COVID-19 Analysis Model

CS2212-W20

created by Group-3

Design Document

Contents

1 Introduction

1.1 Purpose

This document details the design of the system COVID-19 Analysis Model.

This document details the design decisions of a system that allows for retrieving COVID-19 infection related data for a selected set of countries. It details the major design decisions, the architecture of the system through component diagrams and design class diagrams, and specifies the design patterns used for the system that is to be developed. It also addresses the tests that will be used to validate that the system requirements have been met, and the distribution of activities required to complete development of the system through the product and sprint backlogs.

1.2 Overview

The SDD document contains the following information:

1. Major Design Decisions, text describing significant design choices, and modularization criteria.
2. Component Diagram of the system architecture.
3. Detailed Class Diagram showcasing the classes in the system, how they interact with one another, and related interfaces.
4. Product Backlog of system requirements, ranked by priority, that need to be implemented.
5. Sprint Backlog, break down of current Sprint goal into sub tasks required to complete the goal.
6. Test Cases used to validate the implemented code satisfies the requirements of the system.

1.3 References

Project Description -
https://owl.uwo.ca/access/lessonbuilder/item/147446358/group/c9cc47fe-7c6c-49b0-9dd5-247a2977f123/Project%20Resources/CS2212A-Project-Description.pdf

SRS Document -
https://docs.google.com/document/d/1dY1RtnP0JrqbEpwEinEqLjzQrib3aqCXwV1Mlto2WSU/edit?usp=sharing

Covid 19 confirmed cases API -
https://api.covid19api.com/total/dayone/country/%25s/status/confirmed

GANTT - https://www.lucidchart.com/pages/examples/gantt-chart-maker

Domain Model: https://www.lucidchart.com/

Component Diagram -
https://drive.google.com/file/d/1fC8JTmCfzH5YDuS1Xxnz01GSBZz4HNEM/view?usp=sharing

2 Major Design Decisions

Components were compartmentalized and split up into multiple layers. Namely, frontend, backend, and metadata.

This decision allows interchangeability between the components increasing its level of modularity and allowing for a low coupling, high cohesion approach.
For security reasons user access and interactions are limited to the frontend

Assumptions were made regarding the operation of this software:

1. The user must be logged in prior to performing any action
2. The API must be online in order for the data retrieval to operate correctly and data be displayed to the user

Several Design Patterns were utilized:

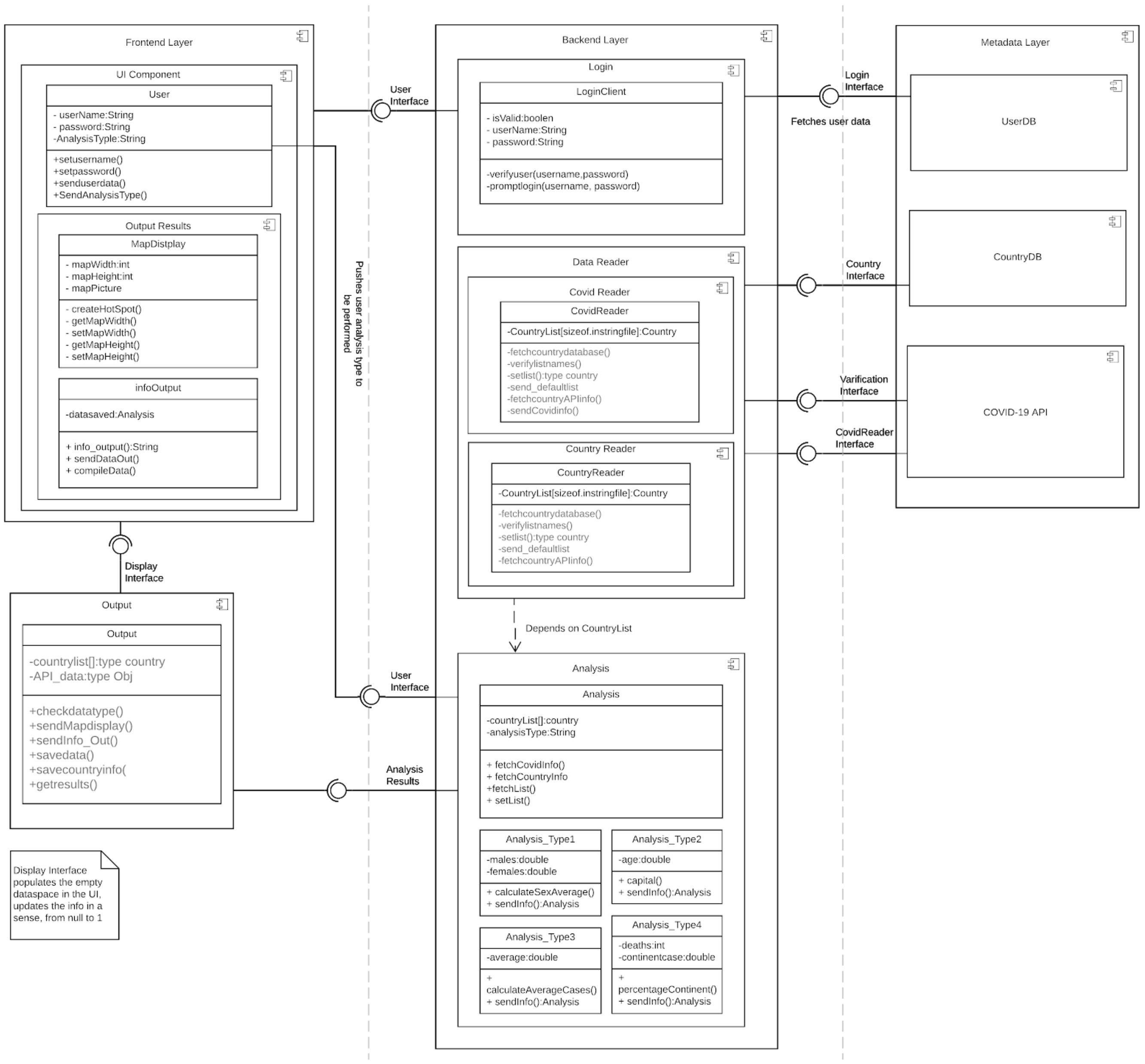
1. The User Component of the UI which is part of the front end uses a proxy design pattern in choosing what information to provide to the login client which is situated in the backend.
2. The analysis component of type selection utilizes a factory design pattern to create objects with the desired specifications in order to perform the analysis.
3. The analysis component where the operations are performed uses a facade design pattern in which the complexity of the analysis calculations are hidden from the user. The user performs an interaction and the results are returned.
4. The UI component uses a singleton design pattern to ensure that only one instance of the user interface is present at any given time.
5. The Analysis Results invocation uses a strategy design pattern in order to choose which analysis type will be performed in order to invoke the correct call. Due to the modularity of the software this design pattern allows for extensibility with regards to different strategies.

