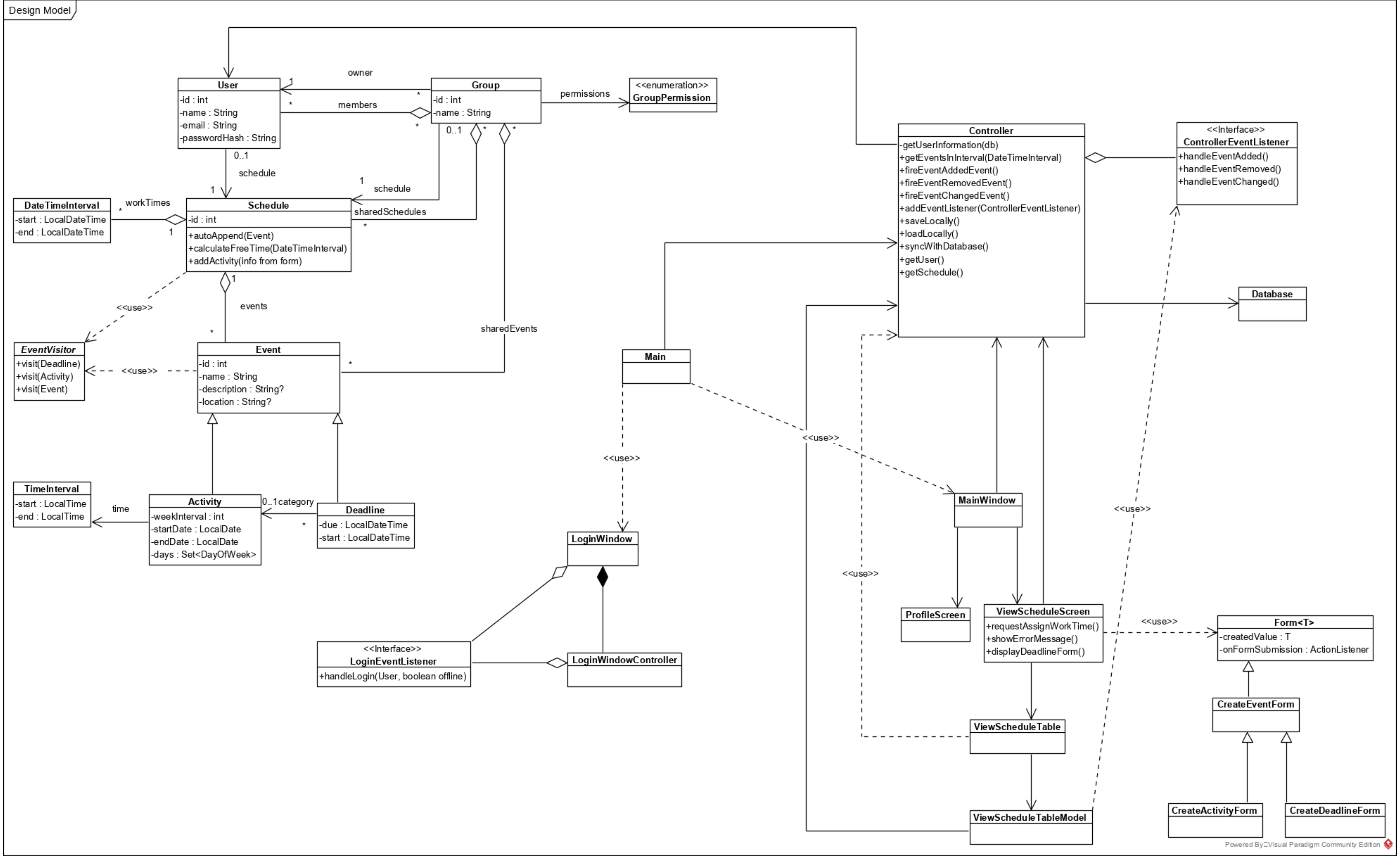
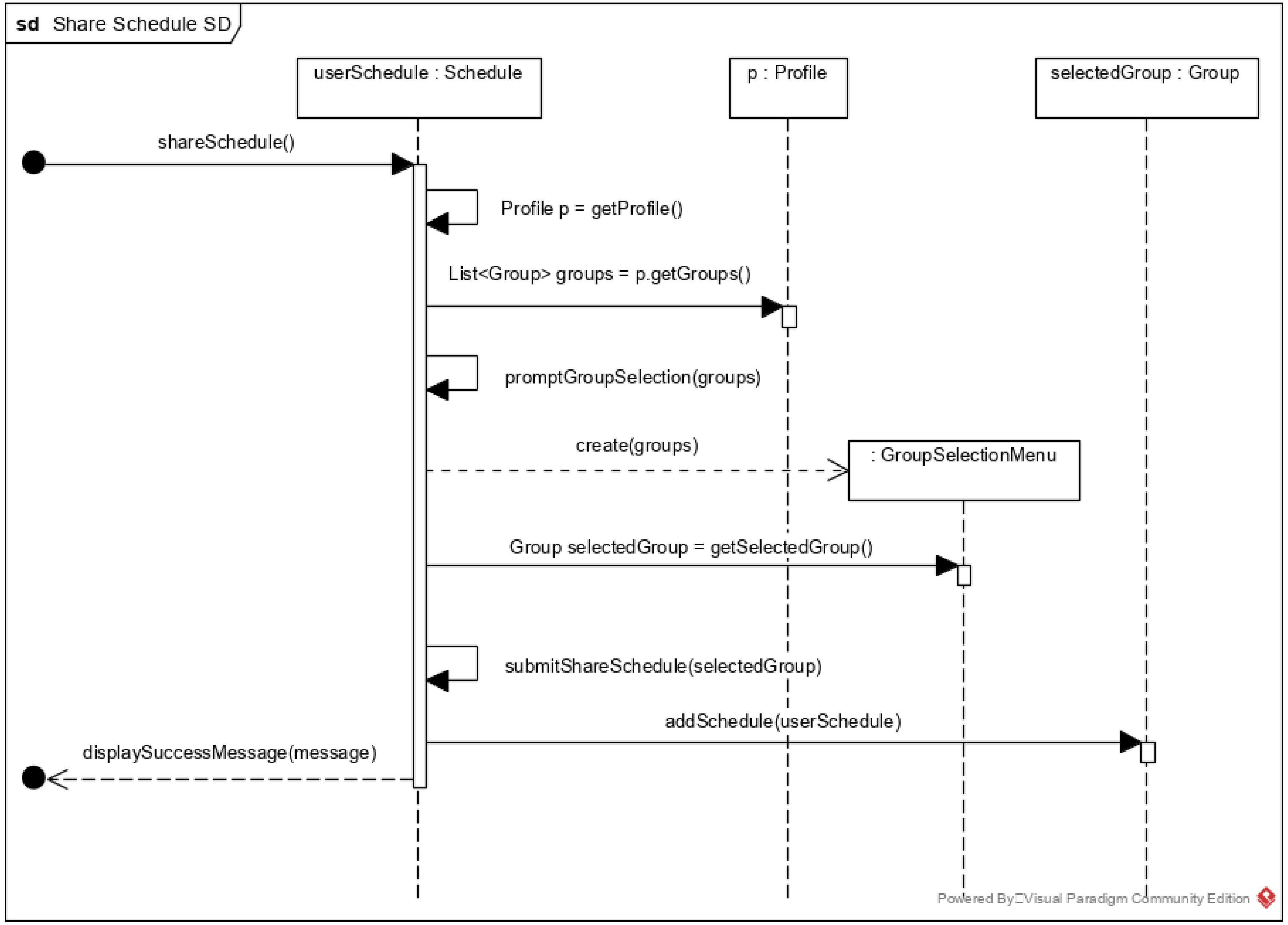
Iteration 2

Authors:

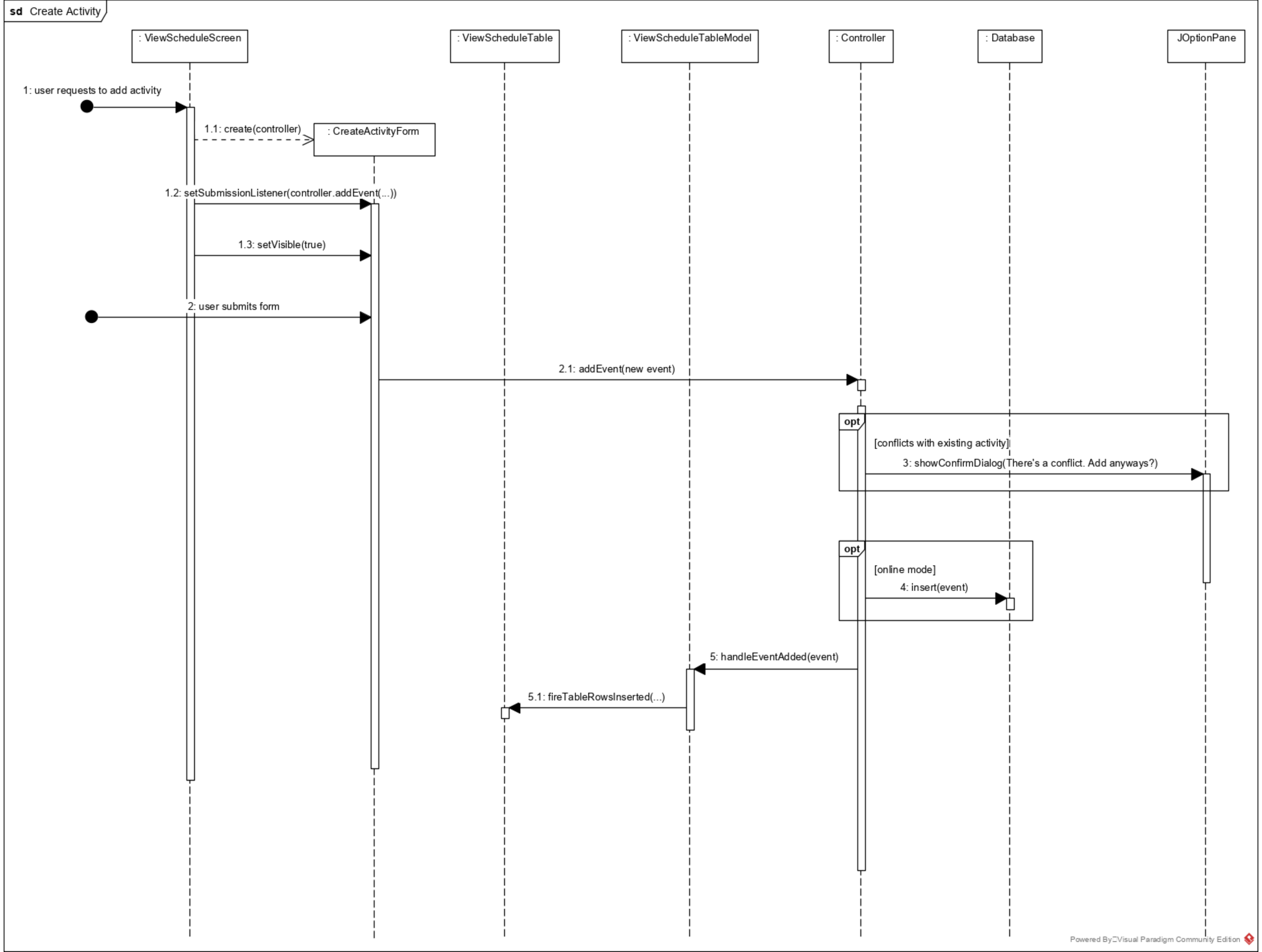
Kevin DeMars, Trenton Strickland, Eric Jaroszewski, Joshua Kanagasabai, Samuel Kim


Design Model
User
-id : int
-name : String
-email : String
-passwordHash : String
owner
1
*
members
*
*
Group
-id : int
-name : String
permissions
<<enumeration>>
GroupPermission
0..1
schedule
1
0..1
*
*
1
schedule
sharedSchedules
*
DateTimeInterval
-start : LocalDateTime
-end : LocalDateTime
workTimes
*
1
Schedule
-id : int
+autoAppend(Event)
+calculateFreeTime(DateTimeInterval)
+addActivity(info from form)
1
events
*
<<use>>
EventVisitor
+visit(Deadline)
+visit(Activity)
+visit(Event)
<<use>>
Event
-id : int
-name : String
-description : String?
-location : String?
*
sharedEvents
TimeInterval
-start : LocalTime
-end : LocalTime
time
Activity
-weekInterval : int
-startDate : LocalDate
-endDate : LocalDate
-days : Set<DayOfWeek>
0..1category
*
Deadline
-due : LocalDateTime
-start : LocalDateTime
Main
<<use>>
<<use>>
LoginWindow
<<Interface>>
LoginEventListener
+handleLogin(User, boolean offline)
LoginWindowController
Controller
-getUserInformation(db)
+getEventsInInterval(DateTimeInterval)
+fireEventAddedEvent()
+fireEventRemovedEvent()
+fireEventChangedEvent()
+addEventListener(ControllerEventListener)
+saveLocally()
+loadLocally()
+syncWithDatabase()
+getUser()
+getSchedule()
<<Interface>>
ControllerEventListener
+handleEventAdded()
+handleEventRemoved()
+handleEventChanged()
Database
<<use>>
MainWindow
<<use>>
ProfileScreen
ViewScheduleScreen
+requestAssignWorkTime()
+showErrorMessage()
+displayDeadlineForm()
<<use>>
Form<T>
-createdValue : T
-onFormSubmission : ActionListener
CreateEventForm
CreateActivityForm
CreateDeadlineForm
ViewScheduleTable
ViewScheduleTableModel
Powered By Visual Paradigm Community Edition

