**Georgia Southern ACM Club Application**

Software Design Document

CSCI 5530

Software Engineering

Team 6

Griffin Bryant, Marquise Jennett, Dylan Myers, Mansi Pandya, Everrick Wright

**Georgia Southern University ACM Club Application**

ABSTRACT

The Georgia Southern’s Association for Computer Machinery (ACM) club connects student members with professional development tools. ACM traditionally had to utilize multiple applications and devices to relay information pertaining to the club to its members. To counter the problem of utilizing multiple applications and devices, the ACM club application will be a consolidated tool that will be primarily a communication application, but include useful extensions.

The following information in this software design document will outline the plans of building the previously stated ACM application. This will start with an introduction to the functional and non-functional requirements of the system and the design methodology. Then, the proposed software architecture will acquaint the client with subsystem decomposition, data management, and security management. Finally, the object design chapter will outline relevant diagrams: ER diagram, Sequence diagram, Class diagram. This section will also include how the systems object interaction will be implemented.

**Table of Contents**

[**Chapter 1 – Introduction**](#_f32b5jk9rehu) **5**

[**1.1 Purpose of System**](#_h47x0dvz28fp) **5**

[**1.2 Functional and Nonfunctional Requirements**](#_ifqf8fhdeemv) **5**

[1.2.1 – Functional](#_miq9t0l3b5hc) 5

[1.2.2 – Non Functional](#_p37qi7ay8z9m) 6

[**1.3 Design Methodology**](#_a1wt54ky7jh) **7**

[**1.4 Definitions, Acronyms, and Abbreviations**](#_c6e5qwf6qysz) **7**

[**1.5 Overview**](#_el6jzjx6mfoz) **8**

[**Chapter 2 – Proposed Software Architecture**](#_k90e74klwyc2) **9**

[**2.1 Overview**](#_4pit0ow56l8r) **9**

[**2.2 Subsystem Decomposition**](#_m89ir3n15ipz) **9**

[**2.3 Hardware and Software Mapping**](#_429w2kt2mini) **10**

[**2.4 Persistent Data Management**](#_ejoyvl9jc3bv) **11**

[**2.5 Security Management**](#_5cs1izx8il81) **11**

[**Chapter 3 – Object Design**](#_g2q75ifog3e1) **13**

[**3.1 Overview**](#_tqda6d591yas) **13**

[**3.2 State Machine**](#_vh28v3j4rnlo) **14**

[**3.3 Object Interaction**](#_2o1iuot0tebl) **14**

[3.4 Detailed Class Design](#_1nkis2hgkjk) 15

[3.4.1](#_l8gv6hzdwi7t) 15

[**Chapter 4 – Appendix**](#_9ajr7yg1o9bs) **16**

[**4.1 Appendix A – Use Case Diagram**](#_p0i4fxuiztc4) **16**

[**4.2 Appendix B – Use Cases**](#_jfkb8dy7aowf) **17**

[Case number: 1.0](#_it5y5swgs5jx) 17

[Case number: 1.1](#_8qbdgllp9e0j) 18

[Case number: 1.2](#_797mwmjc039e) 19

[Case number: 1.3](#_rrqba1fx3i44) 20

[Case number: 1.4](#_erq762k7rcct) 21

[Case number: 1.5](#_1h8uqm724fpm) 22

[Case number: 1.6](#_opltf4p6jwoh) 23

[Case number: 1.7](#_3rxbvovmu1xj) 24

[Case number: 1.8](#_2fgbaa44dg4y) 25

[Case number: 1.9](#_dnfuyylbhna3) 26

[Case number: 2.0](#_an3qoapnf6ws) 27

[Case number: 2.1](#_hl364jrzxa9m) 28

[Case number: 2.2](#_mobk15ug8baq) 29

[Case number: 2.3](#_7hpszjca0v8z) 30

[Case number: 2.4](#_aiatecwony7g) 31

[Case number: 2.5](#_rzs8xdgcllrf) 32

[5.2.19 – Requirement 19](#_lw2oosixfk9) 32

[5.2.20 – Requirement 20](#_hpmbfasi556) 33

[5.2.22 – Requirement 22](#_jjozgi620f06) 34

[5.2.23 – Requirement 23](#_i1lk2pr90mql) 35

[Case Number 3.3](#_2ofavau6jrmi) 36

[Case Number 3.4](#_hbryntswsvpw) 37

[Case Number 3.5](#_5r7dh7f7mqs1) 38

[Case Number: 3.6](#_9u0drxrwjzsd) 39

[**4.3 Appendix C – Detailed Class Diagrams**](#_meudt479nrxb) **40**

[**4.4 Appendix D – Detailed ER Diagrams**](#_gy5g361h2mhx) **40**

[**4.5 Appendix E – Log of Meeting and Tasks**](#_rc4isvj39v77) **41**

[Week of 16 February 2020](#_uzhw0jjleag2) 41

[Week of 22 February 2020](#_m2cm5tc97rlk) 42

[Week of 29 February 2020](#_83mocybydidn) 42

[Week of 7 March 2020](#_kiw02w44xqt6) 42

# Chapter 1 – Introduction

## 1.1 Purpose of System

The ACM application will be a mobile application to provide a platform for communication and resources for ACM members and computer science students.

