Eat-net

Final Report

By
Chen Hao Cheng
Peter Delevoryas
Hulaif Muhammad Ilyas
Eric Murphy

1. Features that were implemented

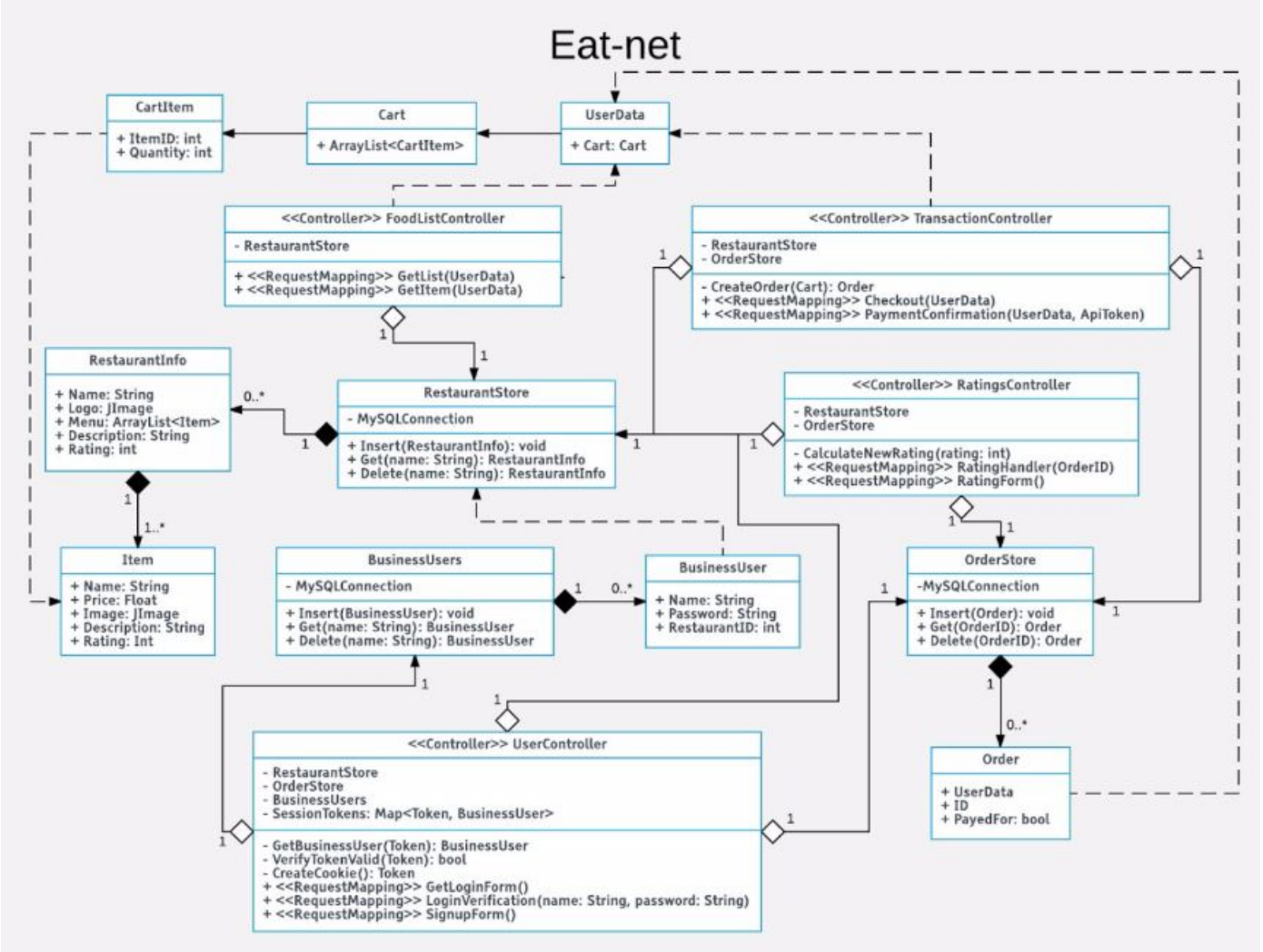
Feature ID	Title
UR-1	Each business user can login to corresponding restaraunt admin page
UR-2	Each business user can sign up
UR-4	Add food item to shopping cart
UR-8	Being able to view orders from customers
UR-10	View restaurants and items

