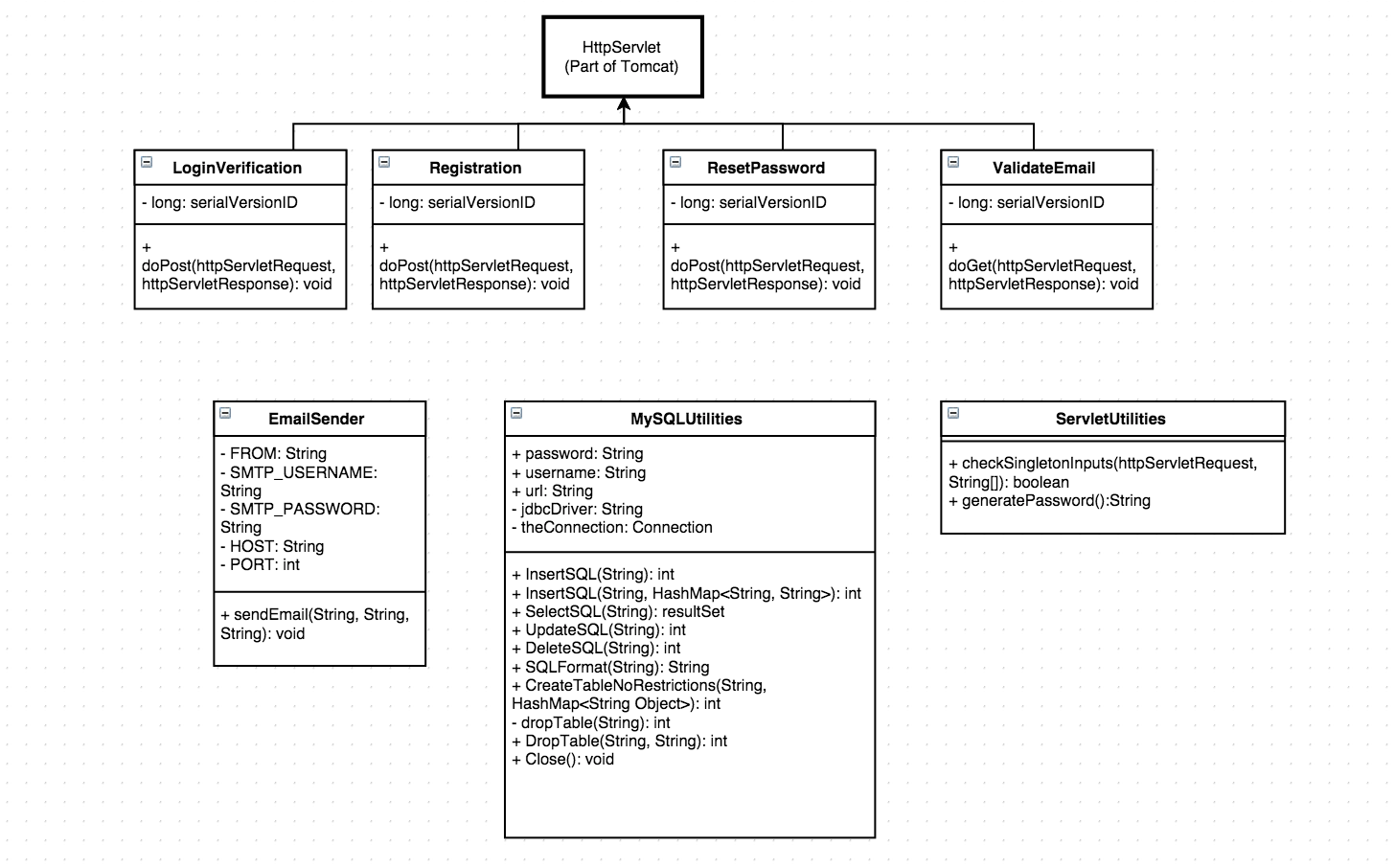
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Iteration | Version | Dates | Comments |
| Mel Chi | 1 | 1.0 | 9/17-9/24 | Added UtilityClass, MySQLUtility Class notes  JUNIT Test |
| Anthony Quigley | 1 | 1.0 | 9/17-9/24 | High Level - Services, Servlets, Database |
| Mitch | 1 | 1.0 | 9/17-9/24 | High Level - All Front-End Components, Added UML Diagram |
| Tim Parzynski | 1 | 1.0 | 9/17-9/24 | JUNIT Test first iteration of MySQLUtility |
| Mel Chi | 2 | 2.0 | 9/24-10/1 | MySQL Utility and String Utility Class, Moved Methods of Utility to the String or Servlet Utility, Helped with new UML, JUNIT Tests on String Utility Class |
| Anthony Quigley | 2 | 2.0 | 9/24-10/1 | Updated High level coded servlets and linked the code |
| Mitch | 2 | 2.0 | 9/24-10/1 | Updated Front-End Components, Added Updated UML Diagram |
| Anthony Quigley | 3 | 3.0 | 10/1-10/15 | Added Admin Servlets, Restaurant Front of House Servlets, Customer Servlets |
| Mitch | 3 | 3.0 | 10/1-10/15 | Added Front End Descriptions |
| Mel Chi | 3 | 3.0 | 10/1-10/15 | Modified what Anthony and Mitch had into the documentation for flowing |