## 1.2 Functional and Nonfunctional Requirements

### 1.2.1 – Functional

* The system shall have the ability to log onto the application
  + (See Case Number 1.0 In Appendix B)
* The system database shall store titles of board members
  + (See Case Number 1.1 In Appendix B)
* The system shall set eboard members as club administrators
  + (See Case Number 1.2 In Appendix B)
* The system shall allow users to view events
  + (See Case Number 1.3 In Appendix B)
* The system shall allow users to open event locations in their default map application
  + (See Case Number 1.4 In Appendix B)
* The system shall allow users to request a mentor
  + (See Case Number 1.5 In Appendix B)
* The system shall allow the tutor to be matched by things like class experience
  + (See Case Number 1.6 In Appendix B)
* The application shall be able to send notifications for meetings and events via email or push notification
  + (See Case Number 1.8 In Appendix B)
* The system shall prevent messages containing profanity from being sent to in the chat
  + (See Case Number 1.7 In Appendix B)
* Users shall be able to message other users directly
  + (See Case Number 1.9 In Appendix B)
* Users shall be able to send messages inside of groups with set topics
  + (See Case Number 2.0 In Appendix B)
* Non-members shall be able see events and create an account
  + (See Case Number 2.1 In Appendix B)
* Users shall have the ability to create a profile
  + (See Case Number 2.2 In Appendix B)
* The system should allow users to see a list of known ACM GSU alumni
  + (See Case Number 2.3 In Appendix B)
* The system shall have the ability to keep a tally of the attendance at each meeting
  + (See Case Number 2.4 In Appendix B)
* The system shall allow a user to be removed from the ACM app after deferring to ACM constitution
  + (See Case Number 2.5 In Appendix B)
* The system shall allow administrators to create anonymous voting polls for the executive board
  + (See Case Number 2.6 In Appendix B)

### 1.2.2 – Non Functional

* The system should allow users to get in contact with possible employers for internships, co-ops, and jobs
  + (See Case Number 2.7 In Appendix B)
* The system should allow users to see a list of known ACM GSU employers
  + (See Case Number 2.8 In Appendix B)
* The system may have the ability for administrators to keep a tally of the attendance at each meeting
  + (See Case Number 2.9 In Appendix B)
* The system may allow administrators to accept electronic payments via paypal or venmo
  + (See Case Number 3.0 In Appendix B)
* The system may allow administrators to see a list of members who have paid their dues
  + (See Case Number 3.1 In Appendix B)
* The system may allow administrators to create anonymous voting polls for the executive board
  + (See Case Number 3.2 In Appendix B)

## 1.3 Design Methodology

The ACM application development will be based on the Agile Software Development Lifecycle. The Agile Lifecycle was chosen because it advocates adaptive planning, evolutionary development, quick delivery, and continual improvement. We will follow the cycle of meeting, planning, designing, developing, testing, and evaluating. The whole process will be driven by the use cases that we designed (see 5.2 – Appendix B) during our planning phase and illustrated in our Software Requirements and Specifications document (SRS). This document addresses the design model, the deployment model, and the implementation models. Our design models are created in Chapter 2 and Chapter 3. The models created are Sequence Diagrams, Class Diagrams, and State Charts. Our design models serve as input into the deployment models. From these diagrams and the Agile approach, section 2.3 discusses the mapping of these models to the hardware of the platform. Once we understand how to map the models to our hardware, we can use the deployment model as a metamodel for the implementation model which produces code.

## 1.4 Definitions, Acronyms, and Abbreviations

***ACM*** - Association for Computing Machinery

***MySQL*** - The database management system

***AWS*** - Amazon Web Services

***MVVM*** - Model-View-ViewMode

## 1.5 Overview

In subsequent chapters a detailed understanding of the ACM application will be obtained. In Chapter 2, Proposed Software Architecture, the architecture and accompanying UML profiles of the ACM Application will be examined. The includes a discussion of the generative architecture used to produce code in Section 2.4. In Chapter 3, Object Design, a detailed overview of the structure of the ACM application will be discussed along with important interactions needed for the correct developpment of the system. This chapter will outline the class structure of several objects needed for the development phase. In chapter 4, Glossary, several terms used throughout this design document are defined. This chapter will be useful when an obscure term is encountered so that the reader can thoroughly understand the material. In Chapter 5, Appendix, additional supplementary information can be found. This section contains use case diagrams, use cases, detailed class diagrams, code, and a log of group meetings. By the end of this document, the reader will have an understanding of the design and possible implementation of the ACM Application.

# Chapter 2 – Proposed Software Architecture

The software architecture is arguably the most important design decision of any software system or component. It is essentially the decision that could make or break the project, guide future design decisions, and provides an overall structural theme that will remain until the "shelf life" of the system expires. Because of the importance of software design architecture, it is vital to design and understand the system correctly. Failure to achieve this goal may lead to the complete failure of the software financially and programatically.

To achieve the goal of understanding the ACM Application design thoroughly and correctly, this chapter strives to address all of the design considerations completely. In section 2.1 an overview of the entire proposed system architecture is discussed. In Section 2.2, Subsystem Decomposition, we will describe each of the major subsystems and identify the use cases associated with each subsystem. In Section 2.3, Hardware and Software Mapping, we will explain how the subsystems will be mapped between hardware and software. In Section 2.4, Persistent Data Management, we will identify the data that we need to store and the structure we will use to store it. In Section 2.5, Security Management, we will describe the security protocols that we plan to use in our design.

