# NETime Planner

**Project Authors:**

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

# NETime Planner Time Card

| Name | Hours Spent |
|---|---|
| Joshua Kanagasabai | 33 (18 meetings, 15 non-meetings) |
| Kevin DeMars | 52 (18.5 meetings, 33.5 non-meetings) |
| Trenton Strickland | 30 (10 meetings, 20 non-meetings) |
| Samuel Kim | 36 (18.5 meetings, 17.5 non-meetings) |
| Eric Jaroszewski | 28 (16 meetings, 12 non-meetings) |

# Suggested Point Distributions

We’re fine with everyone getting the same grade, and an equal point distribution.

# NETime Planner

Apr 2, 2020

http://

## CSI 3471

| | |
|---|---|
| **Project manager** | |
| **Project dates** | Jan 28, 2020 - Apr 29, 2020 |
| **Completion** | 0% |
| **Tasks** | 32 |
| **Resources** | 6 |

Gantt Diagram for first iteration of software project.

# Tasks

| 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 Use Cases | 1/31/20 | 1/31/20 |
| FD Use Cases | 2/3/20 | 2/4/20 |
| Group Meeting 2: Go over Use Cases | 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 Diagrams | 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 |

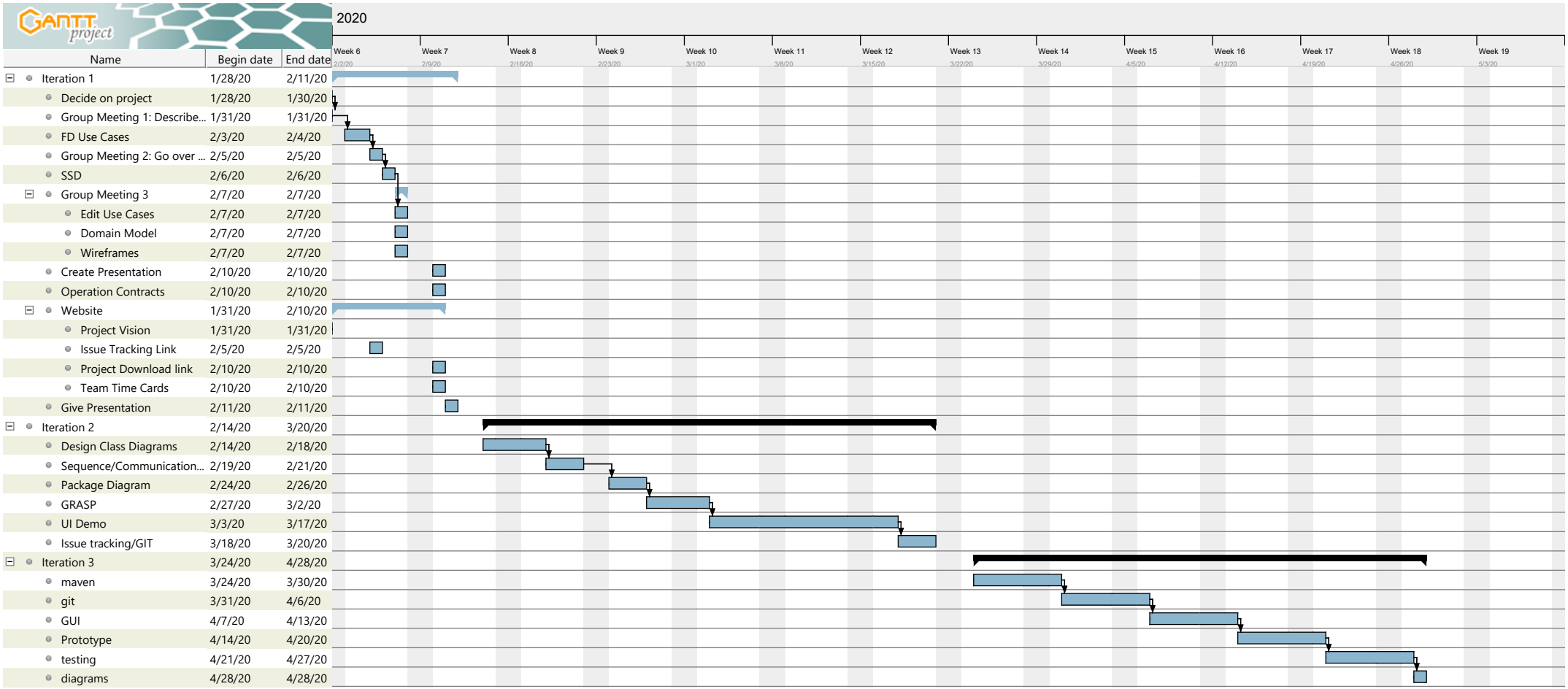
# Tasks

| Name | Begin date | End date |
|---|---|---|
| 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 |

# Resources

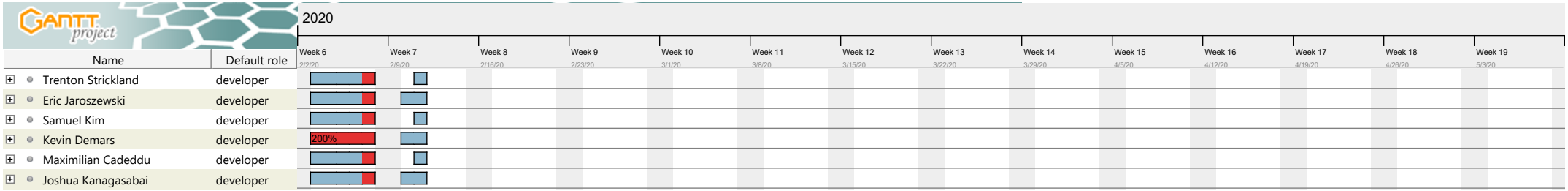
| Name | Default role |
| --- | --- |
| Trenton Strickland | developer |
| Eric Jaroszewski | developer |
| Samuel Kim | developer |
| Kevin Demars | developer |
| Maximilian Cadeddu | developer |
| Joshua Kanagasabai | developer |

