Design Document

CEN4010 – Software Engineering

Prepared for Peter Clarke

By Team 5

Teriq Douglas

Yovanni Jones

M. Kian Maroofi

Armando J. Ochoa

Anthony Sanchez-Ayra

November 12, 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.

Table of Contents

[Abstract 2](#_Toc24413114)

[1 Introduction 6](#_Toc24413115)

[1.1 Purpose of the System 6](#_Toc24413116)

[1.2 Requirements 6](#_Toc24413117)

[1.2.1 Functional Requirements 6](#_Toc24413118)

[1.2.2 Non-Functional Requirements 7](#_Toc24413119)

[1.2.2.1 Usability 7](#_Toc24413120)

[1.2.2.2 Reliability 7](#_Toc24413121)

[1.2.2.3 Performance 7](#_Toc24413122)

[1.2.2.4 Supportability 7](#_Toc24413123)

[1.2.2.5 Implementation 7](#_Toc24413124)

[1.3 Development Methodology 8](#_Toc24413125)

[1.4 Definitions, Acronyms, and Abbreviations 9](#_Toc24413126)

[1.5 Overview of the Document 9](#_Toc24413127)

[2 Proposed Software Architecture 10](#_Toc24413128)

[2.1 Overview 10](#_Toc24413129)

[2.2 Subsystem Decomposition. 11](#_Toc24413130)

[2.3 Hardware and Software Mapping 12](#_Toc24413131)

[2.4 Persistent Data Management 12](#_Toc24413132)

[2.5 Security Management 16](#_Toc24413133)

[2.5.1 Password Management. 16](#_Toc24413134)

[2.5.2 Access Management 17](#_Toc24413135)

[3 Detailed Design 18](#_Toc24413136)

[3.1 Overview 18](#_Toc24413137)

[3.1.1 SOS Website 18](#_Toc24413138)

[3.1.2 SOS Interface 19](#_Toc24413139)

[3.1.3 User Management 20](#_Toc24413140)

[3.1.4 Event Management 20](#_Toc24413141)

[3.1.5 Organization Management 21](#_Toc24413142)

[3.1.6 Security Management 22](#_Toc24413143)

[3.1.7 SOS Storage 23](#_Toc24413144)

[3.1.8 Google Maps GPS API 24](#_Toc24413145)

[3.2 State Machine 25](#_Toc24413146)

[3.3 Object Interaction 25](#_Toc24413147)

[3.3.1 Sequence Diagram for SOS01 – Create an Event 26](#_Toc24413148)

[3.3.2 Sequence Diagram for SOS04 – Attending an Event. 27](#_Toc24413149)

[3.3.3 Sequence Diagram for SOS02 – Grant Organizer Role 28](#_Toc24413150)

[3.3.4 Sequence Diagram for SOS07 – Edit Profile 29](#_Toc24413151)

[3.3.5 Sequence Diagram for SOS16 – Create Organization 30](#_Toc24413152)

[3.3.6 Sequence Diagram for SOS17 – Cancel an Event 31](#_Toc24413153)

[3.3.7 Sequence Diagram for SOS22 – Registration 32](#_Toc24413154)

[3.3.8 Sequence Diagram for SOS10 – Access an Event by Location 33](#_Toc24413155)

[3.3.9 Sequence Diagram for SOS31 – Login 34](#_Toc24413156)

[3.3.10 Sequence Diagram for SOS32 – Logout 35](#_Toc24413157)

[3.4 Detailed Class Design 36](#_Toc24413158)

[3.4.1 Class Description 36](#_Toc24413159)

[3.4.1.1 SOS Website 36](#_Toc24413160)

[3.4.1.2 SOS Interface 37](#_Toc24413161)

[3.4.1.3 User Management 38](#_Toc24413162)

[3.4.1.4 Event Management 39](#_Toc24413163)

[3.4.1.5 Organization Management 39](#_Toc24413164)

[3.4.1.6 Security Management 40](#_Toc24413165)

[3.4.1.7 SOS Storage 40](#_Toc24413166)

[3.4.2 Control Objects Description 41](#_Toc24413167)

[3.4.2.1 SOS Server OCL 41](#_Toc24413168)

[3.4.2.2 User Manager OCL 42](#_Toc24413169)

[3.4.2.3 Organization Manager OCL 42](#_Toc24413170)

[3.4.2.4 Event Manager OCL 43](#_Toc24413171)

[3.4.2.5 PasswordManager OCL 43](#_Toc24413172)

[3.4.2.6 AccessManager OCL 43](#_Toc24413173)

[4 Glossary 44](#_Toc24413174)

[5 Approval Page: 45](#_Toc24413175)

[6 References 46](#_Toc24413176)

[7 Appendices 47](#_Toc24413177)

[7.1 Appendix A – Use Case Diagram 47](#_Toc24413178)

[7.2 Appendix B – Implemented Use Cases 48](#_Toc24413179)

[7.2.1 Create Event 48](#_Toc24413180)

[7.2.2 Grant Organizer Role 50](#_Toc24413181)

[7.2.3 Attending an Event 52](#_Toc24413182)

[7.2.4 Edit Profile 54](#_Toc24413183)

[7.2.5 Access Events by Location 56](#_Toc24413184)

[7.2.6 Create Organization 58](#_Toc24413185)

[7.2.7 Cancel an Event 60](#_Toc24413186)

[7.2.8 Registration 62](#_Toc24413187)

[7.2.9 Log in 64](#_Toc24413188)

[7.2.10 Log Out 66](#_Toc24413189)

[7.3 Appendix C – Detailed Subsystem Class Diagrams 68](#_Toc24413190)

[7.3.1 SOS Website 68](#_Toc24413191)

[7.3.2 SOS Interface 69](#_Toc24413192)

[7.3.3 User Management 69](#_Toc24413193)

[7.3.4 Event Management 70](#_Toc24413194)

[7.3.5 Organization Management 70](#_Toc24413195)

[7.3.6 Security Management 71](#_Toc24413196)

[7.3.7 SOS Storage 71](#_Toc24413197)

[7.4 Appendix D - Class Interfaces 72](#_Toc24413198)

[7.5 Appendix E – Diary of Meetings 101](#_Toc24413199)

[7.5.1 October 7, 2019 101](#_Toc24413200)

[7.5.2 October 14, 2019 103](#_Toc24413201)

[7.5.3 October 21, 2019 104](#_Toc24413202)

[7.5.4 October 28, 2019 104](#_Toc24413203)

[7.5.5 November 4, 2019 106](#_Toc24413204)

[7.5.6 November 8, 2019 106](#_Toc24413205)

# Introduction

The following chapter introduces the Design Document (DD) with the goal of explaining how certain qualities of the Student Organization System (SOS) project should be optimized.

The purpose of the Design Document (DD) is to define the decomposition of the system and object design of the SOS to create both a stable and efficient architecture. When decomposing the system, the team first applied two architectural patterns that were found fit for the system. Afterwards, the team decided to investigate object design patterns to ensure that the software of our system would be reusable and easily modifiable. In addition, in this DD we also listed major policy decisions such as control flow, access control and data storage of the SOS.

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).

## 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.).

## Scope of the System.

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

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

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

1. By collating all the organizations that they belong (or might interested in) to in a single place.
2. By presenting them with events in their surrounding hosted by their organizations or any other public events.
3. 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:

1. Integration with social media sites (e.g., Facebook)
2. Social media features such as comments and postings.
3. User-defined (as contrasted to organization-defined) events.
4. Detailed organization leader management features (e.g., payments, duties, etc.)
5. Detailed organization tasks and projects management systems.
6. Organization-relation features (e.g., community chapters)

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

## 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 third model, the design model described in Chapters 6-9. 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 2.1 and 2.2 contain an overview of these subsystems in the form of a top-level UML Package Diagram. In addition, Section 3.1 contains the detailed designs of each of the subsystems in the form of simplified UML Class Diagrams. A more descriptive view of the UML Class Diagrams for the subsystem decompostion are found in Appendix C. A simplified version of the implementation model, is also presented in this document, in Section 2.3, 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.

## 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

## Overview of the Document

This document is structured into chapters, each describes a different aspect of the project's system design. Chapter 2 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 3 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 4 goes through the glossary which defines the domain-specific keywords and terms used in this document. Chapter 5 is the approval page of the document which contains all of the SOS team members' signatures. Chapter 6 includes the references used in the document.

Finally, Chapter 7 contains five different appendixes (A-E). Appendix A includes the project's use case diagram, Appendix B includes detailed descriptions of each implemented use case in the project (total of 10). Moreover, Appendix C contains detailed subsystem class diagrams (total of 7), Appendix D describes the Java code for the subsystem classes, Appendix E contains diaries for every meeting held during this deliverable milestone.

