

# FINAL YEAR PROJECT REPORT

# MyDiabetiser

# Student: Yavor Nanev Student Number: 11060247 Course: LM051 Computer Systems Supervisor: DR. Nikola S. Nikolov

# Year:2016

Contents

[Declaration 4](#_Toc448404465)

[1 PROJECT SUMMARY 5](#_Toc448404466)

[2 INTRODUCTION 6](#_Toc448404467)

[2.1 Web Application 6](#_Toc448404468)

[2.2 Machine Learning 7](#_Toc448404469)

[2.3 Android Mobile Application 7](#_Toc448404471)

[2.4 Motivation 8](#_Toc448404472)

[3 RESEARCH 9](#_Toc448404473)

[3.1 Tech Research 9](#_Toc448404474)

[3.2 Apps 9](#_Toc448404475)

[3.3 Web Sites 9](#_Toc448404476)

[3.4 Diabetic Ratios 11](#_Toc448404477)

[3.5 High Blood Sugar Correction 11](#_Toc448404478)

[3.6 Glucometers 12](#_Toc448404479)

[4 TECHNOLOGIES 13](#_Toc448404480)

[4.1 Outline 13](#_Toc448404481)

[4.2 HTML 13](#_Toc448404482)

[4.3 CSS 13](#_Toc448404483)

[4.4 JavaScript 13](#_Toc448404484)

[4.5 SQL 14](#_Toc448404485)

[4.6 JAVA 14](#_Toc448404487)

[4.7 Enterprise JavaBeans Technology 14](#_Toc448404488)

[4.7.1 NetBeans IDE 8.1 14](#_Toc448404489)

[4.8 GitHub 15](#_Toc448404490)

[4.9 GlassFish 4.1 15](#_Toc448404491)

[5 DESIGN 16](#_Toc448404492)

[5.1NonFunctional Requirements 16](#_Toc448404493)

[5.1.1Extensibility 16](#_Toc448404494)

[.5.1.2 Performance 16](#_Toc448404495)

[5.1.3 Maintainability 16](#_Toc448404496)

[5.1.4 Testability 16](#_Toc448404497)

[5.1.5 Scalability 17](#_Toc448404498)

[5.1.6 Reusability 17](#_Toc448404499)

[5.1.7 Security 17](#_Toc448404500)

[5.2 Universal requirements 17](#_Toc448404501)

[5.*3* Functional Requirements 18](#_Toc448404502)

[5.5 User Requirements 19](#_Toc448404503)

[5.6 Data Requirements 20](#_Toc448404504)

[5.7 Security Requirements 21](#_Toc448404505)

[5.8 Web Site Features 21](#_Toc448404506)

[5.8.1 Calculate personalized user ratios 21](#_Toc448404507)

[5.8.2 Update user ratios 25](#_Toc448404508)

[5.8.3 High Blood Sugar Correction 26](#_Toc448404509)

[5.8.4 Creating And Saving Meal 27](#_Toc448404510)

[5.8.5 Graphs Representations 27](#_Toc448404511)

[5.9 Web Page 28](#_Toc448404512)

[5.9.1 Sign-In Page 28](#_Toc448404513)

[5.9.2 Log In Page 28](#_Toc448404514)

[5.9.3 Diary Page 28](#_Toc448404515)

[5.9.4 Account page 28](#_Toc448404516)

[5.9.5 Calculator 29](#_Toc448404517)

[5.9.6 Graph Page 29](#_Toc448404518)

[5.9.7Reports Page 29](#_Toc448404519)

[6 SECURITY AND TESTING 30](#_Toc448404520)

[6.1 Outline 30](#_Toc448404521)

[6.2 Injection 30](#_Toc448404522)

[6.3 Broken Authentication & Session Management 32](#_Toc448404523)

[6.4 Cross-Site Scripting(XSS) 33](#_Toc448404524)

[6.5 Unsecure Direct Object References 34](#_Toc448404525)

[6.6 Unvalidated Redirect and Forwards 36](#_Toc448404526)

[6.7 Passwords 37](#_Toc448404527)

[7 IMPLEMENTATION AND TESTING 40](#_Toc448404528)

[7.1 Outline 40](#_Toc448404529)

[7.2 Creating Enterprise Application 40](#_Toc448404530)

[7.3 Web Pages 41](#_Toc448404531)

[7.4 Database and entity classes 45](#_Toc448404532)

[7.5 Sign-Up and Sign-In 48](#_Toc448404533)

[7.6 Graphs and Reports 48](#_Toc448404534)

[7.7 Automated Testing 49](#_Toc448404535)

[7.8 Validation Testing 52](#_Toc448404536)

[7.9 Challenges 53](#_Toc448404537)

[7.9.1 Graphs challenge 53](#_Toc448404538)

[7.9.2 Domain Name Issue 55](#_Toc448404539)

[8 Conclusion 56](#_Toc448404540)

[9 References 57](#_Toc448404541)

Declaration

I declare that I have produced this Final Year Product without the prohibited assistance of third parties; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in an identical or similar form to any other examination board. The work was conducted from September 2015 to April 2016, under the supervision of Dr. Nikola Nikolov at the University of Limerick.

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1 PROJECT SUMMARY

The aim of the “MyDiabetiser” project is to provide assistance and easy management of diets and medicine for people with Diabetes. The project will be suitable to use for all people with Diabetes type1, using either fast active or slow active insulin.

In addition, the project will also keep a record of personal blood sugar levels for the user, as well as the times and units of insulin taken. Records will be presented to the user in tabular and graphical formats for ease-of-use and understanding to the user.

The main goal of the project is to predict the amount of units of insulin that the diabetic user will need to take each time their insulin injection is due. The system will use records for each user in order to characterise user behavior and their individual reactions and response(s) to insulin for more accurate results. The project shall be implemented using a combination of Web Applications, Interfaces, and Machine learning.

Being a diabetic means constantly monitoring your sugar levels, intake and projected future levels. Many diabetics agree that this is a time-consuming task which requires full-time effort to manage. This project will assist real-world diabetic users, allowing them to digitise all the records and notes that a diabetic would need to record every day. All the data will be kept in a database for the user, enabling post-processing of the data for reports and insulin unit suggestions.

2 INTRODUCTION

2.1 Web Application

The Web Application part of this project is implemented using a JAVA EE application. The functionality of the site is to provide basic information for a new user of the application, ranging from site usage to tips, diets and “how to” documents. The user will be able to input relevant personal data (insulin units, blood sugar levels, etc.) and display this data in diagram form for ease of understanding. In addition, the user will be able to get suggestions for how many insulin units to be taken based on previous results and records. They will be able to create and save a variety of meals with different ingredients which can be used in the calculation of the number of insulin units to be taken. In addition, there are more general factors that the user may be prompted to input. All of these factors will play a role in determining the final amount of units needed in order for a more accurate insulin prediction. All of the food products and ingredients will be grouped in the diabetes food pyramid and stored in the database. Reports will be generated on request and displayed in diagram form using JavaScript. The database management system will be implemented using SQL. Open-source application server GlassFish4.1 is used to deploy the project. Apache server was used to handle HTTP requests and send them to the Glassfish application server.

2.2 Machine Learning

The Machine learning algorithm will be responsible for creating the suggested insulin dosages for each user and meal. To predict the correct amount of units to be taken, the algorithm takes into consideration many of the user’s personal details, which is unique to each time the user is calculating their insulin intake. For every single meal the application is used with, it will calculate the number of carbohydrates in the meal. Based on known medical formulae, this information will be used to determine how much rapid-acting insulin will be needed to dispose of those carbohydrates. The units calculated from the meal are only approximate values, additional factors will affect the final calculation, increasing or decreasing the final calculation value for the user. The algorithm will browse through the previously saved scenarios, suggestions and user behavior in the database in order to improve the final calculation. If it is a new user on the system and they do not have any personal records yet, the algorithm will automatically match the user with the nearest user match existing in the database. They will then use the records in order to provide suggested insulin levels until the user profile chosen.

2.3 Android Mobile Application

For the android application to be used in this project, the application will be used for mobility and allow the system to work in either online or offline mode. The application will also have a panic button that can alert family or friends in case of user emergency. For consistency, the input grid will maintain the same format as the diaries that diabetics usually use to log their data. Like the main application, the mobile app can also be used for suggestions on how many insulin units the user should take. The database management system will be implemented in SQLite. The database will be synced with the website database and will upload and download data every time the application goes online.

Due to time constraints, the android mobile application will not be implemented in this project. However, but it is included in the documentation discussion and potential future developments for the project.

2.4 Motivation

I have been a diabetic for many years now. I therefore have the first-hand experience of how difficult it is to manage and control insulin intake in the management of this condition. With this in mind, it is almost impossible to successfully administer the correct level of insulin in order to keep blood sugar levels in order. I think that the system will contribute greatly to this aspect of my life. It will help decrease the time necessary calculating food carbohydrates and figuring out how much insulin to take, both of which can nowadays easily be done from a smartphone platform. This will be much more convenient as the diabetic report diary (paper format) which is used to input records of sugar levels will therefore not need to be carried by the user. Most users will obviously be carrying their phones for day-to-day uses.

In addition, the mobile phone will obviously be a more convenient and easier-to-use method to display personal records in cases such as doctor visits, etc. The doctor can easily view the summary of the user data in diagram format, allowing for huge time-saving for both doctor and patient/user.

Finally, the biggest motivation of all is not the time that I am spending without such a system but the health improvements that I can achieve with such as system. The more accurate the number of insulin units I take are, the better my body will be able to keep my blood sugar levels in order which means a better and longer life for me and anybody else using the system.

3 RESEARCH

3.1 Tech Research

After researching programming languages, I decide to use JAVA for the implementation of this project because it allows for the implementation of the main functionalities and architectures, in both a Web-based application and in potential future Android applications, without rewriting or duplicating code. Modifications could be easily implemented in both systems as the software implementation will be the same. NetBeans was the platform chosen to develop the Java Application, as it supports JAVA EE and has built in GlassFish integration that allowing work on the local host before deployment to the remote server. Also, NetBeans supports java beans which will be instrumental in managing the database and its implementation. For the mobile application, Android OS was selected due to the large installed base which exists at present (approximately 17,000 different devices run Android worldwide). This will mean that the app will be more readily available and accessible to a larger audience when it is developed.

