Designs for Fitness Activity Tracker

Prepared by

Group Name: The Great Unwashed

|  |  |
| --- | --- |
| Jillian Cleek | jrc064@shsu.edu |
| Quan Do | qcd002@shsu.edu |
| Arieus Green | alg045@shsu.edu |
| Deantrey Loche | dfl005@shsu.edu |
| Christopher Noack | cmn009@shsu.edu |
| Jay Pruett | jay\_pruett@shsu.edu |
| Nicholette Schaefer | nas017@shsu.edu |
| Robby Simons | rrs018@shsu.edu |
| Colin Williamson | cmw042@shsu.edu |

|  |  |
| --- | --- |
| Instructor: | Dr. David Burris |
| Course: | Software Engineering |
| Lab Section: | *COSC 4319.01* |

**Table of Contents**

**Data Flow Design** 21

Data Flow Design Metrics 23

**Flow Chart Design** 24

Flow Chart Design Metrics 35

**Object Oriented Design** 37

Use Case Scenarios/Timing Diagrams 39

Object Oriented Design Metrics 50

**Jackson Design** 51

Jackson Design Metrics 60

**Database Design Document** 61

Entity Relationship Diagram 62

Normalization 63

Tables and Relationships 66

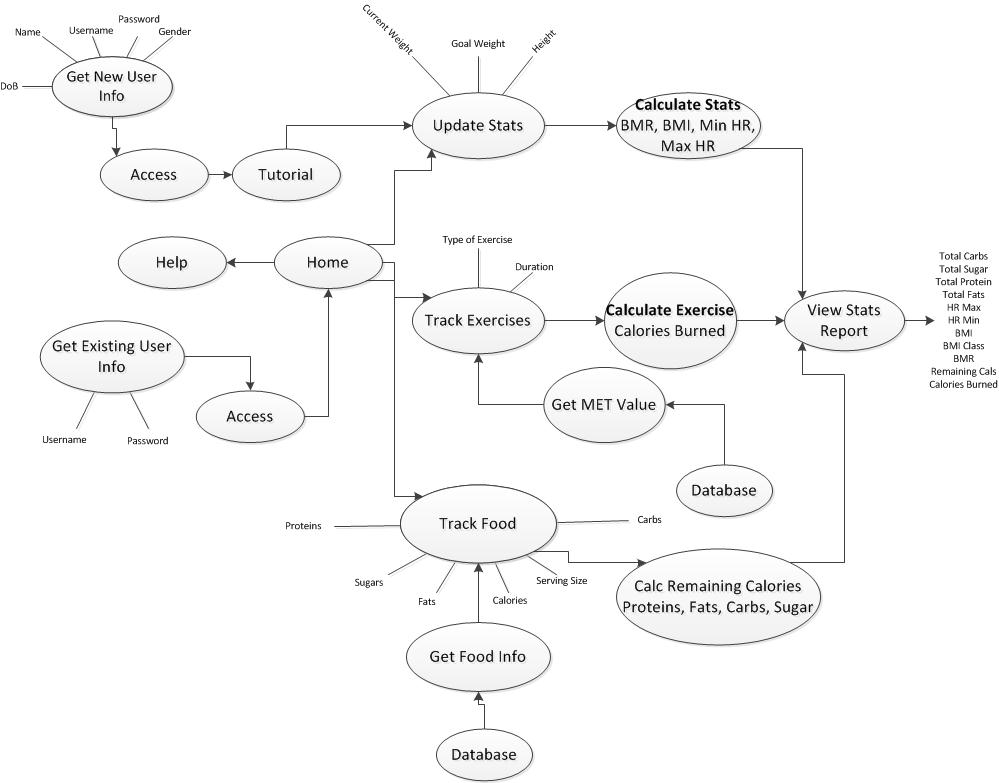
**Server Technologies** 76

**Implementation Plan** 77

**Test Case Document** 78

**Design Selection** 84

**Dataflow Design**



**Dataflow Design Metrics**

**Coupling and Cohesion:**

The goal of Data Flow Diagrams is to have one idea per module, with the modules split into sections of Afferent (Input) data, Central Transforms (functions), and Efferent (Output) data. With this emphasis on the technique, Data Flow Diagrams inherently ensure low coupling and high cohesion when it is created correctly. There is a functional cohesion that comes from the separation of the diagram in the aforementioned three categories—Afferent data, Central Transforms, and Efferent data. All of the elements being processed in the afferent data on the left side of the diagram is necessary to produce the efferent data on the right side of the diagram. All unnecessary components are removed and data is labeled efficiently so the diagram contains the correct level of detail.

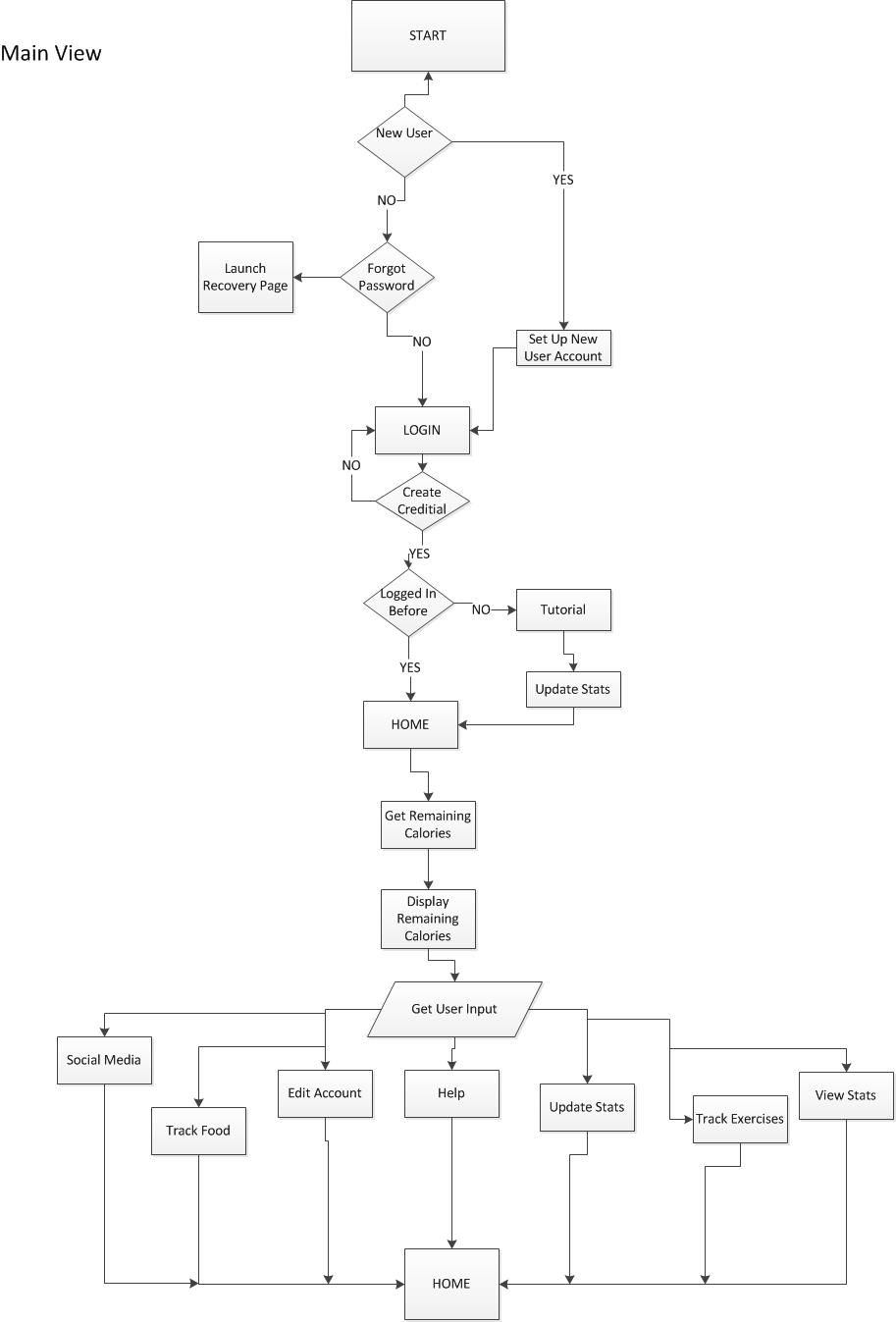
**Fan-In/Fan-Out:**

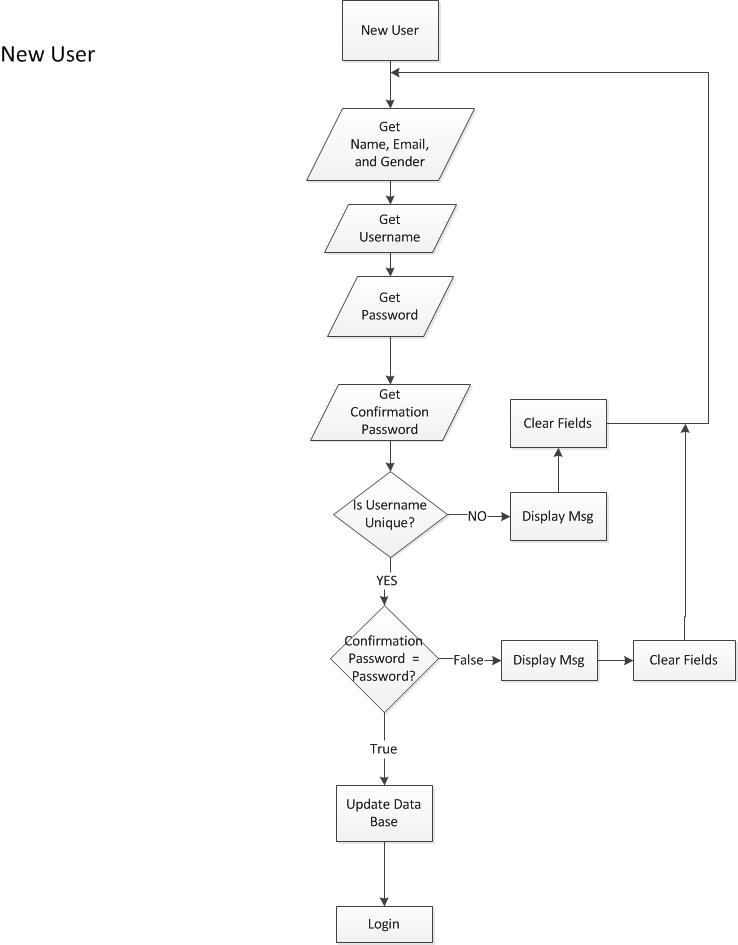
Data Flow Diagrams tend to have a highly factored system with the Afferent data and Central Transforms (the Fan-In) having most of command and control, while the Efferent data (the Fan-Out) has most of the results and details of the design. More accurately, the Afferent data has the raw details required for computing, the Central Transform processes all of the Afferent data, and the Efferent data routes the separate results and raw inputs to the appropriate destination. This leads to a good design that can lead to cost reduction in all stages of specification, design, implementation, maintenance, and modification.