# Current Systems

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 similar to 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).

# 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.

## 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 | Front-End Developer, Document Editor |
| Anthony Sanchez-Ayra | Back-End Developer, Database implementor |
| M. Kian Maroofi | Front-End Developer |
| Yovani Jones | Back-End Developer, Tester |
| Teriq Douglas | Back-End Developer |

*Table 3: The development roles assigned to the team members.*

## Hardware and Software Requirements

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

### 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.

### 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.

## 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.

### 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 4: Task Schedule for the Project*

# Requirements of the System

## Functional and Nonfunctional Requirements

The following subsections define the functional and non-functional requirements of the SOS system. TODO: Add requirements for all use cases.

### 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 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 users to check-in for each event on the platform (see Use Case SOS04 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 find all nearby events based on the user’s current location (see Use Case SOS10 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 visitors to register for a new account (see Use Case SOS22 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).

### 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.

#### 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.

#### 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.

#### 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.

#### Supportability

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

#### Implementation

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

## Use Case Diagram

The Use Case diagram describing the whole system is shown in *Figures 1* to *Figure 8*. 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 1: Use Case diagram for the whole system.



Figure 2: General Use Cases, all of which represent use cases that affects all users of the system.



Figure 3: Location Use Case Diagram, which collects the use cases having to do with the external geolocation API.



Figure 4: Security Use Cases, which collects all the use cases having to do with security.



Figure 5: Event Use Case Diagram, which collects all the use cases related to event creation, destruction, etc.



Figure 6: Score Use Case Diagram, which collects all the use cases having to do with the ranking and point system.



Figure 7: Organization Use Case Diagram, which collects all the use cases having to do with organizations.



Figure 8: 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.

## Requirements Analysis

TODO: Add something here. Summary of Chapter 5 in SRD.

# Software Architecture

NOTE: Update this with descriptions of the actual software structure.

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 2.1 contains a general overview of the system, including a general description of the architectural patterns used. Following that, Section 2.2 contains a subsystem decomposition for the SOS. Section 2.3 contains a UML Deployment Diagram showing the hardware and software mapping expected for the system. Section 2.4 contains the requirements and schema used for persistent data in the system. Finally, Section 2.5 contains the security requirements and schema for the system.

## Overview



Figure 2: Class Diagram showing the top-level architecture of the Student Organization System.

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.

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. 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.

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.

## 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).

## Hardware and Software Mapping

The hardware and software mapping for the SOS system can be seen in the UML Deployment diagram in Figure 3. 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 3: UML Deployment Diagram for the SOS system.

## 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 4.



Figure 4: 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, etc.
* *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 4, as well as field size, data type, and format, can be seen in the data dictionary in Table 2.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |
| location | ------ | Point | ----------------- | FIU | A point to a location using the Google GPS API. |
| 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 2: Data Dictionary for the SOS Persistent Data.

## Security Management

The SOS uses two core security mechanism, Password Management, which is described in Section 2.5.1, and Access Management, which is described in Section 2.5.2. In both cases, the relevant classes are implemented in the Security Subsystem (see Section 3.1.7). Besides these two functionalities, other systems are also used such as API Keys for the Google Maps GPS API and Encryption for network sharing of important data.

### 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.

In order to ensure hard-to-crack password, the following policy will be enforced:

* Passwords must be at least 6 characters long.
* Passwords must have at least 1 uppercase character.
* Passwords must have at least 1 number or special character.

### 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 3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | 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 3: 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.

# 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 3.1 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 3.2 contains a state machine for the SOS in the form of a UML State Chart Diagram. Section 3.3 contains the object interactions for each of the implemented use cases of the SOS. Finally, Section 3.4 contains a detailed description of each class of the implemented subsystems, as well as OCL constrains for the control object on each subsystem.

## 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 2.2. For each minimal Class Diagram, a complete equivalent diagram with attributes and operations can be found in Appendix C.

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

### SOS Website

The minimal class diagram for the SOS Website subsystem can be seen in Figure 5. A full equivalent class diagram can be found in Appendix C.



Figure 5: 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.

### SOS Interface

The minimal class diagram for the SOS Controller subsystem can be seen in Figure 6. A full equivalent class diagram can be found in Appendix C.



Figure 6: Minimal Class Diagram for SOS Interface.

* The following classes belong to this subsystem:
* SOS Server, which the main server instance of the system.
* SOS Session Manager, which has functions relating to system sessions.
* SOS Dispatcher, which propagates front-end request to their specific target controllers.

### User Management

The minimal class diagram for the User Management subsystem can be seen in Figure 7. A full equivalent class diagram can be found in Appendix C.



Figure 7: 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.

### Event Management

The minimal class diagram for the Event Management subsystem can be seen in Figure 8. A full equivalent class diagram can be found in Appendix C.

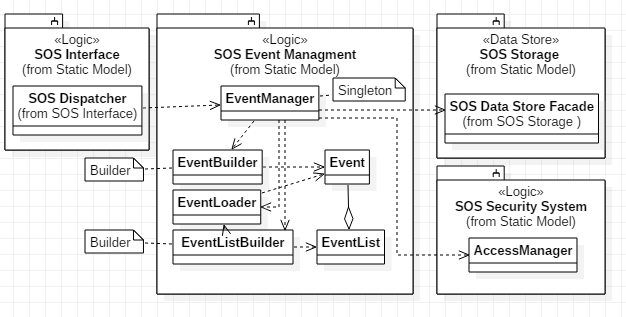


Figure 8: Minimal Class Diagram for SOS Event Management

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.

### Organization Management

The minimal class diagram for the Organization Management subsystem can be seen in Figure 9. A full equivalent class diagram can be found in Appendix C.



Figure 9: 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.

### Security Management

The minimal class diagram for the Security Management subsystem can be seen in Figure 10. A full equivalent class diagram can be found in Appendix C.



Figure 10: Minimal Class Diagram for SOS Security.

The following classes belong to this subsystem:

* PasswordManager, which is a Singleton dealing with password control actions.
* AccessManager, which is a Singleton dealing with access control actions.

### SOS Storage

The minimal class diagram for the SOS Storage subsystem can be seen in Figure 11. A full equivalent class diagram can be found in Appendix C.



Figure 11: 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).

### 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.

## State Machine

The state machine for the SOS is shown in Figure 12. 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.

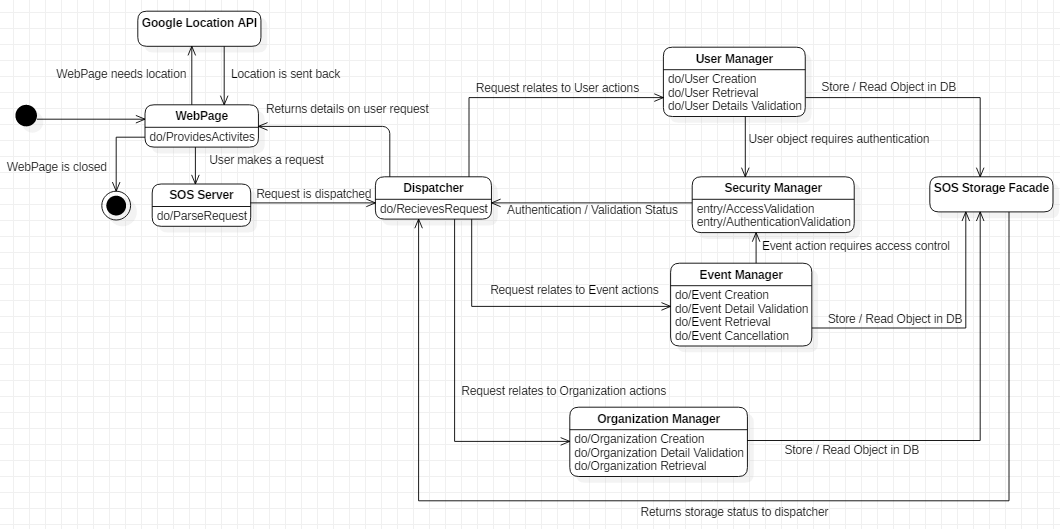


Figure 12: State Machine for the Student Organization System.

## 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.

### Sequence Diagram for SOS01 – Create an Event

The sequence diagram in Figure 13 corresponds to the Use Case in Appendix B, Section 7.2.1.



Figure 13: Sequence Diagram for SOS01 - Create an Event

### Sequence Diagram for SOS04 – Attending an Event.

The sequence diagram in Figure 14 corresponds to the Use Case in Appendix B, Section 7.2.2.



Figure 14: Sequence Diagram for SOS04 – Attending an Event.