## 2.1 Overview

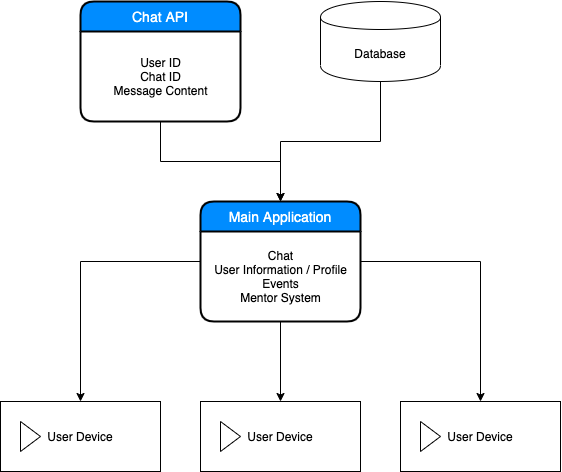
Team 6 will use a Model-View-ViewModel (MVVM) pattern. MVVMs are used when you need flexible design work, when unit testing is required, when it is important to have reusable components, and when development work needs to be simultaneous.

Team 6 also plans to implement the three-tier style architecture. This style includes a presentation tier, an application tier and a data tier. Utilizing this tier, we can improve performance, availability, and scalability.

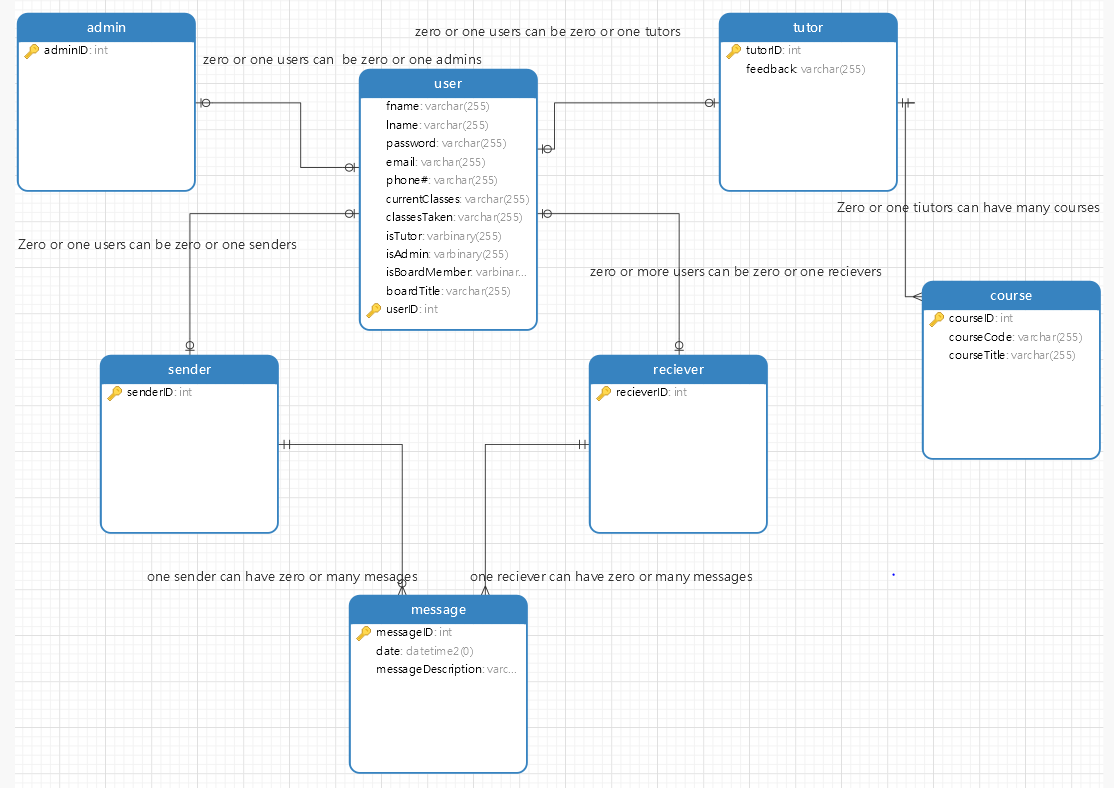
## 2.2 Subsystem Decomposition

The ACM Application is decomposed into four packages. The packages will be the chat communication API, the mentor system, the main application, and the database. Users will log into the main application which will access the database for most of its information including user login and profile information, event information, and other ACM club information. The database subsystem will store the actual information that is accessed from the main application. The mentoring subsystem will allow users to request a mentor with experience in a specific subject to help them with that subject. The chat subsystem will use an API to allow users to send messages directly to each other as well as to a group with multiple users.

## 2.3 Hardware and Software Mapping

**

## 2.4 Persistent Data Management



## 2.5 Security Management

Rivest-Shamir-Adleman (RSA) will be used for password encryption in our application. RSA is an algorithm asymmetric cryptographic algorithm. Asymmetric meaning there are two different keys; one of which will be a public key and given to anyone. The other key must be private. Implementing a private and public key generator and prime factorization passwords will be difficult to break.

A backup of a database can be done in 4 ways: Full backup, incremental backup, differential backup, and mirror backup. The most optimal for our system will be a mirror backup in which it does not store old obsolete files after being deleted in the main database. The downside of downside back ups is if files are accidently deleted, they can be lost from the backup as well if not discovered before the next scheduled backup.