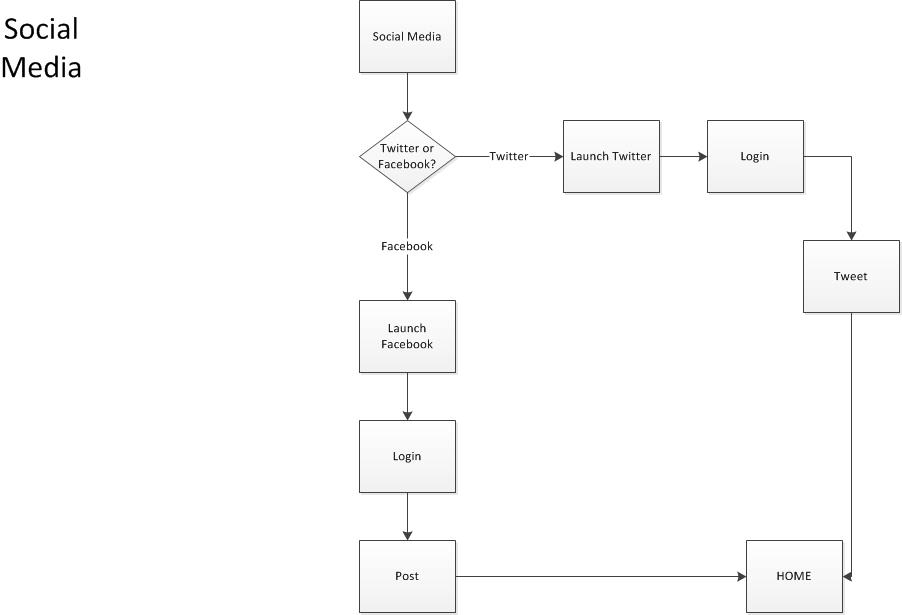
**Graicunas’ Law and Miller’s Law:**

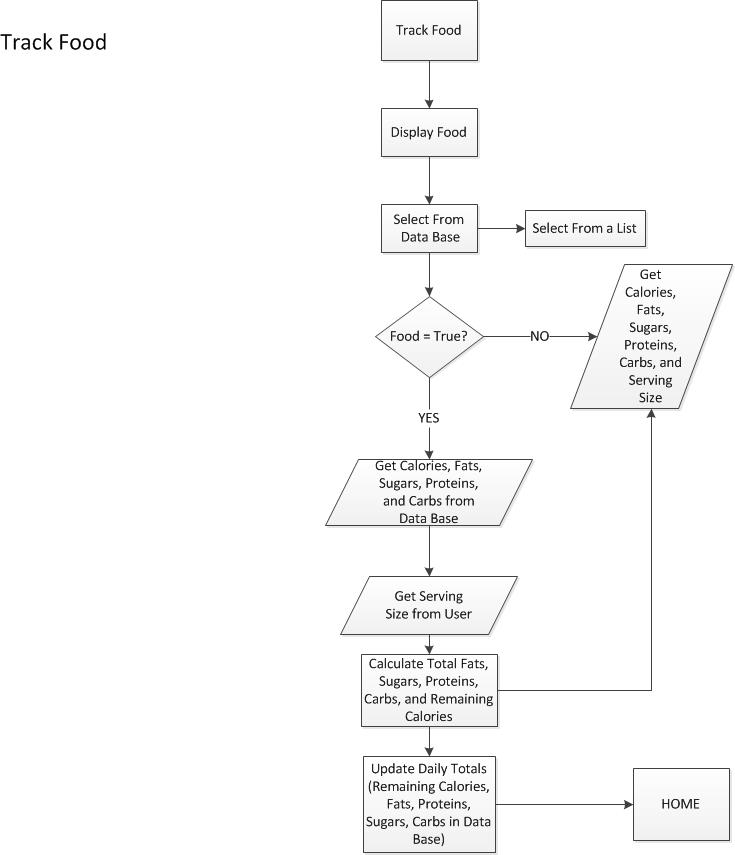
According to Graicunas’ Law, the span of control should not be around four to eight people per module. Miller suggests that mental capacity of a working human brain can handle sever concepts, entities, and objects at one time, plus or minus two. With the way Data Flow is factored, everything can basically be separately implementable and worked on by one person per module. All modules could call another related module directly or indirectly with little problem. Complexity is also reduced by limiting the path from one to another with one path (with the paths detailed in the diagram itself). The Fan-In that Data Flow diagrams have also works with Graicunas’ Law since the Central Transforms are represented as true black boxes. Miller’s Law is satisfied with the highly factored nature of the design criteria, with maximum cohesion and minimal coupling being a key factor from the core of the design.