### Sequence Diagram for SOS02 – Grant Organizer Role

The sequence diagram in Figure 15 corresponds to the Use Case in Appendix B, Section 7.2.3.



Figure 15: Sequence Diagram for SOS02 – Grant Organizer Role.

### Sequence Diagram for SOS07 – Edit Profile

The sequence diagram in Figure 16 corresponds to the Use Case in Appendix B, Section 7.2.4.



Figure 16: Sequence Diagram for SOS07 – Edit Profile.

### Sequence Diagram for SOS16 – Create Organization

The sequence diagram in Figure 17 corresponds to the Use Case in Appendix B, Section 7.2.5.



Figure 17: Sequence Diagram for SOS16 – Create Organization.

### Sequence Diagram for SOS17 – Cancel an Event

The sequence diagram in Figure 18 corresponds to the Use Case in Appendix B, Section 7.2.6.



Figure 18: Sequence Diagram for SOS17 – Cancel an Event.

### Sequence Diagram for SOS22 – Registration

The sequence diagram in Figure 19 corresponds to the Use Case in Appendix B, Section 7.2.7.



Figure 19: Sequence Diagram for SOS22 – Registration.

### Sequence Diagram for SOS10 – Access an Event by Location

The sequence diagram in Figure 20 corresponds to the Use Case in Appendix B, Section 7.2.8.



Figure 20: Sequence Diagram for SOS10 – Access an Event by Location.

### Sequence Diagram for SOS31 – Login

The sequence diagram in Figure 21 corresponds to the Use Case in Appendix B, Section 7.2.9.



Figure 21: Sequence Diagram for SOS31 – Login.

### Sequence Diagram for SOS32 – Logout

The sequence diagram in Figure 22 corresponds to the Use Case in Appendix B, Section 7.2.10.



Figure 22: Sequence Diagram for SOS32 – Logout

## Detailed Class Design

This section contains the detailed class design of each class in each subsystem of the SOS system. Section 3.4.1 contains a detailed description for each class in the system, while Section 3.4.2 contains the Object Constraint Language (OCL) constraints for each control objects of each subsection.

### 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 C.

#### SOS Website

The complete class diagram for the SOS Website, which contains operations and attributes where applicable, can be found in Figure 30 in Appendix C, Section 7.3.1. The Java Code interface for these classes are in Appendix D, Section 7.4.1. 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.

#### SOS Interface

The complete class diagram for the SOS Interface, which contains operations and attributes where applicable, can be found in \*Figure 31\* in Appendix C, Section 7.3.2. The Java Code interface for these classes are in Appendix D, Section 7.4.2. 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. The SOS Server also contains an instance of the SOS Session Manager class to keep and return Session data to the User when necessary. Note that this class is a Singleton, as only one SOS Server is necessary for the entire system.
* SOS Session Manager, which has functions relating to system session. It oversees controlling real-time elements of the system (e.g., system time, which affects the Events that Users can Attend, as well as past and future Events) and keeps track of logged-in Users and their local actions when applicable. It is a Singleton, as only one Session Manager is necessary and more than one could create session conflicts that must be avoided.
* SOS Dispatcher, which is a Command class which propagates the front-end requests to their specific target controllers. The requests messages which are parsed and pre-processed by the SOS Server then are used to call an appropriate dispatch from the SOS Dispatcher, which is tasked with calling all other controllers. Instead of being their own classes, each subcommand is defined in terms of parametrizations of the dispatch function within the SOS Dispatcher. Internally, SOS Dispatcher also keeps track of these requests and stores them in the database.

#### User Management

The complete class diagram for the User Management, which contains operations and attributes where applicable, can be found in Figure 32 in Appendix C, Section 7.3.3. The Java Code interface for these classes are in Appendix D, Section 7.4.3. 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).

#### Event Management

The complete class diagram for the Event Management, which contains operations and attributes where applicable, can be found in Figure 33 in Appendix C, Section 7.3.4. The Java Code interface for these classes are in Appendix D, Section 7.4.4. 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).

#### Organization Management

The complete class diagram for the Organization Management, which contains operations and attributes where applicable, can be found in Figure 34 in Appendix C, Section 7.3.5. The Java Code interface for these classes are in Appendix D, Section 7.4.5. 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).

#### Security Management

The complete class diagram for the Security Management, which contains operations and attributes where applicable, can be found in Figure 35 in Appendix C, Section 7.3.6. The Java Code interface for these classes are in Appendix D, Section 7.4.6. 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.
* AccessManager, which is a Singleton dealing with access control actions. It implements most of the back-end side of the access policy defined in Section 2.5.2 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.

#### SOS Storage

The complete class diagram for the SOS Storage, which contains operations and attributes where applicable, can be found in Figure 36 in Appendix C, Section 7.3.7. The Java code interface for these classes is in Appendix D, Section 7.4.7. 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.

### 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.

#### SOS Server OCL

The OCL specification for the SOS Server class is defined in Figure 23.

**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 23: OCL specification for SOS Sever.

#### User Manager OCL

**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)

The OCL specification for the User Manager class is defined in Figure 24.

Figure 24: OCL specification for User Manager.

#### Organization Manager OCL

The OCL specification for the Organization Manager class is defined in Figure 25: OCL specification for Organization Manager..

**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))

Figure 25: OCL specification for Organization Manager.

#### Event Manager OCL

The OCL specification for the Event Manager class is defined in Figure 26.

**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 26: OCL specification for Event Manager.

#### PasswordManager OCL

The OCL specification for the PasswordManager class is defined in Figure 27.

**context** PasswordManager

**inv**: self.passwordPolicy <> null

Figure 27: OCL specification for PasswordManager.

#### AccessManager OCL

The OCL specification for the AccessManager class is defined in Figure 28.

**context** PasswordManager

**inv**: self.accessPolicy <> null

Figure 28: OCL specification for AccessManager.

# 

# Testing Process

TODO: Add this whole section.

# 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.

# Approval Page:

**Approval Page of System Requirements Document of**

**Student Organization System**

**Member Signatures**

Armando J. Ochoa 11/12/2019

Member Signature Date

Yovanni Jones 11/12/2019

Member Signature Date

M.Kian Maroofi 11/12/2019

Member Signature Date

Teriq Douglas 11/12/2019

Member Signature Date

Anthony Sanchez-Ayra 11/12/2019

Member Signature Date

# References

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

# Appendices

## Appendix A – Project Schedule

The scheduled tasks are contained in Section 3.3. The Gantt chart of the schedule is below, in Figure 31



Figure 31: Gantt chart for the full project schedule.

## 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.

### 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
  1. No previous training or knowledge.
  2. Tutorial or Help frame should be provided.
  3. Organizer should take less than 10 minutes to create an event.
* Reliability
  1. Mean Time to Failure – 5% failure monthly is acceptable.
  2. Availability
     + Downtime for Login Back-up – 30 minutes in a 24-hour period.
     + Downtime for Maintenance – 1 hour in a 2 weeks period.
* Performance
  1. The form should be sent and saved within 10 seconds.
  2. The system should be able to handle 50 requests in 1 minute.
* Supportability
  1. The Event Creation should be supported by Chrome, Mozilla, and IE.
* Implementation
  1. 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

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

**Modification History**

**Owner:** Armando J. Ochoa

**Initiation date:** 09/01/2019

**Date last modified:** 09/15/2019

### 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
  1. No previous training or knowledge. The system should respond without user interaction after the attendance is completed.
* Reliability
  1. Meant Time to Failure: 5% failure monthly is acceptable.
* Performance
  1. The system should be able to handle 20 requests in 1 minute.
  2. The system should update the Event, Member, and Organization logs within 2 seconds.
* Supportability
  1. Point earning should be supported by Chrome, Mozilla, and IE.
* Implementation
  1. 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

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

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

**Modification History**

**Owner:** Armando J. Ochoa

**Initiation date:** 09/01/2019

**Date last modified:** 09/15/2019

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

**Modification History**

**Owner:** Armando J. Ochoa

**Initiation date:** 09/01/2019

**Date last modified:** 09/15/2019

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

**Modification History**

**Owner:** Kian Maroofi

**Initiation date:** 09/10/2019

**Date last modified:** 09/27/2019

### 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
  1. No previous training or knowledge.
* Reliability
  1. Meant Time to Failure: 5% failure monthly is acceptable.
* Performance
  1. The system should be able to handle 20 requests in 1 minute.
  2. Sharing should happen instantly.
* Supportability
  1. Point earning should be supported by Chrome, Mozilla, and IE.
* Implementation
  1. 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

