CSC207 Project Group 047

Yicong Luo, Khloe Tsang, Wenzhen Wang, Peijun Lu, Qin Xu, Zhaoyu Yan

Specification

- Grocery order reservation app
- Phase 0
  - Registration and login
- Phase 1
  - View and order groceries
  - Add and remove items from shopping cart
  - Check out order and confirm pickup
- Phase 2
  - Change username and PIN
  - Payment and credit system
  - Order history
  - Colour settings

Design Decisions

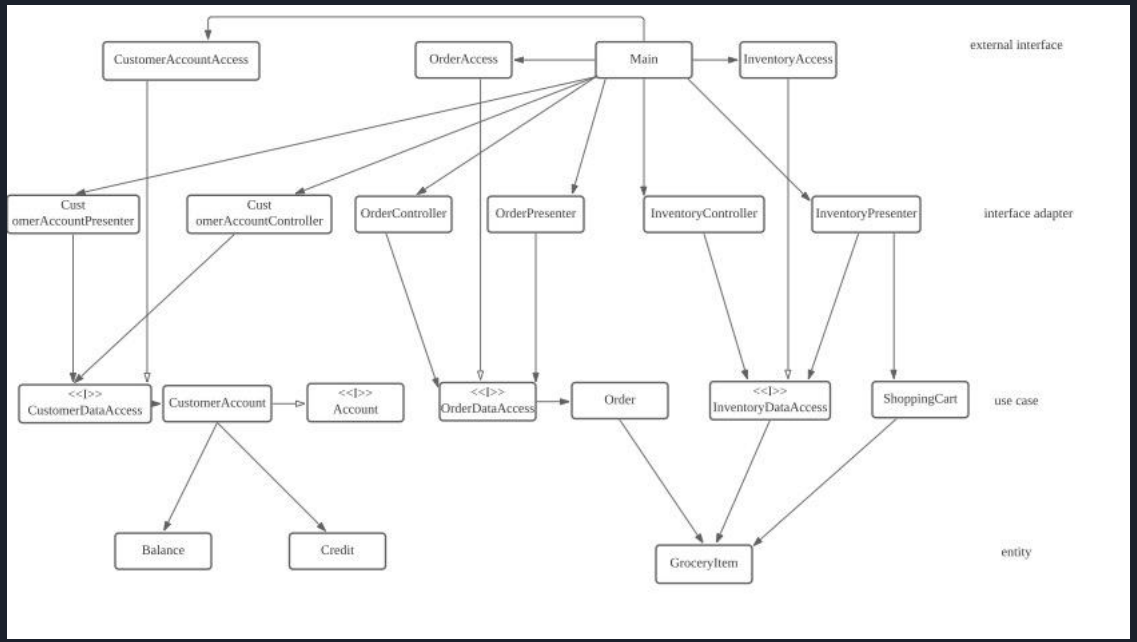
- Renaming of classes
- Extraction of some parts of controller classes into gateways and presenters
- Change of text file location

Clean Architecture

- Used the Dependency Inversion technique (see pull request #75) to ensure dependency is from low level modules to high level modules
- (Continued on next slide)

Clean Architecture (cont.)

- UML diagram as seen below



SOLID Principles

- Single Responsibility Principle
  - All classes only have one responsibility
- Open/Closed Principle
  - No action will alter existing classes
- Liskov Substitution Principle
  - Our Controller classes use a Data Access interface at the use case level, but in actual usage we use the Main class to pass a concrete class that implements the interface
- (continued on next slide)

SOLID Principles (cont.)

- Interface Segregation Principle
  - No classes contain features they don't need, and all concrete classes that implement an interface override and use the corresponding methods
- Dependency Inversion Principle
  - Our Data Access classes are in the outermost layers, and their interfaces are in the use case layer

Design Pattern

- Iterator Pattern
  - Used in our three Presenter classes
  - See Pull requests #91 and #96
- Dependency Injection Pattern
  - Used in our controller classes

Package Strategy and Code Refactoring

- Classes are put into packages by four layers, from high level to low level
  - entity
  - use_case
  - interface_adapter
  - external_interface
- Another folder named ‘file’ to store text files under database
- Combined classes with the same responsibility (see pull request #22)
- Shorten long methods (see pull request #46)