3 Architecture

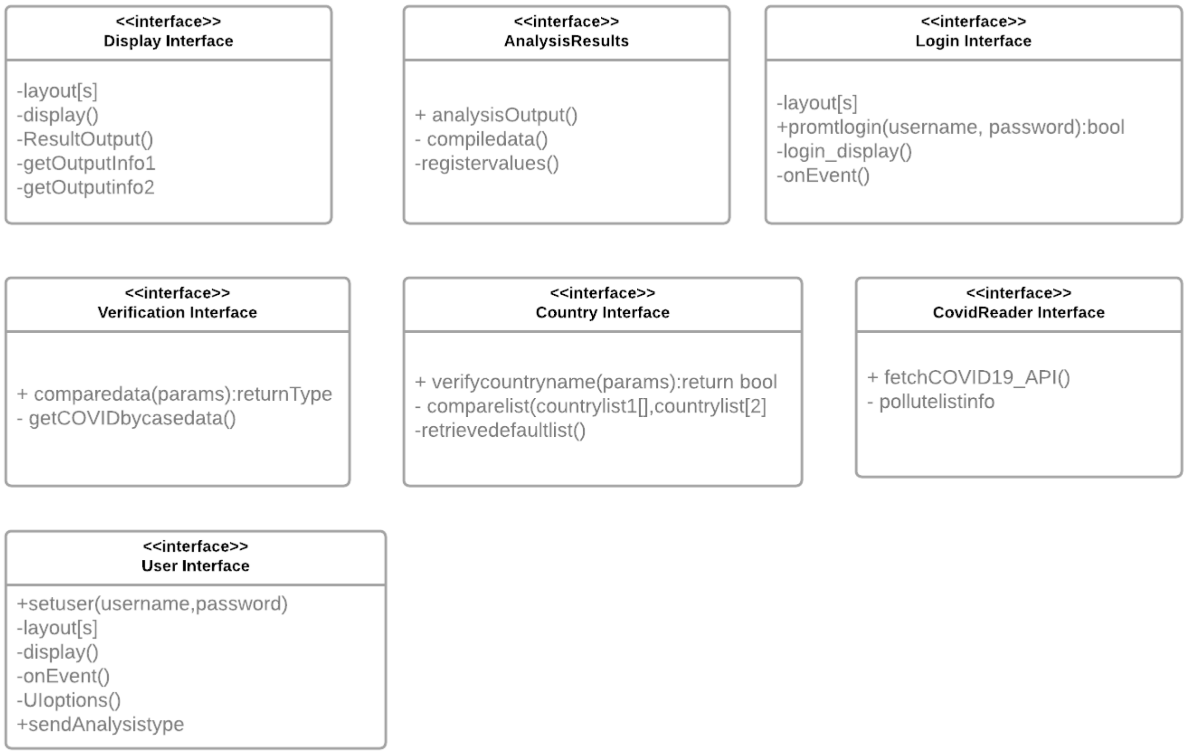
The architectural styles used for this system is a layered component style, where the main components are the Frontend Layer, Backend Layer, and Metadata Layer. This system also uses the Data Abstraction architectural style by having components expose interfaces.

3.1 Component Diagram

(see page below for diagram, or references for high-res version)


Frontend Layer
UI Component
User
- userName:String
- password:String
-AnalysisTyple:String
+setusername()
+setpassword()
+senduserdata()
+SendAnalysisType()
Output Results
MapDistplay
- mapWidth:int
- mapHeight:int
- mapPicture
- createHotSpot()
- getMapWidth()
- setMapWidth()
- getMapHeight()
- setMapHeight()
infoOutput
-datasaved:Analysis
+ info_output():String
+ sendDataOut()
+ compileData()
Display Interface
Output
Output
-countrylist[]:type country
-API_data:type Obj
+checkdatatype()
+sendMapdisplay()
+sendInfo_Out()
+savedata()
+savecountryinfo(
+getresults()
Display Interface populates the empty dataspace in the UI, updates the info in a sense, from null to 1
User Interface
Pushes user analysis type to be performed
User Interface
Analysis Results
Backend Layer
Login
LoginClient
- isValid:boolen
- userName:String
- password:String
-verifyuser(username,password)
-promptlogin(username, password)
Data Reader
Covid Reader
CovidReader
-CountryList[sizeof.instringfile]:Country
-fetchcountrydatabase()
-verifylistnames()
-setlist():type country
-send_defaultlist
-fetchcountryAPIinfo()
-sendCovidinfo()
Country Reader
CountryReader
-CountryList[sizeof.instringfile]:Country
-fetchcountrydatabase()
-verifylistnames()
-setlist():type country
-send_defaultlist
-fetchcountryAPIinfo()
Depends on CountryList
Analysis
Analysis
-countryList[]:country
-analysisType:String
+ fetchCovidInfo()
+ fetchCountryInfo
+fetchList()
+ setList()
Analysis_Type1
-males:double
-females:double
+ calculateSexAverage()
+ sendInfo():Analysis
Analysis_Type2
-age:double
+ capital()
+ sendInfo():Analysis
Analysis_Type3
-average:double
+ calculateAverageCases()
+ sendInfo():Analysis
Analysis_Type4
-deaths:int
-continentcase:double
+ percentageContinent()
+ sendInfo():Analysis
Login Interface
Fetches user data
Country Interface
Varification Interface
CovidReader Interface
Metadata Layer
UserDB
CountryDB
COVID-19 API

