Process MeNtOR 3.o

Uni-SEP

McBal Demographic Analysis Application (MDAA)

Design Document

Version:	1.2.1
Print Date:	
Release Date:	
Release State:	Initial
Approval State:	Draft
Approved by:	
Prepared by:	Henry So, Jacob Chun, Samuel Su, Yan Qing Niu
Reviewed by:	
Path Name:	
File Name:	SDD.pdf
Document No:	1

object oriented pty. ltd.

Document Change Control

Version	Date	Authors	Summary of Changes
1.0	3/15/2021	Henry So, Jacob Chun, Samuel Su, Yan Qing Niu	Initial edits of Main Page, Introduction, and Major Design Decisions.
1.1	3/17/2021	Henry So, Jacob Chun, Samuel Su, Yan Qing Niu	UML component diagrams fully completed. UML class diagrams initially drafted.
1.2	3/20/2021	Henry So, Jacob Chun, Samuel Su, Yan Qing Niu	Class diagrams fully completed. Images were inserted to the appropriate sections.
1.2.1	3/21/2021	Henry So, Jacob Chun, Samuel Su, Yan Qing Niu	Completed the Test Driven Development section of the document. Final edits were made.

Document Sign-Off

Name (Position)	Signature	Date
Henry So	HS	3/21/2021
Jacob Chun	JC	3/21/2021
Samuel Su	SS	3/21/2021
Yan Qing Niu	DN	3/21/2021

Contents

1 Introduction

1.1 Purpose

This document details the design description of a demographic statistical analysis application utilizing data from the World Bank's data repository. It outlines the details and various design decisions relevant to the development of the application, including architecture, class diagrams, and design patterns for the entire system. It outlines the development activities schedule for the application.

The programming tasks for this project are summarized as follows:

1. To create a database system for user login and associated information.
2. To create an interface for the user to interact with.
3. To create a method that allows for retrieval of information from the World Bank Servers
4. To create a method that creates Viewers using the data retrieved from the aforementioned servers that can be displayed to the user interface.

1.2 Overview

The goal of this project is to implement an application system that enables users to analyze and visualize key data metrics derived from the World Bank's data repository. Additionally, the system must: a) enable the retrieval of demographic and other data for one selected country from the World Bank's data repository; b) process the data using different types of analyses; c) render the retrieved data or the processed data using appropriately selected visualization mediums such as bar charts, line graphs, scattered plots, and pie charts.

The SDD document contains the following information:

1. All major design decisions including significant design choices and modularization criteria (high cohesion and low coupling) will be thoroughly explained. Further, the document details how the design of the system achieves these criterias.
2. Component Diagram of the system (Architecture) will be shown including explanations on the functionality of each component. Additionally, the operations on various interfaces exposed by these components will be detailed.
3. Detailed class diagram for all classes created or modified in the system with only one level of associations. The detailed class diagram contains the classes in UML notation and a table for each class with its data members and methods with the appropriate signatures. Design patterns are identified.
4. Pseudo code for all major methods in the classes written or modified. The major methods are the ones in the Login Proxy, Server Communicator, and Interface. These are the methods for login (Login Proxy), retrieval of information from World Bank servers (Server Communicator), rendering of Viewers (Interface), and user interface (Interface).
5. Specific design patterns chosen for the effective implementation of the software system will be discussed. This pertains to the way they affect the development of the system into an efficient application for use.
6. Activities Plan, Product Backlog, and Sprint Backlog for the project so far.
7. Test Driven Development cases for verification of code.

1.3 Resources - References

This SDD follows the design specification specified in CS2212B-Project-Description.pdf and can be found on OWL COMPSCI CS2212B Project Resources Tab.

This SDD extends upon the design specification specified in McBal-SRS.pdf and can be found on OWL COMPSCI CS2212B Assignments Tab under Deliverable 2 - Software Requirements Specification - Part 2.

Eclipse: http://www.eclipse.org/downloads/index.php
Gson: https://sites.google.com/site/gson/gson-user-guide
JSON: https://www.w3schools.com/js/js_json_intro.asp
GANTT: http://wiki.phprojekt.com/index.php/Gantt-diagram
Java URL: https://docs.oracle.com/javase/7/docs/api/java/net/URL.html
HttpURLConn:
https://docs.oracle.com/javase/8/docs/api/java/net/HttpURLConnection.html
Gantt Chart Software: https://www.teamgantt.com/
UML Diagram Builders: https://www.umlet.com/
UML: https://en.wikipedia.org/wiki/Unified_Modeling_Language
Component Diagram: https://www.uml-diagrams.org/component-diagrams.html
Class Diagram: https://www.uml-diagrams.org/class-diagrams-overview.html
Design Patterns: https://sourcemaking.com/design_patterns
Software Design Basics:
https://www.tutorialspoint.com/software_engineering/software_design_basics.htm
API Calls:
https://apifriends.com/api-management/whats-api-call/#:~:text=An%20API%20call%20is%20the,actions%20to%20drive%20digital%20success
Scrum: https://www.scrum.org/

2 Major Design Decisions

Modularization partitions the entire systems overall development effort into clearly defined components, enhancing the organizational structure by seamlessly integrating to satisfy problem requirements. As the system evolves through its stages of development, changes to one module are isolated from changes to other modules. This benefits the principle of continuity such that a change in requirements triggers a change to one module only. Further, modularization allows the system to be more easily understood during development in terms of separable issues. The McBal Demographic Analysis Application development modularization criteria is based on the following design factors:

1. Separation of concerns
2. Functional independence
3. Information hiding
4. High cohesion
5. Low coupling

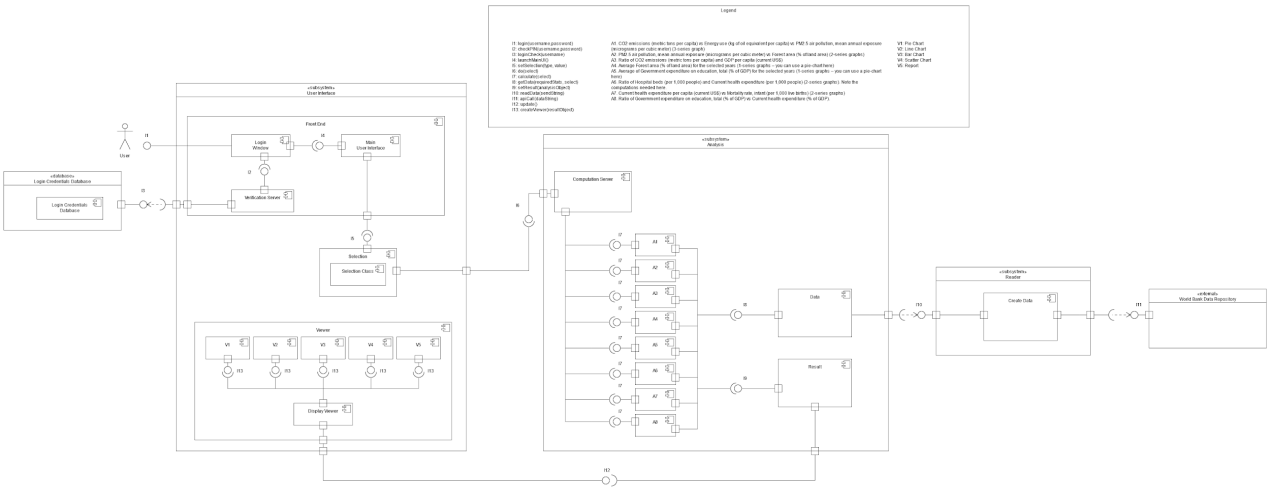
Firstly, separation of concerns will subdivide the system into individual pieces to be solved independently. By separating these concerns into smaller, more manageable tasks, the overall application development will benefit from a reduced project timeline. Secondly, functional independence will constrain each module in its scope

