CIS 350 02

WINTER 2022

Term Project – Release 1

Ben Allen allebenj@mail.gvsu.edu

Devin Elenbaase elenbaad@mail.gvsu.edu

Bryan VanDyke vandybry@mail.gvsu.edu

Studious

A student organizer.

Graphical user interface, text

Description automatically generated with medium confidence

Contents

[Studious: A student organizer. 3](#_Toc95131523)

[Project Description 3](#_Toc95131524)

[Screenshots 3](#_Toc95131525)

[Use Case 4](#_Toc95131526)

[Use Case Diagrams 4](#_Toc95131527)

[Use Case Descriptions 5](#_Toc95131528)

[Design Diagrams 16](#_Toc95131529)

[Source Code Anal 19](#_Toc95131530)

[Coding Standards 19](#_Toc95131531)

[Bug Checks 19](#_Toc95131532)

[Code Repository 20](#_Toc95131533)

[Repository 20](#_Toc95131534)

[Project Website 20](#_Toc95131535)

[Testing 21](#_Toc95131536)

[Unit Tests 21](#_Toc95131537)

[Code coverage 21](#_Toc95131538)

[Member Roles 22](#_Toc95131539)

[Ben Allen 22](#_Toc95131540)

[Devin Elenbaase 22](#_Toc95131541)

[Bryan VanDyke 22](#_Toc95131542)

[Self-Reflections 23](#_Toc95131543)

[Ben Allen 23](#_Toc95131544)

[Devin Elenbaase 23](#_Toc95131545)

[Bryan VanDyke 23](#_Toc95131546)

[Project Demo 24](#_Toc95131547)

[Appendix A 25](#_Toc95131548)

**Green text is from the rubric. TODO: ~~mark done~~ as needed (or delete)**

# Studious: A student organizer.

## Project Description

Project description & List of features implemented in release 1

Studious is planned to be an android-based student organizational assistant, with primary feature goals.

1. Calendar/Event-planner allows students to keep track of their upcoming exams, projects, presentations, etc., in one place.
2. Ability to track how many work hours have been committed to a single project/goal.
3. Ability to set weekly goals/to-do lists for workhours put toward specific projects.

These features will assist a student in organizing and using their time efficiently through the chaos that is college scheduling.

A possible secondary stretch feature would be a repository for students to upload a particular professor's general assignment scheduling for a specific class. For example, it would allow students to see if their CIS 350 class will have a semester-long group project as opposed to CIS 241 having four individual projects throughout the semester.

~~Sample screenshots of your application 10~~

## Screenshots

Graphical user interface, text, application

Description automatically generatedA close-up of a cell phone

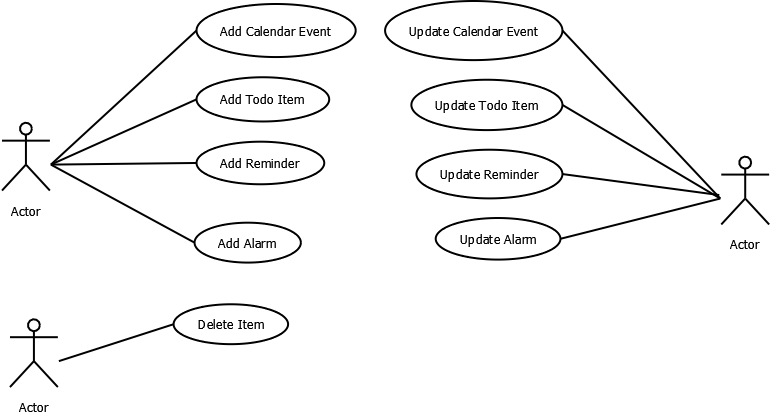
Description automatically generated with medium confidenceGraphical user interface, text, application, chat or text message

Description automatically generated

## Use Case

### Use Case Diagrams

Use case diagram (system boundary diagram)