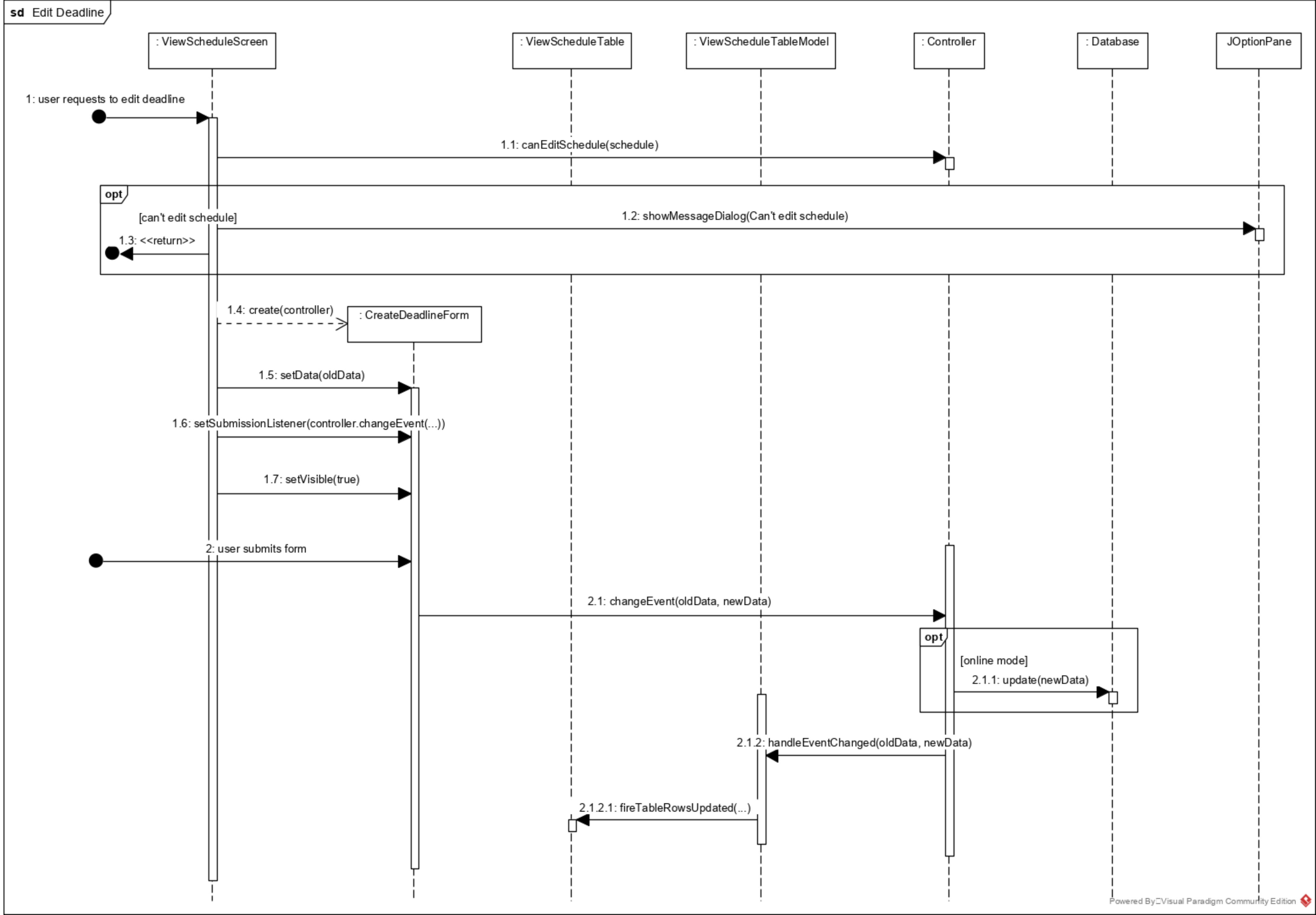
Sequence Diagrams


sd Share Schedule SD
userSchedule : Schedule
p : Profile
selectedGroup : Group
shareSchedule()
Profile p = getProfile()
List<Group> groups = p.getGroups()
promptGroupSelection(groups)
create(groups)
: GroupSelectionMenu
Group selectedGroup = getSelectedGroup()
submitShareSchedule(selectedGroup)
addSchedule(userSchedule)
displaySuccessMessage(message)
Powered By Visual Paradigm Community Edition

Sequence Diagrams


sd Create Activity
: ViewScheduleScreen
: ViewScheduleTable
: ViewScheduleTableModel
: Controller
: Database
JOptionPane
1: user requests to add activity
1.1: create(controller)
: CreateActivityForm
1.2: setSubmissionListener(controller.addEvent(...))
1.3: setVisible(true)
2: user submits form
2.1: addEvent(new event)
opt
[conflicts with existing activity]
3: showConfirmDialog(There's a conflict. Add anyways?)
opt
[online mode]
4: insert(event)
5: handleEventAdded(event)
5.1: fireTableRowsInserted(...)
Powered By Visual Paradigm Community Edition

Sequence Diagrams


sd Edit Deadline
: ViewScheduleScreen
: ViewScheduleTable
: ViewScheduleTableModel
: Controller
: Database
JOptionPane
1: user requests to edit deadline
1.1: canEditSchedule(schedule)
opt
[can't edit schedule]
1.2: showMessageDialog(Can't edit schedule)
1.3: <<return>>
1.4: create(controller)
: CreateDeadlineForm
1.5: setData(oldData)
1.6: setSubmissionListener(controller.changeEvent(...))
1.7: setVisible(true)
2: user submits form
2.1: changeEvent(oldData, newData)
opt
[online mode]
2.1.1: update(newData)
2.1.2: handleEventChanged(oldData, newData)
2.1.2.1: fireTableRowsUpdated(...)
Powered By Visual Paradigm Community Edition

Grasp Justifications

System Operation	Class	GRASP Pattern
requestAssignWorktime	ViewScheduleScreen	Expert, Controller
viewSchedule	MainWindow	Expert, Controller
calculateFreeTime	Schedule	Expert
displayFreeTime	Schedule	Expert
createActivity, editActivity, editDeadline	ViewScheduleScreen	Creator, Controller
createProfile	MainWindow	Creator
editProfile	MainWindow	Controller
editScheduleTable	ViewScheduleTableModel	Pure Fabrication, Controller
editPrivacySettings	Group	Expert
addEvent, removeEvent	Controller	Controller
addMembers, removeMembers	Group	Expert, Controller
createTodoList	Schedule	Expert
shareEvent	Schedule	Expert
displayGroupList	User	Expert
confirmSharedEvent, refuseSharedEvent	User	Expert
shareSchedule	Schedule	Expert
viewTimeLeft, viewTimeSpent	Deadline	Expert

Grasp Justifications

- **Operation:** requestAssignWorktime
- **Class:** ViewScheduleScreen
- **GRASP Patterns:** Expert, Controller
- **Justification:** The operation for requesting the worktime assignment is triggered by a button on the schedule viewing screen. This button is contained within and is the responsibility of the ViewScheduleScreen class; therefore, since the operation requires the knowledge of this button, it is reasonable to use the expert pattern as justification for including the operation in the ViewScheduleScreen class. Because this operation is initiated by an input system event, a controller pattern is used to abstract the enacting of the operation from the button press which triggers it, thereby improving maintainability by allowing for the reuse of the controller when the operation needs to be triggered by additional input events.

Grasp Justifications

- **Operations:** createActivity, editActivity, createDeadline, editDeadline
- **Class:** ViewScheduleScreen
- **GRASP Patterns:** Creator, Controller
- **Justification:** The createActivity, editActivity, createDeadline, and editDeadline operations are all require the creation of a CreateEventForm, in either the form of a CreateActivityForm or a CreateDeadlineForm. These operations therefore use the creator pattern by instantiating a class which is responsible for displaying a form to the user and collecting input. The use of the creator pattern in this case increases cohesion by delegating the responsibility for displaying and collecting information from a form to customized classes which are created for this purpose. In the latter part of the operation, the controller pattern is used to insert the information collected from the user into a separate class for information storage, which a controller then uses to display the information in the ViewScheduleScreen class. Using a controller to determine the information in the ViewScheduleScreen instance rather than having each instance include this functionality improves the cohesion of the ViewScheduleScreen class.

Grasp Justifications

- **Operation:** editScheduleTable
- **Class:** ViewScheduleTableModel
- **GRASP Pattern:** Pure Fabrication, Controller
- **Justification:** The pure fabrication pattern justifies the use of the class, ViewScheduleTableModel, which is responsible for editing and retrieving values from the ViewScheduleTable class. Cohesion is increased by encasing all of the functionality for editing and retrieving data from the ViewScheduleTable into one class. Coupling is decreased by allowing methods which only need to edit and retrieve data from the ViewScheduleTable to interact only with the ViewScheduleTableModel class. The controller pattern is used by the ViewScheduleTableModel class to receive data for the schedule from the database. Having a controller be responsible for interfacing between the database and the schedule increases cohesion by unifying the responsibilities for managing interaction between the database and the schedule under one class, and it increases code reuse by condensing the processes for saving and loading to the database into the methods saveLocally and loadLocally.