and functionality, by giving each module a single purpose to fulfil. This benefits development, testing, and maintenance of modules by reducing the effect that changes to one section have on the software as a whole. It increases cohesiveness of the program, by minimizing interaction with other parts of the program, and decreases coupling, by reducing interdependence among different modules. Lastly, information hiding ties back to the two factors of separation of concerns and functional independence. By effectively utilizing information hiding, this will localize all future change by hiding system details that are likely to change independently. That being said, assumptions and components unlikely to change will be exposed in interfaces. Since the amount of interfaces will be minimal, this aids in reducing the overall level of coupling by decreasing the amount of dependencies between modules

With these modularization choices in mind, the system will follow the principle of functional cohesion to achieve high cohesion. Every module will be designed to focus on exactly one goal to achieve. Thus, a module performing a well defined task will be readily reusable. Further, modularization and functional cohesion will allow developers to isolate fault when testing and deploying. Pinpointing a bug or issue with the system will be simplified when every module performs one well defined task.

With regards to achieving low coupling, the development of the McBal Demographic Analysis Application will consider design decisions such as implementing data coupling. By passing only the required parameters to a module or only simple data types such that no extra information is passed to a module. Further, the development will avoid stamp coupling such as passing entire objects into a separate module. This will avoid making modules dependent on specific data fields, thus increasing portability and reusability.

3 Architecture



Component Diagram for the System Architecture / Design

Login Server

- This component's functionality involves the login screen of the McBal Demographic Analysis System. The login server handles the credentials entered by the User.

Verification Server

- The verification server is responsible for querying the login credentials database for the correct password associated with a username. Further, it verifies the correctness of the supplied password to the actual password.

Login Credentials Database

- The login credentials database contains all combinations of valid username and password combinations. Further, there is an interface exposed to allow retrieval of information.

Main User Interface

- The main User Interface displays an interactive menu that allows the User to select an analysis type, country, and time period to gather data on the chosen analysis type. This data is then rendered onto the user interface through mediums of graphs such as pie charts, bar charts, line charts, scatter plots, and reports.

Selection Class

- The selection component is responsible for creating a selection object that stores all user selected options in the main interface. This includes an interface that will set instance variables of the object such as start year, end year, etc.

Computation Server

- The computation server is responsible for handling business logic associated with the analysis subsystem. It will receive a selection object, and it will parse it for all necessary information. It then uses the extracted information to create analysis objects using the factory design pattern.

A1-A8

- These components represent the various analysis types as indicated by the legend in the upper corner. These all inherit and expose the same interface which is used by the computation server. Further, these classes contain the necessary instance variables utilized for each type such as GDP per capita, etc. These classes interface with the data class and the result class.

Data

- This component is responsible for making data requests using I10 to the Reader subsystem. This functions as a facade, controlling all needs for getting the required data for the analysis component. Every statistic needed for the analysis type will be requested separately. As the data is being requested and received, an array of the required data will be constructed.

Create Data

- The create data component will construct the API Get request string to collect the proper information. Once the JSON is collected, it will be parsed to retrieve the necessary statistics and then passed back to the data class.

World Bank Data Repository

- The World Bank Data Repository represents the external database which holds the information to be retrieved when a country is selected. It is accessed via API calls from Create Data, and will send back a JSON to Create Data

Result

- The result component will take the completed analysis object passed and will construct a result object. The data stored in the result object will be the data from the World Bank Data Repository, formatted to be used with the viewers. It will then send the result object and analysis object to initialize the desired viewers with the correct statistics.

Display Viewer

- After receiving the result object, the display viewer component is responsible for using interface I13 to create the appropriate viewer objects, and then render them once they are all complete.

V1-V5

- These components represent the different viewers that are used to show the retrieved data, as shown by the legend in the upper-right corner.

Associated Interfaces

I1: login(username,password) The login server takes the username and password entered by the user.

I2: checkPIN(username,password) The login server gives the username and password to the verification server, this uses I3 in order to check if the username and password are correct.

I3: loginCheck(username) The verification server sends the username to the login credentials database and the database will return the associated password, if any.

I4: launchMainUI() Once the user's username and password have been successfully confirmed, the login server notifies the main user interface to start the app.

I5: setSelection(type, value) When the user correctly selects an option (analysis type, country, etc.) the main user interface sends the type of selection and value selected to the selection class, where it will be stored in the selection object.

I6: do(select) Given the selection object, the computation server utilizes the factory design pattern to select the appropriate analysis class to create.

I7: calculate(select) Once the appropriate analysis class has been selected by the factory, the factory sends the required data to create the analysis object.

I8: getData(requiredStats, select) When the analysis object requires statistics information, it will send a selection object to the data class and, through a façade design pattern, the data class will get and return the necessary information.

I9: setResult(analysisObject) Once the analysis object has been created with the required data, the result class will format the data appropriately and create a result object to store said data.

I10: readData(sendString) The sendString is used to pass to the reader and utilized to construct the API call.

I11: apiCall(dataString) Given the dataString, the Reader makes the appropriate API/HTTP call to the World Bank Data Repository for the correct data to be returned.

I12: update() Once the result object has been created, the Model View Controller is notified and begins the process of rendering the graphical representations through an observer design pattern.

I13: createViewer(resultObject) The individual viewer class will parse the result object for the statistics to plot/graph the data.

