# Time Management Project

McKenna Woodrow

April 26, 2018

# Contents

# 1 Documentation

## 1.1 Outline

This application is a Time Management tool in which anyone can create an account. You must choose any user name to input and a valid email with a password and confirmation password that are the exact same. You then can create a group with a title which will hold all of your individual tasks. The group names can be edited or deleted after you create them. All of the tasks within your group will be deleted if the group is deleted. A task consists of a name, description, priority, progress, and due date. You can then edit the task and update the progress or priority to remain on schedule and organized. This tool is meant to help prioritize and organize lives and to help you manage your time better.

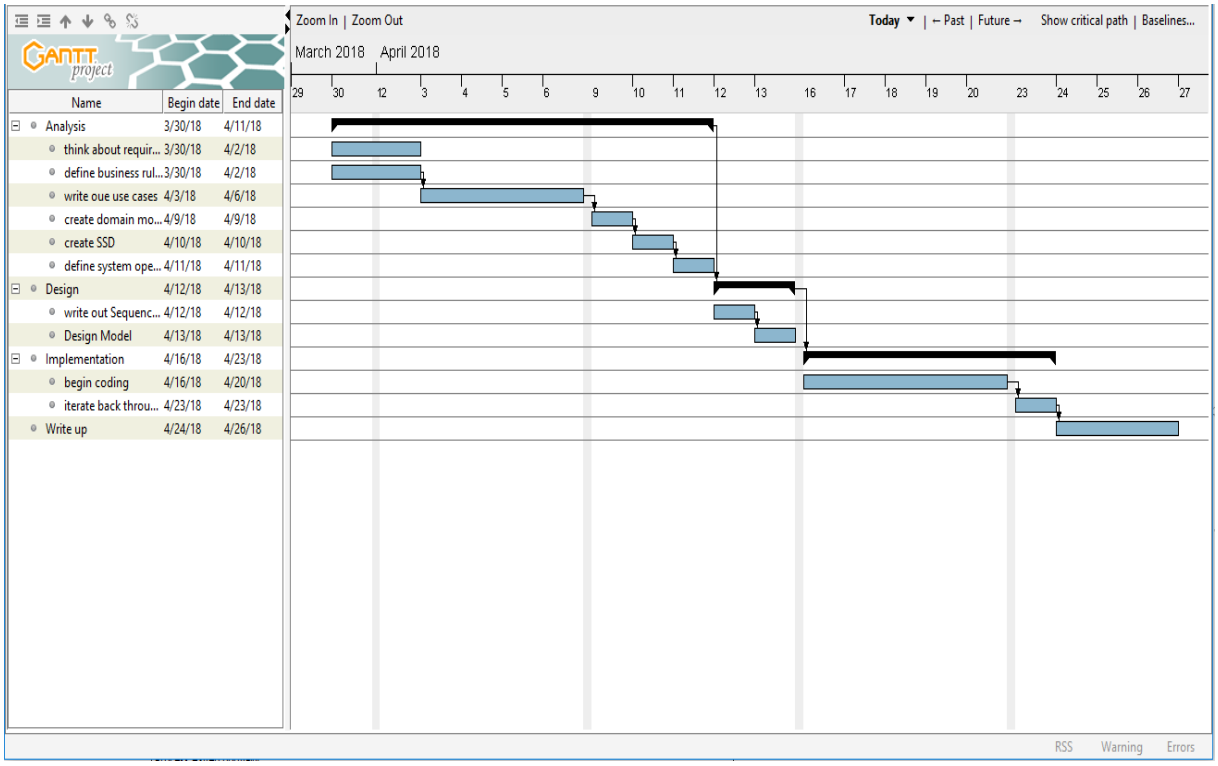
## 1.2 Vision Statement

The vision statement for this project is to help college kids plan their time out better and have a more productive and fulfilling college experience. Often times students will become so overwhelmed by the multitude of things that need to get done that they don't even know where to begin and often waste a lot of time. I created this tool for people to be able to organize their to do list, and remain productive, and accomplish everything that need to get done in life.

