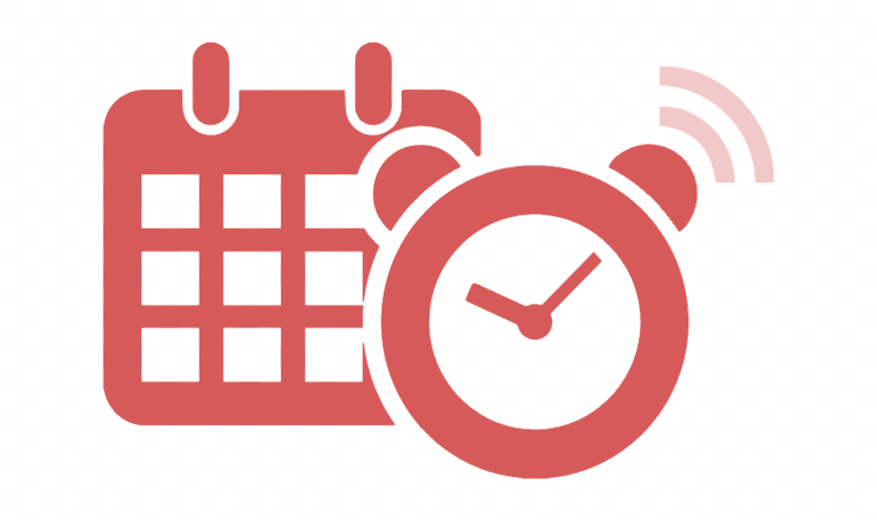


**Reminder App**



Design

CMSC 495 6380

Revised: October 10, 2019

Group 4

Dillon Cobb

Melissa Eardley

Maria Tkacheva

Allan Yu

**Revision History Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Changed By | Change(s) | Version |
| 9/13/2019 | Dillon Cobb  Melissa Eardley  Allan Yu | Initial Revision | 1.0 |
| 9/14/2019 | Allan Yu | Clarified some Pseudocode | 1.1 |
| 9/15/2019 | Melissa Eardley | Cleaned up some graphics | 1.2 |
| 10/6/2019 | Melissa Eardley | Minor changes | 1.3 |
| 10/10/2019 | Melissa Eardley  Allan Yu  Dillon Cobb | Updated to reflect final requirements | 1.4 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Event Trace Diagrams

## Scenario 1: StartUp

Description: The user starts the session

Pre-condition: The user is not already logged in

Post-condition: The user enters login credentials

User

Database

Login

Event Entry

Email

Client

Reminder Display

Starts the Application

Verify User with database

If valid, user is logged in

If invalid, user is allowed to re-enter login data

## Scenario 2: New User Input Data is Valid

Description: The user enters valid information for establishing a new user account

Pre-condition: The user account does not already exist for email address

Post-condition: The user account is created; User receives email indicating that account has been created

User

Database

Login

Event Entry

Email

Client

Reminder Display

Enter

Username,

Email,

Password

Login User

Add User

## Scenario 3: New User Input Data is NOT Valid

Description: The user enters invalid information for establishing a new user account

