would also make it impossible to create instances of PasswordManager subclasses.

Methods

HashPassword

```
public static java.lang.String
HashPassword(java.lang.String username,
java.lang.String password)
```

Hash password.

Parameters:

username - the username
password - is a String to be validated

Returns:

will return an encrypted version of the password as a String

ValidateLogInCredentials

```
public static boolean ValidateLogInCredentials(user.User
user,
java.lang.String pwd)
```

Validate log in credentials.

Parameters:

user - the user
pwd - is the user's password for log in

Returns:

is the validation of the login credentials

ValidatePassword

```
public static boolean ValidatePassword(java.lang.String
password)
```

Validate password.

Parameters:

password - as a String to be validated

Returns:

is true if password successfully validates

instance

```
public static security.PasswordManager instance()
```

Instance.

Returns:

The unique instance of this class.

security

Class TransferManager

```
java.lang.Object
|
+--security.TransferManager
```

public class **TransferManager**
extends java.lang.Object

A Singleton that handles secure data exchange between the front end and the back end.

Methods

decryptMessage

```
public java.lang.String
decryptMessage(org.json.JSONObject msg)
```

Decrypts the message in the given JSONObject.

Parameters:

msg - a json object which must have: key a symmetric key encrypted with this TransferManager's public certificate. iv an
iv value encrypted with this TransferManager's public certificate. text a cyphertext encrypted using AES/CBC/PKCS5Padding and the given key and iv.

Returns:

the plaintext form of text.

encryptMessage

```
public org.json.JSONObject
encryptMessage(java.lang.String msg,
java.lang.String alias)
```

Produces an Base64 version of the encrypted ciphertext for the input given in msg.

Parameters:

msg - the input to be encrypted and encoded.
alias - the alias

Returns:

the JSONObject containing: 'key' the encrypted key parameter, decryptable with the target's private key. 'iv' the encrypted iv parameter, decryptable with the target's private key. 'text' the encrypted text, decrypateble with a AES/CBC/PKCS7Padding using the given key and iv.

---

getSharableCertificate

```
public java.lang.String getSharableCertificate()
```

Gets the sharable (PEM) string version of this object's certificate.

Returns:

a String containing the PEM version of the certificate.

---

instance

```
public static security.TransferManager instance()
```

Instance.

Returns:

The unique instance of this class.

---

setCertificateEntry

```
public void setCertificateEntry(java.lang.String certS,
                                java.lang.String alias)
```

Adds an external certificate to the KeyStore with the given alias.

Parameters:

certS - the certificate, usually in a PEM format.
alias - the alias for the certificate.

Package sosInterface

Class Summary

SOSCommand

The Class SOSCommand.

SOSDispatcher

The Class SOSDispatcher.

SOSDispatcher.REQUEST_TYPES

The Enum REQUEST_TYPES.

SOSServer

SOSServer communicates with the front-end for creation of events.

SOSServer_Driver

The Class SOSServer_Driver.

---

sosInterface

Class SOSCommand

```
java.lang.Object
|
+--sosInterface.SOSCommand
```

---

public abstract class **SOSCommand**
extends java.lang.Object

The Class SOSCommand.

Fields

client

```
protected com.corundumstudio.socketio.SocketIOClient
```
client
The client.

---

errorPayload

`protected org.json.JSONObject` **`errorPayload`**
The error payload.

---

errorStatus

`protected java.lang.String` **`errorStatus`**
The error status.

Constructors

SOSCommand

`protected`
`SOSCommand``(com.corundumstudio.socketio.SocketIOClient client)`

Creates an SOSCommand Object which will report to the given client.

Parameters:

client - the client for this SOSCommand.

Methods

createCommand

```
public static sosInterface.SOSCommand
createCommand(sosInterface.SOSDispatcher.REQUEST_TYPES
request,
com.corundumstudio.socketio.SocketIOClient client,
org.json.JSONObject payload)
```

Creates a Command subclass which implements one of the commands of the server. The list of commands can be seen in the SOSDispatcher.REQUEST_TYPES enumeration. Each subclass implements the execute function which instantiates and calls the relevant action on the managers of the relevant classes.

Parameters:

request - the request type.
client - the client to be passed.
payload - the payload of the request.

Returns:

the SOSCommand object implementing the dispatchable action.

---

errorStatus

```
public java.lang.String errorStatus()
```

Returns the stored error status, which is set by the execute function in case of errors.

Returns:

the string

---

execute

```
public abstract boolean execute()
throws
java.lang.RuntimeException
```

Executes the command. Must be implemeted by subclasses.

Returns:

true if the command executed successfully, false otherwise.

Throws:

java.lang.RuntimeException - the runtime exception

failWith

```
protected void failWith(org.json.JSONObject
errorPayload)
```

Reports a failure to the client, with the given payload.

Parameters:

errorPayload - the failure body.

succeedWith

```
protected void succeedWith(org.json.JSONObject
successPayload)
```

Reports a success to the client, with the given payload.

Parameters:

successPayload - the payload of the success.

sosInterface

Class SOSDispatcher

```
java.lang.Object
|
+--sosInterface.SOSDispatcher
```

public class **SOSDispatcher**
extends java.lang.Object

The Class SOSDispatcher.

Constructors

SOSDispatcher

```
protected      SOSDispatcher()
```

Creates new dispatcher.

thods

dispatch

```
public void
dispatch(sosInterface.SOSDispatcher.REQUEST_TYPES
request,
com.corundumstudio.socketio.SocketIOClient client,
org.json.JSONObject payload)
```

Dispatches an action by creating and executing an SOSCommand.

Parameters:

request - the request.
client - the client to return the request action.
payload - the payload of the request.

---

getInstance

```
public static sosInterface.SOSDispatcher getInstance()
```

Returns the Singleton dispatcher instance.

Returns:

the unique SOSDispatcher.

---

sosInterface

Class SOSDispatcher.REQUEST_TYPES

```
java.lang.Object
|
+--java.lang.Enum
|
+--sosInterface.SOSDispatcher.REQUEST_TYPES
```

All Implemented Interfaces:
java.io.Serializable, java.lang.Comparable

---

public static final class **SOSDispatcher.REQUEST_TYPES** extends java.lang.Enum

The Enum REQUEST_TYPES.

Fields

ATTEND_EVENT

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES ATTEND_EVENT
```
The attend event.

CREATE_EVENT

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES CREATE_EVENT
```
The create event.

CREATE_ORG

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES CREATE_ORG
```
The create org.

CREATE_USER

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES CREATE_USER
```
The create user.

EVENT_CANCEL

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES EVENT_CANCEL
```
The event cancel.

JOIN_ORG

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES JOIN_ORG
```

The join org.

LOAD_USER

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES LOAD_USER
```
The load user.

LOGIN

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES LOGIN
```
The login.

RETR_ALL_EVENTS

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES RETR_ALL_EVENTS
```
The retr all events.

RETR_EVENT

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES RETR_EVENT
```
The retr event.

RETR_EVENTS_BY_LOCATION

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES
RETR_EVENTS_BY_LOCATION
```
The retr events by location.

RETR_EVENTS_FOR_ORG

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES
RETR_EVENTS_FOR_ORG
```
The retr events for org.

RETR_EVENT_FOR_ORG

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES
RETR_EVENT_FOR_ORG
```
The retr event for org.

RETR_MEMBER_FOR_ORG

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES
RETR_MEMBER_FOR_ORG
```
The retr member for org.

RETR_ORG

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES RETR_ORG
```
The retr org.

RETR_ORGS

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES RETR_ORGS
```
The retr orgs.

RETR_ORGS_FOR_USER

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES
RETR_ORGS_FOR_USER
```
The retr orgs for user.

SET_ROLE

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES SET_ROLE
```
The set role.

UPDATE_USER

```
public static final
sosInterface.SOSDispatcher.REQUEST_TYPES UPDATE_USER
```
The update user.

Methods

valueOf

```
public static sosInterface.SOSDispatcher.REQUEST_TYPES
valueOf(java.lang.String name)
```

values

```
public static sosInterface.SOSDispatcher.REQUEST_TYPES[]
values()
```

sosInterface

Class SOSServer

```
java.lang.Object
|
+--sosInterface.SOSServer
```

public class **SOSServer**
extends java.lang.Object

SOSServer communicates with the front-end for creation of events. Also it is held responsible for managing user sessions and keeping track of them, as well as dispatching messages through the system.

Methods

ListenForEvents

```
public void ListenForEvents()
```

Starts the server and sets it to listen for events from a client socket.io front-end.

---

instance

```
public static sosInterface.SOSServer instance()
throws
java.lang.Exception
```

Instance.

Returns:

The unique instance of this class.

Throws:

java.lang.Exception - the exception

---

sosInterface

Class SOSServer_Driver

```
java.lang.Object
|
+--sosInterface.SOSServer_Driver
```

---

public class **SOSServer_Driver**
extends java.lang.Object

The Class SOSServer_Driver.

Constructors

SOSServer_Driver

```
public      SOSServer_Driver()
```

Methods

main

```
public static void main(java.lang.String[] args)
```

The main method.

Parameters:

args - the arguments

Package sosInterface.socket

Class Summary

SOSConnectListener

The listener interface for receiving SOSConnect events.