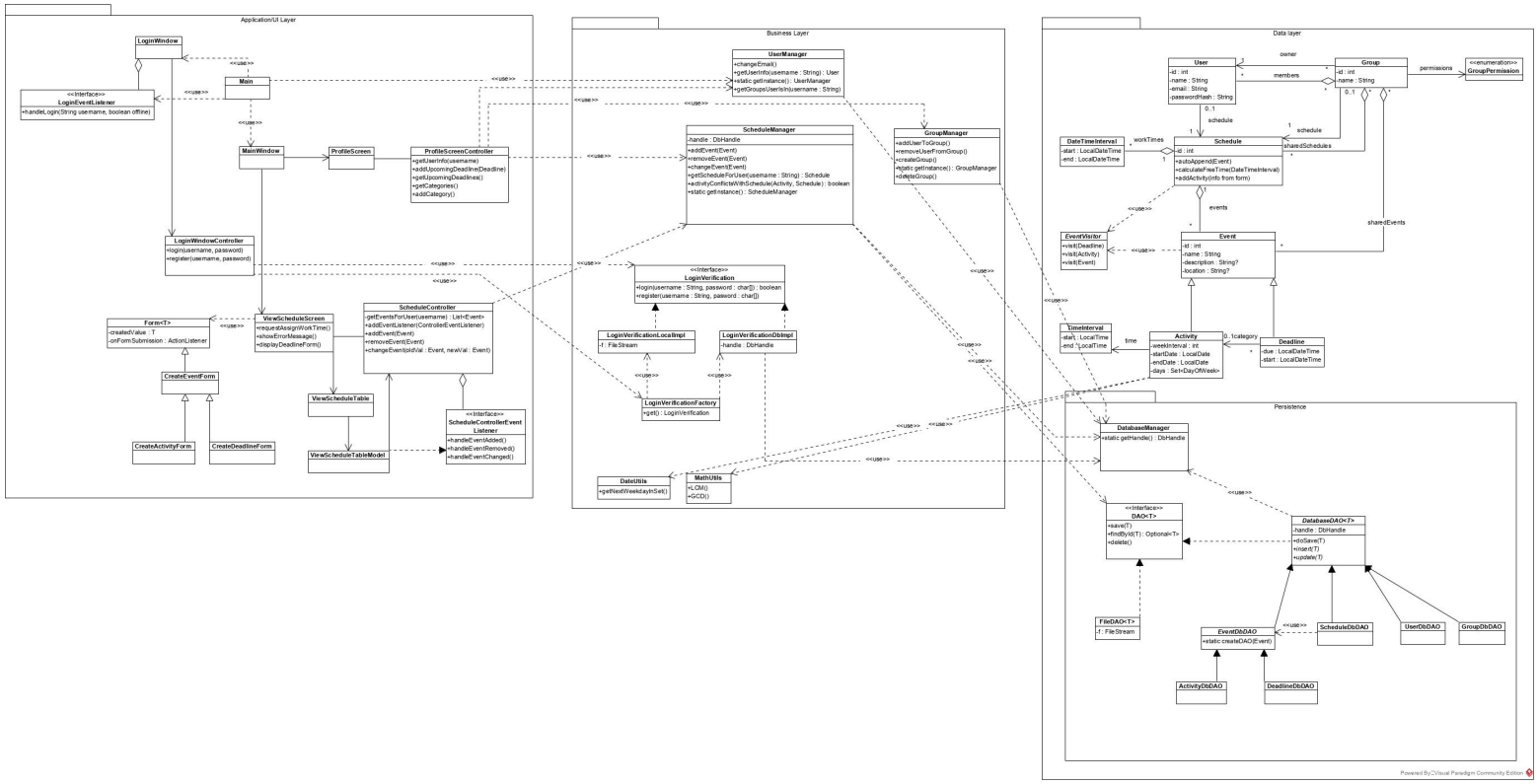
# Gantt Chart

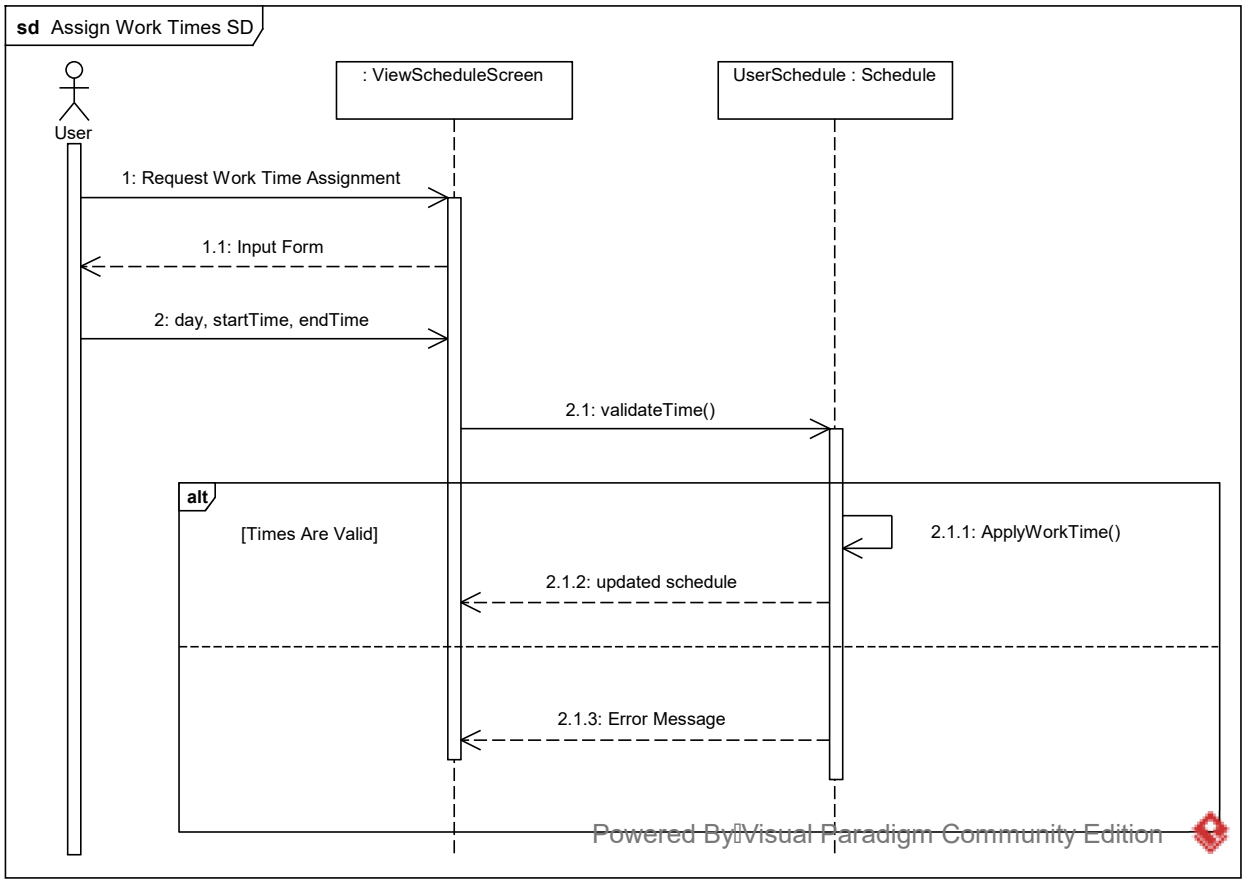


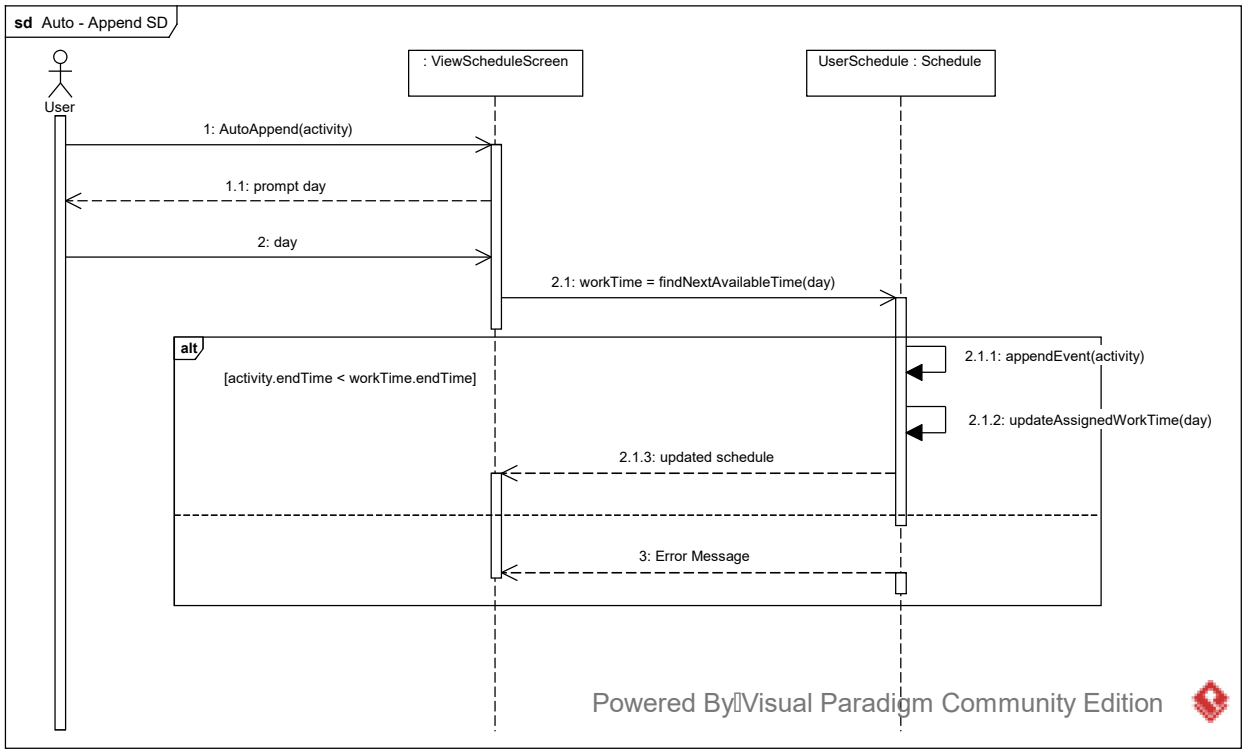

| Name | Begin date | End date |
|---|---|---|
| Iteration 1 | 1/28/20 | 2/11/20 |
| Decide on project | 1/28/20 | 1/30/20 |
| Group Meeting 1: Describe... | 1/31/20 | 1/31/20 |
| FD Use Cases | 2/3/20 | 2/4/20 |
| Group Meeting 2: Go over ... | 2/5/20 | 2/5/20 |
| SSD | 2/6/20 | 2/6/20 |
| Group Meeting 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 |