## 1.3 All Roles

I performed and completed all of the roles of this project

## 1.4 Gantt Diagram




## 1.5 Requirements

### 1.5.1 Functional

user can create public user account
user can create and delete groups
user can create task they own
user can edit task they own
user can delete task they own
user can update task if in group affiliated with task

### 1.5.2 Non Functional

database remains secure with consist data
when user deleted all associated groups and task deleted
when group deleted all associated tasks deleted

## 1.6 Business Rules

1. Groups and Tasks can only belong to one User
2. User can have multiple groups and tasks
3. Groups can have multiple tasks
4. Task has to be apart of one and only one group

## 1.7 Use Cases

Actors:
Public Users
Admin
System

### 1.7.1 UC 1: User creating public account

Precondition: user want to create public account
Postcondition: account is created and made available to user

Main Success Scenario:
1. User wants to create public account
2. System displays info for user to input
3. User inputs valid information
4. System verifies email and password and assigns user userID
5. Account made available for user to use

Alternative Paths:
3.* user inputs invalid email
3.1 system informs user email is invalid

### 1.7.2 UC2: User creating group

Precondition: user wants to create group
postcondition: group is created

Main Success Scenario:
1. User wants to create group
2. System requests group title
3. User inputs group name
5. System creates group and assigns it unique id
4. group is created

Alternative Paths:

3.* User inputs empty group name
3.1 System requests valid name to make group with

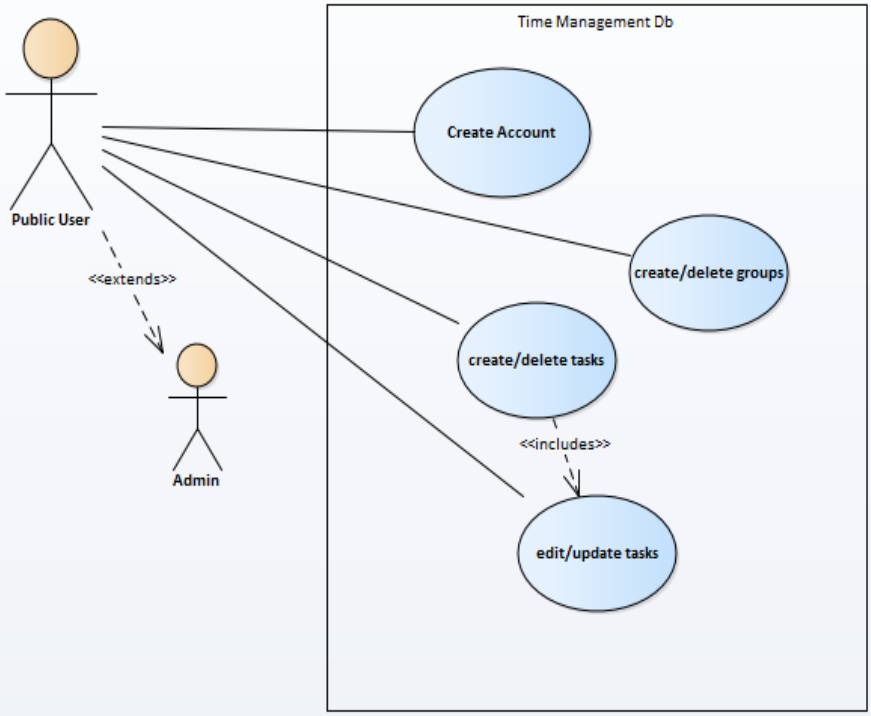
#### 1.7.3 UC3: User creating task

Precondition: User wants to create task
Postcondition: Task created within a group

Main Success Scenario
1. User want to create task
2. User selects valid group to put task in
3. System requests group, task name and description
4. User inputs information
5. System creates task and assigns it a specific task Id