3.2 Apps

The top 10 apps reviewed in (healthline.com, n/a) have similar functionality to the proposed project here, but none of them provide an integrated and easy-to-use solution such as the one proposed here. For example, only one of those ten applications considered claims that they can predict the number of insulin units to take. This feature (unit’s suggestion) is obviously one of the key/important sections of the system but obviously is very difficult to implement and/or maintain. Most of the apps considered are orientated to display reports. There are no web application solutions that I found, and none has a user interaction and similar functionality to this proposed project.

3.3 Web Sites

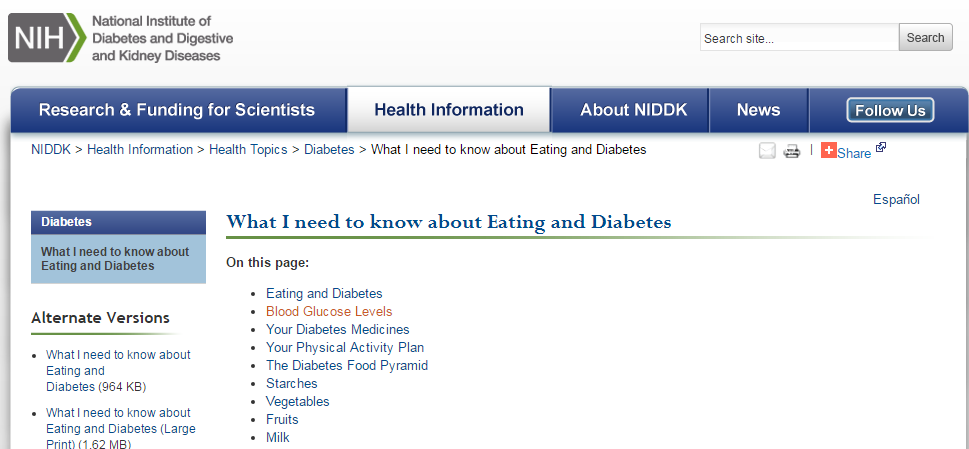
In this section, the outline and functionality of the pertinent websites for this project are considered. These include the following: -

**Diabetes Education Online, University of California, San Francisco website**



(dtc.ucsf.edu, 2016)

The website is quite thorough and complete, containing all of the information a diabetic needs to know. However, the site does not provide data logging or user account functionality, so that information such as suggesting units of insulin, personal records or record displays are not available. However, this site will be very helpful for a people who have been diagnosed as a diabetic.

**NIH website**

(niddk.nih.gov, 2014)

This website is similar to the previous which provides lots of information about how to calculate ratios, what food should eat or not, etc. etc. Again, there is personalization or calculation functionality included, so features such as monitoring, calculation, etc. are missing.

While I was doing my research and I did not find websites that are doing similar functions to the project that I am planning to do.

3.4 Diabetic Ratios

This section will show some of the means and methodologies which diabetics can use to find and count their carbohydrates to insulin ratios. The first method is what is called the“500 rule”. This states that the carbs to insulin ratio can be calculated by simply dividing 500 by this figure (i.e. total amount of units of insulin the person is taking). For example, if the total units are 40 a day, 500 divided by 40 is equal 12.5 hence a person needs to take 1 unit of insulin per every 12.5 carbs that person is going to eat. This is the base ratio from which a new diabetic can start off. As the ratio can vary and depends on many factors, these ratios need to be monitored by the diabetic. It is important to update the ratio from the monitored records in order to keep the blood sugar levels in range. The ratio can vary during the different time of the day so it is a good idea that the diabetic has three different ratios in the record, one for breakfast, one for lunch and one for dinner .The diabetic needs to monitor their blood sugar levels in order to evaluate the efficiency of the respective ratios. If the ratios are not correct, by looking at the previous records the diabetic should be able to determine if there is a need to increase or decrease the insulin units, in order to regulate and recalculate the ratios again and keep the blood sugar levels maintained.

3.5 High Blood Sugar Correction

High blood sugar correction is applied when (before a meal) the blood sugar levels are off target. In this case, the diabetic needs to increase the units of insulin that are calculated from the “500 rule“. This is because the “500 rule” calculates the ratio, with the assumption that the sugar levels are on target. To correct the insulin units the diabetic needs to know how much (mmol) of sugar one unit of insulin will reduce.

This is calculated using the formula 1,800 divided by the total units of insulin that person is taking per day, with the results given in (mg/dl). For example, if the total units are 40.

1,800 divided by 40 is equal 45 .After converting 45 to (mmol) the result appears as 2.5 (mmol). The final answer indicates that every unit of insulin will reduce 2.5(mmol). If a new diabetic is not sure what the amount of total insulin for the day should be, a suggested amount of units can be calculated by multiplying the diabetic’s current weight by 0.55. For example, if the diabetic’s weight is 70 kilograms, 70 multiplied by 0.55 is 38.5. This, therefore, is the daily suggested amount of insulin units approximated for the body weight.

3.6 Glucometers

A glucometer is a device that diabetics use to check the blood sugar levels. This device has its own memory and keeps all the records. Those device has a small screen and are usually limited to displaying one result at a time. The data from the devices can be extracted by connecting the device to the computer using a USB to audio jack cable. This can be very useful for this project as this can be used to automate or streamline the data and record inputting for the user accounts, meaning the user will not have to input all the results manually.

4 TECHNOLOGIES

4.1 Outline

This chapter is about the technologies that have been used in the project.

4.2 HTML

HTML is a markup language for creating web documents (web sites/pages). This project uses HTML in .JSP (Java Servlet Page) pages which allow it to be combined with java. Most modern browsers are capable of reading and displaying HTML documents.

4.3 CSS

Cascading Style Sheets are used to create and call upon predefined styles to help with the visualization the project data and results. In these classes, fonts, colors, and margins are defined for the various styles. Advantages of using CSS include the fact that using the same style sheet among all the .JSP web pages mean there is no need to hard code the style for each HTML element. CSS is very powerful and useful in order to improve the HTML visualization and allows HTML to focus on the site structure rather than appearance.

4.4 JavaScript

JavaScript is a high-level, dynamic and interpreted programming language (wikipedia.org, 5 April 2016). JavaScript, Ajax (asynchronous JavaScript and XML) and JSON (JavaScript Object Notation) are used to transfer data from client to server and vice versa, without the need to reload the current web page. This mix of technologies helps to speed up the process of dynamic data exchange. Furthermore, javascript is used to dynamically display data on the webpages wherever is necessary.

4.5 SQL

SQL (Structured Query Language) is used to access and manipulate databases. SQL is a well-understood technology, it is very powerful and has been used for a number of years. The project uses SQL to insert, retrieve, update and delete data from the database. The project relies heavily on the data collected from the users. SQL provides an easy, simple and secure data manipulation pathway that is required to update the database with this information.

4.6 JAVA

JAVA is a programming language used in this project for both the client and server-side application development. On the client side, java is mostly used to handle the sessions from the browser and collect data sets from server side and displayed in the browser. One of the main reasons behind choosing JAVA for the project is the fact that the majority of the Packages and Design patterns that are created can be used in the future development of the Android application.

4.7 Enterprise JavaBeans Technology

Enterprise JavaBeans (EJB) technology is the server-side component architecture for Java Platform, Enterprise Edition (Java EE). EJB technology enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology. (Oracle.com, Unknown)

Using EJB technology provides an easier and more secure way to access Database Entity Classes. Beans are the classes which handle communication between the database and the servlets. Once a java bean is created and functionality is implemented, the bean can be injected into the servlet so the servlet can use the beans methods.

4.7.1 NetBeans IDE 8.1

NetBeans is an open-source project dedicated to providing rock solid software development products (the NetBeans IDE and the NetBeans Platform) that address the needs of developers, users and the businesses who rely on NetBeans as a basis for their products. In particular, this allows developers to develop these products quickly, efficiently and easily by levering the strengths of the Java platform and other relevant industry standards (netbeans.org,2016). This platform is used to create the project as an Enterprise Web Application since it is more efficient and straight-forward.

In NetBeans all the mapping and configuration files are auto generated which makes working on the important parts of the project more efficient. There are a variety of different plugins with NetBeans, for example, the GitHub plugin for NetBeans allows it to download, upload and commit files to GitHub very fast.

4.8 GitHub

GitHub is a web-based Git repository hosting service. It offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. Unlike Git, which is strictly a command-line tool, GitHub provides a Web-based graphical interface and desktop as well as mobile integration. It also provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project. (wikipedia.org, 6 April 2016).

In the process of developing this project, three different computers/platforms were used, a personal laptop, one of the PCs at the ICT-learning center in University of Limerick and a static IMac at home. GitHub allowed the distributed development of the project across all of these three distinct systems.

4.9 GlassFish 4.1

GlassFish is an open-source application server project started by Sun Microsystems for the Java EE platform and now sponsored by Oracle Corporation. The supported version is called Oracle GlassFish Server. GlassFish is free software, dual-licensed under two free software licenses: the Common Development and Distribution License (CDDL) and the GNU General Public License (GPL) with the classpath exception (wikipedia.org, 5 November 2015).

GlassFish server is used to deploy and run the website. Originally GlassFish was installed on a remote server. After some problems experienced (refer to Issues in developing a process, later in this report for more information) the decision was made to use Glassfish on a local Host instead, to overcome these issues.

.

5 DESIGN

5.1NonFunctional Requirements

5.1.1Extensibility

This will be important for the project to allow for the future growth of the project. I intend to use the factory methods to improve extensibility. Additional food categories and products must be easily added to the website without impacting the existing functions

.5.1.2 Performance

This concerns the speed of the website. All the pages need to load as quickly as possible. Run time of the code execution on the server side needs to be reduced to a minimum in order to provide faster transfer of data.

The original implementation planned to use JavaScript and AJAX to minimize the time or the data transactions by skipping the page reloading. If it became necessary, java threads were employed to save time with accessing data from the database.

5.1.3 Maintainability

The code for this project must be maintainable. That is why good OOD architectures should be employed throughout, the code should be kept clean and commented. This should make it easier if changes need to be made.

5.1.4 Testability

The project needs to be thoroughly and rigorously tested. They are going to be numerous mathematical calculations that need to be validated to ensure they produce the correct results. Needless to say, these calculations could be critical do the user. I also need to test all the inserts, deletions and updates of the database as the user's inputs need to be handled correctly because they will have a critical impact for the some of the calculations that will be done for the user.

5.1.5 Scalability

Scalability is an important aspect of this project. The system should be capable of growing exponentially with new data which is added to the system without affecting the functionality of the system.

5.1.6 Reusability

When creating the framework, this should be made without coupling and avoid dependencies from the servlets. The same packages should be able to be reused in the different systems if necessary.