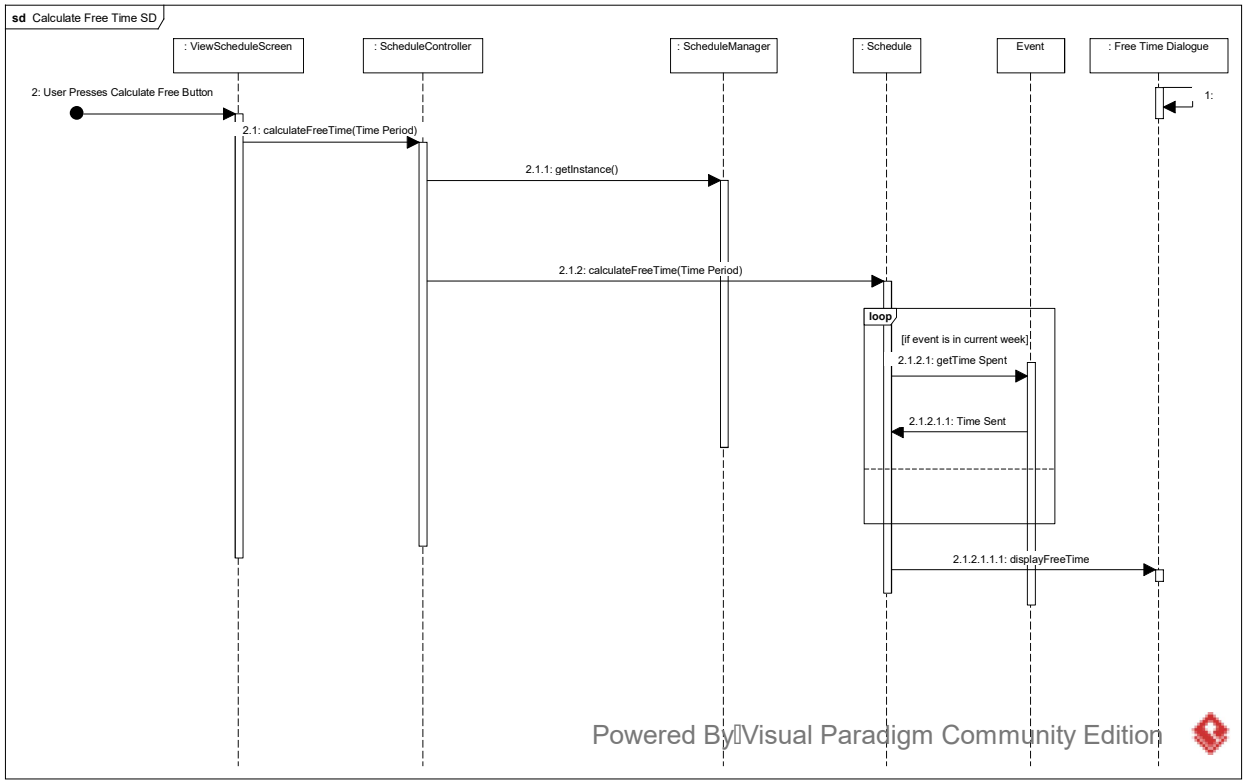
# Resources Chart

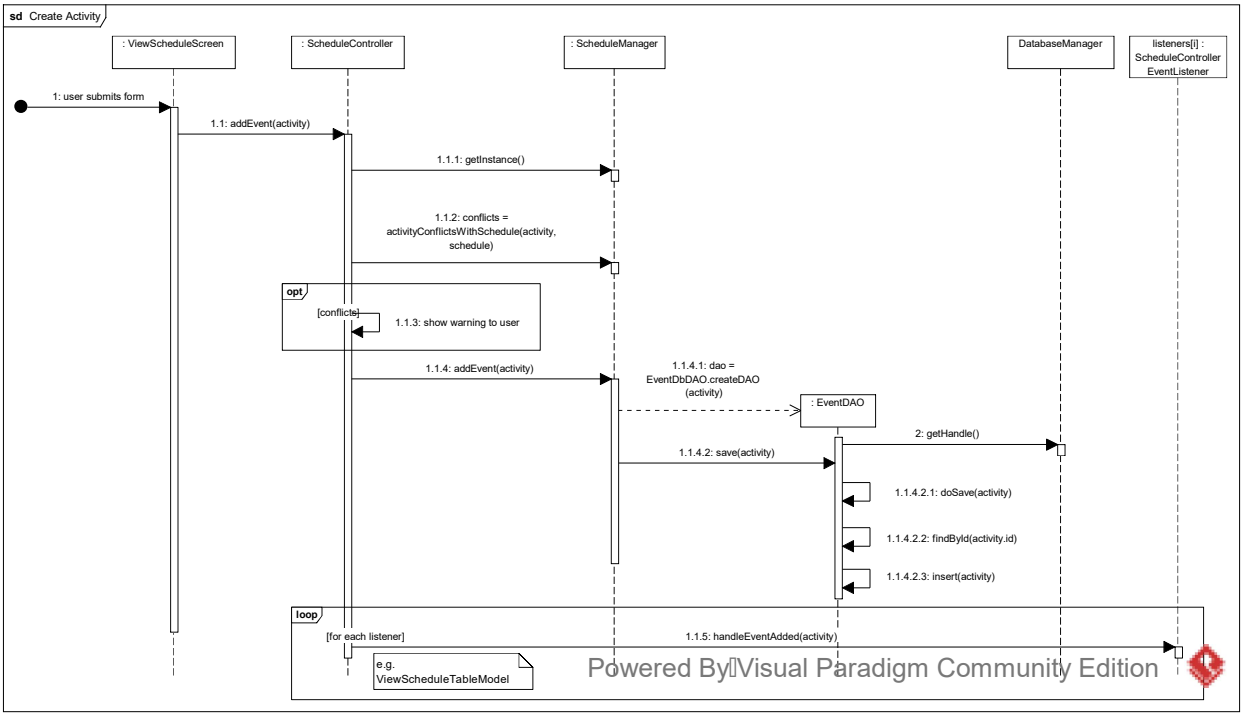


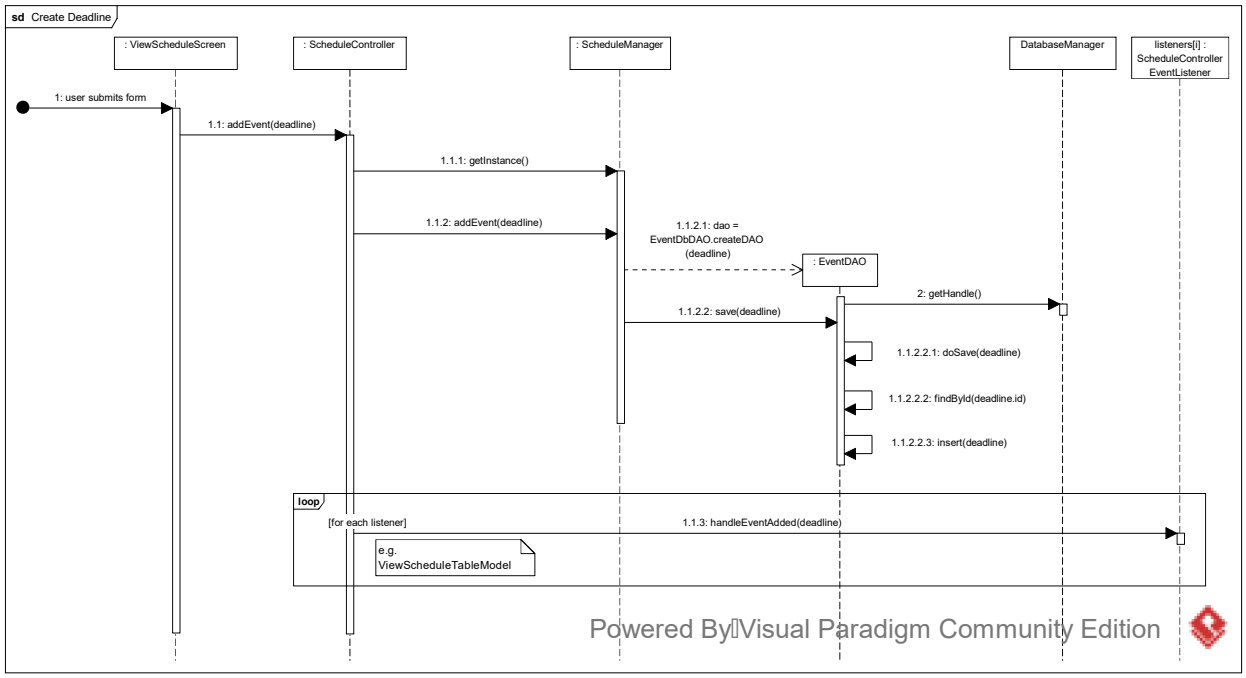

Application/UI Layer
LoginWindow
<<Interface>>
LoginEventListener
+handleLogin(String username, boolean offline)
Main
MainWindow
ProfileScreen
ProfileScreenController
+getUserInfo(username)
+addUpcomingDeadline(Deadline)
+getUpcomingDeadlines()
+getCategories()
+addCategory()
LoginWindowController
+login(username, password)
+register(username, password)
FormcT>
-createdValue : T
-onFormSubmission : ActionListener
ViewScheduleScreen
+requestAssignWorkTime()
+showErrorMessage()
+displayDeadlineForm()
ScheduleController
-getEventsForUser(username) : List<Event>
+addEventListener(ControllerEventListener)
+addEvent(Event)
+removeEvent(Event)
+changeEventId(oldVal : Event, newVal : Event)
CreateEventForm
CreateActivityForm
CreateDeadlineForm
ViewScheduleTable
ViewScheduleTableModel
<<Interface>>
ScheduleControllerEvent
Listener
+handleEventAdded()
+handleEventRemoved()
+handleEventChanged()
Business Layer
UserManager
+changeEmail()
+getUserInfo(username : String) : User
+static getInstance() : UserManager
+getGroupsUserIsIn(username : String)
ScheduleManager
-handle : DbHandle
+addEvent(Event)
+removeEvent(Event)
+changeEvent(Event)
+getScheduleForUser(username : String) : Schedule
+activityConflictsWithSchedule(Activity, Schedule) : boolean
+static getInstance() : ScheduleManager
GroupManager
+addUserToGroup()
+removeUserFromGroup()
+createGroup()
+static getInstance() : GroupManager
+deleteGroup()
<<Interface>>
LoginVerification
+login(username : String, password : char[]) : boolean
+register(username : String, paswword : char[])
LoginVerificationLocalImpl
-f : FileStream
LoginVerificationDbImpl
-handle : DbHandle
LoginVerificationFactory
+get() : LoginVerification
DateUtils
+getNextWeekdaysInSet()
MathUtils
+LCM()
+GCD()
Data layer
User
-id : int
-name : String
-email : String
-passwordHash : String
Group
-id : int
-name : String
<<enumeration>>
GroupPermission
DateTimeInterval
-start : LocalDateTime
-end : LocalDateTime
Schedule
-id : int
+autoAppendEvent()
+calculateFreeTime(DateTimeInterval)
+addActivity(info from form)
EventVisitor
+visit(Deadline)
+visit(Activity)
+visit(Event)
Event
-id : int
-name : String
-description : String?
-location : String?
TimeInterval
-start : LocalTime
-end : LocalTime
Activity
-weekInterval : int
-startDate : LocalDate
-endDate : LocalDate
-days : Set<DayOfWeek>
Deadline
-due : LocalDateTime
-start : LocalDateTime
Persistence
DatabaseManager
+static getHandle() : DbHandle
<<Interface>>
DAO<T>
+save(T)
+findById(T) : Optional<T>
+delete()
DatabaseDAO<T>
-handle : DbHandle
+doSave(T)
+insert(T)
+update(T)
FileDAO<T>
-f : FileStream
EventDbDAO
+static createDAO(Event)
ScheduleDbDAO
UserDbDAO
GroupDbDAO
ActivityDbDAO
DeadlineDbDAO
Powered By:Visual Paradigm Community Edition

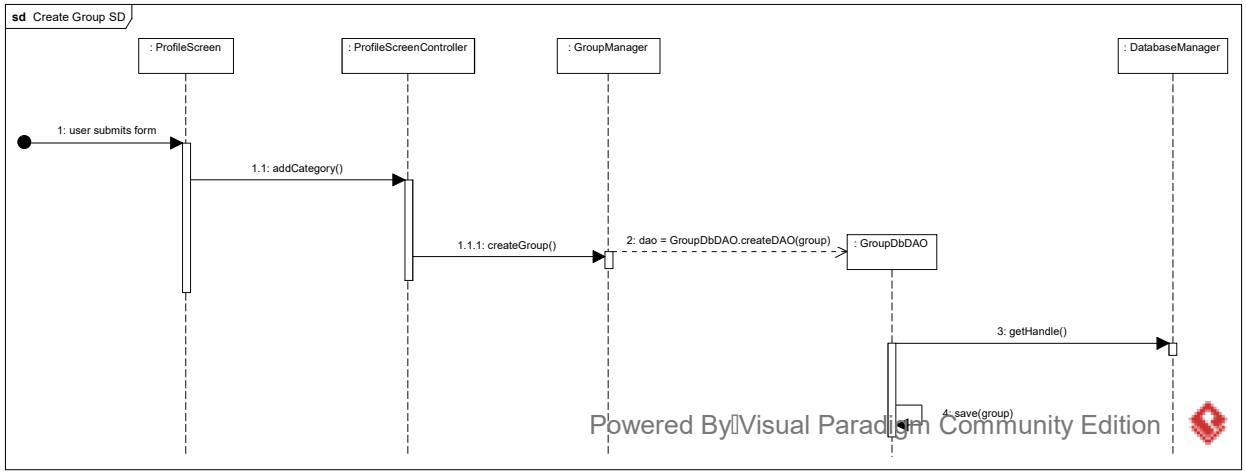

sd Assign Work Times SD
User
: ViewScheduleScreen
UserSchedule : Schedule
1: Request Work Time Assignment
1.1: Input Form
2: day, startTime, endTime
2.1: validateTime()
alt
[Times Are Valid]
2.1.1: ApplyWorkTime()
2.1.2: updated schedule
2.1.3: Error Message
Powered By Visual Paradigm Community Edition

