Design Document

CEN4010 – Software Engineering

Prepared for Peter Clarke

By Team 5

Teriq Douglas

Yovanni Jones

M. Kian Maroofi

Armando J. Ochoa

Anthony Sanches-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. The SOS is developed using the Unified Software Development Process (USDP), the two first sections of which are contained in this document. The specifications of the system are captured in the form of Use Cases, forming the Use Case model of the USDP. Finally, these Use Cases are used to develop the Analysis model. This is done in the form of Unified Modeling Language diagrams, which describes both static, in the form of Class and Object diagrams, and dynamic, in the form of Sequence Diagrams, views of the system. Add here some lines about motivation for the system.

Table of Contents

[Abstract 2](#_Toc23719572)

[1 Introduction 5](#_Toc23719573)

[1.1 Purpose of the System 5](#_Toc23719574)

[1.2 Requirements 5](#_Toc23719575)

[1.2.1 Functional Requirements 5](#_Toc23719576)

[1.2.2 Non-Functional Requirements 6](#_Toc23719577)

[1.2.2.1 Usability 6](#_Toc23719578)

[1.2.2.2 Reliability 6](#_Toc23719579)

[1.2.2.3 Performance 6](#_Toc23719580)

[1.2.2.4 Supportability 6](#_Toc23719581)

[1.2.2.5 Implementation 6](#_Toc23719582)

[1.3 Development Methodology 6](#_Toc23719583)

[1.4 Definitions, Acronyms, and Abbreviations 7](#_Toc23719584)

[1.5 Overview of the Document 8](#_Toc23719585)

[2 Proposed Software Architecture 9](#_Toc23719586)

[2.1 Overview 9](#_Toc23719587)

[2.2 Subsystem Decomposition. 10](#_Toc23719588)

[2.3 Hardware and Software Mapping 10](#_Toc23719589)

[2.4 Persistent Data Management 10](#_Toc23719590)

[2.5 Security Management 10](#_Toc23719591)

[3 Detailed Design 11](#_Toc23719592)

[3.1 Overview 11](#_Toc23719593)

[3.2 State Machine 11](#_Toc23719594)

[3.3 Object Interaction 11](#_Toc23719595)

[3.4 Detailed Class Design 11](#_Toc23719596)

[3.4.1 Class Description 11](#_Toc23719597)

[3.4.2 Control Objects Description 11](#_Toc23719598)

[4 Glossary 12](#_Toc23719599)

[5 Approval Page: 13](#_Toc23719600)

[6 References 14](#_Toc23719601)

[7 Appendices 15](#_Toc23719602)

[7.1 Appendix A – Use Case Diagram 15](#_Toc23719603)

[7.2 Appendix B – Implemented Use Cases 16](#_Toc23719604)

[7.2.1 Create Event 16](#_Toc23719605)

[7.2.2 Grant Organizer Role 16](#_Toc23719606)

[7.2.3 Attending an Event 16](#_Toc23719607)

[7.2.4 Edit Profile 16](#_Toc23719608)

[7.2.5 Access Events by Location 16](#_Toc23719609)

[7.2.6 Create Organization 16](#_Toc23719610)

[7.2.7 Cancel an Event 16](#_Toc23719611)

[7.2.8 Registration 16](#_Toc23719612)

[7.2.9 Log in 16](#_Toc23719613)

[7.2.10 Log Out 16](#_Toc23719614)

[7.3 Appendix C – Detailed Subsystem Class Diagrams 18](#_Toc23719615)

[7.4 Appendix D - Class Interfaces 19](#_Toc23719616)

[7.5 Appendix E – Diary of Meetings 20](#_Toc23719617)

# Introduction

The following chapter introduces the Software Requirements Document (SRD) with the main goal of explaining the ideas and concepts behind the Student Organization System (SOS) project.

The purpose of this System Requirements Document (SRD) is to define the requirements of the SOS system, and to act as a basis for a more detailed Design Document (DD). These requirements, compiled in the form of Use Cases, describe the interactions between the potential users and the system. Moreover, they describe the system holistically, with requirements that apply both to the client- and to server-side system operations.

The purpose of the SOS is defined bellow. 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 including a Use Case model of the planned system (Section 3), an Analysis model (Section 4), and a detailed section on project management (Section 2).

## 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.).

## Requirements

This section defined the functional and non-functional requirements of the SOS system. A more complete and detailed description of the system requirements can be found in the Software Requirements Document (SRD) for this project.

### 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 short description 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 can be found in each Use Case in Appendix B.

#### Usability

Add something here.

#### Reliability

Add something here.

#### Performance

Add something here.

#### 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.

## 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.



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

This document contains the third model, the use desing model in Chapters 2 and 3. 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 and later Sections 3.1 contain the detailed designs of each of the subsystems in the form of simplified UML Class Diagrams. The full UML Class Diagrams for the subsystems are contained in Appendix C. A simplified version of the fourth one, 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 system. The design and deployment models should provide a detailed description of the system structure without reling 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 further 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

Add stuff here.

# Proposed Software Architecture

The following sections contain a top-level description of the architecture of the Student Organization System (SOS), including subsystems decomposition, as well as data management and security requirements. Section 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

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. The 3TA was chosen because it allows the SOS system to be divided into interchangeable layers which can be updated and maintained separately as long as their interfaces are respected. 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, a number of subsystems access and modify data from a single data structure (a repository) which mediates their interaction. This architecture is used in the third layer, the storage. 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 in their interaction with 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:

* Data Store, 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: Add the actual use cases here**.**
* 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 Controller, 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).
* 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).
* 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

Deployment Diagram in SOS.mdj.

## Persistent Data Management

Add Stuff Here.

## Security Management

## Add Stuff Here.

# Detailed Design

Add something here.

## Overview

Add something here.

## State Machine

Add something here.

## Object Interaction

Add something here.

## Detailed Class Design

### Class Description

Add something here. Most of them done in SOS.mdj

### Control Objects Description

Add something here.

# 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 10/01/2019

Member Signature Date

Yovanni Jones 10/01/2019

Member Signature Date

M.Kian Maroofi 10/01/2019

Member Signature Date

Teriq Douglas 10/01/2019

Member Signature Date

Anthony Sanchez-Ayra 10/01/2019

Member Signature Date

# References

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

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

# Appendices

## Appendix A – Use Case Diagram

The Use Case Diagram for the Student Organization System is contained in Figure 11.



Figure 11: Use case diagram for the implemented Use Cases.

## Appendix B – Implemented Use Cases

### 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

### 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

### 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

### 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

### 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

### 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

### 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 – Detailed Subsystem Class Diagrams

## Appendix D - Class Interfaces

## Appendix E – Diary of Meetings