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

**Modification History**

**Owner:** Kian Maroofi

**Initiation date:** 09/10/2019

**Date last modified:** 09/15/2019

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

**Modification History**

**Owner:** Kian Maroofi

**Initiation date:** 09/10/2019

**Date last modified:** 09/15/2019

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

**Modification History**

**Owner:** Kian Maroofi

**Initiation date:** 09/10/2019

**Date last modified:** 09/22/2019

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

**Modification History**

**Owner:** Kian Maroofi

**Initiation date:** 09/10/2019

**Date last modified:** 09/15/2019

### Kick 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:
  1. No previous training required.
  2. On average the user should take 2 minutes to complete the kick request to the system.
* Reliability
  1. Mean time to failure – 5% failures for every month of operation is acceptable.
  2. Availability – Down time for Login Back-up 30 minutes in a 24 hour period.
* Performance
  1. Request should be sent and saved within 6 seconds.
  2. System should be able to handle 100 requests in 1 minute.
* Supportability
  1. The Event Creation should be supported by Chrome, Mozilla, and IE.
* Implementation
  1. 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

### Create 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 Organizerclicks 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:
     1. Kick
     2. Invite
     3. Promote
     4. Manage Event
     5. Manage Roles
   * **Security Requirement**, which come from a set list including:
     1. 2-Factor Authentication
     2. 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:
  1. No previous training required.
  2. On average the user should take 2 minutes to complete the promotion request to the system.
* Reliability
  1. Mean time to failure – 5% failures for every month of operation is acceptable.
  2. Availability – Down time for Login Back-up 30 minutes in a 24 hour period.
* Performance
  1. Request should be sent and saved within 6 seconds.
  2. System should be able to handle 100 requests in 1 minute.
* Supportability
  1. Shall be supported by Chrome, Mozilla, and IE.
* Implementation
  1. 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

### Notifications

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

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

### Cancel 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:
  1. No previous training required.
  2. On average the user should take 2 minutes to complete the notification request to the system.
* Reliability
  1. Mean time to failure – 5% failures for every month of operation is acceptable.
  2. Availability – Down time for Login Back-up 30 minutes in a 24 hour period.
* Performance
  1. Request should be sent and saved within 6 seconds.
  2. System should be able to handle 100 requests in 1 minute.
* Supportability
  1. Shall should be supported by Chrome, Mozilla, and IE.
* Implementation
  1. 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

### Create 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.

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

### Request 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:
  1. No previous training required.
  2. Should take under 5 minutes to acquire info on organization
* Reliability
  1. Mean time to failure – 5% failures for every 24 hours of operation is acceptable.
  2. Availability
     + Downtime for Login Back-up – 30 minutes in a 24-hour period.
     + Downtime for Maintenance – 1 hour in a 2 weeks period.
* Performance
  1. System should be able to handle 100 requests in 1 minute.
* Supportability
  1. Shall be supported by Chrome, Mozilla, and IE.
* Implementation
  1. 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

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

### Avoid 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
  1. No previous training or knowledge.
  2. Tutorial or Help frame should be provided.
  3. Organizer should take less than 10 minutes to create an event.
* Reliability
  1. Mean Time to Failure – 5% failure monthly is acceptable.
  2. Availability
     + Downtime for Login Back-up – 30 minutes in a 24-hour period.
     + Downtime for Maintenance – 1 hour in a 2 weeks period.
* Performance
  1. The form should be sent and saved within 10 seconds.
  2. The system should be able to handle 50 requests in 1 minute.
* Supportability
  1. The Event Creation should be supported by Chrome, Mozilla, and IE.
* Implementation
  1. 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

### Registration

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

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

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

### Filter 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:
  1. No previous training time, no explicit instructions required.
  2. Should take about 30 seconds for the average user to complete the use case.
* Reliability
  1. Mean time to failure – 5% failures for every month of operation is acceptable.
  2. Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
* Performance
  1. The page should be updated in real time as the user clicks on each tag.
* Supportability
  1. The Event Creation should be supported by Chrome, Mozilla, and IE.
* Implementation
  1. 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

### Invite 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:
  1. Might require light training.
  2. One help frame on the Help page provided.
  3. On average the user should take 1 minute to update their roster.
* Reliability
  1. Mean time to failure – 1% failures for every month of operation is acceptable.
  2. Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
* Performance
  1. System should be able to handle 100 requests in 1 minute.
* Supportability
  1. The Event Creation should be supported by Chrome, Mozilla, and IE.
* Implementation
  1. 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

### Remove 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:
  1. Might require light training.
  2. One help frame on the Help page provided.
  3. On average the user should take 1 minute to update their roster.
* Reliability
  1. Mean time to failure – 1% failures for every month of operation is acceptable.
  2. Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
* Performance
  1. System should be able to handle 100 requests in 1 minute.
* Supportability
  1. The Event Creation should be supported by Chrome, Mozilla, and IE.
* Implementation
  1. 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

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

### Unauthorized 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:
  1. Requires no training.
  2. 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
  1. Mean time to failure – 1% failures for every month of operation is acceptable.
  2. Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
* Performance
  1. Should be able to produce results within 3 seconds.
* Supportability
  1. The Event Creation should be supported by Chrome, Mozilla, and IE.
* Implementation
  1. 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

### Unauthorized 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:
  1. Requires no training.
  2. 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
  1. Mean time to failure – 1% failures for every month of operation is acceptable.
  2. Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
* Performance
  1. Should be able to produce results within 3 seconds.
* Supportability
  1. The Event Creation should be supported by Chrome, Mozilla, and IE.
* Implementation
  1. 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

### Log 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:
  1. Requires no training.
  2. On average the user should take less than 10 seconds to type their information and attempt to log in.
* Reliability
  1. Mean time to failure – 1% failures for every month of operation is acceptable.
  2. Availability – Down time for Login Back-up 30 minutes in a 24-hour period.
* Performance
  1. Should be able to produce results within 3 seconds.
* Supportability
  1. The Event Creation should be supported by Chrome, Mozilla, and IE.
* Implementation
  1. 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

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

## Appendix C – User Interface Designs

TODO: Add UI designs for the rest of the site.

This section contains still from the SOS prototype describing the UI layout.

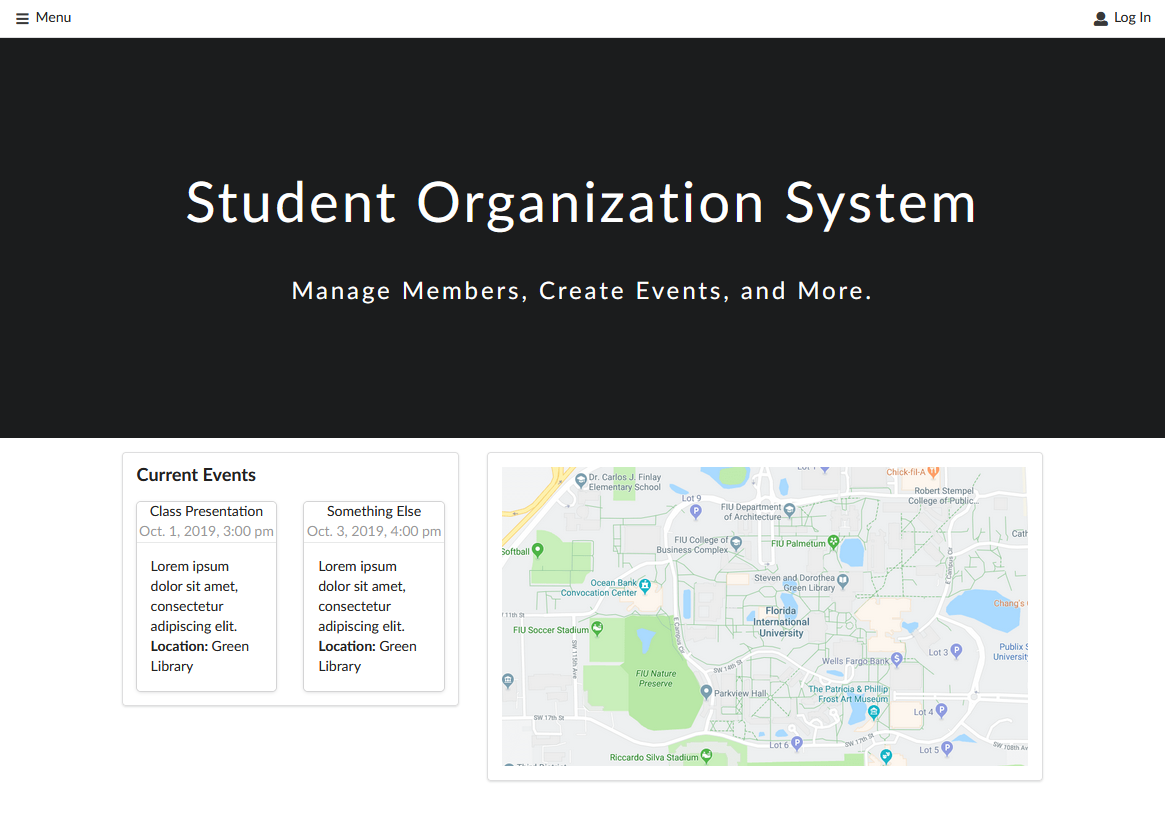


Figure 32: UI Layout for the Homepage. It shows the top-bar, which is a static element of the while system which contains a button to open the navigation menu, and another one to open a log-in pop-up.