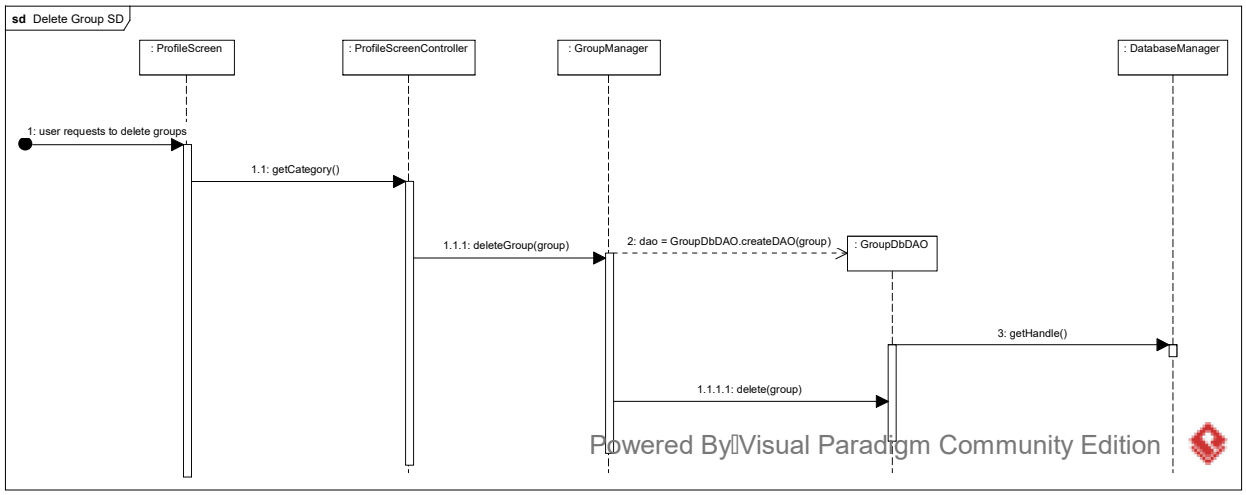

sd Auto - Append SD
User
: ViewScheduleScreen
UserSchedule : Schedule
1: AutoAppend(activity)
1.1: prompt day
2: day
2.1: workTime = findNextAvailableTime(day)
alt
[activity.endTime < workTime.endTime]
2.1.1: appendEvent(activity)
2.1.2: updateAssignedWorkTime(day)
2.1.3: updated schedule
3: Error Message
Powered By Visual Paradigm Community Edition

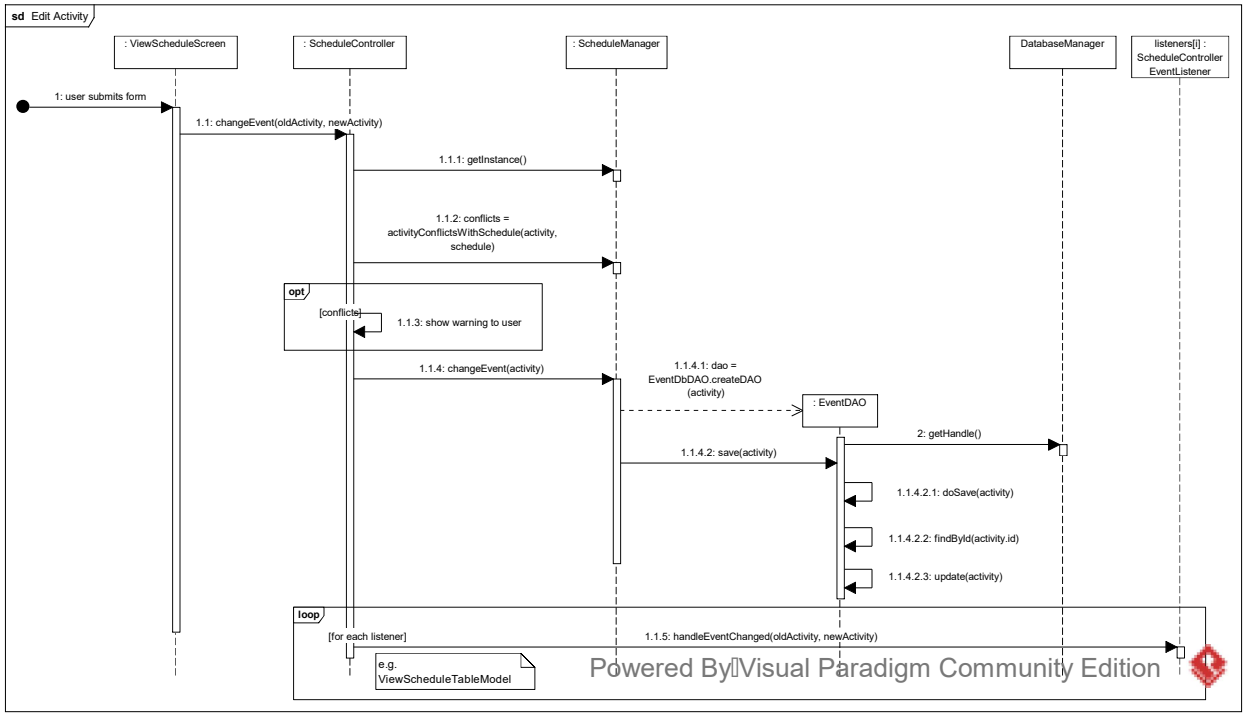

sd Calculate Free Time SD
: ViewScheduleScreen
: ScheduleController
: ScheduleManager
: Schedule
Event
: Free Time Dialogue
2: User Presses Calculate Free Button
1:
2.1: calculateFreeTime(Time Period)
2.1.1: getInstance()
2.1.2: calculateFreeTime(Time Period)
loop
[if event is in current week]
2.1.2.1: getTime Spent
2.1.2.1.1: Time Sent
2.1.2.1.1.1: displayFreeTime
Powered By Visual Paradigm Community Edition

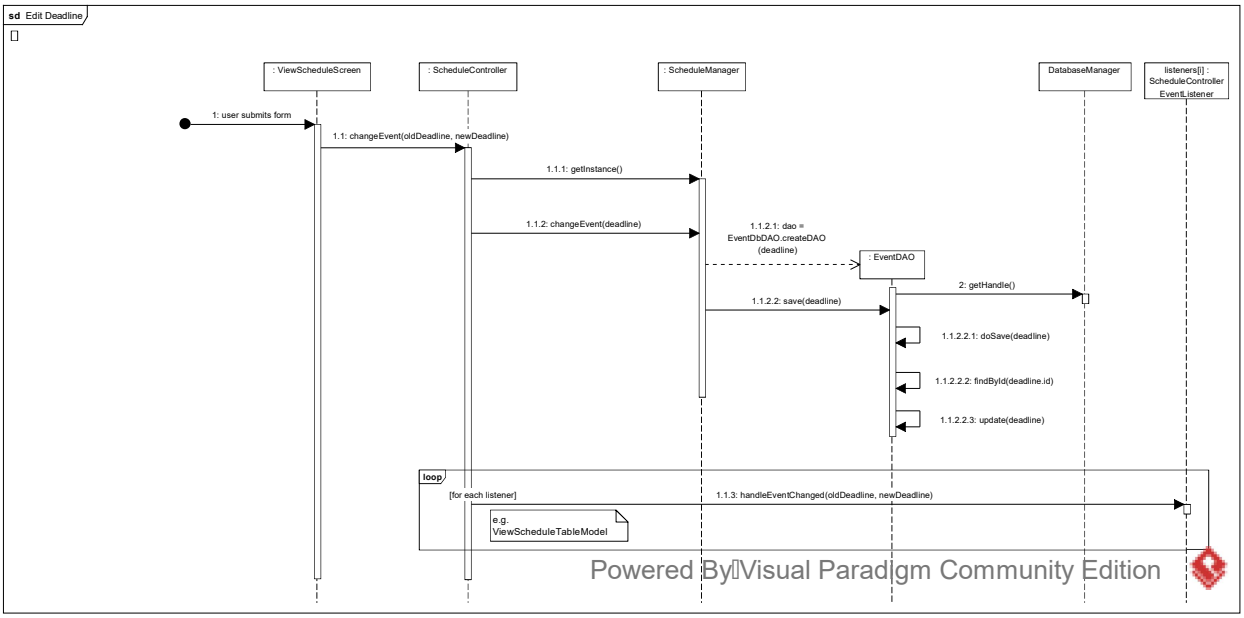

sd Create Activity
: ViewScheduleScreen
: ScheduleController
: ScheduleManager
DatabaseManager
listeners[i] : ScheduleController EventListener
1: user submits form
1.1: addEvent(activity)
1.1.1: getInstance()
1.1.2: conflicts = activityConflictsWithSchedule(activity, schedule)
opt
[conflicts]
1.1.3: show warning to user
1.1.4: addEvent(activity)
1.1.4.1: dao = EventDbDAO.createDAO (activity)
: EventDAO
2: getHandle()
1.1.4.2: save(activity)
1.1.4.2.1: doSave(activity)
1.1.4.2.2: findById(activity.id)
1.1.4.2.3: insert(activity)
loop
[for each listener]
1.1.5: handleEventAdded(activity)
e.g. ViewScheduleTableModel
Powered By Visual Paradigm Community Edition

