GEAR-UP

CSC207 Final Project

Presented by Group 168

TEAM MEMBERS

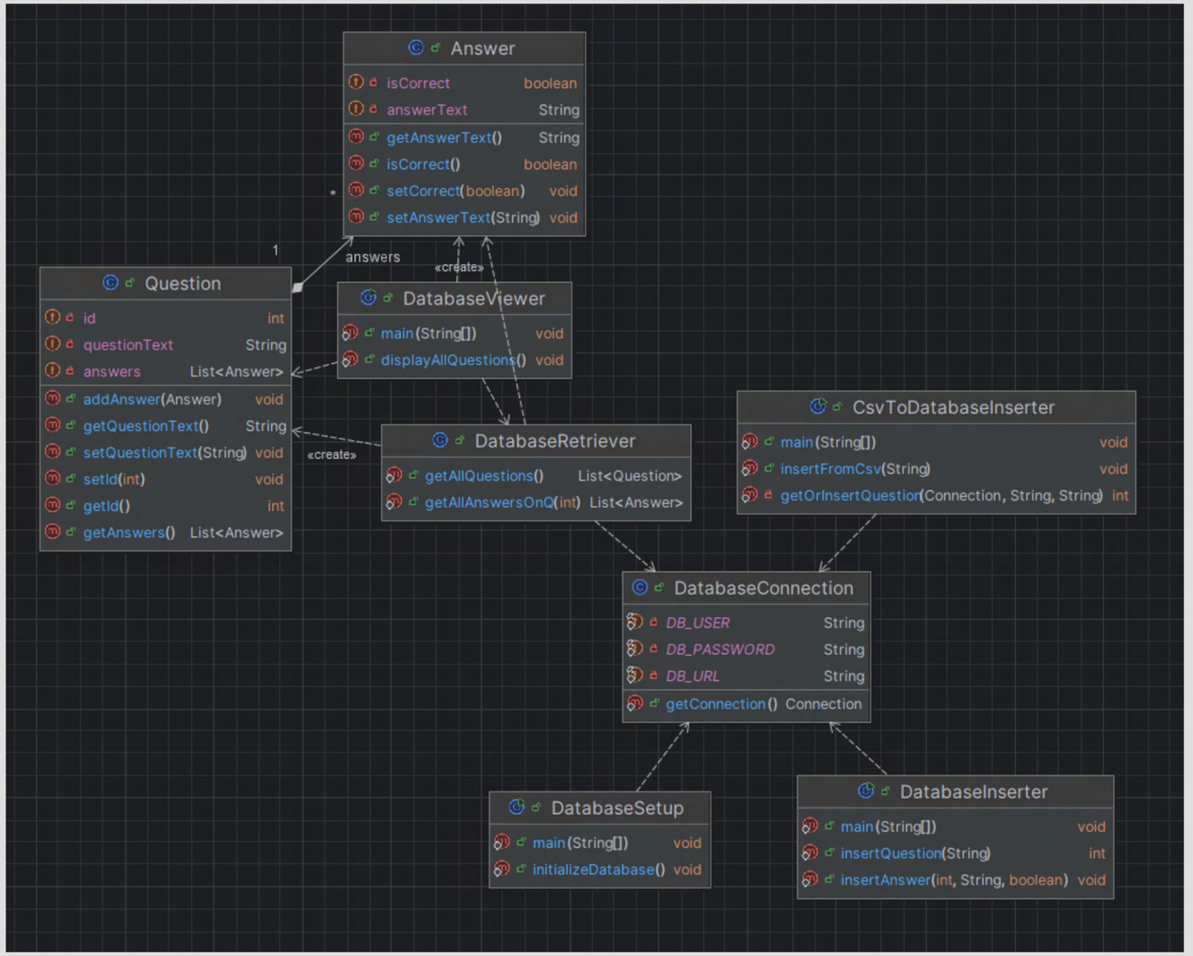
1. Yanting/Clara Fan
2. Zixiang/Terry Huang
3. Kimberly Fu
4. Yixuan/Amelia Wu

SPECIFICATION

This program is designed to help users prepare for their G1 driving test by providing a structured learning environment with both *practice* and *mock tests.*

- **Module-Based Practice**: We broke down all of the available questions into separate modules, where each module contains around 20 questions. Users can practice different modules at their own pace.
- **Mock Test**: The program can simulate a G1 test with randomized questions, helping users become more familiar with the test format.
- **Adaptive Learning**: By offering both modular learning and test simulations, the program ensures that users can effectively build their knowledge. The test result provides users with a list of *incorrectly* answered questions and the *test score*, which enables users to review.

DATABASE



H2 Database

Data Integrity

- Ensure data consistency and prevent duplication.

Querying Capability

- Statistical uses such as filtering, joining, and aggregations.

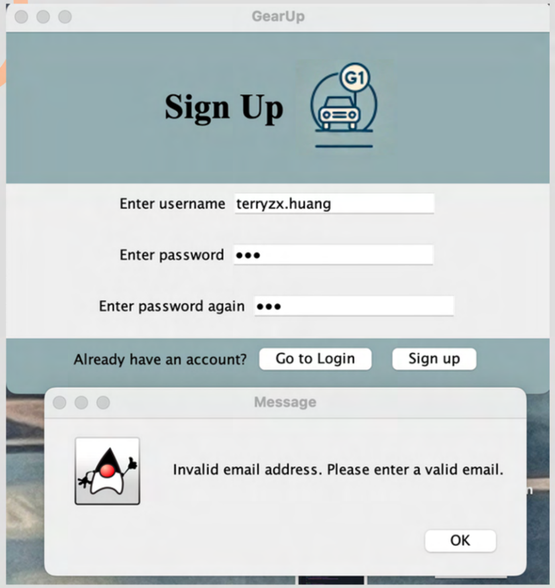
API

The API checks if **email addresses** are valid and reachable before they are stored or used in the project.

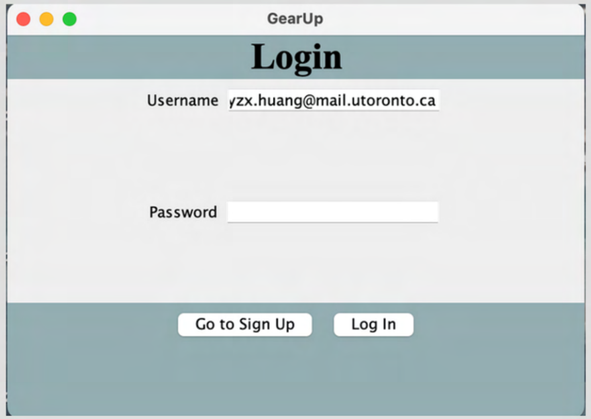
```
final String credentials = USERNAME + ":" + API_KEY;
final String basicAuth = "Basic " + Base64.getEncoder().encodeToString(credentials.getBytes());

// JSON body with the email to validate
final String jsonBody = "{ \"entries\": [ { \"inputData\": \"" + email + "\" } ] }";

// Build the request
final Request request = new Request.Builder()
        .url(API_URL)
        .header( name: "Authorization", basicAuth)
        .header( name: "Content-Type",  value: "application/json")
        .post(RequestBody.create(jsonBody, MediaType.parse( $this$toMediaTypeOrNull: "application/json")))
        .build();
```

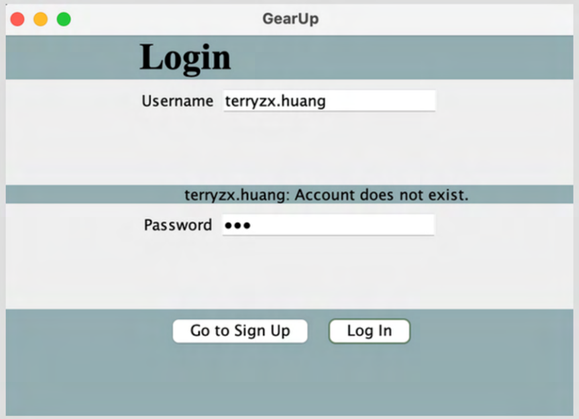
- Ensures accurate and reliable data.
- Reduces errors caused by invalid email entries.
- Advantages compared to other methods (e.g. Regex Validation):
    - Not only checks format; but verifies if an email exists or is reachable.

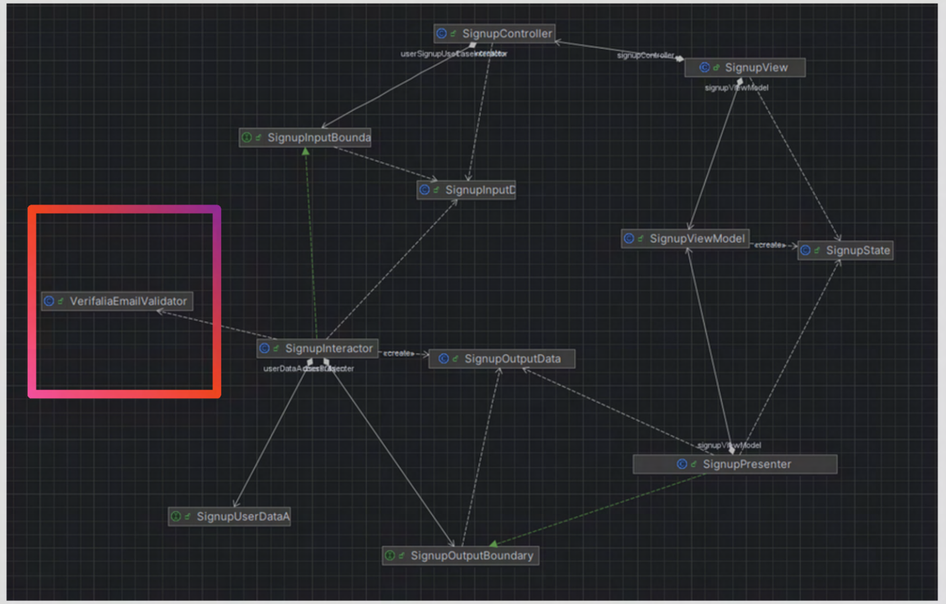
SIGN UP & LOGIN



- Sign Up
  - Only sign the user up if provided valid email address.
  - Only sign the user up if the username exists or two passwords are the same
- Login
  - Only log in if the Username exisit and the user enters the correct password.