2. Features that were not implemented

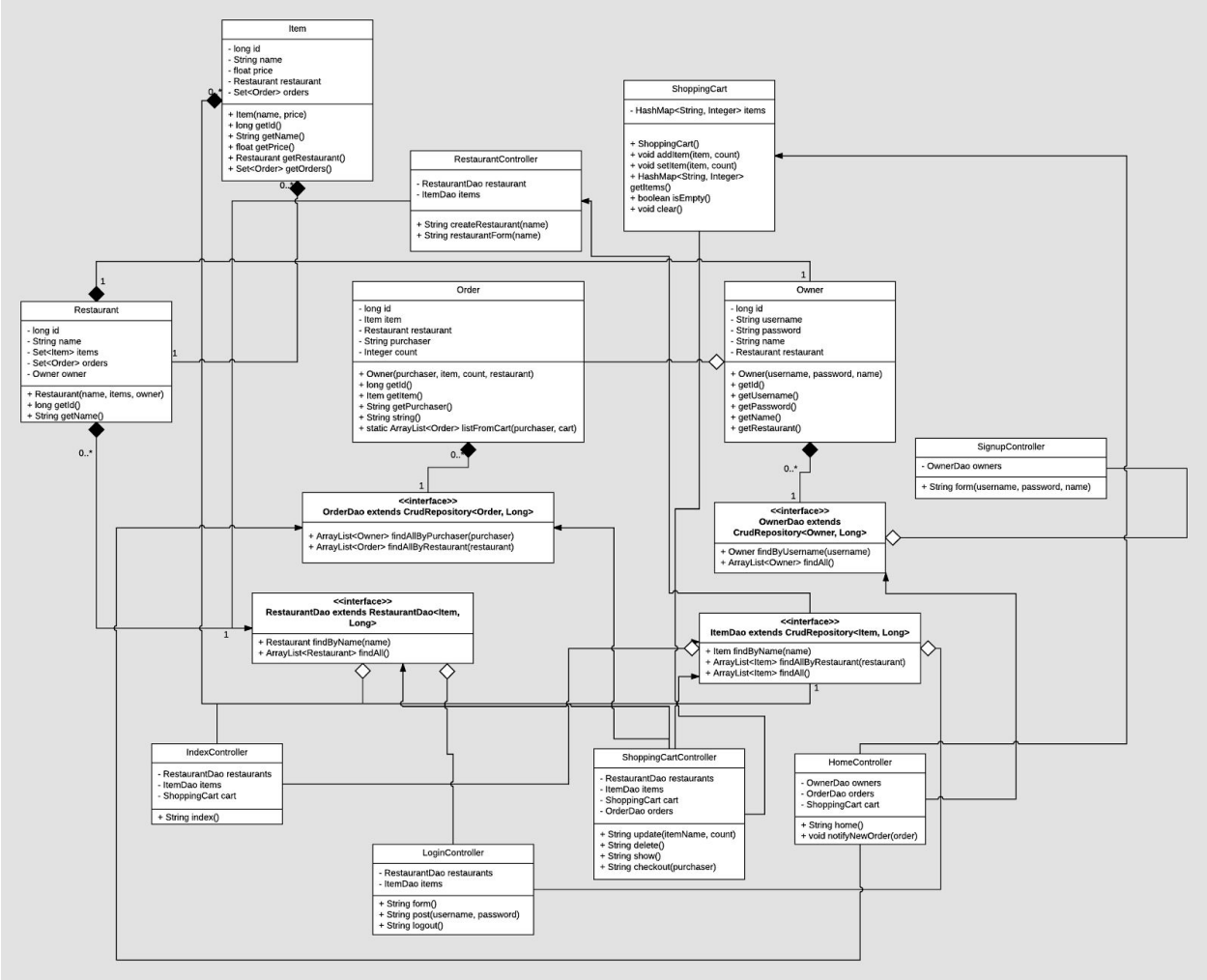
Feature ID	Title
BR-1	Add advertisements to the site to generate revenue
BR-2	Being able to view site usage
BR-3	Being able to pay fees to Paypal
UR-3	Each business user can specify restaurant information (menu, prices, description, address)
UR-5	Review items in shopping cart
UR-6	Pay for items in shopping cart
UR-7	Generate monthly report for each business
UR-9	Rate restaurants and items
NFR-1	Payment Encryption
NFR-2	Rate restaurants and items