Grasp Justifications

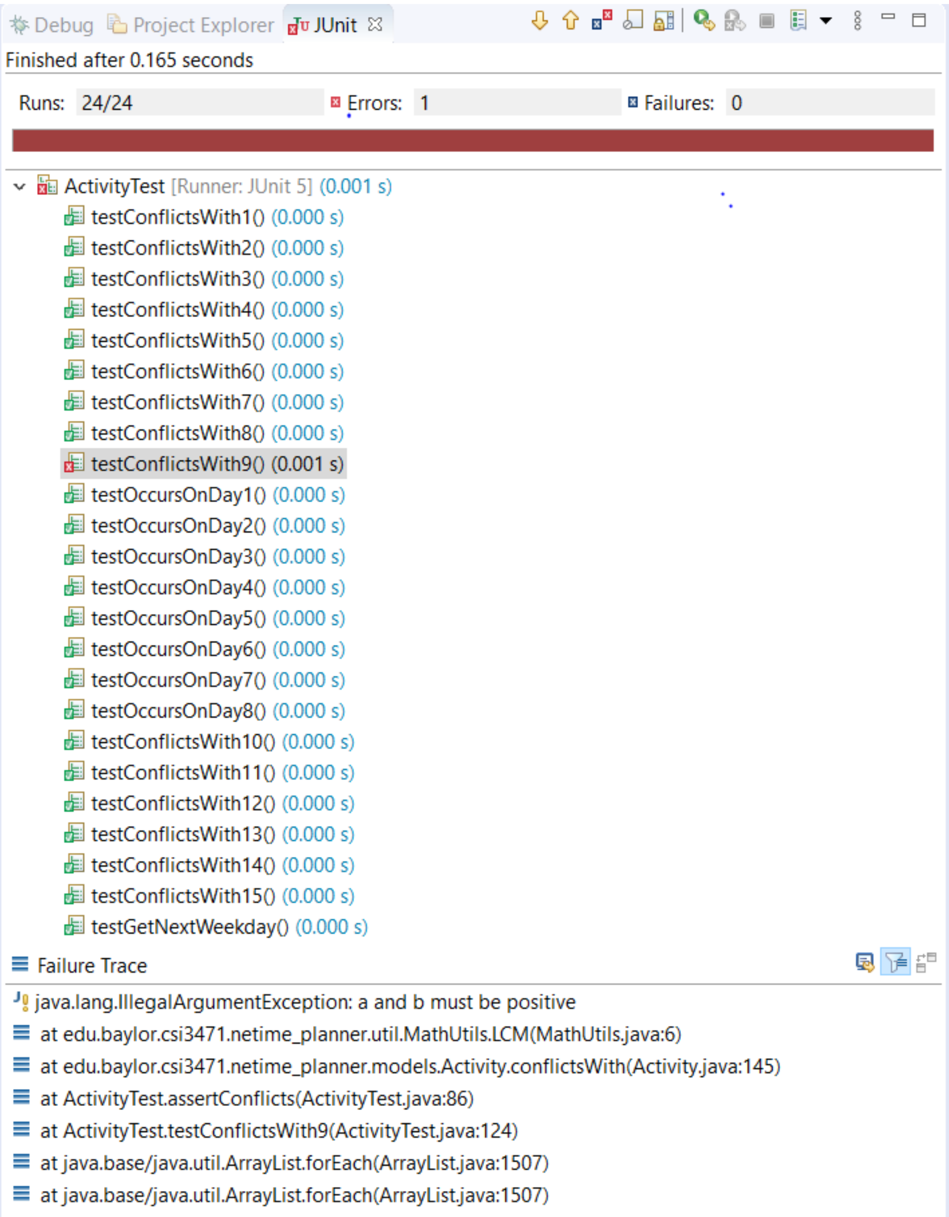
- **Operations:** addEvent, removeEvent
- **Class:** Controller
- **GRASP Pattern:** Controller
- **Justification:** The controller pattern is used to create a layer between the system input events for adding and removing data on the schedule and the execution of these operations. This increases maintainability by making it easier to cause additional system input events to initiate the same controller operation and by also allowing the controller to be extended to handle different kinds of databases without changing the internal logic of the application. When the application calls addEvent and removeEvent, the controller handles how it retrieves these events from the database, which means that different controllers can be created to have their methods fitted to handling a certain type of data source.

Test Coverage Plan

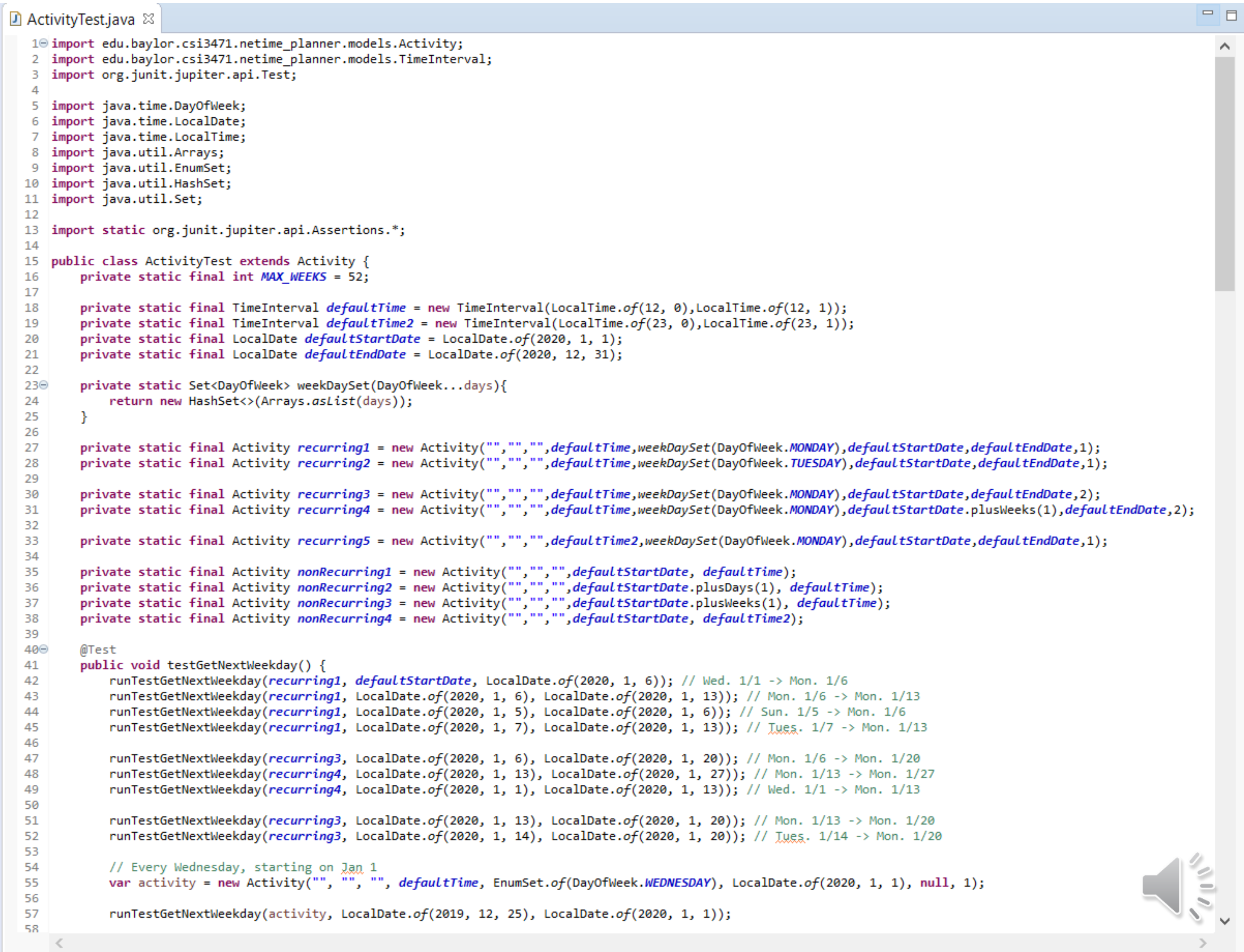
- Activities
  - Test the conflictsWith method, which returns true if the activity conflicts with the inputted activity.
  - Test the occursOnDay method, which returns true if the activity occurs on the inputted day.
- Math Utilities
  - Test the LCM method.
  - Test the GCD method.
  - Creating an array of prime values for Test the LCM and GCD methods.
  - Test whether the prime array has correct values.
- String Utilities
  - Test the usernameToDataFile method.
- Date Utilities
  - Test the getNextWeekDay method.
  - Test the getLastSunday method.
  - Test the weekDaySet method.
- Schedule
  - Test the makeToDoList method.
  - Test the addEvent method.
  - Test the removeEvent method.
  - Test the setWorkTimes method.
- Password Hashing and Login Functionality
  - Test the verifyUsernameAndPassword method.
  - Test the storeUsernameAndPassword method.
- Loading and Saving XML Files
  - Test the saveLocally method.
  - Test the loadLocally method.
- ViewScheduleTableModel
  - Test the add method.
  - Test the remove method.
  - Test the change method.
  - Test the getValueAt method.
  - Test the getRowCount method.
- ViewScheduleScreen
  - Test the showTodoList method.
  - Test the share method.
  - Test the calculateFreeTime method.
  - Test the setWorkTimes method.
  - Test the addActivity method.
  - Test the addDeadline method.
  - Test the save method.
- CreateActivityForm
  - Test the createValue method.

- CreateDeadlineForm
  - Test the createValue method.

