Final Report

Team

- Seth Perry
- Sudeep Galgali
- Christina Nguyen

Title

The Third Meal

Features Implemented

ID	Requirement	Topic Area	Actor	Priority
BR002	Verify if a customer is genuine	Authentication	Customer	Medium
BR006	Track time to fulfillment	Orders	Restaurant owner / System	Low
UR001	Users should be able to place an order for a specific restaurant	Orders	Customer	High
UR002	Users should be able to see all the current active orders	Orders	Restaurant owner / Customer	High
UR003	Users should be able to add and edit menus for their restaurant	Store	Restaurant owner	High
UR005	Users should be able to declare if the restaurant is open or closed	Store	Restaurant owner	High
UR006	Users should be able to create their profiles	Profile	Customer / Restaurant owner	High
UR007	Users should be able to view a restaurant profile	Profile	Customer	High
UR009	Users should be able to search for a restaurant	Store	Customer	High
NF003	Accounts should be password-protected	Security	N/A	High
NF004	Data access should be well-segregated, allowing for database changes	Maintainabiliity	N/A	Medium
NF005	All transactions should be ACID compatible	Reliability	N/A	High

Features Not Implemented

ID	Requirement	Topic Area	Actor	Priority
BR001	Provide options for delivery or store pickup	Orders	Customer	High
BR003	Provide an option for payment methods	Orders / Payment	Restaurant owner	High
BR004	Track total number of orders	Orders	Restaurant owner / System	Low

ID	Requirement	Topic Area	Actor	Priority
BR005	Track total value of orders	Orders	Restaurant owner / System	High
UR004	Users should be able to cancel an active order	Orders	Customer	Medium
UR006	Users should be able to edit their profiles	Profile	Customer / Restaurant owner	High
UR008	Users should be able to update order status	Orders	Restaurant owner	High
NF001	Order confirmations should be sent in under 30s	Performance	N/A	Medium
NF002	Orders updates (to restaurants) should be accurate to 30s	Performance	N/A	Medium
NF006	UI should confirm order with customers before sending order to restaurant owners	UX	N/A	High

Class Diagrams

For the diagrams, please refer to Appendix A.

There a few major differences between our class diagrams for part 2 and part 3. Because of the sheer size and ordering of the original class diagram, it is very hard to comprehend the relation between the objects. In the final class diagrams we have arranged it based on the functionality. This made our class diagram more understandable and clean.

In the final class diagram we have also made use of multiple design patterns, most importantly Dependency Injection and Strategy design patterns(discussed in more detail below). It reduced the coupling between our classes which meant less lines in the class diagram, reduced the complexity of the project and increased the ease of development.

Since most of the design of the project was performed before the actual development, there were fewer decisions to make as we were writing the code. Class diagram helped in understanding all the different moving parts of the project which greatly reduced the time and effort required during the coding of the project.

Design Patterns in Prototype

We used two prominent design patterns in our final prototype.

- Strategy Design Pattern: We have used this design pattern extensively in the project. We have numerous data item repositories that are represented as interfaces, which are given concrete implementations. With such a setup we have an abstract layer of interfaces which can be swapped out for alternate functionalities without any change to their concrete implementations.

The portion of the class digram that represents the Strategy Design Pattern can be found in **Appendix B**.

- **Dependency Injection**: In our final system we have used Dependency Injection in our front-end pages. We have introduced the IRepositoryProvider, which is responsible for providing the repositories to the page. This is the only strict dependency taken by each page. This helps to decouple the pages from the repositories they require by putting an intermediate object in between the two. Any number of repositories can be consumed by each page, and the page's constructors will not need to be modified to consume a new repository. Furthermore, this allows the possibility for lazy repository creation, where it is only generated when it is needed. In our actual implementation, the RepositoryProvider classes and the Repository Factory classes were managed by the Dependency Injection in the Spring Framework itself. Thus, these classes were not implemented, but the general idea was respected in the code.

The portion of the class diagram that represents Dependency Injection can be found in **Appendix C**.