In the component diagram detailed, there are a selection of architectural styles utilized to aid the design of the McBal Demographic Analysis Application. There are three main architectural styles used:

1. Three tier architecture design
2. Transactional database data oriented repository architecture
3. Model-View-Controller architecture

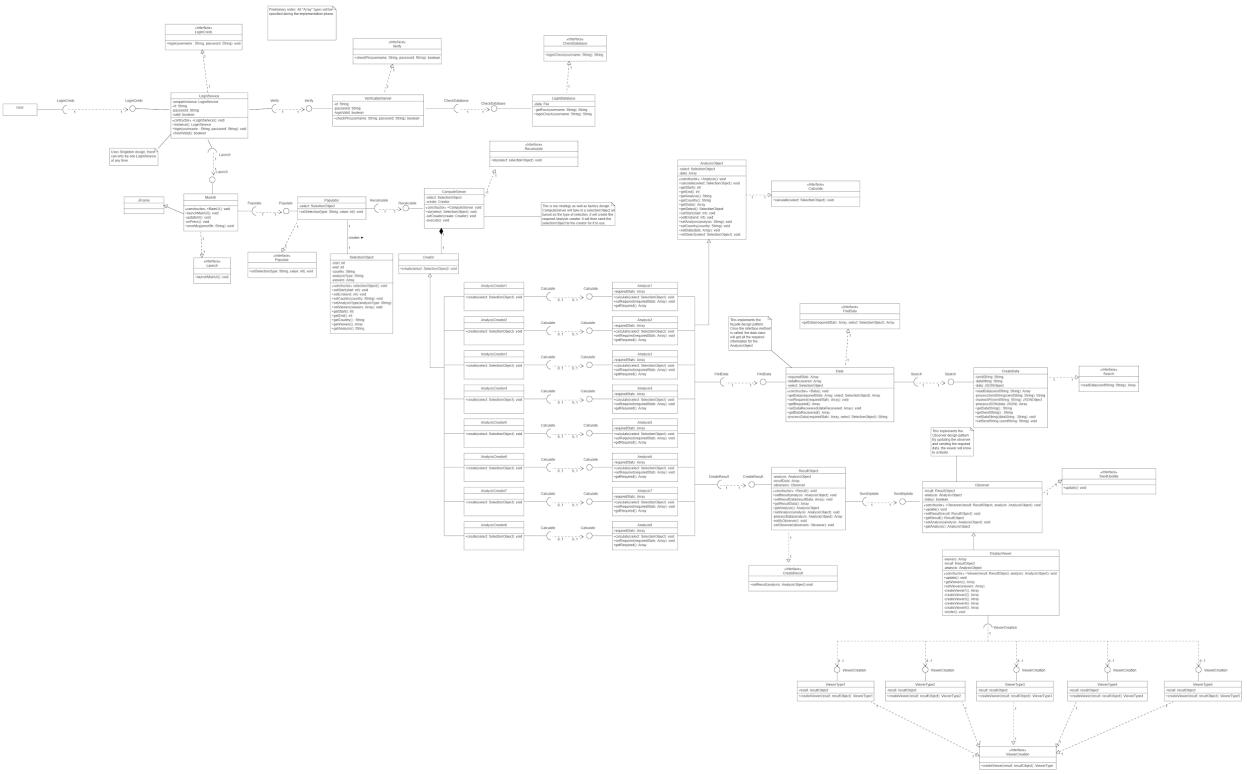
Firstly, there are three main distinct systems that can be observed. There is a user interface layer, an analysis layer, and a backend / database access layer. The user interface layer consists of components such as the front end, selection, and viewers. The middle layer accommodates generic services and handles application execution by directing the construction of analysis classes and retrieval of data. The last layer involves interaction with an external data source, namely the World Bank's API.

Secondly, this backend interaction with World Bank's data repository can be loosely viewed as a transactional database data oriented repository architecture design. Within the API call, the application is limited to one of the four CRUD operations (create, read, update, destroy). Utilizing HTTP requests, we are able to 'read' data from their data repository.

Lastly, the model-view-controller architecture is used with the analysis component. Because there is a notification method, the model is the data/result objects constructed throughout. The controllers are the various analysis classes that are able to construct/change these result objects. The viewer component classes are the views being notified with the MVC.

4 Detailed Class Diagrams

4.1 UML Class Diagrams



Class Diagram for the System Architecture / Design

LoginService	
Method	Functionality
LoginService()	Initializes the login window and begins listening for user input.
instance()	Returns the uniqueInstance variable, if it exists, supporting the singleton design pattern.
login(username, password)	Represents the interface method implemented which allows the user to enter their login credentials.
checkValid()	Utilizes the VerificationServer interface to check if a password is valid.

VerificationServer	
Method	Functionality
checkPin(username, password)	Utilizes the LoginDatabase loginCheck interface method to return the correct user password. Checks the supplied password to the returned password.
applicationExit()	Exits the application when the incorrect credentials are entered.

LoginDatabase	
Method	Functionality
getPass(username)	Queries the plain text file for the correct username password combo and returns the password.
loginCheck(username)	Implements the interface method that is used by VerificationServer.

MainUI	
Method	Functionality
MainUI()	Constructor for MainUI. Will create the front end of the program used for interaction.
launchMainUI()	Tells the MainUI class that the user successfully logged in and it can now render the UI.
updateUI()	Refreshes and updates the UI whenever needed.
onPress()	Checks the component pressed and calls the appropriate methods through various interfaces.
errorMsg(errorStr)	Will show an error message to the user when an error happens.

Populator	
Method	Functionality
setSelection(type, value)	When given the type of selection the user pressed, and what value, we can store that information in the SelectionObject.

SelectionObject	
Method	Functionality
SelectionObject()	Constructor for the SelectionObject. Will instantiate any variable needed on creation.
setStart(start)	Setter method for the start variable. Takes an integer.
setEnd(end)	Setter method for the end variable. Takes an integer.
setCountry(country)	Setter method for the country variable. Takes a String.
setAnalysisType(analysisType)	Setter method for the analysisType variable. Takes a String.
setViewers(viewers)	Setter method for the viewers variable. Takes an Array.
getStart()	Returns the start year variable.
getEnd()	Returns the end year variable.
getCountry()	Returns the country variable.
getViewers()	Returns the selected array of viewers.
getAnalysis()	Returns the analysis type.

ComputeServer	
Method	Functionality
do(select)	Method to handle all control and business logic of the compute server based on a select object.
setCreator(create)	Setter method for the create variable. Takes a Creator object.
execute()	Based on the stored creator class, it will call the creator's create method.

Creator	
Method	Functionality
create(select)	Defines a parent create method with basic functionality. All children classes will override this method.

AnalysisCreator(1..8)	
Method	Functionality
create(select)	Overrides the parent class create method and will construct the appropriate analysis object type.

AnalysisObject	
Method	Functionality
Analysis()	Constructs an analysis object by instantiating all instance variables.
calculate(select)	Defines a basic calculate function that will be overridden in all child classes. This method may begin to process simple data if required.
getStart()	Returns the start variable stored in the SelectionObject.

getEnd()	Returns the end variable stored in the SelectionObject.
getAnalysis()	Returns the analysis variable stored in the SelectionObject.
getCountry()	Return the country variable stored in the SelectionObject.
getData()	Returns the data variable
getSelect()	Returns the SelectionObject stored.
setStart(start)	Setter method for the start variable. Takes an integer
setEnd(end)	Setter method for the end variable. Takes an integer
setAnalysis(analysis)	Setter method for the analysis variable. Takes a String
setCountry(country)	Setter method for the country variable. Takes a String
setData(data)	Setter method for the data variable. Takes an Array
setSelect(select)	Setter method for the select variable. Takes a SelectionObject

Analysis(1..8)	
Method	Functionality
calculate(select)	Processes the SelectionObject to obtain the required information and store them.
setRequired(requiredStats)	Setter method for the requiredStats variable. Takes an Array.
getRequired()	Returns the requiredStats variable.

