Design Specification of “Edible Me”

Jiangyue Yan

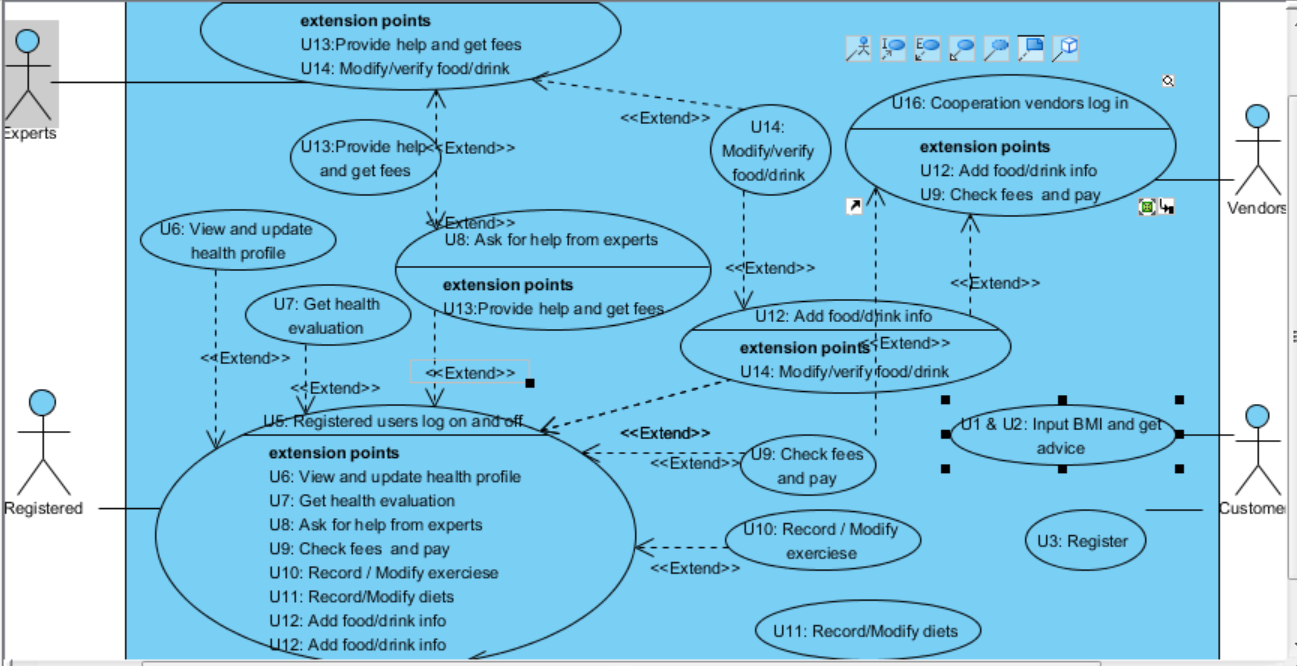
Wenzhou-Kean University

yanjia@kean.edu

This design specification document is used for the further implementation of previous approved project “What to Take” as specified by the project proposal and software requirement specification. At this point, the name of the formal application is designed as “Edible Me”. This name is chosen because it sounds more direct compared to previous project name.

All the design of this project is based on the proposal and the software requirement specification approved before. The process of design considers the detailed database design, package design, database interface design and security policies till this moment.