SOSEventListener

Internal class extending a Socket.IO class to wrap the whole connection within an encryption mechanism.

sosInterface.socket

Class SOSConnectListener

```
java.lang.Object
|
+--sosInterface.socket.SOSConnectListener
```

All Implemented Interfaces:
com.corundumstudio.socketio.listener.ConnectListener

public class **SOSConnectListener**
extends java.lang.Object
implements com.corundumstudio.socketio.listener.ConnectListener

The listener interface for receiving SOSConnect events. The class that is interested in processing a SOSConnect event implements this interface, and the object created with that class is registered with a component using the component's`addSOSConnectListener` method. When the SOSConnect event occurs, that object's appropriate method is invoked.

SOSConnectEvent

Constructors

SOSConnectListener

```
public      SOSConnectListener()
```

Methods

onConnect

```
public void
onConnect(com.corundumstudio.socketio.SocketIOClient
client)
```

On connect.

Parameters:

client - the client

---

sosInterface.socket

Class SOSEventListener

```
java.lang.Object
|
+--sosInterface.socket.SOSEventListener
```

All Implemented Interfaces:
com.corundumstudio.socketio.listener.DataListener

---

public abstract class **SOSEventListener**
extends java.lang.Object
implements com.corundumstudio.socketio.listener.DataListener

Internal class extending a Socket.IO class to wrap the whole connection within an encryption mechanism.

SOSEventEvent

Constructors

SOSEventListener

```
public      SOSEventListener()
```

Methods

doOnData

```
public abstract void
doOnData(com.corundumstudio.socketio.SocketIOClient
client,
org.json.JSONObject json,
com.corundumstudio.socketio.AckRequest ackRequest)
```

Implemented by the event-specific data listener so it can work afte decrypting. For more information, read the javadoc for the netty-socket.io DataListener<>() class.

Parameters:

client - the client
json - the json
ackRequest - the ack request

---

doSendEvent

```
public static void
doSendEvent(com.corundumstudio.socketio.SocketIOClient
client,
                                    org.json.JSONObject data)
```

Encrypts the given data and sends it to the client.

Parameters:

client - the client target.
data - the plaintext JSON data.

---

onData

```
public void
onData(com.corundumstudio.socketio.SocketIOClient
client,
java.lang.String cipher,
com.corundumstudio.socketio.AckRequest ackRequest)
```

Decrypts on data.

Parameters:

client - the client
cipher - the cipher
ackRequest - the ack request

Package storage

Class Summary

DataStoreFacade

The Class DataStoreFacade.

DataStoreFacade_Driver

The Class DataStoreFacade_Driver.

storage

Class DataStoreFacade

```
java.lang.Object
|
+--storage.DataStoreFacade
```

public class **DataStoreFacade**
extends java.lang.Object

The Class DataStoreFacade.

Constructors

DataStoreFacade

```
public      DataStoreFacade()
throws java.lang.Exception
```

Attempts to connect to the Database.

Throws:

java.lang.Exception - Throws an exception if the database connection fails.

Methods

addNewRoleToOrganization

```
public void addNewRoleToOrganization(java.lang.String
roleName,
int organizationID,
int userID,
boolean[]
privileges)
throws java.lang.Exception
```

Adds a new role to the organization provided and assigns the role to the user provided.

Parameters:

roleName - The name that is given to the role. organizationID - The organization ID of the organization that the role belongs to.
userID - The user ID of the user that will own the role. privileges - The list of privileges granted to the user for their particular role.

Throws:

java.lang.Exception - An exception is thrown when the new role fails to be stored in the SOS database.

---

cancelEvent

```
public void cancelEvent(int eventID)
throws java.lang.Exception
```

Requests to cancel the event in the database.

Parameters:

eventID - The ID of the event that needs to be cancelled.

Throws:

java.lang.Exception - Throws an exception if the event could not be cancelled.

---

createNewEvent

```
public void createNewEvent(event.Event event)
throws java.lang.Exception
```

Creates a new event in the SOS system.

Parameters:

event - the event

Throws:

java.lang.Exception - Throws an exception if the parameters are not in the expected format and if the organization hosting the event no longer exists.

---

createNewOrganization

```
public void
createNewOrganization(organization.Organization org, int
userID)
throws java.lang.Exception
```

Creates a new organization on the SOS system.

Parameters:

org - the org
userID - The user ID of the user creating the organization.

Throws:

java.lang.Exception - Throws an exception if there is an issue storing the organization into the database.

---

filterEventsByLocation

```
public java.sql.ResultSet filterEventsByLocation(double
lat_coordinate,
                                                 double
long_coordinate)
throws java.lang.Exception
```

Returns a list of JSON objects .

Parameters:

lat_coordinate - The latitude of the location of interest.
long_coordinate - The longitude of the location of interest.

Returns:

The results from the database of the closest events.

Throws:

java.lang.Exception - An exception is thrown if their is a problem retrieving nearby events.

---

getEvents

```
public java.sql.ResultSet getEvents()
throws java.lang.Exception
```

Gets all the events in the database that are not cancelled.

Returns:

A result set with the events found in the database.

Throws:

java.lang.Exception - An exception is thrown when the procedure fails to retrieve the results from the database.

---

getEventsByOrganization

```
public java.sql.ResultSet getEventsByOrganization(int
orgID)
throws java.lang.Exception
```

Returns the Events of the given Organization.

Parameters:

orgID - the id of the organization.

Returns:

the ResultSet containing the event entries.

Throws:

java.lang.Exception - the exception

---

getEventsByUser

```
public java.sql.ResultSet getEventsByUser(int userID)
throws java.lang.Exception
```

Returns the Events attended by the given user.

Parameters:

userID - the user id

Returns:

a ResultSet containing the target events

Throws:

java.lang.Exception - the exception

---

joinOrganization

```
public void joinOrganization(int userID,
int organizationID)
throws java.lang.Exception
```

Allows the user to join an organization.

Parameters:

userID - The ID of the user that wants to join an organization. organizationID - The ID of the organization that the user wants to join.

Throws:

java.lang.Exception - Throws an exception if the user tries to join an organization they already belong to or does not exist.

---

registerNewUser

```
public void registerNewUser(user.User user)
```

Registers a new user for the SOS system.

Parameters:

user - the user

---

retrieveEventDetails

```
public java.sql.ResultSet retrieveEventDetails(int
eventID)
throws java.lang.Exception
```

Retrieves all the details for a certain event.

Parameters:

eventID - The ID of the event that we want the details for.

Returns:

The details of the events in the form of a result set.

Throws:

java.lang.Exception - Throws an exception if the event details were not found.

---

retrieveMembersOfOrganization

```
public java.sql.ResultSet
retrieveMembersOfOrganization(int organization_id)
throws java.lang.Exception
```

Retrives all the users which are members of a given organization.

Parameters:

organization_id - the id of the organization.

Returns:

the ResultSet containing the user entries.

Throws:

java.lang.Exception - the exception

---

retrieveOrganizationDetails

```
public java.sql.ResultSet
retrieveOrganizationDetails(int organizationID)
throws java.lang.Exception
```

Retrieves the information of the organization that is stored in the database.

Parameters:

organizationID - The ID of the organization that the details were requested for.

Returns:

The set of details found in the database.

Throws:

java.lang.Exception - Throws an exception if there is a problem retrieving the details for the specified information.

---

retrieveOrganizationsForUser

```
public java.sql.ResultSet
retrieveOrganizationsForUser(int userID)
throws java.lang.Exception
```

Retrieves all the organizations which the user belongs to,.

Parameters:

userID - The ID of the user that we want all the organizations for.

Returns:

A set of organizations which the user belongs to within the SOS.

Throws:

java.lang.Exception - Throws an exception if their is an error with the connectivity to the storage of the system.

---

retrievePublicOrganizations

```
public java.sql.ResultSet retrievePublicOrganizations()
throws java.lang.Exception
```

Retrieves all of the public organizations stored in the storage.

Returns:

The set of all the public organizations located in the storage.

Throws:

java.lang.Exception - Throws an exception if their was an issue retrieving all of the organizations from the storage.

---

retrieveUserByUsername

```
public java.sql.ResultSet
retrieveUserByUsername(java.lang.String username) throws
java.lang.Exception
```

Retrieves an user entry from its unique username. User for login in moslty.

Parameters:

username - the username of the user

Returns:

the result set containing the user entry.

Throws:

java.lang.Exception - the exception

---

retrieveUserDetails

```
public java.sql.ResultSet retrieveUserDetails(int
userID)
throws java.lang.Exception
```

Retrieves the details from a given user stored in the DB.

Parameters:

userID - The ID for the user that the details are requested for.

Returns:

A set of details found in the storage.

Throws:

java.lang.Exception - Throws an exception if their is an issue connecting to the database.

---

saveUserAttendance

```
public void saveUserAttendance(long userID, long
eventID)
throws java.lang.Exception
```

Saves the attendance of the user to a particular event in the SOS storage.

Parameters:

userID - The ID of the user that is attending the event.
eventID - The ID of the event that the user is attending.

Throws:

java.lang.Exception - Throws an exception if there is an issue in storing the attendance of the user in the database.

---

terminateConnection

```
public void terminateConnection() throws
java.lang.Exception
```

Terminates connection to the database.

Throws:

java.lang.Exception - Throws an exception if database connection cannot be closed.

---

updateUserInformation

```
public void updateUserInformation(user.User user)
            throws java.lang.Exception
```

Updates the information of the user to the given information.

Parameters:

user - the user

Throws:

java.lang.Exception - If the email is already present in the database under a different user then an exception is thrown.

---

verifyUserLogin

```
public boolean verifyUserLogin(java.lang.String email,
java.lang.String
password)
                  throws java.lang.Exception
```

Verifies the login of the user.

Parameters:

email - The email that the user provides when logging in. password - The encrypted password the user provides when logging in.