3. Part 2 Class Diagram vs Final Class Diagram

a. Part 2 Diagram:



b. Final Diagram:



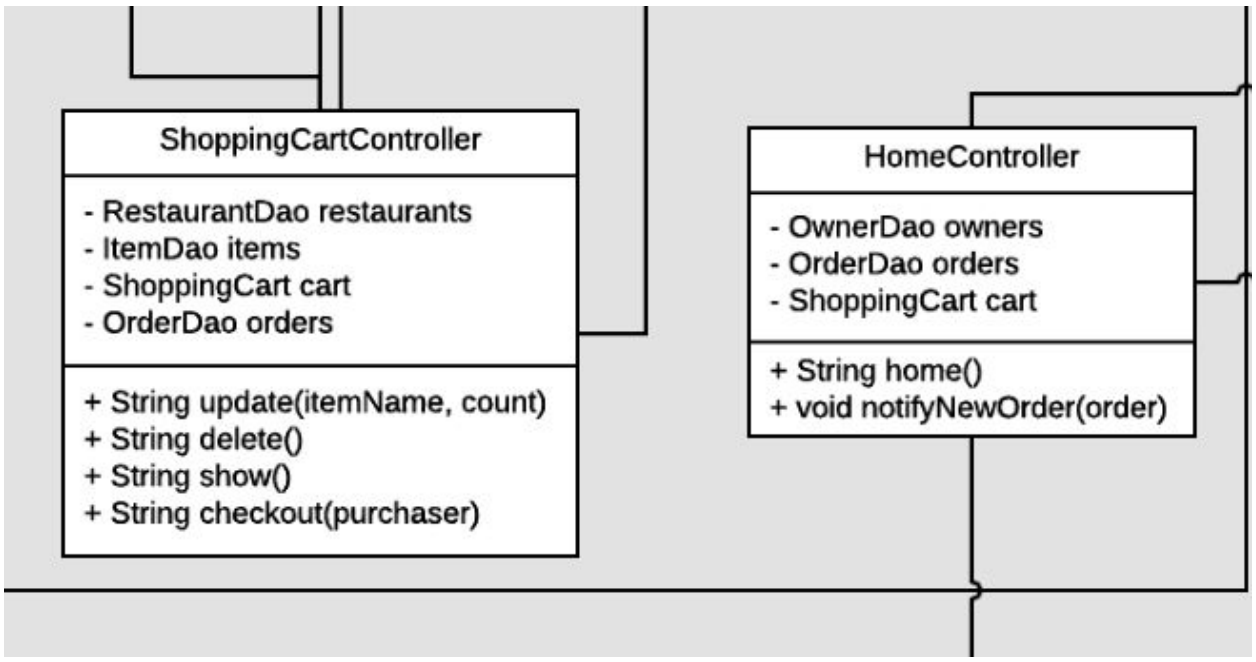
c. What's different and Why

Quite a bit changed from the start to the end, nearly everything in fact. Our final design was to use the repository pattern (specifically, using Hibernate's Autowiring features) to implement persistent storage, using a Dao (data access object) to store and create an interface for accessing every domain object (Restaurant, Owner, Order, Item). Whenever a controller needs access to some data, it includes a data access object, which internally interfaces with a MySQL database that we configured. We also partially implemented some of the observer pattern. Our original intent was that when a user goes to ShoppingControllerCheckout.checkout(), their new order appears on the restaurant owner's homepage (controlled by the HomeController). We did not implement updating the business user's home page using web sockets, but we did make it so that there is a notification method available. The ShoppingCartController notifies the HomeController that a new order has been made, and the home controller decides how to handle it (it saves it to the order table). Registering has not been implemented, because it is assumed that all business users want to register for updates, and so all orders should trigger an update for the corresponding user, but this could be changed by filtering out restaurant owners with a special value in their profile set.