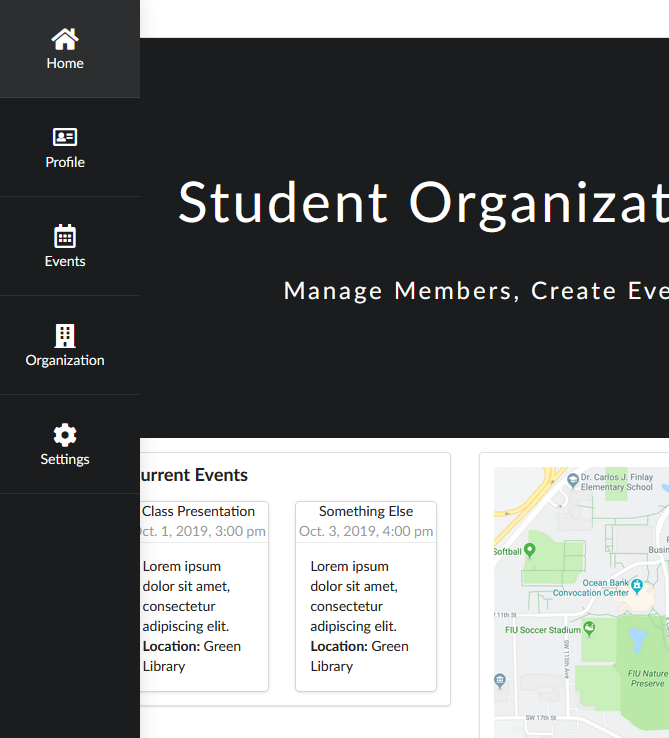


Figure 33: UI Layout for the menu sidebar (with the Homepage on the background). Each button redirects to a specific section on the website that the user can access.

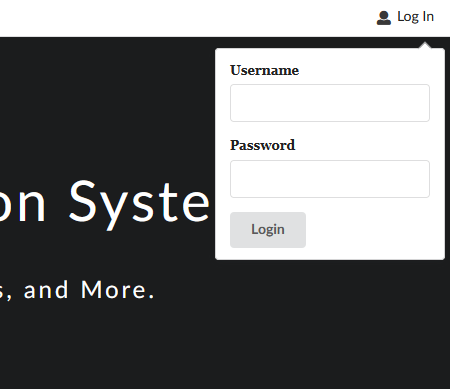


Figure 34: UI Layout for the Log In pop-up (with the Homepage on the background).

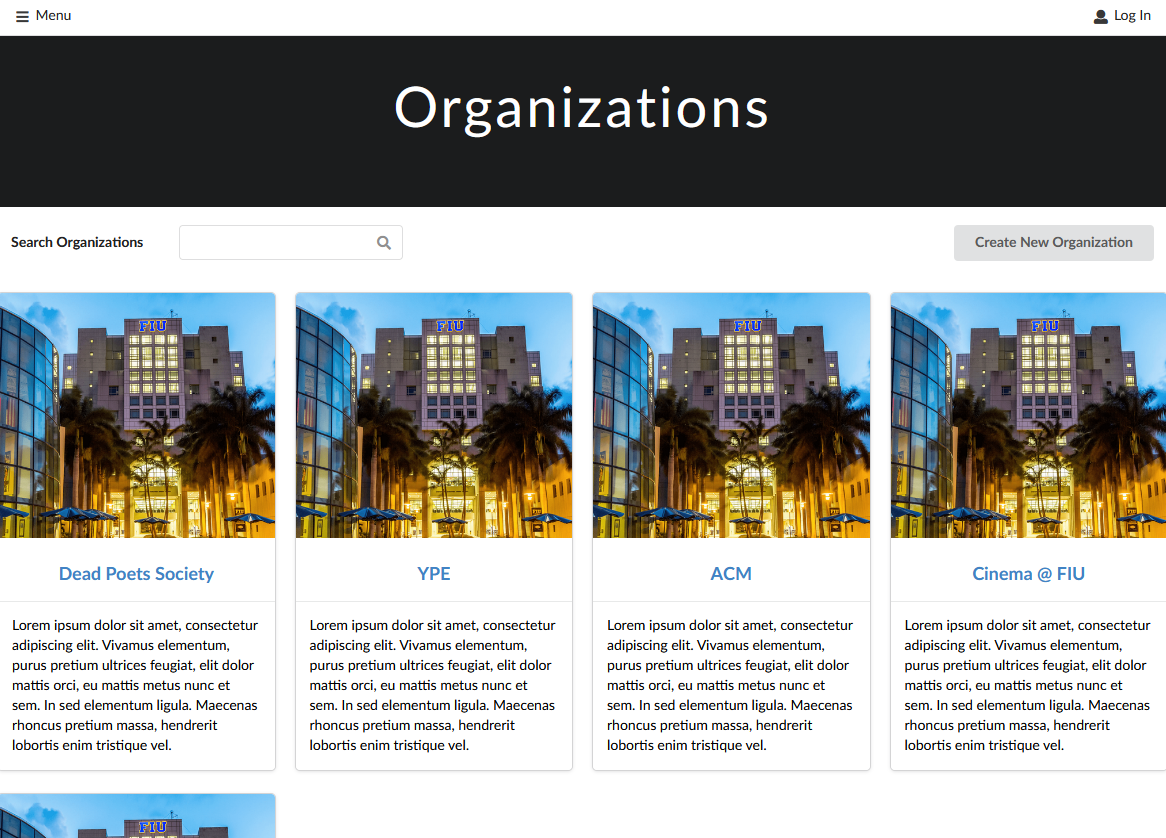


Figure 35: UI layout for the Organizations page. Each organization in the system that is displayed to the user is displayed as at-a-glance card. The create organization button triggers a modal/form (see Figure 36) to define a new Organization.

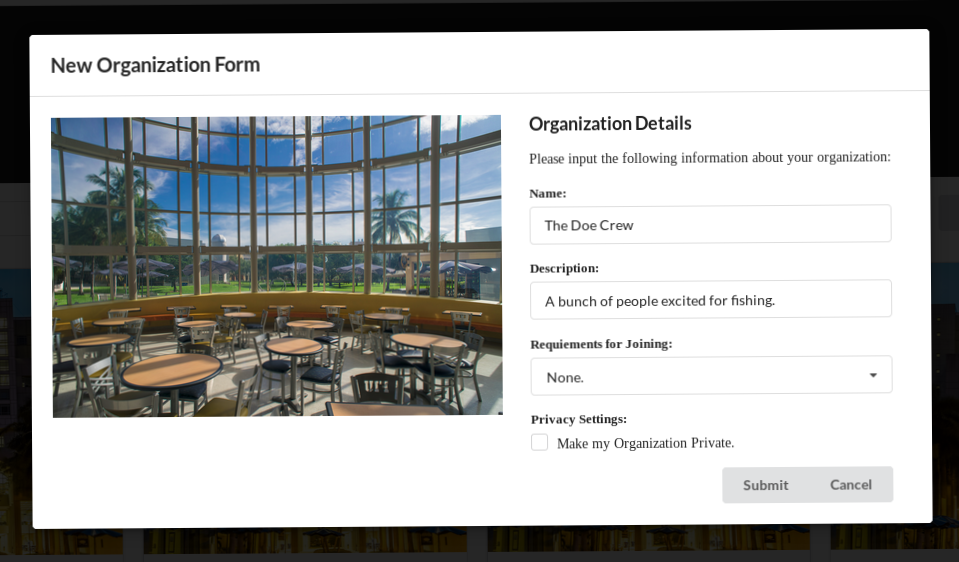


Figure 36: UI Layout for the New Organization Form modal.

