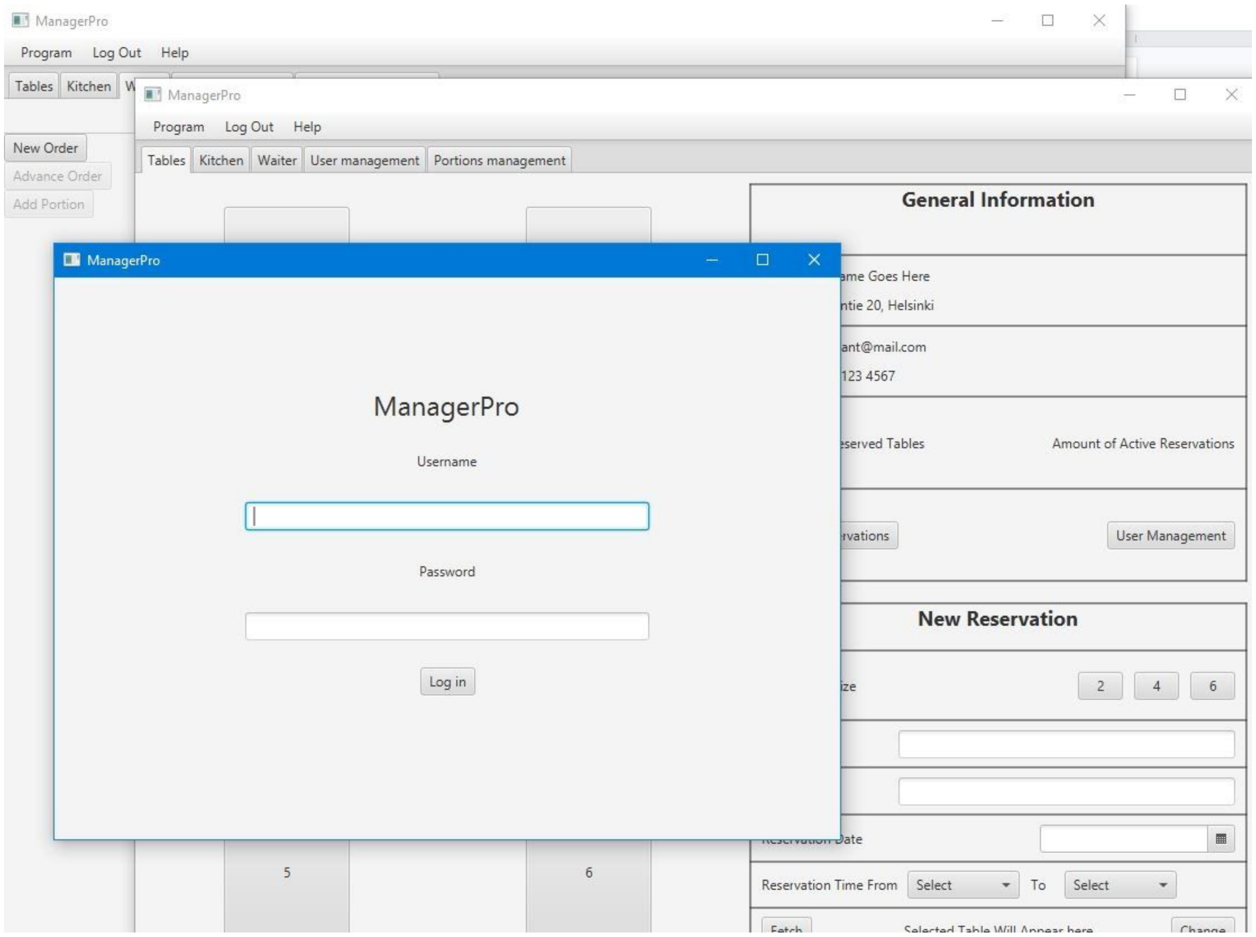
Interim Report - ManagerPro

Ohjelmistotuotanto 1 - Group 7



Kimmo Karjalainen
Amal Kayed
Pettri Liukkonen
Juho Roiha

11.3.2020

INTRODUCTION

ManagerPro is a restaurant managing software. It can be used to keep track of reservations, orders and portions of the menu. The weighted goal of the project is to keep track of the reservations, with other features being secondary.

The User must log in to the software with their username and password, after which the user has access to the features of the software. The software is designed to be used simultaneously with multiple users, so a synchronization feature is implemented to keep the multiple clients in sync with each other. The orders the waiter takes from the customer are immediately updated to the kitchen, so the kitchen staff can start cooking the portions.

The software is programmed in Java11 and it uses JavaFX for the user interface and Hibernate ORM to communicate with a relational database management system (RDBMS), which was chosen to be MariaDB.

The project uses Apache Maven project management and comprehension tool to build, report, document and package the project. GitLab is used for version control and a Jenkins server on the Metropolia EduCloud virtual machine is polling the git repository for continuous integration.