Returns:

A boolean verifying if the user has correct credentials to log into SOS.

Throws:

java.lang.Exception - Throws an exception if the credentials are in an invalid format.

---

storage

Class DataStoreFacade_Driver

```
java.lang.Object
|
+--storage.DataStoreFacade_Driver
```

---

public class **DataStoreFacade_Driver**
extends java.lang.Object

The Class DataStoreFacade_Driver.

Constructors

DataStoreFacade_Driver

```
public      DataStoreFacade_Driver()
```

Methods

main

```
public static void main(java.lang.String[] args)
```

The main method.

Parameters:

args - the arguments

Package user

Class Summary

NewUserBuilder

A Builder which creates new User objects.

User

A run-time representation of a User persistent object.

UserLoader

A class which creates a User object from a database-format User object (e.g., a SQL Table entry for User).

UserManager

A Singleton class which managers all the User functions.

UserUpdater

A class which deals with User modifications.

---

user

Class NewUserBuilder

```
java.lang.Object
|
+--user.NewUserBuilder
```

---

public class **NewUserBuilder**
extends java.lang.Object

A Builder which creates new User objects. It is used to decouple the parts of the process of creating a new User from the actual User class, which is intended to only be a data wrapper class which can be easily parsed into the database format. Namely, this class implements the checks and validations necessary to create a valid User and will reject invalid ones. As part of this validation, it must interact with the SOS Security System classes that implement the password and access policies.

Constructors

NewUserBuilder

```
public      NewUserBuilder()
```

Creates a new NewUserBuilder to instantiate the new User.

Methods

build

```
public user.User build()
throws
java.lang.IllegalArgumentException
```

Builds the User.

Returns:

an User object, or throws an error.

Throws:

java.lang.IllegalArgumentException - thrown if the organization being built is not complete.

---

isNotComplete

```
public boolean isNotComplete()
```

Checks if is not complete.

Returns:

true if the user is not complete false otherwise.

---

setEmail

```
public user.NewUserBuilder setEmail(java.lang.String
setEmail)
```

Sets the email.

Parameters:

setEmail - the email

Returns:

the NewUserBuilder

---

setName

```
public user.NewUserBuilder setName(java.lang.String
name)
```

Sets the name.

Parameters:

name - the name

Returns:

the NewUserBuilder

---

setPassword

```
public user.NewUserBuilder setPassword(java.lang.String
password)
```

Sets the password.

Parameters:

password - the password

Returns:

the NewUserBuilder

---

setPrivacy

```
public user.NewUserBuilder setPrivacy(java.lang.String
privacy)
```

Sets the privacy.

Parameters:

privacy - the privacy

Returns:

the NewUserBuilder

---

setUsername

```
public user.NewUserBuilder setUsername(java.lang.String
username)
```

Sets the name.

Parameters:

username - the username

Returns:

the NewUserBuilder

---

user

Class User

```
java.lang.Object
|
+--user.User
```

---

public class **User**
extends java.lang.Object

A run-time representation of a User persistent object. This class is used as an intermediary for creation, retrieval, and modification of User data within the Java code (and the JVM). It is encodable (or serializable) to a database format (e.g., SQL Entry).

Fields

email

`protected java.lang.String` **`email`**
The email.

---

name

`protected java.lang.String` **`name`**
The name.

---

password

`protected java.lang.String` **`password`**
The password.

---

privacy

`protected java.lang.String` **`privacy`**
The privacy.

---

userName

`protected java.lang.String` **`userName`**
The user name.

---

user_id

`protected int` **`user_id`**
The user id.

Constructors

User

`protected` **`User`**`()`

Creates an empty user object, for the Builder.

User

```
protected      User(java.sql.ResultSet set)
throws java.lang.Exception
```

Constructs a new User class. Called through the UserBuilder class.
Attribute assignations are done through protected scope.

Parameters:

set - the set

Throws:

java.lang.Exception - the exception

Methods

getEmail

```
public java.lang.String getEmail()
```

Gets the email.

Returns:

the email

---

getJSON

```
public org.json.JSONObject getJSON()
```

Returns the JSON form of this User.

Returns:

the json

---

getName

```
public java.lang.String getName()
```

Gets the name.

Returns:

the name

getPassword

`public java.lang.String **getPassword**()`

Gets the password.

Returns:

the password

getPrivacy

`public java.lang.String **getPrivacy**()`

Gets the privacy.

Returns:

the privacy

getUserName

`public java.lang.String **getUserName**()`

Gets the user name.

Returns:

the userName

getUser_id

`public int **getUser_id**()`

Gets the user id.

Returns:

the user_id

user

Class UserLoader

```
java.lang.Object
|
+--user.UserLoader
```

---

public class **UserLoader**
extends java.lang.Object

A class which creates a User object from a database-format User object (e.g., a SQL Table entry for User). This class decouples the parsing from the database to the system logic from the UserManager class and can be extended to include internal checks for data integrity purposes.

Constructors

UserLoader

```
public      UserLoader()
```

Instantiates a new user loader.

Methods

LoadUser

```
public user.User LoadUser(java.sql.ResultSet results)
throws java.lang.Exception
```

Creates a User from a database-format entry.

Parameters:

results - The set of details found in the storage of the SOS.

Returns:

a User object with the given attributes.

Throws:

java.lang.Exception - the exception

---

user

Class UserManager

```
java.lang.Object
|
+--user.UserManager
```

---

public class **UserManager**
extends java.lang.Object

A Singleton class which managers all the User functions. This class receives dispatched actions from the SOS Dispatcher and completes that action using objects internal to its subsystem. It also is in charge of interacting with the SOS Data Store Façade directly. Part of the role of this class is to parse front-end format user data (e.g., JSON-String defining a new User) and calling the appropriate functions on the other classes according to that data. It also is in charge of encoding a User object into database format objects (e.g., SQL Table entry for User).

Constructors

UserManager

`protected` **`UserManager`**`()`

A protected or private constructor ensures that no other class has access to the Singleton.

Methods

ChangeUserDetails

```
public org.json.JSONObject
ChangeUserDetails(org.json.JSONObject json)
throws java.lang.Exception
```

Changes the details of the user in the SOS system.

Parameters:

userID - The ID of the user that wants to change their information.
json2 - The JSON string with the user information and their changes.

Throws:

java.lang.Exception - Throws an exception if an error occurs while attempting to change user information.

CreateNewProfile

```
public org.json.JSONObject
CreateNewProfile(org.json.JSONObject json)
throws java.lang.Exception
```

Creates a new profile when the user registers to the SOS site.

Parameters:

input - A JSON string representing the user's information.

Throws:

java.lang.Exception - Throws an exception if there was an issue creating the user's profile.

LoadUser

```
public org.json.JSONObject LoadUser(org.json.JSONObject
payload)
```

Creates a User from a database-format entry. Done by calling the UserLoader class.

Parameters:

payload - The ID of the user that we want

Returns:

a User object with the given attributes.

getMembersOfOrganization

```
public org.json.JSONObject
getMembersOfOrganization(org.json.JSONObject payload)
```

Returns all the members of a given organization.

instance

```
public static user.UserManager instance()
```

Gives the instance of the UserManager, or creates one if none exists.

Returns:

the unique instance of this class.

login

```
public org.json.JSONObject login(org.json.JSONObject
payload)
```

Checks if the parameters given by a log-in attempt are valid, and returns the user information if so.

main

```
public static void main(java.lang.String[] args)
```

user

Class UserUpdater

```
java.lang.Object
|
+--user.UserUpdater
```

public class **UserUpdater**

extends java.lang.Object

A class which deals with User modifications. User modifications are done on the system logic-level User object first and are only finalized once they are stored to the database. The UserUpdater decouples these modifications from the UserManager class and from the User class itself and implements checks and validations in the same way that NewUserBuilder does. It also ensures that every modification to the User class is saved to the SOS Data Store.

Constructors

UserUpdater

```
public      UserUpdater()
```

Methods

ChangeUser

```
public void ChangeUser(user.User user,
java.util.Map update)
```

Updates a User object with the given changes.

Parameters:

user - the User that will be updated.
update - the update

---

makeUpdatesMap

```
public java.util.Map makeUpdatesMap(org.json.JSONObject
json)
```

Creates an update-map from a json payload.

Parameters:

json - the json

Returns:

the map

Package utils

Class Summary

Constants

The Class Constants.

JSONTranslator

The Class JSONTranslator.

utils

Class Constants

```
java.lang.Object
|
+--utils.Constants
```

public class **Constants**
extends java.lang.Object

The Class Constants.

Fields

DB_HOSTNAME

`public static final java.lang.String` **DB_HOSTNAME**
The Constant DB_HOSTNAME.

DB_PORT

`public static final int` **DB_PORT**
The Constant DB_PORT.

SERVER_HOSTNAME

`public static final java.lang.String` **SERVER_HOSTNAME**
The Constant SERVER_HOSTNAME.

SERVER_PORT

`public static final int` **`SERVER_PORT`**
The Constant SERVER_PORT.

Constructors

Constants

`public` **`Constants`**`()`

utils

Class JSONTranslator

```
java.lang.Object
|
+--utils.JSONTranslator
```

public class **JSONTranslator**
extends java.lang.Object

The Class JSONTranslator.

Constructors

JSONTranslator

`public` **`JSONTranslator`**`()`

Methods

resultSetToJSONArray

```
public static org.json.JSONArray
resultSetToJSONArray(java.sql.ResultSet set)
throws
java.lang.Exception
```

Result set to JSON array.