### Use Case Descriptions

Use case descriptions (using the template provided on Blackboard) 20

|  |  |
| --- | --- |
| **Name** | Add Calendar Event |
| **ID** | UC1 |
| **Brief Description** | Add Calendar Event |
| **Actors (primary and supporting/secondary)** | Primary (use case initiator) and supporting actor(s) |
| **Triggers** | Event that gets the use case started |
| **Preconditions** | Conditions that must be true before the use case can start.  Written in the present tense. |
| **Primary Flow** | Main success scenario (aka primary or happy scenario) as a sequence of actions steps by various actors and system.  - Each action step is written to show a simple, active action  - Number the steps  Example:  1.  2.  3.  4.  .  . |
| **Alternate Flows** | For every point (step) in the basic flow where the behavior can branch because of a particular condition  - Write down the condition  - Write the steps that handle it  - Number the steps within each condition  Example:  2a condition that causes a branch in the main flow  2a.1 action step 1  2a.2 action step 2  .  . . |
| **Minimal Guarantees** | A minimal guarantee represents a condition that will be true when the use case ends, regardless of how it terminates.  Written as assertion(s) in the past tense. |
| **Success Guarantees** | A success guarantee represents a condition that will be true when the use case ends successfully, regardless of which path it took.  Written as assertion(s) in the past tense. |

|  |  |
| --- | --- |
| **Name** | Add Todo Item |
| **ID** | UC2 |
| **Brief Description** | Add Todo Item |
| **Actors (primary and supporting/secondary)** | Primary (use case initiator) and supporting actor(s) |
| **Triggers** | Event that gets the use case started |
| **Preconditions** | Conditions that must be true before the use case can start.  Written in the present tense. |
| **Primary Flow** | Main success scenario (aka primary or happy scenario) as a sequence of actions steps by various actors and system.  - Each action step is written to show a simple, active action  - Number the steps  Example:  1.  2.  3.  4.  .  . |
| **Alternate Flows** | For every point (step) in the basic flow where the behavior can branch because of a particular condition  - Write down the condition  - Write the steps that handle it  - Number the steps within each condition  Example:  2a condition that causes a branch in the main flow  2a.1 action step 1  2a.2 action step 2  .  .  .  . |
| **Minimal Guarantees** | A minimal guarantee represents a condition that will be true when the use case ends, regardless of how it terminates.  Written as assertion(s) in the past tense. |
| **Success Guarantees** | A success guarantee represents a condition that will be true when the use case ends successfully, regardless of which path it took.  Written as assertion(s) in the past tense. |

|  |  |
| --- | --- |
| **Name** | Add Reminder |
| **ID** | UC3 |
| **Brief Description** | Add Reminder |
| **Actors (primary and supporting/secondary)** | Primary (use case initiator) and supporting actor(s) |
| **Triggers** | Event that gets the use case started |
| **Preconditions** | Conditions that must be true before the use case can start.  Written in the present tense. |
| **Primary Flow** | Main success scenario (aka primary or happy scenario) as a sequence of actions steps by various actors and system.  - Each action step is written to show a simple, active action  - Number the steps  Example:  1.  2.  3.  4.  .  . |
| **Alternate Flows** | For every point (step) in the basic flow where the behavior can branch because of a particular condition  - Write down the condition  - Write the steps that handle it  - Number the steps within each condition  Example:  2a condition that causes a branch in the main flow  2a.1 action step 1  2a.2 action step 2  .  .  .  . |
| **Minimal Guarantees** | A minimal guarantee represents a condition that will be true when the use case ends, regardless of how it terminates.  Written as assertion(s) in the past tense. |
| **Success Guarantees** | A success guarantee represents a condition that will be true when the use case ends successfully, regardless of which path it took.  Written as assertion(s) in the past tense. |