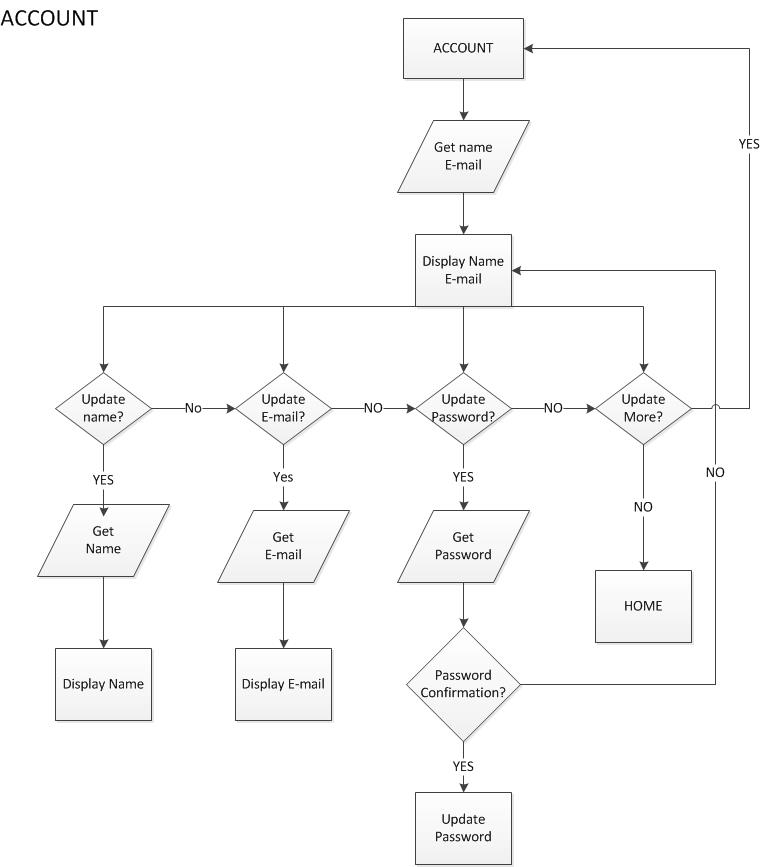
**Flow Chart Design (Functional)**

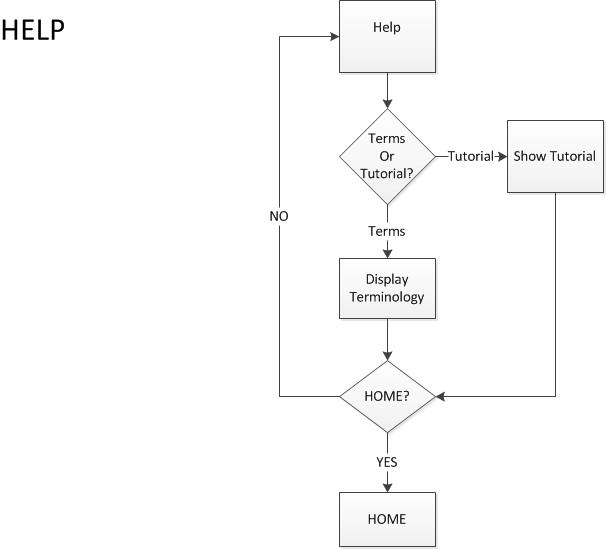


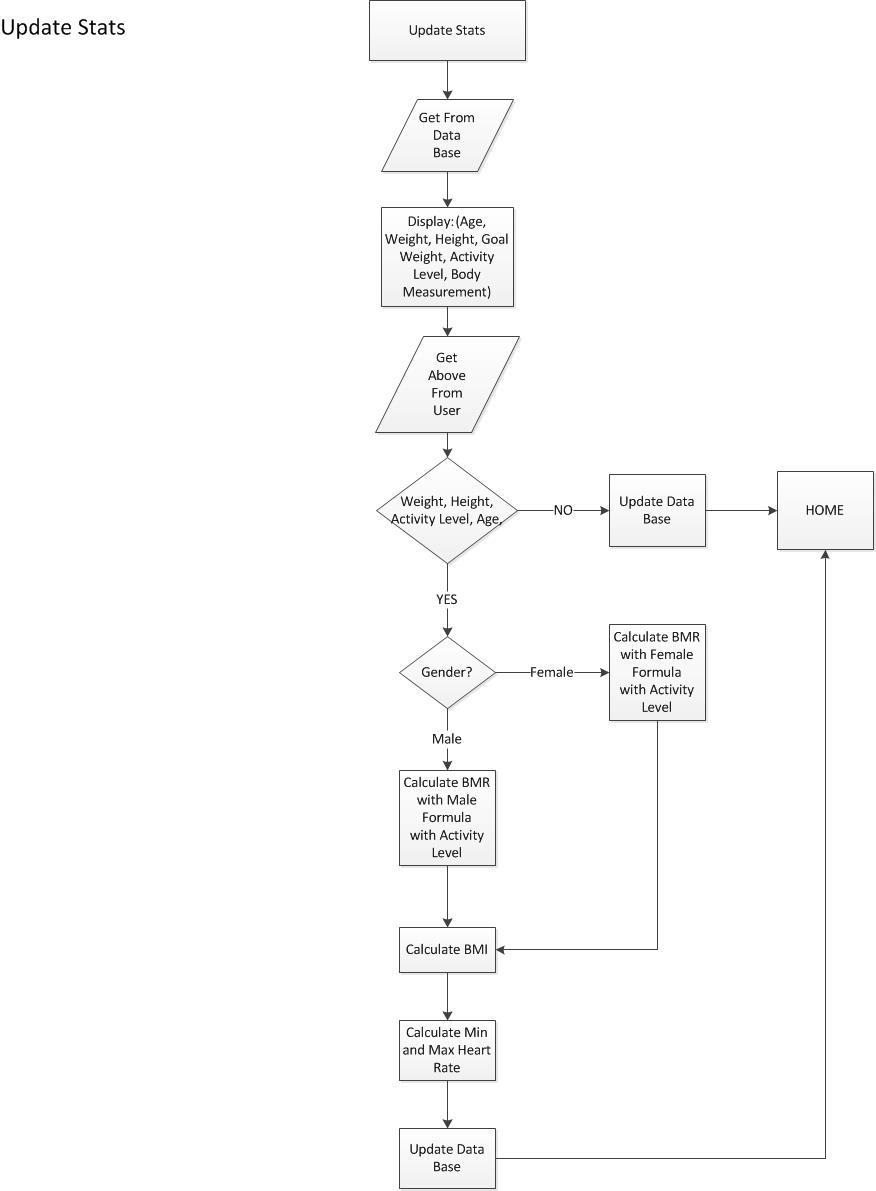


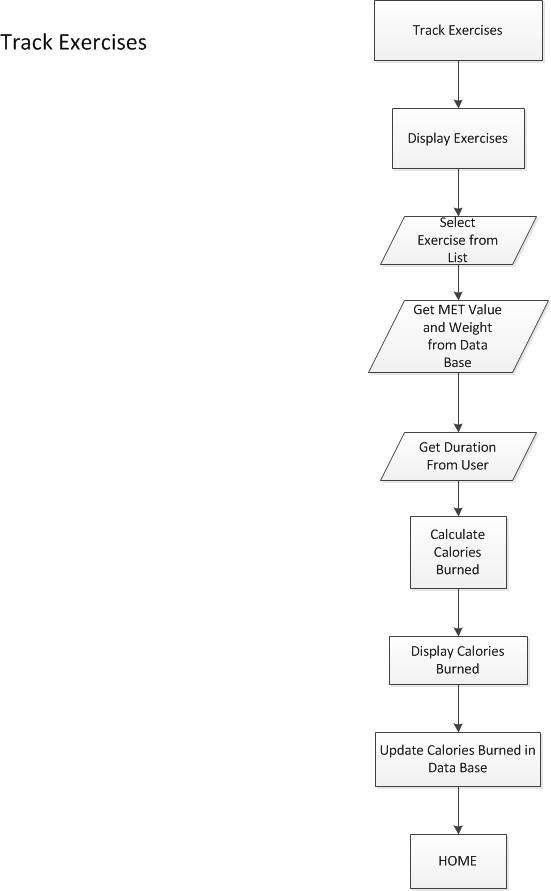


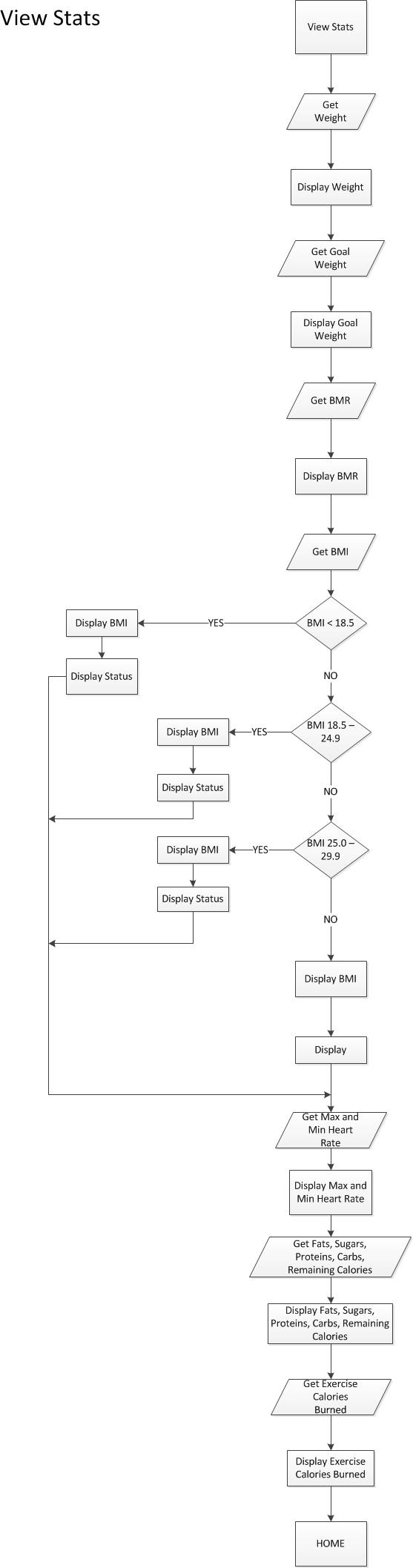


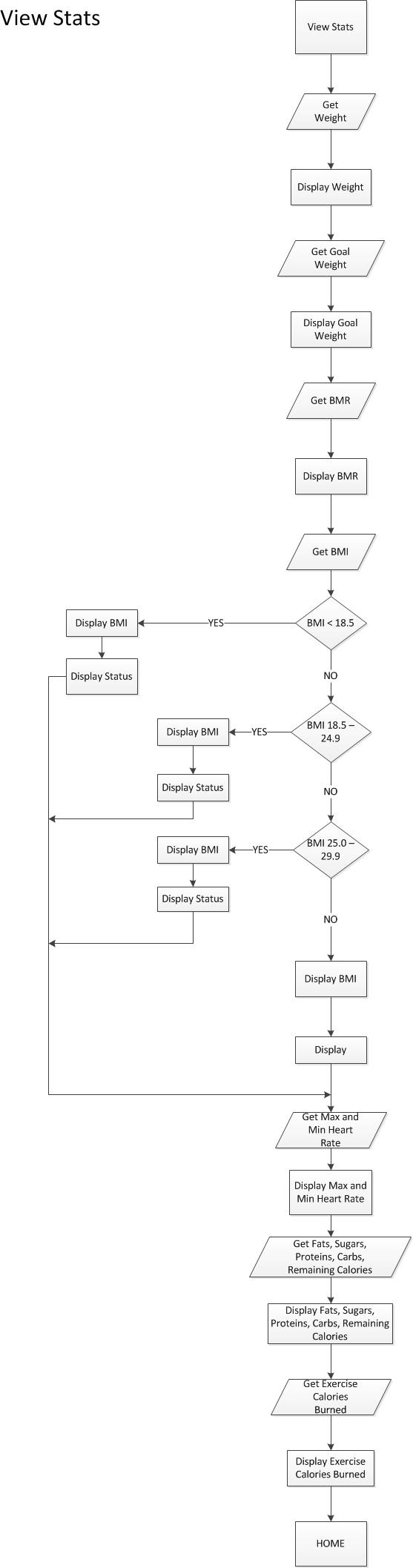












**Flow Chart Metrics**

**Main View**

* This diagram has high coupling with high (Communicational) cohesion which means the design is subject to Miller’s Law.
* Simply because of the many attributes relating to the same get statement, this makes it susceptible to errors.

**Password Recovery**

* The frame work reveals that the design is on track for Graicunas Law which demonstrates control.
* This flow chart guarantee’s Low (Data) coupling and high (Procedural) cohesion.
* Each module is directly related to the other.
* This allows the project to be worked on by multiple members of a group

**Track Food**

* This design infers a high risk of coupling and high (Sequential) cohesion.
* This potentially falls under Miller’s Law, having one module subject to such excessive detail, making it prone to contain errors.
* Leaves opportunity for the breakdown and distribution of work.

**Update Stats**

* This diagram provides low (Data) coupling which requires the sharing of data, and also contain high (Procedural) cohesion followed up by specific calculations.
* These modules however are subject to the fan-in and fan-out approach.

**Track Exercise**

* The frame work design of each module shows the efficiency of low coupling and high (Functional) cohesion.
* All modules directly correlate to the others with respect of data sharing and control.

**View Stats**

* The development of this design shows aspects of low coupling and high (Procedural) cohesion.
* The frame work sets parameters which will simplify the delegation of authority and make decisions more effective.

**Account**

* The process of this module demonstrates low coupling and high (Functional) cohesion.
* No one module depends on the other, it initially falls under Graicunas.
* This particular approach leaves room for the separation of work between project members.

**Help**

* This particular frame work specifies relatively low coupling with high (Functional) cohesion.
* There is no aspect of any module being reused

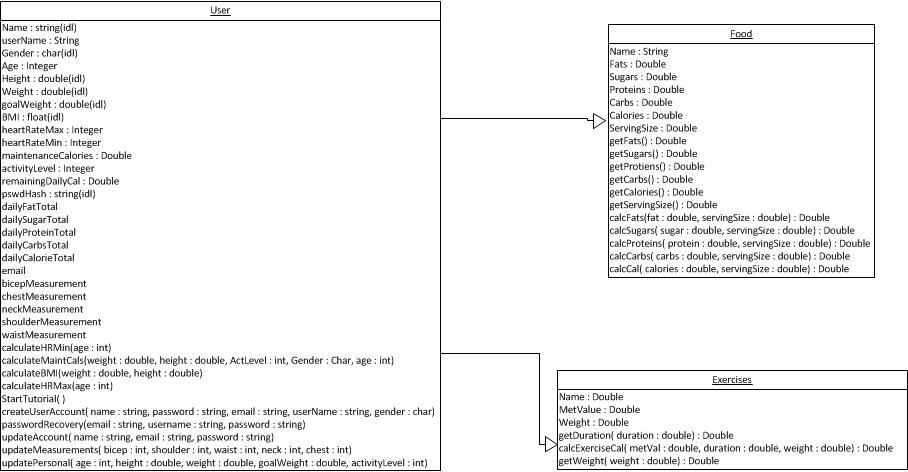
**Social Media**

* The design phase is simple, it does not contain any modules that can be duplicated or reused.
* It is a product of low coupling and high (Functional) cohesion.

**New User**

* This diagram has low coupling and high (Procedural) cohesion.
* This sets in place a certain sequence of execution for the code to follow

**Object Oriented Design**



**Use Case Scenarios/Timing Diagrams**

**Use Case:** Login to Application

**Goal**: To display User Login Screen in order for User to gain access to a login protocol. Allow Users access to the application.

**Scenario:** The User will request to view Login screen. User will be prompted to enter required login credentials. If credentials are correct, access will granted and the User will be directed to the Home page. Otherwise, an error message will be displayed and access will be denied.