IMPLEMENTED FEATURES SO FAR

- Person working till/phone can take reservations from customers
- When making reservations, the tables in the restaurant can be filtered by amount of seats
- When making a reservation, invalid user inputs are detected to prevent invalid reservations (table already booked, date of reservation in the past, start time later than end time etc.)
- Waiter can take orders from the customers
- Orders the waiter takes go to the Kitchen automatically
- Orders that have all it’s portions done will disappear from the kitchen to prevent UI clutter
- Waiter sees when orders are ready to serve

- Waiter can add additional information for a portion that has been ordered from the menu(for allergies/diets/etc.)
- Kitchen sees this additional information
- Login/Logout feature with password salting and hashing
- Users can be created to the database through the software

FEATURES NOT YET IMPLEMENTED

- No analytics of table usage can be produced
- Users cannot be deleted through user interface
- Users cannot be modified through user interface
- Table map of the restaurant cannot be attach to the UI
- Status of the tables in the restaurant is not yet visible from the UI (the table would change appearance in the map if it was booked or not)
- A mobile tablet application was discussed, but not implemented or added to the roadmap. The tablet would be ideal tool for a waiter taking orders
- Warn users of caps lock status when logging in; Java has problems detecting the status of a locking key

ARCHITECTURE DESCRIPTION

The project follows the MVC design pattern. The data the application handles is stored in the model classes, which is accessed by the controller. The controller makes changes to the model and updates the view accordingly, based on the input it receives from the user.

The view consists of a MainApp -class and several JavaFX .fxml -files. The fxml -files hold majority of the user interface elements. The loginUI.fxml is loaded first and after a successful login, a mainUI.fxml file is loaded which has nested .fxml files to allow multiple controllers to be used, depending on the tab the user is on.

The FXML -files have controllers defined in them and JavaFX instantiates the controllers on fxml document load. The first controller to be instantiated is the LogInController, which has the logic for handling user login. It also creates a Hibernate SessionFactory and a Validation factory. The next controller is the main controller -class (Controller.java), which acts as a parent controller for all of the other controller classes.

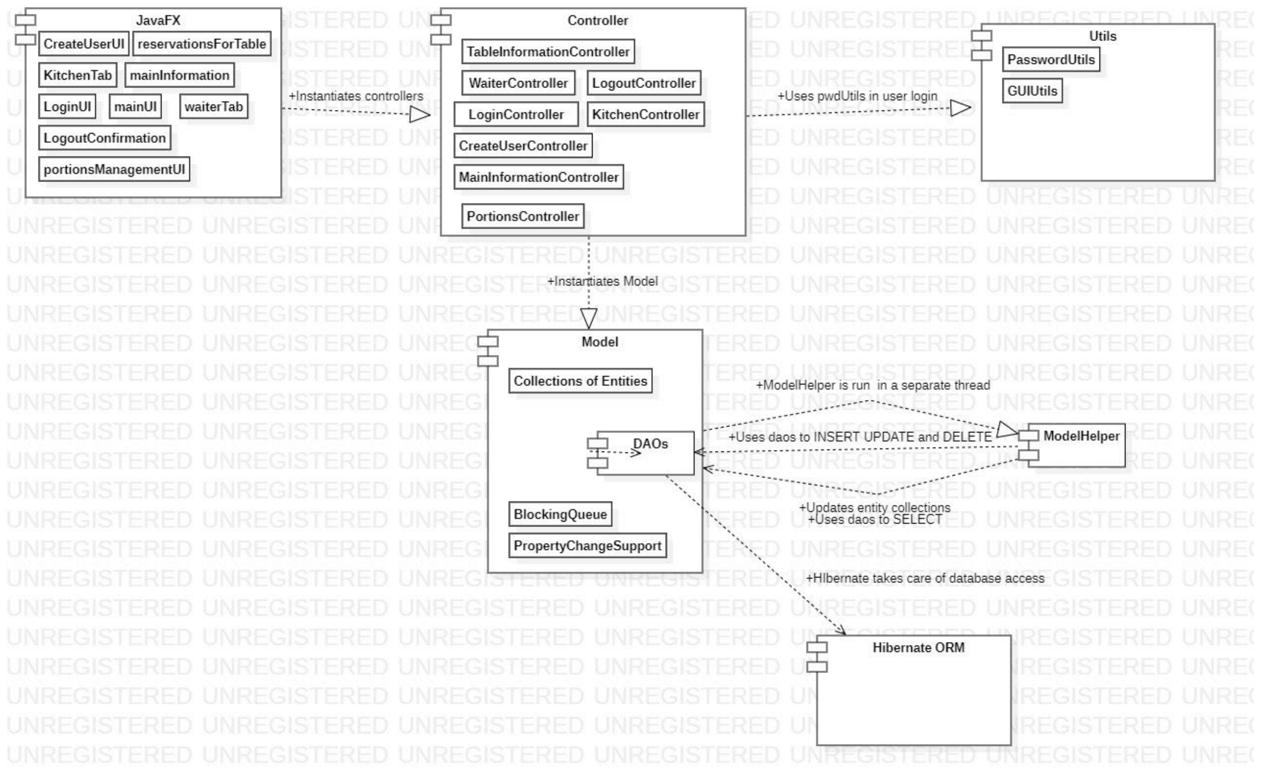
It holds an instance of the Model -class in a static property, allowing access from different parts of the application. The nested structure of the fxml allows for multiple controllers on different parts of the user interface, preventing a massive controller god-object that takes care of everything. Many of the fxml -files have a dedicated controller handling the events of a certain portion of the UI.

The Model part of the application consists of DAO -interfaces, DAOs and Hibernate Entity -classes. The Hibernate ORM is used to create objects based on the rows in the database, allowing for an efficient way of data access and storage. The Hibernate mappings are handled with JPA Annotations that are in the Entity -classes. To access the database, Data Access Objects are created. The DAOs the CRUD methods and possible exceptions that may be thrown.