Bugs and vulnerabilities in a code are starting points for many errors in applications. Encapsulation and minification of code to reduce the chance code can be reversed engineered, testing repeatedly, and clear and easily updatable/patchable code will be implemented in development.

# 

# Chapter 3 – Object Design

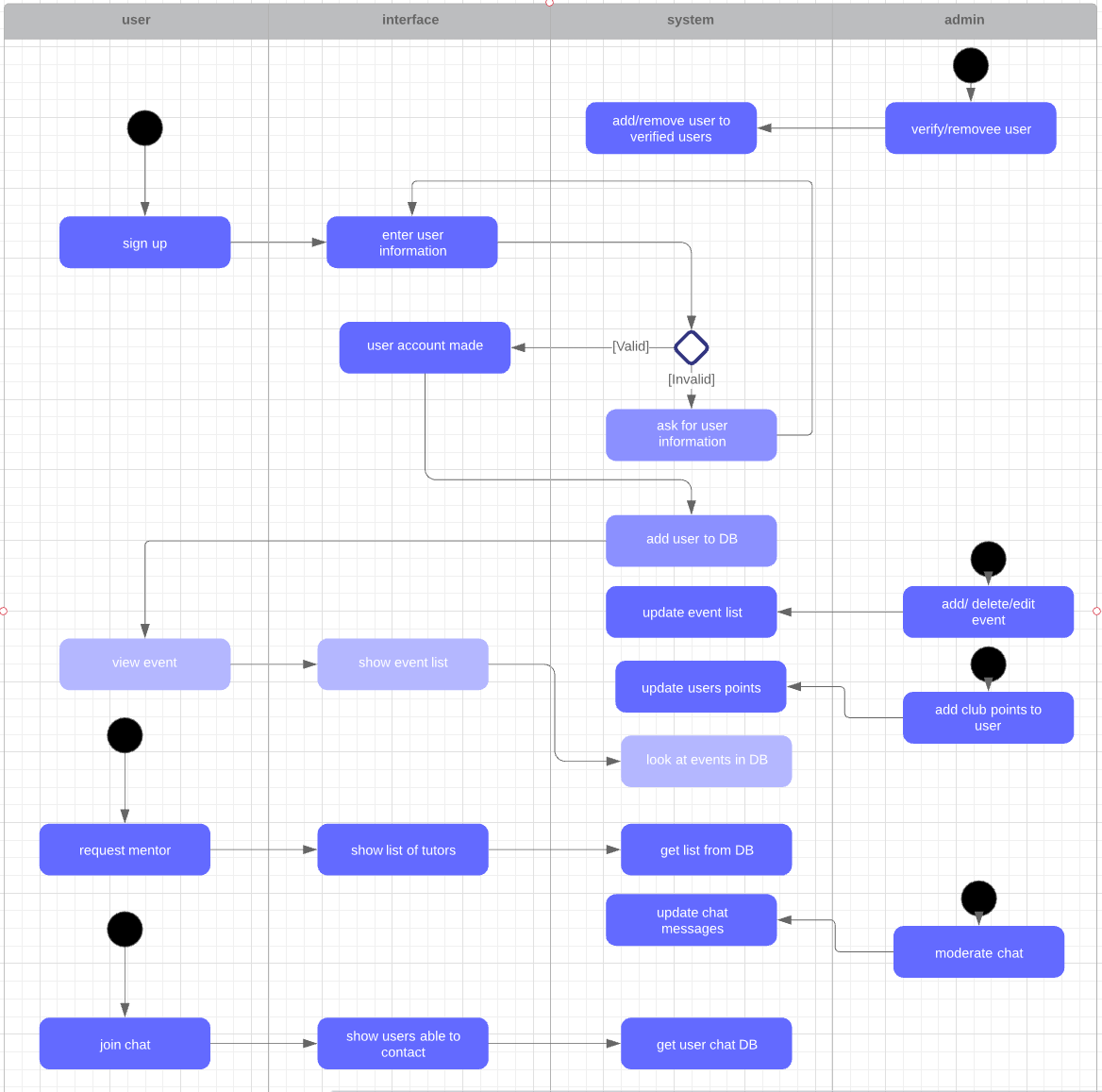
The design and structure of objects, like the system architecture, is important. A proper design of objects can prevent software rot, improve ease of maintenance, improve software efficiency, and improve overall robustness. To achieve this "holy-grail" of design, designers must document the design of objects. To properly document the object design, designers tend to rely on several UML diagramming techniques, such as class diagrams, sequence diagrams, and state charts.

In this chapter, several object design diagrams are discussed for the ACM application. In Section 3.1, an overview of the structural relationships between classes is discussed with an accompanying minimal class diagram. In Section 3.2, State Machine, we will display a state machine for the overall system and the main control object in each major subsystem. In Section 3.3, Object Interaction, we will include more refined sequence diagrams to display object interactions. In Section 3.4, Detailed Class Design, we will explain the purpose of each class and refer to the appropriate class diagram in Appendix C.