Parameters:

set - the set

Returns:

the JSON array

Throws:

java.lang.Exception - the exception

---

resultSetToJSONObject

```
public static org.json.JSONObject
resultSetToJSONObject(java.sql.ResultSet set)
throws
java.lang.Exception
```

Result set to JSON object.

Parameters:

set - the set

Returns:

the JSON object

Throws:

java.lang.Exception - the exception

11.6 Appendix F – Documented Code for Test Driver

11.6.1 Unit Test Code

The following sections contain the Unit Test code, as well as Test Code coverage, for the manager classes. In each one, the code coverage is represented by the highlights, where green means reached and red means skipped.

11.6.1.1 TransferManagerTest

```
/**
 * A Singleton that handles secure data exchange between the front end and the back
end.
 */
public class TransferManager {

	/** The cipher. */
	private Cipher cipher;

	/** The keystore. */
	private KeyStore keystore;

	/**
	 * Creates the unique instance of TransferManager.
	 */
	private TransferManager() {
		try {
			this.cipher = Cipher.getInstance("RSA");
			InputStream ins = new FileInputStream(new
File("resources/keystore.jks"));
			this.keystore = KeyStore.getInstance("JCEKS");
			this.keystore.load(ins, "s3cr3t".toCharArray());
		} catch (NoSuchAlgorithmException | NoSuchPaddingException e) {
			e.printStackTrace();
		} catch (KeyStoreException e) {
			e.printStackTrace();
		} catch (CertificateException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	/**
	 * A handle to the unique PasswordManager
	* instance.
	*/
	static private TransferManager _instance = null;

	/**
	 * Instance.
	 *
	 * @return The unique instance of this
	 * class.
	 */
		static public TransferManager instance( ) {
```

```
            if ( null == _instance) {
                  _instance = new TransferManager( );
            }
            return _instance;
      }

      /**
       * Produces an Base64 version of the encrypted ciphertext for the
       * input given in msg.
       *
       * @param msg             the input to be encrypted and encoded.
       * @param alias the alias
       * @return          the JSONObject containing:
       *                  'key'  the encrypted key parameter, decryptable with the
target's private key.
       *                  'iv'   the encrypted iv parameter, decryptable with the
target's private key.
       *                  'text' the encrypted text, decrypateble with a
AES/CBC/PKCS7Padding using the given key and iv.
       */
      public JSONObject encryptMessage(String msg, String alias) {
            JSONObject retJSON = null;
            try {

                  byte[] iv  = new byte[16];
                  byte[] key = new byte[16];

                  SecureRandom r = new SecureRandom();
                  r.nextBytes(iv);
                  r.nextBytes(key);
                  IvParameterSpec ivPS = new IvParameterSpec(iv);

                  // Encrypt Key using Pu/Pr
                  this.cipher.init(Cipher.ENCRYPT_MODE,
this.keystore.getCertificate(alias));
                  byte[] keycipher = this.cipher.doFinal(key);
                  byte[] ivcipher = this.cipher.doFinal(iv);

                  SecretKeySpec secretKeySpec = new SecretKeySpec(key, "AES");

                  Cipher sCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
                  sCipher.init(Cipher.ENCRYPT_MODE, secretKeySpec, ivPS);

                  byte[] plaintext = msg.getBytes("UTF-8");
                  byte[] ciphertext = sCipher.doFinal(plaintext);

                  retJSON = new JSONObject();
                  retJSON.put("key",
Base64.getEncoder().encodeToString(keycipher));
                  retJSON.put("iv", Base64.getEncoder().encodeToString(ivcipher));
                  retJSON.put("text",
Base64.getEncoder().encodeToString(ciphertext));

            } catch (IllegalBlockSizeException | BadPaddingException e) {
```

```
                e.printStackTrace();
            } catch (InvalidKeyException e) {
                e.printStackTrace();
            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            } catch (KeyStoreException e) {
                e.printStackTrace();
            } catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
            } catch (NoSuchPaddingException e) {
                e.printStackTrace();
            } catch (InvalidAlgorithmParameterException e) {
                e.printStackTrace();
            } catch (JSONException e) {
                e.printStackTrace();
            }
            return retJSON;
      }

      /**
       * Decrypts the message in the given JSONObject.
       * @param msg
       *            a json object which must have:
       *                  key          a symmetric key encrypted with this
TransferManager's public certificate.
       *                  iv           an iv value encrypted with this
TransferManager's public certificate.
       *                  text   a cyphertext encrypted using AES/CBC/PKCS5Padding
and the given key and iv.
       * @return
       *            the plaintext form of text.
       */
      public String decryptMessage(JSONObject msg) {
            String ret = null;
            try {

                this.cipher.init(Cipher.DECRYPT_MODE, this.getPrivateKey());
                byte[] iv  =
this.cipher.doFinal(Base64.getDecoder().decode(msg.getString("iv")));
                byte[] key =
this.cipher.doFinal(Base64.getDecoder().decode(msg.getString("key")));

                Cipher sCipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
                SecretKeySpec sks = new SecretKeySpec(key, "AES");
             IvParameterSpec ivp = new IvParameterSpec(iv);
                sCipher.init(Cipher.DECRYPT_MODE, sks, ivp);

                byte[] ciphertext =
Base64.getDecoder().decode(msg.getString("text"));
                byte[] plaintext = sCipher.doFinal(ciphertext);
                ret = new String(plaintext, "UTF-8");

            } catch (InvalidKeyException e) {
                e.printStackTrace();
```

```
            } catch (IllegalBlockSizeException e) {
                  e.printStackTrace();
            } catch (BadPaddingException e) {
                  e.printStackTrace();
            } catch (JSONException e) {
                  e.printStackTrace();
            } catch (NoSuchAlgorithmException e) {
                  e.printStackTrace();
            } catch (NoSuchPaddingException e) {
                  e.printStackTrace();
            } catch (InvalidAlgorithmParameterException e) {
                  e.printStackTrace();
            } catch (UnsupportedEncodingException e) {
                  e.printStackTrace();
            }
            return ret;
      }

      /**
       * Gets the private key of this object.
       * @return
       *            the private key.
       */
      private PrivateKey getPrivateKey() {
            PrivateKey ret = null;
            try {
                KeyStore.PasswordProtection keyPassword = new
KeyStore.PasswordProtection("s3cr3t".toCharArray());
                KeyStore.PrivateKeyEntry privateKeyEntry =
(KeyStore.PrivateKeyEntry) this.keystore.getEntry("mykey", keyPassword);
                  ret = privateKeyEntry.getPrivateKey();
            } catch (NoSuchAlgorithmException | UnrecoverableEntryException |
KeyStoreException e) {
                  e.printStackTrace();
            }
          return ret;
      }

      /**
       * Gets the public key of this object.
       * @return
       *            the public key.
       */
      private PublicKey getPublicKey() {
            PublicKey ret = null;
            try {
                  Certificate cert = this.keystore.getCertificate("mykey");
                  ret = cert.getPublicKey();
            } catch (KeyStoreException e) {
                  e.printStackTrace();
            }
            return ret;
      }
```

```
	/**
	 * Gets the sharable (PEM) string version of this object's certificate.
	 * @return
	 *            a String containing the PEM version of the certificate.
	 */
	public String getSharableCertificate() {
		String ret = null;
		try {
			X509Certificate cert = (X509Certificate)
this.keystore.getCertificate("mykey");

			ret  = "-----BEGIN CERTIFICATE-----";
			ret += new String(Base64.getEncoder().encode(cert.getEncoded()));
			ret += "-----END CERTIFICATE-----";

		} catch (KeyStoreException e) {
			e.printStackTrace();
		} catch (CertificateEncodingException e) {
			e.printStackTrace();
		}
		return ret;
	}

	/**
	 * Adds an external certificate to the KeyStore with the given alias.
	 * @param certS
	 *            the certificate, usually in a PEM format.
	 * @param alias
	 *            the alias for the certificate.
	 */
	public void setCertificateEntry(String certS, String alias) {
		try {
			CertificateFactory fact =
CertificateFactory.getInstance("X.509");
			X509Certificate cert = (X509Certificate)
fact.generateCertificate(new ByteArrayInputStream(certS.getBytes()));
			this.keystore.setCertificateEntry(alias, cert);
		} catch (CertificateException e) {
			e.printStackTrace();
		} catch (KeyStoreException e) {
			e.printStackTrace();
		}
	}

}
```