|  |  |
| --- | --- |
| **Name** | Add Alarm |
| **ID** | UC4 |
| **Brief Description** | Add Alarm |
| **Actors (primary and supporting/secondary)** | Primary (use case initiator) and supporting actor(s) |
| **Triggers** | Event that gets the use case started |
| **Preconditions** | Conditions that must be true before the use case can start.  Written in the present tense. |
| **Primary Flow** | Main success scenario (aka primary or happy scenario) as a sequence of actions steps by various actors and system.  - Each action step is written to show a simple, active action  - Number the steps  Example:  1.  2.  3.  4.  .  . |
| **Alternate Flows** | For every point (step) in the basic flow where the behavior can branch because of a particular condition  - Write down the condition  - Write the steps that handle it  - Number the steps within each condition  Example:  2a condition that causes a branch in the main flow  2a.1 action step 1  2a.2 action step 2  .  .  .  . |
| **Minimal Guarantees** | A minimal guarantee represents a condition that will be true when the use case ends, regardless of how it terminates.  Written as assertion(s) in the past tense. |
| **Success Guarantees** | A success guarantee represents a condition that will be true when the use case ends successfully, regardless of which path it took.  Written as assertion(s) in the past tense. |

|  |  |
| --- | --- |
| **Name** | Update Calendar Event |
| **ID** | UC5 |
| **Brief Description** | Update Calendar Event |
| **Actors (primary and supporting/secondary)** | Primary (use case initiator) and supporting actor(s) |
| **Triggers** | Event that gets the use case started |
| **Preconditions** | Conditions that must be true before the use case can start.  Written in the present tense. |
| **Primary Flow** | Main success scenario (aka primary or happy scenario) as a sequence of actions steps by various actors and system.  - Each action step is written to show a simple, active action  - Number the steps  Example:  1.  2.  3.  4.  .  . |
| **Alternate Flows** | For every point (step) in the basic flow where the behavior can branch because of a particular condition  - Write down the condition  - Write the steps that handle it  - Number the steps within each condition  Example:  2a condition that causes a branch in the main flow  2a.1 action step 1  2a.2 action step 2  .  .  .  . |
| **Minimal Guarantees** | A minimal guarantee represents a condition that will be true when the use case ends, regardless of how it terminates.  Written as assertion(s) in the past tense. |
| **Success Guarantees** | A success guarantee represents a condition that will be true when the use case ends successfully, regardless of which path it took.  Written as assertion(s) in the past tense. |

|  |  |
| --- | --- |
| **Name** | Update Todo Item |
| **ID** | UC6 |
| **Brief Description** | Update Todo Item |
| **Actors (primary and supporting/secondary)** | Primary (use case initiator) and supporting actor(s) |
| **Triggers** | Event that gets the use case started |
| **Preconditions** | Conditions that must be true before the use case can start.  Written in the present tense. |
| **Primary Flow** | Main success scenario (aka primary or happy scenario) as a sequence of actions steps by various actors and system.  - Each action step is written to show a simple, active action  - Number the steps  Example:  1.  2.  3.  4.  .  . |
| **Alternate Flows** | For every point (step) in the basic flow where the behavior can branch because of a particular condition  - Write down the condition  - Write the steps that handle it  - Number the steps within each condition  Example:  2a condition that causes a branch in the main flow  2a.1 action step 1  2a.2 action step 2  .  .  .  . |
| **Minimal Guarantees** | A minimal guarantee represents a condition that will be true when the use case ends, regardless of how it terminates.  Written as assertion(s) in the past tense. |
| **Success Guarantees** | A success guarantee represents a condition that will be true when the use case ends successfully, regardless of which path it took.  Written as assertion(s) in the past tense. |