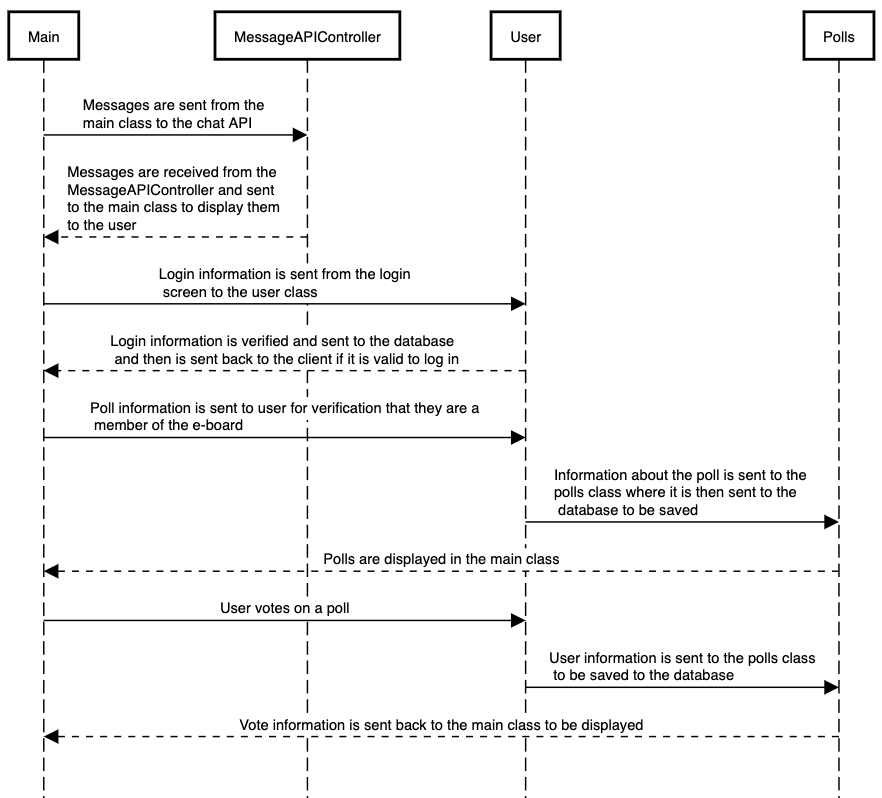
## 3.1 Overview

## 

## 3.2 State Machine



## 3.3 Object Interaction



## 3.4 Detailed Class Design

### 3.4.1

The singleton design pattern is used to limit creation of a class to only one object. This is beneficial for our application in which only one user should be allowed to log into a device. The pattern also provides a way to access its only object which can be accessed directly without needing to instantiate the object of the class.

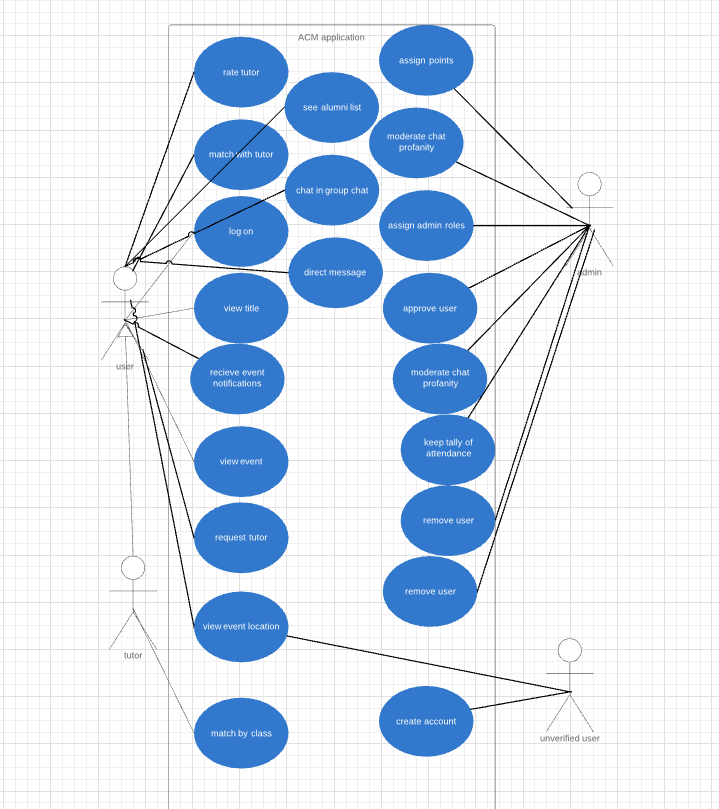
The factory method design pattern is a creational pattern that uses methods to deal with the problem of creating objects without having to specify the exact class of the object that will be created. Objects are created by calling a factory method and are either specified in an interface and implemented by child classes or implemented in a base class and overridden by derived classes.

The observer design pattern is closely related to the model view view controller pattern in which the view part of the MVVC is like the observer pattern. There is a subject that holds all of the data and the state of the data, then, you have observers (users) that will pull the data from the subject.This pattern can be used for updating, notifications, and even handling subscribers.

The strategy pattern is an advanced version of an if else statement. An interface is implemented for a base class and then used to find the right use of a method from a derived class. The right use, or implementation, will be decided at runtime.