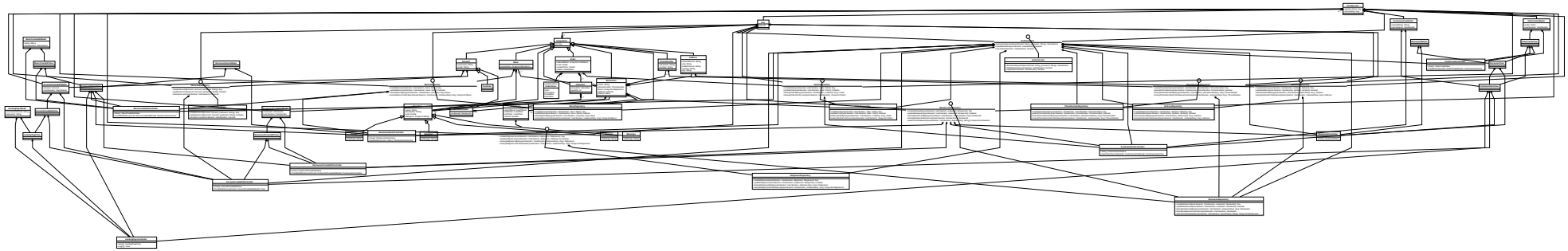
What We Learned

One of the biggest things we learned about design and analysis through this project is how much thought, refinement, and refactoring goes into designing a resilient system. While the notion that computers do not execute procedures in the same was as humans has been reinforced since Intro to CS, the notion that abstract logic is expanded to encompass the entire system is really hammered home in this class. Corrollary to that, it makes it clear just how difficult it is to design a complex system. We felt as if our diagrams and entities in our code were becoming unwieldy, yet we only implemeneted part of the features that we had intended to. Perhaps this could be changed by trying to move parts around and creating different groups / classes to nest things under in a way that is logical to a computer, rather than relying on human notions of what constitutes "similar attributes" in a "class."

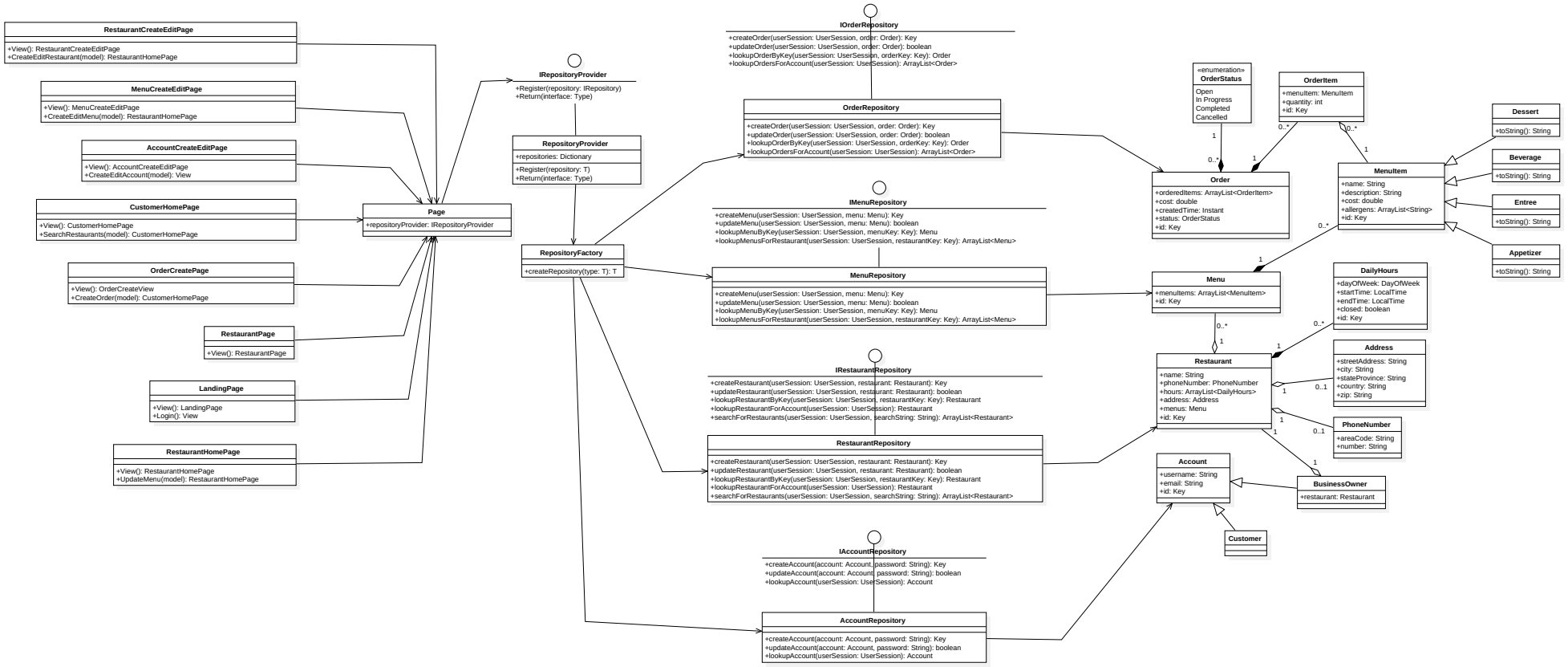
Additionally, we found the diagrams to be really burdensome in some ways, but ultimately they're extremely useful. Translating from a good diagram to code is very easy, and it makes the process very quick. However, it is a bit of a catch-22 while doing the diagrams, because as humans it is hard to imagine the whole system without having a working prototype to play with. As such, creating the diagrams the first time seemed to work in our heads and on paper, but it was hard to verify since we weren't used to diagrams, and we couldn't witness our ideas in action without a prototype yet. This catch-22 really underlines why it's important to constantly refactor throughout the process. Even with refactoring though, the hardest part is ensuring that the user interface is stable and adhereing to open for extension, closed for modification. It would be interesting to explore further how refactoring can affect migrations from one design pattern to another as a project scales or changes scope.