|  |  |
| --- | --- |
| **Name** | Update Reminder |
| **ID** | UC7 |
| **Brief Description** | Update Reminder |
| **Actors (primary and supporting/secondary)** | Primary (use case initiator) and supporting actor(s) |
| **Triggers** | Event that gets the use case started |
| **Preconditions** | Conditions that must be true before the use case can start.  Written in the present tense. |
| **Primary Flow** | Main success scenario (aka primary or happy scenario) as a sequence of actions steps by various actors and system.  - Each action step is written to show a simple, active action  - Number the steps  Example:  1.  2.  3.  4.  .  . |
| **Alternate Flows** | For every point (step) in the basic flow where the behavior can branch because of a particular condition  - Write down the condition  - Write the steps that handle it  - Number the steps within each condition  Example:  2a condition that causes a branch in the main flow  2a.1 action step 1  2a.2 action step 2  .  .  .  . |
| **Minimal Guarantees** | A minimal guarantee represents a condition that will be true when the use case ends, regardless of how it terminates.  Written as assertion(s) in the past tense. |
| **Success Guarantees** | A success guarantee represents a condition that will be true when the use case ends successfully, regardless of which path it took.  Written as assertion(s) in the past tense. |

|  |  |
| --- | --- |
| **Name** | Update Alarm |
| **ID** | UC8 |
| **Brief Description** | Update Alarm |
| **Actors (primary and supporting/secondary)** | Primary (use case initiator) and supporting actor(s) |
| **Triggers** | Event that gets the use case started |
| **Preconditions** | Conditions that must be true before the use case can start.  Written in the present tense. |
| **Primary Flow** | Main success scenario (aka primary or happy scenario) as a sequence of actions steps by various actors and system.  - Each action step is written to show a simple, active action  - Number the steps  Example:  1.  2.  3.  4.  .  . |
| **Alternate Flows** | For every point (step) in the basic flow where the behavior can branch because of a particular condition  - Write down the condition  - Write the steps that handle it  - Number the steps within each condition  Example:  2a condition that causes a branch in the main flow  2a.1 action step 1  2a.2 action step 2  .  .  .  . |
| **Minimal Guarantees** | A minimal guarantee represents a condition that will be true when the use case ends, regardless of how it terminates.  Written as assertion(s) in the past tense. |
| **Success Guarantees** | A success guarantee represents a condition that will be true when the use case ends successfully, regardless of which path it took.  Written as assertion(s) in the past tense. |

|  |  |
| --- | --- |
| **Name** | Delete Item |
| **ID** | UC9 |
| **Brief Description** | Delete Item |
| **Actors (primary and supporting/secondary)** | Primary (use case initiator) and supporting actor(s) |
| **Triggers** | Event that gets the use case started |
| **Preconditions** | Conditions that must be true before the use case can start.  Written in the present tense. |
| **Primary Flow** | Main success scenario (aka primary or happy scenario) as a sequence of actions steps by various actors and system.  - Each action step is written to show a simple, active action  - Number the steps  Example:  1.  2.  3.  4.  .  . |
| **Alternate Flows** | For every point (step) in the basic flow where the behavior can branch because of a particular condition  - Write down the condition  - Write the steps that handle it  - Number the steps within each condition  Example:  2a condition that causes a branch in the main flow  2a.1 action step 1  2a.2 action step 2  .  .  .  . |
| **Minimal Guarantees** | A minimal guarantee represents a condition that will be true when the use case ends, regardless of how it terminates.  Written as assertion(s) in the past tense. |
| **Success Guarantees** | A success guarantee represents a condition that will be true when the use case ends successfully, regardless of which path it took.  Written as assertion(s) in the past tense. |

## Design Diagrams

Design diagrams (such as Class diagrams) 15

Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidenceGraphical user interface, text, application

Description automatically generatedText

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing graphical user interface

Description automatically generated

## Source Code Anal

Usage of static source code analyzers for

### Coding Standards

• Enforcing coding standards/conformance (using tools like Checkstyle or other IDE/language-specific)

### Bug Checks

• Finding potential bugs in the source code (using tools like SpotBugs or other IDE/language specific) 10

## Code Repository

### Repository

~~URL to application code repository (on GitHub)~~

<https://github.com/AllenStudent/CIS-350-Term-Project.git>

git@github.com:AllenStudent/CIS-350-Term-Project.git

### Project Website

~~Project website on GitHub Pages at http://username.github.io/repository/ 10~~

<https://github.com/AllenStudent/CIS-350-Term-Project>

## Testing

Unit tests and code coverage reports from unit testing and functional/system testing using tools such as

