CSC 431

UPlan

System Architecture Specification (SAS)

Group 5

Dana Feeney	Developer
Mariana Baquero Degwitz	Scrum Master
Jonathan Raskauskas	Developer

Version History

Version	Date	Author(s)	Change Comments
1.0.0	4/1/2021	Dana Feeney Mariana Baquero Jonathan Raskauskas	Initial Draft
2.0.0	4/13/2021	Dana Feeney Mariana Baquero Jonathan Raskauskas	Updated Draft

Table of Contents

1. System Analysis 6
  1.1 System Overview 6
  1.2 System Diagram 6
  1.3 Actor Identification 6
  1.4 Design Rationale 7
    1.4.1 Architectural Style 7
    1.4.2 Design Pattern(s) 8
    1.4.3 Framework 8
2. Functional Design 9
  2.1 Functionality DIagram 9
3. Structural Design 10

Table of Tables

Table of Figures

1. System Analysis 6
   1.2 System Diagram 6
2. Functional Design 9
   2.1 Functionality Diagram 9
3. Structural Design 10
   3.1 Class Diagram 10

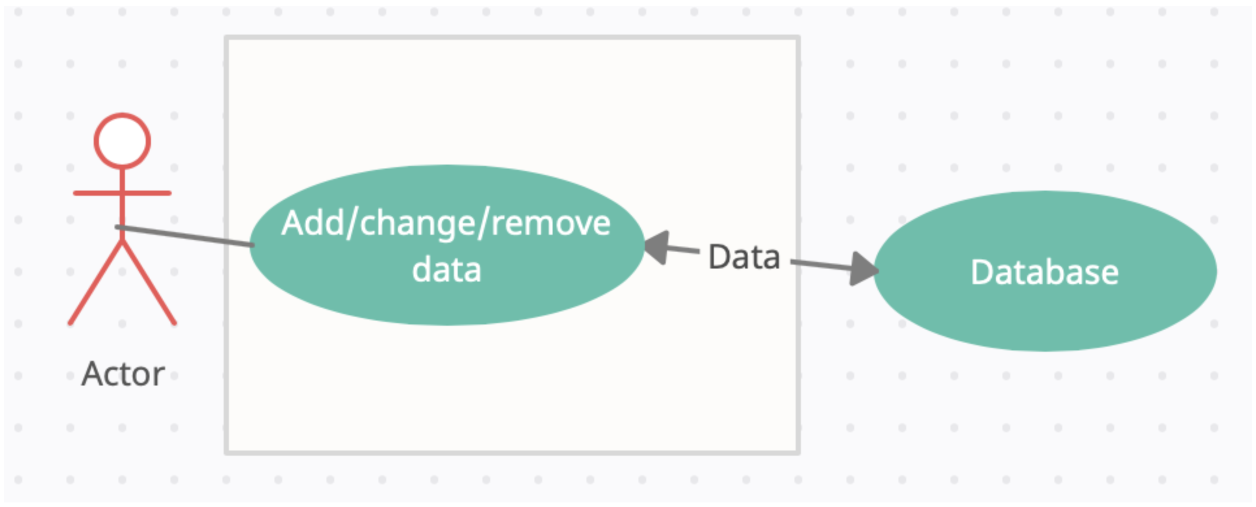
1. System Analysis

1.1 System Overview

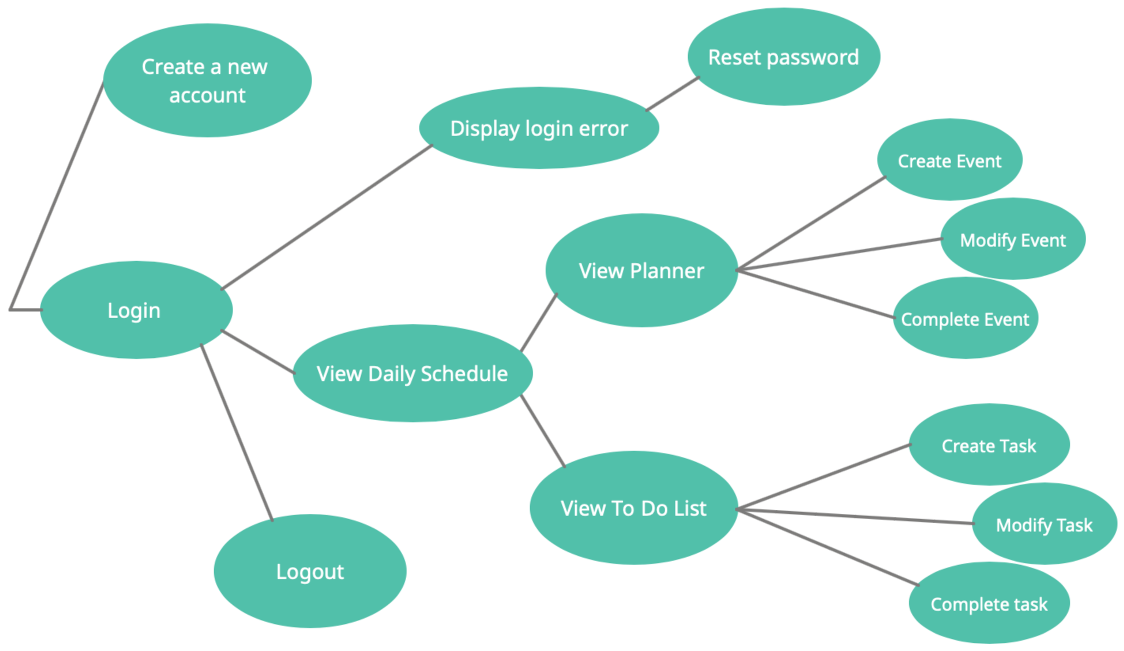
Our System will be composed of the website, which will be hosted through github, the database which will store all of the necessary system and user data, and the user. Each component will interact with each other and have a general flow to them that progresses from the landing page to the more minute details of the application. The main external factor that influences the system will be the user's ability to use the interface to store information by inputting through creating events (calendar events) and tasks (to-do events).

Within the database, each user's login will be stored and encrypted. All of the user data relative to each account will be stored in the database related to the user's login. There will be a table for the calendar and a separate table for the to-do list because the events and tasks have different attributes. To send and access data from the database the system will use PHP and mySQL to make requests and processes responses from the server. Because of the nature of the tasks and events and their respective attributes, there isn't much information that users can store within this application that will not already be visible upon loading the application. The only processes relative to the server that will need to happen during the usage