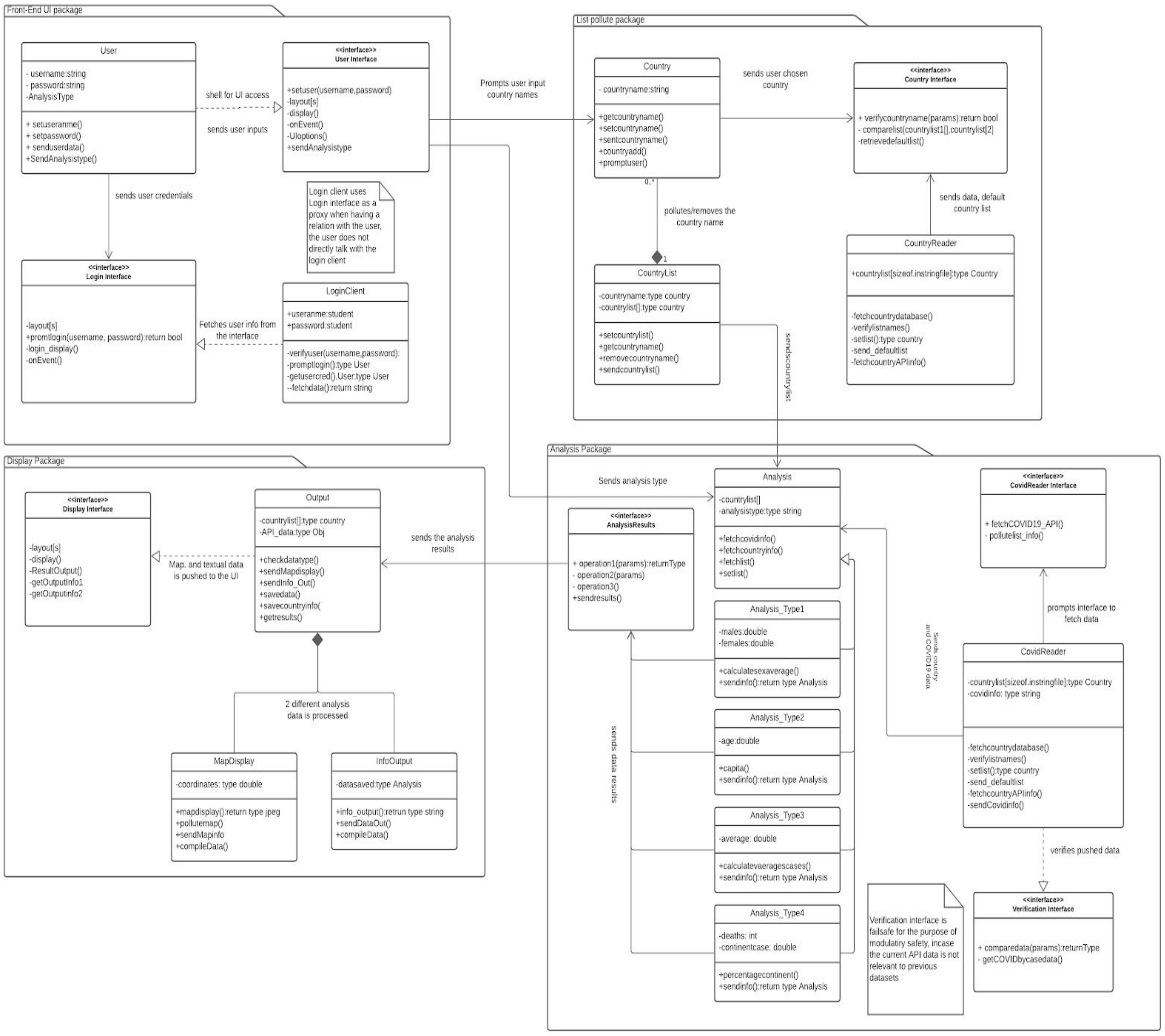
3.2 Component Interfaces



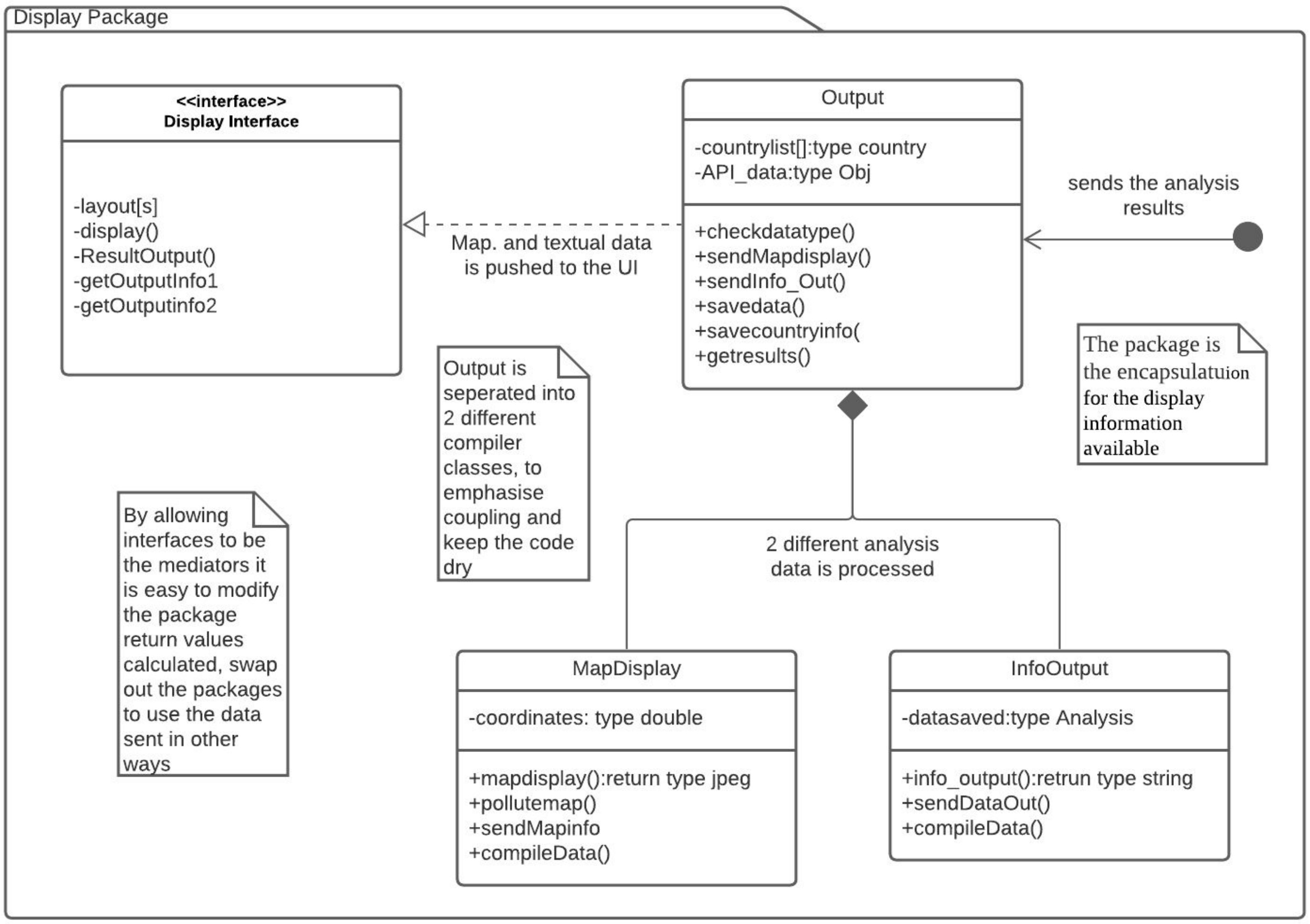