5.1.7 Security

Security is an important part of the project. Personal user information and the user records are critical factors. It is necessary to ensure that no unauthorized parties can gain access to this information and most importantly update the information.

5.2 Universal requirements

|  |  |
| --- | --- |
| NUM | DESCRIPTION |
| 1 | User –Friendly and secured system and interface. |
| 2 | Register new user. |
| 3 | Secure login into the system. |
| 4 | Display user record |
| 5 | Calculate Carbs-to-Insulin ratios |
| 6 | Calculate and display suggestion of insulin to be taken |
| 7 | Display food products that can be selected for creating a meal |

5.*3* Functional Requirements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Num | Description | | Accept | Critical |
| 1 | input field for user kilograms. | | Numeric data | YES |
| 2 | Input field for user daily total amount of units rapid insulin | | Numeric data | NO |
| 3 | Selectable input for user`s body type. | | Radio Group selection | YES |
| 4 | Input fields for user carbs-to-insulin ratios. | | Numeric data | NO |
| 5 | Input fields for daily user record for insulin. | | Numeric data | NO |
| 6 | Input fields for daily user record for blood sugar levels. | | Numeric data | NO |
| 7 | Update User carb-to-insulin ratios -button or automated. | | Accepts click | YES |
| 8 | Save buttons in general. The system should store the user input correctly to the database. | | Accepts click | YES |
| 9 | Reports button. The system should display user personal diary records and ratios updates history. | | Accepts click | NO |
| 10 | Log-out button. The system should destroy all the sessions  and cookies created and redirect to the log-in page. | | Accepts click | YES |
| 11 | Log-in button. The system should match the user credentials with database and return true or false. If true display user  Homepage. If false return appropriates error and redirects to Log-In page for another user attempt. | | Accepts click | YES |
| 12 | Email input field. | | Accepts email format data | YES |
| 13 | Password input field. | | A mixture of numeric, text and symbols. | YES |
| 14 | Graphs button. The system should display page with user records graphics. | | Accepts click | NO |
| 15 | Diary button. The system should display page with a form that user can provide a personal record for the day. | Accepts click | | YES |
| 16 | Calculator button .The system should display calculator page. | Accepts click | | YES |
| 17 | Account button .The system should display user account page. | Accepts click | | YES |
| 18 | Selectable input for user’s current health status. | Radio Group selection | | YES |
| 19 | Selectable input for user`s next two hours activities level. | Radio Group  Selection | | YES |
| 20 | Food categories buttons. The system should display list with all the food ingredients from the category. | Accepts click | | YES |
| 21 | Add to meal button. The system should add the current selections and user input to the currently created meal. | Accepts click | | YES |
| 22 | `X` button. The system should remove the current selection from the currently created meal. | Accepts click | | YES |
| 23 | Grams input. | Numeric data | | YES |

5.5 User Requirements

|  |  |
| --- | --- |
| Num | Description |
| 1 | The user must register for the website in order to be able to use it |
| 2 | The user must log in with an email address and password. |
| 3 | The user must provide personal information required from the system |
| 4 | The user must update the personal information if any changes occurred |
| 5 | The user should be able to create and store their own meals so that user  can reuse the same meal |
| 6 | The user should be able to provide required info for the day, select a meal  and get suggestions of how much insulin units needs to take. |
| 7 | The user should be able to store personal record such as - sugar  levels, units of insulin to be taken, personal notes. |
| 8 | The user should be notified when ratio updates are available. |
| 9 | User should be notified when ratio updates are available |
| 10 | The user should be able to see a visualization of the personal records in the form of diagrams by different timeslots of the day. |
| 11 | The user should be able to print personal records and ratio updates history |

5.6 Data Requirements

Criteria Description

|  |  |
| --- | --- |
| User Info | Register new user details(Email, Password, Surname, date)  User personal info(Kilograms ,Date Of Birth, Insulin to Carbs Ratios, Body Type)  Daly user record(Time, Date, Blood sugar levels, Rapid insulin was taken, Notes, Type) |
| Food | Food products that will be used for creating meal(Category, Product Name, Measurement in grams , Carbohydrates ) |
| Meals | Customized meals(User , Meal Name, Products ) |
| Ratios updates | Ratios Updates(User , Breakfast ratio, lunch ratio ,Dinner ratio ,General ratio) |

5.7 Security Requirements

|  |  |  |
| --- | --- | --- |
| Num | Description | Critical |
| 1 | Website should prevent SQL Injection attacks | YES |
| 2 | Correct implementation of authentication | YES |
| 3 | Web site should provide correct session management | YES |
| 4 | Website should prevent Cross-site scripting (XSS) attacks | YES |
| 6 | Website should not have insecure direct object references | YES |
| 7 | Website should validate all redirects and forwards | YES |
| 8 | User passwords should be encrypted before been store to database | YES |

5.8 Web Site Features

The following features have being designed to help systems goals to be achieved. They also help to meet some of the requirements.

5.8.1 Calculate personalized user ratios

The Carbs-to-Insulin ratio represents how many grams of carbohydrate are covered or disposed of by 1 unit of rapid insulin. This ratio depending on an individual`s sensitivity to insulin and can vary according to the time of the day.

The ratio for the user will be calculated from the system only in the case that the user does not know and provides their own ratios. In this case, the system will use the “500 Rule” to calculate the general ratio for the user. The calculated ratio will be used as a starting base number for the all different times of the day (Breakfast, Lunch, Dinner).

The “500 Rule” needs the total daily units of insulin from the user to be able to calculate the ratio. The system will allow the user to provide this information. If the user it is not aware of what should be the total amount of rapid insulin that they should take each day, the system will calculate what that number should be by a well-known formula that uses the weight of the user in kilograms.

Examples:

Example 1 (if the user knows and provides the ratios information for the different time of the day)

For user input {

B=16(1:16)

L=14(1:14)

D=17(1:17)

G=0(Rounded average between the B, L, D) If No user input for General}

In this case, these values will be inputted to the database:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | Breakfast | Lunch | Dinner | General |
| Y | 16 | 14 | 17 | 15 |

In this case, the general ratio will not be used.

Example 2 (if the user does not know their different day time ratios but know the general ratio)

For user input {

B=0(0)

L=0(0)

D=0(0)

G=16(1:16)}

In this case, these values will be inputted to the database:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | Breakfast | Lunch | Dinner | General |
| Y | 16 | 16 | 16 | 16 |

In this case, the general ratio will be used as a base number for all the rest of the ratios.

Example 3 (if the user knows only one or two of their different day time ratios and knows the general ratio)

For user input {

B=0(0)

L=14(1:14)

D=0(0)

G=17(1:17)}

In this case, these values will be inputted to the database:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | Breakfast | Lunch | Dinner | General |
| Y | 17 | 14 | 17 | 17 |

In this case, the provided ratio will be stored as a number taken from user input but all the rest will be the same as the General.

Example 4 (if the user provides only one ratio for specific time of the day)

For user input {

B=0(0)

L=14(1:14)

D=0(0)

G=0(0)}

In this case, these values will be inputted to the database:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | Breakfast | Lunch | Dinner | General |
| Y | 14 | 14 | 14 | 14 |

In this case, all the ratios will be the same.

Example 5(if the user provide no ratio information but provides the total daily insulin units)

For user input {

B=0 (0)

L=0 (0)

D=0(0)

G=0(0)

Total daily insulin units=40

}In this case, these values will be inputted to the database:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | Breakfast | Lunch | Dinner | General |
| Y | 12 | 12 | 12 | 12 |

In this case, all the ratios will be calculated from the system by using “500 rule”.

Example 6(if the user provides no ratio information and no total daily insulin units)

For user input {

B=0 (0)

L=0 (0)

D=0(0)

G=0(0)

Total daily insulin units=0

User Weight in Kilograms=90}

In this case, these values will be inputted to the database:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | Breakfast | Lunch | Dinner | General |
| Y | 10 | 10 | 10 | 10 |
| If user weight=70 | 12 | 12 | 12 | 12 |
| If user weight=130 | 7 | 7 | 7 | 7 |

In this case, all the ratios will be calculated by first calculating the Total daily insulin for the user considering the user weight and then using the “500 rule” to calculate the ratios.

5.8.2 Update user ratios

Updating the user is one of the most important features in the system. The system should be able to adjust the user ratios from the existing record in order to maximise the correctness of the Insulin units suggested for the user. This is important because if the correct amount of units are suggested to the user, the blood sugar levels will be on target two hours after the meal.

Before updates can be available for the user, there should be at least ten records for ten different days for the same timeslots (Breakfast, Lunch, Dinner).This approach should reduce a false recorded input which will not impact on the new updates for the ratios. In other words, if the user messed up when inputting the record for example instead for the current sugar levels the user input is 12 and the real number is 16 that will be a false record. By taking 10 different records to update the ratios and not updating for each individual record, this will reduce the final false record error.

An update button will be implemented which will be red in color if there are updates available and colored in green if there are no updates yet available. The button can be replaced in the future releases when these updates can be done automatically from the system. This will be achieved by setting up timers for the system to check for updates instead of waiting for the user to click the button.

Updating logic is when the system checks the database from the last updated date to see if there is one. If there has been no last update found the system will start from the date that user has the first record inputted. A usable record entry is when the user inputs data for the specific mealtime and including data for the two hours post meal. The system will run through all the different day times (Breakfast, Lunch, Dinner, Random) and will try to find records where the user has inputted usable data for the updates.

Example 1

Breakfast user current sugar insulin taken note

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 23/02/2016 | y | 10 | 13 | xxxxxxxxxxxxxxxxxxxxxxx |
|  |  |  |  |  |

Two hours after

Breakfast user current sugar insulin is taken note

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 23/02/2016 | y | 8 |  | xxxxxxxxxxxxxxxxxxxxxxx |
|  |  |  |  |  |

In this case, the user has a record where for the date 23/02/2016 where before breakfast current sugar levels were 10, the user has taken 13 units of rapid insulin before eating the meal. Two hours later is the time when the carbs are getting consumed from the body and the insulin starts working. Two hours later the sugar levels for the user is supposed to be in the range between 4.9 and 5.9 .As we see in this scenario two hours later the blood sugar level for the user is 8 which means the user will need more insulin, which indicates that the ratio for a breakfast needs to be updated. When the system finds ten records where the sugar levels are not in range two hours after a meal the system indicates to the user that updates are available for the ratios. The system should decide itself which ratios to update for example if there are ten records for breakfast but only 3 for dinner the updates will be available for the user but the system should update only breakfast ratio as they are not enough records for the rest of the timeslots. If there are enough records for an update, the system should take each timeslot individually and update is there is a required amount of records.