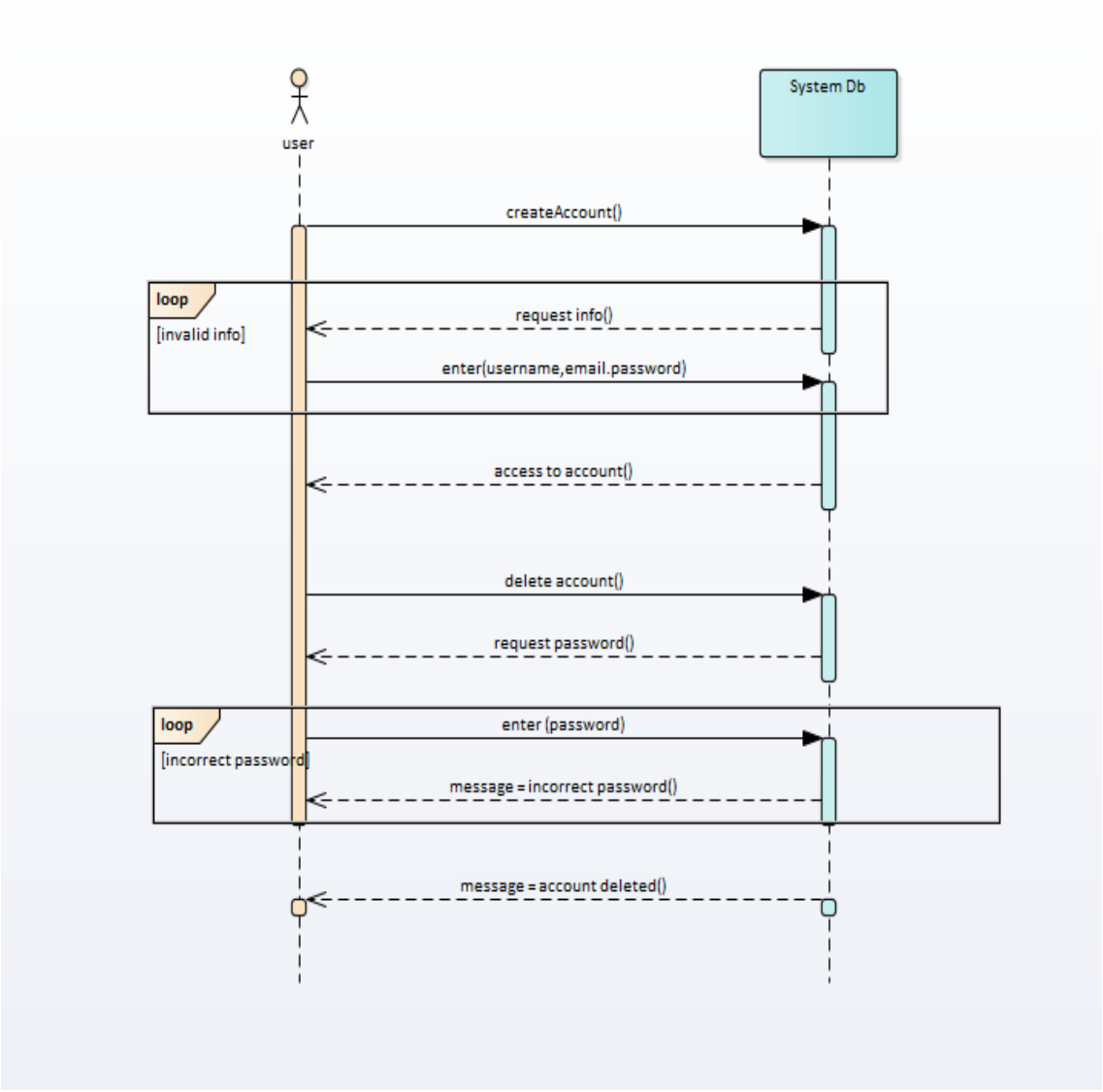
Alternative Paths:
2.* User does not select group to put task in
2.1 System requests user to select group
4.* User does not input all of the information
4.1 System requests more information