of the app will be adding or deleting events and then updating the calendar or to-do list to include the new or removed data. As of now there will be no need for a data manager because checks within the forms built to add data will maintain the integrity of the information entered into the database. If the app were to grow beyond a small startup size then a data manager would be an ideal solution to help maintain the application.

1.2 System Diagram



Use Case Diagram



1.3 Actor Identification

An actor specifies a role played by a user or another element that interacts with the system. There are three types of actors that can appear outside of a system: User (human), System (external system), and Time (system clock). In our project, we have the user, which in this case is the students of University of Miami. The user will create the start of the external events that trigger the rest of our system. Another actor is a system which is MySql which will allow the website to grab data. Also, in our project, there is a time actor, which will allow a notification trigger when the time is appropriate.

1.4 Design Rationale

1.4.1 Architectural Style

Our architecture is event driven. We have multiple triggers as the events. If the user of the creates a calendar event or to-do task, that is the external event that is the trigger for our system to add the task to their daily view and to trigger the notification when the time is appropriate. We have based our system on this architecture choice.

1.4.2 Design Pattern(s)

We will be using the following design patterns to complete our project:

Factory Method: This fits our overall design of the system because the event class that we are creating does not have to be known ahead of time, for the external factor (the user) is the one who actually chooses the class. Both of the two main events that the user can choose (event or task) fall under a central class, which we can call the external event.

Decorator: The decorator design pattern fits our project because it adds functionality to an existing component without modifying its source code. This will allow us to add the same functionality to the calendar view and task view without having to modify existing source code.