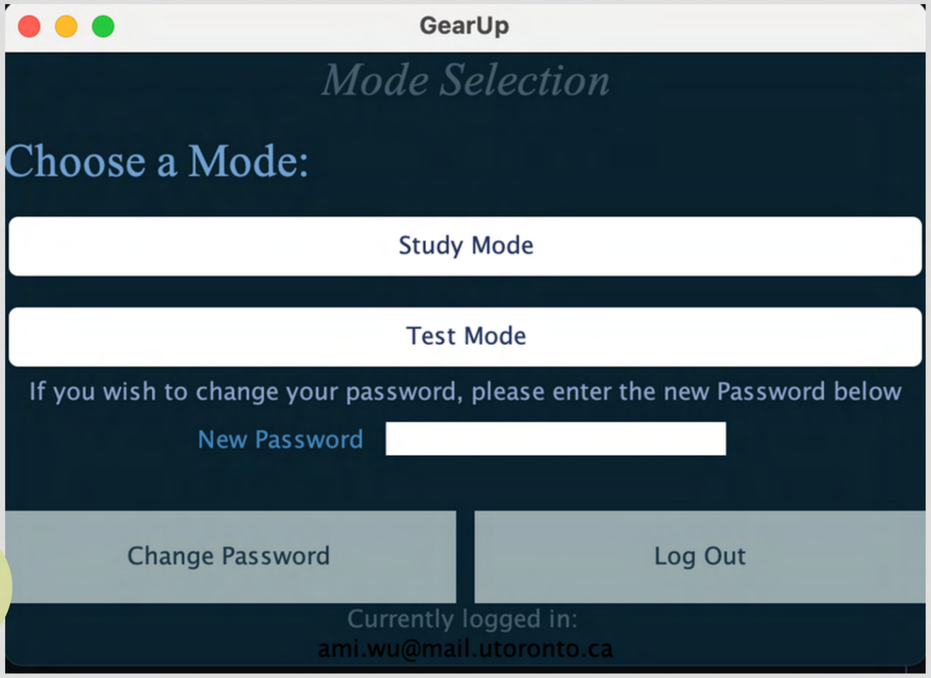


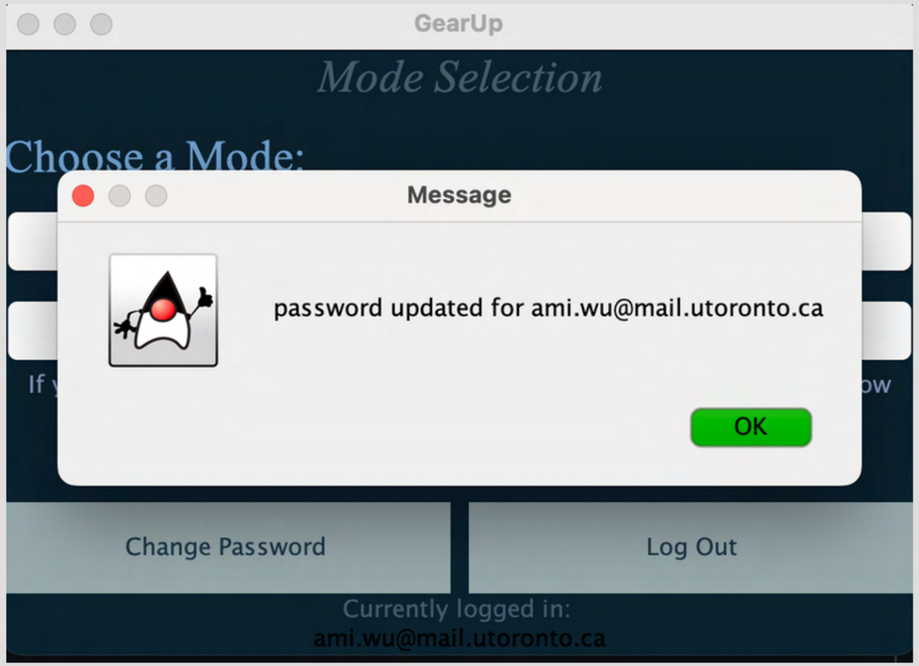




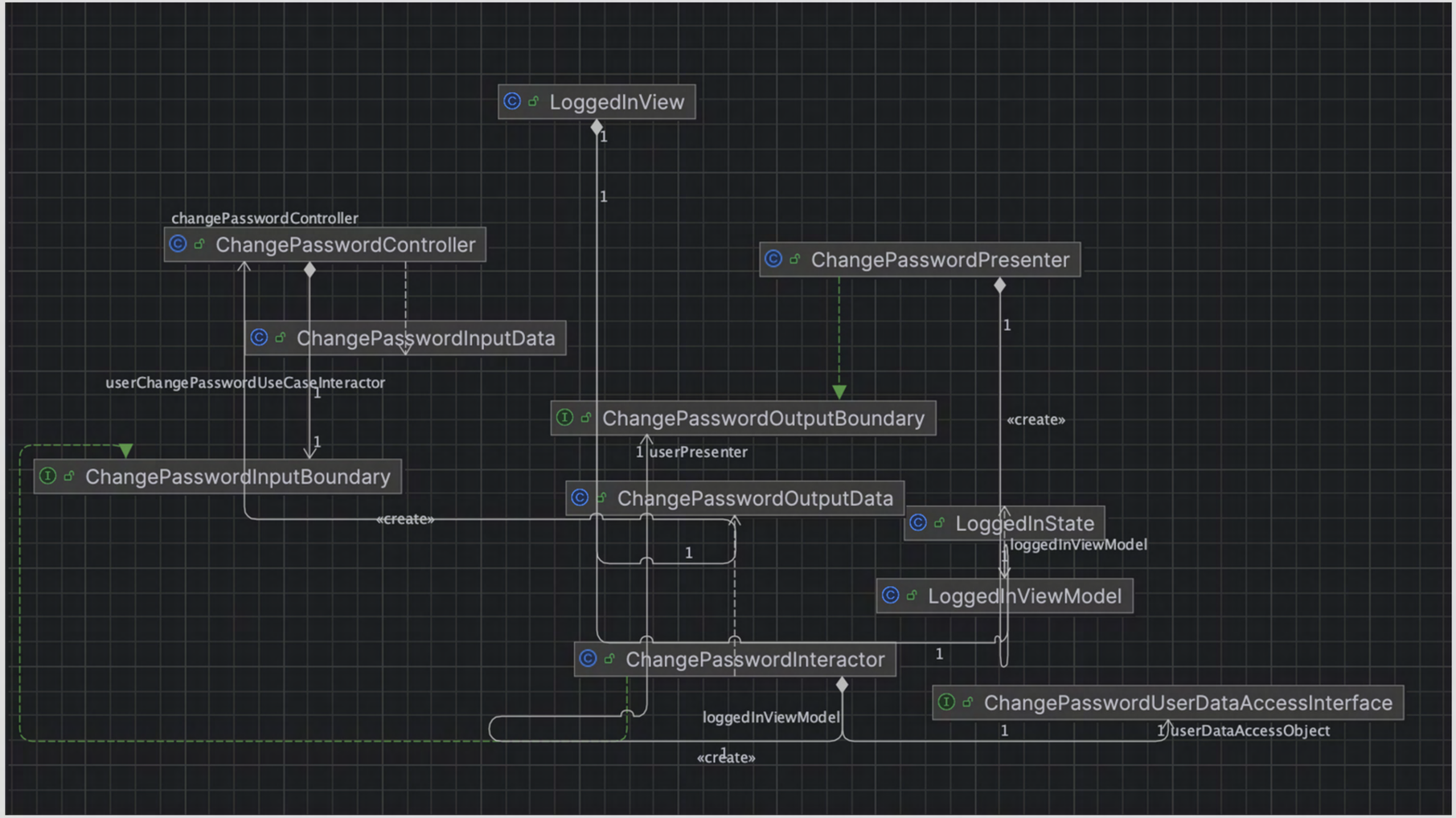
MODE SELECTION

- Users can choose either study mode or test mode after they successfully log in.
- They may also change password when they are logged in.

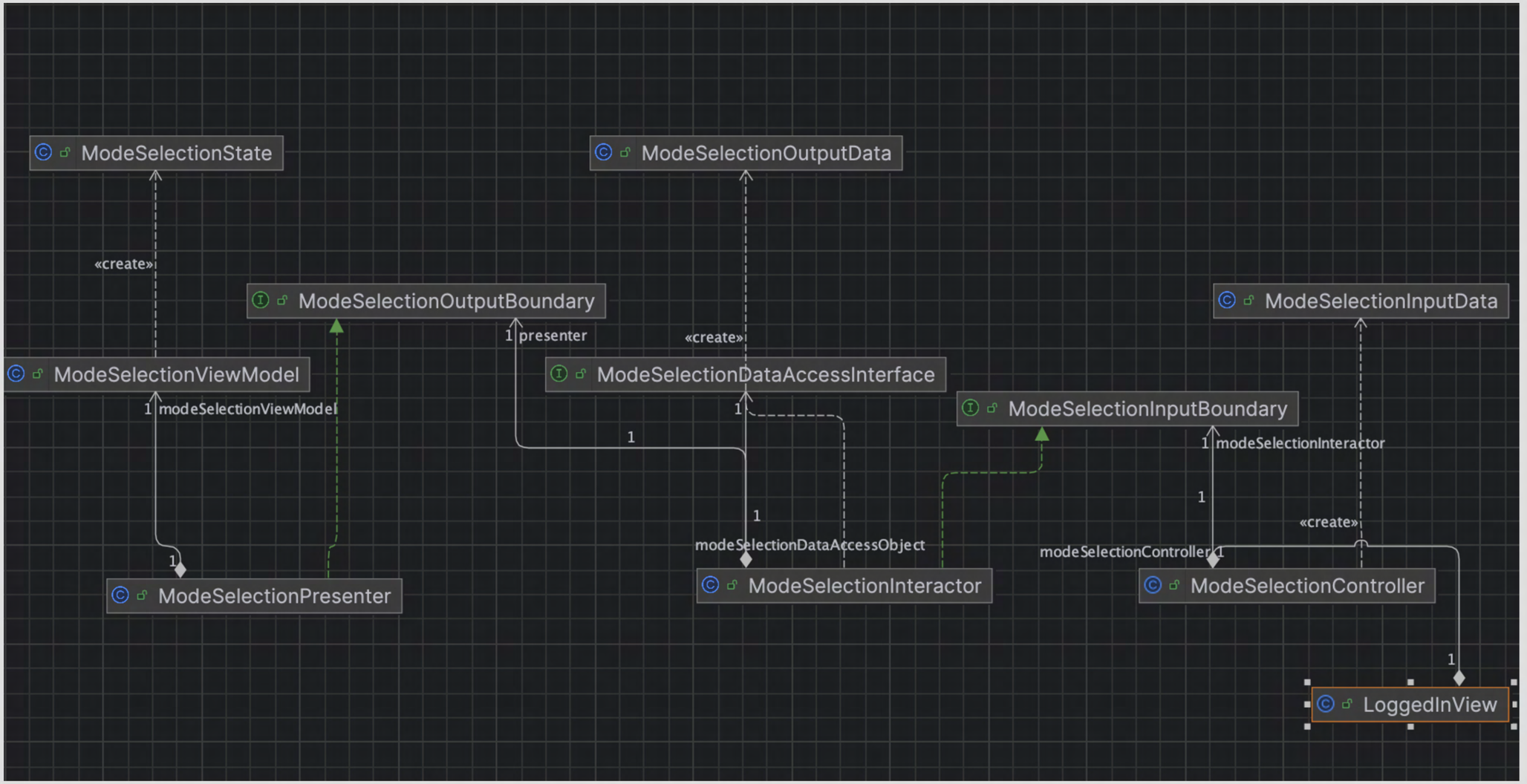




UML - CHANGE PASSWORD



UML - MODE SELECTION



MODE SELECTION

```java
/**
 * The Mode Selection Interactor.
 */
public class ModeSelectionInteractor implements ModeSelectionInputBoundary {  6 usages  amiii +1
    private final ModeSelectionDataAccessInterface modeSelectionDataAccessObject;  2 usages
    private final ModeSelectionOutputBoundary presenter;  5 usages

    public ModeSelectionInteractor(ModeSelectionDataAccessInterface modeSelectionDataAccessInterface,  5 usages  amiii
                                   ModeSelectionOutputBoundary outputBoundary) {
        this.modeSelectionDataAccessObject = modeSelectionDataAccessInterface;
        this.presenter = outputBoundary;
    }

    @Override  amiii +1
    public void execute(ModeSelectionInputData inputData) {
        final String mode = inputData.getSelectedMode();

        if (!isValidMode(mode)) {
            presenter.prepareFailView( errorMessage: "Invalid mode selected, Please choose Study or Test");
        }
        else {
            modeSelectionDataAccessObject.saveSelectedMode(mode);
            final ModeSelectionOutputData outputData = new ModeSelectionOutputData(mode);
            presenter.prepareSuccessView(outputData);
        }
    }

    @Override  2 usages  amiii
    public void switchToStudyModeView() {
        presenter.switchToStudyModeView();
    }

    @Override  2 usages  amiii
    public void switchToTestModeView() {
        presenter.switchToTestModeView();
    }
```