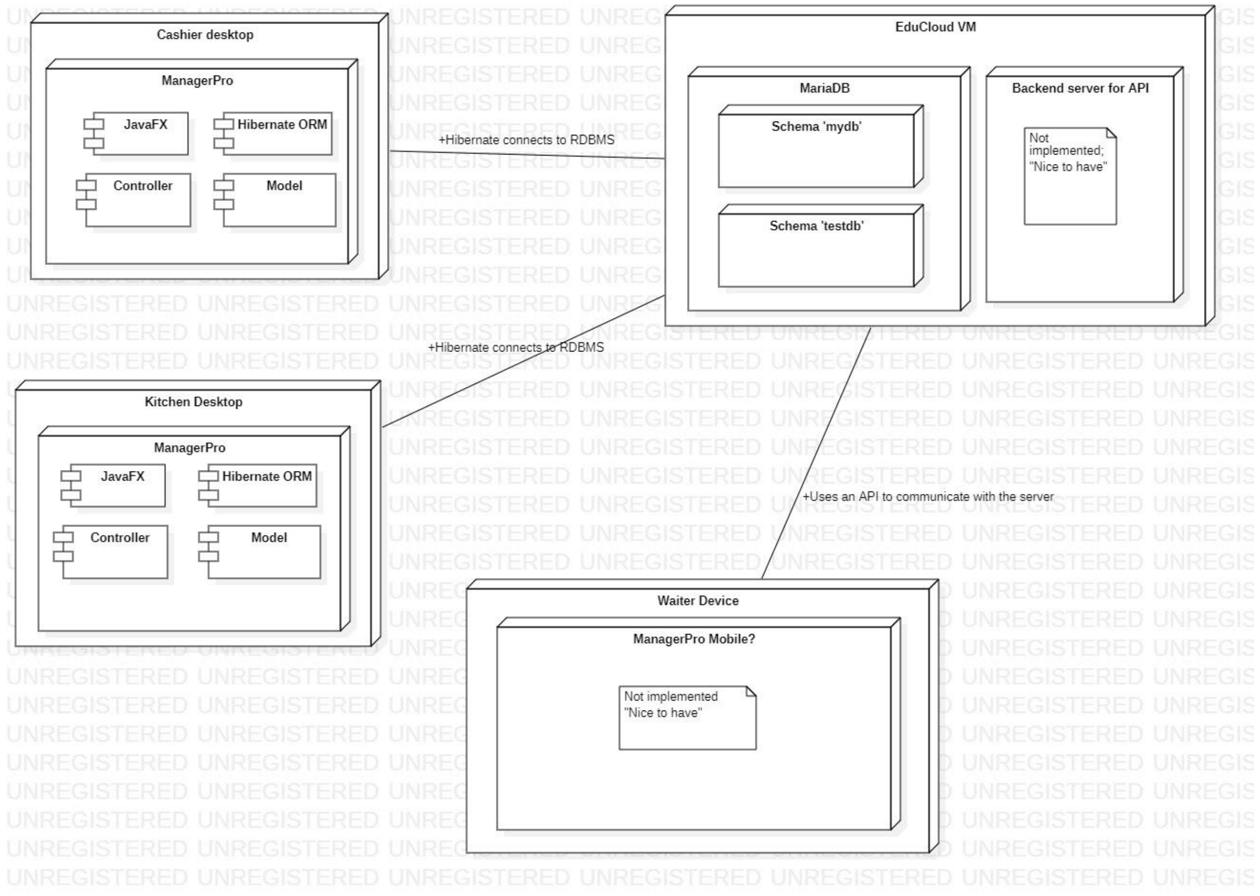
The multi-user nature of the application calls for a way to keep all the clients in sync. To address this issue, several ideas were thrown around and we decided to use polling to keep the data in sync. An intermediary datastore - Model.java - was created to hold data between the controllers and the database. The Model -class holds collections of the Hibernate Entities, that are periodically compared to the database and updated if necessary. A ModelHelper -class that extends TimerTask was created to fetch the database information to the client. The data the ModelHelper fetched is compared against the intermediary data and if differences are noticed, the controllers are notified of the change with PropertyChangeListeners. This allows the controllers to update the UI when necessary and only when necessary.

The ModelHelper is also run whenever changes are made to the database by the user. When the CRUD methods of the Model -class are called, a new thread is spawned to fetch the changes made to the database back to the client. This ‘round trip’ approach is a bit ineffective, but ensures no data is lost even if race conditions occur. The Model -class also features locking to prevent concurrent reads and writes.