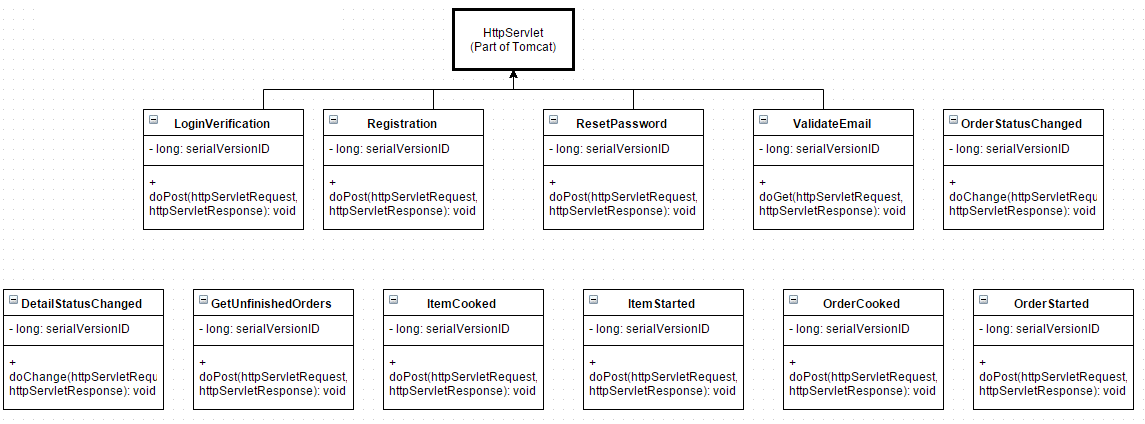
• Analysis and Design o Architectural Analysis • High level system components, processes, services, threads, servers etc. o Detailed Design • Static design o Database design o Class diagrams • Dynamic design o Sequence / State diagrams

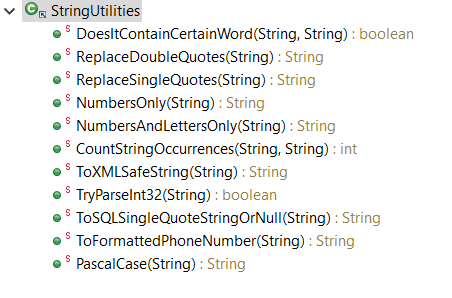
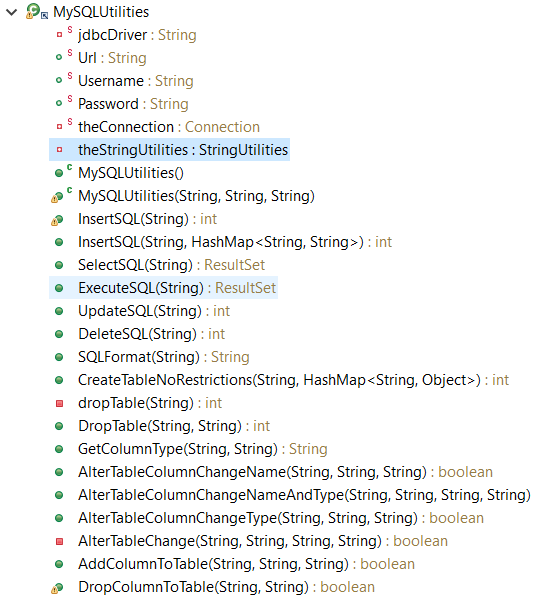
• Implementation o Code o Comment o Structure and style of code o Unit tests

Version 1.0 UML:



### Version 2.0 Just FrontEnd





### High Level

* Services
  + AWS CloudFormation to handle our tech stack
  + AWS RDS MySQL Database
  + AWS EC2 server for managing front end callbacks
    - Running Tomcat to host java servlets
    - Communicates with MySQL database in RDS
  + AWS SES for sending emails on Phoodie’s behalf
* Servlets
  + General (currently located in (\HTML\ajaxdad\WebContent\WEB-INF\classes\)
    - Login Verification (LoginVerification.java)
    - Account Registration (Registration.java)
      * Also sends an email with a validation link (EmailSender.java)
    - Reset Password (ResetPassword.java)
    - Validate Email (ValidateEmail.java)
  + Kitchen
    - Get Unfinished Orders (GetUnfinishedOrders.java)
    - Items Cooked (ItemCooked.java)
    - Items Started (ItemStarted.java)
    - Orders Cooked (OrderCooked.java)
    - Order Started (OrderStart.java)
    - Order Status Changed (OrderStatusChanged.java)
  + Admin
    - Add Restaurant User (AddRestaurantUser.java)
    - Add Menu (AddMenu.java)
    - Add Menu Item (AddMenuItem.java)
    - Change Menu (ChangeMenu.java)
    - Change Menu Item (ChangeMenuItem.java)
    - Change RestaurantUser (ChangeRestaurantUser.java)
    - Delete Menu (DeleteMenu.java)
    - Delete Menu Item (DeleteMenuItem.java)
    - Delete Restaurant User (DeleteResturantUser.java)
    - Detail Status Changed (DetailStatusChange.java)
      * Changes user information
    - Get Restaurant User (GetRestaurantUser.java)
      * Gets the users that have been to the restaurant
    - ListMenus (ListMenus.java)
  + Front of House
    - Place Order (Place Order.java)
      * Places Order
  + Customer Interface
    - Place Order (Place Order.java)
* Front-End Components
  + Implement framework 7, and AJAX in javascript to interact with server
  + Login Page
    - Basic login in that allows user to login, create account, or recover password
  + Registration Page
    - Allows a user to register an account. Asks for Username, password, and email
  + Email Validation Page
    - Simple page that shows user they validated their email
  + Forgot Password Page
    - Allows user to request email to recover password
  + Admin Home Page
    - Allowed to add, edit, and delete users for that restaurant account
    - Allowed to add, edit, and delete menu items for that restaurant
  + Wait Staff Home Page
    - Allows the user to view a dynamically created menu where the items are in an organized way, also allowing them to place an order with each item being able to specify quantity and special instructions.
    - Also contains a page for viewing the current order being placed.
  + Customer Home Page
    - So far it is the same thing as the Wait staff home page. Same functionality, but it is a different page all together.

### MySQLUtility Class

1. InsertSql(sql string) or (string tablename, hashmap <column name, value>
   1. Can take both a insert sql command or Table Name, and values that are listed in a hashmap
2. SelectSQL(sql string)
   1. Takes the SQL string and returns a result set
3. DeleteSQL(sql string)
   1. Takes the SQL string and executes it
4. ExecuteSQL(sql string)
   1. Just executes the sql string
5. CreateTableNoRestrictions(TableName, Hashmap<String, Object>
   1. Creates a generic table
6. DropTable(tablename, password)
   1. Drops the tablename given, password to be added on as well to prevent drops
7. SQLFormat(string)
   1. If there is a ‘ in the string, it will add another ‘ to prevent insertions as well as if the string is “” or blank, it will return the string “Null”
8. AlterTableColumnChangeName(string tablename, old column, new column)
   1. Keeps the column type the same but changes the column name
9. AlterTableColumnChangeNameAndType(String tablename, old column, newcolumn, new type)
10. AlterTableColumnChangeType(String tablename, columnname, newtype)
    1. Changes only the type of the column ,will make data messy if used but still will be given the option for the admin
11. AlterTableChange(String)
    1. Changes everything about the table, can add extra features like is null, or asc int
12. AddColumnToTable(String tablename, columnname)
    1. Adds a new column
13. DropColumnToTable(String tablename, columnname)
    1. Drops the column to an existing table

StringUtility Helpers Class

1. DoesItContainWord(String, string target)
   1. Checks if the word is contained even with spaces inbetween and can take phrases with extra spaces inbetween words
2. ReplaceDoubleQuotes(string)
   1. Returns any string that has double quotes with non escapable double quotes
3. ReplaceSingleQuotes(string)
   1. Returns the string after replacing single quotes with 2 single quote to prevent escape
4. NumbersOnly(string)
   1. Returns a string with only numbers
5. NumbersAndLettersOnly(string)
   1. Returns a string with only numbers
6. CountStringOccurrences(string, string target)
   1. Counts the number of times a string appears, aka, if drop appears twice you may want to avoid the string
7. TryParseInt32(String)
   1. returns the int from a string if successful, useful to remove some clutter in main code
8. ToSQLSingleQuoteStringOrNull(string)
   1. Identical to SQLFormat()
9. ToFormattedPhoneNumber(String)
   1. Since we may be putting phone numbers in, this way it formats into something nice
10. PascalCase(string)
    1. Changes a string to pascal case

### Database Design

UserInfo

* Username
  + Primary Key
* Password
* Email
* Unverified Hash

### Junit Tests

### 

|  |  |
| --- | --- |
| Test Case ID: SQL\_1 | Test Title: InsertSQL |
| Description | |
| Tested whether or not we could add new records to a table. | |
| Pass/Fail | Pass |

### 

|  |  |
| --- | --- |
| Test Case ID: SQL\_2 | Test Title: SelectSQL |
| Description | |
| Tested whether or not we could select fields from the database and return a result set. | |
| Pass/Fail | Pass |

### 

|  |  |
| --- | --- |
| Test Case ID: SQL\_3 | Test Title: DeleteSQL |
| Description | |
| Tested whether or not we could delete fields, then check if it was deleted with select. | |
| Pass/Fail | Pass |

### 

|  |  |
| --- | --- |
| Test Case ID: SQL\_4 | Test Title: CreateTableNoRestrictions |
| Description | |
| Tested whether or not we could create a basic table with no advanced options (IS NOT NULL, etc) | |
| Pass/Fail | Pass |

### 

|  |  |
| --- | --- |
| Test Case ID: SQL\_5 | Test Title: DropTable |
| Description | |
| Tested whether or not we could delete tables from the database. | |
| Pass/Fail | Pass |