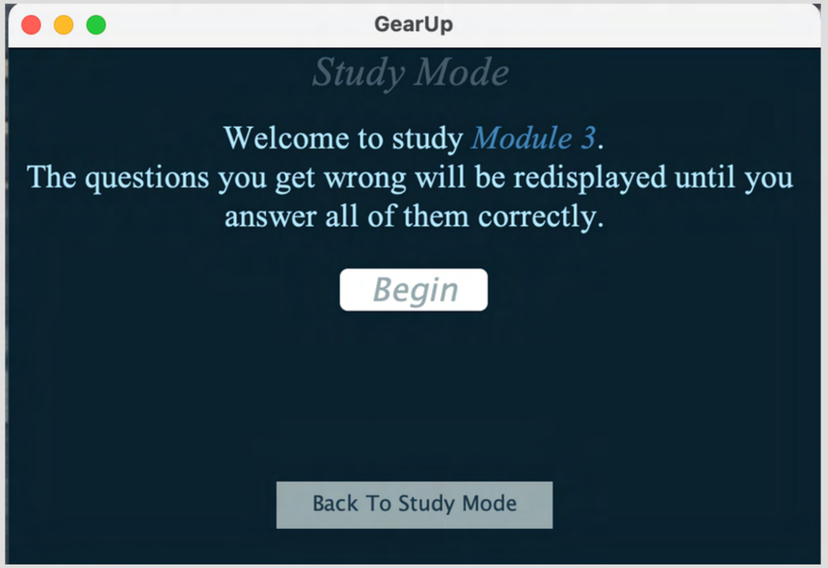
```java
/**
 * The Change Password Interactor.
 */
public class ChangePasswordInteractor implements ChangePasswordInputBoundary {  2 usages  Jonathan Calver
    private final ChangePasswordUserDataAccessInterface userDataAccessObject;  2 usages
    private final ChangePasswordOutputBoundary userPresenter;  2 usages
    private final UserFactory userFactory;  2 usages

    public ChangePasswordInteractor(ChangePasswordUserDataAccessInterface changePasswordDataAccessInterface,  1
                                    ChangePasswordOutputBoundary changePasswordOutputBoundary,
                                    UserFactory userFactory) {
        this.userDataAccessObject = changePasswordDataAccessInterface;
        this.userPresenter = changePasswordOutputBoundary;
        this.userFactory = userFactory;
    }

    @Override  Jonathan Calver
    public void execute(ChangePasswordInputData changePasswordInputData) {
        final User user = userFactory.create(changePasswordInputData.getUsername(),
                                             changePasswordInputData.getPassword());
        userDataAccessObject.changePassword(user);

        final ChangePasswordOutputData changePasswordOutputData = new ChangePasswordOutputData(user.getName(),
                                                                                               useCaseFailed: false);
        userPresenter.prepareSuccessView(changePasswordOutputData);
    }
}
```

STUDY/TEST MODE

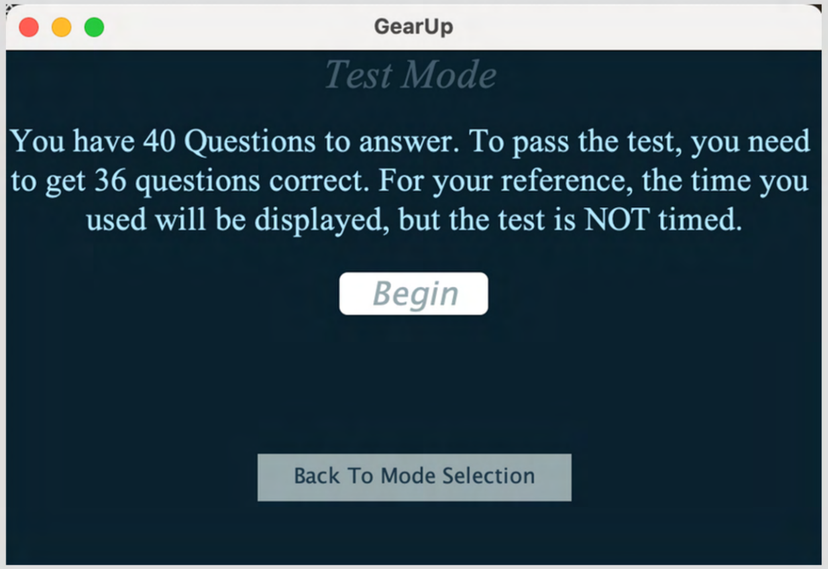
Starter Views

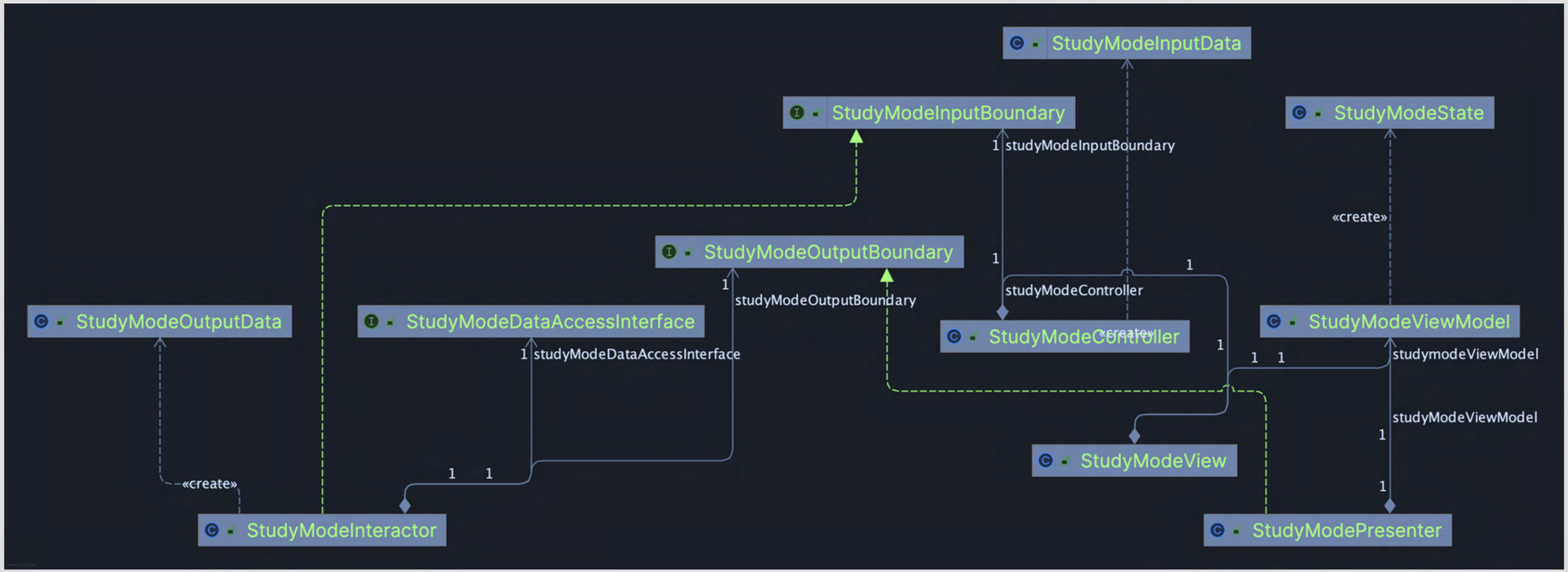
Study Mode

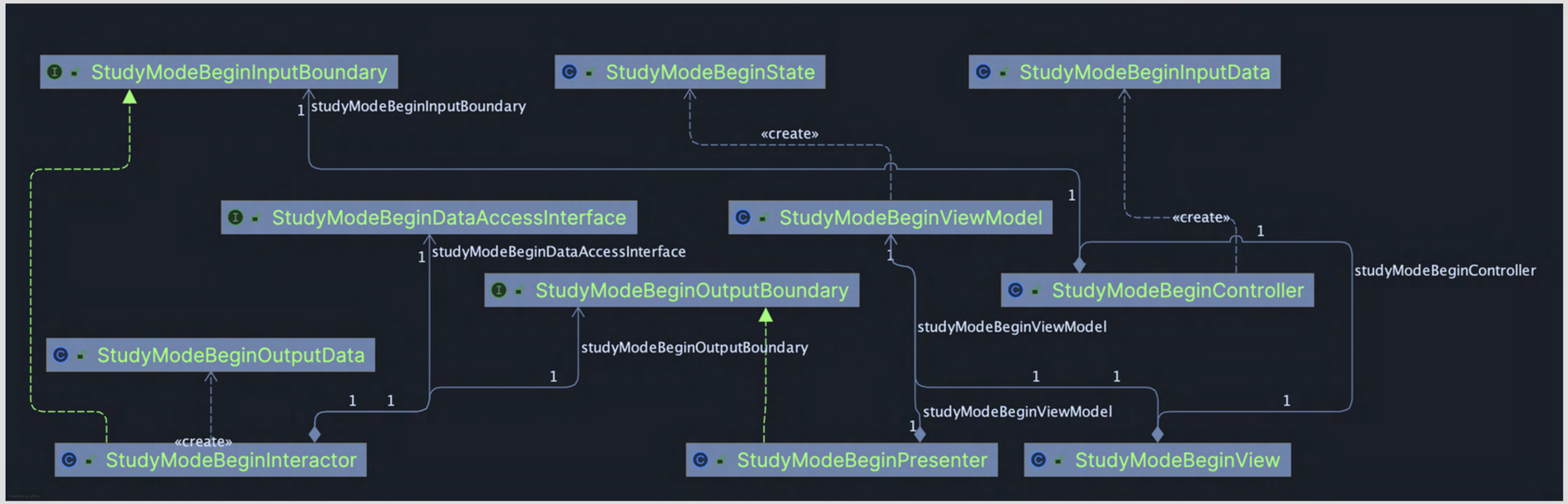
Study Mode Begin



Test Mode Begin




StudyModeInputData
StudyModeInputBoundary
StudyModeState
1 studyModeInputBoundary
«create»
StudyModeOutputBoundary
1
1
studyModeController
1
StudyModeOutputData
StudyModeDataAccessInterface
1 studyModeOutputBoundary
«create»
StudyModeController
StudyModeViewModel
1
1 1
studymodeViewModel
1 studyModeDataAccessInterface
studyModeViewModel
1
StudyModeView
1 1
«create»
1
StudyModeInteractor
StudyModePresenter

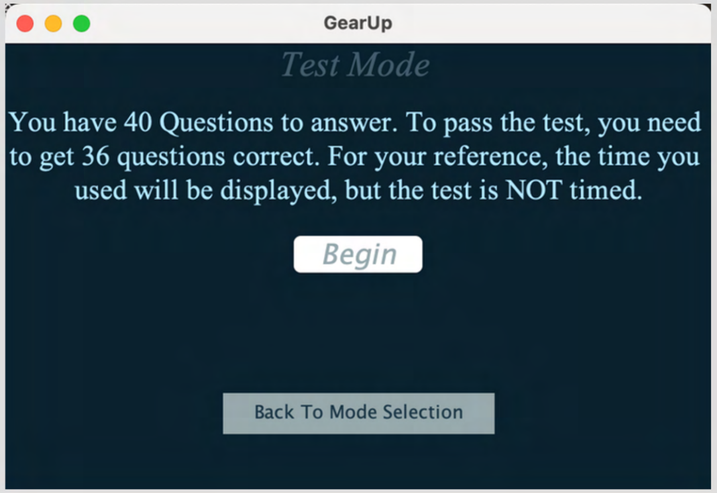

StudyModeBeginInputBoundary
StudyModeBeginState
StudyModeBeginInputData
1 studyModeBeginInputBoundary
«create»
1
StudyModeBeginDataAccessInterface
StudyModeBeginViewModel
«create»
1
1 studyModeBeginDataAccessInterface
1
studyModeBeginController
StudyModeBeginOutputBoundary
StudyModeBeginController
studyModeBeginViewModel
StudyModeBeginOutputData
studyModeBeginOutputBoundary
1
1 1
1 1
studyModeBeginViewModel
1
«create»
1
StudyModeBeginInteractor
StudyModeBeginPresenter
StudyModeBeginView