11.6.1.2 OrganizationManagerTest

```
/**
 * The Class OrganizationManagerTest.
 */
class OrganizationManagerTest {

	/** The om. */
	OrganizationManager om;
```

```
	/**
	 * Sets the up.
	 *
	 * @throws Exception the exception
	 */
	@BeforeEach
	void setUp() throws Exception {
		om = OrganizationManager.instance();
	}

	/**
	 * Tear down.
	 *
	 * @throws Exception the exception
	 */
	@AfterEach
	void tearDown() throws Exception {
		om = null;
	}

	/**
	 * Test instance.
	 */
	@Test
	void testInstance() {
		// Must return the same object if called twice.
		OrganizationManager s1 = OrganizationManager.instance();
		OrganizationManager s2 = OrganizationManager.instance();
		assertEquals("same instance", s1, s2); }

	/**
	 * Test load organization details.
	 */
	@Test
	void testLoadOrganizationDetails() {

		try {
			JSONObject ret = null;
			// Invalid JSONObject, missing parameter
			ret = om.loadOrganizationDetails(new
JSONObject("{something:else}"));
			assertEquals("error message", true, ret.has("error"));
			assertEquals("error message", "true", ret.get("error"));
			assertEquals("error message", true, ret.has("type"));
			assertEquals("error message", "orgidValueError",
ret.get("type"));

			// No orgid, not found
			ret = om.loadOrganizationDetails(new
JSONObject("{organization:{}}"));
			assertEquals("error message", true, ret.has("error"));
			assertEquals("error message", "true", ret.get("error"));
			assertEquals("error message", true, ret.has("type"));
```

```
                    assertEquals("error message", "noSearchParameter",
ret.get("type"));

                    // Valid org id, found.
                    ret = om.loadOrganizationDetails(new
JSONObject("{organization:{organization_id:1}}"));
                    assertEquals("error message", true, ret.has("user"));


                    // 1   Fishing John's      A fishing club.     PUBLIC Must own
fishing rod
                    JSONObject org = ret.getJSONObject("data");
                    System.out.println(org.toString());
                    assertEquals("error message", true, org.has("description"));
                    assertEquals("error message", "A fishing club.",
org.get("description"));
                    assertEquals("error message", true, org.has("organization_id"));
                    assertEquals("error message", 1, org.getInt("organization_id"));
                    assertEquals("error message", true, org.has("name"));
                    assertEquals("error message", "Fishing John's", org.get("name"));
                    assertEquals("error message", true, org.has("privacy"));
                    assertEquals("error message", "PUBLIC", org.get("privacy"));
                    assertEquals("error message", true, org.has("requirement"));
                    assertEquals("error message", "Must own fishing rod",
org.get("requirement"));

             } catch (Exception e) {
                    e.printStackTrace();
             }

      }
}
```

11.6.1.3 EventManagerTest

```
/**
 * The Class EventManagerTest.
 */
class EventManagerTest {

      /** The em. */
      EventManager em;

      /**
       * Sets the up.
       *
       * @throws Exception the exception
       */
      @BeforeEach
      void setUp() throws Exception {
            em = EventManager.instance();
      }

      /**
       * Tear down.
```

```
         *
         * @throws Exception the exception
         */
        @AfterEach
        void tearDown() throws Exception {
                em = null;
        }

        /**
         * Test instance.
         */
        @Test
        void testInstance() {
                // Must return the same object if called twice.
                EventManager um1 = EventManager.instance();
                EventManager um2 = EventManager.instance();
                assertEquals("same instance", um1, um2);
        }

        /**
         * Test load user.
         */
        @Test
        void testLoadUser() {
                try {

                        JSONObject ret = null;
                        // Invalid JSONObject, missing parameter
                        ret = em.loadEventDetails(new JSONObject("{something:else}"));
                        assertEquals("error message", true, ret.has("error"));
                        assertEquals("error message", "true", ret.get("error"));
                        assertEquals("error message", true, ret.has("type"));
                        assertEquals("error message", "payloadError", ret.get("type"));

                        // Valid username, found.
                        ret = em.loadEventDetails(new JSONObject("{event_id:6}"));
                        assertEquals("error message", true, ret.has("data"));

                        // event_id, description, date, is_cancelled, visibility, time,
name, event_type, hosted_by, lat_coordinate, long_coordinate
                        // 6   Come to fish and relax    2019-11-30   1      1
        12:00:01      Fishing      1      1      10.34  21.31
                        JSONObject loadedUser = ret.getJSONObject("data");
                        assertEquals("error message", true, loadedUser.has("event_id"));
                        assertEquals("error message", 6, loadedUser.getInt("event_id"));

                        assertEquals("error message", true,
loadedUser.has("description"));
                        assertEquals("error message", "Come to fish and relax",
loadedUser.get("description"));
                        assertEquals("error message", true, loadedUser.has("date"));
                        assertEquals("error message", "2019-11-30",
loadedUser.getString("date"));
```

```
                    assertEquals("error message", true,
loadedUser.has("is_cancelled"));
                    assertEquals("error message", true,
loadedUser.get("is_cancelled"));
                    assertEquals("error message", true,
loadedUser.has("visibility"));
                    assertEquals("error message", 1,
loadedUser.getInt("visibility"));
                    assertEquals("error message", true, loadedUser.has("time"));
                    assertEquals("error message", "12:00:01",
loadedUser.get("time"));
                    assertEquals("error message", true, loadedUser.has("name"));
                    assertEquals("error message", "Fishing", loadedUser.get("name"));
                    assertEquals("error message", true,
loadedUser.has("event_type"));
                    assertEquals("error message", 1,
loadedUser.getInt("event_type"));
                    assertEquals("error message", true, loadedUser.has("hosted_by"));
                    assertEquals("error message", 1, loadedUser.getInt("hosted_by"));
                    assertEquals("error message", true,
loadedUser.has("lat_coordinate"));
                    assertEquals("error message", 10.34,
loadedUser.getDouble("lat_coordinate"));
                    assertEquals("error message", true,
loadedUser.has("long_coordinate"));
                    assertEquals("error message", 21.31,
loadedUser.getDouble("long_coordinate"));

             } catch (Exception e) {
                    e.printStackTrace();
             }
      }
```

11.6.1.4 UserManagerTest

```
/**
 * The Class UserManagerTest.
 */
class UserManagerTest {

      /** The um. */
      UserManager um;

      /**
       * Sets the up.
       *
       * @throws Exception the exception
       */
      @BeforeEach
      void setUp() throws Exception {
            um = UserManager.instance();
      }

      /**
       * Tear down.
```

```
         *
         * @throws Exception the exception
         */
        @AfterEach
        void tearDown() throws Exception {
                um = null;
        }

        /**
         * Test user manager.
         */
        @Test
        void testUserManager() {
                // Must have an non-null DataStoreFacade, UserLoader, and UserUpdater.
                UserManager um1 = new UserManager();
                assertNotNull(um1.ds);
                assertNotNull(um1.ul);
                assertNotNull(um1.up);
        }

        /**
         * Test instance.
         */
        @Test
        void testInstance() {
                // Must return the same object if called twice.
                UserManager um1 = UserManager.instance();
                UserManager um2 = UserManager.instance();
                assertEquals("same instance", um1, um2);
        }

        /**
         * Test load user.
         */
        @Test
        void testLoadUser() {
                try {

                        JSONObject ret = null;
                        // Invalid JSONObject, missing parameter
                        ret = um.LoadUser(new JSONObject("{something:else}"));
                        assertEquals("error message", true, ret.has("error"));
                        assertEquals("error message", "true", ret.get("error"));
                        assertEquals("error message", true, ret.has("type"));
                        assertEquals("error message", "usernameValueError",
ret.get("type"));

                        // Invalid Username, not found
                        ret = um.LoadUser(new JSONObject("{user:{username:nothere}}"));
                        assertEquals("error message", true, ret.has("error"));
                        assertEquals("error message", "true", ret.get("error"));
                        assertEquals("error message", true, ret.has("type"));
                        assertEquals("error message", "usernotfoundError",
ret.get("type"));
```

```
                    // Valid username, found.
                    ret = um.LoadUser(new JSONObject("{user:{username:jdoe001}}"));
                    assertEquals("error message", true, ret.has("user"));

                    JSONObject loadedUser = ret.getJSONObject("user");
                    System.out.println(loadedUser.toString());
                    assertEquals("error message", true, loadedUser.has("password"));
                    assertEquals("error message", "password123",
loadedUser.get("password"));
                    assertEquals("error message", true, loadedUser.has("user_id"));
                    assertEquals("error message", 2, loadedUser.getInt("user_id"));
                    assertEquals("error message", true, loadedUser.has("user_name"));
                    assertEquals("error message", "jdoe001",
loadedUser.get("user_name"));
                    assertEquals("error message", true, loadedUser.has("name"));
                    assertEquals("error message", "John", loadedUser.get("name"));
                    assertEquals("error message", true, loadedUser.has("privacy"));
                    assertEquals("error message", "PUBLIC",
loadedUser.get("privacy"));
                    assertEquals("error message", true, loadedUser.has("email"));
                    assertEquals("error message", "test@gmail.com",
loadedUser.get("email"));

             } catch (Exception e) {
                    e.printStackTrace();
             }
       }

       /**
        * Test login.
        */
       @Test
       void testLogin() {
             try {

                    JSONObject ret = null;
                    // Invalid JSONObject, missing parameter
                    ret = um.login(new JSONObject("{something:else}"));
                    assertEquals("error message", true, ret.has("error"));
                    assertEquals("error message", "true", ret.get("error"));
                    assertEquals("error message", true, ret.has("type"));
                    assertEquals("error message", "loginValueError",
ret.get("type"));

                    // Invalid Username, not found
                    ret = um.login(new
JSONObject("{user:{username:nothere,password:notthere}}"));
                    System.out.println(ret.toString());
                    assertEquals("error message", true, ret.has("error"));
                    assertEquals("error message", "true", ret.get("error"));
                    assertEquals("error message", true, ret.has("type"));
                    assertEquals("error message", "usernotfoundError",
ret.get("type"));
```

```
                    // Valid username, wrong password.
                    ret = um.login(new
JSONObject("{user:{username:jdoe001,password:notthere}}"));
                    assertEquals("error message", true, ret.has("error"));
                    assertEquals("error message", "true", ret.get("error"));
                    assertEquals("error message", true, ret.has("type"));
                    assertEquals("error message", "invalidcredentials",
ret.get("type"));

                    // Valid username, Valid password.
                    ret = um.login(new
JSONObject("{user:{username:jdoe001,password:password123}}"));
                    assertEquals("correct message", true, ret.has("type"));
                    assertEquals("correct message", "doLogin", ret.get("type"));
                    assertEquals("correct message", true, ret.has("user"));

             } catch (Exception e) {
                    e.printStackTrace();
             }
      }

      /**
       * Test change user details.
       */
      @Test
      void testChangeUserDetails() {
            try {
                    JSONObject ret = null;
                    // Invalid JSONObject, missing parameter
                    ret = um.ChangeUserDetails(new JSONObject("{something:else}"));
                    assertEquals("error message", true, ret.has("error"));
                    assertEquals("error message", "true", ret.get("error"));
                    assertEquals("error message", true, ret.has("type"));
                    assertEquals("error message", "userupdateValueError",
ret.get("type"));

                    // Invalid user id, not found
                    ret = um.ChangeUserDetails(new
JSONObject("{user_id:1,password:notthere}"));
                    assertEquals("error message", true, ret.has("error"));
                    assertEquals("error message", "true", ret.get("error"));
                    assertEquals("error message", true, ret.has("type"));
                    assertEquals("error message", "usernotfoundError",
ret.get("type"));

                    // Valid user id, wrong password.
                    ret = um.ChangeUserDetails(new
JSONObject("{user_id:2,password:notthere}"));
                    assertEquals("error message", null, ret);

                    // Valid username, Valid password, Update
                    ret = um.ChangeUserDetails(new
JSONObject("{user_id:2,password:notthere,update:{}}"));
```

```
                    assertEquals("correct message", "{}", ret.toString());

              } catch (Exception e) {
                    e.printStackTrace();
              }
        }

        /**
         * Test get members of organization.
         */
        @Test
        void testGetMembersOfOrganization() {
              try {
                    JSONObject ret = null;
                    // Invalid JSONObject, missing parameter
                    ret = um.getMembersOfOrganization(new
JSONObject("{something:else}"));
                    assertEquals("error message", true, ret.has("error"));
                    assertEquals("error message", "true", ret.get("error"));
                    assertEquals("error message", true, ret.has("type"));
                    assertEquals("error message", "orgidValueError",
ret.get("type"));

                    // Invalid org id, not found
                    ret = um.getMembersOfOrganization(new
JSONObject("{organization:{}}"));
                    assertEquals("error message", true, ret.has("error"));
                    assertEquals("error message", "true", ret.get("error"));
                    assertEquals("error message", true, ret.has("type"));
                    assertEquals("error message", "noSearchParameter",
ret.get("type"));

                    // Invalid org id, returns empty.
                    ret = um.getMembersOfOrganization(new
JSONObject("{organization:{organization_id:20, startIndex:0}}"));
                    System.out.print(ret);
                    assertEquals("error message", true, ret.has("data"));
                    assertEquals("error message", 0,
ret.getJSONArray("data").length());

                    // Valid org id, contains:
                    // 2   password123   test@gmail.com      John   jdoe001      PUBLIC
                    ret = um.getMembersOfOrganization(new
JSONObject("{organization:{organization_id:21, startIndex:0}}"));
                    assertEquals("correct message", true, ret.has("data"));
                    assertEquals("correct message", 1,
ret.getJSONArray("data").length());
                    assertEquals("correct message", 2,
ret.getJSONArray("data").getJSONObject(0).get("user_id"));

              } catch (Exception e) {
                    e.printStackTrace();
              }

}
```