## 1.8 Use Case Diagram

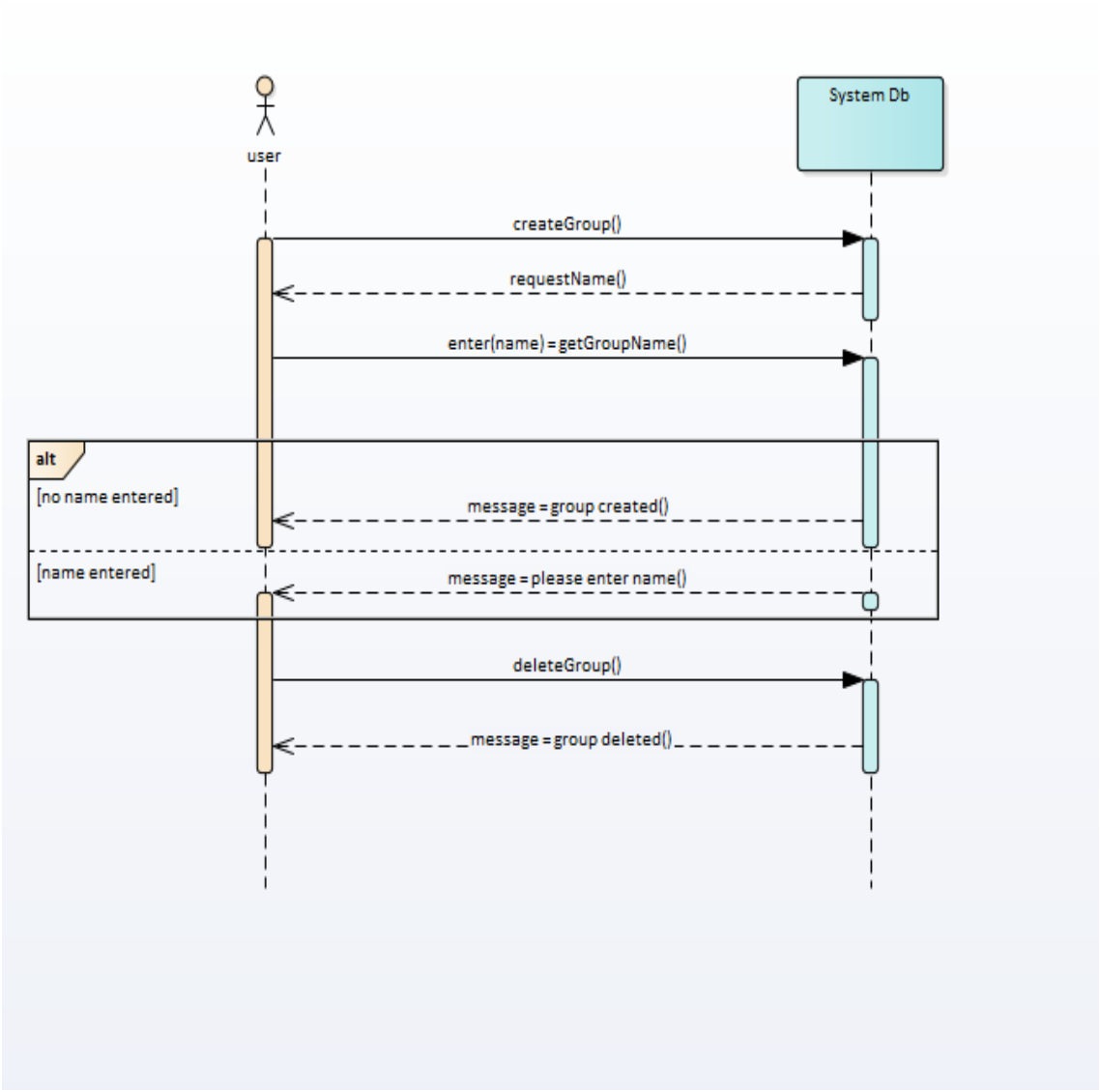


## 1.9 System Sequence Diagram

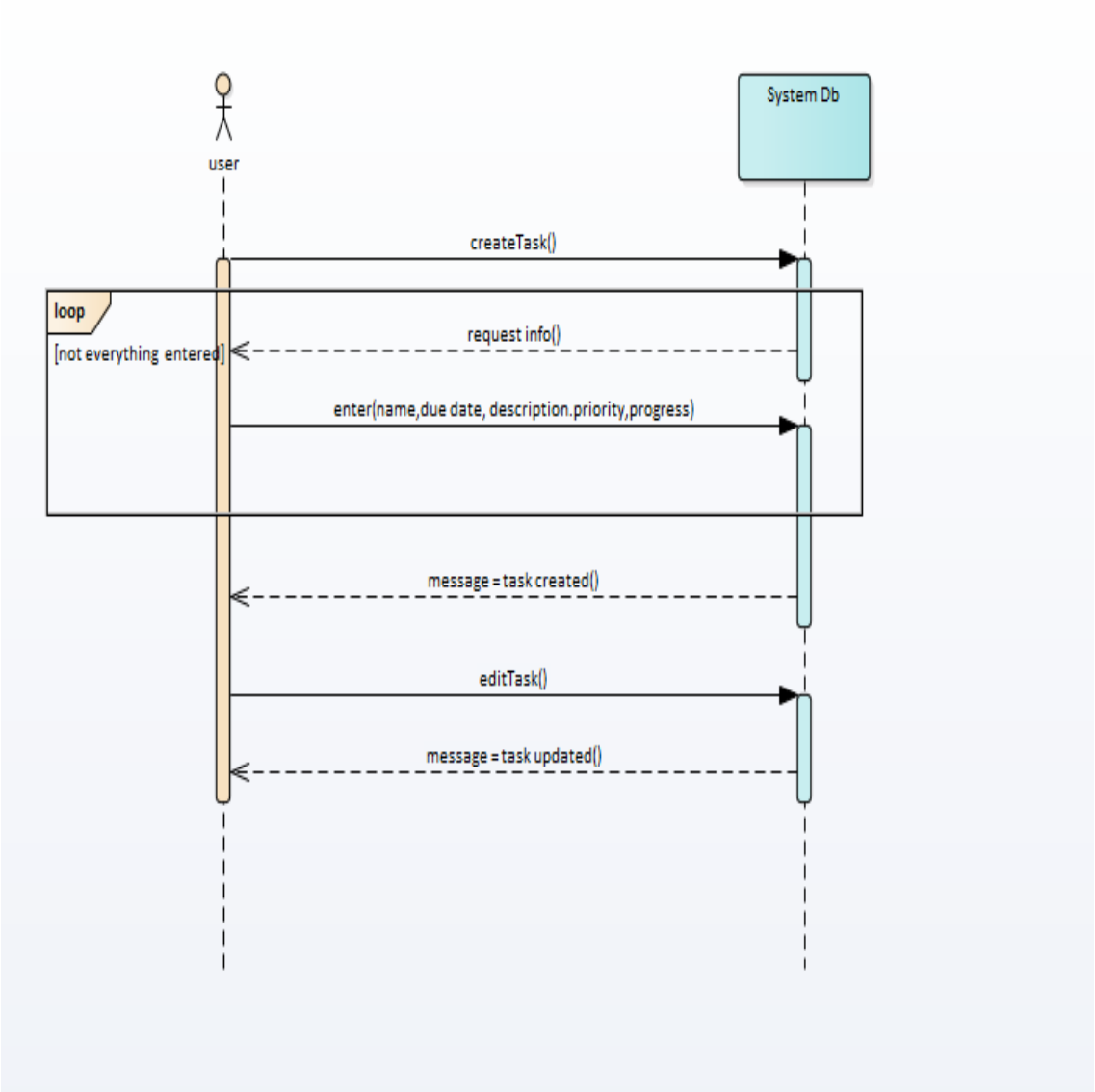
### 1.9.1 Use Case 1



### 1.9.2 