Data	
Method	Functionality

Data(): void	This is the constructor for the Data class. This will initialize the class on creation.
getData(requiredStats, select)	Implements the Interface method. When given the requiredStats array and a SelectionObject, the data class acts as the façade and gets the data the AnalysisObject requires.
setRequired(requiredStats)	Setter method for the requiredStats variable. Takes an Array.
getRequired()	Returns the requiredStats variable.
setDataRecovered(dataRecovered)	Setter method for the dataRecovered variable. Takes an Array
getDataRecovered()	Returns the dataRecovered variable.
processData(requiredStats, select)	This method will take the requiredStats array and the SelectionObject. Then it will take the required information and format each variable needed (ie. A vs B vs C) with a copy of the required information (years, country). This String will only contain one of the required variables and the information. Multiple String will be made and each String will be sent one by one to the CreateData class.

CreateData	
Method	Functionality
readData(sendString)	This method will take the rough String sent by the Data class and, by using the other methods, will process the String to be sent to the API. This is the implementation of the interface method.
processSendString(sendString)	This Method will take the sendString made by the Data class and process it into a usable URL for the API.
invokeAPI(sendString)	By using the process String, this method will call the API with the usable URL to get the information we need.

processJSON(data)	This method will take the JSON object given by the api and process the information into an Array.
getDataString()	Returns the dataString variable.
getSendString()	Returns the sendString variable.
setDataString(dataString)	Setter method for the dataString variable. Takes a String.
setSendString (sendString)	Setter method for the sendString variable. Takes a String.

ResultObject	
Method	Functionality
Result()	This is the constructor for the Result class. This will initialize the class on creation.
setResult(analysis)	Implements the interface method.
setResultData(resultData)	Setter method for the resultData variable. Takes an Array
getResultData()	Returns the resultData variable.
getAnalysis()	Returns the AnalysisObject stored
setAnalysis(analysis)	Setter method for the analysis variable. Takes an AnalysisObject
processData(analysis)	This method takes the data recovered from the database and formats it to be used by the viewers.
setObserver(observers)	Setter method for the one Observer we have. Takes an Observer Object.
notifyObserver()	Once everything is done, this object will notify any observers connected to it, signalling them to update.

Viewer	
Method	Functionality

Viewer(result, analysis)	Constructor for the Viewer class. It requires a reference to the ResultObject and AnalysisObject when first created.
update()	When the ResultObject notifies all the observers, this Viewer will check the object for updates and will start creating the viewers. Overrides the Parent function.
getViewers()	Returns the array of Viewers
setViewer(viewers)	Setter method for the viewers variable. Takes an Array
createViewer1()	Method for creating the viewer of type 1.
createViewer2()	Method for creating the viewer of type 2.
createViewer3()	Method for creating the viewer of type 3.
createViewer4()	Method for creating the viewer of type 4.
createViewer5()	Method for creating the viewer of type 5.
render()	Loops through the viewers array and renders each viewer object to the main UI.

Observer	
Method	Functionality
Observer(result, analysis)	Constructor for the Observer class. It requires a reference to the ResultObject and AnalysisObject when first created.
update()	Parent function. When called by the ResultObject, the Observer class can do something.
setResult(result)	Setter method for the result variable. Takes a ResultObject
getResult()	Returns the ResultObject stored.

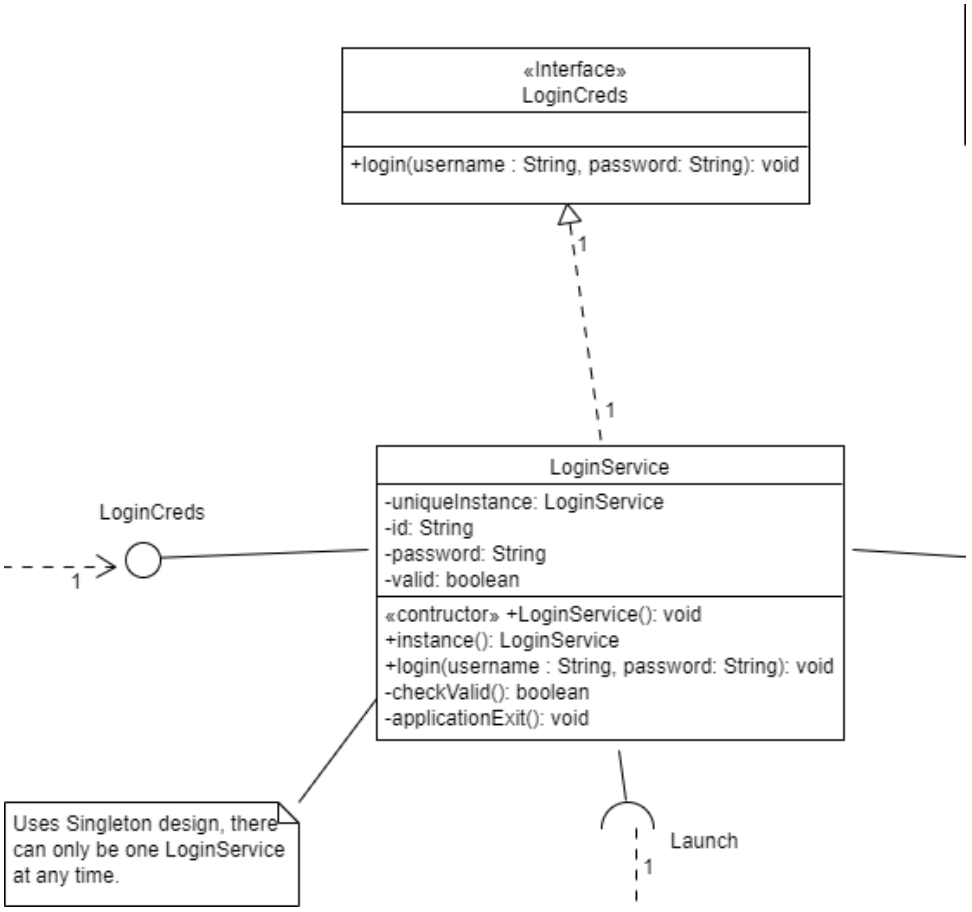
setAnalysis(analysis)	Setter method for the analysis variable. Takes an AnalysisObject
getAnalysis()	Returns the AnalysisObject Stored

ViewerType(1..5)	
Method	Functionality
createViewer(result)	Implements the interface method and creates the specific viewer object utilizing the result object for all the required statistics.