Test Coverage Plan

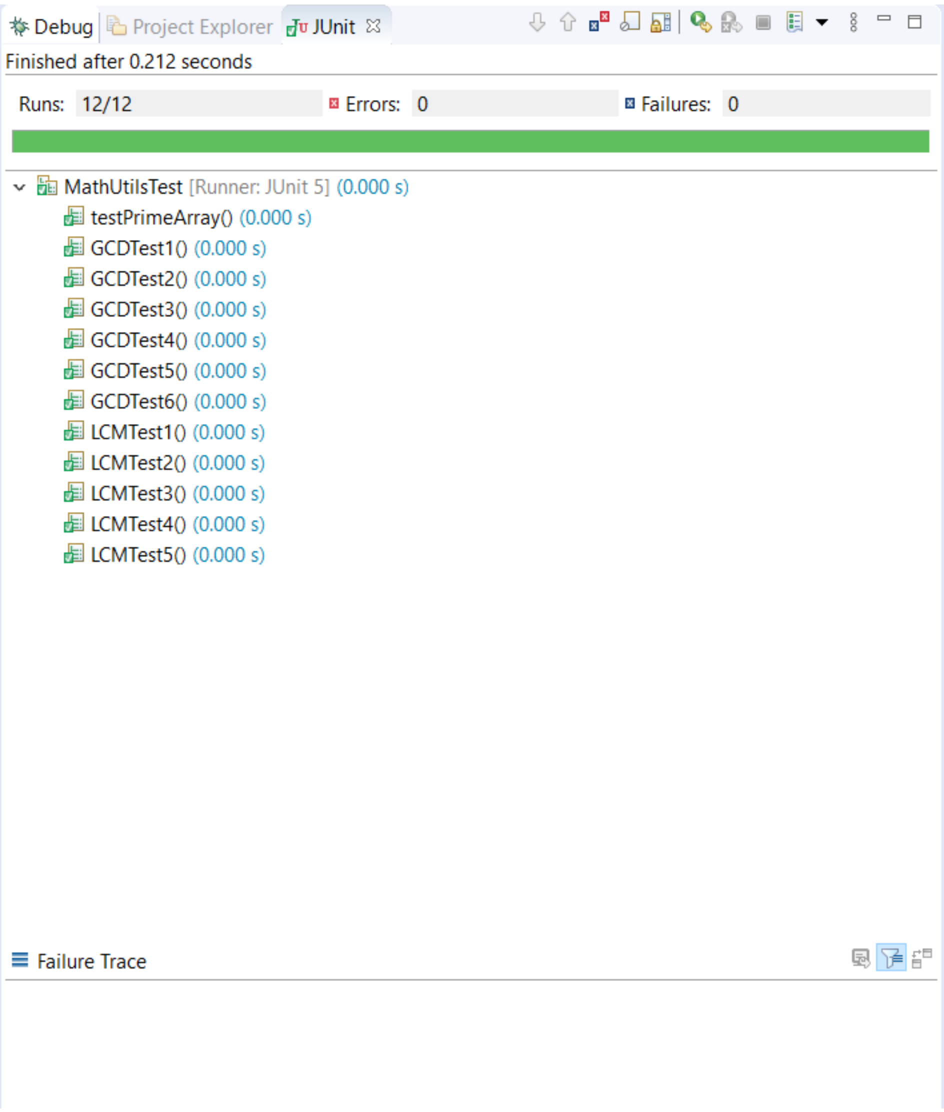


```
Finished after 0.165 seconds

Runs:  24/24        Errors:  1        Failures:  0

ActivityTest [Runner: JUnit 5] (0.001 s)
    testConflictsWith1() (0.000 s)
    testConflictsWith2() (0.000 s)
    testConflictsWith3() (0.000 s)
    testConflictsWith4() (0.000 s)
    testConflictsWith5() (0.000 s)
    testConflictsWith6() (0.000 s)
    testConflictsWith7() (0.000 s)
    testConflictsWith8() (0.000 s)
    testConflictsWith9() (0.001 s)
    testOccursOnDay1() (0.000 s)
    testOccursOnDay2() (0.000 s)
    testOccursOnDay3() (0.000 s)
    testOccursOnDay4() (0.000 s)
    testOccursOnDay5() (0.000 s)
    testOccursOnDay6() (0.000 s)
    testOccursOnDay7() (0.000 s)
    testOccursOnDay8() (0.000 s)
    testConflictsWith10() (0.000 s)
    testConflictsWith11() (0.000 s)
    testConflictsWith12() (0.000 s)
    testConflictsWith13() (0.000 s)
    testConflictsWith14() (0.000 s)
    testConflictsWith15() (0.000 s)
    testGetNextWeekday() (0.000 s)

Failure Trace
java.lang.IllegalArgumentException: a and b must be positive
at edu.baylor.csi3471.netime_planner.util.MathUtils.LCM(MathUtils.java:6)
at edu.baylor.csi3471.netime_planner.models.Activity.conflictsWith(Activity.java:145)
at ActivityTest.assertConflicts(ActivityTest.java:86)
at ActivityTest.testConflictsWith9(ActivityTest.java:124)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1507)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1507)
```



```java
import edu.baylor.csi3471.netime_planner.models.Activity;
import edu.baylor.csi3471.netime_planner.models.TimeInterval;
import org.junit.jupiter.api.Test;

import java.time.DayOfWeek;
import java.time.LocalDate;
import java.time.LocalTime;
import java.util.Arrays;
import java.util.EnumSet;
import java.util.HashSet;
import java.util.Set;

import static org.junit.jupiter.api.Assertions.*;

public class ActivityTest extends Activity {
    private static final int MAX_WEEKS = 52;

    private static final TimeInterval defaultTime = new TimeInterval(LocalTime.of(12, 0),LocalTime.of(12, 1));
    private static final TimeInterval defaultTime2 = new TimeInterval(LocalTime.of(23, 0),LocalTime.of(23, 1));
    private static final LocalDate defaultStartDate = LocalDate.of(2020, 1, 1);
    private static final LocalDate defaultEndDate = LocalDate.of(2020, 12, 31);

    private static Set<DayOfWeek> weekDaySet(DayOfWeek...days){
        return new HashSet<>(Arrays.asList(days));
    }

    private static final Activity recurring1 = new Activity("","","",defaultTime,weekDaySet(DayOfWeek.MONDAY),defaultStartDate,defaultEndDate,1);
    private static final Activity recurring2 = new Activity("","","",defaultTime,weekDaySet(DayOfWeek.TUESDAY),defaultStartDate,defaultEndDate,1);

    private static final Activity recurring3 = new Activity("","","",defaultTime,weekDaySet(DayOfWeek.MONDAY),defaultStartDate,defaultEndDate,2);
    private static final Activity recurring4 = new Activity("","","",defaultTime,weekDaySet(DayOfWeek.MONDAY),defaultStartDate.plusWeeks(1),defaultEndDate,2);

    private static final Activity recurring5 = new Activity("","","",defaultTime2,weekDaySet(DayOfWeek.MONDAY),defaultStartDate,defaultEndDate,1);

    private static final Activity nonRecurring1 = new Activity("","","",defaultStartDate, defaultTime);
    private static final Activity nonRecurring2 = new Activity("","","",defaultStartDate.plusDays(1), defaultTime);
    private static final Activity nonRecurring3 = new Activity("","","",defaultStartDate.plusWeeks(1), defaultTime);
    private static final Activity nonRecurring4 = new Activity("","","",defaultStartDate, defaultTime2);

    @Test
    public void testGetNextWeekday() {
        runTestGetNextWeekday(recurring1, defaultStartDate, LocalDate.of(2020, 1, 6)); // Wed. 1/1 -> Mon. 1/6
        runTestGetNextWeekday(recurring1, LocalDate.of(2020, 1, 6), LocalDate.of(2020, 1, 13)); // Mon. 1/6 -> Mon. 1/13
        runTestGetNextWeekday(recurring1, LocalDate.of(2020, 1, 5), LocalDate.of(2020, 1, 6)); // Sun. 1/5 -> Mon. 1/6
        runTestGetNextWeekday(recurring1, LocalDate.of(2020, 1, 7), LocalDate.of(2020, 1, 13)); // Tues. 1/7 -> Mon. 1/13

        runTestGetNextWeekday(recurring3, LocalDate.of(2020, 1, 6), LocalDate.of(2020, 1, 20)); // Mon. 1/6 -> Mon. 1/20
        runTestGetNextWeekday(recurring4, LocalDate.of(2020, 1, 13), LocalDate.of(2020, 1, 27)); // Mon. 1/13 -> Mon. 1/27
        runTestGetNextWeekday(recurring4, LocalDate.of(2020, 1, 1), LocalDate.of(2020, 1, 13)); // Wed. 1/1 -> Mon. 1/13

        runTestGetNextWeekday(recurring3, LocalDate.of(2020, 1, 13), LocalDate.of(2020, 1, 20)); // Mon. 1/13 -> Mon. 1/20
        runTestGetNextWeekday(recurring3, LocalDate.of(2020, 1, 14), LocalDate.of(2020, 1, 20)); // Tues. 1/14 -> Mon. 1/20

        // Every Wednesday, starting on Jan 1
        var activity = new Activity("", "", "", defaultTime, EnumSet.of(DayOfWeek.WEDNESDAY), LocalDate.of(2020, 1, 1), null, 1);

        runTestGetNextWeekday(activity, LocalDate.of(2019, 12, 25), LocalDate.of(2020, 1, 1));
```