Appendix A

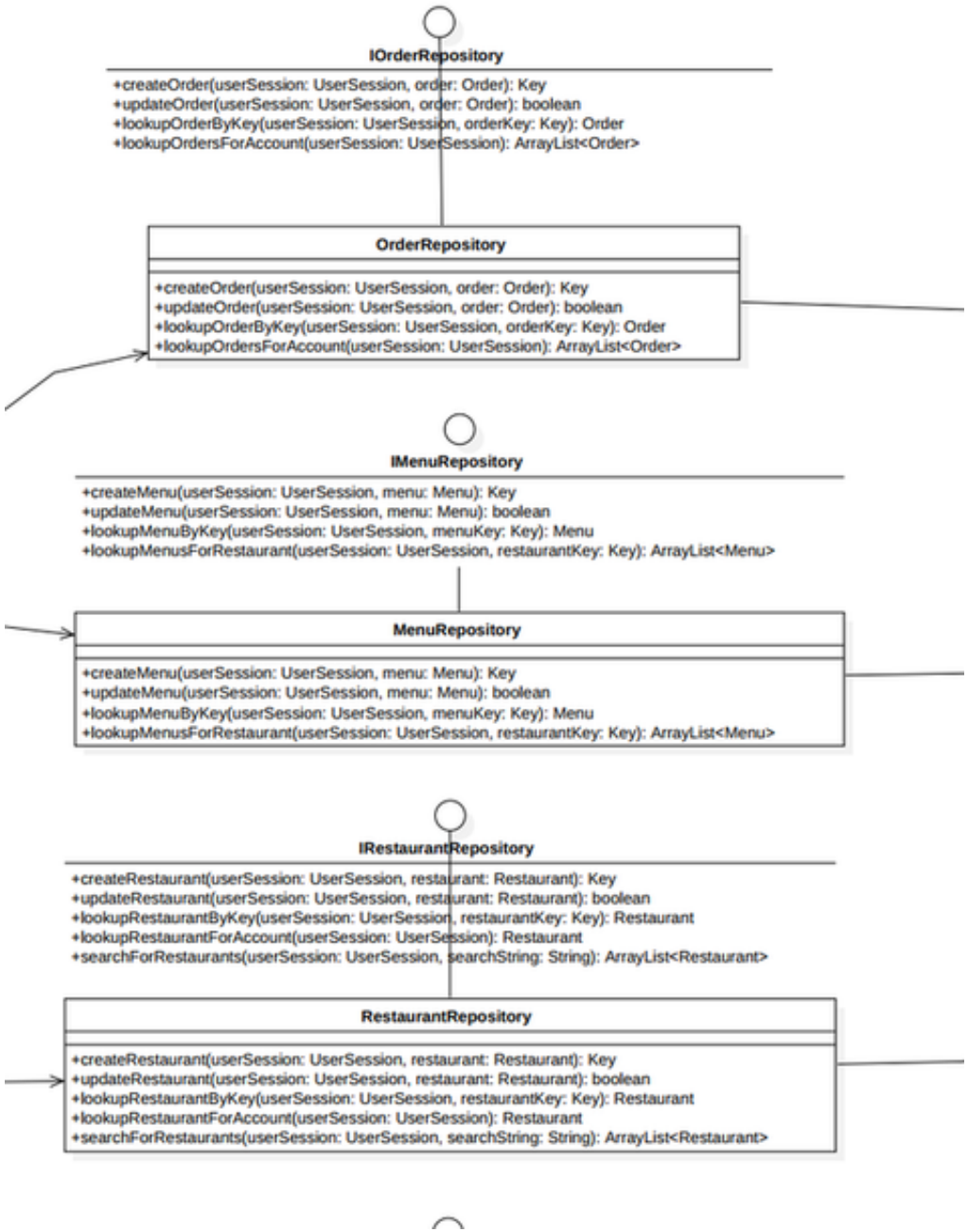
Model::Main



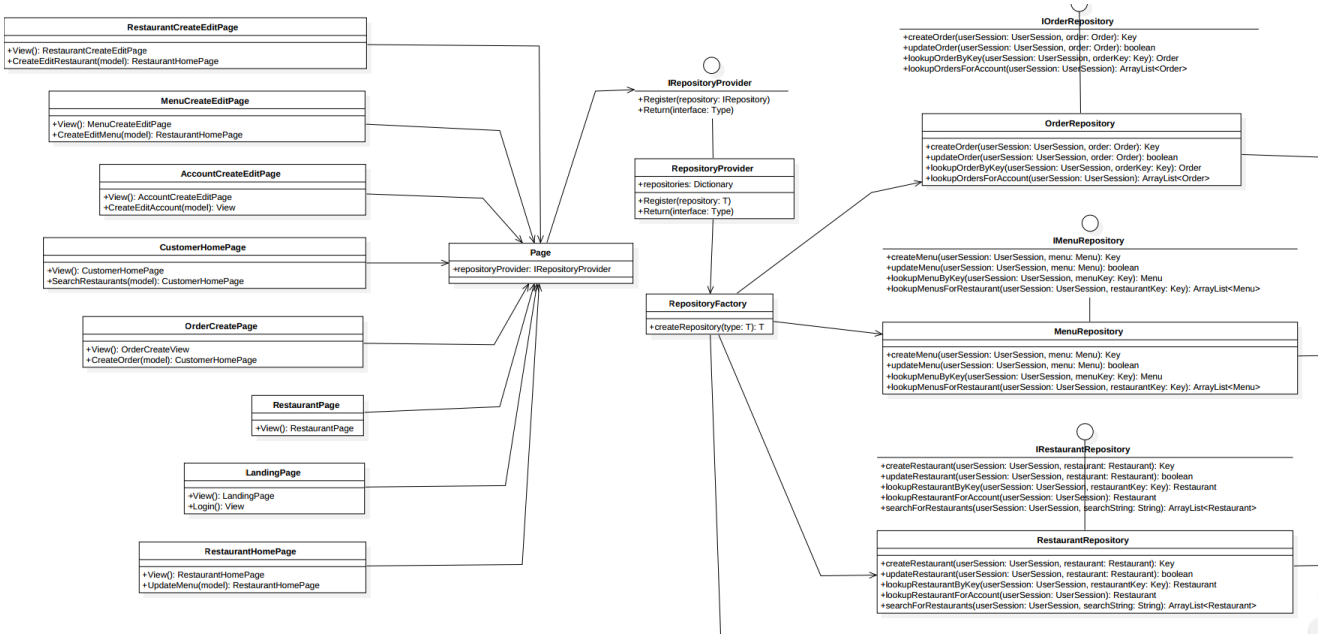
Model::Main


RestaurantCreateEditPage
+View(): RestaurantCreateEditPage
+CreateEditRestaurant(model): RestaurantHomePage
MenuCreateEditPage
+View(): MenuCreateEditPage
+CreateEditMenu(model): RestaurantHomePage
AccountCreateEditPage
+View(): AccountCreateEditPage
+CreateEditAccount(model): View
CustomerHomePage
+View(): CustomerHomePage
+SearchRestaurants(model): CustomerHomePage
OrderCreatePage
+View(): OrderCreateView
+CreateOrder(model): CustomerHomePage
RestaurantPage
+View(): RestaurantPage
LandingPage
+View(): LandingPage
+Login(): View
RestaurantHomePage
+View(): RestaurantHomePage
+UpdateMenu(model): RestaurantHomePage
Page
+repositoryProvider: IRepositoryProvider
IRepositoryProvider
+Register(repository: IRepository)
+Return(interface: Type)
RepositoryProvider
+repositories: Dictionary
+Register(repository: T)
+Return(interface: Type)
RepositoryFactory
+createRepository(type: T): T
IOrderRepository
+createOrder(userSession: UserSession, order: Order): Key
+updateOrder(userSession: UserSession, order: Order): boolean
+lookupOrderByKey(userSession: UserSession, orderKey: Key): Order
+lookupOrdersForAccount(userSession: UserSession): ArrayList<Order>
OrderRepository
+createOrder(userSession: UserSession, order: Order): Key
+updateOrder(userSession: UserSession, order: Order): boolean