5 Use of Design Patterns

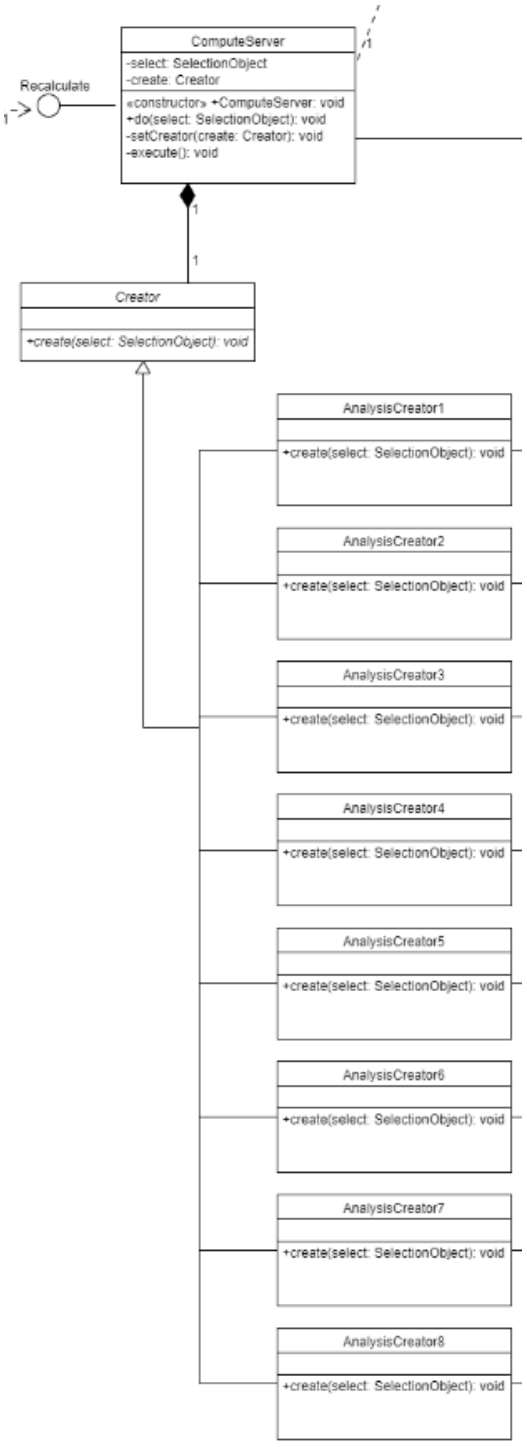
5.1 Singleton Design Pattern

Since the sign-in page is a one-time unique process, the singleton design pattern can be applied. This is done in order to ensure that in the software application, we only construct one object from the login page class. We want to guarantee that the class produces only one object no matter how many times we call the constructor, and we want to have a unique global reference point to this object. This will ultimately assure security and consistency within the application.



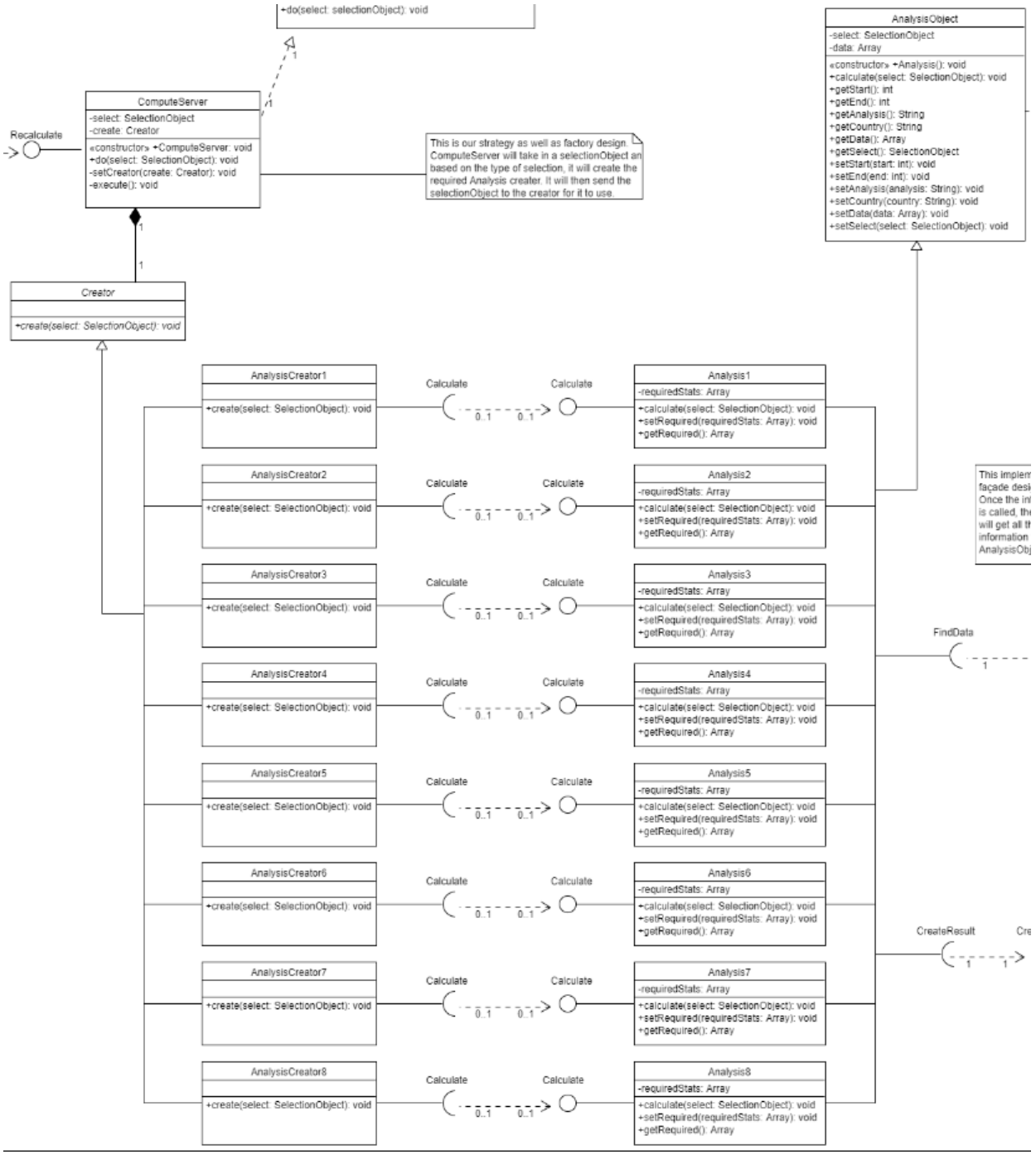
5.2 Strategy Design Pattern

The strategy design directly compliments the factory design. Once the factory has created the analysis object, the strategy design will request the object. Since all of the specific analysis objects have the same parent class, the strategy can request and interact with all the objects. Also, this design allows the object to be passed to other programs when needed, for example, the façade design aspect. By incorporating this design, our program will be much more flexible with the movement of necessary objects and variables.