Use Case 2



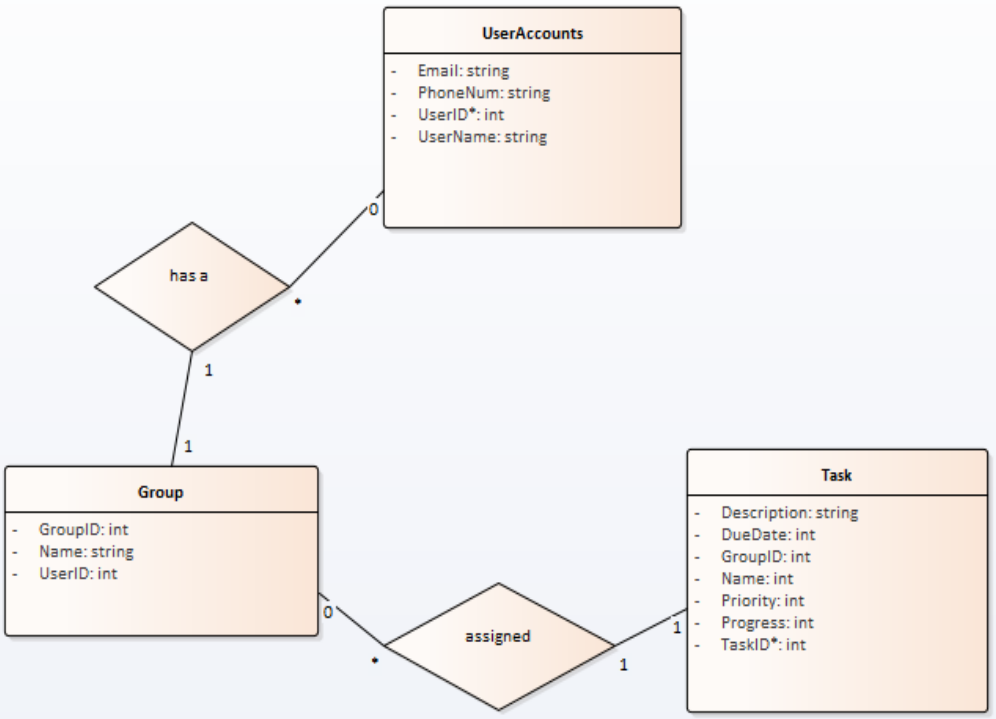
### 1.9.3 Use Case 3




## 1.10 System Operations

createAccount()
createGroup()
deleteGroup()
createTask()
deleteTask()