+lookupOrderByKey(userSession: UserSession, orderKey: Key): Order
+lookupOrdersForAccount(userSession: UserSession): ArrayList<Order>
IMenuRepository
+createMenu(userSession: UserSession, menu: Menu): Key
+updateMenu(userSession: UserSession, menu: Menu): boolean
+lookupMenuByKey(userSession: UserSession, menuKey: Key): Menu
+lookupMenusForRestaurant(userSession: UserSession, restaurantKey: Key): ArrayList<Menu>
MenuRepository
+createMenu(userSession: UserSession, menu: Menu): Key
+updateMenu(userSession: UserSession, menu: Menu): boolean
+lookupMenuByKey(userSession: UserSession, menuKey: Key): Menu
+lookupMenusForRestaurant(userSession: UserSession, restaurantKey: Key): ArrayList<Menu>
IRestaurantRepository
+createRestaurant(userSession: UserSession, restaurant: Restaurant): Key
+updateRestaurant(userSession: UserSession, restaurant: Restaurant): boolean
+lookupRestaurantByKey(userSession: UserSession, restaurantKey: Key): Restaurant
+lookupRestaurantForAccount(userSession: UserSession): Restaurant
+searchForRestaurants(userSession: UserSession, searchString: String): ArrayList<Restaurant>
RestaurantRepository
+createRestaurant(userSession: UserSession, restaurant: Restaurant): Key
+updateRestaurant(userSession: UserSession, restaurant: Restaurant): boolean
+lookupRestaurantByKey(userSession: UserSession, restaurantKey: Key): Restaurant
+lookupRestaurantForAccount(userSession: UserSession): Restaurant