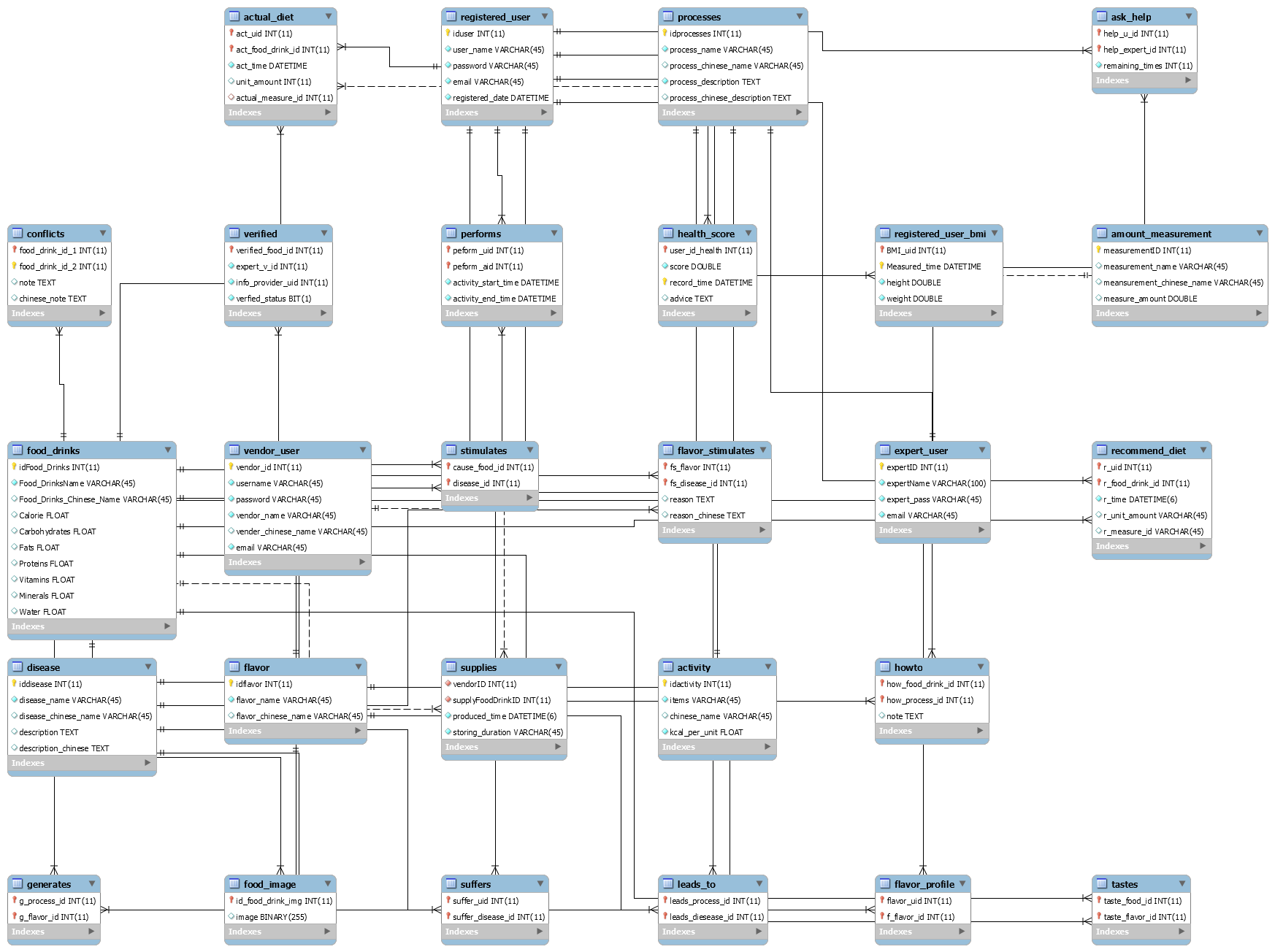
In general, the whole design processes follows the principle of “bottom up”. It starts from database design and ends up to class design. As user interface design works are done previously in the software specification requirement, this stage does not consider about user interface for a second time. This design takes into consideration of grouping different classes for different usages into different packages. It separates the linkage between database API which links to the database and project objects (classes), considering the possibility that objects might be used for purposes other than linking to database. Furthermore, it uses SHA1 as encryption policy for users’ one-way encryption data such as users’ password. The whole design refers to the previous specified user cases as shown below:



Use case diagram of project “Edible Me”

Section I Database Design

The database design is the basic core of this project. The goal of the implemented database is that it should support full data of the daily running of this application. During the design of this database, E-R model is used and the convention of 4NF is followed. In order to be more specific and more easier to check, maintain and extend potential transaction requirements at later stages, the entities and relations are further categorized into three categories, namely, entities without constraints, entities with constraints (functional entities) and relationships.