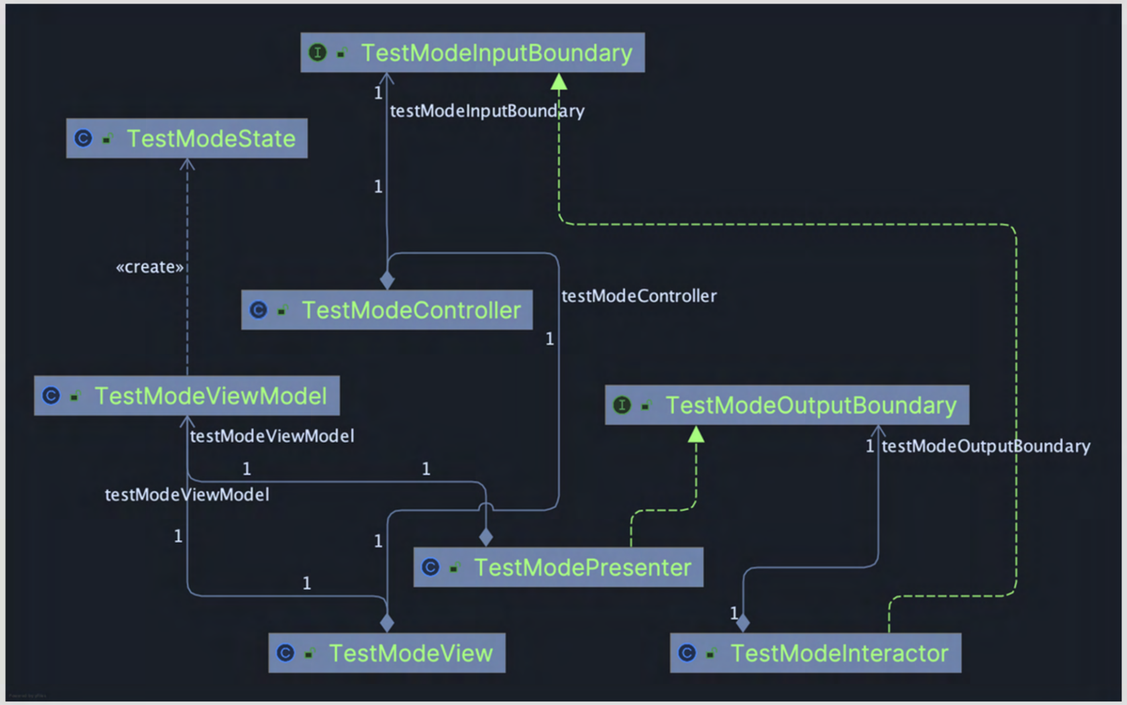
STUDY MODE INTERACTOR

```java
public class StudyModeInteractor implements StudyModeInputBoundary {  6 usages  fries anyone? *
    @Override  fries anyone?
    public void execute(StudyModeInputData studyModeInputData) {
        final String module = studyModeInputData.getModule();

        if (!isValidModule(module)) {
            studyModeOutputBoundary.prepareFailView( errorMessage: "Invalid module selected, Please choose from the
                    + "following (Module 1 to 6)");
        }
        else {
            studyModeDataAccessInterface.setModule(module);
            final StudyModeOutputData outputData = new StudyModeOutputData(module);
            studyModeOutputBoundary.prepareSuccessView(outputData);
        }
    }

    private boolean isValidModule(String module) {  1 usage  fries anyone? *
        final Set<String> validModules = new HashSet<>(Arrays.asList("Module 1", "Module 2", "Module 3",
                "Module 4", "Module 5", "Module 6"));
        return validModules.contains(module);
    }

    @Override  2 usages  fries anyone?
    public void switchToStudyModeBeginView() {
        studyModeOutputBoundary.switchToStudyModeBeginView();
    }

    @Override  fries anyone?
    public void switchToModeSelectionView() {
        studyModeOutputBoundary.switchToModeSelectionView();
    }
}
```

```java
public class StudyModeBeginInteractor implements StudyModeBeginInputBoundary {  6 usages  fries anyone? *

    @Override  fries anyone?
    public void execute(StudyModeBeginInputData studyModeBeginInputData) {
        final String module = studyModeBeginInputData.getModule();

        if (!isValidModule(module)) {
            studyModeBeginOutputBoundary.prepareFailView( errorMessage: "Invalid module, we only have Module 1 to 6.");
        }
        else {
            studyModeBeginDataAccessInterface.setModule(module);
            final StudyModeBeginOutputData outputData = new StudyModeBeginOutputData(module);
            studyModeBeginOutputBoundary.prepareSuccessView(outputData);
        }
    }

    private boolean isValidModule(String module) {  1 usage  fries anyone? *
        final Set<String> validModules = new HashSet<>(Arrays.asList("Module 1", "Module 2", "Module 3",
                "Module 4", "Module 5", "Module 6"));
        return validModules.contains(module);
    }

    @Override  2 usages  fries anyone?
    public void switchToStudyModeQuestionView() {
        studyModeBeginOutputBoundary.switchToStudyModeQuestionView();
    }

    @Override  fries anyone?
    public void switchToStudyModeView() {
        studyModeBeginOutputBoundary.switchToStudyModeView();
    }
}
```