+searchForRestaurants(userSession: UserSession, searchString: String): ArrayList<Restaurant>
IAccountRepository
+createAccount(account: Account, password: String): Key
+updateAccount(account: Account, password: String): boolean
+lookupAccount(userSession: UserSession): Account
AccountRepository
+createAccount(account: Account, password: String): Key
+updateAccount(account: Account, password: String): boolean
+lookupAccount(userSession: UserSession): Account
«enumeration»
OrderStatus
Open
In Progress
Completed
Cancelled
OrderItem
+menuItem: MenuItem
+quantity: int
+id: Key
Order
+orderedItems: ArrayList<OrderItem>
+cost: double
+createdTime: Instant
+status: OrderStatus
+id: Key
MenuItem
+name: String
+description: String
+cost: double
+allergens: ArrayList<String>
+id: Key
Dessert
+toString(): String
Beverage
+toString(): String
Entree
+toString(): String
Appetizer
+toString(): String
Menu
+menuItems: ArrayList<MenuItem>
+id: Key
DailyHours
+dayOfWeek: DayOfWeek
+startTime: LocalTime
+endTime: LocalTime
+closed: boolean
+id: Key
Restaurant
+name: String
+phoneNumber: PhoneNumber
+hours: ArrayList<DailyHours>
+address: Address
+menus: Menu
+id: Key
Address
+streetAddress: String
+city: String
+stateProvince: String
+country: String
+zip: String
PhoneNumber
+areaCode: String
+number: String
Account
+username: String
+email: String
+id: Key
BusinessOwner
+restaurant: Restaurant
Customer
1
0..*
0..1