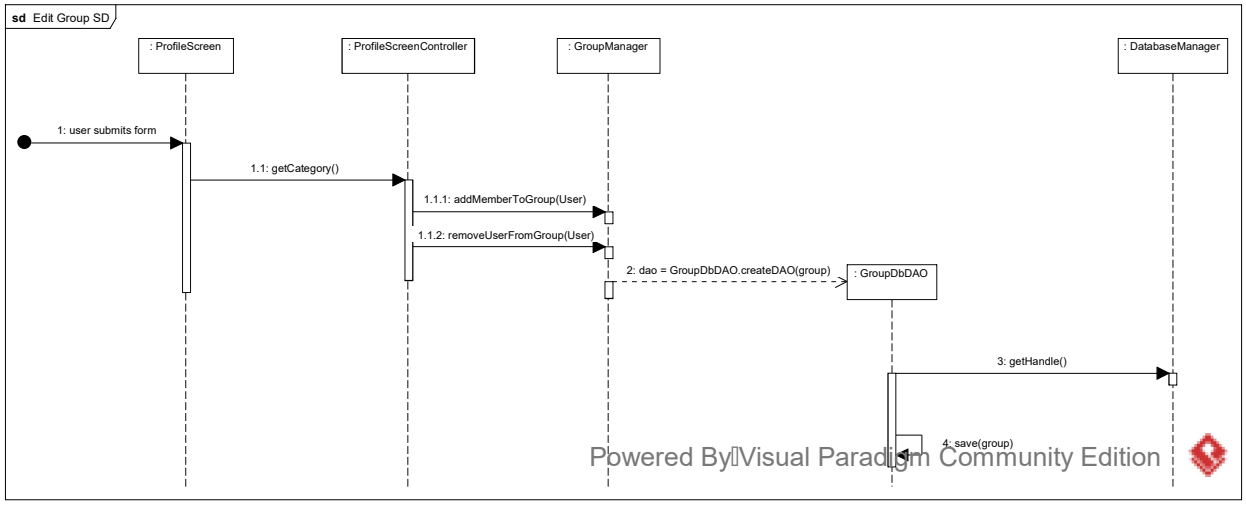

sd Create Deadline
: ViewScheduleScreen
: ScheduleController
: ScheduleManager
DatabaseManager
listeners[i] : ScheduleController EventListener
1: user submits form
1.1: addEvent(deadline)
1.1.1: getInstance()
1.1.2: addEvent(deadline)
1.1.2.1: dao = EventDbDAO.createDAO (deadline)
: EventDAO
2: getHandle()
1.1.2.2: save(deadline)
1.1.2.2.1: doSave(deadline)
1.1.2.2.2: findById(deadline.id)
1.1.2.2.3: insert(deadline)
loop
[for each listener]
1.1.3: handleEventAdded(deadline)
e.g. ViewScheduleTableModel
Powered By Visual Paradigm Community Edition

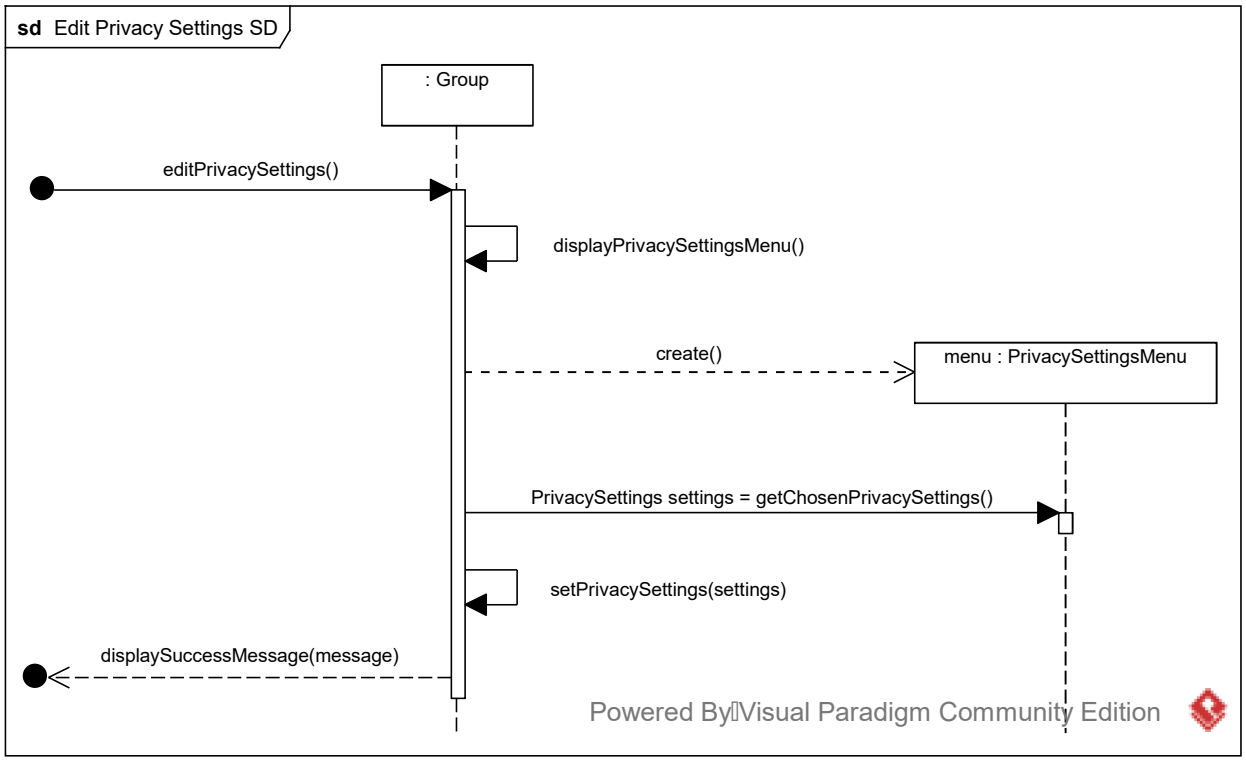

sd Create Group SD
: ProfileScreen
: ProfileScreenController
: GroupManager
: DatabaseManager
1: user submits form
1.1: addCategory()
1.1.1: createGroup()
2: dao = GroupDbDAO.createDAO(group)
: GroupDbDAO
3: getHandle()
4: save(group)
Powered By Visual Paradigm Community Edition

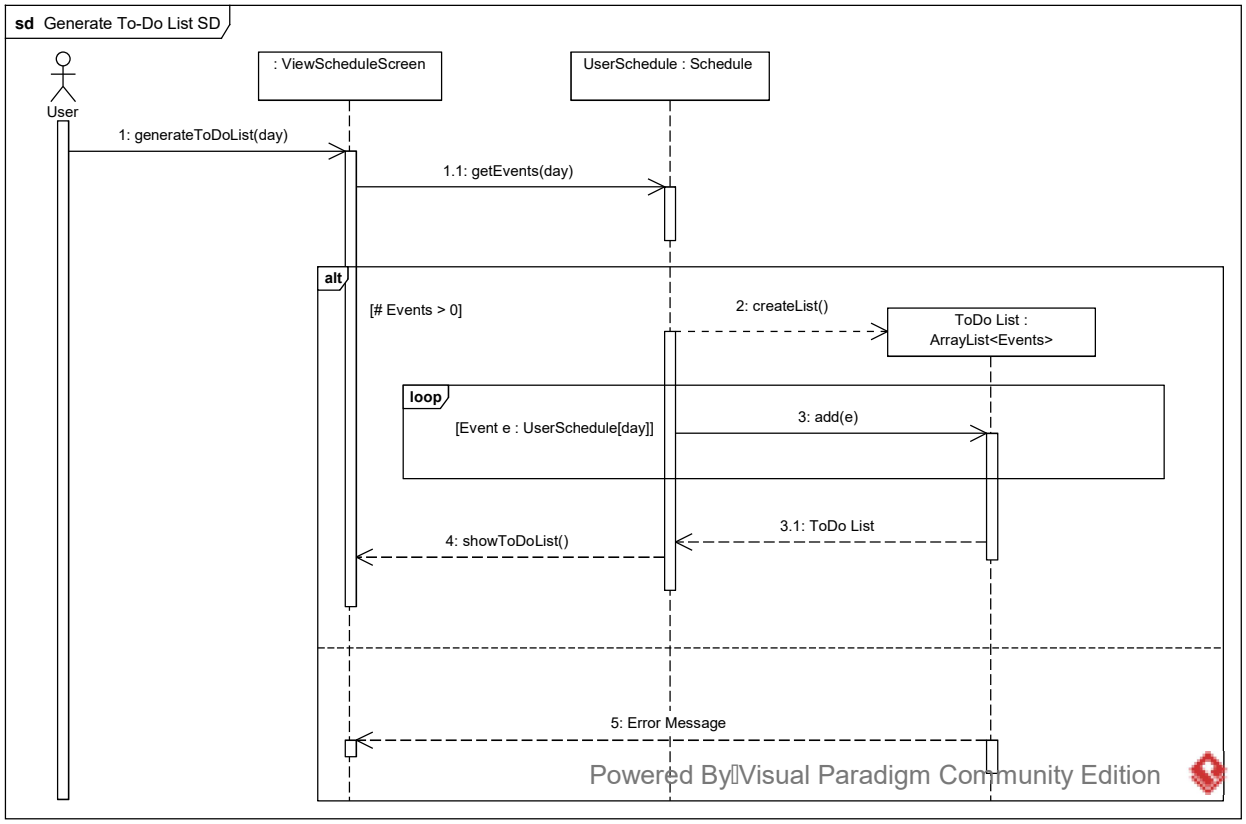

sd Delete Group SD
: ProfileScreen
: ProfileScreenController
: GroupManager
: DatabaseManager
1: user requests to delete groups
1.1: getCategory()
1.1.1: deleteGroup(group)
2: dao = GroupDbDAO.createDAO(group)
: GroupDbDAO
3: getHandle()
1.1.1.1: delete(group)
Powered By Visual Paradigm Community Edition