4 Detailed Class Diagrams

4.1 UML Class Diagrams

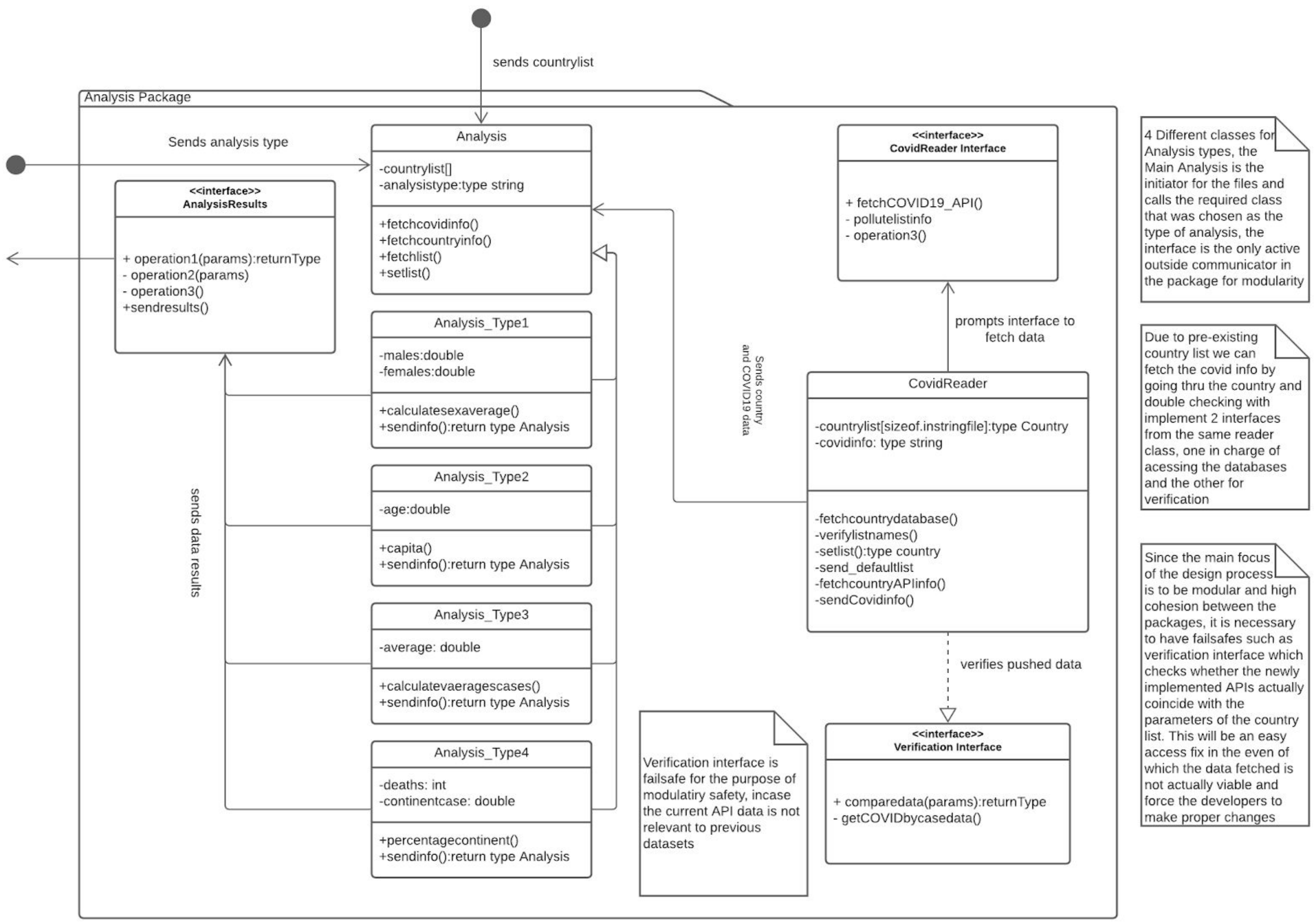
(see page below for diagram)