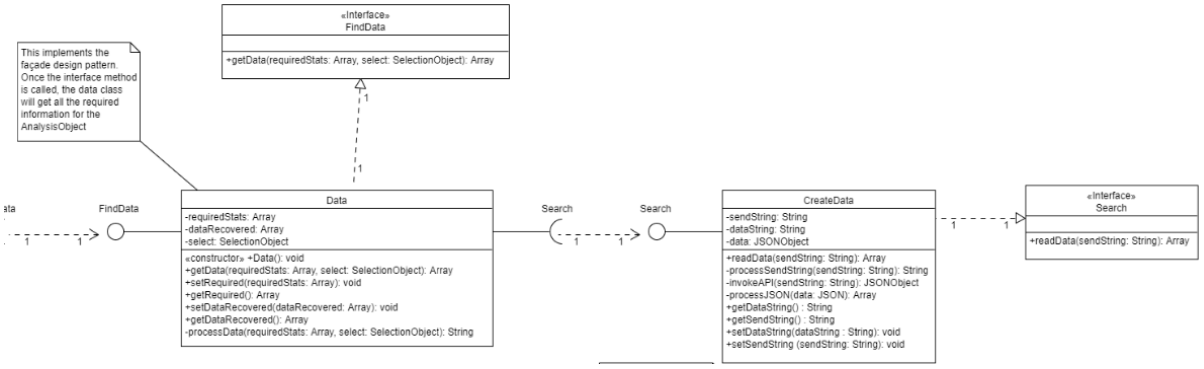
5.3 Factory Design Pattern

Once the user has selected an analysis type, the analysis component of the system will receive the id of the selected analysis. From there, we can apply the factory design pattern. With the given analysis id, we can create the required specified analysis object. This factory design allows for more flexibility as the system develops further. Since there are so many analysis types, it would be cumbersome and inefficient to specify the specific analysis object to construct. By using the factory design pattern utilizing a specific analysis id, the client code will not know what kind of objects are being built.



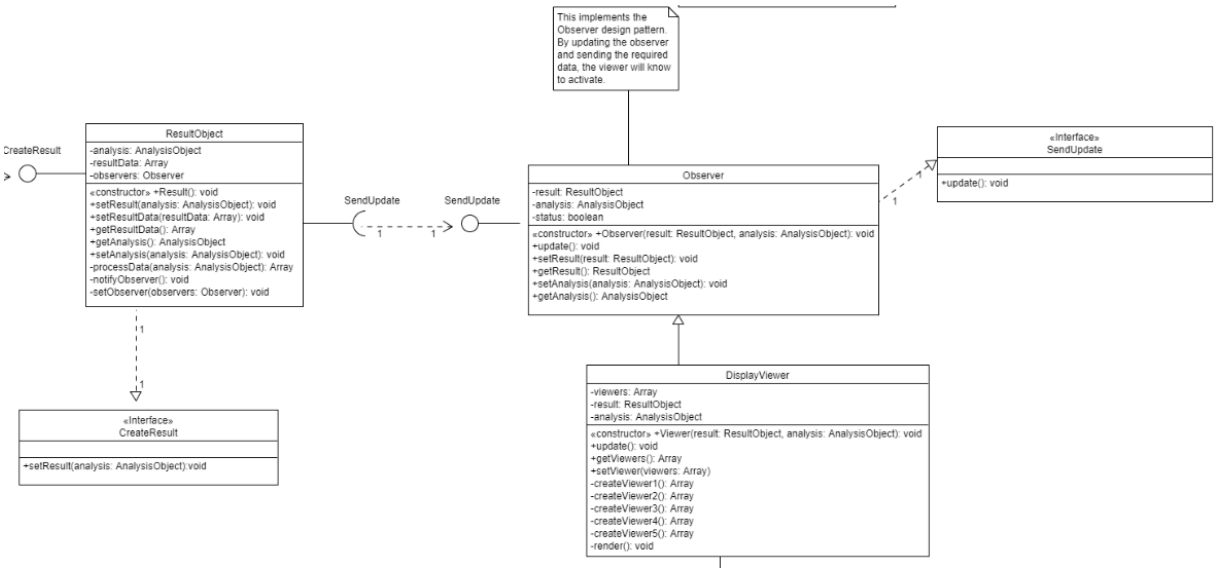
5.4 Façade Design Pattern

When a user selects ‘recalculate’, the retrieval and processing of data must be handled effectively. Although there are many moving parts and subcomponents, the façade design pattern will be used to simplify this process. A simple interface will be exposed that handles all of the underlying logic related to the interaction with data. This includes, but is not limited to, the retrieval, processing, and storing of data. The use of a simplified interface will minimize the dependencies on the underlying subsystem. Thus, if the World Bank Data Repository or some other data component were to change, other parts of the program will not be affected significantly.



5.5 Observer Design Pattern

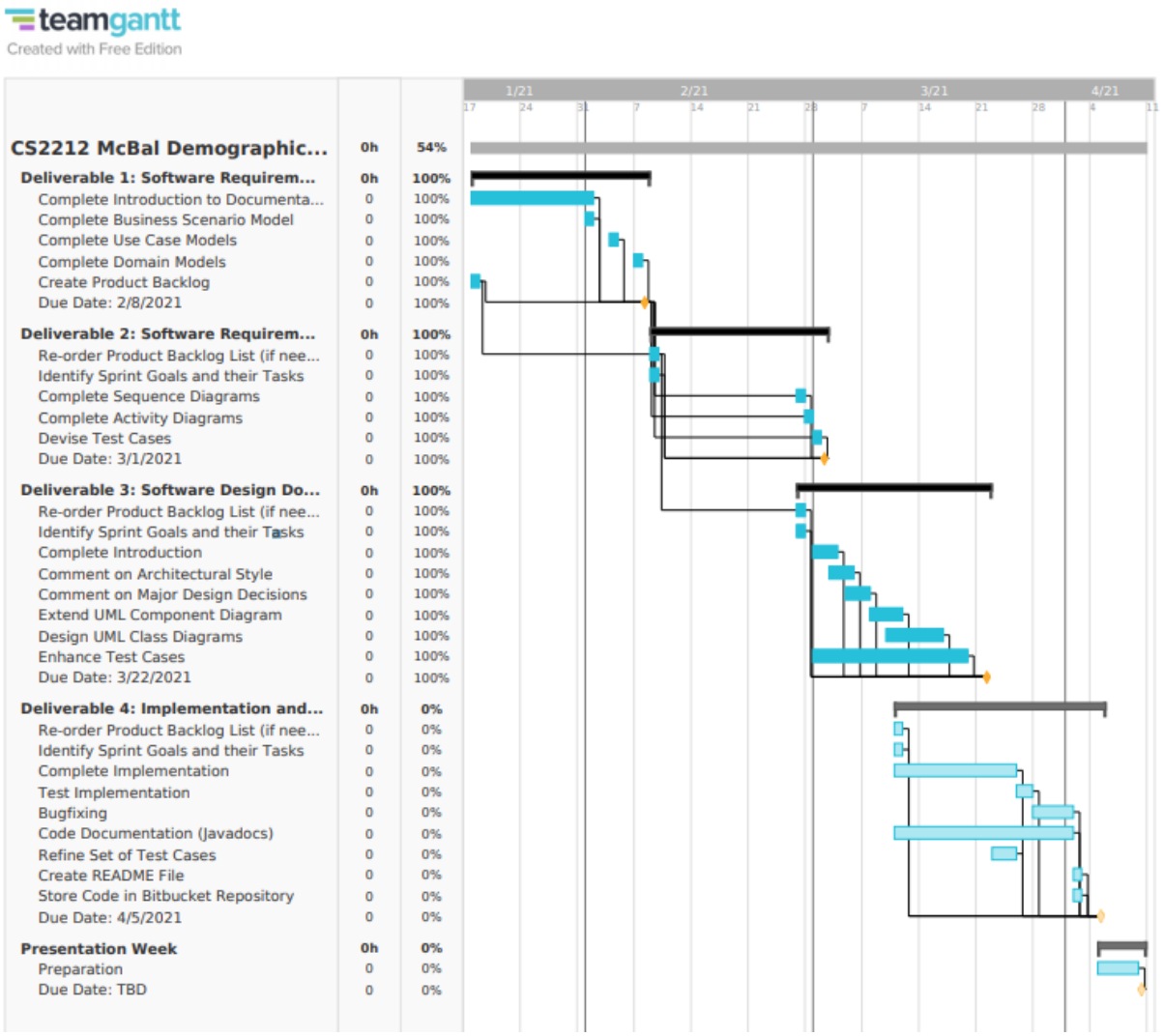
Because a change to the resultObject (its creation) requires the viewer classes to render, this update / notification can be modeled with the observer design pattern. After the instantiation of the resultObject, the class will call the notifyObserver() method and notify the object stored in its observer instance variable. Once notified, the observer will be updated and perform its operations automatically. This is useful because of the extensibility of the pattern. If we need to ever perform more operations or update more observers, it can be easily programmed in.