# Chapter 4 – Appendix

## 4.1 Appendix A – Use Case Diagram



## 4.2 Appendix B – Use Cases

### **Case number:** 1.0

**Title (Goal):** The system shall have the ability to log onto the application

**Description:** Board member signs up and waits for approval for full usability of application

**Primary Actors:** User, Database system

**Main:** *Pre-condition: User must be a verified board member via the eboard*

1. User clicks ‘sign up’ page

2. The application sends user application to database

3. Eboard member logs on

4. Eboard member clicks ‘member verification’

5. Selects users to approve

6. Board member now has access to full application

Post Condition: None

### **Case number:** 1.1

**Title (Goal):** The system database shall store titles of e-board members

**Description:** Board member signs up

**Primary Actors:** User, Eboard member, Database system

**Main:** *Pre-condition: User must be a verified eboard member via the president*

1. User clicks ‘sign up’ page

2. The application asks if the user is a board member

3. The database checks if the given user is approved by the President

4. After approval, user profile now has attached title

Post Condition: None

### 

### **Case number:** 1.2

**Title (Goal):** The system shall set eboard members as club administrators

**Description:** Admins will change a boolean in database to allow eboard to have limited administrative abilities

**Primary Actors:** Admin, User, Database system

**Main:** *Pre-condition: User must be a verified eboard member via the admin*

1. Administrator logs in

2. The administrator finds user needing club admin status

3. The admin updates user to club user via isClubAdmin checkbox

4. Database updates

5. User now has club admin capabilities

Post Condition: None

### **Case number:** 1.3

**Title (Goal):** The system shall allow users to view events

**Description:**

**Primary Actors:** Admin, User, Database system

**Main:** *Pre-condition: User must be a verified board member via the admin*

1. Administrator logs in

2. The administrator finds user needing club admin status

3. The admin updates user to club user via isClubAdmin checkbox

4. Database updates

5. User now has club admin capabilities

Post Condition: None

### 

### **Case number:** 1.4

**Title (Goal):** The system shall allow users to open event locations in their default map application

**Description:** After user selects event to attend, the option of opening event location in default map application appears

**Primary Actors:** User, Database system

**Main:** *Pre-condition: User must be logged into application*

1. User logs in

2. User clicks on ‘events’ tab

3. User selects event wanting to attend

4. Notification appears stating “Open Event Location in Maps?”

5. User clicks ‘yes’

6. Default map application opens with location of event

Post Condition: None

### **Case number:** 1.5

**Title (Goal):** The system shall allow users to request a mentor

**Description:** User can request a mentor for a class subject

**Primary Actors:** User, Database system

**Main:** *Pre-condition: User must be logged into application and mentor must be approved tutor*

1. User logs in

2. User clicks on ‘mentor’ tab

3. User selects class they need mentoring in

4. User selects request tutor

5. User request/profile is sent to database to await matching of tutor

6. Notification is sent to eboard member handling matching of tutors

Post Condition: Go to Case #1.7

### 

### **Case number:** 1.6

**Title (Goal):** The system shall allow the tutor to be matched by things like class experience

**Description:** Tutor is matched with requested user for specific class experience

**Primary Actors:** User, Database system

**Main:** *Pre-condition: A request must have been sent for a tutor*

1. Club admin logs in

2. Club admin clicks on ‘mentor requests’ tab

3. Club admin selects mentor request application

4. Club admin selects users in application with specific class experience requested

5. Club admin approves tutor

6. Tutor receives mentor link message

7. Tutor approves link request

8. User receives notification that he/she is matched with tutor

Post Condition: None

### **Case number:** 1.7

**Title (Goal):** The system shall prevent messages containing profanity from being sent to in the chat

**Description:** Predefined explicit words that are not allowed will not appear in any chat

**Primary Actors:** User, Database system

**Main:** *Pre-condition: User must be logged into application and mentor must be approved tutor*

1. User logs in

2. User clicks on ‘mentor’ tab

3. User selects class they need mentoring in

4. User selects request tutor

5. User request/profile is sent to database to await matching of tutor

6. Notification is sent to eboard member handling matching of tutors

Post Condition: Go to Case #1.7

### 

### **Case number:** 1.8

**Title (Goal):** The application shall be able to send notifications for meetings and events via email or push notification