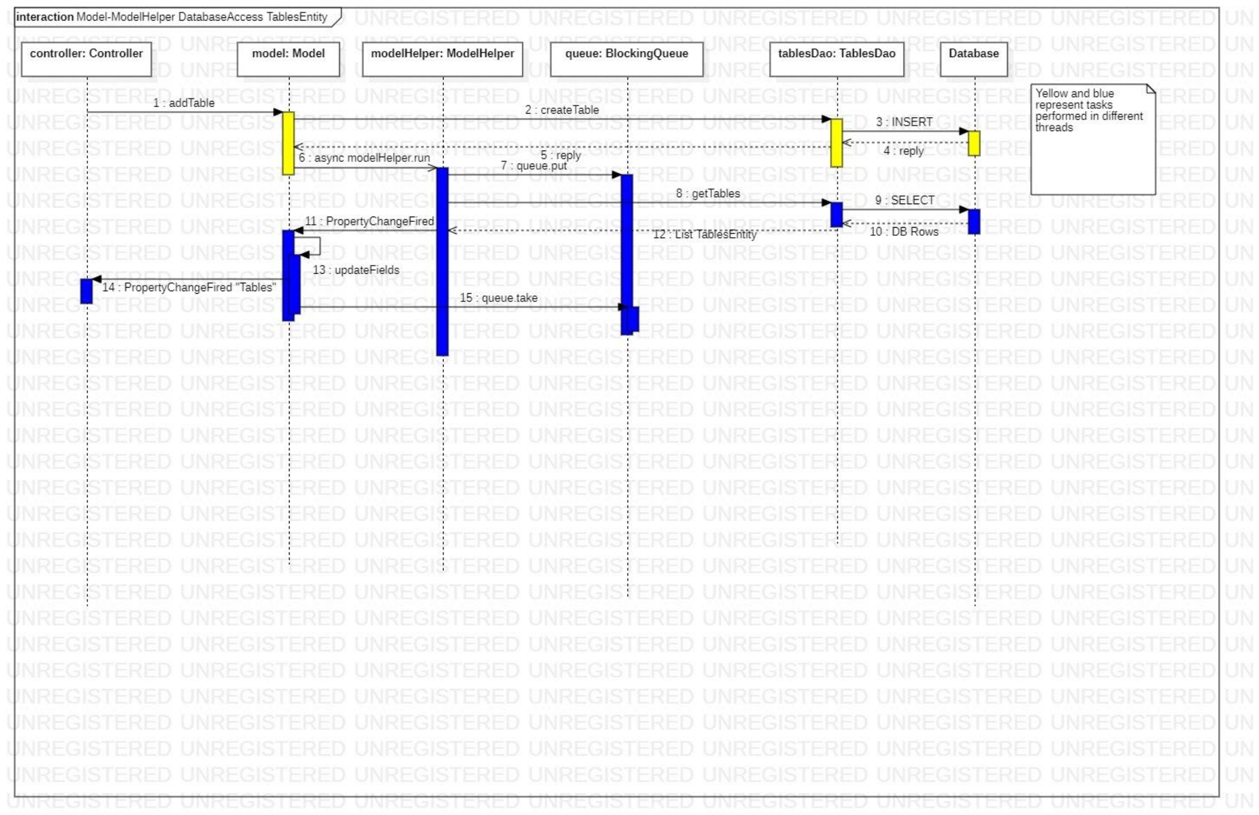
APPENDICES



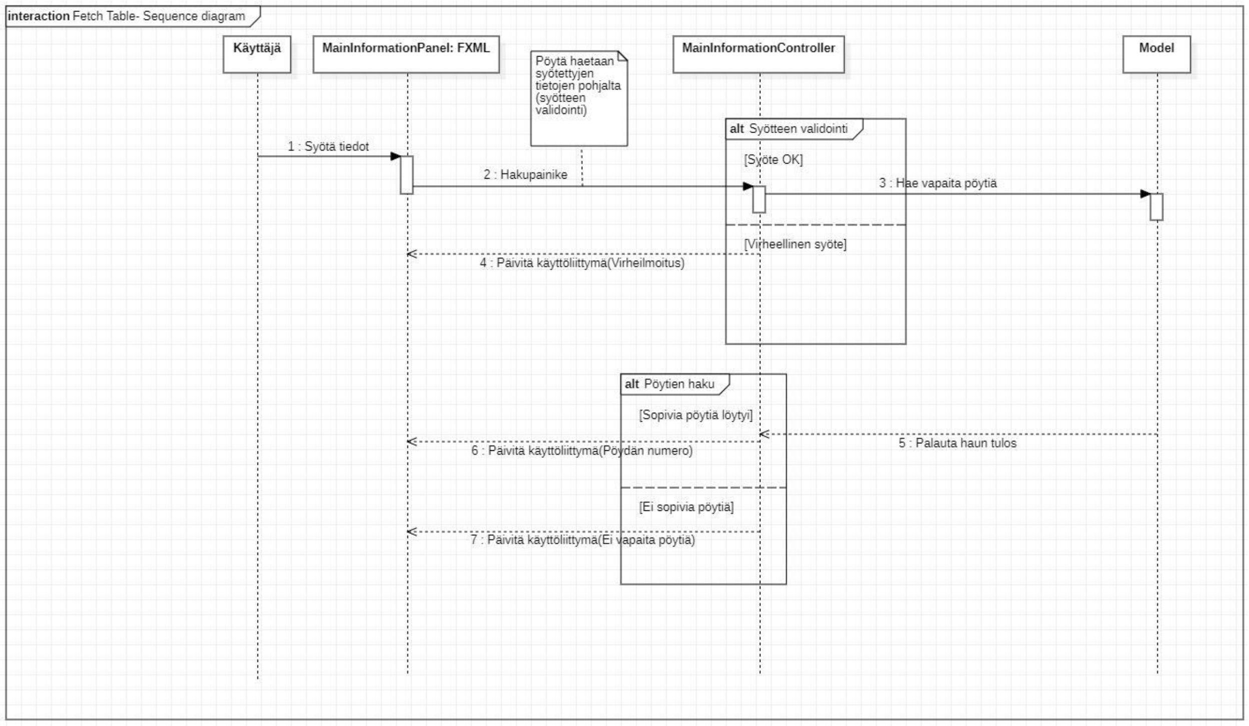
Component diagram of the software



Deployment diagram of the software



Sequence diagram illustrating the Model - ModelHelper communications


interaction Fetch Table- Sequence diagram
Käyttäjä
MainInformationPanel: FXML
Pöytä haetaan syötettyjen tietojen pohjalta (syötteen validointi)
MainInformationController
Model
alt Syötteen validointi
[Syöte OK]
1 : Syötä tiedot
2 : Hakupainike
3 : Hae vapaita pöytiä
[Virheellinen syöte]
4 : Päivitä käyttöliittymä(Virheilmoitus)
alt Pöytien haku
[Sopivia pöytiä löytyi]
5 : Palauta haun tulos
6 : Päivitä käyttöliittymä(Pöydän numero)
[Ei sopivia pöytiä]
7 : Päivitä käyttöliittymä(Ei vapaita pöytiä)