E-R Diagram of database design of this project

Tables that are entities without foreign keys:

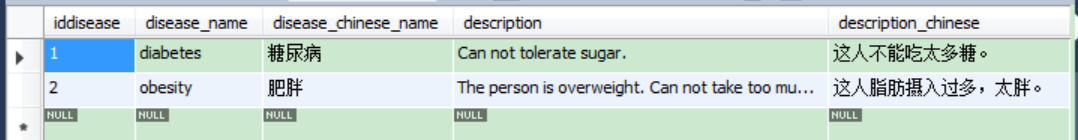
* Activity: Entity which records the user’s exercise information, satisfying user case 10. During the implementation of phase 1, actual data is already put into the activity table. This table allow **users to add information only**, but experts can either **add, modify or delete information.** Data organization example:



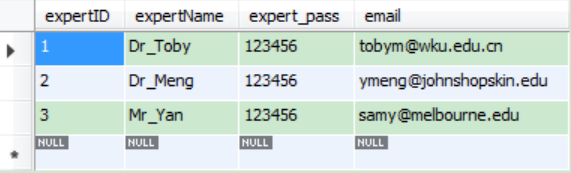
* Amount measurement: Entity which contains information about the usual measurement of amounts. During the implementation of phase 1, tested data will be used. Data organization example:



* Disease: Entity which records different types of diseases. This entity is related to possible further usage of many other user cases. During the implementation of phase 1, fake data was entered into this table. This table allow **users to add information only**, but experts can either **add, modify or delete information.** Data organization example:



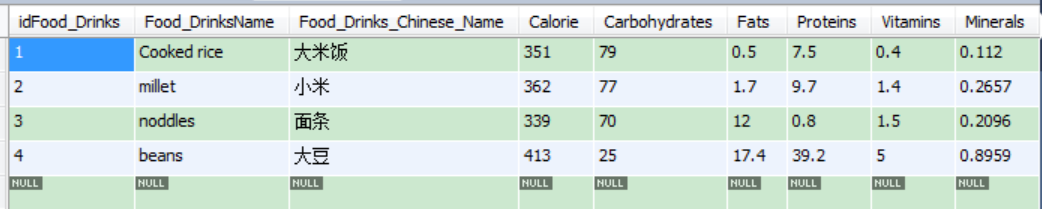
* Expert users: Entity which records basic information about experts. During the implementation of phase 1, test data was used in this table. Data organization example:



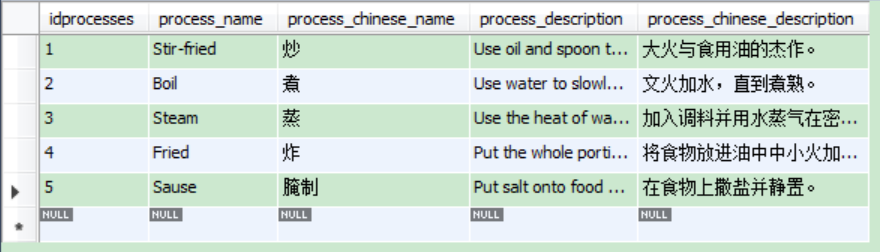
* Flavor: Entity which contains flavors of different dishes, such as bitter, sour or sweet. During the implementation of phase 1, real data will be used. **This table allows experts to access to add, modify or delete only.** Data organization example:



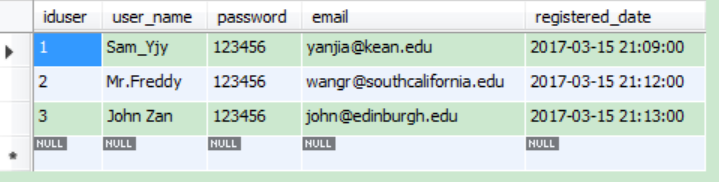
* Food & Drinks: Entity which contains information about food and drink this application refers to. In the implementation phase 1, this table contains partially finished actual data. This table allows users to add data and experts to access to add, modify and delete data. Data organization example:



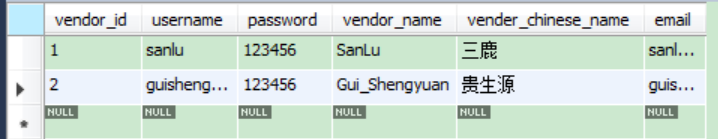
* Processes: Entity which records processes of how food and drinks got processed. This table usually does not have to be modified and only expert users are allowed to reach the table for adding, modifying or deleting data. During the implementation of phase 1, fake data will be used. Data organization example:



* Registered users: Entity which contains information about registered users. This table allows registered users to change their username, password and email address. During the implementation of phase 1, test data will be used. Data organization example:

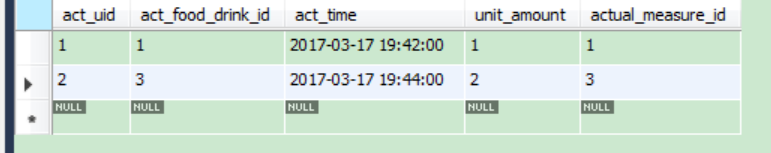


* Vendor users: Entity which contains information about vendors. This table allows cooperation vendors to change their username, password, name of cooperation and email. During the implementation of phase 1, test data will be used. Data organization example:

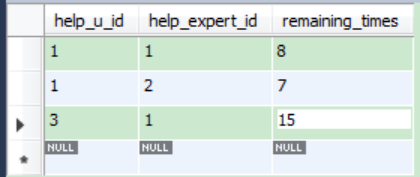


Tables that are entities with foreign keys: (Functional tables)

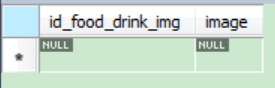
* **Actual user diet:** This entity contains information about who actually ate what at when of what amount, which serves as a very important table managing the information of user health. During the implementation of phase 1, test data will be used for this table. Data organization example:



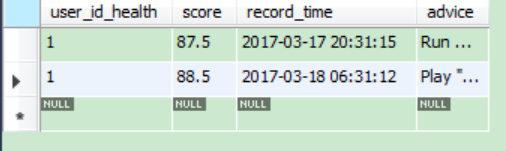
* Ask help: This entity allows registered users to maintain a relationship with certain expert(s) by telling how many times remained for a certain user to ask for help from certain experts. During the implementation of phase 1, test data will be used for this table. Data organization example:



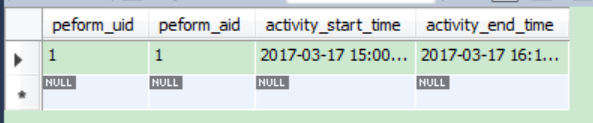
* Food image: This entity contains image of each kind of food. If image(BLOB data structure) are not able to be stored in database, they will be stored as I/O files instead. This table is temporarily not considered during the implementation of phase 1. Data organization example:



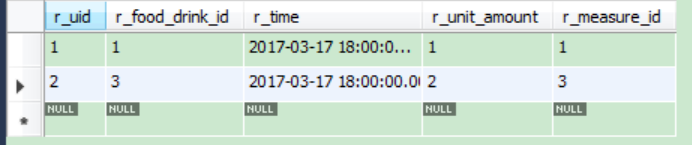
* **Health score:** Users can ask the system to automatically evaluate their health based on all the aspects of data they input to the system. The system can record their health conditions and provide advice for them. This table uses tested data during the implementation of phase 1. During the implementation of phase 2 (and later), real scores shall be evaluated based on the algorithms specified. Data organization example:



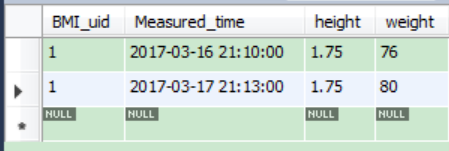
* **Performs:** This table records information about the exercising information of each user. Testing data will be used during the implementation of phase 1. Data organization example:



* Recommend diet: This table records information about recommended diet each meal (denote breakfast as 07:00, lunch as 11:30 and dinner as 18:00) for users. Fake data will be used during the implementation of phase 1. Data organization example:



* Registered user BMI: This table records information about users’ BMI. Testing data will be used during the implementation of phase 1. Data organization example:

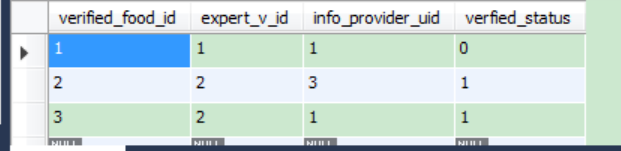


* Verifies: This table records information between food and experts, which means food information is verified by (or waiting to be verified) by experts and the quality of food is ensured. Specifically, this table uses bit 0 to indicate the food is not verified, using 1 to suggest the food is qualified. Thus, the accessing of data may look like following:

INSERT INTO what\_to\_eat.verified

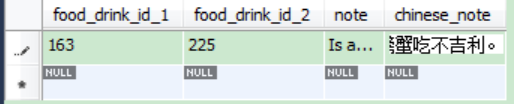
VALUES ('3', '2', '1', b'1');

During the implementation of phase 1, testing data will be used. Data organization example:

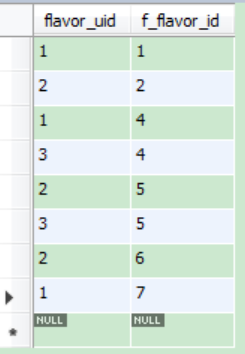


Tables that are relations:

* Conflicts: This is a one-to-one relationship between different types of food, showing the combination of which two food may cause potential problems. The note columns indicates how the conflict may cause problems. Fake data are used during the implementation of phase 1. Only experts are allowed to achieve this table. Data organization example:



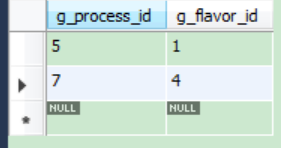
* Flavor\_profile: Record which user prefers which kind(s) of flavors. This table is available for each users to add / modify their own information about their flavor preferences and for experts to view when users are needing help from experts. Data organization example:



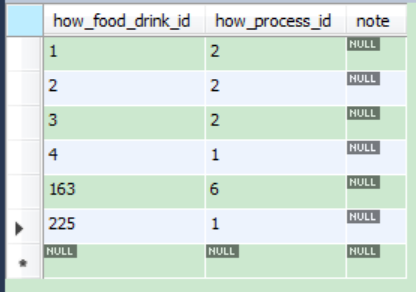
* Flavor\_Simulates: This is one-to-many relationship indicating the possible incurs of certain diseases due to taking too much specific kind of food / drink of similar flavors. For example, eating too much rice may exacerbates the situation of diabetes. Testing data are used during the implementation of phase 1. This table is designed only for experts to add / modify or delete information. Data organization example:



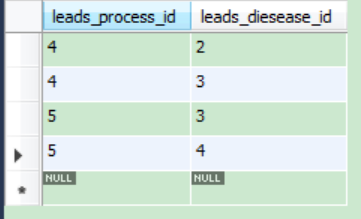
* Generates: This table contains the relationship between processes and flavors that this certain process tends to lead to. For example, the process honeydew tends to lead to sweet flavors while the process sauce tends to lead to salty flavors. Testing data are used during the implementation of phase 1. Data organization example:



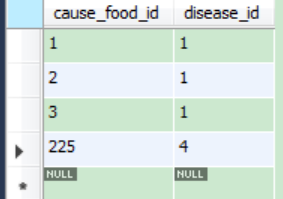
* How to: This is a one-to-one relationship showing how food is usually processed. During the implementation of phase 1, tested data will be used. This table allows all kinds of users to access / modify and delete. Data organization example:



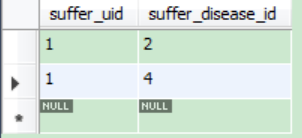
* Leads to: This table specifies which processes may lead to which kind of diseases, can be add or modified by expert users. Fake data will be used during the implementation of phase 1. Data organization example:



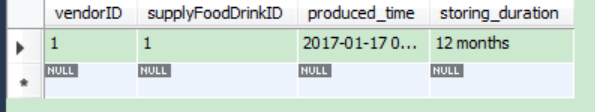
* Simulates: This table builds a many-to-many relationship between food and cause of diseases, which means which food might cause which diseases. Fake data will be used during the implementation of phase 1. This table allows expert users and registered users to add and modify. Data organization example:



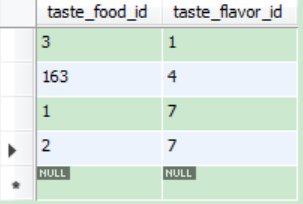
* Suffers: This table forms a many-to-many relationship between users and diseases. Testing data will be used during the implementation of phase 1. Data organization example:



* Supplies: This table forms a one-to-many relationship between vendors and food / drink. It specifies which food / drink is sold by which vendors. Data organization example:



* Tastes: This table provides information about the tastes (flavor) of different food/drinks. This table allows both experts and registered users to add / modify information. Testing data will be used during the implementation of phase 1. Data organization example:

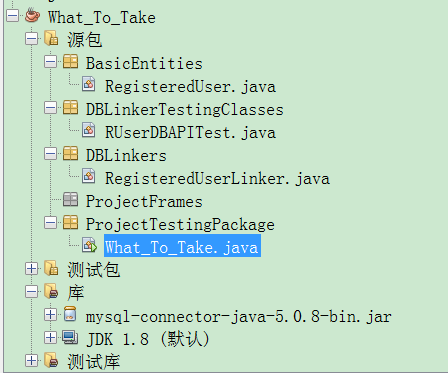


Note: Table names that are in bold are considered to be the essential supports of this application and thus should be implemented first and as soon as possible.

Tables that are entities without foreign keys are most important tables to consider. Tables that are relations are second important to consider. Tables that are entities with foreign keys are mainly work as auxiliaries to this application.

Section II Package Design

This application consists of various components, thus the management and design of packages[[1]](#footnote-0) for different usages becomes particularly important. The whole project package structure is shown as below:



Pic: Package management structure of this application

The whole package structure is divided into two parts: application packages and testing packages. The packages for application packages are:

* Basic Entities: This package used to store user roles and application entities, such as the RegisteredUser class, the FoodDrink class and the Activity class.
* DBLinkers: This package is used to provide database linking application interface (mainly for entities). The usage of package will be further demonstrated in section III of this document.
* ProjectFrames: Which provides user interfaces for this application to interact with project users of all kinds of roles.

The packages for testing are:

* + DBLinkerTestingClasses: Classes in this package are used to test the linking between database and basic entities.
  + ProjectTestingClasses: Classes which are written to test certain objects or frameworks.

Section III Database-Application Interface Design:

For each entity and relation in this project, there should be a database application interface designed. The interface subroutine provides only basic add, modify and update functions for this certain entity or relation. For example, the database application interface of entity registered user is shown as below (Due to the scale of the document, the body part of constructors or methods are omitted):

package DBLinkers;

//import statements, omitted.

/\*\*

\* @author Sam Yan

\*/

public class RegisteredUserLinker {

//Variables for linking database:

private Connection connect;

private PreparedStatement preparedStatement;

private ResultSet resultSet;

//Variables for registered user information.

private RegisteredUser ruser;

//Initialize linking environment

public RegisteredUserLinker() {}//end constructor

/\*

\* Set and Get: Accessors for entity/relationship able to acess this interface.

\*/

public void setRUser(RegisteredUser ruser){}//end method

public RegisteredUser getRUser() {}//end method

//Read information according to information already known (such as primary key):

private void readFromDB() {}//end method

/\*Method of registering user to database. Including reading his/her id

and write his/her information into database.

In this case, it is a special method being written in the interface\*/

public void registerUser() {}//end method

//Typical writing to database function

private void addToDB() {}//end method

//Typical updating function.

public void updateDB(boolean isPasswordChanged) { }//end method

}//end class

Code Example of showing a typical database application interface for this project

There are 28 entities and relationships in total, which means that 27 such database linking interfaces shall be built. They are packed in the project package DBLinkers.

Each application interface shall then be tested by subroutines in the test package, one testing example of the database application interface is designed as following:

package TestingClasses;

import BasicEntities.RegisteredUser;

import DBLinkers.RegisteredUserLinker;

/\*\*

\*

\* @author Sam

\*/

public class RUserDBAPITest {

private RegisteredUserLinker userLinker;

public RUserDBAPITest() {

userLinker = new RegisteredUserLinker();

testRead();

testWrite();

testUpdate();

}//end cons

public void testRead() {

// Testing reading from registered user db link:

RegisteredUser oldUser = new RegisteredUser();

oldUser.setRe\_userid(1);

userLinker.setRUser(oldUser);

oldUser = userLinker.getRUser();