## 1.11 Domain Model



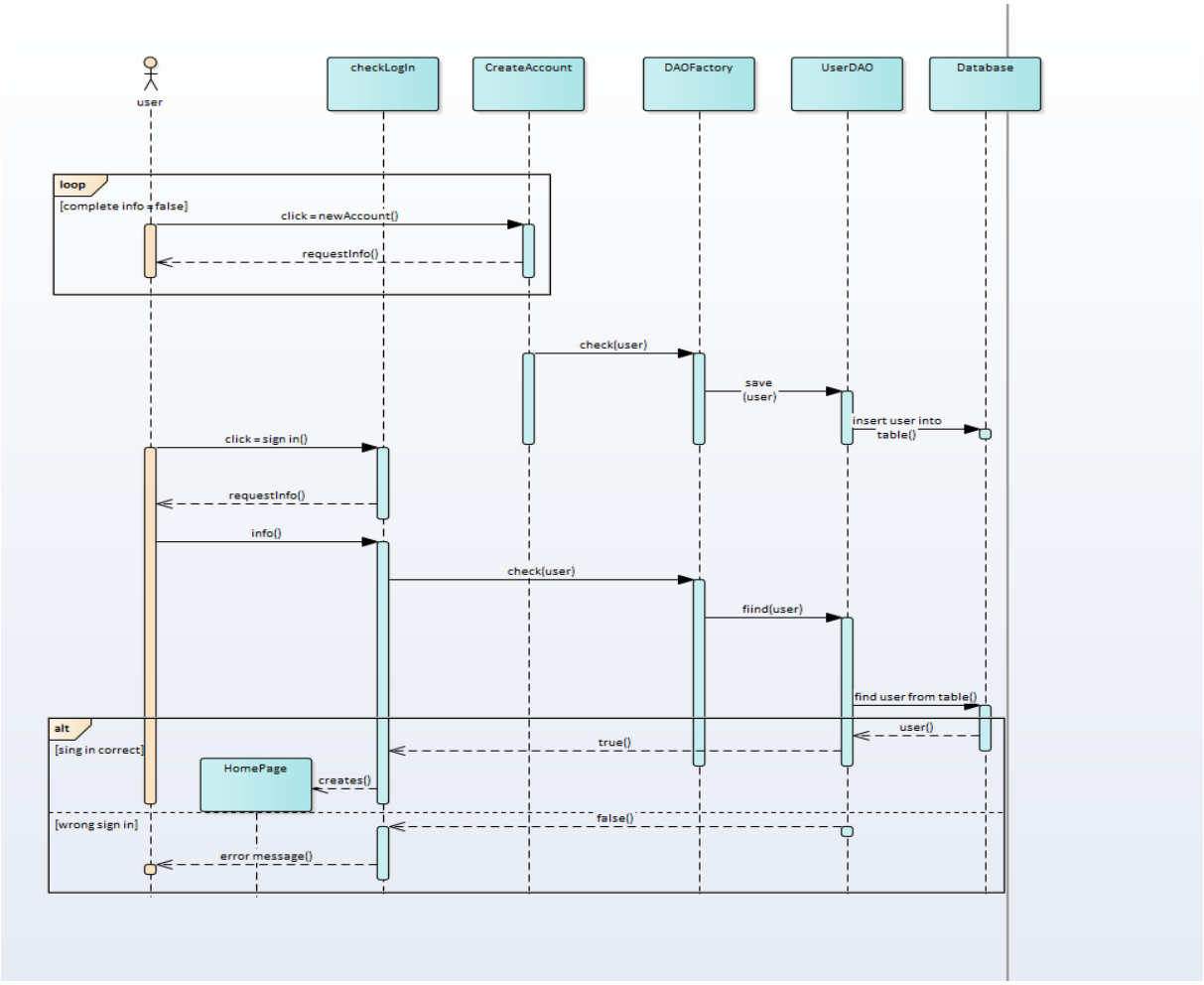
## 1.12 Operation Contracts

Contract CO1: createAccount
Operation: creatAccount(user:User)
Cross References: Cross Reference UC1
Precondition: user want to create public account
Postcondition:
user input information
a UserAccount instance was created
userAccount was inserted into database
userAccount assigned a userID