**Description:** Notifications are sent to user when they sign up for events and meetings

**Primary Actors:** User, Club admin, Database system

**Main:** *Pre-condition: user must be logged into application*

1. User logs in

2. User clicks on ‘Upcoming events’

3. The user selects an event or a meeting to attend

4. A push notification and an email is sent to the user on the event day

Post Condition: None

### **Case number:** 1.9

**Title (Goal):** Users shall be able to message other users directly

**Description:** User uses the personal messaging feature to connect with another registered user

**Primary Actors:** User, database system

**Main:** *Pre-condition: user must be logged into application*

1. User logs in

2. User selects another user from ‘direct message’ list

3. The user connects to another user

Post Condition: None

### **Case number:** 2.0

**Title (Goal):** Users shall be able to send messages inside of groups with set topics

**Description:** The topics are set by the admin for discussions

**Primary Actors:** User, Admin

**Main:** *Pre-condition: user must be logged into application*

1. User logs in

2. User clicks on ‘Discussions’

3. User gets added to the “Discussions” chat with other members

Post Condition: None

### 

### **Case number:** 2.1

**Title (Goal):** Non-members shall be able see events and create an account

**Description:** Prospective users have access to read club’s upcoming events

**Primary Actors:** User, database system

**Main:** *Pre-condition: user must have the application installed*

1. User opens the application

2. User clicks on ‘Upcoming events’

3. To sign up and get more details, the application asks for the user to sign up

4. User signs up using the ‘sign up’ page

5. User creates an account

Post Condition: None

### **Case number:** 2.2

**Title (Goal):** Users shall have the ability to create a profile

**Description:** the user adds personal information for the profile

**Primary Actors:** User, database system

**Main:** *Pre-condition: user must be approved by the admin to sign up for the club*

1. User logs in

2. User clicks on ‘User profile’

3. User updates ‘contact address’, ‘courses taken’, ‘current courses’

4. User saves the profile

Post Condition: Go to case #2.3

### **Case number:** 2.3

**Title (Goal):** The system should allow users to see a list of known ACM GSU alumni

**Description:** Members will be able to see all members of ACM that have graduated from GSU

**Primary Actors:** User, database system

**Main:** *Pre-condition: user must be logged in and be a member*

1. User opens application

2. User must log in

3. Users will be able to see other users with the tag of Alumni

Post Condition: None

### 

### **Case number:** 2.4

**Title (Goal):** The system shall have the ability to keep a tally of the attendance at each meeting

**Description:** ACM board members and/or the administrator can use the app to keep track of who has/hasn't show up to a meeting

**Primary Actors:** User, database system

**Main:** *Pre-condition: User must be a board member or an administrator*

1. User opens application

2. User open ‘Events’ Page

3. User selects event or meeting that they would like to do attendance for

4. User selects which user attended that meeting/event

Post Condition: None

### **Case number:** 2.5

**Title (Goal):** The system shall allow a user to be removed from the ACM app after deferring to ACM constitution

**Description:** Board members will be allowed to ban users from using the app based on violation of ACM constitution

**Primary Actors:** Administrator, database system

**Main:** *Pre-condition: user must be an administrator*

1. User opens application

2. User clicks the name of another user

3. User selects ‘Ban’

4. User selects reason for ban

5. Selected user is then banned

Post Condition: None

### 

### **Case number:** 2.6

**Title (Goal):** The system shall allow administrators to create anonymous voting polls for the executive board

**Description:** Polls will be created for ACM board to vote on

**Primary Actors:** Board Members, Administrators, database system

**Main:** *Pre-condition: To post a poll the user must be an administrator and to vote user must be a part of the ACM executive board*

1. User opens application

2. User goes to the ‘Chat’ page

3. User selects ‘Create Poll’

4. User fills out poll form

5. User submits post

6. Applicable users will be able to vote on the poll

Post Condition: None

### **Case number:** 2.7

**Title (Goal):** The system should allow users to get in contact with possible employers for internships, co-ops, and jobs

**Description:** Users of the app will be able to see information of companies that are hiring

**Primary Actors:** users

**Main:** *Pre-condition: The user must be a member*

1. User opens application

2. User goes to the ‘Employment Opportunity' page

3. Users can browse possible employment options

Post Condition: None

### 

### **Case number:** 2.8

**Title (Goal):** The system should allow users to see a list of known ACM GSU employers

**Description:** Users of the app will be able to see information of companies that have hired ACM GSU alumni

**Primary Actors:** users

**Main:** *Pre-condition: The user must be a member*

1. User opens application

2. User goes to the ‘Employment Opportunity' page

3. Users can browse possible employment option

### Case Number 2.9

**Title (Goal):** The system may have the ability for administrators to keep a tally of the attendance at each meeting

**Description:** There may be a calendar or notes section for the e-board members to record meeting attendance

**Primary Actors:** User, Database system

**Main:** *Pre-condition: E-board member must be logged in to log attendance*

1. E-board member logs into the application

2. E-board member clicks on the e-board control panel

3. E-board member clicks on the attendance tab

4. E-board member either creates a note to log the attendance for that date

OR

5. E-board member selects the date of the meeting in a calendar to log the attendance for that date

Post Condition: None

### 

### Case Number 3.0

**Title (Goal):** The system may allow administrators to accept electronic payments via paypal or venmo

**Description:** There may be a page with links for members of the club to submit electronic payments to pay their dues

**Primary Actors:** User, Database system

**Main:** *Pre-condition: Users must be logged in to pay their dues*

1. User logs into the application

2. User clicks on the club information tab

3. User clicks on the pay dues button

4. User clicks on the paypal/venmo button

5. User is redirected to paypal/venmo to send their payment

Post Condition: User profile will be updated with paid symbol

### Case Number 3.1

**Title (Goal):** The system may allow administrators to see a list of members who have paid their dues

**Description:** It is important for members of the e-board to be able to keep track of who has and hasn't paid dues. This feature will solve this problem by providing e-board members with an easy-to-access place to keep track of who has paid their dues

**Primary Actors:** User, Database system

**Main:** *Pre-condition: Members of the e-board must be logged in to see this information*

1. E-board member logs in

2. E-board member goes to the e-board control panel

3. E-board member opens the member information tab

4. Each member will have a check mark next to their name if they have paid their dues and are a member of the ACM club

Post Condition: None

### 

### Case Number: 3.2

**Title (Goal):** The system may allow administrators to create anonymous voting polls for the executive board

**Description:** Users will be able to vote on polls that are created by members of the e-board

**Primary Actors:** User, Database system

**Main:** *Pre-condition: Members of the e-board must be logged into the application to create polls and users must be logged into the application in order to anonymously vote*