For example, the breakfast input and post breakfast input, the system should be able to calculate what the ratio should be for these particular inputs in order for the sugar levels to be in range. After that, the system will take the average between all the recalculated ratios from the previous ten records of the breakfast inputs recorded and then adjust the current user ratio.

5.8.3 High Blood Sugar Correction

The high blood sugar is necessary when the current sugar levels before a meal are off target. For example, sugar level before a meal should be between 4.9 and 5.9, but current sugar level is 10.In this case, there will be additional units of insulin added to the original units that have been calculated from the user ratio. High blood sugar correction can be calculated by the “1,800 rule“. This rule is used to calculate how much additional units of insulin is needed according to his current sugar level.

Example

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| user | current sugar | ratio | Meal carbs | Insulin to be taken |  |
| X | 5.5(on target) | 1:14 | 200 carbs | 10 |  |
| X | 10(off target) | 1:14 | 200 carbs | 12 |  |
|  |  |  |  |  |  |

In the second roll, current sugar is off target and for the same scenario more insulin units needs to be added to the original one. This is where the high blood sugar correction comes into play and calculates the addition units of insulin.

5.8.4 Creating And Saving Meal

This feature will allow the user to select products from the existing food database and put together a meal. When a user creates the meal, which is simply achieved by selecting a different product and inputting the weight for the product to be consumed the system will be able to determine how much carbohydrates this meal contains. Knowing a number of carbs is important when calculating the insulin units to be taken. When the user creates a meal they should have two options to calculate the insulin units or to save the meal. If the user decides to save that meal, they should be able to name the meal and have the option of using that saved meal again by selecting the meal by its name from the user’s personal meal list.

5.8.5 Graphs Representations

The idea of this feature is to display all the user records by time in graphs. There should be four graphs. One graph for each time of the day (breakfast, lunch, dinner, before bed).Each graph should display the blood sugar levels for each date that was recorded. With this approach, the user will be able to easily determine if there are constant high blood sugar levels and take action.

5.9 Web Page

The following paragraphs will outline the idea of the design of the structure and location of all options that the site will provide.

5.9.1 Sign-In Page

This page will be the first page when the site loads. The user will have a choice between signing in if does not have an account or click on the log in button if the user has an account already. The user will input an email address, name and a password which will be needed to be validated before user registration is successfully completed.

5.9.2 Log In Page

This will be the page where the user can input email address and password and login to the system. At this point if the login is successful the system should create sessions to keep user information on the client site.

5.9.3 Diary Page

This will be the first displayed page after the user logs in. The page will contain a form that allows the user to input records for the current day. The user will be able to input a single result, for example, only records for the breakfast time or it will be able to input the records for the entire day.

5.9.4 Account page

The account page will be the page that after a new registered user needs to visit, in order to provide the personal information that the system needs. The form with user inputs such as kilograms, date of birth, body type , total daily units insulin and ratios need to be submitted after the user provides the information. Not all of the input fields will be required from the user, some of the fields can be left empty. However if the user already has provided this information, the information will be displayed when the user revisits the page. If the user needs to update some of the information they will simply type the new information into the correct field and after submitting the information this will be changed in the database.

5.9.5 Calculator

This page will be where the user can create meals by selecting products from the database. The user will be able to save meals for further use. User saved meals will be displayed under section MyMeals. When calculating the units of insulin the user will be allowed to input the current sugar levels, health status at that moment and the next two hours activities. After inputting this information the system will be able to predict and display how many units of insulin the user needs for this particular situation.

5.9.6 Graph Page

This displays a graphical representation of the user records.

5.9.7Reports Page

Reports will be the page where the user can choose to print out the

ratios updates history or the record sorted by dates.

6 SECURITY AND TESTING

6.1 Outline

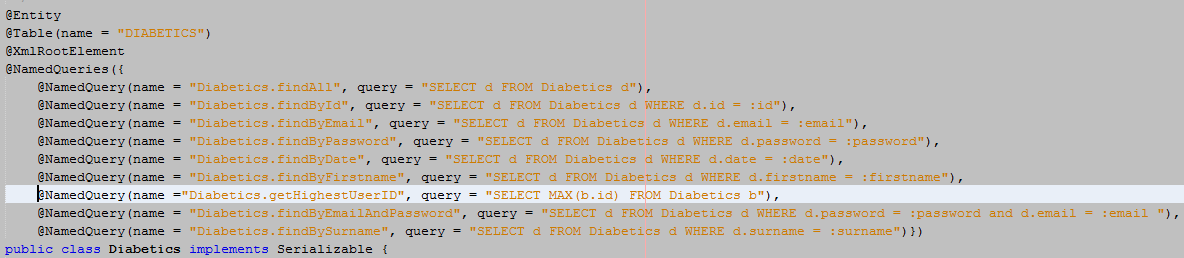
This chapter will outline some of the approaches and techniques that are used on the website in order to improve security.

6.2 Injection

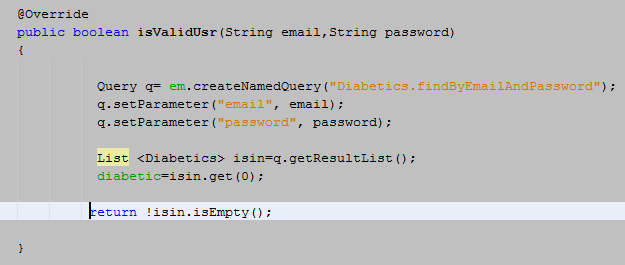
SQL injection is a technique where malicious users can inject SQL commands into an SQL statement, via web page input.

Injected SQL commands can alter SQL statement and compromise the security of a web application. (w3schools.com, 2016)

The technique used in the website is using entity classes and prepared statements.



**Figure 6.2.1-An example of prepared statements in Entity class that connects to the Users table.**



**Figure 6.2.2-An example session bean**

Into the figure 6.2.2 example of a method in session bean which uses entity manager to connect to the entity class. The method invokes named query with set parameters and return true or false if a user with such a credentials exists in the database.

With this technique, the original statement is not derived from the external input which automatically means that SQL injection cannot occur. Prepare statement in entity class also improves efficiency as they need to be compiled only once. However, not all the optimisation can be done at compile time as this might depend on the specific values of the parameters.

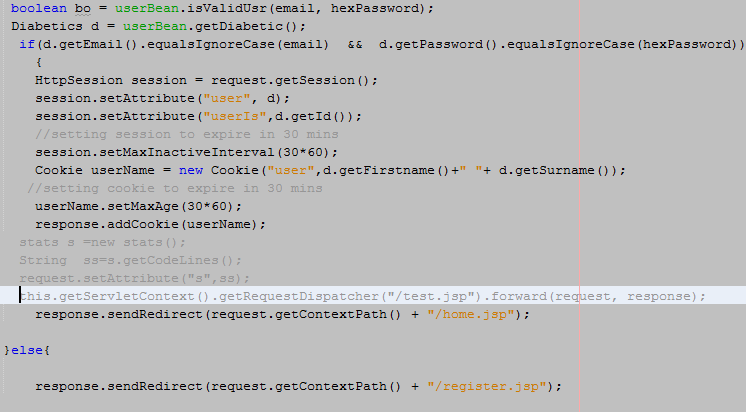
Example from my website of how SQL injection cannot be performed.

If the user goes to the login page and inserts a valid username and inserts the password “or ’1’=’1’ “ , an error message will be displayed to the user stating that the password is invalid.

6.3 Broken Authentication & Session Management

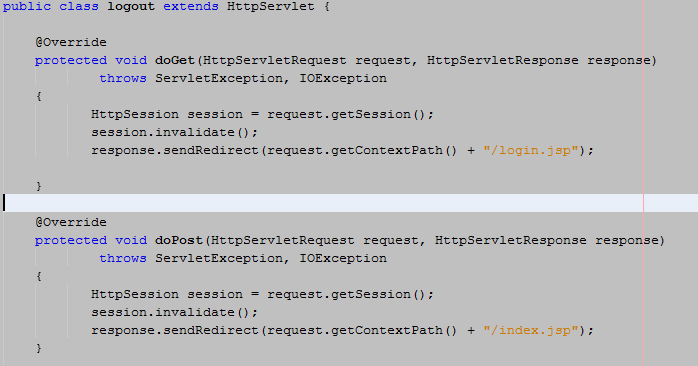
When authentication functions related to the application are NOT implemented correctly which will allow hackers to compromise passwords or session ID's or to exploit other implementation flaws using other users credentials (tutorialspoint.com, 2016).

The first technique used is setting up time for the sessions to be time-out, this gives to the user a set amount of time that the system can be used before the user gets logged out .This approach prevents unauthorised access to the user content. For example, if the user uses a public computer and forgets to logout of the system or forgets to close the browser and leaves the computer, a person using the computer after them could gain access to the user content page as the session will still be alive. By destroying the session after some amount of time or when the user logs out this reduces the risk of unauthorised access.



**Figure 6.3.1-An example of creating a session and cookies and setting up a live time of 30 minutes for them**.

Creation of the session is done in the log-in handler servlet after the user login credentials have been checked and the user gets access to the system.



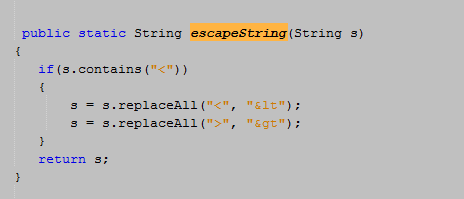
**Figure 6.3.2-** **An example of the servlet methods which invalidate the sessions when the user logs out.**

Session ID is not included in the URL because they can be cached by the browser and they are long, complicated and random numbers. They cannot be easily guessed.