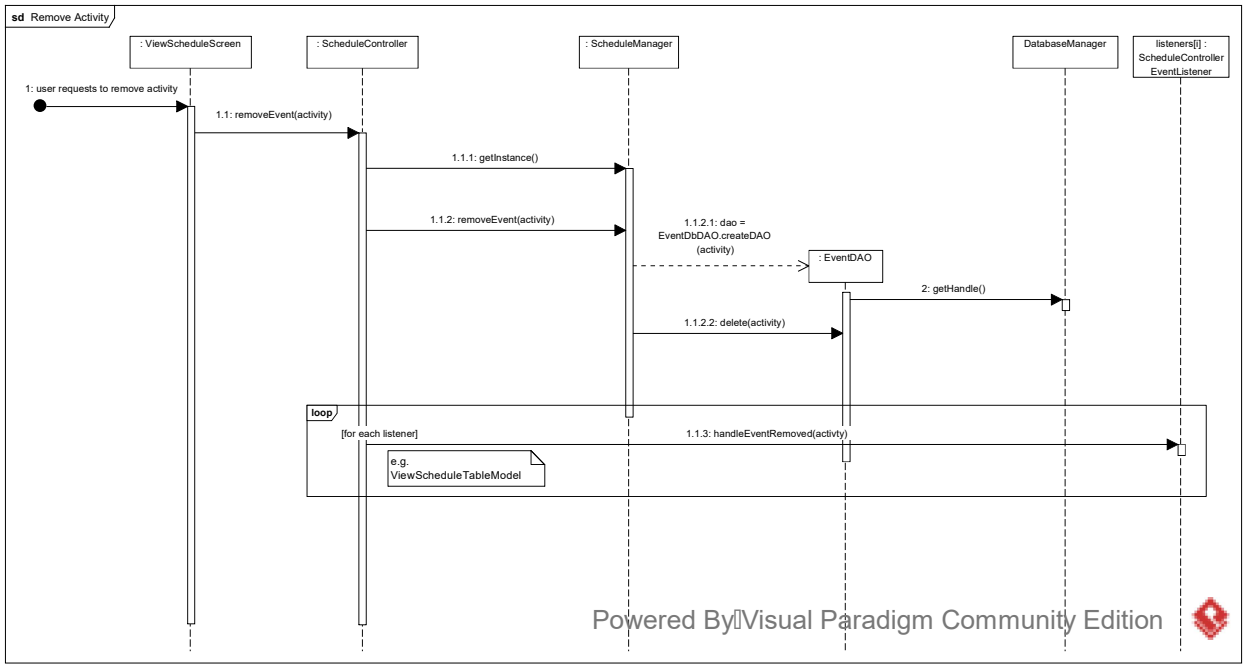

sd Edit Activity
: ViewScheduleScreen
: ScheduleController
: ScheduleManager
DatabaseManager
listeners[i] : ScheduleController EventListener
1: user submits form
1.1: changeEvent(oldActivity, newActivity)
1.1.1: getInstance()
1.1.2: conflicts = activityConflictsWithSchedule(activity, schedule)
opt
[conflicts]
1.1.3: show warning to user
1.1.4: changeEvent(activity)
1.1.4.1: dao = EventDbDAO.createDAO (activity)
: EventDAO
2: getHandle()
1.1.4.2: save(activity)
1.1.4.2.1: doSave(activity)
1.1.4.2.2: findById(activity.id)
1.1.4.2.3: update(activity)
loop
[for each listener]
1.1.5: handleEventChanged(oldActivity, newActivity)
e.g. ViewScheduleTableModel
Powered By Visual Paradigm Community Edition

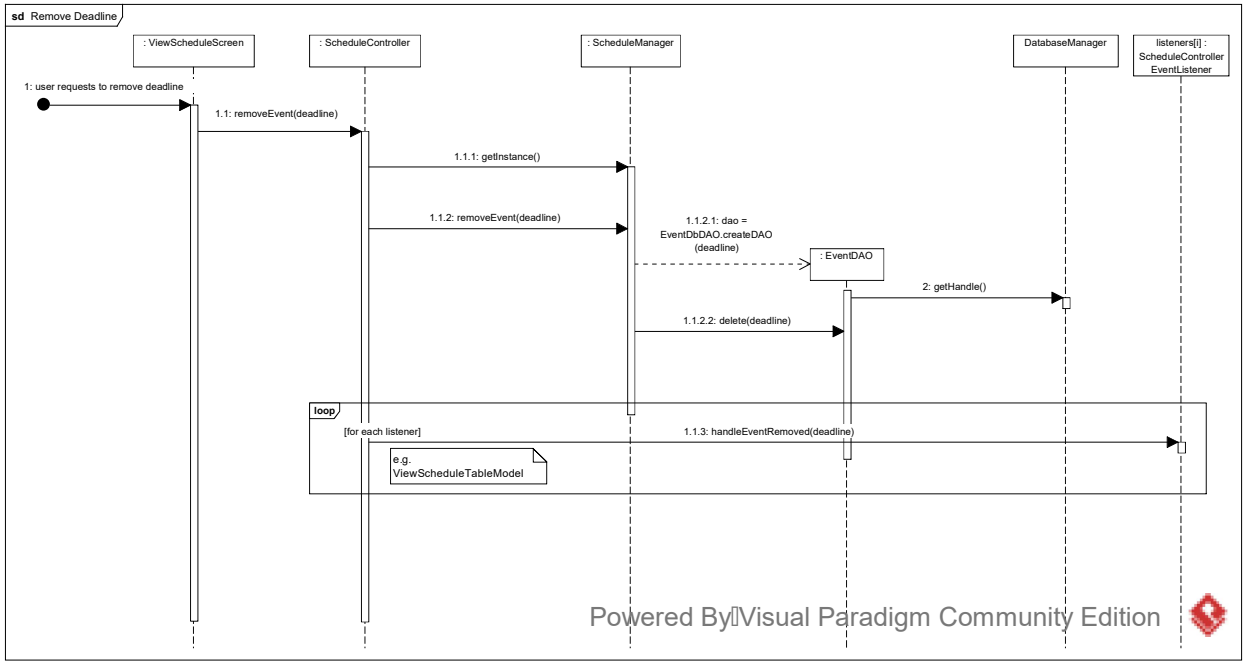

sd Edit Deadline
: ViewScheduleScreen
: ScheduleController
: ScheduleManager
DatabaseManager
listeners[i] : ScheduleController EventListener
1: user submits form
1.1: changeEvent(oldDeadline, newDeadline)
1.1.1: getInstance()
1.1.2: changeEvent(deadline)
1.1.2.1: dao = EventDbDAO.createDAO (deadline)
: EventDAO
2: getHandle()
1.1.2.2: save(deadline)
1.1.2.2.1: doSave(deadline)
1.1.2.2.2: findById(deadline.id)
1.1.2.2.3: update(deadline)
loop
[for each listener]
1.1.3: handleEventChanged(oldDeadline, newDeadline)
e.g. ViewScheduleTableModel
Powered By Visual Paradigm Community Edition

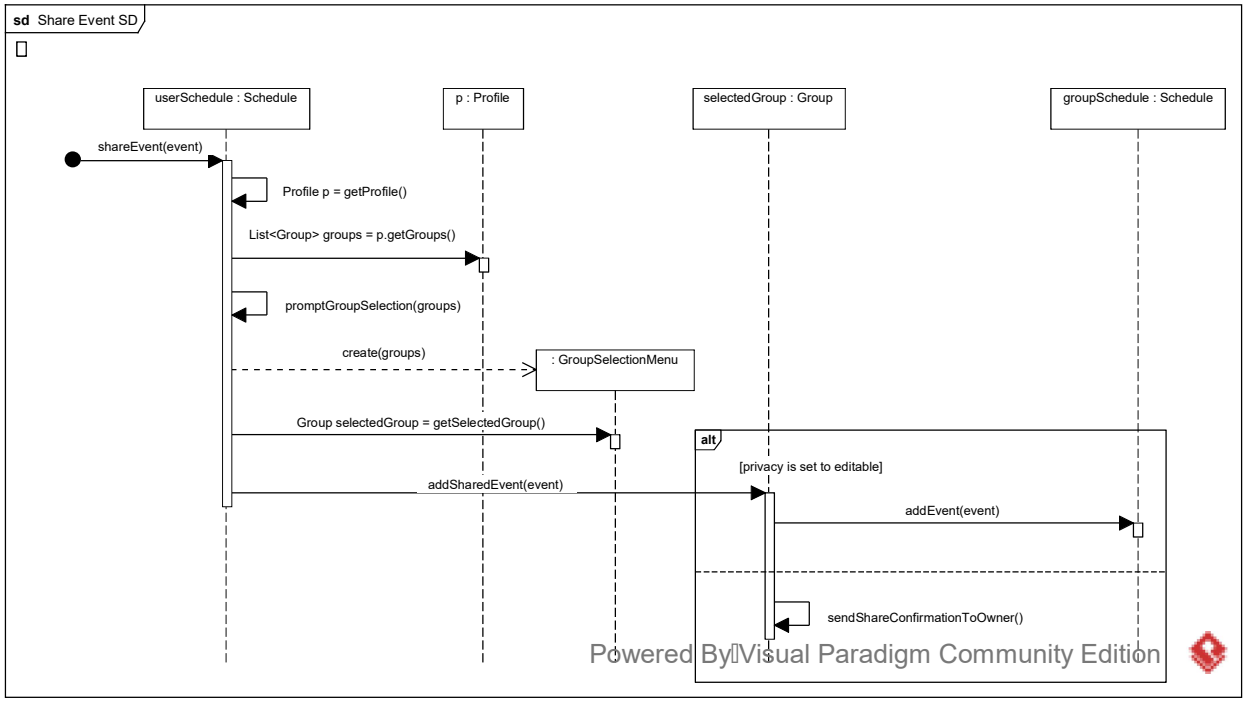

sd Edit Group SD
: ProfileScreen
: ProfileScreenController
: GroupManager
: DatabaseManager
1: user submits form
1.1: getCategory()
1.1.1: addMemberToGroup(User)
1.1.2: removeUserFromGroup(User)
2: dao = GroupDbDAO.createDAO(group)
: GroupDbDAO
3: getHandle()
4: save(group)
Powered By Visual Paradigm Community Edition