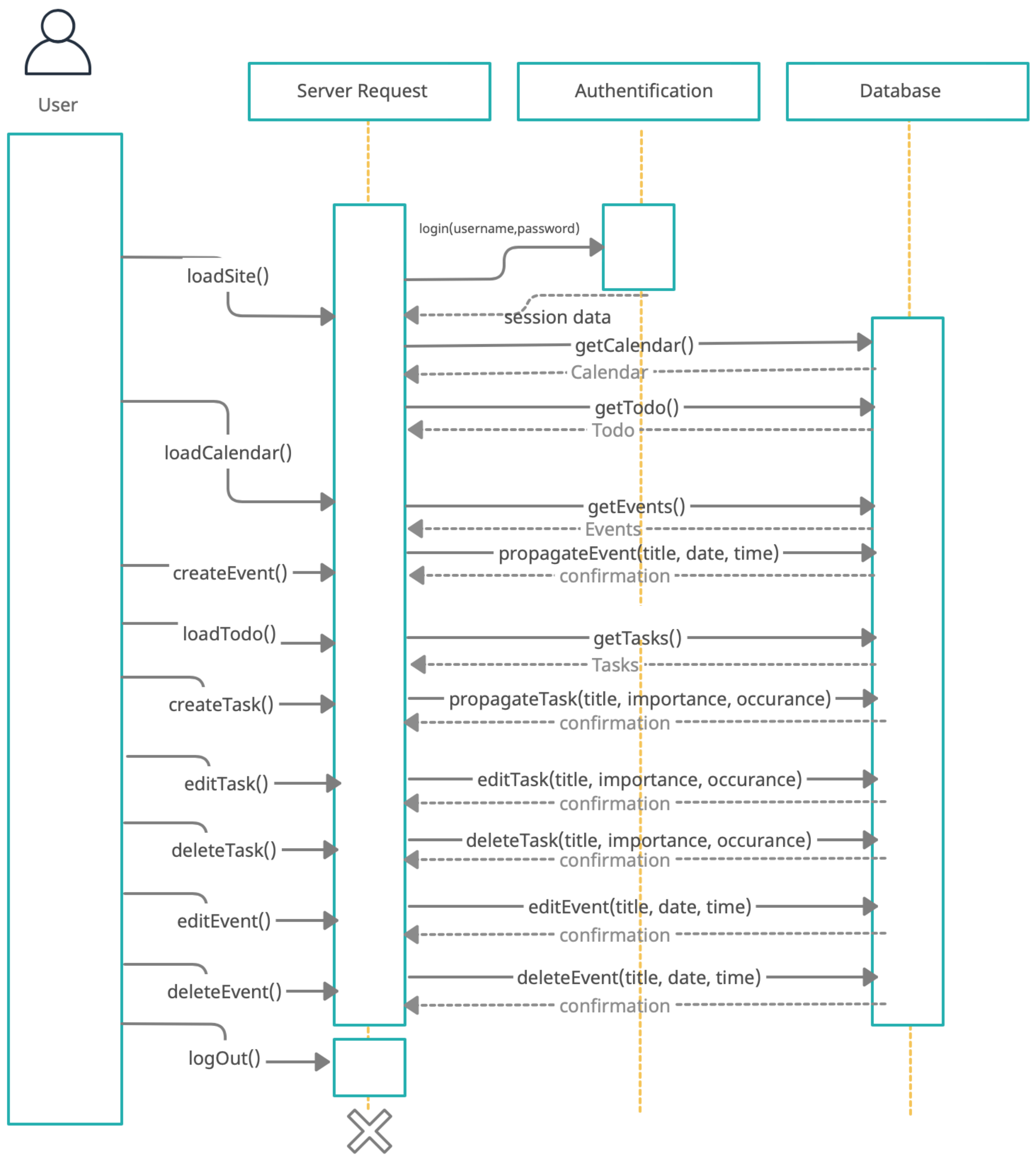
1.4.3 Framework

For our framework, we will be using Bootstrap, which is an open source CSS framework. Bootstrap will allow us to work with PHP and MySql which is necessary for managing the data such as usernames, passwords, events and to-do tasks. Since our project is web-based, Bootstrap provides an efficient framework for responsive websites.