Sequence diagram illustrating the fetching of a table from the database

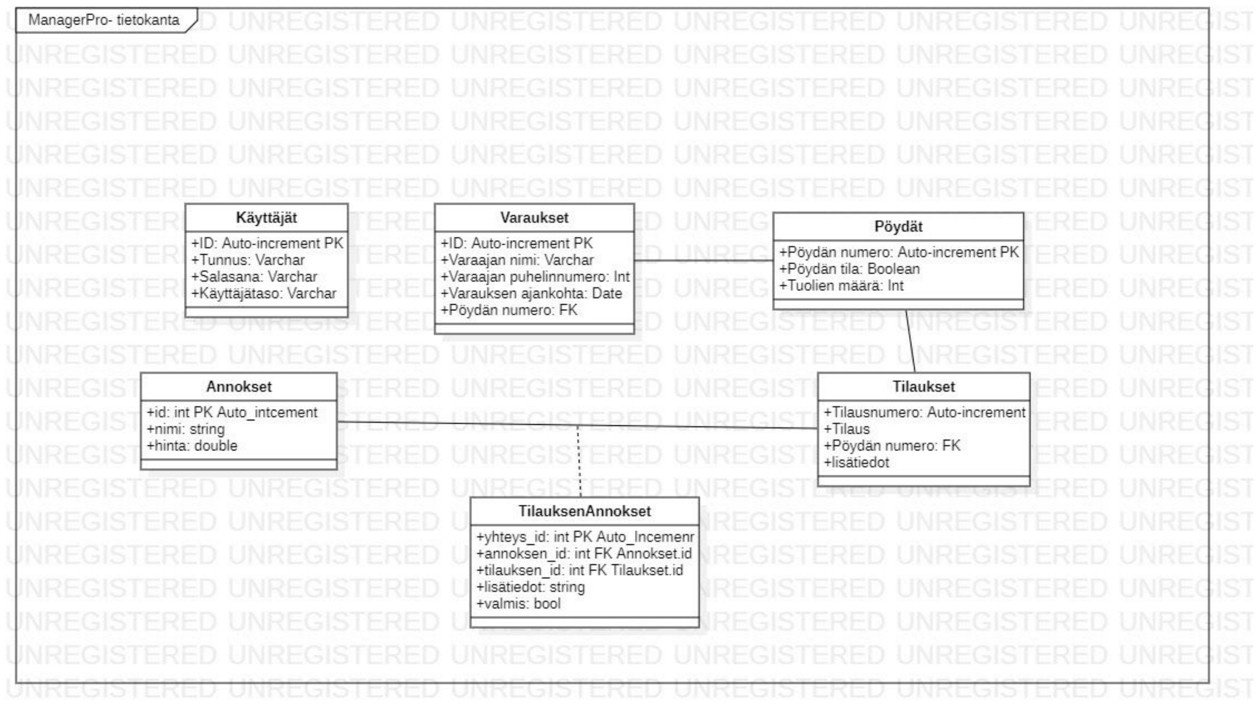