Pre-condition: N/A

Post-condition: The user account is NOT created; error message is displayed

User

Database

Login

Event

Entry

Email

Client

Reminder Display

Enter

Username,

Email,

Password

Invalid Entry

Warning Displayed

## Scenario 4: User Enters Valid Login Data

Description: The user enters valid log information

Pre-condition: User exists in the database

Post-condition: User is able to access the web display; User can enter events

User

Database

Login

Event

Entry

Email

Client

Reminder Display

Enter

Username,

Password

Login User

Validate User

## Scenario 5: User Enters Invalid Login Data

Description: The user enters invalid log information

Pre-condition: N/A

Post-condition: Error message is displayed

User

Database

Login

Event

Entry

Email

Client

Reminder Display

Enter

Username,

Password

Invalid Entry

Warning Displayed

Validate User

## Scenario 6: User Enters Valid Event Data

Description: The user enters valid event information

Pre-condition: User is already logged in

Post-condition: Event is added

User

Database

Login

Event

Entry

Email

Client

Reminder Display

Enter

Username,

Password

Login User

Enter name, description, date, time

Add Event

Display Events

Send notifications

as set by the user

## Scenario 7: User Enters Invalid Event Data

Description: The user enters invalid event information

Pre-condition: User is already logged in

Post-condition: Event is NOT added; Error message is displayed

User

Database

Login

Event

Entry

Email

Client

Reminder Display

Enter

Username,

Password

Login User

Enter name, description, date, time

Invalid Entry

Warning Displayed

## Scenario 8: Notification Data Sent

Description: Notification data is scheduled to be sent

Pre-condition: Notification time is set; Time for notification to be sent is met

Post-condition: Notification is sent

Email Client sends email to user

User

Database

Login

Event

Entry

Email

Client

Reminder Display

Sends email notification request to Email Client

## Scenario 9: Shutdown

Description: Session complete user log out

Pre-condition: User completed everything

Post-condition: Application shutdown

User

Database

Login

Event

Entry

Email

Client

Reminder Display

User selects logout

User is logged

out

# Class Design

class User {

int userID;

String username;

String password;

String email;

VerifyData (userID, username, password , email) {

//regex=regexes for the input lengths and accepted characters

if ((username.match(regex)) && password.match(regex) && email.match(regex)){

return sendToDb(userID,username,password,email);

}

return false;

}

class Login {

String username;

String password;

verifyUser(Username, password);

Boolean verifyUser(Username, Password) {

if (Username == “valid”) and (password == “valid”) {

verifyUser = sendToDb(userID,Password);

} else {

verifyUser = false;

}

Return verifyUser;

}

Void LoginUser(userID, Password) {

If (verifyUser == true) {

loginUser;

}

}

Class Database {

Tables: User info (Username, UserID, Password, Email),  
 Event info (eventName, description, date, time)

}

class EventEntry {

String eventName;

String eventDescription

Date eventDate;

String time;

void sendNotifications{

sendRequestToMailClient();

}

}

class MailClient {

//third party software

}

class ReminderDisplay {

JFrame f = new JFrame("Reminder App");

JTable table = new JTable(new createTableModel());

String UserID;

String Username;

class createTableModel extends AbstractTableModel {

private String[] columnNames = *[“Name”, “Description”, “Date”, “Time”]*

private Object[][] data =  *from the database*

public int getColumnCount() {

return columnNames.length;

}

public int getRowCount() {

return data.length;

}

public String getColumnName(int col) {

return columnNames[col];

}

public Object getValueAt(int row, int col) {

return data[row][col];

}

public Class getColumnClass(int c) {

return getValueAt(0, c).getClass();

}

public void setValueAt(Object value, int row, int col) {

data[row][col] = value;

fireTableCellUpdated(row, col);

}

class LogOut {

invalidateSession();

return;

}

}

# Unresolved Risks and Risk Mitigation

The following table identifies risks and potential mitigation:

|  |  |  |
| --- | --- | --- |
| Risk #1 | Risk Description | Mitigation |
| 1 | Server malfunctions prevent user from accessing system or sending notifications | Possible mitigation, though beyond the scope of this effort, would be to add redundant servers |
| 2 | User enters invalid data | When possible, have data entry validation which prevents the user from adding invalid data |
| 3 | Daylight Savings time | Ensure that daylight savings time is accounted for when required; perhaps as a user defined flag |
| 4 | Updates to web browser interfere with ReminderApp | Monitor changes to Chrome, and ensure continued compatibility; update as necessary |
| 5 | Issues interfacing the subsystems | Choose software that is already known to interface |
| 6 | Weather impacts developers (loss of power or Internet, displacement) | All team members are geographically dispersed, so it is unlikely that all would face simultaneous catastrophes; a developer affected by a weather event or personal emergency is responsible for contacting the group to alert them of his situation; if needed the group will perform that person’s responsibilities to keep the project on track |
| 7 | User data is compromised | Store data in a way that minimizes the chance it will be compromised |