Link: https://getbootstrap.com/

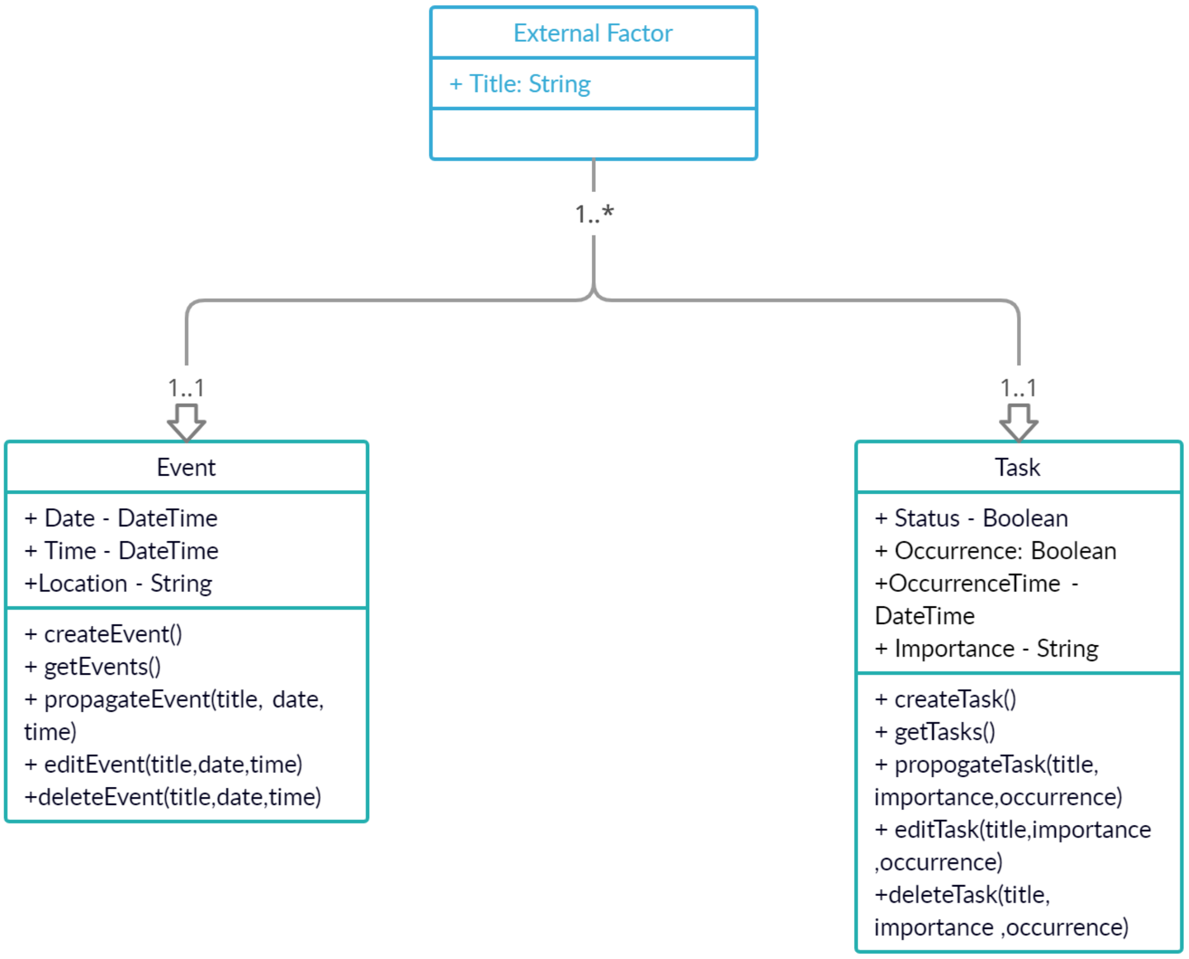
2. Functional Design

2.1 Functionality Diagram



3. Structural Design

3.1 Class Diagram


External Factor
+ Title: String
1..*
1..1
1..1
Event
+ Date - DateTime
+ Time - DateTime
+Location - String
+ createEvent()
+ getEvents()
+ propagateEvent(title, date, time)
+ editEvent(title,date,time)
+deleteEvent(title,date,time)
Task
+ Status - Boolean
+ Occurrence: Boolean
+OccurrenceTime - DateTime
+ Importance - String
+ createTask()
+ getTasks()
+ propogateTask(title, importance,occurrence)
+ editTask(title,importance ,occurrence)
+deleteTask(title, importance ,occurrence)