System.out.println("Old user: " + oldUser.getRe\_username());

System.out.println("Password: " + oldUser.getRe\_password());

}//end method test read

public void testWrite() {

//Testing register user to db link:

RegisteredUser newUser = new RegisteredUser();

newUser.setRe\_username("Sam J");

newUser.setRe\_password("123456");

newUser.setRe\_email("samy@163.com");

newUser.setRe\_registeredTime("2017-03-18 00:00:00");

userLinker = new RegisteredUserLinker();

userLinker.setRUser(newUser);

userLinker.registerUser();

}//end method test write

public void testUpdate() {

// Testing registered user to db link:

RegisteredUser updatingUser = new RegisteredUser();

updatingUser.setRe\_userid(1);

//The second bool is for whether user is registering.

userLinker = new RegisteredUserLinker();

//Set the user object first in order to get information later:

userLinker.setRUser(updatingUser);

//Read the user according to id:

updatingUser = userLinker.getRUser();

System.out.println("Username: " + updatingUser.getRe\_username());

//Change the user's information

updatingUser.setRe\_password("123457");

updatingUser.setRe\_email("223560119@qq.com");

//Set user

userLinker.setRUser(updatingUser);

System.out.println(updatingUser.getRe\_username());

//password changed is true.

userLinker.updateDB(true);

}//end method test update

}//end class

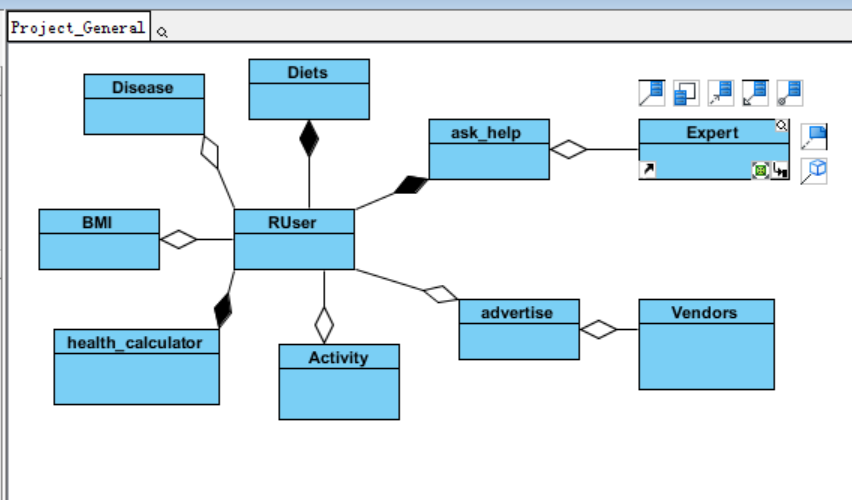
The testing result above indicates that all inserting, updating and reading operations for

registered users’ table works pretty well. Thus, this database application interface is considered as designed and implemented properly.

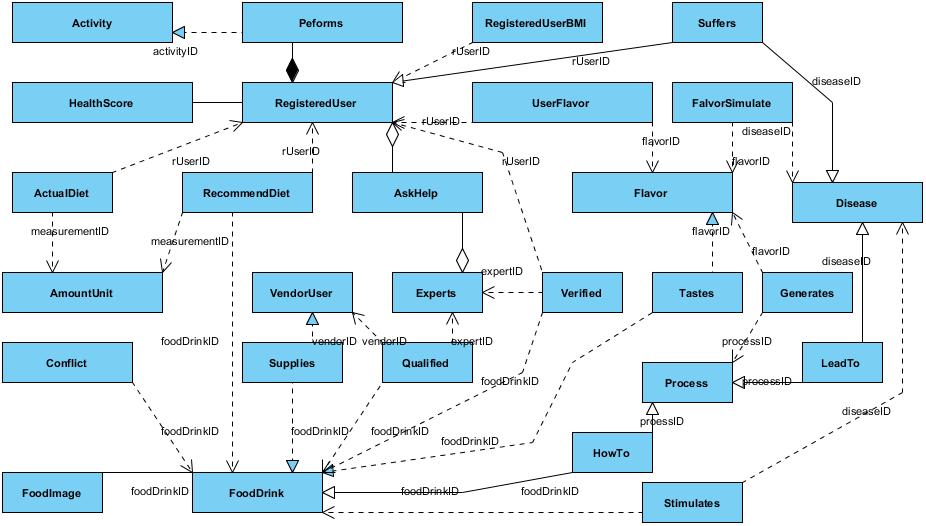
There shall be 28 such testing examples, one entity / relation each. They are all grouped together in the project’s package DBLinkerTestingClass.