*User System*

|  |  |
| --- | --- |
| *Initiates request for login screen* |  |
|  | *Request for Login received* |
|  | *Gathers Login page data* |
|  | *Builds Login page* |
|  | *Delivers Login page* |
| *Display Login page and allow for User input (Username, Password and Button Click)* |  |
| *Password will be hashed and input will be discarded* |  |
|  | *Receives User account information* |
|  | *Sends User account information to the database* |
|  | *Compares input obtained to the information stored in the database. If correct and present, then send success message to server*  *If incorrect or not present, send “login unsuccessful” (error) message to server* |
|  | *If an error message occurs, refresh Login page and prompt the User to try again*  *If a success message occurs, then advance to the default page* |
| *If User sees Login page with an error, he/she will repeat the process until granted access to the application*  *If no error message is displayed, the User will see the default page* |  |

**Use Case:** Create Account

**Goal:** To allow new Users to create an account and gain access to the application.

**Scenario:** The new User selects the “New User” button from the Login screen. The User will be prompted to enter required account information. If the username is not unique an error message will be displayed. Otherwise, no message will be displayed and the User will be redirected to the Login page. The User will then need to Login using the credentials just created.

*User System*

|  |  |
| --- | --- |
| *Initiates request for creation of an account* |  |
|  | *Account creation received* |
|  | *Gathers and sends “Create Account” page information* |
| *The “Create Account” page will be displayed and will allow for User input (required information for a new account will be retrieved)* |  |
| *The password will be hashed* |  |
| *Clicks the submit button and all account information will be sent* |  |
|  | *Account data packet received (username and password)* |
|  | *Compares input obtained to the information stored in the database. If present, then send error message to server .If Username is not unique do not update database*  *If Username is unique, message “successful” sent to server (account creation successful). Update database with new account information* |
|  | *If error message is present, refresh “Create Account” page with error message displayed*  *If account creation is successful redirect user to Login page with success message displayed* |
| *If User sees error message repeat the process until account creation successful*  *If created account is successful User will see Login page with a success message displayed* |  |

**Use Case:** Password Recovery

**Goal:** To allow User access into the application if password has been forgotten.

**Scenario:** The User selects the “Forgot Password” button on the Login page. The User will then be prompted for their Username and email. If there is a match then the User will be allowed to update their password. If there is not a match an error message will be displayed and the User will be redirected appropriately.

*User System*

|  |  |
| --- | --- |
| *Initiates request for password recovery by clicking the “Forgot Password” link* |  |
|  | *Receives password recovery request* |
|  | *Gathers and sends password recovery page information* |
| *The recovery password page will be displayed allowing for User input (Username and Email)* |  |
| *Clicks submit button and Username and Email will be sent* |  |
|  | *Receives information packet from User (Username and Email)* |
|  | *Compares input obtained to the information stored in the database. If present and correct, then send successful message and display new password page*  *If the information packet is not a match, then send error message and allow for User to try again* |
| *If the success message is displayed on the new password page, the User will then be prompted for a new password and continue with the process*  *If an error is encountered, the User will repeat the above process until successful. Once successful User will continue with the process* |  |
| *User enters new password and then clicks submit* |  |
| *New password will then be hashed* |  |
|  | *Receives new password hash* |
|  | *Update database with new password. Redirect to Login page with success message displayed* |
| *User will view Login page with success message then will be allowed to Login using new password* |  |

**Use Case:** First Time Login Tutorial

**Goal:** To provide User assistance in navigating through the application.

**Scenario:** Upon User’s first login he/she will be sent to a tutorial that specifies the general functionality of the application at hand.

*User System*

|  |  |
| --- | --- |
| *Upon first login, initiates request for tutorial* |  |
|  | *Receives request for tutorial page* |
|  | *Gathers Tutorial page data* |
|  | *Builds Tutorial page* |
|  | *Delivers Tutorial page* |
| *Views Tutorial* |  |
| *After User has completed the tutorial, submit by clicking “Done”* |  |
|  | *Receives tutorial completion information and redirects User to Home page* |
|  | *Gathers Home page data* |
|  | *Builds Home page* |
|  | *Delivers Home page* |
| *User will be able to observe Home page* |  |

**Use Case:** Track Food

**Goal:** To allow the User to track nutrition facts based on their daily consumption of food.

**Scenario:** After the User has been granted access to the application, he/she will be directed to the Home page. From the Home page, the User will select “Track Food” in order to track the nutrition facts consumed in a day. The User will have the ability to select food already existing in the database or input nutrition facts for items not found in the database. The User may return Home after completion.

*User System*

|  |  |
| --- | --- |
| *Initiates request for the Track Food page by clicking on the Track Food link* |  |
|  | *Receives request for Track Food page* |
|  | *Gathers and Sends Track Food page* |
| *Views Track Food page and selects food currently in the database by using dropdown list* |  |
|  | *Receives submission. If food found in database, gather and send nutrition facts.*  *If “other” is selected, refresh Track Food page and display a message allowing User to enter nutrition facts* |
| *User will view nutrition facts populated with their values for the food selected. User will have the ability to click submit*  *Enters nutrition facts for food consumed not found in the database* |  |
|  | *Receives nutrition facts inputted from the User and stores them temporarily* |
| *Submit upon reviewing nutrition facts* |  |
|  | *Receives submission and updates all values for User daily totals. Refresh Track Food Page to allow for more food to be tracked* |
| *User repeats the process until all food has been entered.*  *User can return to the Home page by clicking the Home button* |  |
|  | *Receives request for Home page* |
|  | *Gathers and Sends Home page information* |
| *User will then view the Home page* |  |

**Use Case:** TrackExercise

**Goal:** To allow the User to track calories burned from exercise completed in a day.

**Scenario:** After the User has been granted access to the application, he/she will be directed to the Home page. From the Home page, the User will select “Track Exercises” in order to track the calories burned in a day. The User will have the ability to select an exercise already existing in the database and enter the duration for the completed exercise. Based on this information, the number of calories burned will be calculated. The User may return Home after completion.

*User System*

|  |  |
| --- | --- |
| *Initiates request for Track Exercise page by clicking on the Track Exercise link* |  |
|  | *Receives request for Track Exercise Page* |
|  | *Gathers Track Exercise page data* |
|  | *Builds Track Exercise page* |
|  | *Delivers Track Exercise page* |
| *Views Track Exercise page and selects completed exercise from list given* |  |
| *Enters duration for completed exercise* |  |
| *Clicks submit* |  |
|  | *Receives submission (exercise and duration)* |
|  | *Stores submission information temporarily for calculation purposes* |
|  | *Updates calculated calorie information in database (remaining and daily). Refresh Track Exercise page displaying calories burned (success) and allowing for more exercises to be tracked* |
| *Repeats process until all exercises have been tracked*  *Has the ability to return Home* |  |
| *Clicks Home link* |  |
|  | *Receives request for Home page* |
|  | *Gathers Home page data* |
|  | *Builds Home page* |
|  | *Delivers Home page* |
| *Views the Home page* |  |

**Use Case:** Update Stats

**Goal:** To allow the User to update individualized goals/measurements.

**Scenario:** After the User has been granted access to the application, he/she will be directed to the Home page. From the Home page, the User will select “Update Stats” in order update weight, goal weight, height, activity level, and body measurements (biceps, chest, neck, shoulders, and waist). The User may return Home after completion.

*User System*

|  |  |
| --- | --- |
| *Initiates request for Update Stats by clicking on Update Stats link* |  |
|  | *Receives request for Update Stats page* |
|  | *Gathers Update Stats page data. This includes page data as well as all User stats (weight, goal weight, height, activity level, and body measurements) stored in the database*  *If User statistics are NULL in the database, then fill those fields with dummy values* |
|  | *Build Update Stats page* |
|  | *Deliver Update Stats page* |
| *Views Update Stats page and enter in all values (weight, goal weight, height, activity level, and body measurements)* |  |
| *Clicks submit* |  |
|  | *Receives User stat information* |
|  | *Update all User stat information in the database. This includes several calculations* |
|  | *If successful, refresh Update Stats page and display success message*  *Otherwise, refresh Update Stats page and display error message* |
| *Views Update Stats page with error message. User may wish to repeat the process or continue to the Home page.*  *Views Update Stats page with success message. User has the ability to return Home* |  |
| *Clicks Home link* |  |
|  | *Receives request for Home* |
|  | *Gathers and sends Home page data* |
| *Views Home page* |  |

**Use Case:** View Stats

**Goal:** To allow the User to view current individual statistics.

**Scenario:** After the User has been granted access to the application, he/she will be directed to the Home page. From the Home page, the User will select “View Stats” in order to view the current statistics. This includes: current weight, goal weight, maintenance calories, activity level, BMI, BMI status, body measurements, and Ideal heart range. The User may return Home after completion.

*User System*

|  |  |
| --- | --- |
| *Initiates request for View Stats page by clicking the View Stats link* |  |
|  | *Receives request for View Stats page.* |
|  | *Gathers View Stats page data. This includes page information as well as all User stats in the database (current weight, goal weight, maintenance calories, activity level, BMI, BMI status, body measurements, and ideal heart range).*  *If User statistics in the database are NULL then fill those fields with dummy values.* |
|  | *Builds the View Stats page* |
|  | *Delivers View Stats page* |
| *User then observes View Stats page with current individual statistics* |  |
| *User has the ability to return Home* |  |
| *Clicks the Home page link* |  |
|  | *Receives request for Home page* |
|  | *Gathers Home page data* |
|  | *Builds Home page* |
|  | *Delivers Home page* |
| *Views Home page* |  |

**Use Case:** Edit Account

**Goal:** To allow the User to update account information.

**Scenario:** After the User has been granted access to the application, he/she will be directed to the Home page. From the Home page, the User will select “Account” in order to update account information. The User will be able to update the name, email, and password on the account. The User may return Home after completion.

*User System*

|  |  |
| --- | --- |
| *Initiates request for Edit Account page by clicking the Account link* |  |
|  | *Receives request for Edit Account page* |
|  | *Gathers Edit Account page data. This includes the page information as well as User information (name and email)* |
|  | *Builds Edit Account page* |
|  | *Delivers Edit Account page* |
| *Views Edit Account page. May update name, email, and/or password* |  |
| *Clicks submit*  *If new password is entered, it will be hashed* |  |
|  | *Receives User packet information ( name, email, and/or new password hash)* |
|  | *Updates the database with the new information* |
|  | *Refresh Edit Account page displaying new information (do not display password). Display success message* |
| *Views Edit Account page with new information and success message* |  |
| *Clicks Home link* |  |
|  | *Receives request for Home page* |
|  | *Gathers Home page data* |
|  | *Builds Home page* |
| *Views Home page* |  |

**Use case:** Help Tutorial

**Goal**: To allow the User to review the first login tutorial and some basic terminology found within the application.

**Scenario:** After the User has been granted access to the application, he/she will be directed to the Home page. From the Home page, the User will select “Help” in order to review the tutorial he/she viewed upon first login. The User will also be able to observe definitions to the terminology used in the application itself. The User may return Home after completion.