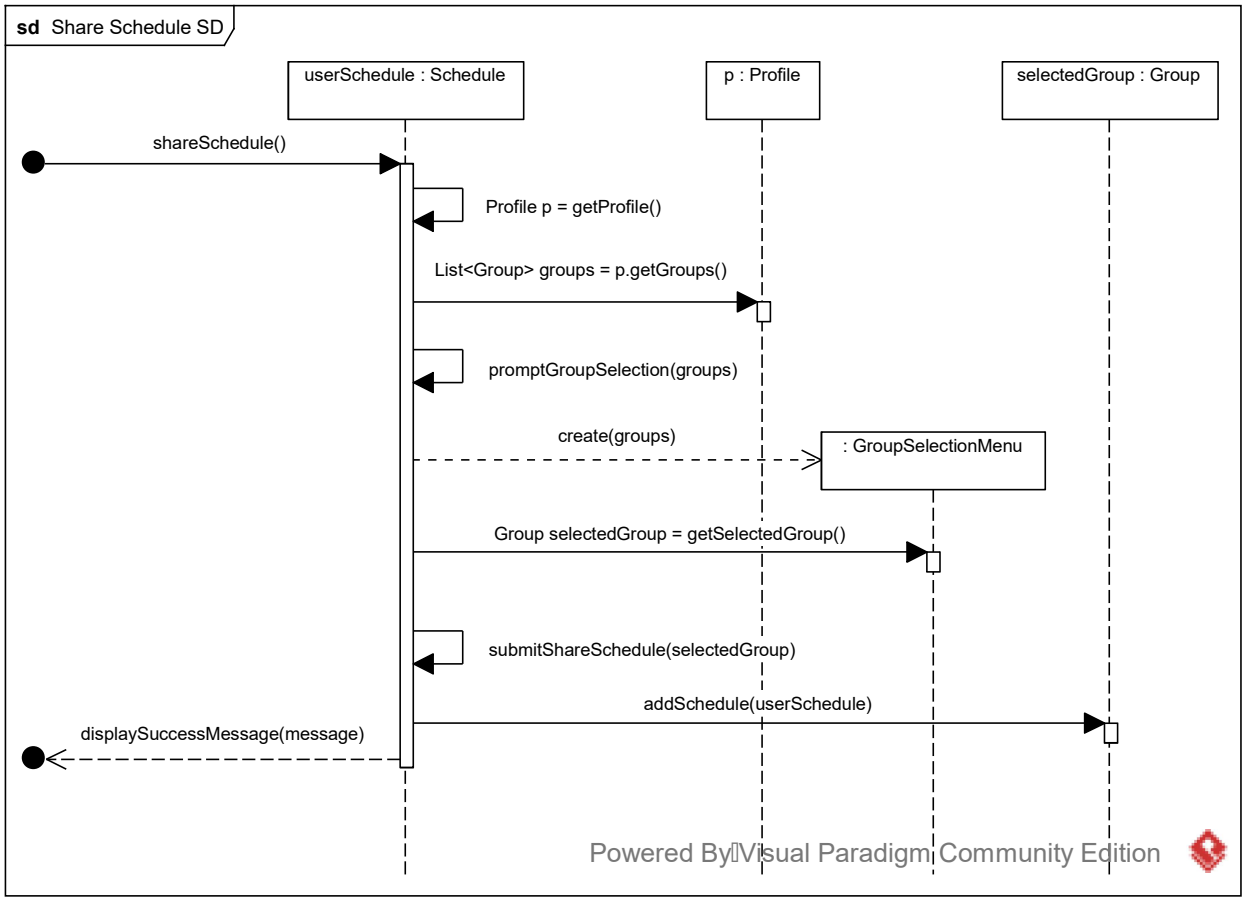

sd Edit Privacy Settings SD
: Group
editPrivacySettings()
displayPrivacySettingsMenu()
create()
menu : PrivacySettingsMenu
PrivacySettings settings = getChosenPrivacySettings()
setPrivacySettings(settings)
displaySuccessMessage(message)
Powered By Visual Paradigm Community Edition

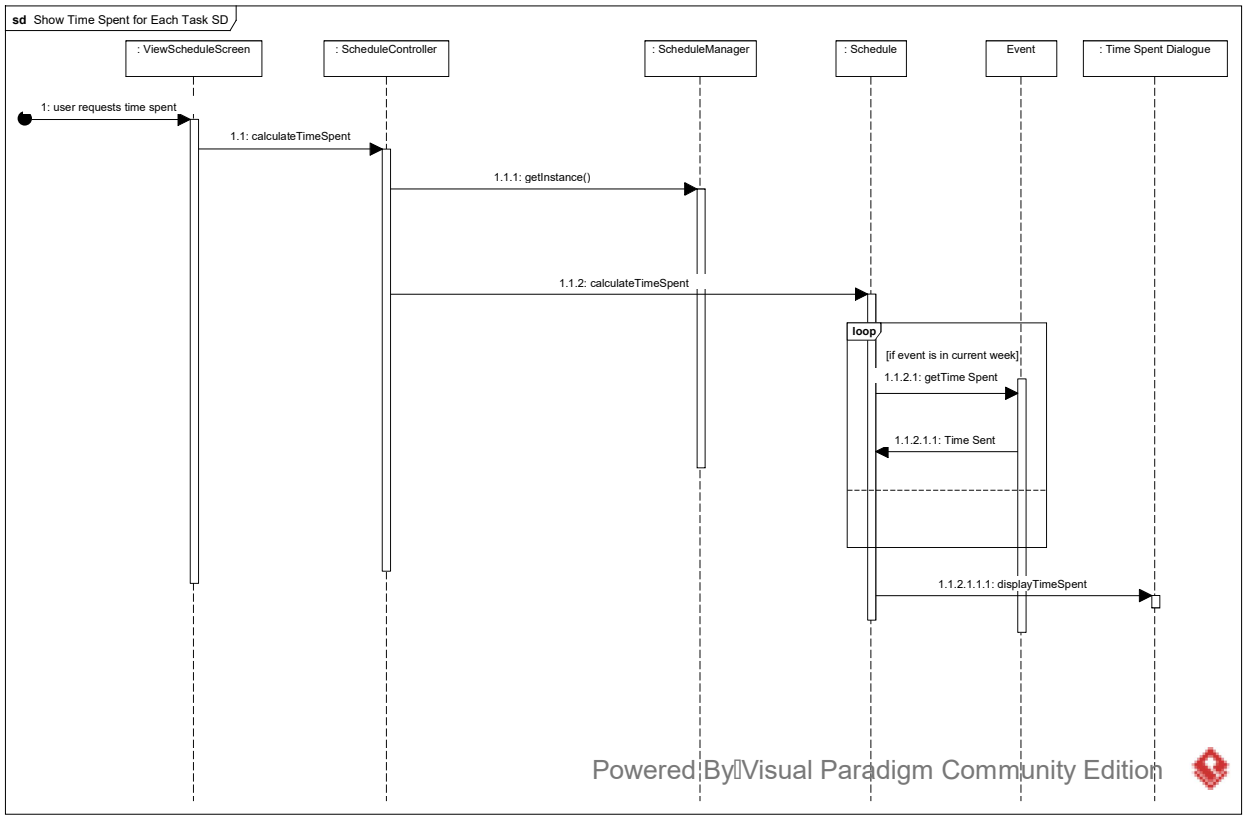

sd Generate To-Do List SD
User
: ViewScheduleScreen
UserSchedule : Schedule
1: generateToDoList(day)
1.1: getEvents(day)
alt
[# Events > 0]
2: createList()
ToDo List :
ArrayList<Events>
loop
[Event e : UserSchedule[day]]
3: add(e)
3.1: ToDo List
4: showToDoList()
5: Error Message
Powered By Visual Paradigm Community Edition

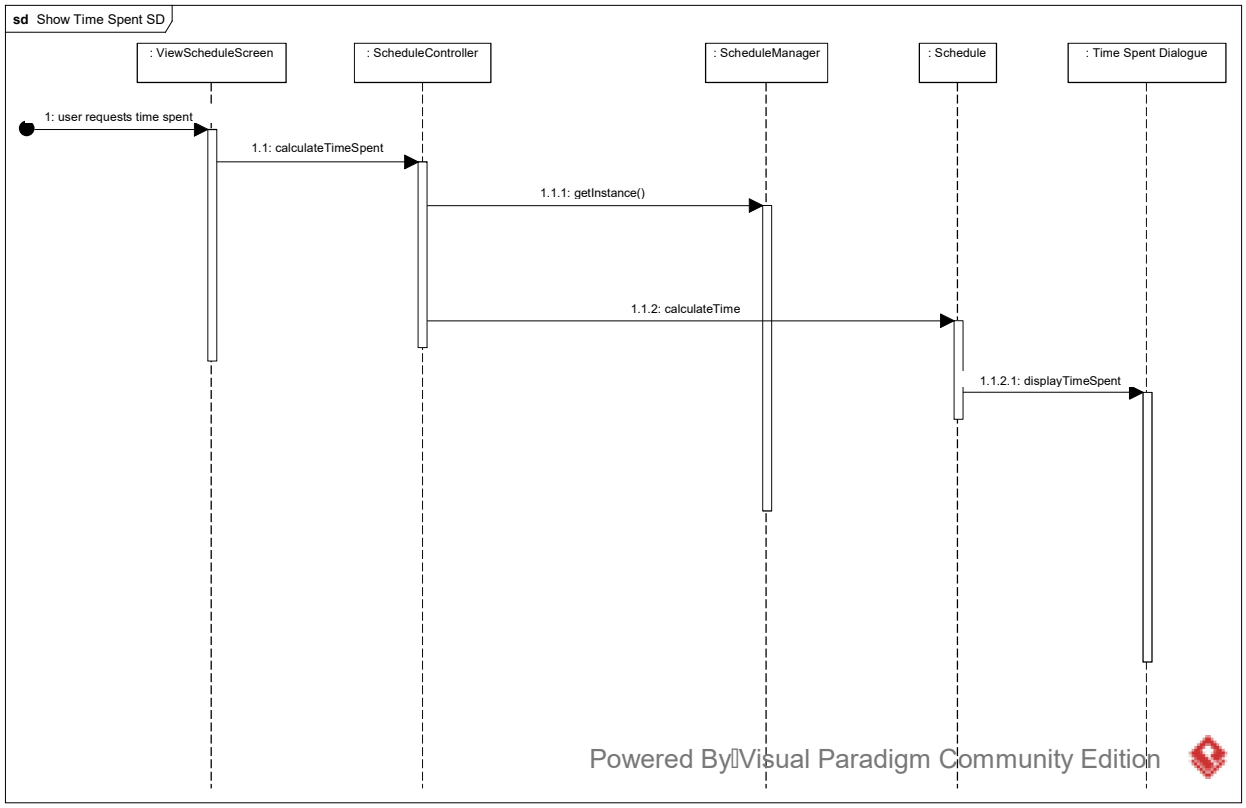

sd Remove Activity
: ViewScheduleScreen
: ScheduleController
: ScheduleManager
DatabaseManager
listeners[i] : ScheduleController EventListener
1: user requests to remove activity
1.1: removeEvent(activity)
1.1.1: getInstance()
1.1.2: removeEvent(activity)
1.1.2.1: dao = EventDbDAO.createDAO (activity)
: EventDAO
2: getHandle()
1.1.2.2: delete(activity)
loop
[for each listener]
1.1.3: handleEventRemoved(activty)
e.g. ViewScheduleTableModel
Powered By Visual Paradigm Community Edition