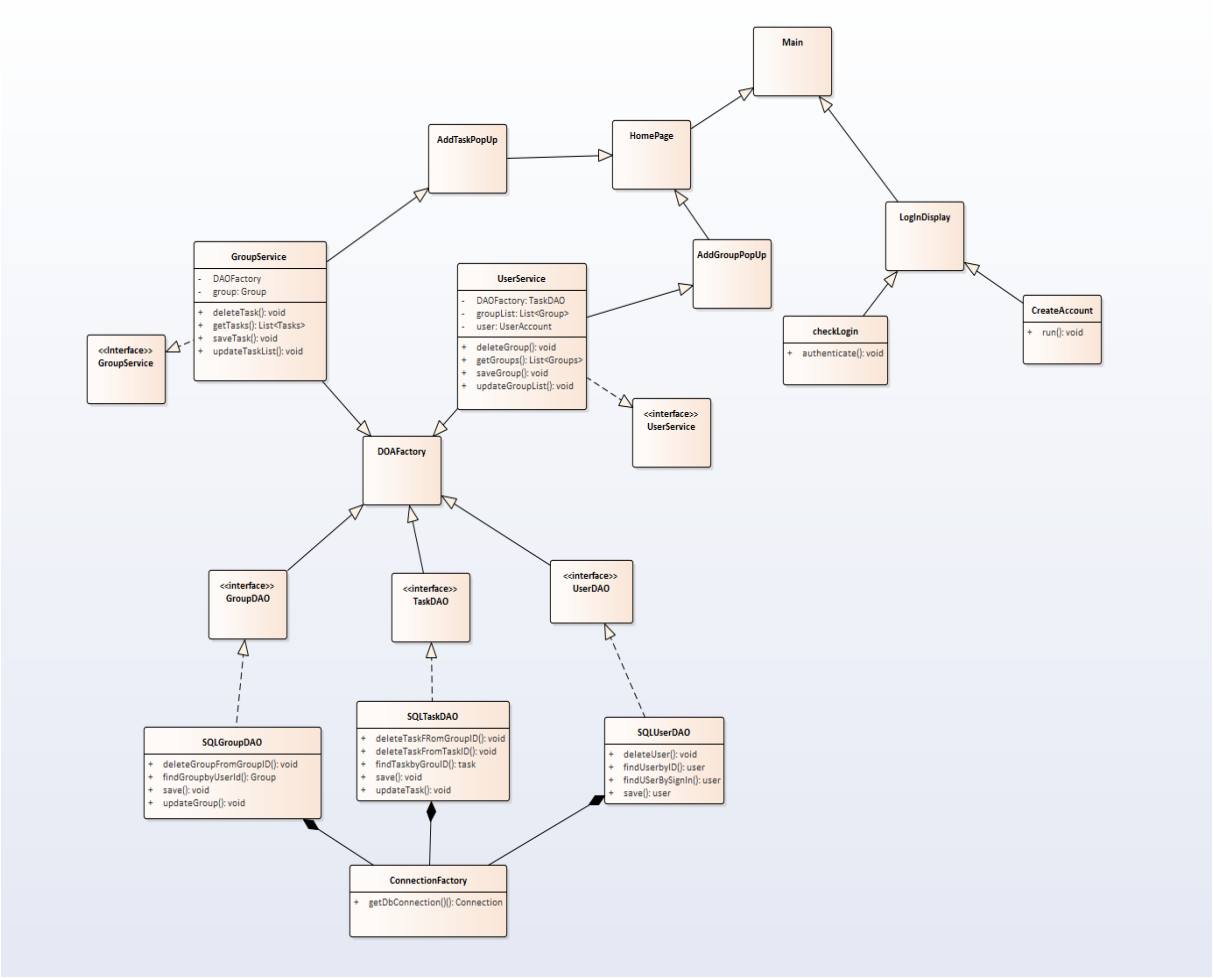
Contract CO2: createGroup
Operation: creatGroup(group:Group)
Cross References: Cross Reference UC2
Precondition: user want to create group
Postcondition:
user gives group a name
a group instance was created
group was inserted into database
group assigned a groupID