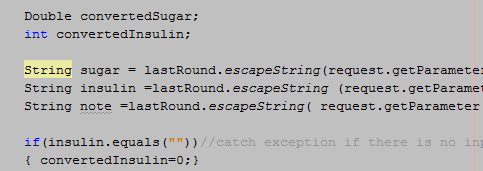
6.4 Cross-Site Scripting(XSS)

Cross-site scripting (XSS) is an injection attack which is carried out on Web applications that accept input but does not properly separate data and executable code before the input is delivered back to a user’s browser. (searchsoftwarequality.techtarget.com, 2016)

To prevent the application from XSS attacks I decided to sanitize the User input and create my own function to escape some HTML character



**Figure 6.4.1-An example of replacing HTML chars.**



**Figure 6.4.2-An example of using the function.**

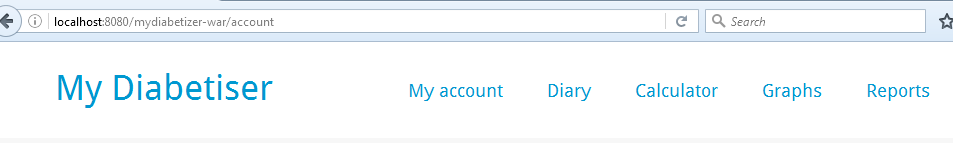
In the example above all the strings that are passed from the request are getting converted before been stored in the database so if the javascript been passed as an input system will remove the HTML chars and will store the input as a simple text. If this text is needed to be displayed it will be as a simple text. In the case of not checking for escape chars and this string is invoked from the database instead of displaying, the system will run a javascript which could cost a lot of the damage, data loss, information leaks .

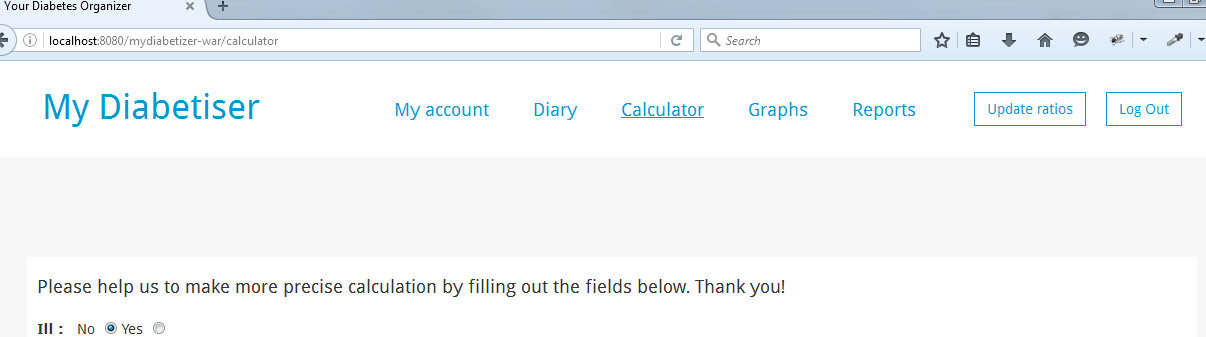
6.5 Unsecure Direct Object References

Insecure Direct Object References allow attackers to bypass authorisation and access resources directly by modifying the value of a parameter used to directly point to an object. Such resources can be database entries belonging to other users, files in the system, and more. This is caused by the fact that the application takes user supplied input and uses it to retrieve an object without performing sufficient authorization checks. (Owasp.org , 2014)

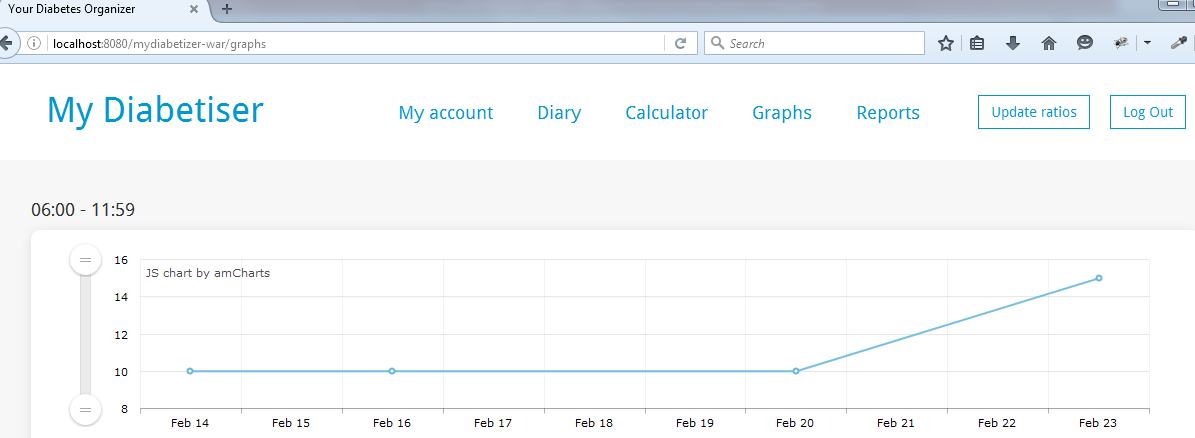
In order to prevent attackers from trying to gain access to other user accounts, no parameters were passed through the URL. Instead passing information in the URL JSON objects are used to transfer the data.

With this approach, no object reference can be modified because they will never be visible to the user.





**Figure 6.5.1-Clean URL example**



**Figure 6.5.2-Clean URL example 2**

No matter what page you try to go to all the URL are clean no parameters going through URL.

6.6 Unvalidated Redirect and Forwards

Unvalidated redirects and forwards are possible when a web application accepts untrusted input that could cause the web application to redirect the request to a URL contained within the untrusted input. By modifying untrusted URL input to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials. Because the server name in the modified link is identical to the original site, phishing attempts may have a more trustworthy appearance. Unvalidated redirect and forward attacks can also be used to maliciously craft a URL that would pass the application’s access control check and then forward the attacker to privileged functions that they would normally not be able to access. (owasp.org, 17 March 2016)

In a matter of security techniques, redirects are avoided. However if redirects are used, SAFE URL REDIRECTS and “Pretty URLs” are used. Also, GET Methods are avoided. Another technique that is used is mapping all the servlets which handle the requests and redirects. When mapping the servlets their names are changed so the actual names of the servlets classes are hidden from the URL.

Examples:

<servlet-mapping>

<servlet-name>AccountHandler</servlet-name>

<url-pattern>/account</url-pattern>

</servlet-mapping><servlet-mapping>

<servlet-name>GraphHandler</servlet-name>

<url-pattern>/graphs</url-pattern>

</servlet-mapping>

The displayed text in the URL is account but the servlet class who actually handle the requests and redirects is AccountHandler.

6.7 Passwords

To improve the user’s security all the passwords are encrypted before being stored in the database in case the database gets compromised. To achieve strong and good encryption passwords are triple-encrypted. When new user registers, the password that the user provides gets encrypted once then to the password is concatenated with the date of the register which is also encrypted with a different algorithm. After the date and original password are encrypted and concatenated the string goes into a third algorithm which loops over the string 2000 times.

private String hexPassword;

public String getHexPassword() {

return hexPassword; {

try {

this.hexPassword=new FirstRoundEncryption().start (this.password);

this.hexPassword+=new TokenEncryption().start ();

for (int i=0;i<2000;i++)

{

this.hexPassword=new lastRound().start(this.hexPassword);

}

}

The example above is where the password is getting encrypted .The first line in the try block invokes a method that encrypts the original password and returns it back. The second line does the same thing but with a different encryption algorithm and instead of the password encrypt the date of the user registered to the system. A for loop runs 2000 times and encrypts the entire date and password.

Public class FirstRoundEncryption

{

public String start(String password)throws Exception

{

MessageDigest md = MessageDigest.getInstance("MD5");

md.update(password.getBytes());

byte byteData[] = md.digest();

StringBuffer sb = new StringBuffer();

for (int i = 0; i < byteData.length; i++) {

sb.append (Integer.toString ((byteData[i] & 0xff) + 0x100, 16).substring (1));

public class TokenEncryption

{

Example of encrypting original password with ("MD5") algorithm

public String start()throws Exception

{

DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd");

Date date = new Date();

String password=dateFormat.format(date);

MessageDigest md = MessageDigest.getInstance("SHA-256");

md.update(password.getBytes());

byte byteData[] = md.digest();

StringBuffer sb = new StringBuffer();

for (int i = 0; i < byteData.length; i++)

{

sb.append(Integer.toString((byteData[i] & 0xff) + 0x100, 16).substring(1));

}

return sb.toString ();

}

Example of encrypting date of registration with ("SHA-256") algorithm

public class lastRound

{

public String start(String password)throws Exception

{

MessageDigest md = MessageDigest.getInstance("SHA-512");

md.update(password.getBytes());

byte byteData[] = md.digest();

StringBuffer sb = new StringBuffer()

for (int i = 0; i < byteData.length; i++)

{

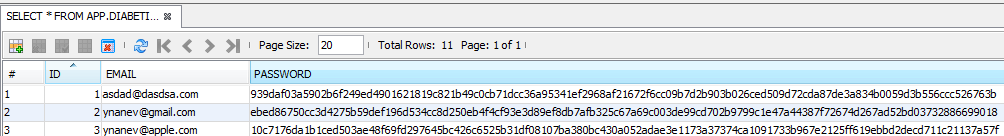
sb.append(Integer.toString((byteData[i] & 0xff) + 0x100, 16).substring(1));

}

return sb.toString();

}

Example of encrypting the combined encrypted original password and concatenated to it encrypted date of register using ("SHA-512") algorithm



**Figure 6.7.1** **Example of stored encrypted users passwords**

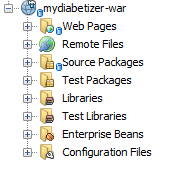
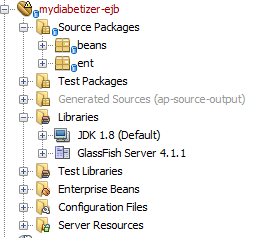
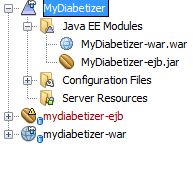
7 IMPLEMENTATION AND TESTING

7.1 Outline

This chapter will cover some actual steps of creating the website. For better understanding their samples of the code and screenshots will be shown. Most aspects of the site will be explained.

7.2 Creating Enterprise Application

At the beginning, the creation of an Enterprise Application with the correct corresponding modules was necessary for the website were established. There are two main components that are in the project, WAR and EJB.



**Figure 7.2.1- Project components break down.**

The EJB module contains the Java Beans and Entity classes. The module also contains the persistence class which makes the connection with the database.

The WAR module contains all the web pages, CSS, javascript, servlets, and java classes).

7.3 Web Pages

The CSS, the header, and the footer were created first. They are the pages included in every page.



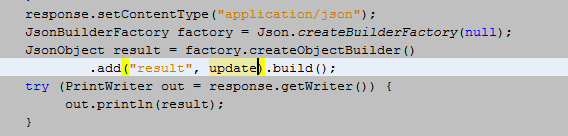
**Figure 7.3.1 - Header includes files**

There is a part of the header where the CSS files are included. The style.CSS is the one that I have created my own.