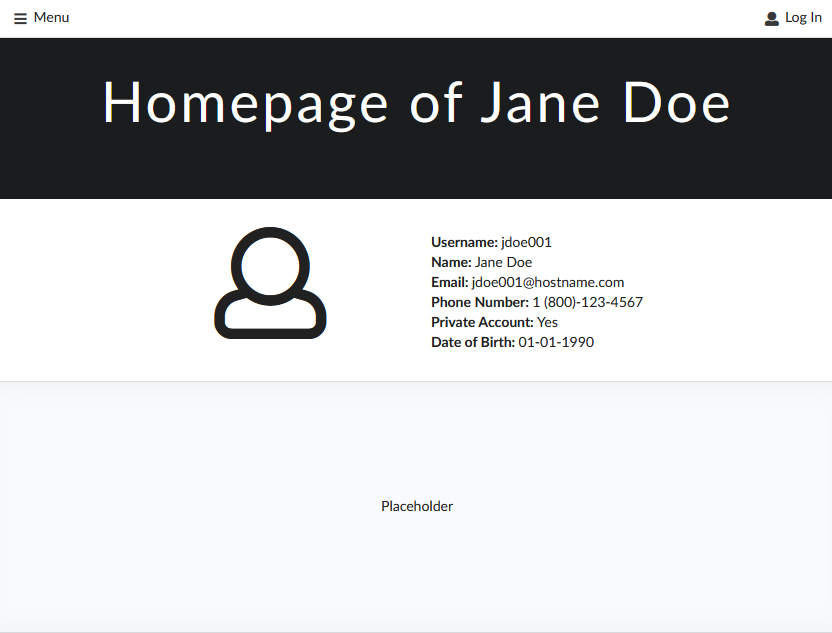


Figure 37: UI layout for the Profile page of a given user, in this case, Jane Doe. When the user is logged in, a edit profile option is given to the user, which triggers a Edit Profile Form modal (see Figure 38). The placeholder section shall include information about the user’s clubs and / or events.

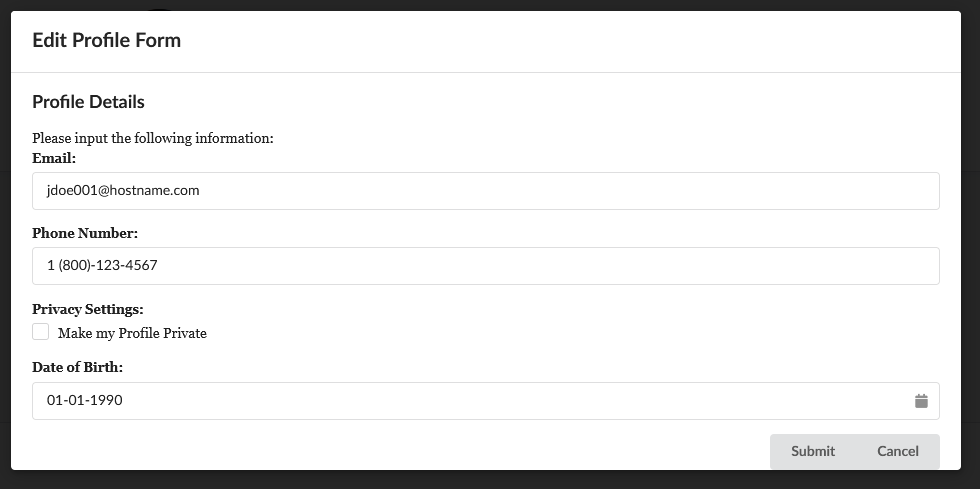


Figure 38: UI Layout for the Edit Profile Form modal

## Appendix D – Detailed Subsystem Class Diagrams

### SOS Website

The full class diagram can be seen in Figure 30.

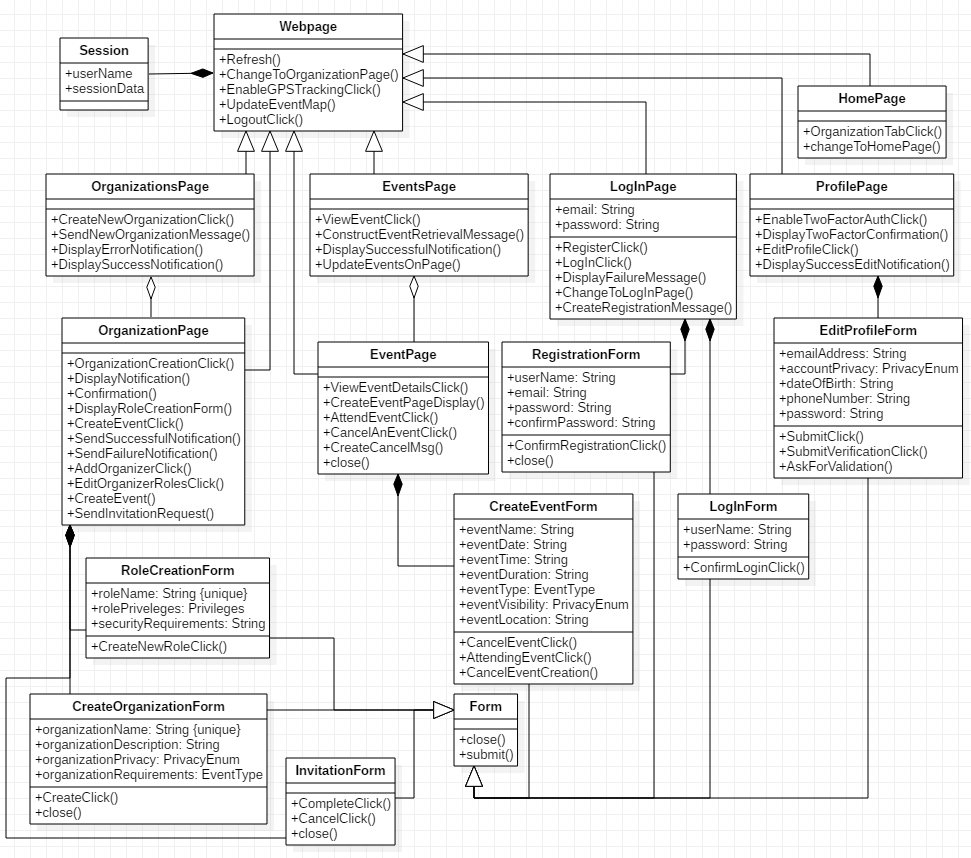


Figure 30: Full Class Diagram for the SOS Website subsystem.

### SOS Interface

The full class diagram can be seen in Figure 31.

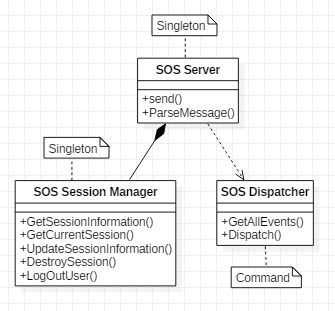


Figure 31: Full Class Diagram for the SOS Interface.

### User Management

The full class diagram can be seen in Figure 32.

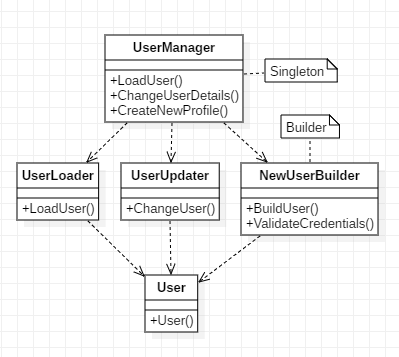


Figure 32: Full Class Diagram for the User Management.

### Event Management

The full class diagram can be seen in Figure 33.

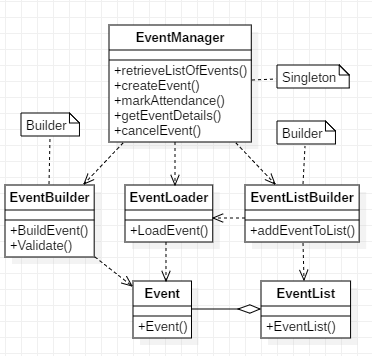


Figure 33: Full Class Diagram for the Event Management.

### Organization Management

The full class diagram can be seen in Figure 34.

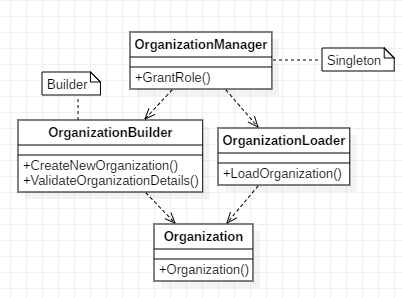


Figure 34: Full Class Diagram for the Organization System.

### Security Management

The full class diagram can be seen in Figure 35.

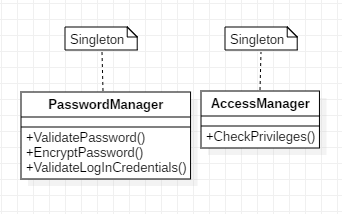


Figure 35: Full Class Diagram for the Security System.

### SOS Storage

The full class diagram can be seen in Figure 36.