*User System*

|  |  |
| --- | --- |
| *Initiates request for Help page by clicking on the Help link* |  |
|  | *Receives request for Help page* |
|  | *Gathers information for the Help page* |
|  | *Builds the Help page* |
|  | *Sends the Help page* |
| *Views the Help page* |  |
| *Clicks the Home link* |  |
|  | *Receives request for Home page* |
|  | *Gathers and sends Home page* |
| *Views Home page* |  |

**Use Case:** Social Media

**Goal:** To allow the User to keep family and friends up-to-date with current progress via social media.

**Scenario:** After the User has been granted access to the application, he/she will be directed to the Home page. From the Home page, the User will select “Social Media” in order to post progress/goals met. The User may return Home after completion.

*User System*

|  |  |
| --- | --- |
| *Initiates request for Social Media page by clicking on the Social Media link* |  |
|  | *Receives request for Social Media page* |
|  | *Gathers information for the Social Media page* |
|  | *Builds Social Media page* |
|  | *Delivers Social Media page* |
| *Views the Social Media page* |  |
| *User enters Username and Password for external site (Facebook or Twitter)* |  |
| *Clicks Home link* |  |
|  | *Receives request for Home page* |
|  | *Gathers and sends Home page* |
| *Views Home page* |  |

Object-Oriented Design Metrics

# Cohesion

The level of cohesion for this design was calculated using the Lack of Cohesion Methods formula (or LCOM).

This method uses the number of methods that can access an attribute (R (A)), the number of attributes in the class (A), and the number of methods in the class (M).

This algorithm should produce results in the range 0 to 1 with zero representing perfect cohesion and 1 representing atrocious cohesion.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Class** | **ER(A)** | **A** | **M** | **LCOM** |
| **User** | 29 | 25 | 10 | 0.982222 |
| **Food** | 10 | 7 | 11 | 0.957143 |
| **Exercises** | 3 | 3 | 3 | 1 |
| **Avg LCOM** |  |  |  | **0.979788** |
|  |  |  |  |  |
|  |  |  |  |  |

The results from using the LCOM metric show that all of our modules within our current design are poorly cohesive. The general cut off for acceptance is .8 and all of our modules are over this limit. Looking over our classes we believe that with more time we could break down the user class and adjust the exercise class in order to improve cohesiveness. To reduce complexity we must reduce the number of interactions between modules. By limiting the amount of interactions between modules we will reduce the R in Graicunas Law which is R = M(2M-1 + M-1)

# Coupling

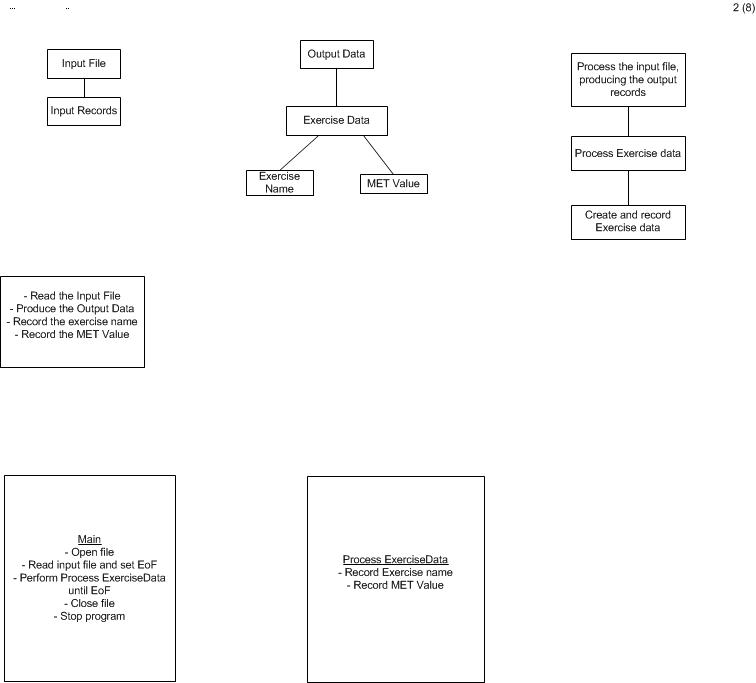
We measure the coupling in our design with the Coupling Instability formula.

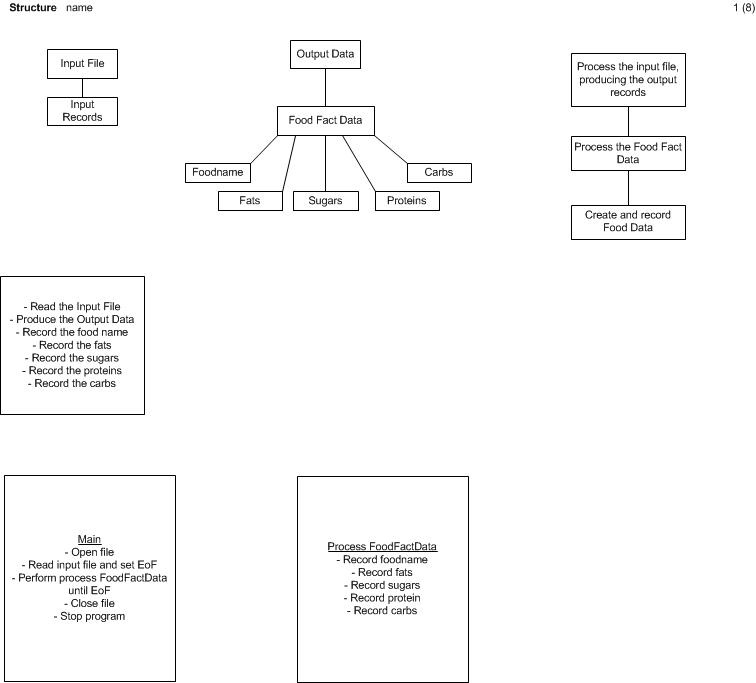
Ca is afferent couplings, the number of modules in other components that depend on modules in this component. Ce is the efferent couplings, the number of modules in other components. If a component has an I equal to zero, the module is considered to have maximum stabilization. If I is equal to one the module is unstable.

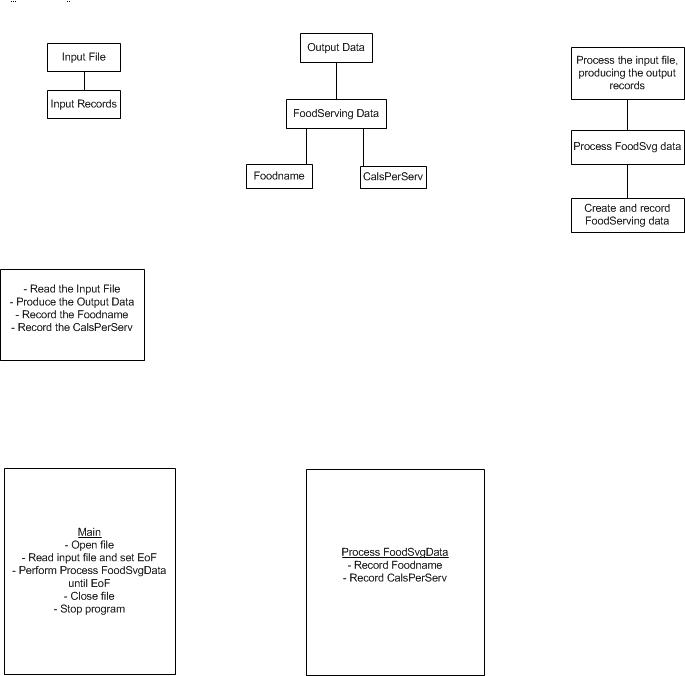
|  |  |  |  |
| --- | --- | --- | --- |
| **Class** | **Ca** | **Ce** | **I** |
| **User** | 3 | 6 | 0.666666667 |
| **Food** | 5 | 0 | 0 |
| **Exercises** | 2 | 3 | 0.6 |
| **Avg Instability** | |  | 0.422222222 |

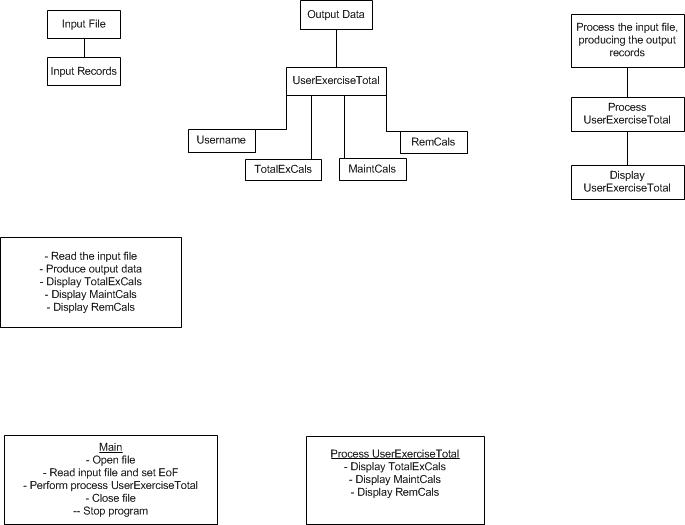
Based on the results our average instability was .422 which makes our program semi-stable. The stability of our food class helped our program a lot. With more time I’m positive our group could modify other classes besides food to improve stability and cohesion of the overall design.