TEST MODE

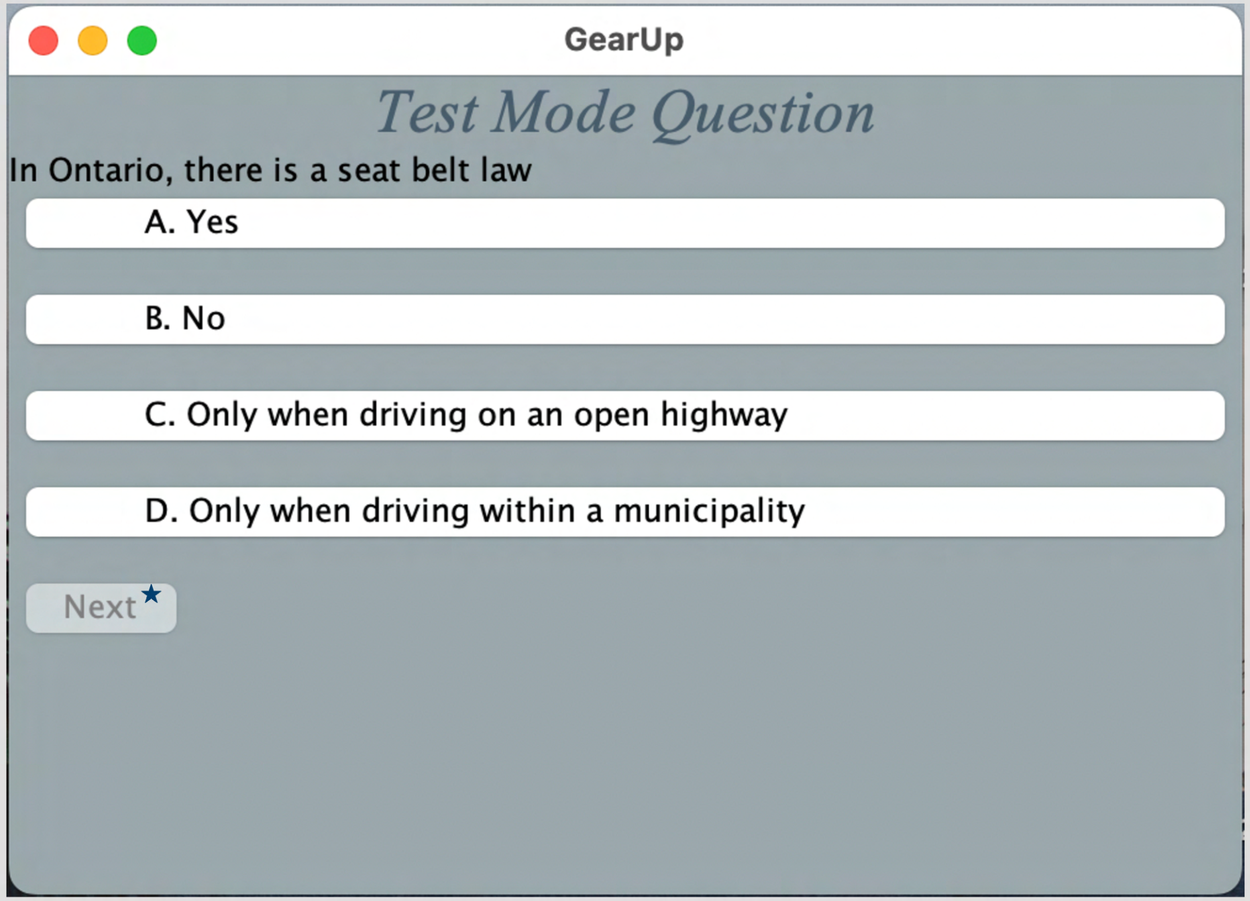


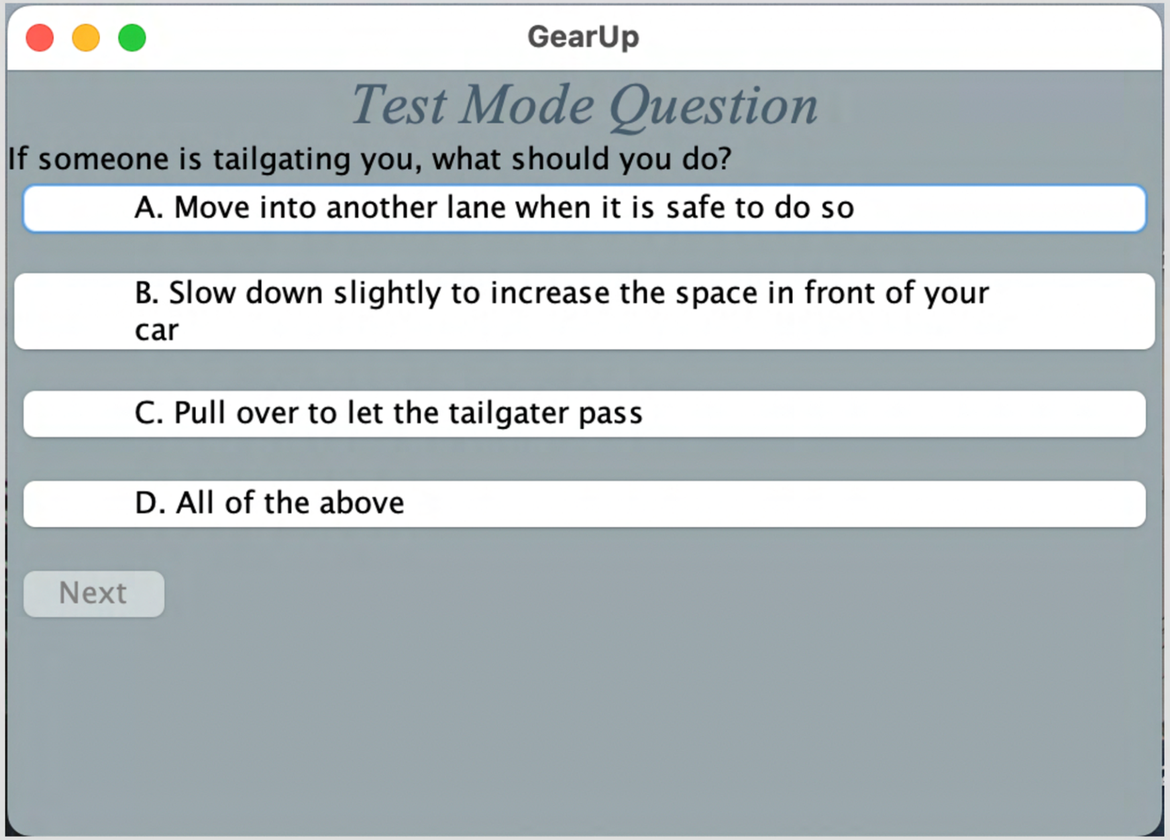


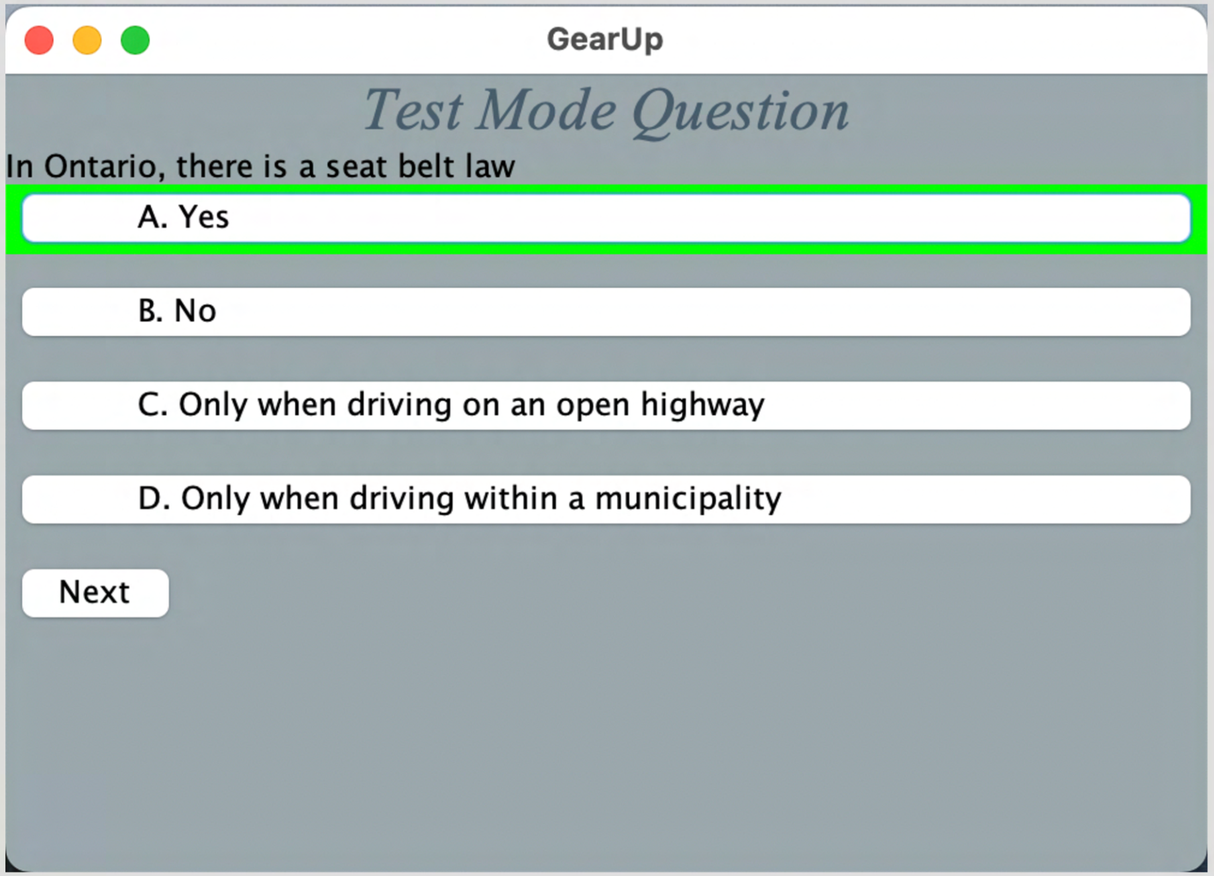

```java
/**
 * The Test Mode Interactor.
 */
public class TestModeInteractor implements TestModeInputBoundary {  4 usages  fries anyone?
    private final TestModeOutputBoundary testModeOutputBoundary;  4 usages

    public TestModeInteractor(TestModeOutputBoundary testModeOutputBoundary) {  3 usages  fries anyone?
        this.testModeOutputBoundary = testModeOutputBoundary;
    }

    @Override  fries anyone?
    public void execute() {
        testModeOutputBoundary.prepareSuccessView();
    }

    @Override  2 usages  fries anyone?
    public void switchToTestModeQuestionView() { testModeOutputBoundary.switchToTestModeQuestionView(); }

    @Override  fries anyone?
    public void switchToModeSelectionView() {
        testModeOutputBoundary.switchToModeSelectionView();
    }
}
```

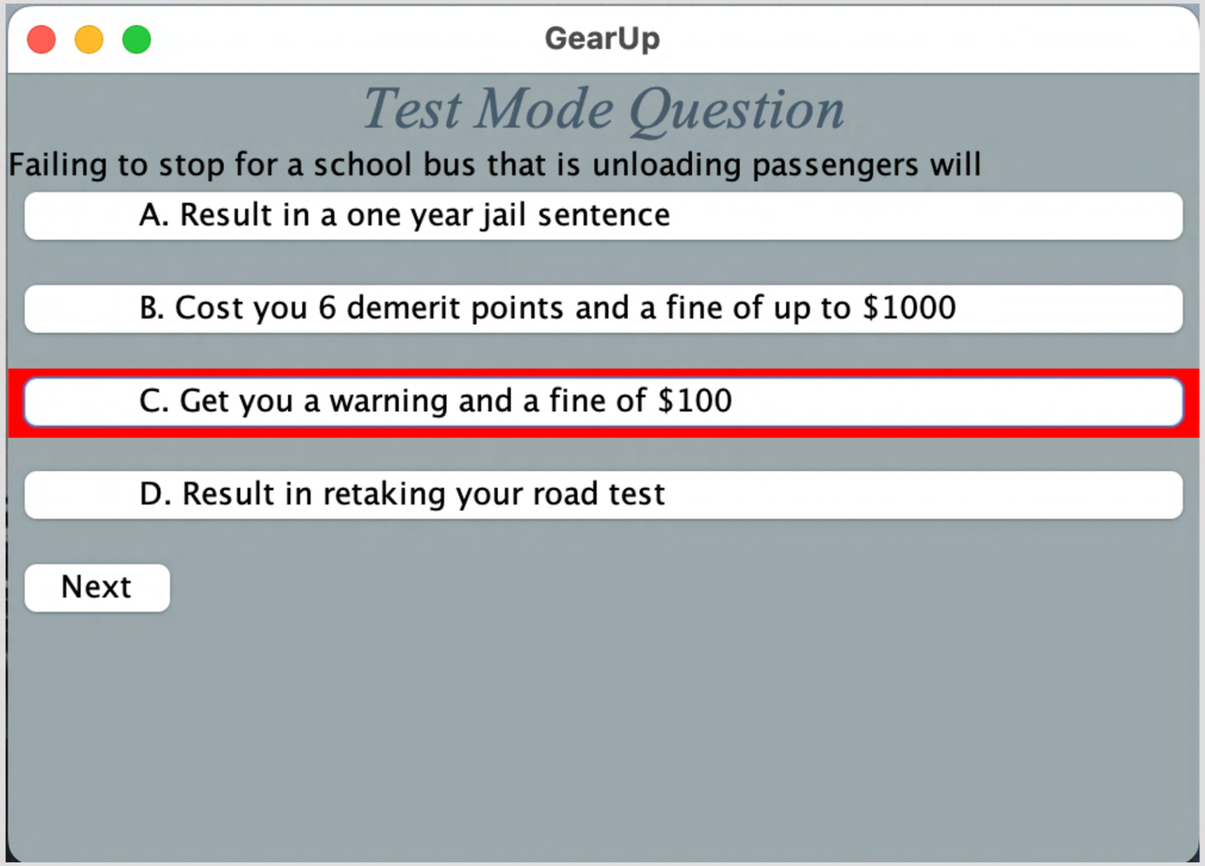
QUESTION DISPLAY

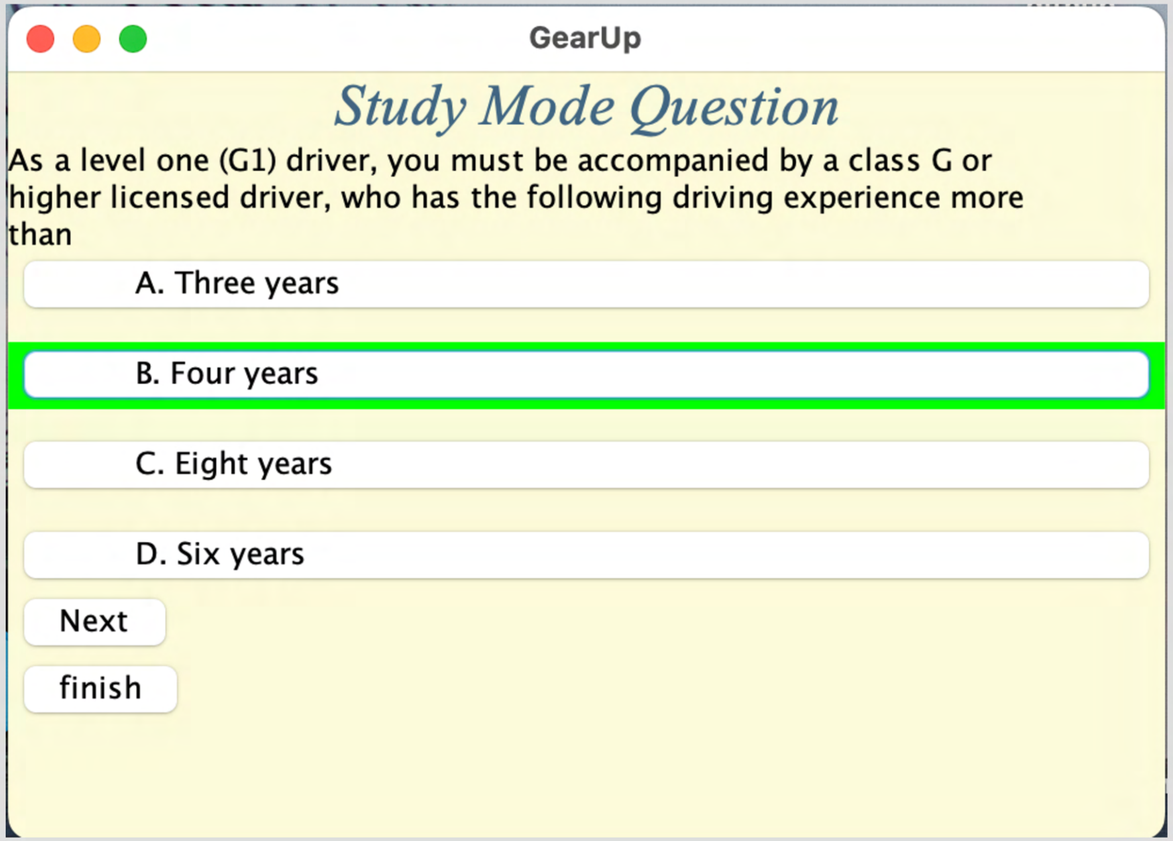
- Study Mode Questions:
  - Access database and totally present 20 questions due to module
  - If the selected answer is :
    - correct ->background turn to green to indicate
    - wrong -> background turn to red to indicate
      add it to the end of the list and retest it.
  - Next button / finish button
- Test Mode Question:
  - Access database and totally present 40 random questions
  - If the selected answer is :
    - correct -> turn green
      accumulator ++
    - wrong -> turn red
      add it to the return wrong question set