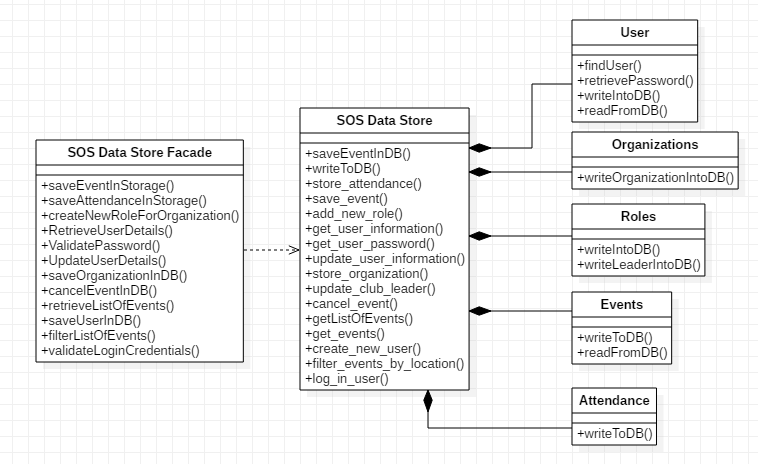


Figure 36: Full Class Diagram for the SOS Storage.

## Appendix E - Class Interfaces

**Package event**

**Class Summary**

[**Event**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark1)

A run-time representation of an Event persistent Object.

[**EventBuilder**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark4)

A Builder class which creates new Events.

[**EventList**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark18)

A class that aggregates Events.

[**EventListBuilder**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark21)

A Builder which creates new EventList objects.

[**EventManager**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark26)

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

**event**

**Class Event**

java.lang.Object

|

+--event.Event

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark2) >

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).

**Constructors**  **Event**

protected **Event**()

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

**event**

**Class EventBuilder**

java.lang.Object

|

+--event.EventBuilder

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark5) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark7) >

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**

**BuildEvent**

public [Event](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark1) **BuildEvent**()

Finalizes the Event creation and returns it.

**Returns:**

returns the final Event.

**Validate**

protected boolean **Validate**()

Checks the current Event, returning True if it is valid so far and False otherwise.

**Returns:**

True, if the Event is valid so far. False if otherwise.

**setDate**

public [EventBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark4) **setDate**(java.lang.String date)

Adds a date value to the current Event.

**Parameters:**

date - the value to be added.

**Returns:**

the current builder.

**setDescription**

public [EventBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark4) **setDescription**(java.lang.String description)

Adds a description value to the current Event.

**Parameters:**

description - the value to be added.

**Returns:**

the current builder.

**setEventtype**

public [EventBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark4) **setEventtype**(java.lang.String eventType)

Adds a Event Type value to the current Event.

**Parameters:**

eventType - the value to be added.

**Returns:**

the current builder.

**setHost**

public [EventBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark4) **setHost**(java.lang.String host)

Adds a host value to the current Event.

**Parameters:**

host - the value to be added.

**Returns:**

the current builder.

**setLocation**

public [EventBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark4) **setLocation**(java.lang.String location)

Adds a location value to the current Event.

**Parameters:**

location - the value to be added.

**Returns:**

the current builder.

**setName**

public [EventBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark4) **setName**(java.lang.String name)

Adds a name value to the current Event.

**Parameters:**

name - the value to be added.

**Returns:**

the current builder.

**setTime**

public [EventBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark4) **setTime**(java.lang.String time)

Adds a time value to the current Event.

**Parameters:**

time - the value to be added.

**Returns:**

the current builder.

**setVisibility**

public [EventBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark4) **setVisibility**(java.lang.String visibility)

Adds a visibility value to the current Event.

**Parameters:**

visibility - the value to be added.

**Returns:**

the current builder.

**event**

**Class EventList**

java.lang.Object

|

+--event.EventList

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark19) >

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.

**event**

**Class EventListBuilder**

java.lang.Object

|

+--event.EventListBuilder

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark22) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark24) >

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**

**setAddeventtolist**

public [EventListBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark21) **setAddeventtolist**(java.lang.String addEventToList)

Adds an Event value to the current EventList.

**Parameters:**

addEventToList - the value to be added.

**Returns:**

the current builder.

**event**

**Class EventManager**

java.lang.Object

|

+--event.EventManager

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark27) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark29) >

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 void **cancelEvent**(int event\_id)

Sets the is\_cancelled property of the given Event to True.

**Parameters:**

event\_id - the wanted Event.

**createEvent**

public [Event](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark1) **createEvent**(java.lang.String jsonString)

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

**Parameters:**

jsonString - the JSON object describing the new Event.

**Returns:**

a Event object with the given attributes.

**getEventDetails**

public [Event](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark1) **getEventDetails**(int event\_id)

Loads an Event with the given Event id.

**Parameters:**

event\_id - the id of the wanted Event.

**Returns:**

the Event with the corresponding id.

**instance**

public static [EventManager](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark26) **instance**()

**Returns:**

The unique instance of this class.

**markAttendance**

public void **markAttendance**(int user\_id,

int event\_id)

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

**Parameters:**

user\_id - the id of the User event\_id - the id of teh Event

**retrieveListOfEvents**

public [EventList](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark18) **retrieveListOfEvents**(int[] event\_ids)

Retrieves a list of Events in the for of an EventList. This is done through an EventListBuilder.

**Parameters:**

event\_ids - the ids of the Events to be added.

**Returns:**

the EventList containing the given Events.

**Package organization**

**Class Summary**

[**Organization**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark37)

A run-time representation of an Organization persistent object.

[**OrganizationBuilder**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark40)

A Builder which creates new Organization objects.

[**OrganizationLoader**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark50)

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

[**OrganizationManager**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark55)

A Singleton which manages all the Organization functions.

**organization**

**Class Organization**

java.lang.Object

|

+--organization.Organization

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark38) >

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)

**Constructors**  **Organization**

protected **Organization**()

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

**organization**

**Class OrganizationBuilder**

java.lang.Object

|

+--organization.OrganizationBuilder

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark41) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark43) >

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.

**Constructors**  **OrganizationBuilder**

public **OrganizationBuilder**()

Creates a new OrganizationBuilder to instantiate the new Event.

**Methods**

**CreateNewOrganization**

public [Organization](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark37) **CreateNewOrganization**()

Finalizes the Organization creation and returns it.

**Returns:**

returns the final Organization.

**ValidateOrganizationDetails**

protected boolean **ValidateOrganizationDetails**()

Checks the current Organization, returning True if it is valid so far and False otherwise.

**Returns:**

True, if the Organization is valid so far. False if otherwise.

**setDescription**

public [OrganizationBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark40) **setDescription**(java.lang.String description)

Adds a description value to the current Organization.

**Parameters:**

description - the value to be added.

**Returns:**

the current builder.

**setName**

public [OrganizationBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark40) **setName**(java.lang.String name)

Adds a name value to the current Organization.

**Parameters:**

name - the value to be added.

**Returns:**

the current builder.

**setPrivacy**

public [OrganizationBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark40) **setPrivacy**(java.lang.String privacy)

Adds a privacy value to the current Organization.

**Parameters:**

privacy - the value to be added.

**Returns:**

the current builder.

**setRequirements**

public [OrganizationBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark40) **setRequirements**(java.lang.String requirements)

Adds a requirements value to the current Organization.

**Parameters:**

requirements - the value to be added.

**Returns:**

the current builder.

**organization**

**Class OrganizationLoader**

java.lang.Object

|

+--organization.OrganizationLoader

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark51) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark53) >

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 static [Organization](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark37) **LoadOrganization**(java.lang.String sqlEntry)

Creates a Organization from a database-format entry.

**Parameters:**

sqlEntry - a sql entry for the given organization.

**Returns:**

a Organization object with the given attributes.

**organization**

**Class OrganizationManager**

java.lang.Object

|

+--organization.OrganizationManager

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark56) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark58) >

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**

**grantRole**

public void **grantRole**(int userId,

int orgId, int[] privIds)

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

**Parameters:**

userId - the unique id of the User

orgId - the unique id of the Organization

privIds - the unique ids of the Privileges given to the User.

**instance**

public static [OrganizationManager](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark55) **instance**()

**Returns:**

The unique instance of this class.

**Package security**

**Class Summary**

[**AccessManager**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark62)

A Singleton dealing with access control actions.

[**PasswordManager**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark68)

A Singleton which deals with password control actions.

**security**

**Class AccessManager**

java.lang.Object

|

+--security.AccessManager

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark63) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark65) >

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**()

**Returns:**

The result of privilege check for the current user class.

**instance**

public static [AccessManager](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark62) **instance**()

**Returns:**

The unique instance of this class.