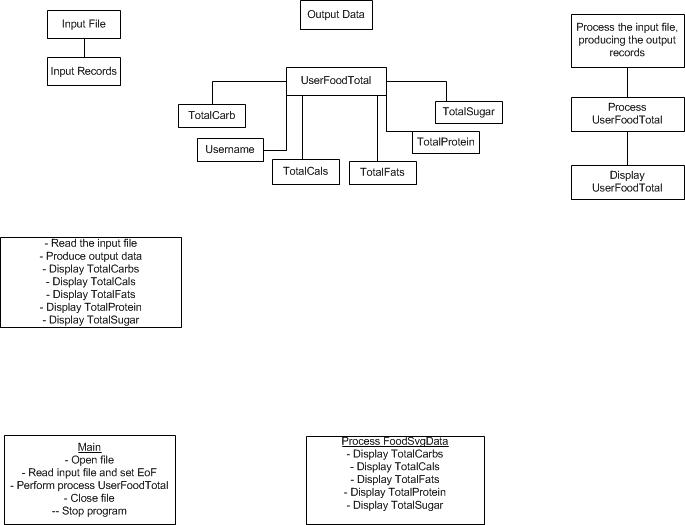
**Jackson Design (Static)**

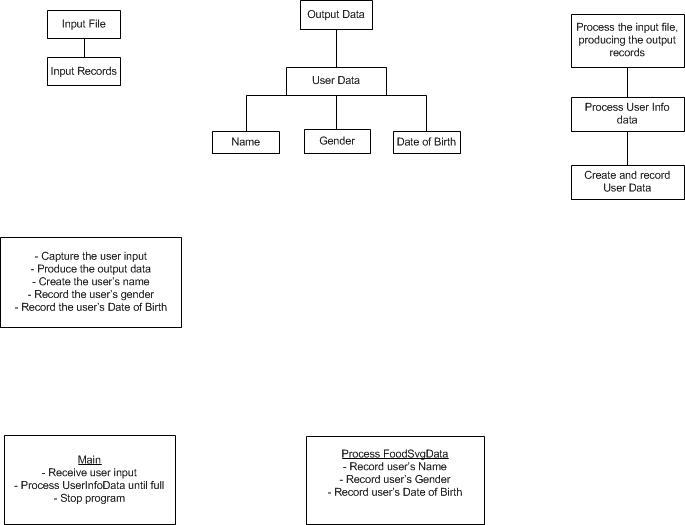


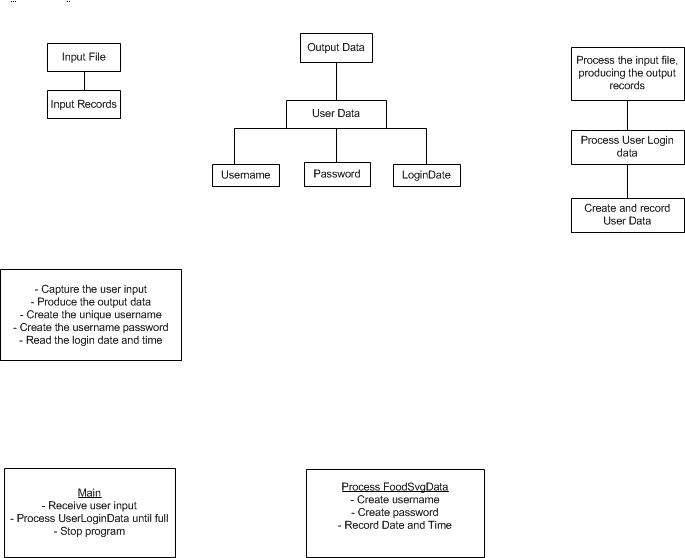


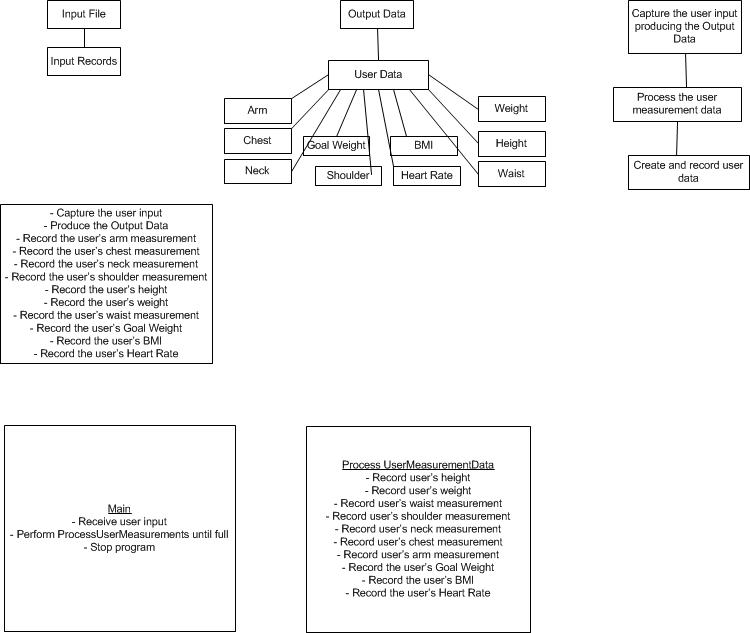












**Jackson Design Metrics**

By default Applications designed using M.A. Jackson’s static data flow analysis tends to provide a small number of closely related designs from N individuals. In contrast, typical design methodologies will tend to produce N number of deigns from N individuals. By using this metric and the other listed below we can quantify the quality of the software design and specification.

Using Miller’s & Graicunas Law we are able to determine if our application was successful in reducing its complexity. We achieve this by ensuring that the path of control and data between modules is limited to a single path. This exhibits a design sufficient to Miller’s Law. By ensuring this single path format we have successfully been able to implement Graicunas Law by limiting the contents of the modules to just one idea per module.

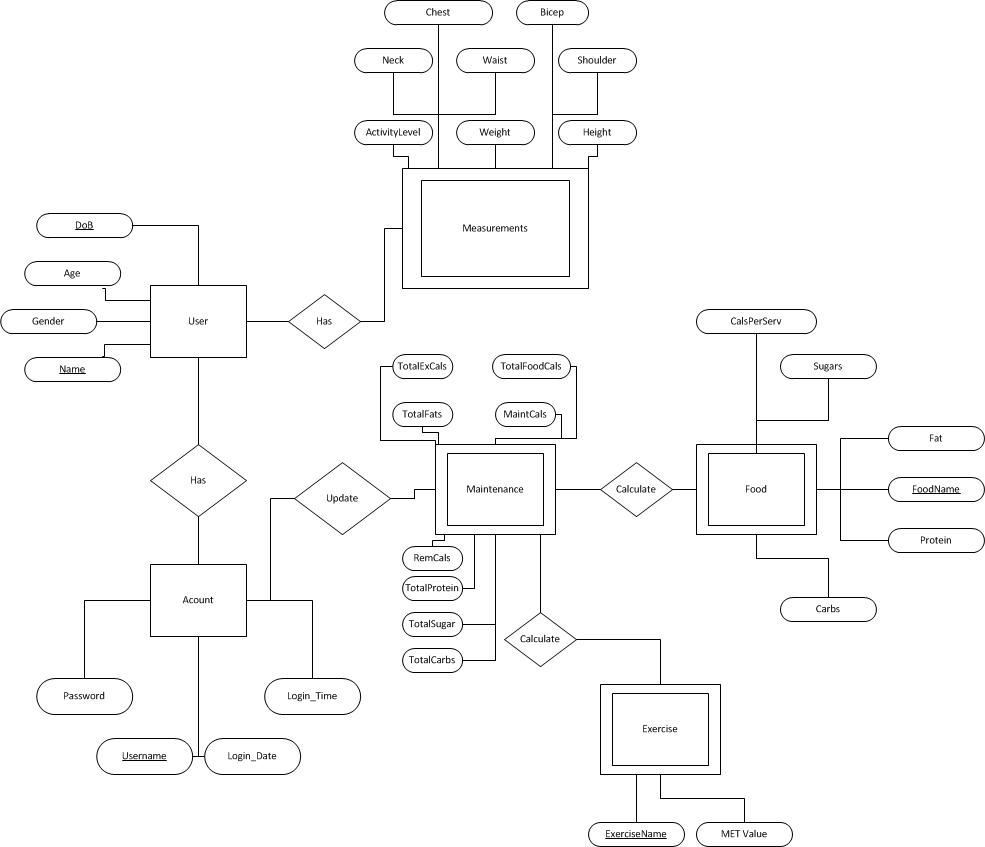
By effectively implementing Miller & Graicunas Laws, we are able to achieve a design that is highly modularized allowing for the implementation of the application to be considerably faster as compared to traditional designs. This decrease in time is achieved through the ability to employ several people on the project simultaneously.

This modularization of design alternatively produces a design rich in Cohesion and draught in Coupling. Very little Coupling can be seen throughout the design of our application. While at the same time our application is fraught with Sequential Cohesion.

Finally, the modularization also allows us to create a design where each module is separately implementable, maintainable, and modifiable. This concept is crucial to the implementation of new software. By using similar modules from previous applications that have been tested and working, we are able to reduce the time to distribution and the number of errors in the software. This allows us to find errors in the software faster, thus reducing the MTTR.

**Database Design Document**

**Entity Relationship Diagram**



**Database Normalization**

By using decomposition methods, a normalized version of the original database has been designed. This ensures that there will not be any update anomalies and eliminates redundancies. This allows for the database to function at its peak performance rate without any hiccups. The ER (Entity Relationship) diagram has given aid in decomposing the database design into its simplest form. In order to get the finalized set of tables, the ER diagram was studied and broken down using 1NF, 2NF, and 3NF. The resulting tables will be used for the final database design. Below are the tables derived from the ER diagram as wells as the decomposition for each Normal Form.

***Tables established from ER diagram:***

User (Username, name, DoB, age, gender)

Account (Username, password, login\_date, login\_time)

Measurement (Username, ActivityLevel, BMI, minHR, maxHR, weight, goalWeight, height, neck, waist, shoulder, chest, bicep)

Maintenance (Username, FoodName, Exercise name, totalExercise*,* calsPerServ, totalFoodCals, totalFats, totalProtein, totalSugar, totalCarbs, totalExerciseCals, maintenanceCals, remainingCals)

Food (FoodName, calsPerServ, sugar, fat, protein, carbs)

Exercise (ExerciseName, metValue)

***Functional Dependency for each table:***

User:

name, DoB 🡺 age, gender

name, DoB 🡺 Username

Account:

Username 🡺 password, login\_date, login\_time

Measurement:

Username 🡺 ActivityLevel, BMI, minHR, maxHR, weight, goalWeight, height, neck, waist, shoulder, chest, bicep

Maintenance:

Username, FoodName 🡺 totalFoodCals, totalFats, totalProtein, totalSugar, totalCarbs, remainingCals

Username, Exercise name 🡺 totalExerciseCals

Username 🡺 maintenanceCals

Food:

FoodName 🡺 calsPerServ, sugar, fat, protein, carbs

Exercise:

ExerciseName 🡺 metValue

***First Normal Form for each table:*** eliminate repeating groups in individual tables, create a separate for each set of related data, and identify each set of related data with a primary key

* User (Username ,name, DoB, age, gender)
* Account (Username, password, login\_date, login\_time)
* Measurement (Username, BMI, minHR, maxHR, ActivityLevel, goalWeight,weight, height, neck, waist, shoulder, chest, bicep)
* Maintenance (Username, FoodName, Exercise name, totalExercise*,* calsPerServ,totalFoodCals, totalFats, totalProtein, totalSugar, totalCarbs, totalExerciseCals, maintenanceCals, remainingCals)
* Food (FoodName, calsPerServ, sugar, fat, protein, carbs)
* Exercise (ExerciseName, metValue)

***Second Normal Form for each table:*** all non-key attributes are FD on the entire primary key, eliminate redundancy data

* UserInfo ( name,DoB, age, gender)
* AccountName (Username , name,DoB,)
* AccountInfo(Username, password, login\_date, login\_time)
* Measurement(Username BMI, minHR, maxHR, ActivityLevel, goalWeight,weight, height, neck, waist, shoulder, chest,bicep)
* FoodMaintenance(Username, FoodName, totalFoodCals, totalFats, totalProtein, totalSugar, totalCarbs, remainingCals)
* ExerciseMaintenance(Username, ExerciseName, totalExerciseCals)
* UserMaintenanceCalories(Username, maintenanceCals)
* Food(FoodName, calsPerServ, sugar, fat, protein, carbs)
* Exercise(ExerciseName ,metValue)

**Third Normal Form for each table:** all tables exhibited third normal form from second normal. That is, all the non-key attributes only depend on the primary key.