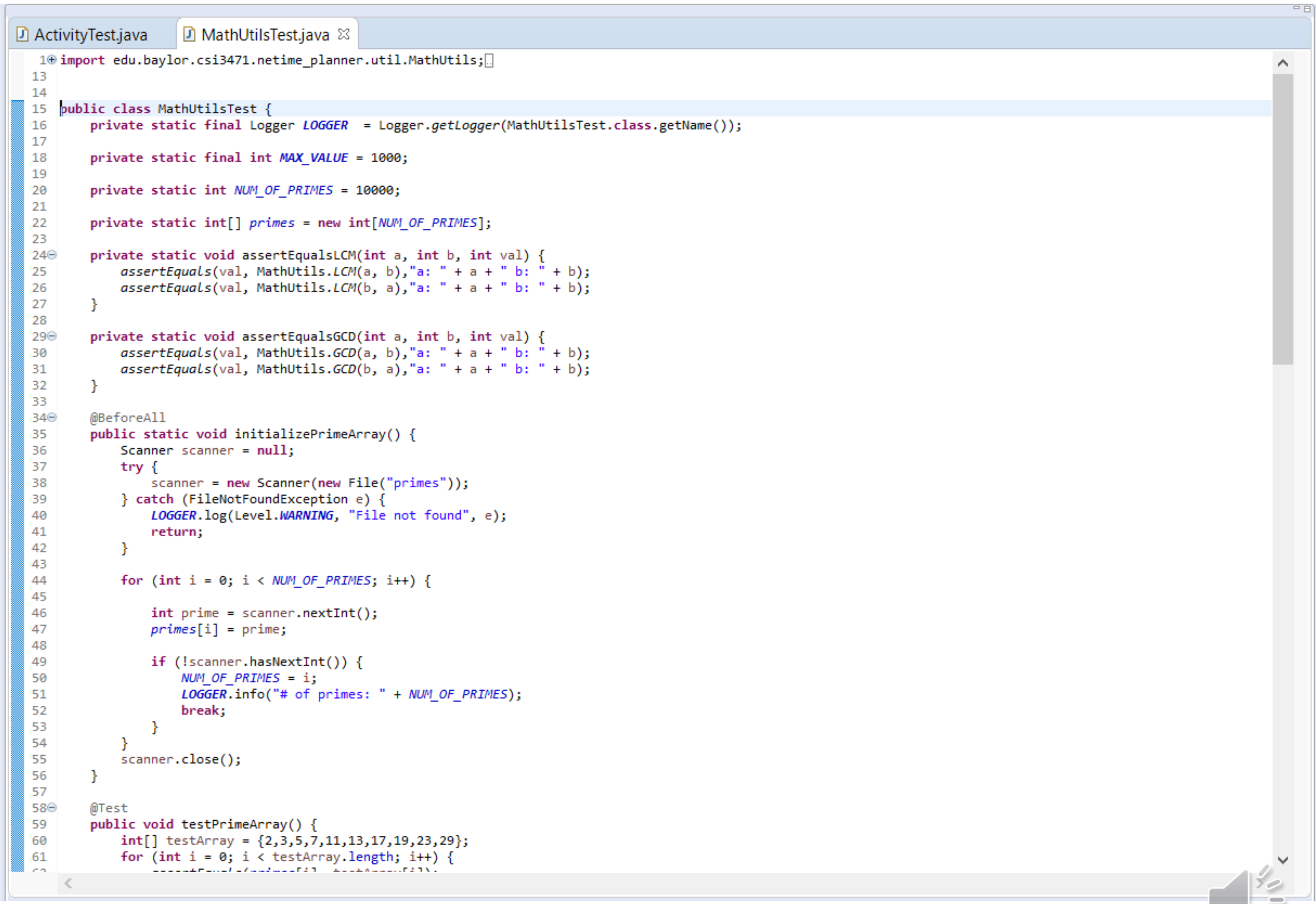
Test Coverage Plan



Debug | Project Explorer | JUnit

```
Finished after 0.212 seconds

Runs: 12/12     Errors: 0     Failures: 0

MathUtilsTest [Runner: JUnit 5] (0.000 s)
    testPrimeArray() (0.000 s)
    GCDTest1() (0.000 s)
    GCDTest2() (0.000 s)
    GCDTest3() (0.000 s)
    GCDTest4() (0.000 s)
    GCDTest5() (0.000 s)
    GCDTest6() (0.000 s)
    LCMTest1() (0.000 s)
    LCMTest2() (0.000 s)
    LCMTest3() (0.000 s)
    LCMTest4() (0.000 s)
    LCMTest5() (0.000 s)
```