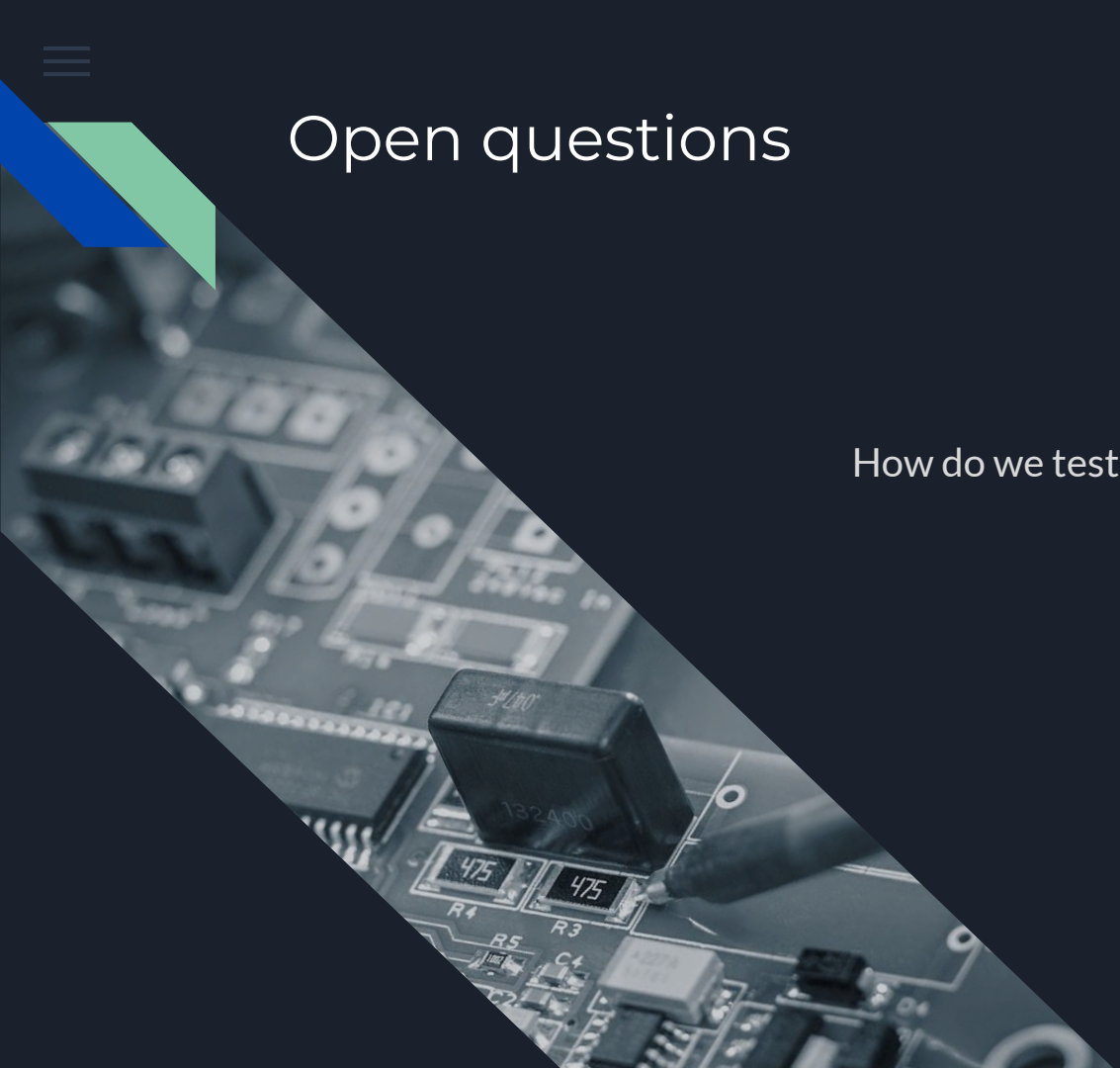
Accessibility

- Equitable use
  - Three text colour options to make user experience more equitable
- Flexibility in use
  - Not applicable at current stage as the program can only be used in command line, can be implemented in later stages
- Simple and intuitive use
  - In-app instructions are clear, short, and easy-to-understand
- Perceptible information
  - Different coloured messages to allow for easy distinguishability of information
- (continued on next slide)

Accessibility (cont.)

- Tolerance for error
  - Warning messages for bad input
- Low physical effort
  - Only requires integer input from the user
  - Serialization for colour settings
- Size and space for approach and use
  - Not applicable at current stage, but can be implemented in later stages

Open questions



How do we test the UI with user input?

Thank you!