Front-End UI package
User
- username:string
- password:string
-AnalysisType
+ setuseranme()
+ setpassword()
+ senduserdata()
+SendAnalysistype()
shell for UI access
sends user inputs
<<interface>>
User Interface
+setuser(username,password)
-layout[s]
-display()
-onEvent()
-UIoptions()
+sendAnalysistype
sends user credentials
Login client uses Login interface as a proxy when having a relation with the user, the user does not directly talk with the login client
<<interface>>
Login Interface
-layout[s]
+promtlogin(username, password):return bool
-login_display()
-onEvent()
Fetches user info from the interface
LoginClient
+useranme:student
+password:student
-verifyuser(username,password):
-promptlogin():type User
-getusercred().User:type User
--fetchdata():return string
Prompts user input country names
List pollute package
Country
- countryname:string
+getcountryname()
+setcountryname()
+sentcountryname()
+countryadd()
+promptuser()
sends user chosen country
<<interface>>
Country Interface
+ verifycountryname(params):return bool
- comparelist(countrylist1[],countrylist[2]
-retrievedefaultlist()
0..*
pollutes/removes the country name
1
sends data, default country list
CountryList
-countryname:type country
-countrylist[]:type country
+setcountrylist()
+getcountryname()
+removecountryname()
+sendcountrylist()
CountryReader
+countrylist[sizeof.instringfile]:type Country
-fetchcountrydatabase()
-verifylistnames()
-setlist():type country
-send_defaultlist
-fetchcountryAPIinfo()
sendscountrylist
Display Package
<<interface>>
Display Interface
-layout[s]
-display()
-ResultOutput()
-getOutputInfo1
-getOutputinfo2
Map. and textual data is pushed to the UI
Output
-countrylist[]:type country
-API_data:type Obj
+checkdatatype()
+sendMapdisplay()
+sendInfo_Out()
+savedata()
+savecountryinfo(
+getresults()
2 different analysis data is processed
MapDisplay
-coordinates: type double
+mapdisplay():return type jpeg
+pollutemap()
+sendMapinfo
+compileData()
InfoOutput
-datasaved:type Analysis
+info_output():retrun type string
+sendDataOut()
+compileData()
sends the analysis results
Analysis Package
Sends analysis type
Analysis
-countrylist[]
-analysistype:type string
+fetchcovidinfo()
+fetchcountryinfo()
+fetchlist()
+setlist()
<<interface>>
AnalysisResults
+ operation1(params):returnType
- operation2(params)
- operation3()
+sendresults()
sends data results
Analysis_Type1
-males:double
-females:double
+calculatesexaverage()
+sendinfo():return type Analysis
Analysis_Type2
-age:double
+capita()
+sendinfo():return type Analysis
Analysis_Type3
-average: double
+calculatevaeragescases()
+sendinfo():return type Analysis
Analysis_Type4
-deaths: int
-continentcase: double
+percentagecontinent()
+sendinfo():return type Analysis
<<interface>>
CovidReader Interface
+ fetchCOVID19_API()
- pollutelist_info()
prompts interface to fetch data
Sends country and COVID19 data
CovidReader
-countrylist[sizeof.instringfile]:type Country
-covidinfo: type string
-fetchcountrydatabase()
-verifylistnames()
-setlist():type country
-send_defaultlist
-fetchcountryAPIinfo()
-sendCovidinfo()
verifies pushed data
Verification interface is failsafe for the purpose of modulatiry safety, incase the current API data is not relevant to previous datasets
<<interface>>
Verification Interface
+ comparedata(params):returnType
- getCOVIDbycasedata()