Section IV Class Diagram Design

Except from database application interfaces and entity classes which works together to support the basis of this project, some interactive classes shall be considered in order to satisfy the use cases and functions specified in previous documents. Those classes are depicted as a further project general class diagram as shown below.



Pic: Very Brief Interactive classes Diagram of the project



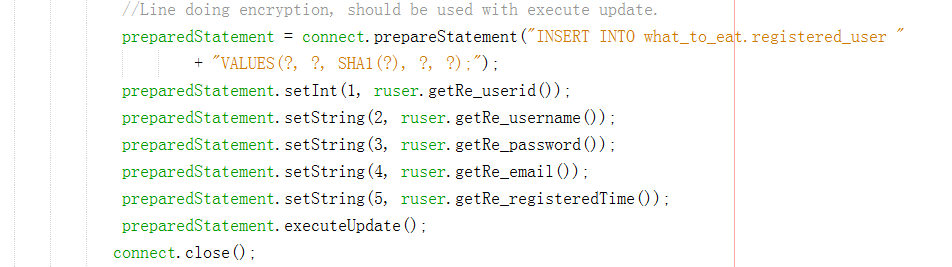
Pic: Detailed class diagram of this project

This diagram is a role-based class diagram considering the interactive functions between different roles of the application user. The class AskHelp forms a many-to-many relationship between registered user and expert users. Once a user is asking help from experts about his or her nutritional plans, the expert than can access users’ different data as shown above.

Also, through Advertise class, which forms a one-to-many relationship between user and vendors (a user can view many advertisements), the cooperation vendors are able to advertise their products.

Section V Security Policies

According to official police report cases, users tend to use their password for critical services (such as bank services) for general services as well. Thus, this project provides one-way security guard using SHA1 algorithm for user passwords. In later stages, user-sensitive data such as users’ health profile shall also satisfy the security policies. However, due to the health profile is a bidirectional information, thus AES algorithm shall be used to encrypt and decrypt such information. For example, while registering a new user, this application uses following subroutine to write encrypted password into database:



Also, similar cases applies when users try to update their password.

References

1. Abraham Silberschartz, Henry F. Korth and S.Sudarshan, Database Concepts sixth edition, in 2011, McGrawHill, New York
2. Calorie consumption of different activities: URL: <http://fitness.39.net/a/161013/4990792.html?open_source=weibo_search>

<http://wenku.baidu.com/link?url=dbabDB1SuPL1AP9NrfOmzAkXwgsXnBpF6kWG2sifd_ehq1W_xqxJn8lZcjJ4GICfTplzPiNoFklIAyk_TQNHLu5_vjNf0IRlUeah7M9IMhq>

1. CCTV program “Laws on line”, report cases of collation, URL: <https://v.qq.com/x/cover/d2uhjthvr0w1301/y00233mfntv.html?ptag=tips.xw&pt_src=3&ADUIN=896298116&ADSESSION=1489818134&ADTAG=CLIENT.QQ.5515_.0&ADPUBNO=26657>
2. Mike O’ Docherty, Object-Oriented Analysis & Design: Understanding System Development with UML 2.0, in 2005, John Wiley & Sons, Ltd
3. Sam Yan, Proposal About Automatic Menu System: What to Take, in Feb, Wenzhou-Kean University
4. Sam Yan, Software Requirement Specification of Automatic Menu System: What to Take, in Feb, Wenzhou-Kean University
5. Xie Fuen, Huang Hongzhong and Zeng Xun e.t., Similar Sources of Food and Medicine, in Aug, 2012, Guangzhou Science and Technology

1. Here, package means a file structure that groups similar classes of objects, rather than packages on Internet protocols or other occasion. [↑](#footnote-ref-0)