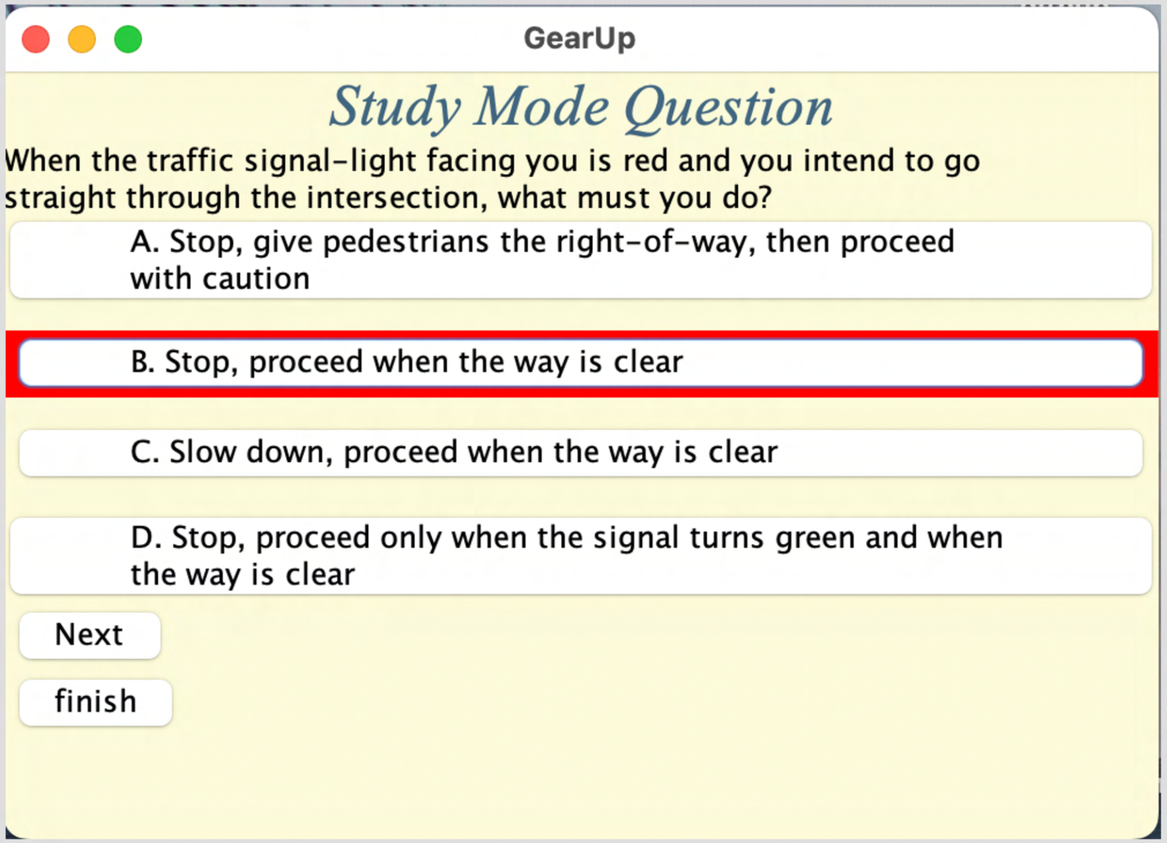

GearUp
Test Mode Question
In Ontario, there is a seat belt law
A. Yes
B. No
C. Only when driving on an open highway
D. Only when driving within a municipality
Next


GearUp
Test Mode Question
If someone is tailgating you, what should you do?
A. Move into another lane when it is safe to do so
B. Slow down slightly to increase the space in front of your car
C. Pull over to let the tailgater pass
D. All of the above
Next

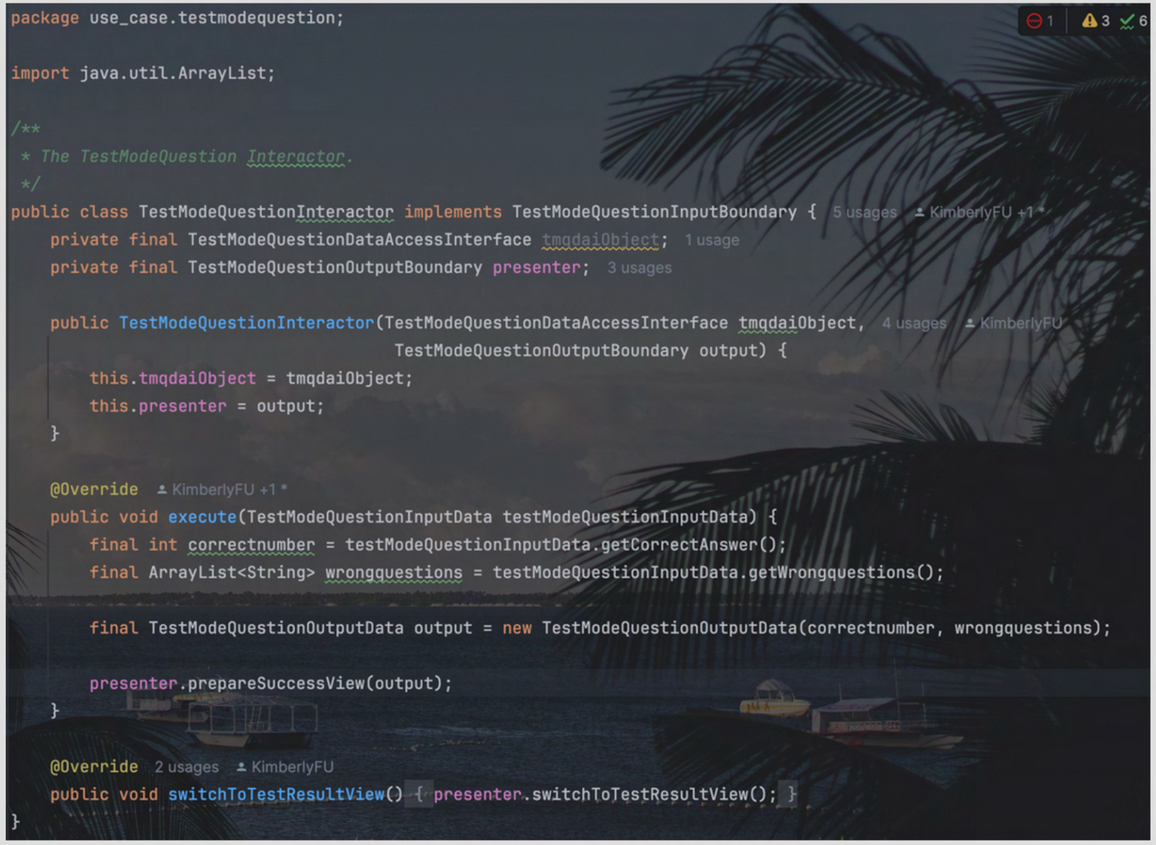

GearUp
Test Mode Question
In Ontario, there is a seat belt law
A. Yes
B. No
C. Only when driving on an open highway
D. Only when driving within a municipality
Next


GearUp
Test Mode Question
Failing to stop for a school bus that is unloading passengers will
A. Result in a one year jail sentence
B. Cost you 6 demerit points and a fine of up to $1000
C. Get you a warning and a fine of $100
D. Result in retaking your road test
Next

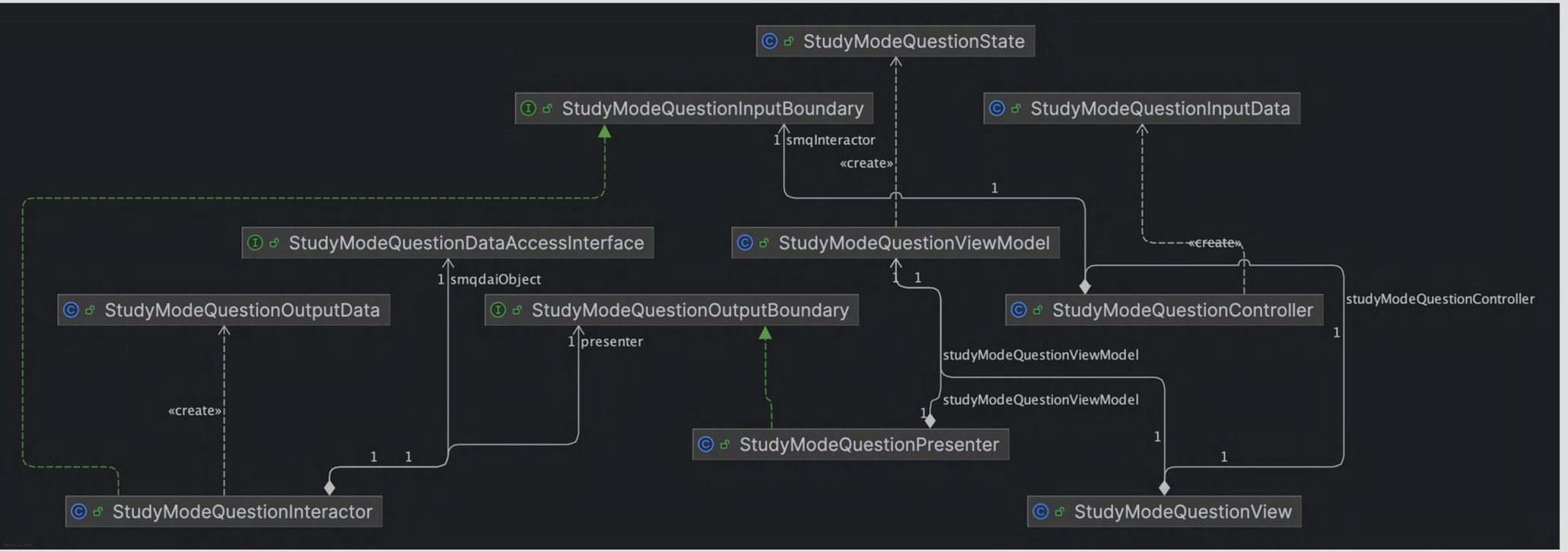

GearUp
Study Mode Question
As a level one (G1) driver, you must be accompanied by a class G or higher licensed driver, who has the following driving experience more than
A. Three years
B. Four years
C. Eight years
D. Six years
Next
finish