**Figure 7.3.2-** **Header JavaScript**

The purpose of the JavaScript in the header is that the colour of the Update Ratios button changes if there are updates available for the user. If there are updates the button is colored in red, after user clicks on the button ratios are getting updated and the button gets paint in green. If the button is green that means that all the ratios are updated and even if the user clicks on the button nothing will happen. The system checks for updates on every entry in the system from the user.

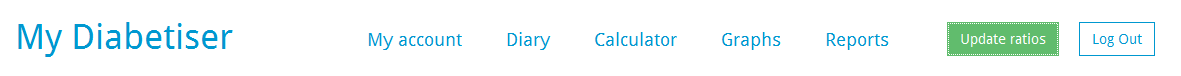


**Figure 7.3.3-** **Return update value from servlet**

This is the part of the method where after checking out the database returns the update status to the header.

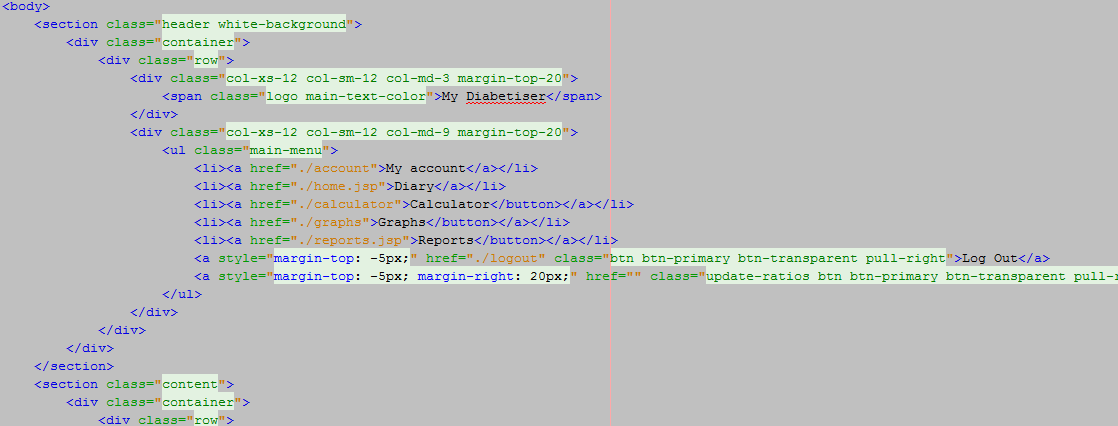


**Figure 7.3.4 -** **Available updates.**



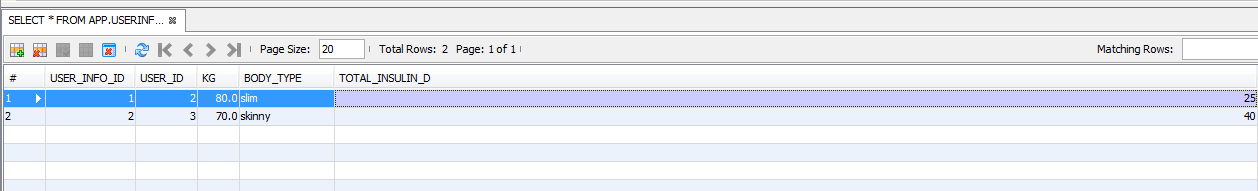
**Figure 7.3.5 -** **Up to date**

The last part of the header includes the simple HTML with the buttons and the corresponding links.

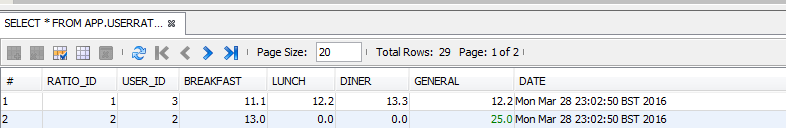


**Figure 7.3.6-** **HTML header links**.

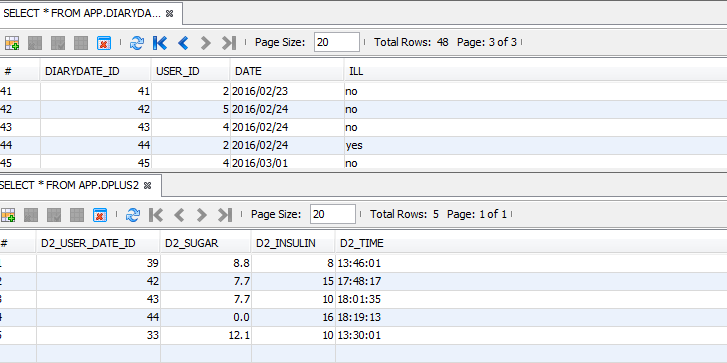
After finishing with the header a couple of other web pages were created (account.jsp, home.jsp,). Those were the pages that were able to be created without having to interact with the database at this point. All the rest of the pages was developed after the implementation of the database as some of the content displayed on this web pages had to be derived from the database. However, after the database was created these two pages were connected to the servlets and database. The account.jsp page is the one that collects personal information as an input from the user and stores it in the database for further use. The home.JSP page is the Diary of the user. User’s input their own daily records and they are stored in the database .The information from both those pages is critically important as it has a significant impact with the insulin units suggestions and updating the user ratios.



**Figure 7.3.7 Example of stored user information from account page.**

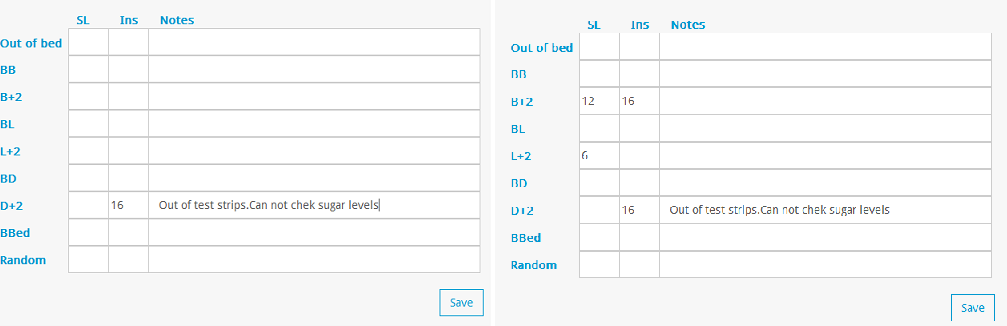


**Figure 7.3.8 Example of stored user ratios from account page.**



**Figure 7.3.9 Example records stored from the home.JSP(Diary page)**

In figure 7.3.9 user number 2 has records for two different dates. For the date that DIARYDATE\_ID is 44, there is matching record into the DPLUS2 table which comes from an input for two hours after dinner. This user has provided no information about the current sugar levels and that is why 0.0 has been stored. The units of insulin that user has taken are 16.

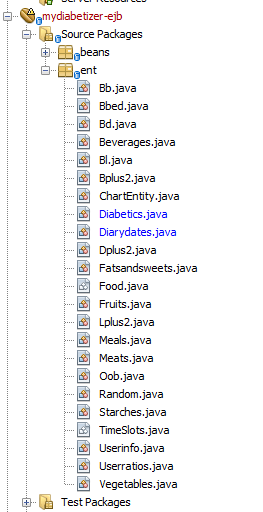
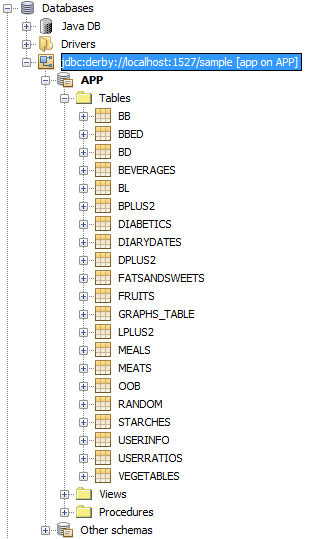


**Figure 7.3.9 Example single and multiple user inputs in the home.JSP(Diary page)**

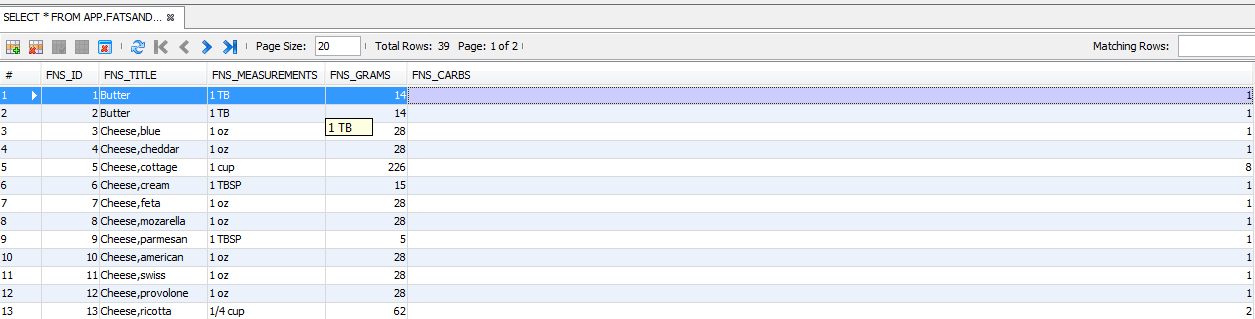
In the example above when the user provided multiple inputs, all of the different rows with inputs are going to the different time database tables, but with the same DIARYDATE\_ID which is unique for this user and the current date. If the user gives a single result and then later makes another input for a different field there will be no DIARYDATE\_ID created, it will use the same one from the previous input for that day.

7.4 Database and entity classes

Next step was the implementation of the database tables and the corresponding entity classes. The database was initially being created using for every table created there is a corresponding entity class which allows the tables data to be handled as objects.