11.6.1.5 SOSServerTest

```
/**
 * The Class SOSServerTest.
 */
class SOSServerTest {

	/**
	 * Test instance.
	 */
	@Test
	void testInstance() {
		// Must return the same object if called twice.
		try {
			SOSServer s1 = SOSServer.instance();
			SOSServer s2 = SOSServer.instance();
			assertEquals("same instance", s1, s2);;
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

}
```

11.7 Appendix G – Diary of Meetings

11.7.1 September 9, 2019

When and Where	Role
Date: 9/9	**Primary Facilitator:** Armando
Start: 1:30 pm	**Timekeeper:** Kian
End: 3:00 pm	**Minute Taker:** Anthony
Room: GL 625A	**Attending:** Armando, Kian, Teriq (late), Anthony, Yovani

1. Status

Initial meetings. Team members are introduced to each other and the initial tasks where distributed. An initial analysis of the system and its components was done.

2. Discussion

Use case requirements are a priority as everything else depends on them. Armando is assigned to formatting all the use cases into a single format (assigned the Document Editor role). As a task, Yovani and Kian have to finish their Use Cases as soon as possible.

Group split into front-end and back-end teams. Front-end team consists of Armando and Kian. Back-end team consists of Anthony, Teriq, and Yovani. Front-end team decided on using the React environment for the website. Research is to be done in several React components such as Redux, Saga, Router, etc. Front-end team needs to research the java software libraries that will be used for the backend. Some starting points are Apache, Java SQL and Restful API

The group discussed the potential entities of the system (in relation to the use cases). The following where agreed on: (a) Actors include Guests, Member, Admin, and Organizer; (b) Other system entities are Events, Clubs/Organizations.

We explored the possibility of using Geolocation and Calendars to present events. The website layout was discussed.

3. Wrap Up

- Yovani and Kian are behind on their use cases.
- Group divided into front-end and back-end teams.
- Layout and logic of the system started to be defined.

11.7.2 September 16, 2019

When and Where	Role
Date: 9/16	**Primary Facilitator:** Armando
Start: 1:30 pm	**Timekeeper:** Kian
End: 2:30 pm	**Minute Taker:** Teriq
Room: PG6 116	**Attending:** Armando, Kian, Teriq, Anthony, Yovani

1. Status

Update regarding use cases. Some members have to redo their use case in order to comply to the content and style requirements. Starting discussion about UML diagrams.

2. Discussion

To follow the requirements for the use cases, some members have to update their files. Anthony and Armando's use cases are complete and in the correct format. Teriq, Yovani, and Kian must update theirs. Information regarding what's missing was left on the project's GitHub's readme. Some use cases have to be fully redone as they overlap with outer members' use case.

In total, 18 use cases are completed. Of this group, the following set is planned for implementation: create roles, create organization, create event, two-factor authentication, cancel event, attending an event, ranking, earn points, and access events.

Some sequence diagrams are demoed by Anthony. Currently, 3 sequence diagrams (out of 10) have been completed (create role, create organization, create event). Currently, both Anthony and Armando will work on diagrams as the other team members focus on updating their use cases.

Discussion about the prototype/mock-up. The front-end team is using react and discussed about modules that would help implement geolocation, namely Google Map React.

3. Wrap Up

- Teriq, Yovani, and Kian must redo or retouch some of their use cases
- Implementation use cases have been partially decided upon.
- Some sequence diagrams where implemented.

11.7.3 September 23, 2019

When and Where	Role
Date: 9/23	**Primary Facilitator:** Armando
Start: 2:00 pm	**Timekeeper:** Kian
End: 4:00 pm	**Minute Taker:** Teriq
Room: PG6 116	**Attending:** Armando, Kian, Teriq, Anthony, Yovani

1. Status

Update regarding section 4.1 being completed. Initial prototype is also built. Also, some sequence diagrams where completed, and each member is assigned a sequence diagram to start during the meeting.

2. Discussion

Section 4.1, Use Cases, of the Requirements Elicitation is completed, which means that all use cases are set on a uniform format. With the complete set, a final implementation set can be decided on (in parenthesis who is tasked with creating the sequence diagram):

- SOS01 – create event
- SOS02 – grant organization roles*(Armando)
- SOS04 – attending event
- SOS10 – access events by location*(Teriq)
- SOS14 – create roles
- SOS16 – create organization
- SOS17 – cancel event
- SOS18 – create a task*(Yovanni)
- SOS12 – set up 2 Factor authorization
- SOS07 – set private accounts* (Kian)

The back-end group discussed how to implement the web service. Socket.io, which has implementations for java and react, so intercommunication should be simplified. For database, the group decided on using a java implementation of SQL.

Tasks were assigned for the following week for each member: Armando will work on chapter 3 of the SRD, project plan. He is also tasked with editing the document and making sure style, format, and language is uniform. Teriq will work on chapter 2 of the SRD, current systems. Kian will work on chapter 1 of the SRD, introduction. Yovani and Anthony will work on chapter 5, including object diagrams and sequence diagrams.

3. Wrap Up

- The implementation use cases where decided upon.
- The back-end group decided on Socket.io for the core web-service functionality.
- Tasks were assigned to each member for the following week.

11.7.4 October 7, 2019

When and Where	Role
Date: 10/7/19	**Primary Facilitator:** Teriq
Start: 2:30 pm	**Timekeeper:** Yovanni
End: 3:30 pm	**Minute Taker:** Anthony
Room: GL 693	**Attending:** Armando, Kian (late), Teriq, Anthony, Yovanni (late)

4. Status

First meeting for deliverable 2. The focus of the meeting was generating tasks from the second deliverable document and assigning the tasks that are immediately available.

5. Discussion

Task Decomposition:

- Read the second deliverable. **All**
- Cover Page – Refine from SRD
- Abstract – Something to do at the end.
- Table Of Contents – Something to do at the end.
- Introduction – Refine from SRD
- Section 1.1 – Refine from SRD
- Section 1.2 Requirements
- Functional Requirements - **Kian**
- Non-Functional Requirements - **Kian**
- Section 1.3 Minimal edits to the SRD.
- Section 1.4 Something to do at the end.
- Section 1.5 Something to do at the end.
- Section 2
- 2.1 Overview – Identify the different subsystems. Depends on 2.2
- 2.2 *Subsystem decomposition is the 1st task.* **All**
- 2.3 Hardware and Software Mapping
- 2.4 Persistent Data Management. **Teriq & Anthony**
- 2.5 Security Management. **Armando**
- Section 3
- Introduction
- 3.1 Class diagrams for subsystems that will be implemented (no details). Depends on 2.2.
- 3.2 Depends on 2.2
- 3.4.1 Cannot do yet.
- 3.4.2 Cannot do yet. Object Constraint Language (OCL)
- 4 Glossary – Something to do at the end
- Appendices A – Refine from SRD
- Appendices B – Refine from SRD

- Appendices C – Create new class diagram for all the subsystems that will be implemented.
- Appendices D – Javadoc on coding
- Appendices E – In progress.