6 Activities Plan

6.1 Gantt Chart

This Gantt Chart (designed using TeamGantt) designates a tentative plan for the McBal Demographic Analysis Application. It is updated daily to ensure quality project management.



6.2 Project Backlog and Sprint Backlog

1. Create login system
    1.1. Create the LoginService interface
    1.2. Create the LoginService
    1.3. Create the VerificationServer interface
    1.4. Create the VerificationServer
    1.5. Create the CheckDatabase interface
    1.6. Create the LoginDatabase class (using plain text file)
    1.7. Create initial registration system
2. Create main application UI
    2.1. Create the Launch main UI interface
    2.2. Create the UI refresh methods
    2.3. Create the onPress methods
    2.4. Create list of countries selectable
    2.5. Create year drop-down menus
    2.6. Create viewer drop-down menu
    2.7. Implement buttons
3. Create the Populator Component
    3.1. Create the Populator class
    3.2. Implement the SelectionObject class
    3.3. Create interface to populate SelectionObjects
4. Create the Analysis Component
    4.1. Create the compute server class exposing a recalculate interface
    4.2. Create the Factory Method
        4.2.1. Create a Creator class
        4.2.2. Create the 8 analysis creators
        4.2.3. Create a analysis object class exposing a calculate interface
        4.2.4. Create the 8 analysis types
    4.3. Create the Data class exposing a FindData interface
    4.4. Create the ResultObject class exposing CreateResult interface
5. Create data retrieval API component
    5.1. Create CreateData class with ability to make API calls, implementing the Search interface
6. Create the Viewer Components
    6.1. Create the Observer class that implements SendUpdate interface
    6.2. Create the DisplayViewer logic class that handles program execution
    6.3. Create the ViewerCreation interface
    6.4. Create the 5 ViewerType child classes that implement the interface
7. Test application with test cases
8. Debug potential errors/bugs
9. Upload code to BitBucket
10. Create README file

6.3 Group Meeting Logs

Present Group Members	Meeting Date	Issues Discussed / Resolved
Henry So, Jacob Chun, Yan Qing Niu, Samuel Su	March 15/2021 10:00 PM - 2:00 AM	- Discussed the overall plan for the document - Created the main page, introduction, major design decisions - Set meeting up for next date
Henry So, Jacob Chun, Yan Qing Niu, Samuel Su	March 17/2021 6:00 PM - 12:00 AM	- Finished the UML component diagram - Wrote the content and architecture styles for the component diagram section - Started the initial draft of the UML class diagram
Henry So, Jacob Chun, Yan Qing Niu, Samuel Su	March 20/2021 2:00 PM - 6:00 PM	- Class diagrams fully completed - All images from the component diagram and class diagram pasted into document - Edits made to the formatting of the document
Henry So, Jacob Chun, Yan Qing Niu, Samuel Su	March 21/2021 3:00 PM - 6:00 PM	- Completion of the Test Driven Development section - Final edits made before submitting the document.

7 Test Driven Development

Initial test cases will be provided in the form of a table as follows:

Test ID	1.1
Category	Login System / Verification
Requirements Coverage	UC1-Successful-User-Login
Initial Condition	System initiates and runs.
Procedure	1. The User opens the system 2. The User provides a user name to the login system 3. The User provides a password to the login system 4. The verification server checks to see if the user's credentials are valid 5. The verification server invokes the loginCheck method from the LoginDatabase to see if password matches with the given username 6. The User credentials are correct 7. The User is presented with the main UI window
Expected Outcome	Login form closes, and the user is presented with the main UI window
Notes	The User logins successfully and the methods invoked to check the User credentials are login(username,password) from the LoginService class, checkPin(username, password) from the VerificationServer class, and loginCheck(username) from the LoginDatabase class. Once the User log in successfully, the valid instance variable is set to True. Classes involved: LoginService, VerificationServer, LoginDatabase

Test ID	1.2
Category	Login System / Verification
Requirements Coverage	UC1-Unsuccessful-User-Login
Initial Condition	The application initiates and runs.
Procedure	1. The User opens the application 2. The User provides a user name to the login system 3. The User provides a password to the login system 4. The verification server checks to see if the user's credentials are valid 5. The verification server invokes the loginCheck method from the LoginDatabase to see if password matches with the given username

	6. The User credentials are incorrect 7. Incorrect credentials message appears on the screen 8. The application terminates through usage of a method from LoginSystem, applicationExit()
Expected Outcome	Credentials cannot be verified and application terminates.
Notes	The User logins unsuccessfully and the methods invoked to check the User credentials are login(username,password) from the LoginService class, checkPin(username, password) from the VerificationServer class, and loginCheck(username) from the LoginDatabase class. Classes involved: LoginService, VerificationServer, LoginDatabase

Test ID	2.1
Category	Analysis Selection
Requirements Coverage	UC2-Successful-Analysis-Select
Initial Condition	System runs and the user successfully logs in.
Procedure	1. User logged in 2. User selects analysis type
Expected Outcome	The system confirms the user's selection and stores the chosen analysis type.
Notes	Country selection and year selection is dependent on analysis selection. The User populates the SelectionObject. When the analysis type is chosen, the SelectionObject invokes the setAnalysisType(analysisType) method and passes in the analysis type chosen by the User. Classes involved: MainUI, Populator, SelectionObject

Test ID	3.1
Category	Country Selection
Requirements Coverage	UC3-Successful-Country-Select
Initial Condition	System runs and the user successfully logs in. The user has already selected the analysis type.
Procedure	3. User logged in 4. User selected analysis type 5. User selects allowed country 6. System stores country selection
Expected Outcome	The system confirms the user's selection and stores their country. The drop down menu reflects the user's selection.