* UserInfo ( name, DoB, age, gender)
* AccountName (Username , name, DoB)
* AccountInfo (Username, password, login\_date, login\_time)
* Measurement (Username, BMI, minHR, maxHR, ActivityLevel, goalWeight, weight, height, neck, waist, shoulder, chest, bicep)
* FoodMaintenance (Username, FoodName, totalFoodCals, totalFats, totalProtein, totalSugar, totalCarbs, remainingCals)
* ExerciseMaintenance (Username, ExerciseName, totalExerciseCals)
* UserMaintenanceCalories (Username, maintenanceCals)
* Food (FoodName, calsPerServ, sugar, fat, protein, carbs)
* Exercise (ExerciseName , metValue)

**Tables and Relationships**

Food

Food entities in the database serve to identify their respective characteristics

Attributes: *foodName(PK), calsPerServ, fats, sugars, proteins, carbs*

*foodName* (Primary Key)

A unique food identifier. Should be prime and non-null. (must contain name)

*calsPerServ*

A description for the amount of calories in a serving of selected food. .

*Fats*

A description for the amount of fats in a serving. Should be non-null.

*Sugars*

A description for the amount of sugars in a serving. Should be non-null.

*Proteins*

A description for the amount of proteins in a serving. Should be non-null

*Carbs*

A description for the amount of carbs in a serving. Should be non-null.

Relationships with other tables:

Many-to-One relationship with FoodMaintenance table

Functional Dependencies:

foodName 🡺 calsPerServ, sugars, fats, proteins, carbs

Exercises

Exercise entities represent specific types or workouts and their respected MET Values.

Attributes: *ExerciseName(PK), MetValue*

*ExerciseName(Primary Key)*

A unique exercise identifier. Should be prime and non-null. (Must contain name)

*MetValue*

A numeric description associated with the exercise. Should be non-null. This is used in a formula to calculate the amount of calories burned during the exercise.

Relationships with other tables:

Many-to-one relationship with ExerciseMaintenance table

Functional Dependencies:

ExerciseName 🡺metValue

UserMeasurements

User measurement entities will represent measurements provided via the user or calculations.

Attributes: *Username(PK), goalWeight, weight, height, activityLevel, BMI, minHR, maxHR, bicepMeasurement, chestMeasurement, neckMeasurement, shoulderMeasurement, waistMeasurement*

*Username(Primary Key)*

A unique identifier for users. Should be prime and non-null.

*goalWeight*

A user defined numeric value of a weight they want to achieve. If no input is received, a placeholder will be assigned.

*Weight*

User defined numeric value of their current weight.

*Height*

User defined numeric value of their current height.

*activityLevel*

A pre-defined value that the user will select. This represents how physically active the user is. This will be used in calculation(s). Should be non-null.

*BMI*

BMI will be a numeric value calculated with user defined values rom the User\_Info table. Should be non-null.

*minHR*

minHR will be a numeric value calculated with user defined values rom the User\_Info table. Should be non-null.

*maxHR*

maxHR will be a numeric value calculated with user defined values rom the User\_Info table. Should be non-null.

*bicepMeasurement*

A user defined numeric value for the measurement of user’s bicep in inches.

*chestMeasurement*

A user defined numeric value for the measurement of user’s chest in inches.

*neckMeasurement*

A user defined numeric value for the measurement of user’s neck in inches.

*shoulderMeasurement*

A user defined numeric value for the measurement of user’s shoulder in inches.

*neckMeasurement*

A user defined numeric value for the measurement of user’s neck in inches.

Relationship with other tables:

One-to-one relationship with AccountName table.

One-to-one relationship with AccountInfo table.

One-to-one relationship with FoodMaintenance table.

One-to-one relationship with ExerciseMaintenance table.

One-to-one relationship with UserMaintenanceCalories table.

Functional Dependencies:

username🡺 goalWeight, weight, height, activityLevel, BMI, minHR, maxHR, bicepMeasurement, chestMeasurement, neckMeasurement, shoulderMeasurement, waistMeasurement

UserInfo

UserInfo will contain user defined values to aid in calculation of certain variables and login information.

Attributes: *name(PK), DoB(PK), age, gender*

*Name(Primary Key)*

User’s personal name. Should be non-null.

*DoB(Primary Key)*

User’s date of birth. Should be non-null.

*Gender*

A user defined value representing male or female. Should be non-null.

*Age*

A user defined value representing how old they are. This is used in multiple calculations. Should be non-null.

Relationships with other tables:

One-to-one relationship with AccountName table.

Functional Dependencies:

Name, DoB🡺age, gender

AccountInfo

AccountInfo will contain information in which will be access at the time of login. This information will be used for resetting daily totals.

Attributes: *username(PK), password, loginDate, loginTime)*

*username*(Primary Key)

A unique identifier for users. Should be prime and non-null.

*Password*

A created identifier by users for gaining access to the application. Should be non-null.

*loginDate*

A value stored upon login and used to clear daily values. This will be checked for changes on each login. Should be non-null.

*loginTime*

A value stored upon login and used to clear daily values. This will be checked for changes on each login. Should be non-null.

Relationships with other tables:

One-to-one relationship with AccountName table.

One-to-one relationship with FoodMaintenance table.

One-to-one relationship with ExerciseMaintenance table.

One-to-one relationship with UserMaintenanceCalories table.

One-to-one relationship with Measurement table.

Functional Dependencies:

username🡺password, loginDate, loginTime

AccountName

AccountName will contain information strictly relating to user entered data.

Attributes*: username(PK), name, DoB*

*username*(Primary Key)

A unique identifier for users. Should be prime and non-null.

*Name*

A name that the user chooses to be called. Should be non-null.

*DoB*

User’s date of birth. Should be non-null.

Relationships with other tables:

One-to-one relationship with AccountInfo table.

One-to-one relationship with FoodMaintenance table.

One-to-one relationship with ExerciseMaintenance table.

One-to-one relationship with UserMaintenanceCalories table.

One-to-one relationship with Measurement table.

Functional Dependencies:

username🡺name, DoB

FoodMaintenance

FoodMaintenance entities will represent an accumulation of food information.

Attributes: *username(PK), foodName, totalFoodCals, totalFats, totalProteins, totalSugars, totalCarbs, remainingCals)*

*username*(Primary Key)

A unique identifier for users. Should be prime and non-null.

*foodName*

The name of the food found in the database.

*totalFoodCals*

An accumulation of daily calories from food selected by users. Should be non-null.

*totalFats*

An accumulation of daily fats from food selected by users. Should be non-null.

*totalProteins*

An accumulation of daily proteins from food selected by users. Should be non-null.

*totalSugars*

An accumulation of daily sugars from food selected by users. Should be non-null.

*totalCarbs*

An accumulation of daily carbs from food selected by users. Should be non-null.

*remainingCalories*

A numeric value representing the daily amount of remaining calories the user needs to consume to meet their maintenance caloric intake. Should be non-null.

Relationships with other tables:

One-to-one relationship with AccountInfo table.

One-to-one relationship with AccountName table.

One-to-one relationship with ExerciseMaintenance table.

One-to-one relationship with UserMaintenanceCalories table.

One-to-one relationship with userMeasurements table.

One-to-many relationship with Food table.

Functional Dependencies:

username🡺foodName, totalFoodCals, totalFats, totalProteins, totalSugars, totalCarbs, remainingCals

ExerciseMaintenance

ExerciseMaintenance entities will represent an accumulation of exercise information.

Attributes: *username(PK), exerciseName, totalExerciseCals*

*username*(Primary Key)

A unique identifier for users. Should be non-null and prime.

*exerciseName*

The name of the exercise found in the database. Should be non-null.

*totalExerciseCals*

A numeric value representing accumulation of daily calories burned from using a defined formula. Should be non-null.

Relationships to other tables:

One-to-one relationship with AccountInfo table.

One-to-one relationship with AccountName table.

One-to-one relationship with FoodMaintenance table.

One-to-one relationship with UserMaintenanceCalories table.

One-to-one relationship with userMeasurements table.

One-to-many relationship with Exercise table.

Functional Dependencies:

username🡺ExerciseName, totalExerciseCals

userMaintenanceCalories

UserMaintenanceCalories will hold the value for the user’s maintenance calories.

Attributes: *username(PK), maintenanceCals*

*username*(Primary Key)

A unique identifier for users. Should be non-null and prime.

*MaintenanceCals*

A numeric value representing the daily calorie consumption to maintain current weight, using the Harris Benedict Equation. Should be non-null.

Relationships with other tables:

One-to-one relationship with AccountInfo table.

One-to-one relationship with AccountName table.

One-to-one relationship with FoodMaintenance table.

One-to-one relationship with ExerciseMaintenance table.

One-to-one relationship with userMeasurements table.

Functional Dependencies:

username🡺maintenanceCals

**Server Technologies**

Servlets are a Java class used to extend the capabilities of a server. This is primarily used when an Application is hosted on a Web server. They are the Java counterparts to Microsoft’s ASP.Net web technology. They are most commonly used for processing or storing data submitted in an HTML form, providing results of a database query, and managing state information not stored in the stateless HTTP protocol. The content of a servlet is commonly HTML but can also be XML. A web container must be used to run or deploy a servlet. This web container or servlet container is what interfaces between the web server that is publishing the application and the servlet itself.

Our application is a 3-tiered system that will employ servlets to process the users request for information, make modifications to their records in the database, and take advantage of it’s faster performance, ease of use, and better memory utilization as compared to other web application models such as CGI.

The use of Java as out implementation language allows us to use these servlet resources to improve the performance of our application by reducing the time it takes for a user to create, modify, or delete a record in the database.