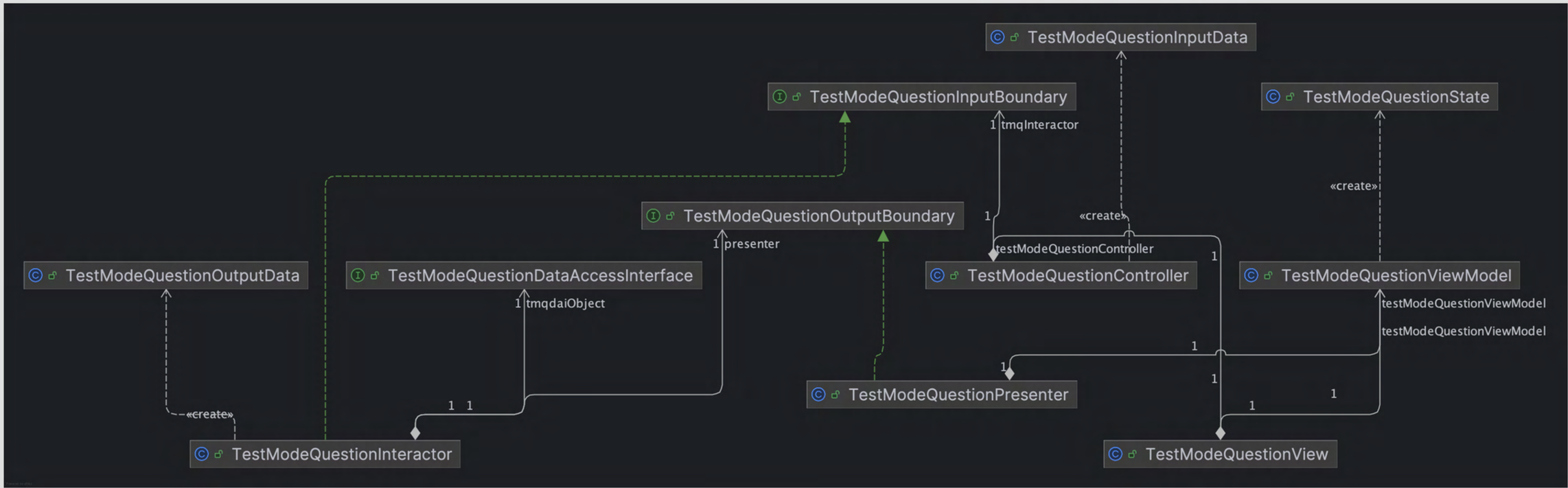

GearUp
Study Mode Question
When the traffic signal-light facing you is red and you intend to go straight through the intersection, what must you do?
A. Stop, give pedestrians the right-of-way, then proceed with caution
B. Stop, proceed when the way is clear
C. Slow down, proceed when the way is clear
D. Stop, proceed only when the signal turns green and when the way is clear
Next
finish

```java
package use_case.studymodequestion;

/**
 * The TestModeQuestion Interactor.
 */
public class StudyModeQuestionInteractor implements StudyModeQuestionInputBoundary {  5 usages  KimberlyFU
    private final StudyModeQuestionDataAccessInterface smqdaiObject;  1 usage
    private final StudyModeQuestionOutputBoundary presenter;  3 usages

    public StudyModeQuestionInteractor(StudyModeQuestionDataAccessInterface smqdaiObject,  4 usages  KimberlyFU
                                       StudyModeQuestionOutputBoundary outputBoundary) {
        this.smqdaiObject = smqdaiObject;
        this.presenter = outputBoundary;
    }

    @Override  KimberlyFU

    public void execute(StudyModeQuestionInputData input) {
        final StudyModeQuestionOutputData output = new StudyModeQuestionOutputData();
        presenter.prepareSuccessView(output);
    }

    /**
     * Switch back to the StudyMode View.
     */
    @Override  KimberlyFU
    public void switchToStudyModeView() {
        presenter.switchToStudyMode();
    }
}
```



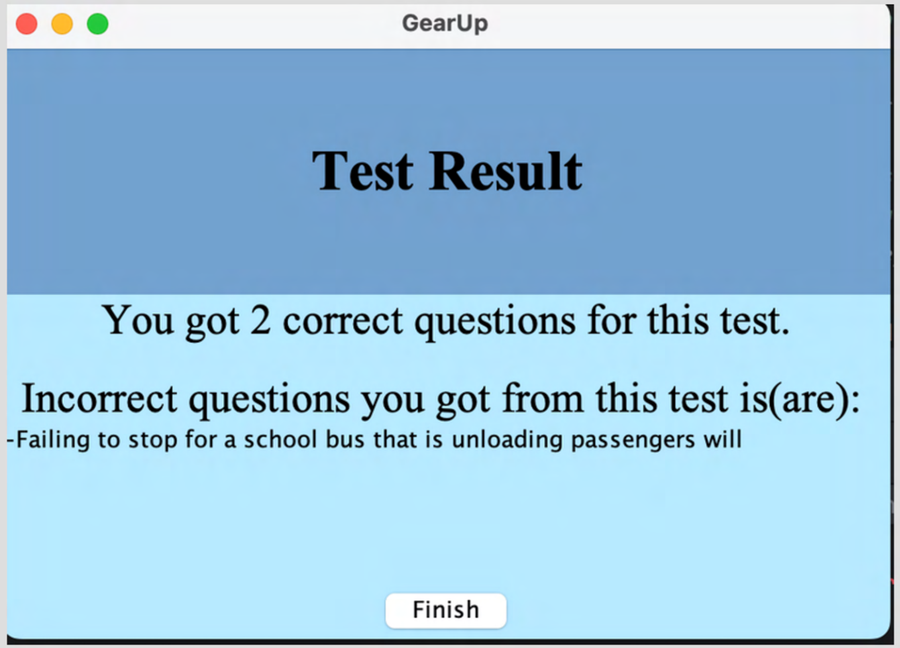
```java
package use_case.testmodequestion;

import java.util.ArrayList;

/**
 * The TestModeQuestion Interactor.
 */
public class TestModeQuestionInteractor implements TestModeQuestionInputBoundary {  5 usages  KimberlyFU +1 *
    private final TestModeQuestionDataAccessInterface tmqdaiObject;  1 usage
    private final TestModeQuestionOutputBoundary presenter;  3 usages

    public TestModeQuestionInteractor(TestModeQuestionDataAccessInterface tmqdaiObject,  4 usages  KimberlyFU
                                      TestModeQuestionOutputBoundary output) {
        this.tmqdaiObject = tmqdaiObject;
        this.presenter = output;
    }

    @Override  KimberlyFU +1 *
    public void execute(TestModeQuestionInputData testModeQuestionInputData) {
        final int correctnumber = testModeQuestionInputData.getCorrectAnswer();
        final ArrayList<String> wrongquestions = testModeQuestionInputData.getWrongquestions();

        final TestModeQuestionOutputData output = new TestModeQuestionOutputData(correctnumber, wrongquestions);

        presenter.prepareSuccessView(output);
    }

    @Override  2 usages  KimberlyFU
    public void switchToTestResultView() { presenter.switchToTestResultView(); }
}
```

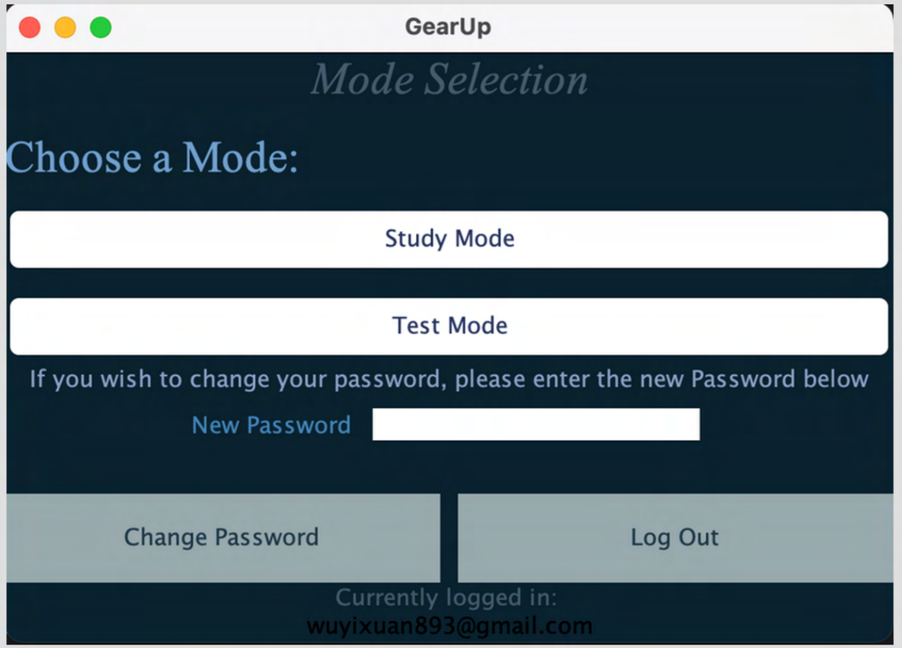

StudyModeQuestionState
StudyModeQuestionInputBoundary
StudyModeQuestionInputData
1 smqInteractor
«create»
1
StudyModeQuestionDataAccessInterface
StudyModeQuestionViewModel
«create»
1 smqdaiObject
1 1
studyModeQuestionController
StudyModeQuestionOutputData
StudyModeQuestionOutputBoundary
StudyModeQuestionController
1 presenter
1
studyModeQuestionViewModel
studyModeQuestionViewModel
«create»
1
1
StudyModeQuestionPresenter
1
1 1
1
StudyModeQuestionInteractor
StudyModeQuestionView


TestModeQuestionInputData
TestModeQuestionInputBoundary
TestModeQuestionState
1 tmqInteractor
«create»
«create»
TestModeQuestionOutputBoundary
1 presenter
1
testModeQuestionController
TestModeQuestionOutputData
TestModeQuestionDataAccessInterface
TestModeQuestionController
TestModeQuestionViewModel
1 tmqdaiObject
testModeQuestionViewModel
testModeQuestionViewModel
1
1
1
«create»
1 1
1
1
TestModeQuestionPresenter
TestModeQuestionInteractor
TestModeQuestionView