Notes	Available countries are determined by the engineers. Upon successful country selection, the User populates the SelectionObject. When the country is chosen, the SelectionObject invokes the setCountry(country) method and passes in the country chosen by the User. Classes involved: MainUI, Populator, SelectionObject

Test ID	3.2
Category	Country Selection
Requirements Coverage	UC3-Unsuccessful-Country-Select
Initial Condition	System runs and the user successfully logs in. The user has already selected the analysis type.
Procedure	1. User logged in 2. User selects analysis type 3. User selects unavailable country 4. System prints error message for user
Expected Outcome	System does not allow country to be selected and prints an error message for the user
Notes	Unavailable countries are determined by the engineers. The method errorMsg(errorStr) will be invoked and a window will pop up, alerting the user that the country was unsuccessfully selected. Classes involved: MainUI

Test ID	4.1
Category	Year Selection
Requirements Coverage	UC4-Successful-Year-Select
Initial Condition	System runs and the user successfully logs in. The user has already selected the analysis type and a country.
Procedure	1. User logged in 2. User selects analysis type 3. User selects country successfully 4. User selects start year 5. User selects end year 6. System stores start and end year selection and checks the time period in between to see if it is valid for the analysis type chosen
Expected Outcome	The system confirms the user's selection and stores their start and end year. The drop down menu reflects the user's selection.
Notes	The user successfully populates the variable start and end of the SelectionObject when selecting a suitable time period in which

	analysis is available. Invokes the setStart(start) and setEnd(end) method and passes in the start and end years that the user selects. Classes involved: MainUI, Populator, SelectionObject

Test ID	4.2
Category	Year Selection
Requirements Coverage	UC4-Unsuccessful-Year-Select
Initial Condition	System runs and the user successfully logs in. The user has already selected the analysis type and country.
Procedure	1. User logged in 2. User selects analysis type 3. User selects country successfully 4. User selects start year 5. User selects end year 6. System stores start and end year selection and checks the time period in between to see if it is valid for the analysis type chosen 7. The time period is invalid for the analysis type chosen 8. Error message is displayed 9. User is prompted to reselect a new start year and end year
Expected Outcome	The user is prompted by the system to reselect a new start year and end year that is valid for the analysis type chosen.
Notes	The method errorMsg(errorStr) will be invoked and a window will pop up, alerting the user that the years were unsuccessfully selected. Classes involved: MainUI, Populator

Test ID	5.1
Category	Additional / Removal of Viewers
Requirements Coverage	UC5/6-Successful-Viewers
Initial Condition	System runs and the user has successfully logged in. The user successfully selected analysis, country, start and end years.
Procedure	1. User logged in 2. User selects analysis type 3. User selects country successfully 4. User selects start and end year 5. User selects viewer from drop-down menu 6. User selects to add viewer a. Addition of viewer successful b. Removal of viewer successful
Expected Outcome	If the user adds the viewer successfully, the system should store the type of viewer to be displayed. If the user removes the viewer

	successfully, the system should remove the type of the viewer from storage.
Notes	The user successfully populates the viewers array. Each successive viewer added is put into the array. Upon the population of the viewersArray, variable viewers is set with the method call setViewers(viewers). Classes involved: MainUI, Populator, SelectionObject

Test ID	5.2
Category	Additional / Removal of Viewers
Requirements Coverage	UC5/6-Unsuccessful-Viewers
Initial Condition	System runs and the user has successfully logged in. The user successfully selected analysis, country, start and end years.
Procedure	1. User logged in 2. User selects analysis type 3. User selects country successfully 4. User selects start and end year 5. User selects viewer from drop-down menu 6. User selects to add or remove viewer a. Addition of viewer unsuccessful b. Removal of viewer unsuccessful 7. Error message is displayed
Expected Outcome	If the user adds the viewer unsuccessfully, then the system already has the viewer stored. If the user removes the viewer unsuccessfully, the system does not have the viewer stored yet. In both cases, an error message is displayed -- The method errorMsg(errorStr) will be invoked and a window will pop up, alerting the user that the viewers were unsuccessfully selected.
Notes	Classes involved: MainUI, Populator, SelectionObject

Test ID	7.1
Category	Data Procurement
Requirements Coverage	UC7-Successful-Data-Retrieval
Initial Condition	System runs and the user has successfully logged in. The user successfully selected analysis, country, start and end years.
Procedure	1. User logged in 2. User selects analysis type 3. User selects country successfully 4. User selects start and end year 5. User presses “Recalculate” button 6. System takes selection and requests from Web Service 7. Web Service sends necessary data to System

	8. Data is usable 9. Data is processed to be used 10. Data is stored
Expected Outcome	Based on the selection the user made, the system requests and receives data from the web service to be processed and stored.
Notes	Once the user has populated the SelectionObject fully, the ComputeServer class handles it and implements a factory design based on the type of selection. It will create the required AnalysisCreator, and then build the Analysis(1...8) object based on the analysis type selected. Subsequently, the AnalysisObject goes to the Data class and implements the facade design pattern, where the data from this class will eventually create an object from the CreateData class, where the data requests are made. The CreateData class invokes the World Bank Repository API through the usage of method invokeAPI(sendString) and the data from the JSON file returned is then processed and a ResultObject is made from the contents of the JSON file. Classes involved: MainUI, AnalysisObject, Data, CreateData, ResultObject, Populator

Test ID	7.2
Category	Data Procurement
Requirements Coverage	UC7-Unsuccessful-Data-Retrieval
Initial Condition	System runs and the user has successfully logged in. The user successfully selected analysis, country, start and end years.
Procedure	1. User logged in 2. User selects analysis type 3. User selects country successfully 4. User selects start and end year 5. User presses “Recalculate” button 6. System takes selection and requests from Web Service 7. Web Service sends necessary data to System 8. Data is unusable 9. Display error message, prompt user to reselect choices
Expected Outcome	An error message is displayed to prompt the user to reselect new choices as the analysis cannot be done/ data is unusable.
Notes	Classes involved: MainUI, AnalysisObject, Data, CreateData

Test ID	8.1
Category	Render Results, System UI Update
Requirements Coverage	UC8-Successful-Displaying-Graphs

Initial Condition	System runs and the user has logged in successfully. The user has selected an analysis type, country, start and end years, added viewers, and analysis has been performed on the computed data.
Procedure	1. User logged in 2. User selects analysis type 3. User selects country successfully 4. User selects start and end year 5. User selects viewer from drop-down menu 6. User selects to add/remove viewer 7. User presses the “Recalculate” button. 8. Data from Web Service is received based on selected analysis type, country, start and end years. 9. Data is processed and usable 10. User interface updates, data is rendered onto chosen viewers
Expected Outcome	The analysis is shown in graphical summaries based on viewers selected.
Notes	Taking the ResultObject, the DisplayViewer class creates the viewers based off of user selection, and renders the ResultObject into a graphical chart. The user will recognize that the graphs have been displayed successfully when DisplayViewer invokes update(). The Main UI will then update and the chosen viewers will be displayed on the screen with all the AnalysisObject that the user procured. Classes involved: MainUI, ResultObject, Observer, DisplayViewer, ViewerType(1..5)