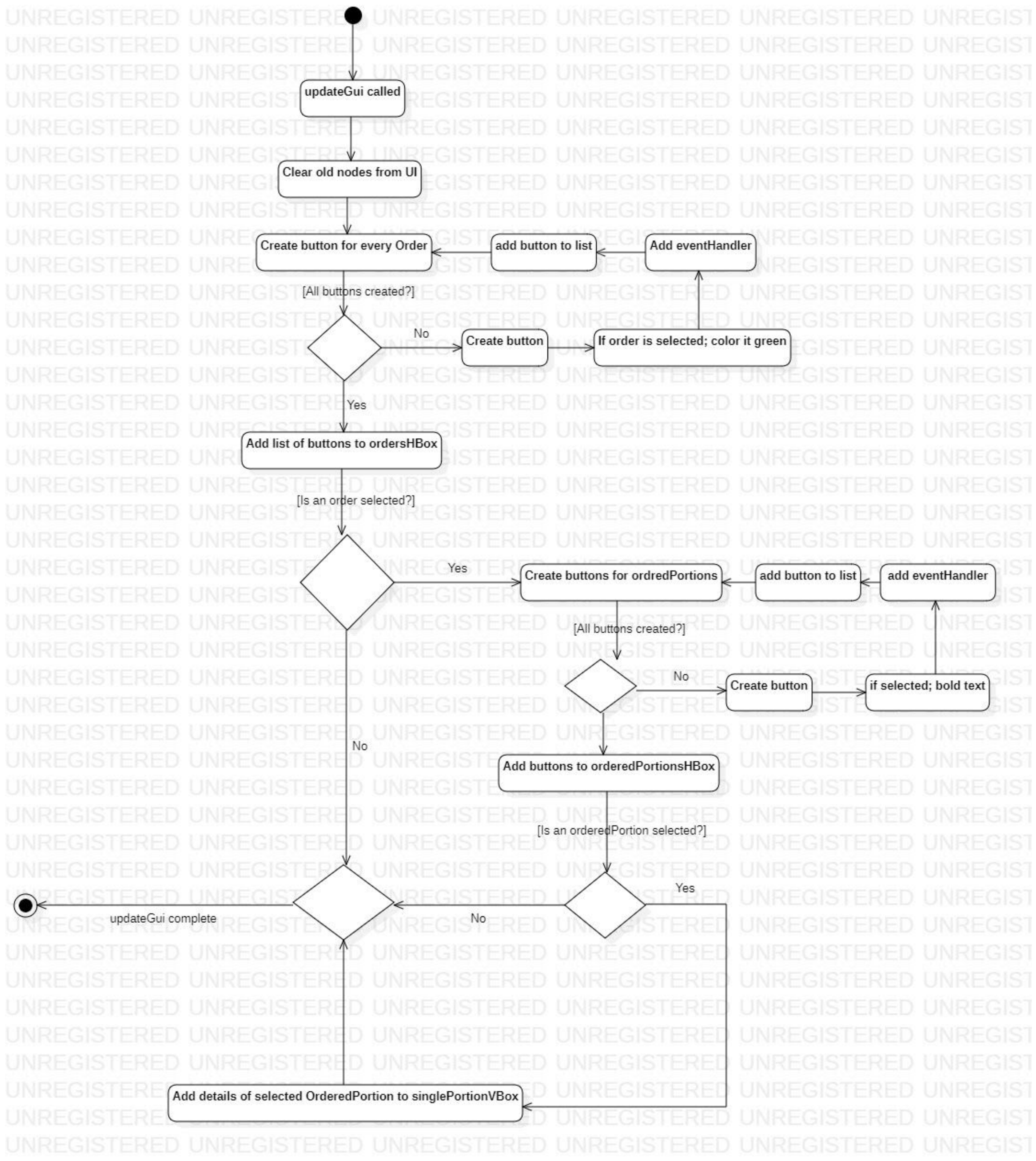
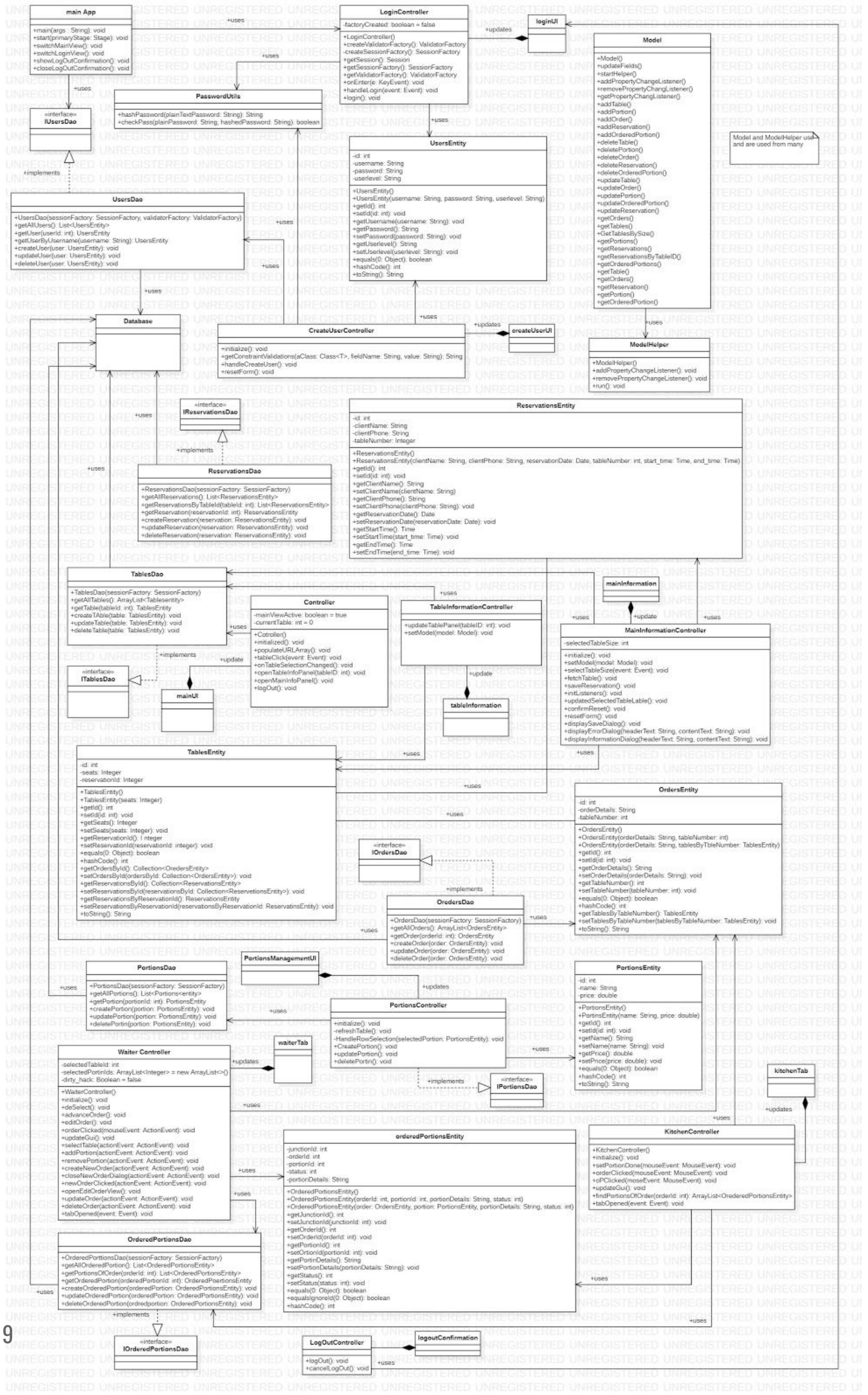