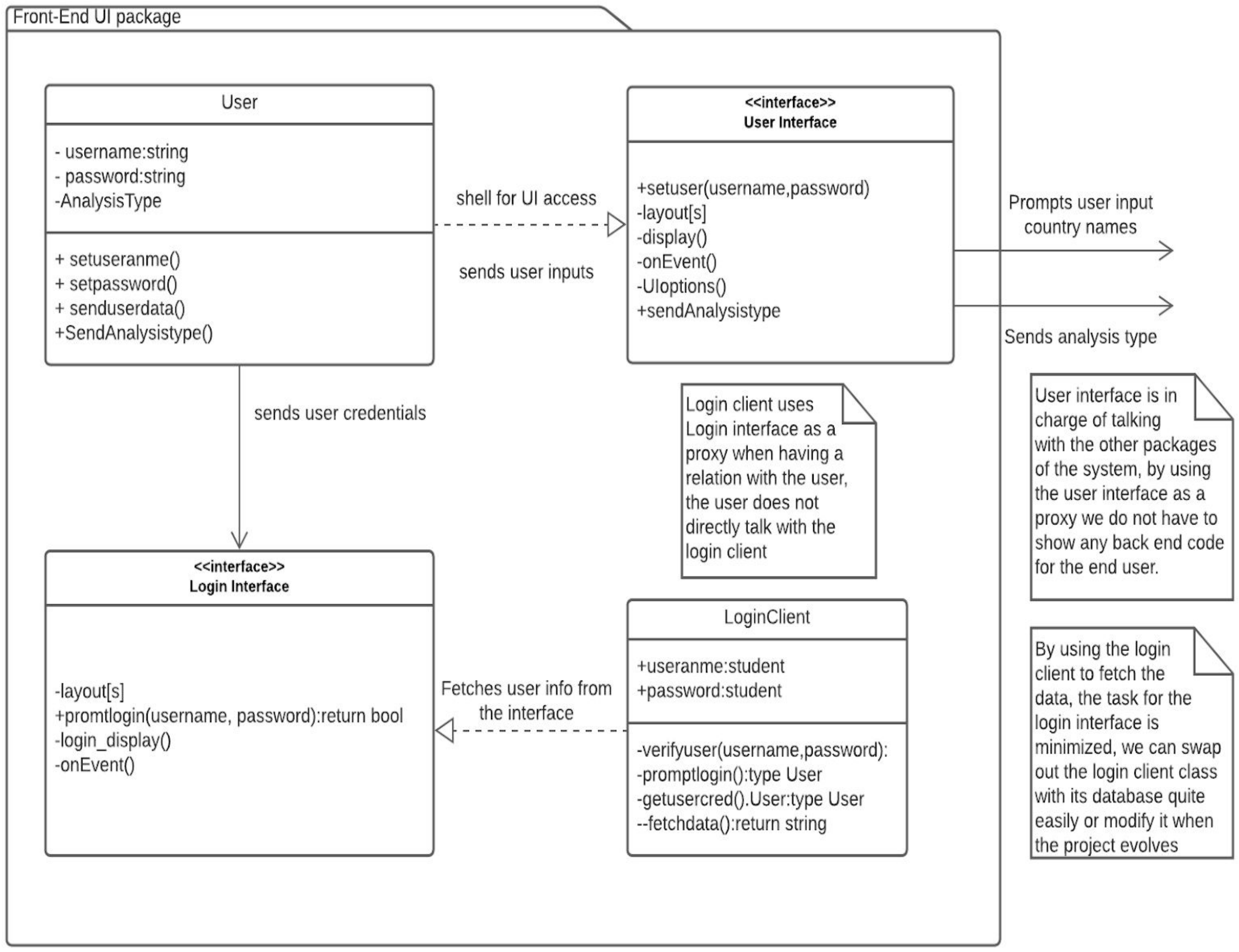
4.1.1 Display Package


Display Package
<<interface>>
Display Interface
-layout[s]
-display()
-ResultOutput()
-getOutputInfo1
-getOutputinfo2
Map. and textual data is pushed to the UI
Output
-countrylist[]:type country
-API_data:type Obj
+checkdatatype()
+sendMapdisplay()
+sendInfo_Out()
+savedata()
+savecountryinfo(
+getresults()
sends the analysis results
The package is the encapsulatuion for the display information available
Output is seperated into 2 different compiler classes, to emphasise coupling and keep the code dry
By allowing interfaces to be the mediators it is easy to modify the package return values calculated, swap out the packages to use the data sent in other ways
2 different analysis data is processed
MapDisplay
-coordinates: type double
+mapdisplay():return type jpeg
+pollutemap()
+sendMapinfo
+compileData()
InfoOutput
-datasaved:type Analysis
+info_output():retrun type string
+sendDataOut()
+compileData()