**security**

**Class PasswordManager**

java.lang.Object

|

+--security.PasswordManager

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark69) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark71) >

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**

**EncryptPassword**

public static java.lang.String **EncryptPassword**(java.lang.String password)

**Parameters:**

password - is a String to be validated

**Returns:**

will return an encrypted version of the password as a String

**ValidateLogInCredentials**

public static boolean **ValidateLogInCredentials**(java.lang.String username,

java.lang.String pwd)

**Parameters:**

username - is the user name for log in 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)

**Parameters:**

password - as a String to be validated

**Returns:**

is true if password successfully validates

**instance**

public static [PasswordManager](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark68) **instance**()

**Returns:**

The unique instance of this class.

**Package sosInterface**

**Class Summary**

[**SOSDispatcher**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark77)

A Command class which propagates the front-end requests to their specific target controllers.

[**SOSServer**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark83)

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

[**SOSSessionManager**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark91)

SOSSessionManager keeps track of each session for every user.

**sosInterface**

**Class SOSDispatcher**

java.lang.Object

|

+--sosInterface.SOSDispatcher

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark78) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark80) >

public class **SOSDispatcher**

extends java.lang.Object

A Command class which propagates the front-end requests to their specific target controllers. The requests messages which are parsed and pre-processed by the SOS Server then are used to call an appropriate dispatch from the SOS Dispatcher, which is in charge of directly calling all other controllers. Instead of being their own classes, each subcommand is defined in terms of parametrizations of the dispatch function within the SOS Dispatcher. Internally, SOS Dispatcher also keeps track of these requests and stores them in the Database

**Constructors**  **SOSDispatcher**

public **SOSDispatcher**()

**Methods**

**Dispatch**

public void **Dispatch**()

The method for dispatching events.

**GetAllEvents**

public java.util.ArrayList **GetAllEvents**()

This method returns all of the events.

**Returns:**

is an ArrayList contain of Strings of events.

**sosInterface**

**Class SOSServer**

java.lang.Object

|

+--sosInterface.SOSServer

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark84) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark86) >

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.

**Constructors**  **SOSServer**

protected **SOSServer**()

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

**Methods**

**CreateEvent**

public static boolean **CreateEvent**(java.lang.String event)

This method creates an event and stores its data.

**Parameters:**

event - is a String coming from the front-end including a JSON which stores all of the event details, including name, type, location, etc.

**Returns:**

is true if event is successfully created and false otherwise.

**ParseMessage**

public static java.lang.String **ParseMessage**(java.lang.String jsonString)

This method parses a message coming from the front-end which supposed to be in JSON format.

**Parameters:**

jsonString - is a String coming from the front-end including the JSON

**Returns:**

is the parsed message

**instance**

public static [SOSServer](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark83) **instance**()

**Returns:**

The unique instance of this class.

**send**

public static void **send**(java.lang.String action,

java.lang.Object payload)

Dispatch an action with the parameters defined in the payload.

**Parameters:**

action - the action to be dispatched. payload - the payload of the action.

**sosInterface**

**Class SOSSessionManager**

java.lang.Object

|

+--sosInterface.SOSSessionManager

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark92) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark94) >

public class **SOSSessionManager**

extends java.lang.Object

SOSSessionManager keeps track of each session for every user. It can gather information about the current session, update the session, or even destroy it.

**Constructors**  **SOSSessionManager**

protected **SOSSessionManager**()

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

**Methods**

**DestroySession**

public static boolean **DestroySession**(java.lang.String sessionID)

This method destroys a given user session

**Parameters:**

sessionID - is the ID for the session to be destroyed

**Returns:**

is true if DestroySession is successful and false otherwise.

**GetCurrentSession**

public static java.lang.String **GetCurrentSession**()

This method returns the information regarding a current live session

**Returns:**

is a String containing current session's ID.

**GetSessionInformation**

public static java.lang.String **GetSessionInformation**(java.lang.String sessionID)

This method returns the information regarding a given session.

**Parameters:**

sessionID - is the ID for the desired session

**Returns:**

is a String containing current session's information.

**LogOutUser**

public static boolean **LogOutUser**()

This method logs out the user from their current session.

**Returns:**

is true if Logout is successful and false otherwise.

**UpdateSessionInformation**

public static boolean **UpdateSessionInformation**(java.lang.String data,

java.lang.String sessionID)

This method updates the given session with the modified data

**Parameters:**

sessionID - is the ID of the session to be updated

data - is the modified data to be updated on the given session

**Returns:**

is true if update information is successfull and false otherwise.

**instance**

public static [SOSSessionManager](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark91) **instance**()

**Returns:**

The unique instance of this class.

**Package user**

**Class Summary**

[**NewUserBuilder**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark102)

A Builder which creates new User objects.

[**User**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark113)

A run-time representation of a User persistent object.

[**UserLoader**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark116)

A class which creates a User object from a database-format User object (e.g., a SQL Table entry for User).

[**UserManager**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark121)

A Singleton class which managers all the User functions.

[**UserUpdater**](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark129)

A class which deals with User modifications.

**user**

**Class NewUserBuilder**

java.lang.Object

|

+--user.NewUserBuilder

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark103) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark105) >

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**

**BuildUser**

public [User](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark113) **BuildUser**()

Finalizes the User creation and returns it.

**Returns:**

returns the final User.

**ValidateCredentials**

protected boolean **ValidateCredentials**()

Checks the current User, returning True if it is valid so far and False otherwise.

**Returns:**

True, if the Event is valid so far. False if otherwise.

**setEmail**

public [NewUserBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark102) **setEmail**(java.lang.String email)

Adds a email value to the current User.

**Parameters:**

email - the value to be added.

**Returns:**

the current builder.

**setName**

public [NewUserBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark102) **setName**(java.lang.String name)

Adds a name value to the current User.

**Parameters:**

name - the value to be added.

**Returns:**

the current builder.

**setPassword**

public [NewUserBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark102) **setPassword**(java.lang.String password)

Adds a password value to the current User.

**Parameters:**

password - the value to be added.

**Returns:**

the current builder.

**setPrivacy**

public [NewUserBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark102) **setPrivacy**(java.lang.String privacy)

Adds a privacy value to the current User.

**Parameters:**

privacy - the value to be added.

**Returns:**

the current builder.

**setUsername**

public [NewUserBuilder](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark102) **setUsername**(java.lang.String username)

Adds a username value to the current User.

**Parameters:**

username - the value to be added.

**Returns:**

the current builder.

**user**

**Class User**

java.lang.Object

|

+--user.User

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark114) >

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).

**Constructors**  **User**

protected **User**()

Constructs a new User class. Called through the UserBuilder class. Attribute assignations are done through protected scope.

**user**

**Class UserLoader**

java.lang.Object

|

+--user.UserLoader

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark117) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark119) >

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**()

**Methods**

**LoadOrganization**

public static [User](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark113) **LoadOrganization**(java.lang.String sqlEntry)

Creates a User from a database-format entry.

**Parameters:**

sqlEntry - a sql entry for the given organization.

**Returns:**

a User object with the given attributes.

**user**

**Class UserManager**

java.lang.Object

|

+--user.UserManager

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark122) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark124) >

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 void **ChangeUserDetails**([User](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark113) user,

java.util.Map change)

Updates a User object with the given changes. Done through the UserUpdater class.

**Parameters:**

user - the User that will be updated.

change - a Map where the key is the variable name and the value is the update.

**CreateNewProfile**

public [User](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark113) **CreateNewProfile**(java.lang.String jsonString)

Creates a new User from a json User description. Done by calling the NewUserBuilder class.

**Parameters:**

jsonString - the JSON object describing the new User.

**Returns:**

a User object with the given attributes.

**LoadUser**

public [User](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark113) **LoadUser**(java.lang.String sqlEntry)

Creates a User from a database-format entry. Done by calling the UserLoader class.

**Parameters:**

sqlEntry - a sql entry for the given user.

**Returns:**

a User object with the given attributes.

**instance**

public static [UserManager](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark121) **instance**()

Gives the instance of the UserManager, or creates one if none exists.

**Returns:**

the unique instance of this class.

**user**

**Class UserUpdater**

java.lang.Object

|

+--user.UserUpdater

< [Constructors](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark130) > < [Methods](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark132) >

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](file:///C:\Users\Arm\AppData\Local\Temp\html-converted-1.docx#_bookmark113) user,

java.util.Map update)

Updates a User object with the given changes.

**Parameters:**

user - the User that will be updated.

change - a Map where the key is the variable name and the value is the update.

## Appendix G – Documented Code for Test Driver

TODO: Add this section.

## Appendix E – Diary of Meetings