6. Wrap Up

- Teriq, Anthony and Yovanni will continue to conduct research on the back-end development.
- Started doing subsystem decomposition.

11.7.5 October 14, 2019

When and Where	Role
Date: 10/14/2019	**Primary Facilitator:** Armando
Start: 2:00 pm	**Timekeeper:** Yovanni
End: **3**:00 pm	**Minute Taker:** Anthony
Room: GL 595A	**Attending:** Armando, Kian, Anthony, Yovani

4. Status

Update regarding status of project contributions. The focus of the meeting was to discuss the different architectural patterns and identify which of the two the team would use in our product. Tasks needed to be assigned.

5. Discussion

The discussions started with Armando which said that he had no updates and that he would work on the security system as soon as he got the chance to. Anthony then talked about Teriq and his development of the ER diagram. He queried about the structure of the data base and spurred lively debate on certain attributes for persistent data. Anthony also talked about the retrieval of address locations from the Google Location API. Kian status was that he had worked on the front-end and that he had investigated an API called Springboot that applied encryption on the front-end of the SOS website. He also talked about the google maps module, the container component and the API key. Lastly, Yovanni claimed that he had started to work on the tasks that had been assigned to him in the previous meeting and that he was almost finished with them.

After the discussion and status of the work of all the team-members we discussed the different architectures within the team, and we decided that the two architectures ideal for our system would be 3-tier architecture and repository architecture. The 3-tier architecture would serve as our primary architecture and the repository architecture would serve as the secondary architecture use to store and access information from our database. After deciding our architectural patterns, the meeting was disbanded.

6. Wrap Up
   - The architectural patterns for our system were decided.
   - Armando will work on the security section.
   - Anthony will keep working on implementing the database and refining ER diagram.
   - Kian will finish the functional requirements in section 1.
   - Yovanni will work on section 1, specifically the nonfunctional requirements.

11.7.6 October 21, 2019

When and Where	Role
Date: 10/21/19	**Primary Facilitator:** Teriq
Start: 2:30 pm	**Timekeeper:** Yovanni
End: 3:30 pm	**Minute Taker:** Anthony
Room: ECS 243	**Attending:** Armando, Kian, Teriq, Anthony, Yovanni

1. Status

   Update regarding tasks assigned previous week. Some members have successfully completed their tasks while others must revisit their works due to slight misconceptions found. Start decomposing the system into subsystems.

2. Discussion

   Armando, Yovanni and Kian all finished their tasks successfully. Section 1 of the Design Document is 35% complete and Armando completed the security section for the Design Document. Teriq and Anthony made a slight mistake while creating the ER Diagram as they did not use a validation software so it must be redone. Teriq took responsibility of doing the data dictionaries for the persistent data section.

   Afterwards, the team discussed how the system should be effectively decomposed. The team decided that the system should be decomposed into three layers which includes the Presentation layer, Logic layer and the Storage layer. Within these major subsystems the team identified key partitions that would ensure high cohesion and low coupling between different subsystems.

   The team decided to make 7 different subsystems in the logic layer including SOS server, SOS session manager, SOS Dispatcher, User Management, Event Management, Organization Management and Security Management. The first three subsystems are specialized to retrieve information from the user, keep track of the current status of the system and send data back to the system. The rest of the subsystems found within the Logic layer are to delegate certain operations for the persistent objects that exist within our system which include events, organizations and users.

3. Wrap Up
   - Teriq will complete the data dictionaries for the persistent data section of the design document.
   - Anthony will revisit the ER Diagram and re-do it on STAR UML.
   - Armando and Yovanni will work on the hardware and software mapping of the SOS system.
   - Kian will continue to work on the front end of the SOS system and on Section 1 of the Design Document.

11.7.7 October 28, 2019

When and Where	Role
Date: 10/28/19	**Primary Facilitator:** Teriq
Start: 2:30 pm	**Timekeeper:** Yovanni
End: 3:30 pm	**Minute Taker:** Anthony
Room: ECS 243	**Attending:** Armando, Kian, Teriq, Anthony, Yovanni

1. Status

Update regarding tasks assigned previous week. Member have all successfully completed their tasks. This week we start formatting the Design Document and assigning additional tasks to team members.

2. Discussion

Armando and Yovanni completed the hardware and software mapping of the SOS system. They presented it to the team and after a short debate we decided that it was an adequate representation of how SOS system would be launched. Afterwards, Kian gave an update of what he had completed in the front end of the system and informed us that he had completed Section 1 of the Design Document.

Teriq completed approximately half of the data dictionaries because he was waiting for Anthony to finalize the ER diagram to ensure that both data dictionaries and ER diagram reflected the persistent objects found in our system. Anthony presented the finalize version do on STAR-UML to the team and there was a consensus that made it the final representation of the system database design.

Afterwards, the team decided to go through an overview of the design patterns that our system may have, and we decided to start creating the class diagrams that would represent those patterns. In addition, the team also started thinking about the way the minimal class diagrams would be connected and looking at the requirements to complete section 3 of the design document.

3. Wrap Up

- Teriq needs to finish the data dictionaries as soon as possible.
- Anthony needs to implement the ER diagram into MySQL.
- Armando needs to complete the minimal class diagrams for all the subsystems.
- Kian needs to continue working on the front end as well as the structuring of the design document.
- Yovanni need to start brainstorming and researching the OCL statements for each major subsystem.

11.7.8 November 4, 2019

When and Where	Role
Date: 11/04/19	**Primary Facilitator:** Teriq
Start: 2:30 pm	**Timekeeper:** Yovanni
End: 3:30 pm	**Minute Taker:** Anthony
Room: ECS 243	**Attending:** Armando, Kian, Teriq, Anthony, Yovanni

1. Status

   Update on the status of the tasks that were assigned in the previous week. Assignment of remaining tasks to the team members. Update on the status of the system and the completion of the design document.

2. Discussion

   In this meeting the team decided to first see how much was missing in the design document and what was currently implemented. All of section 1 was complete. All of section 2 had been complete. Armando had finished the overview of the class diagrams for the subsystems along with their descriptions. Yovanni had generated the OCL for the major control objects in each major subsystem. Teriq had finished the data dictionaries and started writing the java class interfaces for the main control object in each subsystem.

   Anthony had finished implementing all of the tables that were specified in the persistent data section in MySQL and he started working on the state machine and object interaction sections of section 3 and was about 25% done. Kian presented his additions in the front end and decided to help Armando with section 3.4.1 and Appendix C.

   The missing tasks for the design document are Appendix E which is currently being finished. Appendix A which is found on the SRD. The approval page, references, glossary and introduction to the object design chapter. Armando decided to take responsibility for these remaining roles. We decided that everything should be finished by November 08, 2019 to allow time for revision.

3. Wrap Up
   - Teriq needs to finish the java class interface for the main control object in each subsystem.
   - Anthony needs to finish the state machine and object interactions.
   - Armando, Yovanni and Kian must work together to finish the Detailed Class Design section along with Appendix C.

11.7.9 November 8, 2019

When and Where	Role
Date: 11/08/19	**Primary Facilitator:** Teriq
Start: 11:30 am	**Timekeeper:** Yovanni
End: 04:30 pm	**Minute Taker:** Anthony
Room: ECS 243	**Attending:** Armando, Kian, Teriq, Anthony, Yovanni

1. Status

   Update with the tasks assigned last week. Proofread the document and revisit all the charts found in the design document to ensure correctness of the deliverable.

2. Discussion

   All members, except Anthony, seem to have finished their sections of the report. Anthony finished object design but seemed to be struggling with the state machine diagram. After reviewing Teriq's interface implementation we decided to readjust some of the code that he had written. Armando, Kian and Yovanni had finished the section that were assigned to them.

   After giving the update of our progress the team spent some time reading the document and debating on the correctness of the ideas expressed within the document. More importantly we made sure that the document was in the correct format and that the reader could find things easily and view diagrams with ease.

   By the time the meeting was coming to a halt Teriq had finished his java class interfaces and had published them in the document. Anthony after help from everyone on the team created a state machine diagram for the overall system and the main control object in each major subsystem. The Design Document was about 90% complete and we decided as a team that over the long weekend we would email the professor about our questions regarding the correctness of our approach and proofread the content found within it.

3. Wrap Up
   - Proofread the document.
   - Ask the professor questions about the confusions found in the document.
   - Turn in the document.

11.7.10 November 13, 2019

When and Where	Role
Date: 11/13/19	**Primary Facilitator:** Anthony
Start: 12:30 pm	**Timekeeper:** Yovanni
End: 03:30 pm	**Minute Taker:** Armando
Room: ECS 243	**Attending:** Armando, Kian, Teriq, Anthony, Yovanni

7. Status

First meeting for deliverable 3. The focus of the meeting was generating tasks from the second deliverable document and assigning the tasks that are immediately available.