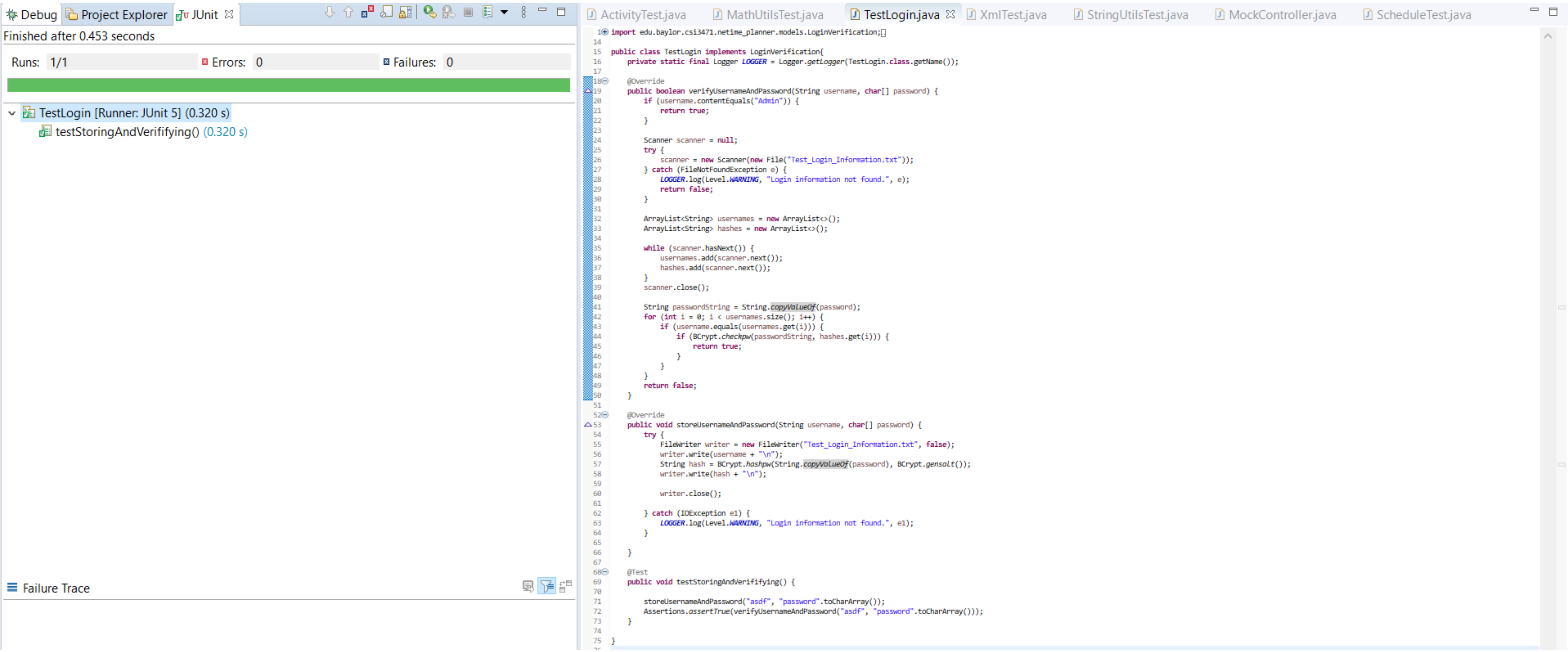
Failure Trace



ActivityTest.java | MathUtilsTest.java

```java
import edu.baylor.csi3471.netime_planner.util.MathUtils;


public class MathUtilsTest {
    private static final Logger LOGGER  = Logger.getLogger(MathUtilsTest.class.getName());

    private static final int MAX_VALUE = 1000;

    private static int NUM_OF_PRIMES = 10000;

    private static int[] primes = new int[NUM_OF_PRIMES];

    private static void assertEqualsLCM(int a, int b, int val) {
        assertEquals(val, MathUtils.LCM(a, b),"a: " + a + " b: " + b);
        assertEquals(val, MathUtils.LCM(b, a),"a: " + a + " b: " + b);
    }

    private static void assertEqualsGCD(int a, int b, int val) {
        assertEquals(val, MathUtils.GCD(a, b),"a: " + a + " b: " + b);
        assertEquals(val, MathUtils.GCD(b, a),"a: " + a + " b: " + b);
    }

    @BeforeAll
    public static void initializePrimeArray() {
        Scanner scanner = null;
        try {
            scanner = new Scanner(new File("primes"));
        } catch (FileNotFoundException e) {
            LOGGER.log(Level.WARNING, "File not found", e);
            return;
        }

        for (int i = 0; i < NUM_OF_PRIMES; i++) {

            int prime = scanner.nextInt();
            primes[i] = prime;

            if (!scanner.hasNextInt()) {
                NUM_OF_PRIMES = i;
                LOGGER.info("# of primes: " + NUM_OF_PRIMES);
                break;
            }
        }
        scanner.close();
    }

    @Test
    public void testPrimeArray() {
        int[] testArray = {2,3,5,7,11,13,17,19,23,29};
        for (int i = 0; i < testArray.length; i++) {
```

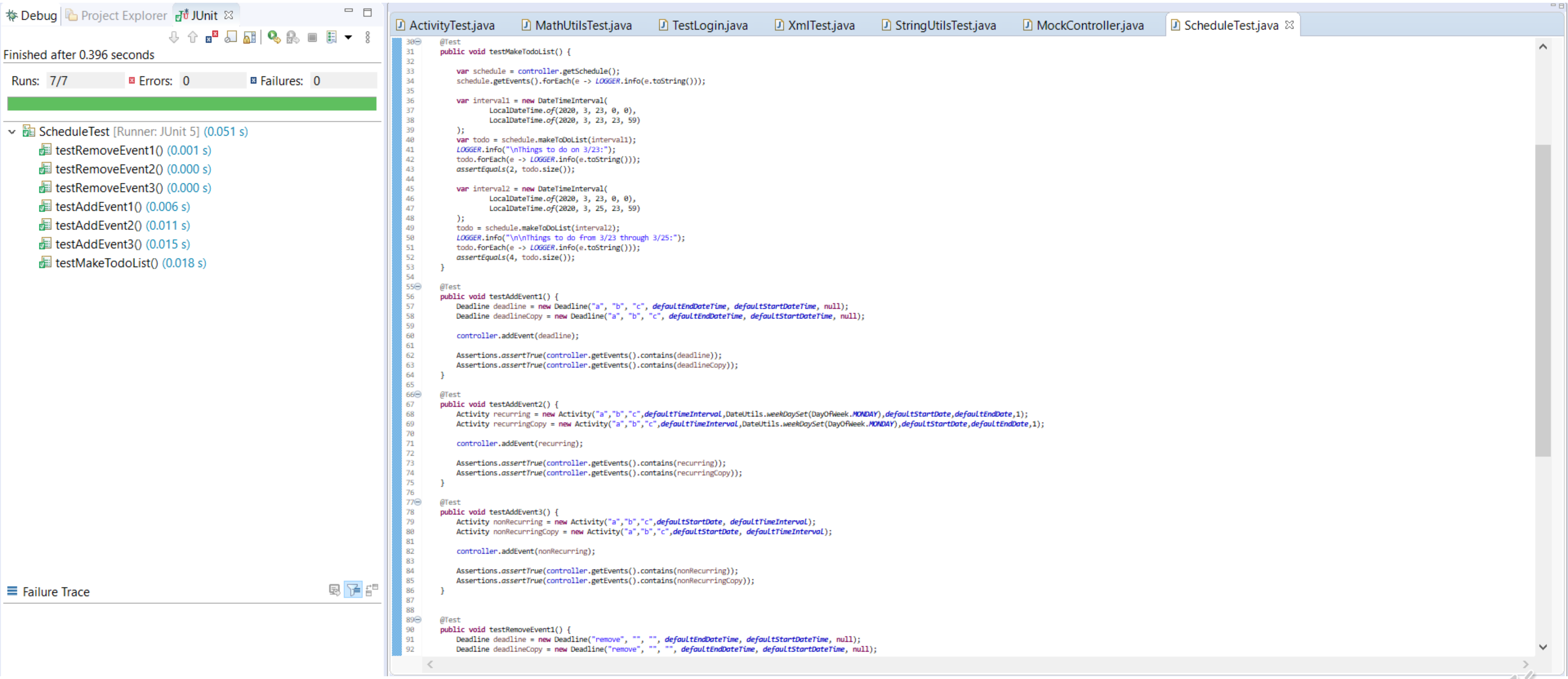
Test Coverage Plan



```
Finished after 0.453 seconds

Runs: 1/1    Errors: 0    Failures: 0

TestLogin [Runner: JUnit 5] (0.320 s)
    testStoringAndVerififying() (0.320 s)
```

```java
import edu.baylor.csi3471.netime_planner.models.LoginVerification;

public class TestLogin implements LoginVerification{
    private static final Logger LOGGER = Logger.getLogger(TestLogin.class.getName());

    @Override
    public boolean verifyUsernameAndPassword(String username, char[] password) {
        if (username.contentEquals("Admin")) {
            return true;
        }

        Scanner scanner = null;
        try {
            scanner = new Scanner(new File("Test_Login_Information.txt"));
        } catch (FileNotFoundException e) {
            LOGGER.log(Level.WARNING, "Login information not found.", e);
            return false;
        }

        ArrayList<String> usernames = new ArrayList<>();
        ArrayList<String> hashes = new ArrayList<>();

        while (scanner.hasNext()) {
            usernames.add(scanner.next());
            hashes.add(scanner.next());
        }
        scanner.close();

        String passwordString = String.copyValueOf(password);
        for (int i = 0; i < usernames.size(); i++) {
            if (username.equals(usernames.get(i))) {
                if (BCrypt.checkpw(passwordString, hashes.get(i))) {
                    return true;
                }
            }
        }
        return false;
    }

    @Override
    public void storeUsernameAndPassword(String username, char[] password) {
        try {
            FileWriter writer = new FileWriter("Test_Login_Information.txt", false);
            writer.write(username + "\n");
            String hash = BCrypt.hashpw(String.copyValueOf(password), BCrypt.gensalt());
            writer.write(hash + "\n");

            writer.close();

        } catch (IOException e1) {
            LOGGER.log(Level.WARNING, "Login information not found.", e1);
        }

    }

    @Test
    public void testStoringAndVerififying() {

        storeUsernameAndPassword("asdf", "password".toCharArray());
        Assertions.assertTrue(verifyUsernameAndPassword("asdf", "password".toCharArray()));
    }

}
```