Appendix B


IOrderRepository
+createOrder(userSession: UserSession, order: Order): Key
+updateOrder(userSession: UserSession, order: Order): boolean
+lookupOrderByKey(userSession: UserSession, orderKey: Key): Order
+lookupOrdersForAccount(userSession: UserSession): ArrayList<Order>
OrderRepository
+createOrder(userSession: UserSession, order: Order): Key
+updateOrder(userSession: UserSession, order: Order): boolean
+lookupOrderByKey(userSession: UserSession, orderKey: Key): Order
+lookupOrdersForAccount(userSession: UserSession): ArrayList<Order>
IMenuRepository
+createMenu(userSession: UserSession, menu: Menu): Key
+updateMenu(userSession: UserSession, menu: Menu): boolean
+lookupMenuByKey(userSession: UserSession, menuKey: Key): Menu
+lookupMenusForRestaurant(userSession: UserSession, restaurantKey: Key): ArrayList<Menu>
MenuRepository
+createMenu(userSession: UserSession, menu: Menu): Key
+updateMenu(userSession: UserSession, menu: Menu): boolean
+lookupMenuByKey(userSession: UserSession, menuKey: Key): Menu
+lookupMenusForRestaurant(userSession: UserSession, restaurantKey: Key): ArrayList<Menu>
IRestaurantRepository
+createRestaurant(userSession: UserSession, restaurant: Restaurant): Key
+updateRestaurant(userSession: UserSession, restaurant: Restaurant): boolean
+lookupRestaurantByKey(userSession: UserSession, restaurantKey: Key): Restaurant
+lookupRestaurantForAccount(userSession: UserSession): Restaurant
+searchForRestaurants(userSession: UserSession, searchString: String): ArrayList<Restaurant>
RestaurantRepository
+createRestaurant(userSession: UserSession, restaurant: Restaurant): Key
+updateRestaurant(userSession: UserSession, restaurant: Restaurant): boolean
+lookupRestaurantByKey(userSession: UserSession, restaurantKey: Key): Restaurant
+lookupRestaurantForAccount(userSession: UserSession): Restaurant
+searchForRestaurants(userSession: UserSession, searchString: String): ArrayList<Restaurant>

Appendix C


RestaurantCreateEditPage
+View(): RestaurantCreateEditPage
+CreateEditRestaurant(model): RestaurantHomePage
MenuCreateEditPage
+View(): MenuCreateEditPage
+CreateEditMenu(model): RestaurantHomePage
AccountCreateEditPage
+View(): AccountCreateEditPage
+CreateEditAccount(model): View
CustomerHomePage
+View(): CustomerHomePage
+SearchRestaurants(model): CustomerHomePage
OrderCreatePage
+View(): OrderCreateView
+CreateOrder(model): CustomerHomePage
RestaurantPage
+View(): RestaurantPage
LandingPage
+View(): LandingPage
+Login(): View
RestaurantHomePage
+View(): RestaurantHomePage
+UpdateMenu(model): RestaurantHomePage
Page
+repositoryProvider: IRepositoryProvider
IRepositoryProvider
+Register(repository: IRepository)
+Return(interface: Type)
RepositoryProvider
+repositories: Dictionary
+Register(repository: T)
+Return(interface: Type)
RepositoryFactory
+createRepository(type: T): T
IOrderRepository
+createOrder(userSession: UserSession, order: Order): Key
+updateOrder(userSession: UserSession, order: Order): boolean
+lookupOrderByKey(userSession: UserSession, orderKey: Key): Order
+lookupOrdersForAccount(userSession: UserSession): ArrayList<Order>
OrderRepository
+createOrder(userSession: UserSession, order: Order): Key
+updateOrder(userSession: UserSession, order: Order): boolean
+lookupOrderByKey(userSession: UserSession, orderKey: Key): Order
+lookupOrdersForAccount(userSession: UserSession): ArrayList<Order>
IMenuRepository
+createMenu(userSession: UserSession, menu: Menu): Key
+updateMenu(userSession: UserSession, menu: Menu): boolean
+lookupMenuByKey(userSession: UserSession, menuKey: Key): Menu
+lookupMenusForRestaurant(userSession: UserSession, restaurantKey: Key): ArrayList<Menu>
MenuRepository
+createMenu(userSession: UserSession, menu: Menu): Key
+updateMenu(userSession: UserSession, menu: Menu): boolean
+lookupMenuByKey(userSession: UserSession, menuKey: Key): Menu
+lookupMenusForRestaurant(userSession: UserSession, restaurantKey: Key): ArrayList<Menu>
IRestaurantRepository
+createRestaurant(userSession: UserSession, restaurant: Restaurant): Key
+updateRestaurant(userSession: UserSession, restaurant: Restaurant): boolean
+lookupRestaurantByKey(userSession: UserSession, restaurantKey: Key): Restaurant
+lookupRestaurantForAccount(userSession: UserSession): Restaurant
+searchForRestaurants(userSession: UserSession, searchString: String): ArrayList<Restaurant>
RestaurantRepository
+createRestaurant(userSession: UserSession, restaurant: Restaurant): Key
+updateRestaurant(userSession: UserSession, restaurant: Restaurant): boolean
+lookupRestaurantByKey(userSession: UserSession, restaurantKey: Key): Restaurant
+lookupRestaurantForAccount(userSession: UserSession): Restaurant
+searchForRestaurants(userSession: UserSession, searchString: String): ArrayList<Restaurant>