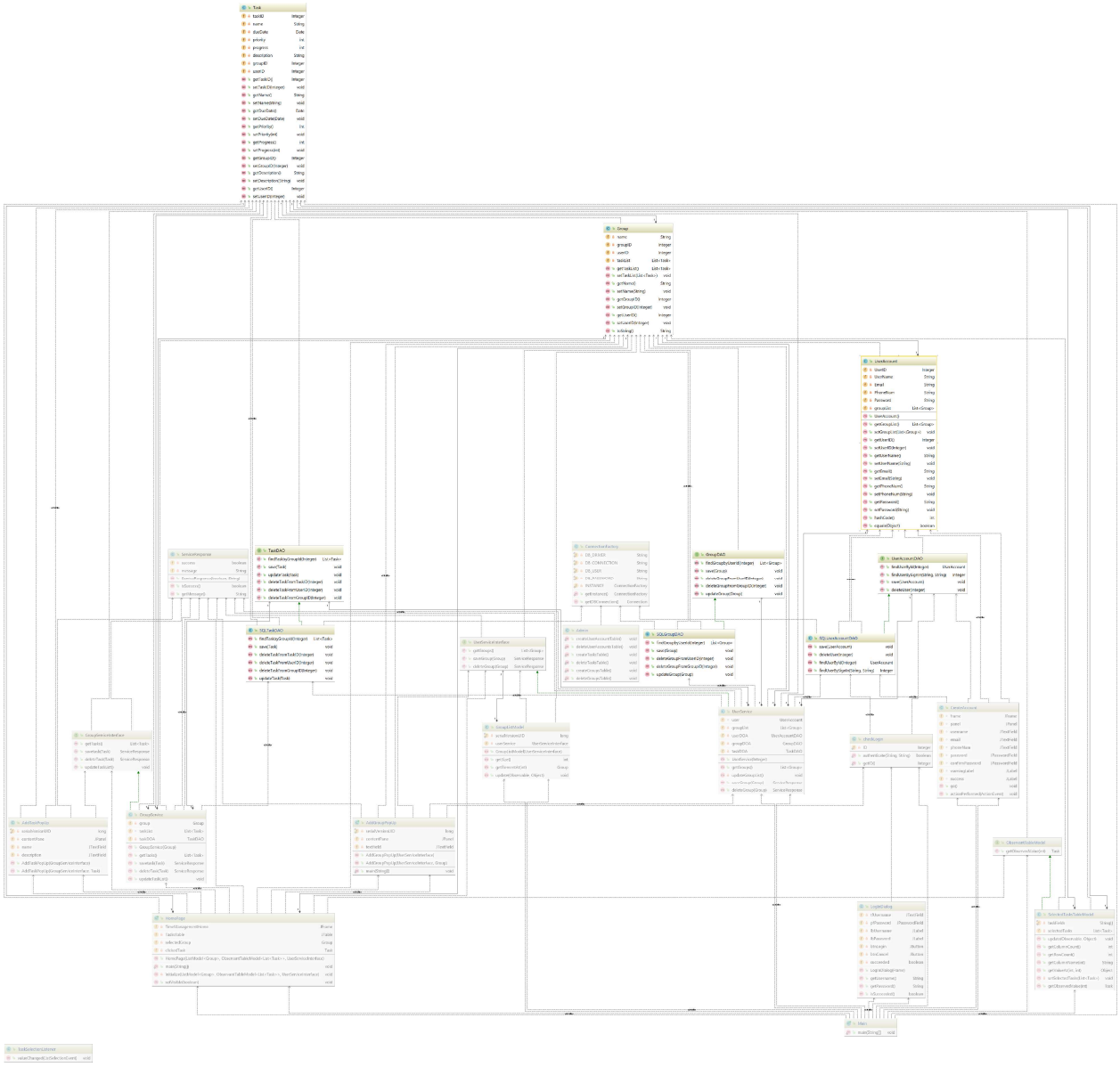
Contract CO3: createTask
Operation: createTask(task:Task)
Cross References: Cross Reference UC3
Precondition: user want to create group
Postcondition:
user inputs task information
a task instance was created
task was inserted into database
task assigned a groupID

## 1.13 Sequence Diagram: UC1



## 1.14 Design Model





## 1.15 Justification of Grasp

General Responsibilities Assignment Software Patterns:
When creating the DAO classes I used the pattern of low coupling and information expert to assign the responsibility of accessing the database to only a few classes with the most information on the database.This also applies the concept of high cohesion. I then used the pattern indirection to delegate the responsibilities of different DAO classes through the FactoryDAO. I then used the idea of high cohesion to delegate create the Service classes to handle the user response. The PopUp classes then implement the controller pattern becuase they are the ones who handle and control the clicking of the different buttons on the user interface.

## 1.16 Design Patterns

1.)Singleton: single instance of connection class to database

2.)Abstract Factory: connectionFactory also abstract factory, Single instance

3.)Abstract Factory: DAOFactory encapsulate the SQLDOA classes. Instances of the SQLDOA classes only exit within the DOAFactory, so that it is the only ones who have knowledge of them. The delegation of responsibilities for accessing the database is condensed to one class. User does not need to worry about how it is accessing the database or storing info only needs to worry about the provided functions.

4.)Prototype: AddGroupPopUp and AddTaskPopUp have prototypes in their constructors of UserServiceInterface and GroupServiceInterface

5.)Command: Usage of the ActionListener(e) uses the command pattern because the ActionLister(e) will give a message, shown as a button, and it depends on the user when they want to respond and click it, activating the methods inside the ActionListener block.

6.)Mediator: AddTaskPopUp and AddGroupPopUp uses the mediator pattern because it handles all of the events, button clicks, of a specific pop up in these single classes.

7.)DAO Access Patterns: DAO classes use the DAO pattern to access the database and condense the amount of classes that are accessing the database. They also have interfaces to hide this functionality from to user.

## 1.17 Testing

In my project I tested the three use cases in which the database is accessed. I thoroughly tested each function of the SQLTaskDAO, SQLGroupDAO, and

the SQLUserDAO. This was to insure before I began to add any of the user interface or patterns to my project I was safely and correctly interacting with the database so that before I moved on to the next level of my project I was sure that the base level functionality was functioning properly.

### 1.18 Total Hours Spent

60hrs

## 2 Implementation

git clone https://kenna33@bitbucket.org/kenna33/timemanagementproj.git