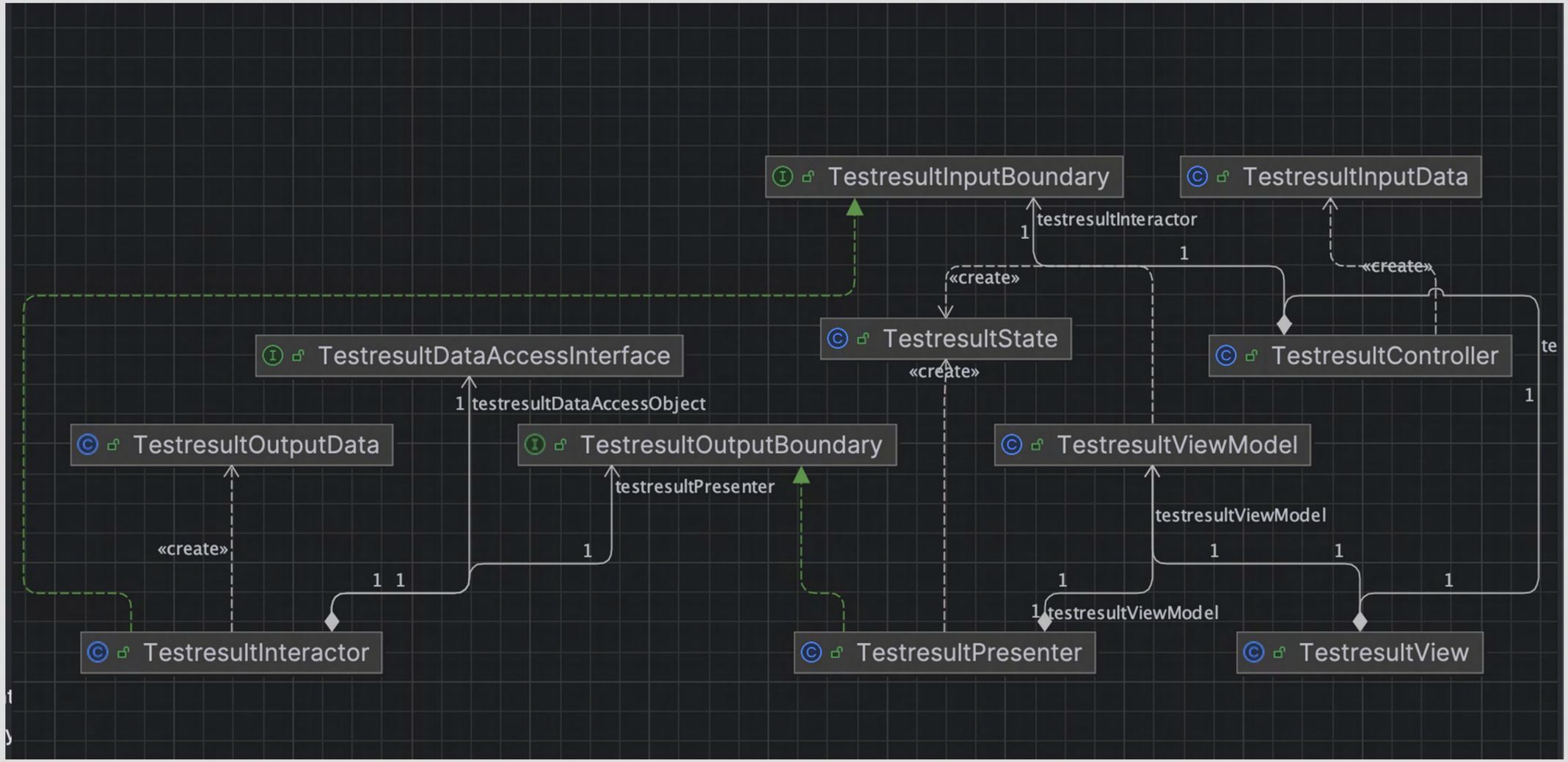
TEST RESULT

- Test result is displayed so that the users know how many questions they do correctly, as well as what questions(s) they do incorrectly





UML - TEST RESULT



TEST RESULT

```java
public class TestresultInteractor implements TestresultInputBoundary {  6 usages  amiii
    private TestresultDataAccessInterface testresultDataAccessObject;  3 usages
    private TestresultOutputBoundary testresultPresenter;  4 usages

    public TestresultInteractor(TestresultDataAccessInterface testresultDataAccessInterface,  5 usages  amiii
                                TestresultOutputBoundary testresultOutputBoundary) {
        this.testresultDataAccessObject = testresultDataAccessInterface;
        this.testresultPresenter = testresultOutputBoundary;
    }

    @Override  amiii
    public void execute(TestresultInputData testresultInputData) {
        // Validate the input data
        if (!isValid(testresultInputData)) {
            testresultPresenter.prepareFailView( errorMessage: "Invalid test result provided");
        }
        else {
            final int correctQuestions = testresultInputData.getCorrectQuestions();
            final ArrayList<String> incorrectQuestions = testresultInputData.getIncorrectQuestions();
            // If the input is valid, proceed with prepare success view.
            testresultDataAccessObject.saveCorrectQuestions(correctQuestions);
            testresultDataAccessObject.saveIncorrectQuestions(incorrectQuestions);
            final TestresultOutputData outputData = new TestresultOutputData(correctQuestions, incorrectQuestions);
            testresultPresenter.prepareSuccessView(outputData);
        }
    }
```

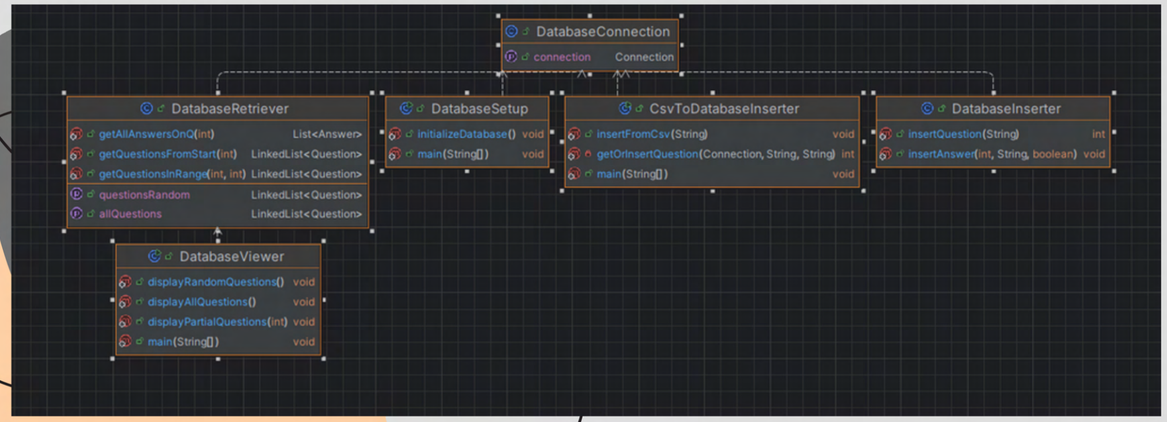
```java
// Private method to validate the input data
private boolean isValid(TestresultInputData testresultInputData) {  1 usage  amiii
    return testresultInputData.getCorrectQuestions() >= 0
            && testresultInputData.getTime() > 0 && testresultInputData.getIncorrectQuestions() != null;
}
```

- Private method helps to check which one of the two scenarios this falls into!

SOLID

- **Single-responsibility Principle (SRP)**

- The Database files adhere to the Single Responsibility Principle (SRP) by focusing each class on a distinct responsibility. For example, CsvToDatabaseInserter handles CSV file processing and insertion, DatabaseRetriever focuses on data retrieval... This clear separation ensures maintainability and modularity. Each class handles a single aspect of the system, minimizing overlap and making the code easier to maintain and extend.



- **Open-closed Principle (OCP)**

- The TestModeQuestion use case follows OCP by being open for extension but closed for modification. Instead of modifying the existing class to add new test types, new classes or interfaces (e.g., MultipleChoiceQuestion, TrueFalseQuestion) could be extended in the future to extend the existing MockTest functionality, ensuring the core mock test logic remains unchanged.

DESIGN PATTERN

- Creational Design Pattern
  - Dependency Injection

- Before

```
public class AppBuilder {  22 usages  Jonathan Calver +4 *
    private final JPanel cardPanel = new JPanel();  12 usages
    private final CardLayout cardLayout = new CardLayout();  2 usages
    // thought question: is the hard dependency below a problem?
    private final UserFactory userFactory = new CommonUserFactory();  2 usages
    // thought question: is the hard dependency below a problem?
    private final InMemoryUserDataAccessObject userDataAccessObject = new InMemoryUserDataAccessObject();  4 usages
```

- After

```
public class AppBuilder {  22 usages  Jonathan Calver +4 *
    private final JPanel cardPanel = new JPanel();  12 usages
    private final CardLayout cardLayout = new CardLayout();  2 usages

    // thought question: is the hard dependency below a problem?
    private final UserFactory userFactory;  3 usages
    // thought question: is the hard dependency below a problem?
    private final InMemoryUserDataAccessObject userDataAccessObject;  5 usages

    public AppBuilder(UserFactory userFactory, InMemoryUserDataAccessObject userDataAccessObject) {  1 usage  Jonatha
        this.userFactory = userFactory;
        this.userDataAccessObject = userDataAccessObject;
        cardPanel.setLayout(cardLayout);
    }
```

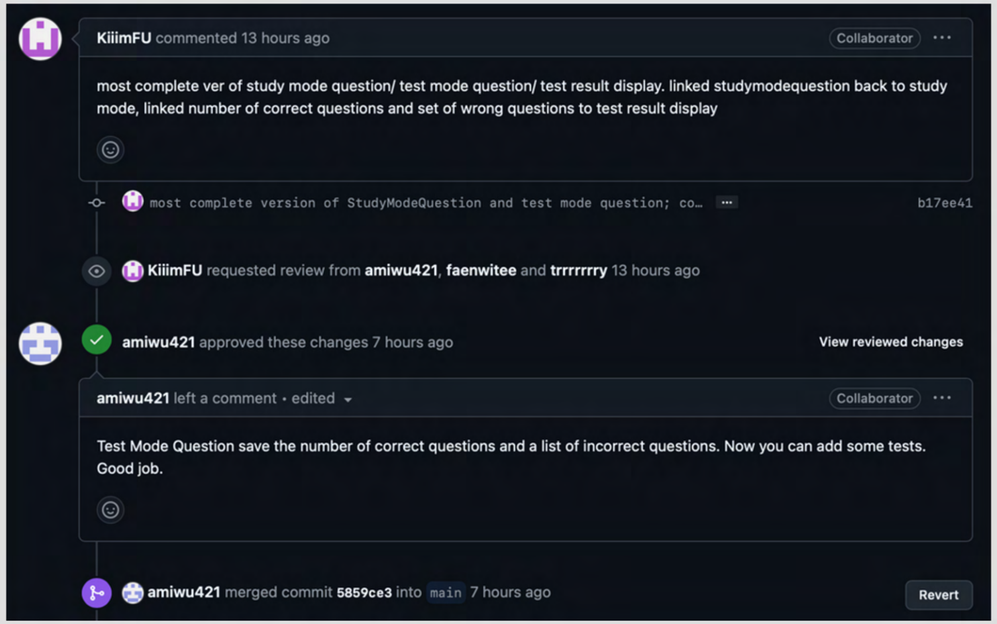
```
/**
 * The Main class of our application.
 */
public class Main {  Jonathan Calver +3 *
    /**
     * Builds and runs the CA architecture of the application.
     * @param args unused arguments
     * @throws SQLException throw exception
     */
    public static void main(String[] args) throws SQLException {  fries anyone? +3 *
        // Create instances of dependencies
        final UserFactory userFactory = new CommonUserFactory();
        final InMemoryUserDataAccessObject userDataAccessObject = new InMemoryUserDataAccessObject();

        // Inject dependencies into AppBuilder
        final AppBuilder appBuilder = new AppBuilder(userFactory, userDataAccessObject);
```

Benefits:

- Loose Coupling
- Testability
- Flexibility
- Dependency Inversion Principle

CODE ORGANIZATION

- Our project uses the Inside/Outside packing Scheme, which best aligns with the Clean Architecture. This packing scheme helps with decoupling, which ensures loose coupling and high cohesion in our project.
- The main four layers in our project are the **Entities**, **Interface Adapter**, **Use case**, and **View**.
- The inside layer contains the business rules and core logic. The outside contains the infrastructure, UI, and other layers that provide interactions with users or other systems.
- The middle layers (interface adapter and use case) are how data are translated and processed.



CODE QUALITY

- Our team maintained high code quality by constantly using CheckStyle while implementing new features, using pull requests for code reviews and meeting weekly in person to review code together.
- This approach ensured consistency, adherence to best practices, and allowed for collaborative feedback before merging changes.

ACCESSIBILITY REPORT

- **Principle 2: Flexibility in Use**
  - Allows users to practice at their own pace through different modules.
- **Principle 4: Perceptible Information**
  - Provides clear feedback on incorrect answers
- **Target Audience**
  - Individuals preparing for the G1 driving test. However, users with visual impairments may face challenges unless accessible features, like screen reader compatibility, are incorporated.

README

- The README file is formating in markdown (.md) located in the repository (https://github.com/faenwitee/GearUp)
- There is a “Table of Contents” that make this README easy to read and naviagte.
- Apporapriate images, title, and font are used to ensures better readability.

Table of Contents

FUNCTIONALITY DEMONSTRATION

GearUp

Study Mode Question

When 15 or more Demerit Points have accumulated on a record, the driver's licence is suspended

A. Automatically, and for 30 days from receipt of licence

B. At the discretion of the Ministry

C. Only if the licence is NOT used for business purposes

D. For 3 months

Next

finish

THANK YOU!