**Figure 7.4.1-** **Database tables and the corresponding entity classes**

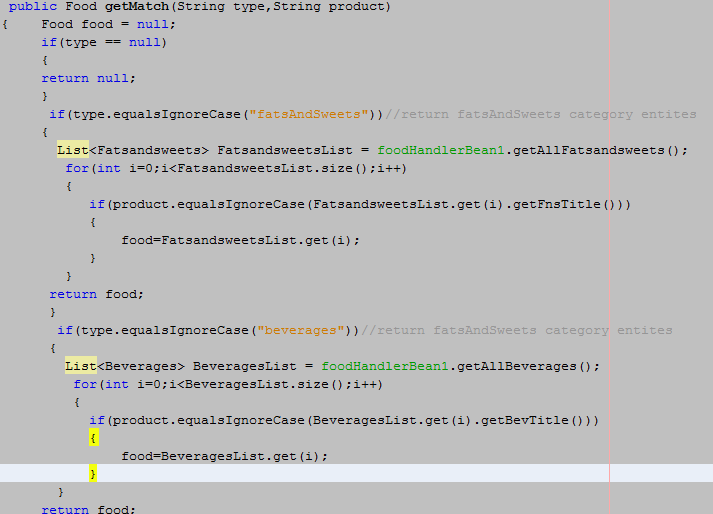


**Figure 7.4.2-**  **Example of the fatsAndSweets table.**

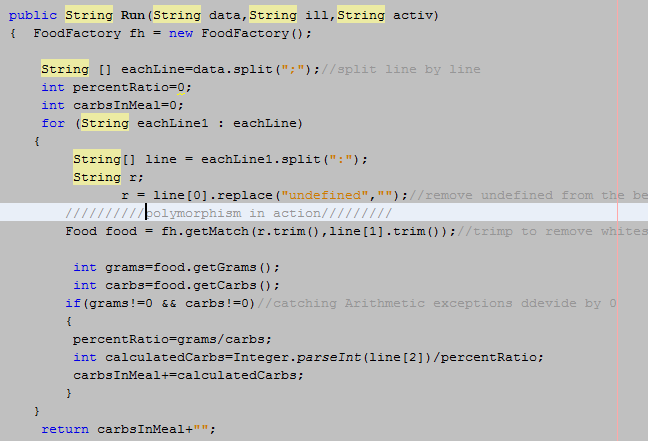


**Figure 7.4.3-** **Example of the diarydates corresponding entity class.**

However, all the food category entity classes are extended from the Food.java abstract class as it allows polymorphism to be used. Factory method has been built to handle which category needs to be retrieved from the database. This is important because there are some interactions that required content to be changed dynamically.The same methodology is used with creating the timeslots database tables and the corresponding entity classes.

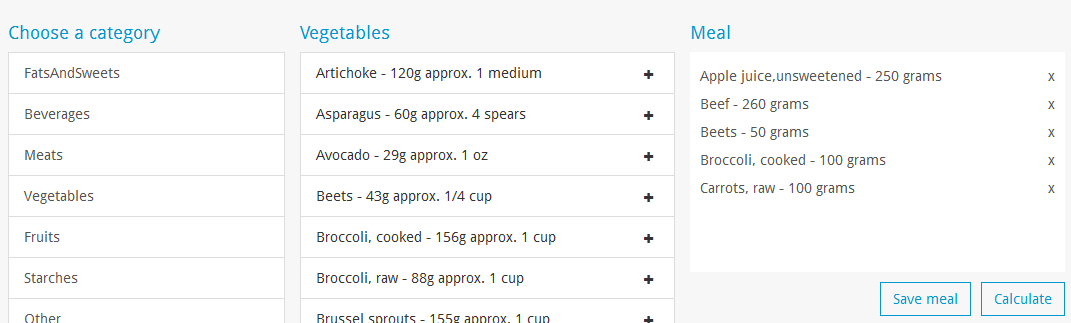


**Figure 7.4.4 -**  **Food factory method**



**Figure 7.4.5**  **Polymorphism and factory method in action**.

Returns for calculation of the total carbs from the selected user meal. This is done by retrieving the information for each product from the database and calculates how much carbs are in the user input for that product. For example, Apple juice, unsweetened they are 28 grams of carbohydrates in 248 grams of juice .If the user inputs 500 grams Apple juice, unsweetened the system will calculate 56 grams of carbs.



**Figure 7.4.6** **Example of user interaction when Figure 7.4.4 And Figure 7.4.5 code examples are triggered (on Calculate click).**

7.5 Sign-Up and Sign-In

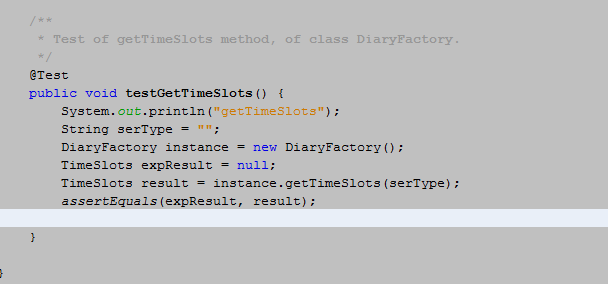
They are two .JSP pages created to deal with user’s credentials forms submissions. All the data is actually handled on the server side no matter if there is new user registration or just matching the existing user. There are password and session handling tasks performed as well at this stage which are explained in sections 6.6 -Passwords and 6.3- Broken Authentication & Session Management.

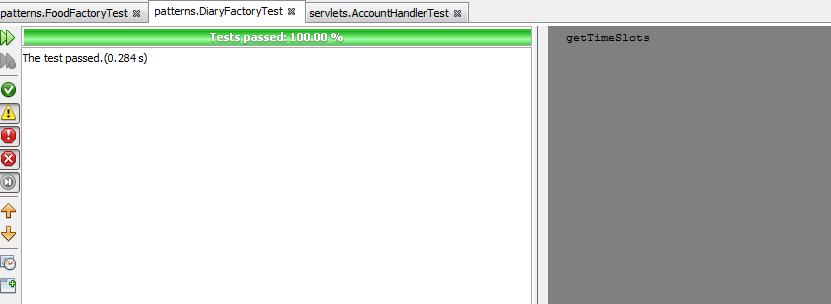
7.6 Graphs and Reports

They are four graphs created in the graphs page. Each graph representing the different timeslot of the. The general code was created to fetch the data from the database and pass it to the required graph.

7.7 Automated Testing

JUnit classes were created to run the test against some of the java classes. The test is over the entire classes rather than chunks of the code.

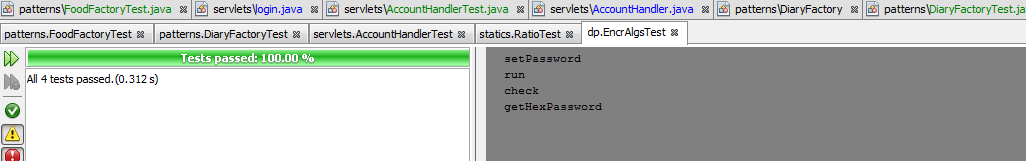


**Figure 7.7.1 –Example of test method in test class testing DiaryFactory.java**

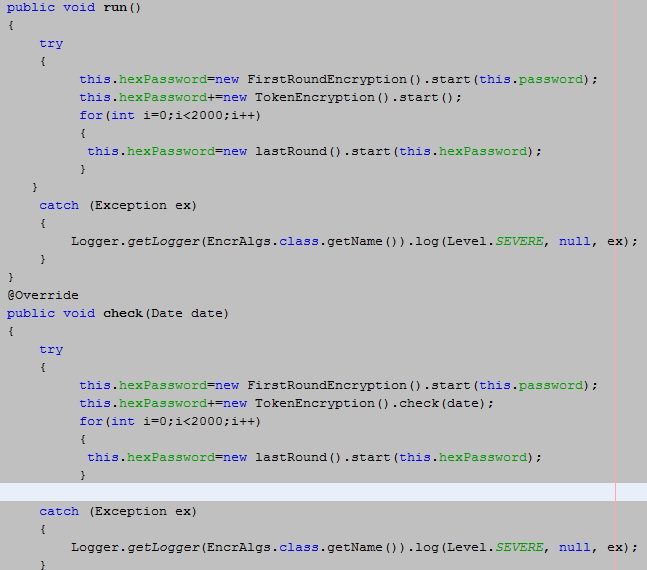
**Figure 7.7.2-Result of running the test method in figure 7.7.1**



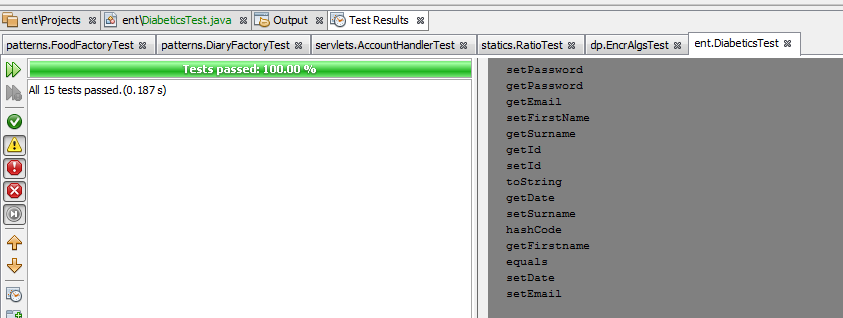
**Figure 7.7.3 –Example of test methods in test class, testing EncrAlgs.java**



**Figure 7.7.4-Result of running the test method in figure 7.7.3**

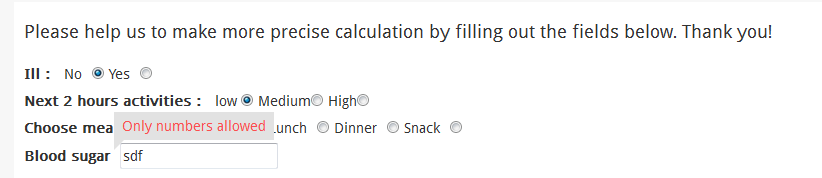


**Figure 7.7.5 –Code example of the tested methods in EncrAlgs.java**

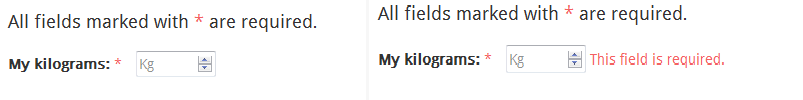


**Figure 7.7.6-Result of testing entity class(Diabetics.java)**

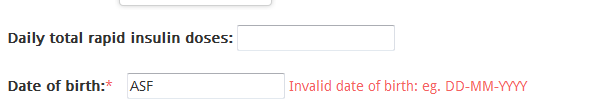
7.8 Validation Testing



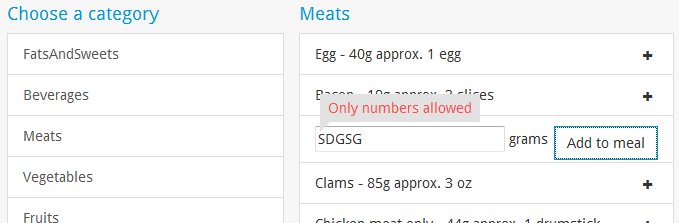
**Figure 7.8.1-Example of the system accept only numerical input.**