Illustration of the RDBMS



Activity diagram of the KitchenView updateGui method, illustrating the dynamic creation of UI elements based on database contents.


main App
+main(args : String): void
+start(primaryStage: Stage): void
+switchMainView(): void
+switchLoginView(): void
+showLogOutConfirmation(): void
+closeLogOutConfirmation(): void
LoginController
-factoryCreated: boolean = false
+LoginController()
+createValidatorFactory(): ValidatorFactory
-createSessionFactory(): SessionFactory
+getSession(): Session
+getSessionFactory(): SessionFactory
+getValidatorFactory(): ValidatorFactory
+onEnter(e: KeyEvent): void
+handleLogin(event: Event): void
+login(): void
loginUI
Model
+Model()
+updateFields()
+startHelper()
+addPropertyChangeListener()
+removePropertyChangListener()
+getPropertyChangListener()
+addTable()
+addPortion()
+addOrder()
+addReservation()
+addOrderedPortion()
+deleteTable()
+deletePortion()
+deleteOrder()
+deleteReservation()
+deleteOrderedPortion()
+updateTable()
+updateOrder()
+updatePortion()
+updateOrderedPortion()
+updateReservation()
+getOrders()
+getTables()
+GetTablesBySize()
+getPortions()
+getReservations()
+getReservationsByTableID()
+getOrderedPortions()
+getTable()
+getOrders()
+getReservation()
+getPortion()
+getOrderedPortion()
Model and ModelHelper used and are used from many
PasswordUtils
+hashPassword(plainTextPassword: String): String
+checkPass(plainPassword: String, hashedPassword: String): boolean
«interface»
IUsersDao
UsersEntity
-id: int
-username: String
-password: String
-userlevel: String
+UsersEntity()
+UsersEntity(username: String, password: String, userlevel: String)
+getId(): int
+setId(id: int): void
+getUsername(): String
+setUsername(username: String): void
+getPassword(): String
+setPassword(password: String): void
+getUserlevel(): String
+setUserlevel(userlevel: String): void
+equals(0: Object): boolean
+hashCode(): int
+toString(): String
UsersDao
+UsersDao(sessionFactory: SessionFactory, validatorFactory: ValidatorFactory)
+getAllUsers(): List<UsersEntity>
+getUser(userId: int): UsersEntity
+getUserByUsername(username: String): UsersEntity
+createUser(user: UsersEntity): void
+updateUser(user: UsersEntity): void
+deleteUser(user: UsersEntity): void
Database
CreateUserController
+initialize(): void
+getConstraintValidations(aClass: Class<T>, fieldName: String, value: String): String
+handleCreateUser(): void
+resetForm(): void
createUserUI
ModelHelper
+ModelHelper()
+addPropertyChangeListener(): void
+removePropertyChangeListener(): void
+run(): void
«interface»
IReservationsDao
ReservationsEntity
-id: int
-clientName: String
-clientPhone: String
-tableNumber: Integer
+ReservationsEntity()
+ReservationsEntity(clientName: String, clientPhone: String, reservationDate: Date, tableNumber: int, start_time: Time, end_time: Time)
+getId(): int
+setId(id: int): void
+getClientName(): String
+setClientName(clientName: String)
+getClientPhone(): String
+setClientPhone(clientPhone: String): void
+getReservationDate(): Date
+setReservationDate(reservationDate: Date): void
+getStartTime(): Time
+setStartTime(start_time: Time): void
+getEndTime(): Time
+setEndTime(end_time: Time): void
ReservationsDao
+ReservationsDao(sessionFactory: SessionFactory)
+getAllReservations(): List<ReservationsEntity>
+getReservationsByTableId(tableId: int): List<ReservationsEntity>
+getReservation(reservationId: int): ReservationsEntity
+createReservation(reservation: ReservationsEntity): void
+updateReservation(reservation: ReservationsEntity): void
+deleteReservation(reservation: ReservationsEntity): void
TablesDao
+TablesDao(sessionFactory: SessionFactory)
+getAllTables(): ArrayList<Tablesentity>
+getTable(tableId: int): TablesEntity
+createTAble(table: TablesEntity): void
+updateTable(table: TablesEntity): void
+deleteTable(table: TablesEntity): void
Controller
-mainViewActive: boolean = true
-currentTable: int = 0
+Cotroller()
+initialize(): void
+populateURLArray(): void
+tableClick(event: Event): void
+onTableSelectionChanged(): void
+openTableInfoPanel(tableID: int): void
+openMainInfoPanel(): void
+logOut(): void
TableInformationController
+updateTablePanel(tableID: int): void
+setModel(model: Model): void
mainInformation
MainInformationController
-selectedTableSize: int
+initialize(): void
+setModel(model: Model): void
+selectTableSize(event: Event): void
+fetchTable(): void
+saveReservation(): void
+initListeners(): void
+updatedSelectedTableLable(): void
+confirmReset(): void
+resetForm(): void
+displaySaveDialog(): void
+displayErrorDialog(headerText: String, contentText: String): void
+displayInformationDialog(headerText: String, contentText: String): void
«interface»
ITablesDao
mainUI
tableInformation
TablesEntity
-id: int
-seats: Integer