**Implementation Plan**

Phase 1:

* Implement servlet
* Build the Home page (default page)
  + Verify it is working properly
* Build the Login page
  + Verify it is working properly
  + Password recovery page as well
    - Verify it is working properly
* Build database and set connections
  + Verify that all connections are correct

Phase 2:

* Build other pages for major functionality
  + View Stats page
    - Verify it is working properly
  + Update Stats page
    - Verify it is working properly
  + Track Exercises page
    - Verify it is working properly
  + Track Food page
    - Verify it is working properly
  + Build Tutorial
    - Verify it is working properly
* Build the rest of the pages
  + Help page
    - Verify it is working properly
  + Account page
    - Verify it is working properly

Phase 3: (If time allows)

* Build Social Media page
  + Verify it is working properly
* Clean up the website
  + Add graphics and make it more eye appealing
* Test the system using the “Test Cases” document
* Add more User functionality
* May extend the application to other platforms

**Test Case Document**

This project, an android application, is a 3-tiered system. The user population will connect to the application, which in turn is hosted by a server. This server is then connected to the database where certain information regarding food and exercises is stored. This information will be used for calculation purposes only. Along with this information, the user’s information will be stored as well. Designing the project this way has eliminated a direct connection between the user and the database.

The FAT (Fitness Activity Tracker) application will allow individual users to keep track of their daily calories and goal measurements. The user will use the database to look up information for calculations only. The user will not be storing any information in the Food table or the Exercise table. Once the information has been calculated (BMI, maintenance calories, and ect.) that information will be stored in a User table. By doing this, we have avoided all possible cases for race conditions because the user only has the ability to update their personal information. No two users (or more) will be trying to update the same information at the same time. The database is strictly setup to only update information that pertains to the user that is currently logged in to the system. The same is true for deadlocks. The way the system and database are designed should help eliminate all plausible threats for deadlocks or race conditions.

In the pages following, the possible test cases are listed and how each case will be tested. It is quite obvious that the group will encounter some issues that are not anticipated because of the complexity of this project. The majority of the tests will be error checking and if an error is noticed the appropriated measures will be taken to get it fixed. The functionality of the FAT application and the way it was designed requires the user to navigate through the application and submit data with numerous button clicks. Because there are a copious amount of button clicks, The Great Unwashed will spend the majority of the time testing the buttons (links) and ensuring that the appropriate function (s) result.

Login Screen

Upon login, the system will prompt the user for his/her credentials (username and password). The user will then be required to press the “Sign In” button on the Login Screen. This will allow the user to access the application or it could display an error message if there was a problem with the credentials. If the client is not a previous user (does not have an account) then the “Sign up” button must be clicked prior to signing in. If a previous user has forgotten the password for the account, he/she may press the “Forgot Password” button in order to enter the password recovery application.

Testing the “Sign In” button and fields on the Login Screen:

* Using blank fields for login credentials
  + Application should return an error message
* Using incorrect login credentials (correct password, but wrong username OR correct username, but wrong password OR both wrong)
  + Application should return an error message
* Using correct combination of login credentials
  + Application should grant the user access and direct the user to the Home page
* Upon clicking the “Sign In” button
  + Application should do one of the above

Testing the “Sign up” button and fields on the New Account Screen:

* Upon clicking the “Sign Up” button
  + Application should redirect the user to the New Account page
* Using incorrect data types for fields (integer when should be string OR all combinations)
  + Application should display an error message
* Using blank fields for required information (all combinations)
  + Application should display an error message
* Using a username that is not unique
  + Application should display an error message
* Confirmation password does not match password
  + Application should display an error message
* Confirmation password does match password
  + Application should display success message
* Upon clicking the “Cancel” button
  + Application should abort
* Upon clicking “Submit “ and all fields filled in correctly (no other problem persists)
  + Application should redirect user to Login Screen and display success message and show account created

Testing the “Forgot Password” button and fields on the Password Recovery Screen:

* Upon clicking the “Forgot Password” button
  + Application should redirect the user to the Password Recovery Screen
* Using incorrect data types for fields (integer when should be string OR all combinations)
  + Application should display an error message
* Using blank fields for required information (all combinations)
  + Application should display an error message
* Confirmation password does not match password
  + Application should display an error message
* Email/username not found in database (no match)
  + Application should display an error message
* Email and username found in database (match)
  + Application should allow for new password
* Confirmation password does match password
  + Application should display success message
* Upon clicking the “Cancel” button
  + Application should abort and return to Login page

Home Screen

Upon successful login the user will be redirected to the Home screen. From here the user has the ability to directly access many other parts of the application. First time users will be redirected to a start tutorial explaining major functionality of the FAT application.

Testing first time users:

* Upon a successful login for first time users
  + Application should redirect user to the tutorial
* User cannot exit tutorial until it is complete
  + Application should not allow for user to access parts outside the tutorial
* After completion of the tutorial (selecting done)
  + Application should allow user to have full functionality of the FAT application
* User completes tutorial and selects “See Again”
  + Application should restart to tutorial

Testing returning users:

* Upon successful login
  + Application should redirect user to Home page
  + Remaining daily calories should be visible
* Upon clicking “View Stats”
  + Application should redirect user to the View Stats screen
* Upon clicking “Update Stats”
  + Application should redirect user to the Update Stats screen
* Upon clicking “Help”
  + Application should redirect user to the Help page
* Upon clicking “Account”
  + Application should redirect user to the Account page
* Upon clicking “Track Exercises”
  + Application should redirect user to the Track Exercise page
* Upon clicking “Track Food”
  + Application should redirect user to the Track Food page
* Upon clicking the “Social Media” button
  + Application should redirect user to the Social Media page

View Stats Screen

Upon successful login and when the user clicks the “View Stats” button, he/she will be redirected to the View Stats screen. This page will display all current statistics and future goals for the user.

Testing View Stats:

* Once the user is redirected to the View Stats page
  + Application should gather statistical data for individual user stored in the database and display all information appropriately and in the correct fields
  + All calculations should be displayed correctly
* Upon viewing the View Stats page
  + User should not have the ability to change any information on the page
* Upon clicking “Home” button
  + Application should redirect user to the Home page

Update Stats Screen

Upon successful login and when the user clicks the “Update Stats” button, he/she will be redirected to the Update Stats screen. This page will allow the user to update any individualized goals/measurements.

Testing Update Stats:

* Once the user is redirected to the Update Stats page and if the user has not recorded data previously
  + Application should display all fields NULL
* Once the user is redirected to the Update Stats page and if the user has recorded data previously
  + Application should gather from the database and display the users individualized prerecorded data
* Upon clicking “Save”
  + User tries to update information leaving some fields blank
    - Application should display an error message requiring all fields to be full
  + User tries to update will all fields blanks
    - Application should display an error message requiring all fields to be full
  + User tries to update information with wrong data types in fields
    - Application should display an error message
  + User tries to update information with correct data types in fields
    - Application should refresh page and display success message and refresh page
* Upon clicking “Home”
  + Application should redirect user to the Home page

Help Screen

Upon successful login and when the user clicks the “Help” button, he/she will be redirected to the Help page. This page will allow the user to review the starting tutorial and view basic terminology found throughout the FAT application.

Testing the Help page:

* Once the user is redirected to the Help page, he/she may click the “Tutorial” button
  + Application should redirect user to the tutorial and after completion user will be redirected to the Home page
* Once the user is redirected to the Help page, he/she may click the “Terminology” button
  + Application should redirect the user to the terminology page
  + User will click the “Back” button , which should take him/her to the Help page
* When the user is on the Help page and clicks “Home” button
  + Application should redirect user to the Home page

Account Screen

Upon successful login and when the user clicks the “Account” button, he/she will be redirected to the Account page. This page will allow the user to update account information (username, email, and password).

Testing Account:

* Once user has been redirected to the Account page, he/she should have the ability to view username and email
  + Application should gather information from database and display information to the user
* Upon clicking “Confirm”
  + User attempts to update account information with a mixture of characters (uppercase, lower, and special characters)
    - Application should allow this to happen
  + User attempts to update with NULL fields
    - Application should display error message and update attempt will be voided
  + User attempts to update one field
    - Application should allow this to happen
* Upon clicking “Home” button
  + Application should redirect the user to the Home screen

Track Exercises Screen

Upon successful login and when the user clicks the “Track Exercises” button, he/she will be redirected to the Track Exercises page. This page will allow the user to track calories burned for a particular exercise.

Testing Track Exercises:

* Once user has been redirected to the Track Exercises page, he/she should have the ability to view all the exercises in the database via a dropdown list
  + Application should display all exercises in a dropdown list currently in the database
  + Application should allow for user selection
* Upon clicking “Submit” button
  + User attempts to enter a duration with an incorrect data type
    - Application should display an error message and refresh page
  + User attempts to enter a duration with the correct data type
    - Application should do all calculations and update the database
    - Application should display calories burned to the user (success)
* Upon clicking “Home” button
  + Application should redirect the user to the Home page

Track Food Screen

Upon successful login and when the user clicks the “Track Food” button, he/she will be redirected to the Track Food page. This page will allow the user to keep track of food nutrition facts (fats, carbohydrates s, calories, sugars, and proteins) consumed in a day. The user will select food existing in the database for information needed for the calculations. If the food is not in the database the user will be prompted for the information needed.

Testing Track Food:

* Once user has been redirected to the Track Food page, he/she should have the ability to select food from the database (via a dropdown list or searchable method).
  + Application should allow user to select the food he/she chooses
  + Application should allow user to only select one food at a time
* If food not in the database
  + Application should allow user to enter in nutrition facts
* Upon clicking “Submit”
  + User attempts to enter incorrect data type for nutrition facts
    - Application should display error message
  + User attempts to enter incorrect data type for serving size
    - Application should display an error message
  + User enters in all correct data
    - Application should perform calculations, update database
    - Application should display success message
* Upon clicking “Home”
  + Application should redirect the user to the Home page

Social Media Screen

Upon successful login and when the user clicks the “Social Media” button, he/she will be redirected to the Social Media page. This page will act as a redirecting tool that will aid the user in posting/tweeting daily progress.

Testing Social Media:

* Once user has been redirected to the Social Media page, he/she should have the ability to view links to Facebook or Twitter
* Upon clicking on the “Facebook” link
  + Application should redirect user to Facebook
* Upon clicking on the “Twitter” link
  + Application should redirect user to Twitter
* Upon clicking on the “Home”
  + Application should redirect user to the Home page

**Design Selection**

In determining which design was most appropriate for our application, we employed the design metrics for each possible design solution. After reviewing these metrics it was decided that M.A. Jackson’s Static Data Analysis would be best suited for our application. In Implementing M.A. Jackson’s Static Data Flow Analysis we should be able to achieve high levels of cohesion and low levels of coupling. This design will allow us to comply with Miller’s Law by producing a single path for both data and control flow while allowing for singular modularity thus fulfilling Graicunas Law. By complying with these it will produce modules that are separately maintainable, implementable, and modifiable. This modularization will enable the modules to be deposited into a library for reuse on further projects.