**E-board perspective**

1. E-board member logs in

2. E-board member goes to the board control panel

3. E-board member opens the polls tab

4. E-board member creates and submits poll

**User perspective**

1. User logs into the app

2. User goes to polls page

3. User chooses their vote and submits it

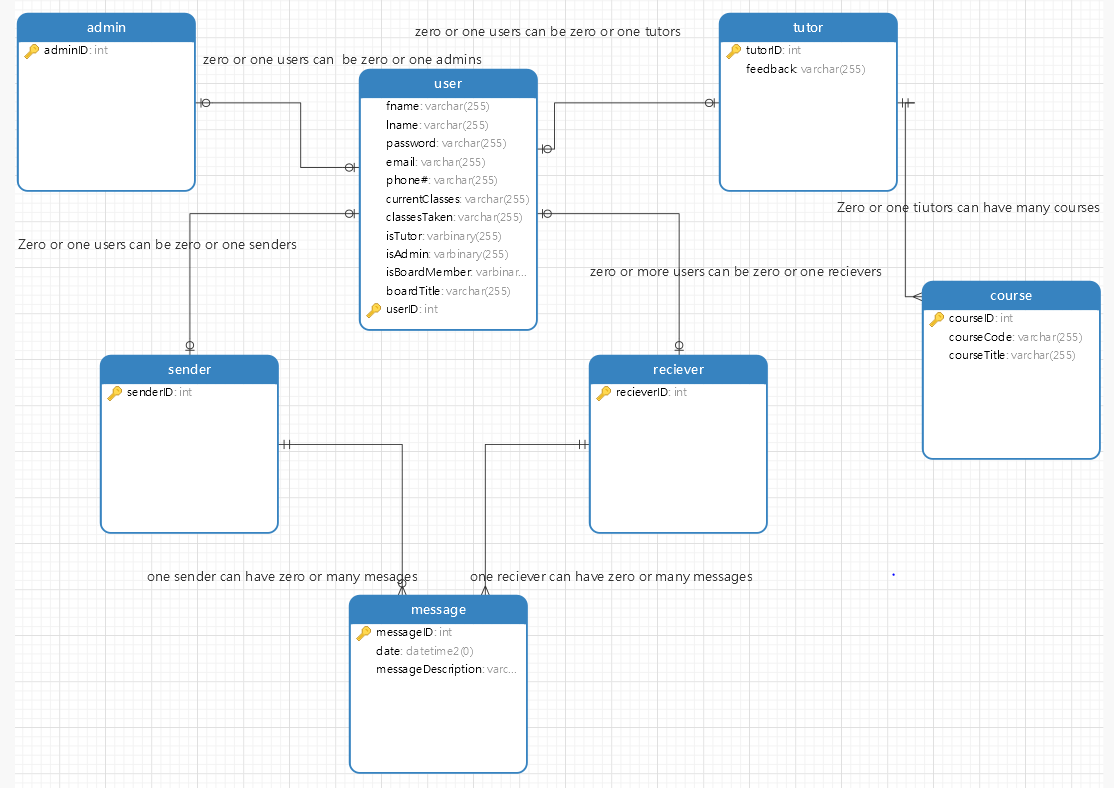
4. User's vote is recorded and statistics are shown

Post Condition: None

## 4.3 Appendix C – Detailed Class Diagrams

## 

## 4.4 Appendix D – Detailed ER Diagrams



## 4.5 Appendix E – Log of Meeting and Tasks

### Week of 16 February 2020

Notes:

* Team met on Wednesday, 02/12
  + Attendance: Griffin Bryant, Marquise Jennett, Dylan Myers, Mansi Pandya, Everrick Wright
* Followed up as a team from last week's meeting with Dr. Allen
  + Discussed the details for the questions about the project requirements
  + Began requirements elicitation questions for each requirement
  + Began rough draft for Software Requirements Specifications Document

Current Plans:

* We plan on meeting again on Monday to go over the requirements elicitation questions and begin the SRS Document
* We also plan to meet again on Wednesday to collaborate on the SRS Document

### Week of 22 February 2020

Notes:

* Team meeting on Monday, February 17th:
  + Everybody was present except for Everrick, he was sick that day but we caught him up on what he missed
  + We went over the example SRS documents and began a rough outline for ours
* Team meeting on Wednesday, February 19th:
  + Everybody was present except for Marquise, but we caught him up on what he missed
  + Continued working on the SRS document as a team
* Team meeting on Thursday
  + Everybody was present
  + Continued working on finalizing SRS document and began conversation about design planning

Current Plans:

* Finish final version of SRS document on Monday, February 24th and submit it
* Continue design planning and have a high level design by Friday
* Assign design roles on Monday, February 24th
* Begin in depth design on Monday, March 2nd

### Week of 29 February 2020

Notes:

* Team meeting on Monday, February 24th:
  + Everybody was present
  + Finalized and submitted SRS document
  + Assigned Design Roles:
    - Database Design: Griffin, Dylan, Mansi
    - UI Design: Marquise, Griffin, Mansi
    - Architectural Design: Everrick

Current Plans:

* Finish high level designs
* Meeting with Dr. Allen on Monday, March 2nd to discuss SRS Document and show early designs for review

### Week of 7 March 2020

Notes:

* Team meeting on Monday, March 2nd:
  + Everybody was present.
  + We met with Dr. Allen to discuss the SRS and review for any changes that need to be made.
* Team meeting on Thursday, March 5th:
  + Everybody was present.
  + Discussed initial design plans for UI and database

Current Plans:

* Mansi and Dylan are working on the database schema and ER diagrams
* Griffin, Marquise, and Everrick are working on UI Designs
* Design Document is due on Friday

### 

## 