8. Discussion

Task Decomposition:

- Read the second deliverable. **All**
- Cover Page – Refine from DD
- Abstract – Something to do at the end.
- Table Of Contents – Something to do at the end.
- Introduction – Refine from SRD
- Section 1.1 – Refine from SRD
- Section 1.2 – Refine from SRD
- Section 1.3 – Refine from DD
- Section 1.4 – Something to do at the end.
- Section 1.5 – **Anthony**
- Section 2 – Something when development is finalized.
- Section 3 – Refinement from SRD **Armando**
- Section 4 – Introduction **Armando**
- Section 4.1 – **Kian & Yovanni**
- Section 4.2 – StarUML.
- Section 4.3 - **Anthony**
- Section 5 Introduction - **Armando**
- 5.1 Refinement from DD
- 5.2 Refinement from DD
- 5.3 Refinement from DD
- 5.4 Refinement from DD
- 5.5 Refinement from DD
- Section 6 Introduction - **Armando**
- 6.1 Refinement from DD
- 6.2 Refinement from DD
- 6.3 Refinement from DD
- 6.4 Refinement from DD
- Section 7 Introduction - **Armando**
- 7.1 **Yovanni**
- 7.2 **Yovanni & Teriq**

- 7.3 **All**
- 7.4 **Yovanni & Teriq**
- Section 8 Refinement from DD & SRD
- Section 9 Take from DD
- Section 10 Something to be done at the end
- Appendices A – Refine from SRD
- Appendices B – Refine from SRD
- Appendices C – Get user interfaces from SOS
- Appendices D – Refine from DD
- Appendices E – Get Java code from SOS
- Appendices F – Get UI from SOS
- Appendices G – Get from SRD, DD, **in progress**

9. Wrap Up

- FSD development will start with the refinement and additions of the above tasks.
- Everyone on the SOS team will create 4 test cases for 2 of the use cases that we will be implementing.
- Yovanni will research testing tools and refresh his software testing knowledge.
- Armando & Kian will finish the implementation of the front end of the SOS
- Anthony & Teriq will finish the implementation of the back-end of the SOS and find a method of connecting the front-end to the back-end.

11.7.11 November 18, 2019

When and Where	Role
Date: 11/18/19	**Primary Facilitator:** Anthony
Start: 02:30 pm	**Timekeeper:** Yovanni
End: 03:30 pm	**Minute Taker:** Armando
Room: ECS 243	**Attending:** Armando, Kian, Teriq, Anthony, Yovanni

1. Status
   Update regarding status of project contributions. The focus of the meeting was to discuss the updates on the FSD. In addition, updates were necessary from all teams including front-end, back-end and testing team.
2. Discussion
   All members seem to have finished their sections of the FSD report. Anthony finished refining all of Section 1 for the report, gave an overview of the FSD in section 1.5 and completed section 4.3. Armando also completed his tasks of adjusting the project schedule and adding all the necessary adjustments in section 5. Kian and Yovanni finalized Section 4.1 and 4.3. Yovanni still seems not to be very comfortable with the testing software that he has researched, he mentioned he will keep searching for solutions. Teriq also agreed. Everyone had finished their test cases except for Anthony because he still did not understand the format of test case and was having trouble coming up with effective test cases.
   After giving the update of our progress the team spent some time reading the document and debating on the correctness of the ideas expressed within the document. More importantly we made sure that the document was in the correct format and that the reader could find things easily and view diagrams with ease.
   By the time the meeting was over we agreed that the refinement of the document was good and we had about 45% of the document complete. Afterwards, the front-end team discussed that they were about 90% with their implementation of the UI. They specifically mentioned they were having trouble with the google maps API. The back-end team announced that they had started filling in the Java skeletons that they had made for the previous deliverable and we argued about the reflection of implementation and our design. The back-end team also had completed most of the stored procedure calls to the mySQL DB. As mentioned before, Yovanni did not seem to understand much of the testing software but he said that he would research more heavily.
3. Wrap Up
   - Anthony and Teriq should re-adjust and finalize their implementation of the back-end by the next team meeting so that Yovanni can test with enough time.
   - Anthony should finish his test cases and compile and add all test cases to the FSD.
   - Armando and Kian should finish with the front-end implementation of the SOS.
   - Yovanni must heavily research how to use the software testing tools to ensure a good section 7.

11.7.12 November 25, 2019

When and Where	Role
Date: 11/25/19	**Primary Facilitator:** Anthony
Start: 02:30 pm	**Timekeeper:** Yovanni
End: 03:30 pm	**Minute Taker:** Armando
Room: ECS 243	**Attending:** Armando, Kian, Teriq, Anthony, Yovanni

1. Status

   Update regarding status of project contributions. The focus of the meeting was to discuss the updates on the development of the SOS. In addition, updates were necessary from the status of the FSD and testing.

2. Discussion

   The meeting began with Anthony saying that he had finished all his use cases and that he had compiled all the use cases into the FSD. In addition, he mentioned that he had used Netty socket.io framework to establish a server that listened for commands from the front-end. Teriq then announced that both him and Anthony had finished the object and class implementations of the back-end and that they were waiting to test them with the connectivity of the front-end.

   Afterwards, the front-end team claimed that they had sort of figured out the google map API function call. They were elated to hear the news and they decided that by the next team meeting they would have a socket.io client implementation on the front-end to start testing for faults and failures in our system before having Yovanni test it out. Lastly, Yovanni gave an update of the knowledge of testing software that he had gained. He claimed that he was more comfortable with it but he just needed to have the software done so that he could give it a shot. Teriq who had been helping Yovanni also said that he sort of understands the software but he is still a little confused about it. We used the last 15 minutes of our meeting to talk and research the testing software.

3. Wrap Up
   - Anthony and Teriq must make sure that the back-end is commented appropriately and that all stored procedure calls to the system function as expected.
   - Kian and Armando must finish the socket.io client in the front-end of the system to connect the UI to the back-end.
   - Yovanni and Teriq must keep researching the software testing applications to ensure they are ready to start testing the system once the front-end and the back-end are connected.

11.7.13 November 27, 2019

When and Where	Role
Date: 11/27/19	**Primary Facilitator:** Anthony
Start: 12:30 pm	**Timekeeper:** Yovanni
End: 06:00 pm	**Minute Taker:** Armando
Room: ECS 243	**Attending:** Armando, Kian, Teriq, Anthony, Yovanni

1. Status

   Update with the tasks assigned last week. Test the functionality of the front-end and back-end communication and discuss tasks given during break.

2. Discussion

   The meeting began with the front-end team saying that they had finished the client emitters for the socket.io. Afterwards we ran the driver for the back-end server, and we attempted to listen for a connection. Since it was the first time most of us were doing this it took us a while before we finally got a message from the front end. After getting the first message from the front end, we started testing all the workflows. Most of the workflows did work but many of them did not. Both the front-end and the back-end team sat together to debug the errors found in the data storage and server side of the system.

   Afterwards, we decided that we had a SOS that worked effectively except for the fact that we did not have encryption. Armando said that he would like to work on the encryption because he was interested in understanding how it works. In addition, we told Yovanni and Teriq that the website was ready for testing and they said that they were ready for the action. We explained to Yovanni how most of the backend works and he started deciding what subsystems he would test first using our past class diagrams in the DD.

3. Wrap Up
   - Anthony will work on the PowerPoint presentation for the possibility of the SOS team presenting on Tuesday.
   - Armando, Anthony and Kian will work to finalize the FSD.
   - Yovanni and Teriq will work on the testing of the SOS.
   - Armando and Anthony will revisit the DD sections of the FSD and refine it based on past criticisms from Dr. Clarke.

11.7.14 December 2, 2019

When and Where	Role
Date: 12/02/19	**Primary Facilitator:** Anthony
Start: 12:30 pm	**Timekeeper:** Yovanni
End: 03:30 pm	**Minute Taker:** Armando
Room: ECS 243	**Attending:** Armando, Kian, Teriq, Anthony, Yovanni

1. Status

   Update with the tasks assigned last week. Rehearse presentation timing and speech. Find a flash drive and put all the presentations, documents, UML diagrams and source code.

2. Discussion

   After a long Thanksgiving break the team met up one last time. Yovanni and Teriq said they still needed more time finishing up some of the test cases and that they would have it done by latest tomorrow noon. Anthony, Armando and Kian had almost finished the FSD but they were missing the testing parts of the report. Armando and Anthony had revised Dr. Clarke's previous criticism and ensured that everything was updated appropriately.
   Kian volunteered his USB to turn in the work for this project. The team plans on finalizing everything by tomorrow noon and writing the User's Guide today to turn in our project. The team will put everything the have on Github on the USB. Afterwards, the team presented 2-3 times until everyone was confident and had a great tone when presenting in public. Teriq struggled a little in the presenting but he seemed more confident after he practiced with the team. Anthony gave everyone a copy of the PowerPoint presentation and split the slides among the team members in the following manner:

   Kian:
   -Title Page
   - SOS07 – Functional & Nonfunctional Requirements
   - SOS07 – Test Case Sunny

   Yovanni:
   -UML Class Diagrams
   - SOS16 – Functional & Nonfunctional Requirements

   Teriq:
   -SOS07 – Rainy
   -SOS16 – Sunny & Rainy

   Armando:
   -Purpose and Scope of System
   -Project Schedule
   -Security and Privacy

   Anthony:
   -Data Management
   -Minimal Class Diagram
   -Sequence Diagram

3. Wrap Up

- Anthony will work on the PowerPoint presentation for the possibility of the SOS team presenting on Tuesday.
- Armando, Anthony and Kian will work to finalize the FSD.
- Yovanni and Teriq will work on the testing of the SOS.
- Armando and Anthony will revisit the DD sections of the FSD and refine it based on past criticisms from Dr. Clarke.