• JUnit (and EclEmma for Eclipse)

• Or, other appropriate tools for IDE/language used 15

### Unit Tests

### Code coverage

## Member Roles

Roles/Responsibilities of each team member of the project

### Ben Allen

### Devin Elenbaase

### Bryan VanDyke

## Self-Reflections

Self-reflection by each team member 10

### Ben Allen

### Devin Elenbaase

### Bryan VanDyke

## Project Demo

Project Demo (not part of the release document; TBD) 10

## Appendix A

Old Git Repo

Git log for the development of the android project portion of project. Merged in group project. Original git repository for Android project files. The git repository can be found in android\_git.zip.

commit 9cdfa73923ad12d49e253ec77761d9f8c506a14b

Author: Bryan VanDyke <bryan.vandyke@gmail.com>

Date: Sat Feb 5 23:31:16 2022 -0500

Added custom icons to list.

commit d701769153b8ce1bbada9a46ca3230d90729fb79

Author: Bryan VanDyke <bryan.vandyke@gmail.com>

Date: Sat Feb 5 22:30:07 2022 -0500

done with basic app.

commit 3479199165d09b7d0896760cd402797764331f70

Author: Bryan VanDyke <bryan.vandyke@gmail.com>

Date: Sat Feb 5 21:25:58 2022 -0500

add delete by swipe

commit 1c8b5d4adffc17a269e15b102f1eaebe85f79fa0

Author: Bryan VanDyke <bryan.vandyke@gmail.com>

Date: Sat Feb 5 16:53:09 2022 -0500

import expandable fab library

https://github.com/nambicompany/expandable-fab

commit 1bcb27643affc8d6939a97d1100ce06730cd0e1c

Author: Bryan VanDyke <bryan.vandyke@gmail.com>

Date: Sat Feb 5 15:15:19 2022 -0500

cealn up activities

commit ff9f7d0f31a7c875fb4e22c9753d3dbcc654073c

Author: Bryan VanDyke <bryan.vandyke@gmail.com>

Date: Sat Feb 5 14:02:32 2022 -0500

got recycler view working

commit ff63c16085ca5966fe341f0e95a52e562984fc87

Author: Bryan VanDyke <bryan.vandyke@gmail.com>

Date: Fri Feb 4 16:21:45 2022 -0500

More comments added.

commit 2c4e41a98c4e386e7b8938f92e779ee9eb1d1bb1

Author: Bryan VanDyke <bryan.vandyke@gmail.com>

Date: Fri Feb 4 15:58:42 2022 -0500

Update comments.

commit 346dd651754857c13f9783f1467f8c72e2e40f94

Author: Bryan VanDyke <bryan.vandyke@gmail.com>

Date: Mon Jan 31 08:39:58 2022 -0500

Note on what needs to be changed.

commit 6c02cf8fc3053c1060c1b154783adb6be1525c5c

Author: Bryan VanDyke <bryan.vandyke@gmail.com>

Date: Sun Jan 30 17:00:57 2022 -0500

Add basic save to sqlite.

commit e9de6465df84b4bb554a76decac1ee5c5b92f64a

Author: Bryan VanDyke <bryan.vandyke@gmail.com>

Date: Sun Jan 30 10:28:16 2022 -0500

save some sqlite stuff

commit d77c98e96de88f6bc82586846dcbee8807e85ce5

Author: Bryan VanDyke <bryan.vandyke@gmail.com>

Date: Wed Jan 26 17:11:57 2022 -0500

Add readme file.

commit ffe7e6b08b3ab9a2e9934e16790c72f7de487544

Author: Bryan VanDyke <bryan.vandyke@gmail.com>

Date: Wed Jan 26 17:04:28 2022 -0500

Initial commit