**Figure 7.8.2-Example of the system require input for a mandatory field.**



**Figure 7.8.3-Example of the system require input for a date in a specific format.**

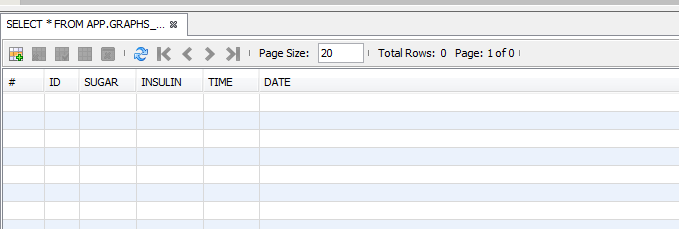


**Figure 7.8.4- Example of the system accept only numerical input.**

7.9 Challenges

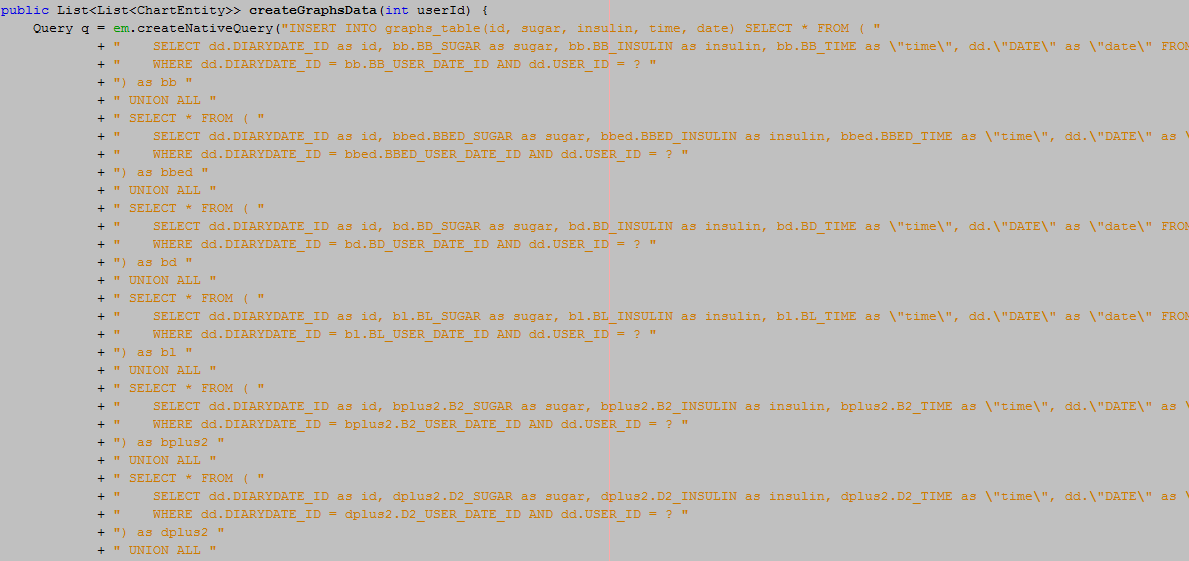
7.9.1 Graphs challenge

One of the challenges experienced during development process was how to retrieve the specific data for the graphics on the graph page. The problem was that each graph requires data in a specific six-hour time slot that needed to be retrieved for a particular user. In each time slot, there were potential results from 1 to many for the same date. They are nine tables in the database which keep the data for the different time slots, the date of the actual result is kept in a different table. However, pulling all the data into a nine different list and looping through each of them for each graph was extremely inefficient.Solution to the problem was to create an additional table in which all data needed from all different tables was joined to that table. Then the data from that table was able to be pulled in the desired range of time, there was no more use of loops.

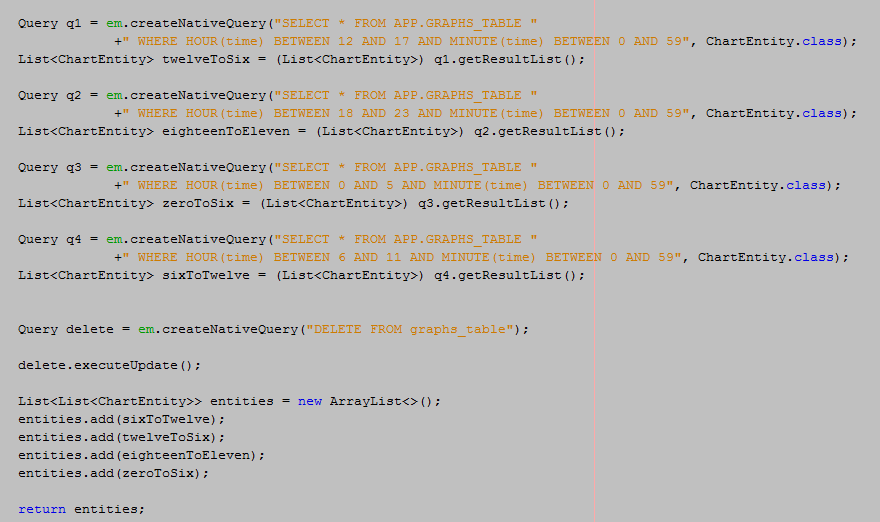


**Figure 7.9.1.1- Example of the additionally created table.**

The table is empty because after the data is combined from the other tables is handled straight away and the table is cleared because there is no need of that data until user logs out and reenter the site again.



**Figure 7.9.1.2- Example of the part of the SQL query which handles pulling the data and inserting it into the additional table .**



**Figure 7.9.1.3-Example of SQL queries which pulling the data from the table by selecting the time range.**

The data is stored in “list of lists” of type CharEntity where CharEntity is the entity class corresponding to the additional table. The data is passed to the particular server which resending the data in JSON format to the corresponding JSP. The page where JavaScript handles the manipulation of the data.

Coming with a solution for this problem was challenging as much as implementing the solution.

7.9.2 Domain Name Issue

At the start of the implementation of the project a host service and a domain name (“MyDiabetizer”) were purchased. Three weeks later the domain name was starting to be used as the project was deploying on the remote server at that stage. A week after the domain name started to be used an email was received from a man who claims that the word “diabetizer” is a reserved word and cannot be used in the domain name as that man has the rights. The email was saying that if the domain is not taken down in three days I will be prosecuted. However, the company that the domain was purchased from was also the company that providing the hosting of the project. Turns out that this company actually notify that man about the usage of the reserved word in my domain name after they sold it to me. After that happen decision was made to take down the site released the host and the domain name and save the hassle by deploying on localhost as it was not that critical for the sake of the project. The interesting part is that two days after I released the domain name the same domain was up and running and is redirecting to this man’s website. However, the diabetizer word is really preserved by this man. The name of the project had to be changed from “MyDiabetizer” to “MyDiabetise”.

8 Conclusion

To get to the conclusion of the overall success of the project we need to go back and examined the objectives set at the beginning have been satisfied.

* Design and implementation of the website interface.
* Handling users information and records by following the requirements
* Represent this record in the graphical representation.
* Calculates and updates user ratios.
* Provided food database
* Creating of users own meals
* Security approaches followed by the requirements.

While the site has being designed to meet all of these objectives, it cannot be said that the final product has fully completed. It was managed to meet the majority of the objectives, saying that the project, in general, has completed will not be justified.

Due to a time constraint, the Android Mobile Application has not been fully designed and been frozen at the research stage. Also, the algorithm that actually doing the calculations for the ratio updates needs to be extended and modified to produce more accrued results.

In a potential future fully completing all parts the project, there are some aspects that need to be included or complete. The Android mobile application interface needs to be developed. The functionality and the some requirements can be used from the existing one for the website as the mobile application should have similar to the same functionality. The feature that extracts the date from the user glucometer and stores the results automatically needs to design and add it to the website. A one of the most important aspects is that the updating the ratio algorithm needs to be improved to achieve higher levels of correctness.

The experience and the knowledge that I gained from this project are extremely good. My programming knowledge had improved significantly, also the ability to follow the correct process of developing the system. Overall the way I feel after completing this task is that I have succeeded, as the knowledge and the skills I gained from it are priceless

9 References

((wikipedia.org, 5 April 2016) *JavaScript definition* [online],

<https://en.wikipedia.org/wiki/JavaScript>*[accessed 02th Apr 2016])*

((netbeans.org,2016) *About Net Beans* [online],

<https://netbeans.org/about/>*[accessed 02th Apr 2016]*)

(wikipedia.org, 6 April 2016) *GitHub explanation* [online],

<https://en.wikipedia.org/wiki/GitHub>*[accessed 02th Apr 2016]*)

((wikipedia.org,5 November 2015) *About GlassFish Server*  [online], [online],<https://en.wikipedia.org/wiki/GlassFish>*[accessed 02th Apr 2016]*)

(tutorialspoint.com,2016)*Broken authentication definition and testing [*online*]*

<http://www.tutorialspoint.com/security_testing/testing_broken_authentication.htm>*[accessed 02th Apr 2016]*)

((w3schools.com, 2016) *SQL injection definition [*online*]*

<http://www.w3schools.com/sql/sql_injection.asp>*[accessed 02th Apr 2016]*)

((searchsoftwarequality.techtarget.com , 2016)*XSS definition [*online*]*

<http://searchsoftwarequality.techtarget.com/definition/cross-site-scripting>*[accessed 02th Apr 2016]*)

(( Owasp.org , 2014) *Insecure Direct Object Ref. [*online*]*

<https://www.owasp.org/index.php/Testing_for_Insecure_Direct_Object_References_(OTG-AUTHZ-004>*[accessed 02th Apr 2016]*)

((owasp.org , 17 March 2016) *Unvalidated redirects and forwards. [*online*]*

<https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet>*[accessed 02th Apr 2016]*)

((dtc.ucsf.edu, 2016) *Diabetes ucsf information. [*online*]*

<http://dtc.ucsf.edu/types-of-diabetes/type2/treatment-of-type-2-diabetes/medications-and-therapies/type-2-insulin-rx/calculating-insulin-dose/#coverage>*[accessed 02th Oct 2015]*)

((niddk.nih.gov, 2014) *Diabetes niddk information. [*online*]*

http://www.niddk.nih.gov/health-information/Pages/default.aspxscs*[accessed 02th Oct 2015]*)

(healthline.com, n/a) *Diabetes ucsf information. [*online*]*

http://www.healthline.com/*[accessed 13th Sep2015]*)