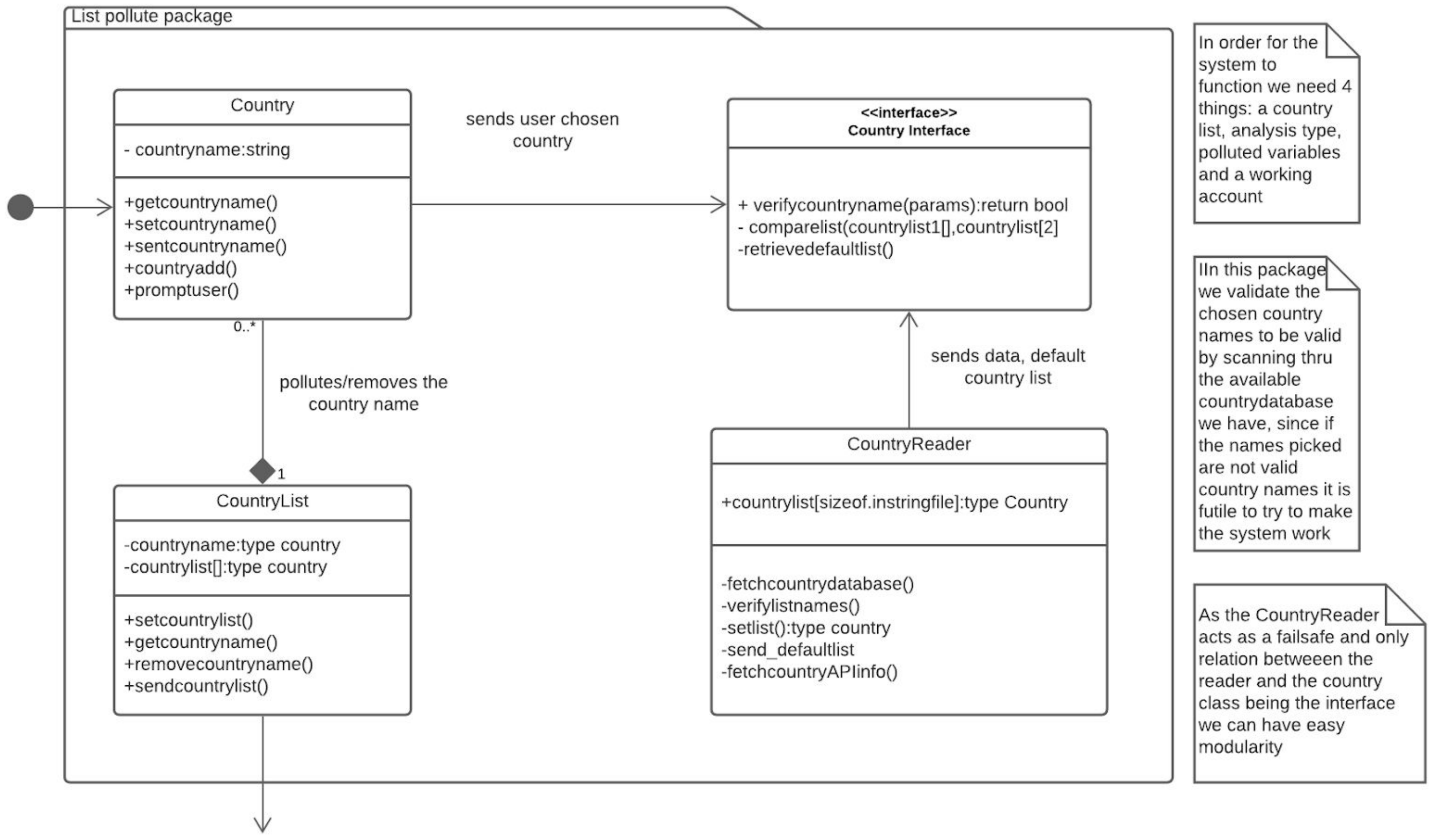
4.1.2 Analysis Package


sends countrylist
Analysis Package
Sends analysis type
Analysis
-countrylist[]
-analysistype:type string
+fetchcovidinfo()
+fetchcountryinfo()
+fetchlist()
+setlist()
<<interface>>
AnalysisResults
+ operation1(params):returnType
- operation2(params)
- operation3()
+sendresults()
Analysis_Type1
-males:double
-females:double
+calculatesexaverage()
+sendinfo():return type Analysis
Analysis_Type2
-age:double
+capita()
+sendinfo():return type Analysis
Analysis_Type3
-average: double
+calculatevaeragescases()
+sendinfo():return type Analysis
Analysis_Type4
-deaths: int
-continentcase: double
+percentagecontinent()
+sendinfo():return type Analysis
sends data results
Sends country and COVID19 data
<<interface>>
CovidReader Interface
+ fetchCOVID19_API()
- pollutelistinfo
- operation3()
prompts interface to fetch data
CovidReader
-countrylist[sizeof.instringfile]:type Country
-covidinfo: type string
-fetchcountrydatabase()
-verifylistnames()
-setlist():type country
-send_defaultlist
-fetchcountryAPIinfo()
-sendCovidinfo()
verifies pushed data
<<interface>>
Verification Interface
+ comparedata(params):returnType
- getCOVIDbycasedata()
Verification interface is failsafe for the purpose of modulatiry safety, incase the current API data is not relevant to previous datasets
4 Different classes for Analysis types, the Main Analysis is the initiator for the files and calls the required class that was chosen as the type of analysis, the interface is the only active outside communicator in the package for modularity
Due to pre-existing country list we can fetch the covid info by going thru the country and double checking with implement 2 interfaces from the same reader class, one in charge of acessing the databases and the other for verification
Since the main focus of the design process is to be modular and high cohesion between the packages, it is necessary to have failsafes such as verification interface which checks whether the newly implemented APIs actually coincide with the parameters of the country list. This will be an easy access fix in the even of which the data fetched is not actually viable and force the developers to make proper changes

4.1.3 Front-End UI package


Front-End UI package
User
- username:string
- password:string
-AnalysisType
+ setuseranme()
+ setpassword()
+ senduserdata()
+SendAnalysistype()
shell for UI access
sends user inputs
<<interface>>
User Interface
+setuser(username,password)
-layout[s]
-display()
-onEvent()
-UIoptions()
+sendAnalysistype
Prompts user input country names
Sends analysis type
sends user credentials
Login client uses Login interface as a proxy when having a relation with the user, the user does not directly talk with the login client
User interface is in charge of talking with the other packages of the system, by using the user interface as a proxy we do not have to show any back end code for the end user.
<<interface>>
Login Interface
-layout[s]
+promtlogin(username, password):return bool
-login_display()
-onEvent()
Fetches user info from the interface
LoginClient
+useranme:student
+password:student
-verifyuser(username,password):
-promptlogin():type User
-getusercred().User:type User
--fetchdata():return string
By using the login client to fetch the data, the task for the login interface is minimized, we can swap out the login client class with its database quite easily or modify it when the project evolves

4.1.4 List Populate Package


List pollute package
Country
- countryname:string
+getcountryname()
+setcountryname()
+sentcountryname()
+countryadd()
+promptuser()
sends user chosen country
<<interface>>
Country Interface
+ verifycountryname(params):return bool
- comparelist(countrylist1[],countrylist[2]
-retrievedefaultlist()
0..*
pollutes/removes the country name
1
sends data, default country list
CountryList
-countryname:type country
-countrylist[]:type country
+setcountrylist()
+getcountryname()
+removecountryname()
+sendcountrylist()
CountryReader
+countrylist[sizeof.instringfile]:type Country
-fetchcountrydatabase()
-verifylistnames()
-setlist():type country
-send_defaultlist
-fetchcountryAPIinfo()
In order for the system to function we need 4 things: a country list, analysis type, polluted variables and a working account
IIn this package we validate the chosen country names to be valid by scanning thru the available countrydatabase we have, since if the names picked are not valid country names it is futile to try to make the system work
As the CountryReader acts as a failsafe and only relation betweeen the reader and the country class being the interface we can have easy modularity

4.2 UML Class Diagrams Discussion

Since the system is partmentalized, it is easy to modify or even replace the packages for easy upgrading as well as modification, it is easy to find issues with the system and the relation between the classes are a proxy between the interfaces they implement.

Overall the system has high potential for modularity and a foundation for an evolving software.

5 Use of Design Patterns

The use of the **Singleton Design Pattern** is evident in the UI component of the Frontend layer. The code ensures that this class only has one instance and it provides access to that static instance via the method getInstance().
This design pattern is very important as only one user interface can be instantiated at any given time.

The use of the **Proxy Design Pattern** is evident in the User component of the UI located in the frontend layer. This is used in the choosing of what information to provide to the login client which is situated in the backend. The proxy design pattern is used as a structural design pattern. It is crucial for our software as it provides a way of keeping a reference to an object and will be used as a way to authenticate the user and provide access controls. The proxy will be able to communicate between the layers and provide substitute objects. This will be useful for relaying information while performing authentication or providing enhanced security to the system. The proxy class also serves the purpose of not requiring the authentication to complete before presenting any information to the user and allows for a reduced memory footprint.