sd Remove Deadline
: ViewScheduleScreen
: ScheduleController
: ScheduleManager
DatabaseManager
listeners[i] : ScheduleController EventListener
1: user requests to remove deadline
1.1: removeEvent(deadline)
1.1.1: getInstance()
1.1.2: removeEvent(deadline)
1.1.2.1: dao = EventDbDAO.createDAO (deadline)
: EventDAO
2: getHandle()
1.1.2.2: delete(deadline)
loop
[for each listener]
1.1.3: handleEventRemoved(deadline)
e.g. ViewScheduleTableModel
Powered By Visual Paradigm Community Edition


sd Share Event SD
userSchedule : Schedule
p : Profile
selectedGroup : Group
groupSchedule : Schedule
shareEvent(event)
Profile p = getProfile()
List<Group> groups = p.getGroups()
promptGroupSelection(groups)
create(groups)
: GroupSelectionMenu
Group selectedGroup = getSelectedGroup()
alt
[privacy is set to editable]
addSharedEvent(event)
addEvent(event)
sendShareConfirmationToOwner()
Powered By Visual Paradigm Community Edition


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


sd Show Time Spent for Each Task SD
: ViewScheduleScreen
: ScheduleController
: ScheduleManager
: Schedule
Event
: Time Spent Dialogue
1: user requests time spent
1.1: calculateTimeSpent
1.1.1: getInstance()
1.1.2: calculateTimeSpent
loop
[if event is in current week]
1.1.2.1: getTime Spent
1.1.2.1.1: Time Sent
1.1.2.1.1.1: displayTimeSpent
Powered By Visual Paradigm Community Edition


sd Show Time Spent SD
: ViewScheduleScreen
: ScheduleController
: ScheduleManager
: Schedule
: Time Spent Dialogue
1: user requests time spent
1.1: calculateTimeSpent
1.1.1: getInstance()
1.1.2: calculateTime
1.1.2.1: displayTimeSpent
Powered By Visual Paradigm Community Edition

# GRASP: System Operation 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.

**Operation:** viewSchedule

**Class:** MainWindow

**GRASP Patterns:** Expert, Controller

**Justification:** The operation for viewing the schedule is triggered by selecting a tab on the main window. Since knowledge of the main window tabs is restricted to the MainWindow class, the expert pattern is therefore used to justify the incorporation of the viewSchedule operation in the MainWindow class. For the second part of this operation, a controller populates the table with the saved schedule information. The controller pattern is used to allow the instance of the ViewScheduleScreen class to be ignorant of the information within the schedule, so that the class can instead focus on only being dedicated to how the schedule is displayed in general terms. This improves the cohesion of the ViewScheduleScreen class.

**Operation:** calculateFreeTime

**Class:** Schedule

**GRASP Pattern:** Expert

**Justification:** This operation requires information that is only known by the Schedule class, as it calculates the user's free time based on the events currently added to the schedule. Therefore,

since the operation requires this restricted information, the expert pattern is used to justify the inclusion of the calculateFreeTime operation in the Schedule class.

**Operation:** displayFreeTime

**Class:** Schedule

**GRASP Pattern:** Expert

**Justification:** Responsibility for displaying the free time is given to the Schedule class by the expert pattern. This is because only the Schedule class can calculate the free time, which is restricted information that the displayFreeTime operation needs.

**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.

**Operation:** createProfile

**Class:** MainWindow

**GRASP Pattern:** Creator

**Justification:** The operation uses the creator pattern by creating an instance of the Profile class that is customized by the user before it is added to the database. The use of the creator pattern in this operation is the necessary consequence of having a dedicated Profile class that increases the

cohesion of the system by including all information and functionality that is relevant to the user's profile.

**Operation:** editProfile

**Class:** MainWindow

**GRASP Pattern:** Controller

**Justification:** The controller pattern is used to allow the editProfile operation to be initiated by various input events. This improves the sustainability of the system by allowing additional input events to simply call the controller, reducing the code needed to handle that input event.

**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.

**Operation:** editPrivacySettings

**Class:** Group

**GRASP Pattern:** Expert

**Justification:** The expert pattern is used to justify the placement of the editPrivacySettings operation in the Group class. This is because the operation requires access to private information within that class.

**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.

**Operations:** addMembers, removeMembers

**Class:** Group

**GRASP Pattern:** Expert, Controller

**Justification:** The expert pattern is used to justify placing the addMembers and removeMembers operations in the Group class. This is because these operations require private information within the Group class. The controller pattern is used for updating the database to reflect the change in the members of the group. This allows the application to be more easily extended to handle different types of databases.

**Operation:** createTodoList

**Class:** Schedule

**GRASP Pattern:** Expert

**Justification:** The expert pattern justifies the placement of the createTodoList operation in the Schedule class. This operation uses private information within the Schedule class, such as the activities for the current day and upcoming deadlines, to form a list of tasks it recommends that the user completes. Since the Schedule class is the expert on this information, it should have this operation.

**Operation:** shareEvent

**Class:** Schedule

**GRASP Pattern:** Expert

**Justification:** The expert pattern justifies the placement of the shareEvent operation in the Schedule class. This operation requires access to a schedule's event, which is private information within the Schedule class. Since the Schedule class is the expert on this information, it should have this operation.

**Operation:** displayGroupList

**Class:** User

**GRASP Pattern:** Expert

**Justification:** The expert pattern justifies the placement of the displayGroupList operation within the User class, because this operation requires access to the list of groups contained within the User object, which is private information.

**Operation:** confirmSharedEvent, refuseSharedEvent

**Class:** User

**GRASP Pattern:** Expert

**Justification:** The expert pattern justifies the placement of the confirmSharedEvent and refuseSharedEvent operations within the User class, because this operation requires access to the list of shared events contained within the User object, which is private information.

**Operation:** shareSchedule

**Class:** Schedule

**GRASP Pattern:** Expert

**Justification:** The expert pattern justifies the placement of the shareSchedule operation within the Schedule class, because this operation requires access to the information contained within the Schedule object, which is private.

**Operation:** viewTimeLeft, viewTimeSpent

**Class:** Deadline

**GRASP Pattern:** Expert

**Justification:** The expert pattern justifies the placement of the viewTimeLeft and viewTimeSpent operations within the Deadline class, because this operation requires access to the time currently spent on the deadline as well as the due date. Because these operations are highly related to the Deadline class, including these operations in that class increases the cohesion of the application's design.

# Test Coverage Plan

***The following project modules will be tested as shown:***

- **Activities**

  **Module(s):**

  edu/baylor/csi3471/netime_planner/models/Activity

  edu/baylor/csi3471/netime_planner/models/TimeInterval

  **Tests:**

  - 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**

  **Module(s):**

  edu/baylor/csi3471/netime_planner/util/MathUtils

  **Tests:**

  - 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**

  **Module(s):**

  edu/baylor/csi3471/netime_planner/util/StringUtils

  **Tests:**

  - Test the usernameToDataFile method.

- **Date Utilities**

  **Module(s):**

  edu/baylor/csi3471/netime_planner/util/DateUtils

  **Tests:**

  - Test the getNextWeekDay method.
  - Test the getLastSunday method.
  - Test the weekDaySet method.

- **Schedule**

  **Module(s):**

  edu/baylor/csi3471/netime_planner/models/DateTimeInterval

  edu/baylor/csi3471/netime_planner/models/Schedule

  **Tests:**

  - Test the makeToDoList method.
  - Test the addEvent method.
  - Test the removeEvent method.

- Test the setWorkTimes method.

- **Password Hashing and Login Functionality**

  **Module(s):**

  edu/baylor/csi3471/netime_planner/models/LoginVerification

  **Tests:**

  - Test the verifyUsernameAndPassword method.
  - Test the storeUsernameAndPassword method.

- **Loading and Saving XML Files**

  **Module(s):**

  edu/baylor/csi3471/netime_planner/models/Controller

  **Tests:**

  - Test the saveLocally method.
  - Test the loadLocally method.

- **ViewScheduleTableModel**

  **Module(s):**

  edu/baylor/csi3471/netime_planner/gui/ViewScheduleTableModel

  **Tests:**

  - Test the add method.
  - Test the remove method.
  - Test the change method.

- Test the getValueAt method.
- Test the getRowCount method.

- **ViewScheduleScreen**

  **Module(s):**

  edu/baylor/csi3471/netime_planner/gui/ViewScheduleScreen

  **Tests:**

  - 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**

  **Module(s):**

  edu/baylor/csi3471/netime_planner/gui/CreateActivityForm

  **Tests:**

  - Test the createValue method.

- **CreateDeadlineForm**

  **Module(s):**

edu/baylor/csi3471/netime_planner/gui/CreateDeadlineForm

**Tests:**

- Test the createValue method.

## Links and Resources

Application Repository: https://github.com/KevinDeMars/netime-planner-application

Issue Tracking: https://github.com/KevinDeMars/netime-planner-application/issues

Website Repository: https://github.com/KevinDeMars/netime-planner

Website: https://kevindemars.github.io/netime-planner/

Revised Iteration 1 Deliverables: https://kevindemars.github.io/netime-planner/downloads/NETime-Planner-Iteration-1-Deliverables-rev3.zip

Iteration 2 Presentation: https://kevindemars.github.io/netime-planner/downloads/Iteration-2-Presentation.mp4

Iteration 2 Presentation Powerpoint: https://kevindemars.github.io/netime-planner/downloads/Iteration-2-Powerpoint.pdf