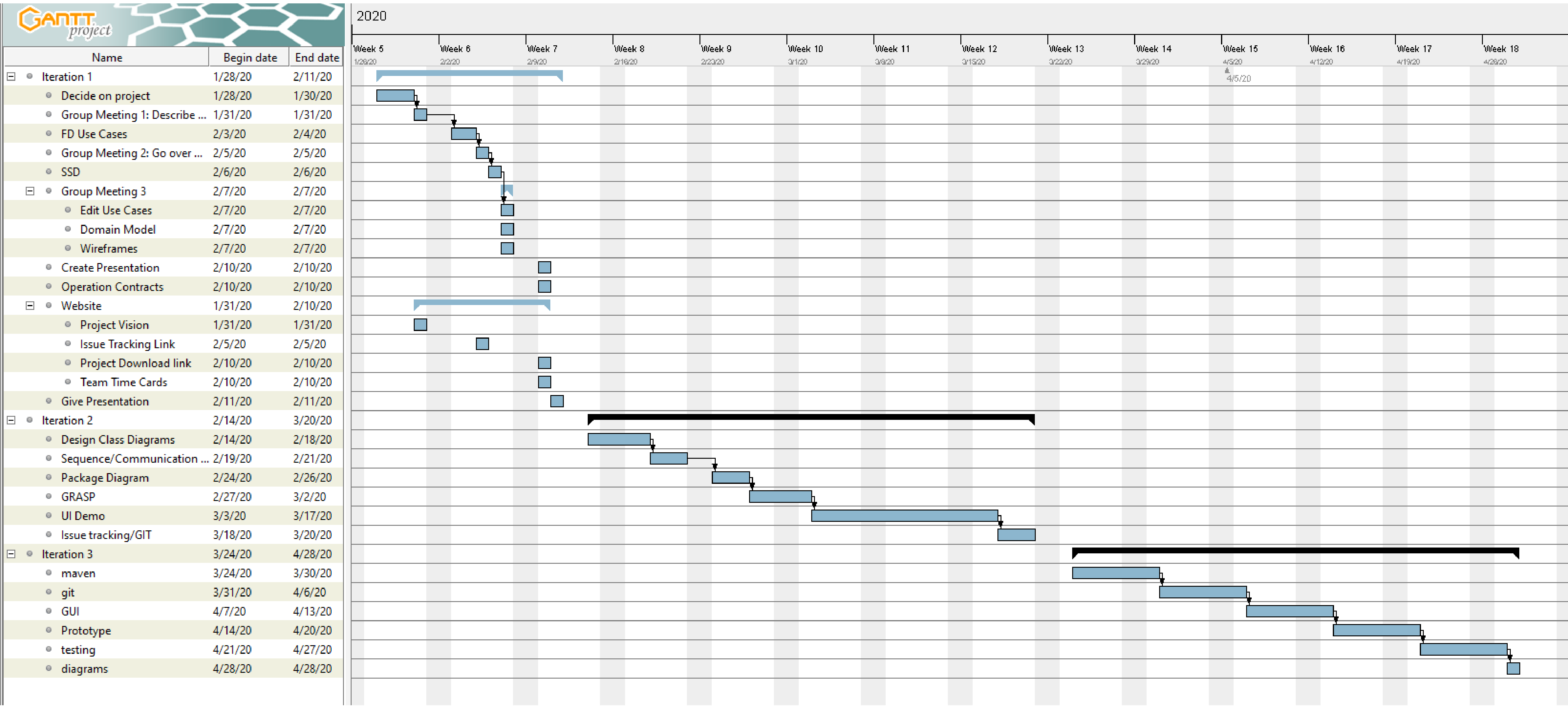
Test Coverage Plan



```
Finished after 0.396 seconds

Runs: 7/7     Errors: 0     Failures: 0

ScheduleTest [Runner: JUnit 5] (0.051 s)
    testRemoveEvent1() (0.001 s)
    testRemoveEvent2() (0.000 s)
    testRemoveEvent3() (0.000 s)
    testAddEvent1() (0.006 s)
    testAddEvent2() (0.011 s)
    testAddEvent3() (0.015 s)
    testMakeTodoList() (0.018 s)
```

```java
    @Test
    public void testMakeTodoList() {

        var schedule = controller.getSchedule();
        schedule.getEvents().forEach(e -> LOGGER.info(e.toString()));

        var interval1 = new DateTimeInterval(
                LocalDateTime.of(2020, 3, 23, 0, 0),
                LocalDateTime.of(2020, 3, 23, 23, 59)
        );
        var todo = schedule.makeToDoList(interval1);
        LOGGER.info("\nThings to do on 3/23:");
        todo.forEach(e -> LOGGER.info(e.toString()));
        assertEquals(2, todo.size());

        var interval2 = new DateTimeInterval(
                LocalDateTime.of(2020, 3, 23, 0, 0),
                LocalDateTime.of(2020, 3, 25, 23, 59)
        );
        todo = schedule.makeToDoList(interval2);
        LOGGER.info("\n\nThings to do from 3/23 through 3/25:");
        todo.forEach(e -> LOGGER.info(e.toString()));
        assertEquals(4, todo.size());
    }

    @Test
    public void testAddEvent1() {
        Deadline deadline = new Deadline("a", "b", "c", defaultEndDateTime, defaultStartDateTime, null);
        Deadline deadlineCopy = new Deadline("a", "b", "c", defaultEndDateTime, defaultStartDateTime, null);

        controller.addEvent(deadline);

        Assertions.assertTrue(controller.getEvents().contains(deadline));
        Assertions.assertTrue(controller.getEvents().contains(deadlineCopy));
    }

    @Test
    public void testAddEvent2() {
        Activity recurring = new Activity("a","b","c",defaultTimeInterval,DateUtils.weekDaySet(DayOfWeek.MONDAY),defaultStartDate,defaultEndDate,1);
        Activity recurringCopy = new Activity("a","b","c",defaultTimeInterval,DateUtils.weekDaySet(DayOfWeek.MONDAY),defaultStartDate,defaultEndDate,1);

        controller.addEvent(recurring);

        Assertions.assertTrue(controller.getEvents().contains(recurring));
        Assertions.assertTrue(controller.getEvents().contains(recurringCopy));
    }

    @Test
    public void testAddEvent3() {
        Activity nonRecurring = new Activity("a","b","c",defaultStartDate, defaultTimeInterval);
        Activity nonRecurringCopy = new Activity("a","b","c",defaultStartDate, defaultTimeInterval);

        controller.addEvent(nonRecurring);

        Assertions.assertTrue(controller.getEvents().contains(nonRecurring));
        Assertions.assertTrue(controller.getEvents().contains(nonRecurringCopy));
    }


    @Test
    public void testRemoveEvent1() {
        Deadline deadline = new Deadline("remove", "", "", defaultEndDateTime, defaultStartDateTime, null);
        Deadline deadlineCopy = new Deadline("remove", "", "", defaultEndDateTime, defaultStartDateTime, null);
```

Gantt Diagram



Name	Begin date	End date
Iteration 1	1/28/20	2/11/20
Decide on project	1/28/20	1/30/20
Group Meeting 1: Describe ...	1/31/20	1/31/20
FD Use Cases	2/3/20	2/4/20
Group Meeting 2: Go over ...	2/5/20	2/5/20
SSD	2/6/20	2/6/20
Group Meeting 3	2/7/20	2/7/20
Edit Use Cases	2/7/20	2/7/20
Domain Model	2/7/20	2/7/20
Wireframes	2/7/20	2/7/20
Create Presentation	2/10/20	2/10/20
Operation Contracts	2/10/20	2/10/20
Website	1/31/20	2/10/20
Project Vision	1/31/20	1/31/20
Issue Tracking Link	2/5/20	2/5/20
Project Download link	2/10/20	2/10/20
Team Time Cards	2/10/20	2/10/20
Give Presentation	2/11/20	2/11/20
Iteration 2	2/14/20	3/20/20
Design Class Diagrams	2/14/20	2/18/20
Sequence/Communication ...	2/19/20	2/21/20
Package Diagram	2/24/20	2/26/20
GRASP	2/27/20	3/2/20
UI Demo	3/3/20	3/17/20
Issue tracking/GIT	3/18/20	3/20/20
Iteration 3	3/24/20	4/28/20
maven	3/24/20	3/30/20
git	3/31/20	4/6/20
GUI	4/7/20	4/13/20
Prototype	4/14/20	4/20/20
testing	4/21/20	4/27/20
diagrams	4/28/20	4/28/20