4. Design Patterns Used

a. Observer

i. Diagram:

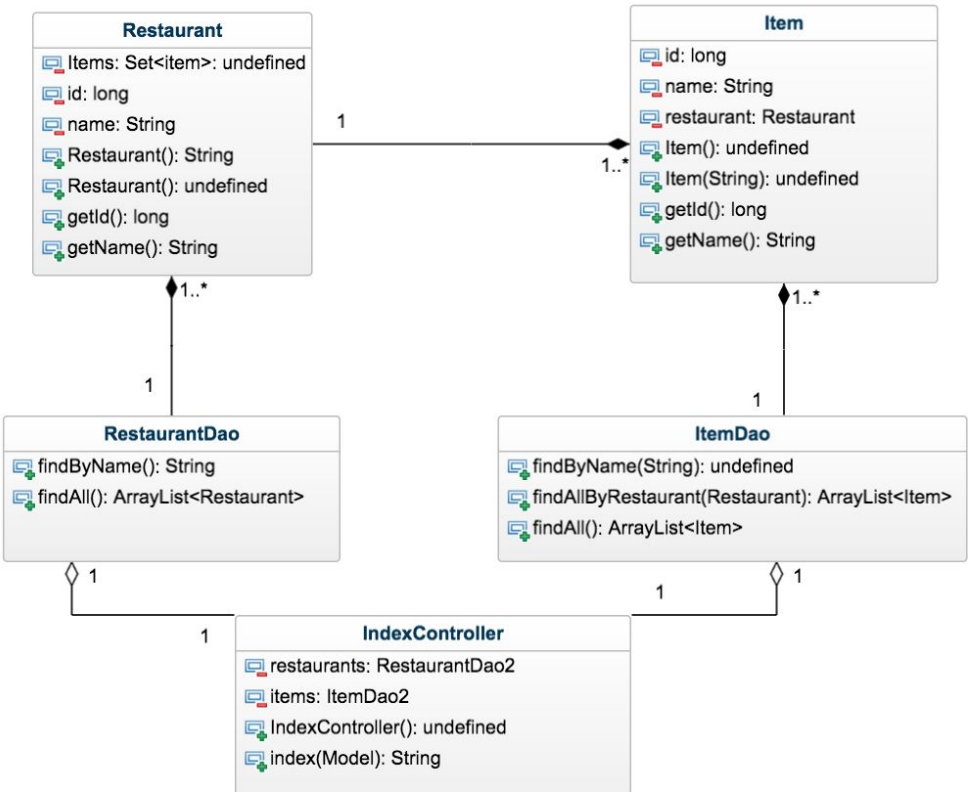


ii. How we made use of this pattern

1. Through the Observer pattern: Provide a real-time respond to business users on orders placed on their restaurants. Registering is not implemented, so this is not the full observer pattern. Registering isn't that useful to our use cases, because we assume all restaurant owners want to be notified of orders pertinent to them, so we essentially imply that all restaurant owners have registered for order events where the order includes an item from their restaurant.

b. Repository

i. Diagram:



ii. How we made use of this pattern

1. Through the Repository pattern: Represent our database as a persistent database with only one database to read, write, and store the content of our business users

5. What we Learned

We've learned that every development project, we need to start creating the system on paper by drafting out outline of the desired system design. This also include creating scenarios defining the purpose of the project and to discover the uniqueness of the project that is different from our competitor. In that sense, we were able to identify which part of the project are critical to fulfill our goal and the goal of our actors and which are non-essential but holds a part in making our project more robust or more comprehensive compared to our competitor. The whole semester working on the project have given us a glimpse on the backbone of developing softwares and application in the real world. In our trial catching up with the dateline, we were able to experiment with various software development tools and study up on multiple programming languages. With each deadline passed, we become more aware of our mistakes in writing java codes that are not up to standard and tried to make them elegant and avoid using if-statements that are redundant in some area of our code. Most importantly, we learned the concept of object-oriented design which all software design have and how one design is differ from the other in terms of speed, efficiency, organization management, memory usage, and creativity.