-reservationId: Integer
+TablesEntity()
+TablesEntity(seats: Integer)
+getId(): int
+setId(id: int): void
+getSeats(): Integer
+setSeats(seats: Integer): void
+getReservationId(): Integer
+setReservationId(reservationId: integer): void
+equals(0: Object): boolean
+hashCode(): int
+getOrdersById(): Collection<OredersEntity>
+setOrdersById(ordersById: Collection<OrdersEntity>): void
+getReservationsById(): Collection<ReservationsEntity>
+setReservationsById(reservationsById: Collection<ReservetionsEntity>): void
+getReservationsByReservationId(): ReservationsEntity
+setReservationsByReservationId(reservationsByReservationId: ReservatinsEntity): void
+toString(): String
OrdersEntity
-id: int
-orderDetails: String
-tableNumber: int
+OrdersEntity()
+OrderEntity(orderDetails: String, tableNumber: int)
+OrdersEntity(orderDetails: String, tablesByTbleNumber: TablesEntity)
+getId(): int
+setId(id: int): void
+getOrderDetails(): String
+setOrderDetails(orderDetails: String): void
+getTableNumber(): int
+setTableNumber(tableNumber: int): void
+equals(0: Object): boolean
+hashCode(): int
+getTablesByTableNumber(): TablesEntity
+setTablesByTableNumber(tablesByTableNumber: TablesEntity): void
+toString(): String
«interface»
IOrdersDao
OredersDao
+OrdersDao(sessionFactory: SessionFactory)
+getAllOrders(): ArrayList<OrdersEntity>
+getOrder(orderId: int): OrdersEntity
+createOrder(order: OrdersEntity): void
+updateOrder(order: OrdersEntity): void
+deleteOrder(order: OrdersEntity): void
PortionsManagementUI
PortionsDao
+PortionsDao(sessionFactory: SessionFactory)
+getAllPortions(): List<Portions<entity>
+getPortion(portionId: int): PortionsEntity
+createPortion(portion: PortionsEntity): void
+updatePortion(portion: PortionsEntity): void
+deletePortion(portion: PortionsEntity): void
PortionsEntity
-id: int
-name: String
-price: double
+PortionsEntity()
+PortinsEntity(name: String, price: double)
+getId(): int
+setId(id: int): void
+getName(): String
+setName(name: String): void
+getPrice(): double
+setPrice(price: double): void
+equals(0: Object): boolean
+hashCode(): int
+toString(): String
PortionsController
+initialize(): void
-refreshTable(): void
-HandleRowSelection(selectedPortion: PortionsEntity): void
+CreatePortion(): void
+updatePortion(): void
+deletePortin(): void
waiterTab
Waiter Controller
-selectedTableId: int
-selectedPortinIds: ArrayList<Integer> = new ArrayList<>()
-dirty_hack: Boolean = false
+WaiterController()
+initialize(): void
+deSelect(): void
+advanceOrder(): void
+editOrder(): void
+orderClicked(mouseEvent: ActionEvent): void
+updateGui(): void
+selectTable(actionEvent: ActionEvent): void
+addPortion(actionEvent: ActionEvent): void
+removePortion(actionEvent: ActionEvent): void
+createNewOrder(actionEvent: ActionEvent): void
+closeNewOrderDialog(actionEvent: ActionEvent): void
+newOrderClicked(actionEvent: ActionEvent): void
+openEditOrderView(): void
+updateOrder(actionEvent: ActionEvent): void
+deleteOrder(actionEvent: ActionEvent): void
+tabOpened(event: Event): void
«interface»
IPortionsDao
kitchenTab
orderedPortionsEntity
-junctionId: int
-orderId: int
-portionId: int
-status: int
-portionDetails: String
+OrderedPortionsEntity()
+OrderedPortionsEntity(orderId: int, portionId: int, portionDetails: String, status: int)
+OrderedPortionsEntity(order: OrdersEntity, portion: PortionsEntity, portionDetails: String, status: int)
+getJunctionId(): int
+setJunctionId(junctionId: int): void
+getOrderId(): int
+setOrderId(orderId: int): void
+getPortionId(): int
+setOrtionId(portionId: int): void
+getPortinDetails(): String
+setPortionDetails(portionDetails: String): void
+getStatus(): int
+setStatus(status: int): void
+equals(0: Object): boolean
+equalsIgnoreId(0: Object): boolean
+hashCode(): int
KitchenController
+KitchenController()
+initialize(): void
+setPortionDone(mouseEvent: MouseEvent): void
+orderClicked(mouseEvent: MouseEvent): void
+oPClicked(moseEvent: MouseEvent): void
+updateGui(): void
+findPortionsOfOrder(orderId: int): ArrayList<OrederedPortionsEntity>
+tabOpened(event: Event): void
OrderedPortionsDao
+OrderedPortionsDao(sessionFactory: SessionFactory)
+getAllOrderedPortion(): List<OrderedPortionsEntity>
+getPortionsOfOrder(orderId: int): List<OrderedPortionsEntity>
+getOrderedPortion(orderedPortionId: int): OrderedPoertionsEntity
+createOrderedPortion(orderedPortion: OrderedPortionsEntity): void
+updateOrderedPortion(orderedPortion: OrderedPortionsEntity): void
+deleteOrderedPortion(ordredportion: OrderedPortionsEntity): void
«interface»
IOrderedPortionsDao
LogOutController
+logOut(): void
+cancelLogOut(): void
logoutConfirmation
+uses
+updates
+implements
+update

Class diagram of the software.
(https://drive.google.com/open?id=1Bgt0EkzSaHfup9ruuDcI7_o-S5vRo4ij)