The use of the **Factory Design Pattern** is evident in the analysis component with regards to the method of how analysis objects are created. This design pattern allows for the easy creation of objects without exposing the creation logic to the client and is able to use a common interface to refer and interact with the created object. For our software we have opted to use this in order to create the objects needed to perform analysis on the list of selected countries. Each analysis method requires its own object with specific attributes and an AnalysisFactory can be utilized in order to get Analysis objects. Our user interface uses the factory to create and retrieve the correct Analysis object to utilize. Once it has been created and the correct object has been returned the proper analysis can be done on the data.

The use of the **Facade Design Pattern** is evident in the analysis component with regards to the execution of the analysis. This pattern is used as a way of placing all the complex method calls and code related to performing the analysis into a single class in which the user is able to interact with.This simplifies the interface that the client sees and keeps them from being privy to the underlying logic and behind-the-scenes work being performed. Multiple calls are being handled through the facade such as the retrieval of the analysis object that was selected and created and the information returned from the API. The user performs an interaction with the user interface and the results are returned and displayed accordingly.

The use of the **Strategy Design Pattern** is evident in the Analysis component and the use of the AnalysisResults interface. This interface is responsible for receiving and sending the results that came from the analysis performed. This design pattern is critical in providing the modularity of the software. It allows a wide variety of strategies to be used which would be able to adapt the software to different scenarios. This pattern allows us to create objects with varying contexts and executes algorithms that are able to be interchanged if needed.

6 Activities Plan

6.1 Product Backlog

Backlog Item	**Estimate**
As a user I want to add new countries to the list of countries to perform analysis of.	12
As a user I want to remove countries from the list of countries to perform analysis of.	10
As a user I want to select the analysis type to perform.	13
As a user I want to initiate the analysis and see the results displayed.	18
Create additional tests for implemented code.	11
Total Hours	**64**

6.2 Sprint Backlog

Current Sprint Goal: Allow the user to log into the system.

Tasks	**Mon**	**Tues**	**Wed**	**Thur**	**Fri**
Code the user class + user interface	3	2			
Code the login client class + login interface	3	2			
Start debugging		2	2		

Compile and test the test cases	1	1	1		
Re-evaluate necessary code				2	
Finalize the user login panel					3

6.3 Group Meeting Logs

Members	**Date/Length**	**Issues Discussed/Resolved**
All	Nov 17th/30mins	Split up work for deliverable 3
All	Nov 18th/60mins	Discussed potential architecture styles
All	Nov 19th/30mins	Reviewed class diagram work
Jake, Daniel	Nov 19th/30mins	Discussed applicable design patterns
Jake, Mete	Nov 19th/60 mins	Adjustments required for components diagram
All	Nov 19th/30 mins	Created pointers for members to follow, detailed changes were addressed.

6.4 Member assignments/responsibilities

Jake B: Architecture, Component Diagram, assisted with Detailed Class Diagrams, Product backlog, Sprint Backlog, Tests cases, Introduction section.

Daniel R: Design patterns, Major decisions section ,Test cases, Product Backlog, assisted with Component Diagrams.

Mete I: Detailed Class Diagrams and all related Packages, assisted with Components Diagram, Sprint Backlog. UML Class Diagram Discussion, Notes added for the Class Diagram/Components Diagram.

7 Test Driven Development

7.1 Test Cases

7.1.1 TC01a - Successful user login

Test ID	**TC01a**
Category	Evaluation of user credentials stored in file
Requirements Coverage	UC1-Successful-User-Login
Initial Condition	The user has valid credentials, and the system must be running.
Procedure	1. The user selects login 2. The user provides a user name 3. The user provides a password 4. The user logs-in into the system and is presented with the main UI window
Expected Outcome	The login form closes, and the user is presented with the main UI window
Notes	

7.1.2 TC01b - Unsuccessful user login

Test ID	**TC01b**
Category	Evaluation of user credentials stored in file
Requirements Coverage	UC1-Unsuccessful-User-Login
Initial Condition	The user has invalid credentials, and the system must be running.
Procedure	1. The user selects login 2. The user provides a user name 3. The user provides a password 4. The attempts to login and a popup window notifies them there was an error with the provided credentials and the application will terminate

Expected Outcome	The user is notified that their credentials are invalid and the application terminates.
Notes	

7.1.3 TC02a - Successfully adding a country

Test ID	**TC02a**
Category	evaluation of user-added countries stored on file or DB
Requirements Coverage	UC2-Successful-Country-Added
Initial Condition	The user has successfully logged on to the system and is presented with the main application UI
Procedure	1. User Types in country name / selects country from dropdown list 2. User selects '+' button to add country 2.1. The user input is validated by checking if the country is in the Country Database 3. The country is displayed in a panel of the list of countries to be analyzed
Expected Outcome	The textfield or country dropdown closes and the country appears on the country list
Notes	The user should only be able to add valid countries

7.1.4 TC02b - Unsuccessfully adding a country

Test ID	**TC02b**
Category	evaluation of user-added countries stored on file or DB
Requirements Coverage	UC2-Unsuccessful-Country-Added
Initial Condition	The user has successfully logged on to the system and is presented with the main application UI

Procedure	1. User Types in country name / selects country from dropdown list 2. User selects '+' button to add country 2.1. The user input is validated by checking if the country is in the Country Database 3. The country is not displayed in the list and an error message is shown instead
Expected Outcome	The textfield or country dropdown closes and an error message is displayed to the user
Notes	The user should only be able to add valid countries

7.1.5 TC03a - Successfully removing a country

Test ID	**TC03a**
Category	evaluation of user-added countries stored on file or DB
Requirements Coverage	UC3-Successful-Country-Removed
Initial Condition	The user successfully added a country to the list of countries to analyze
Procedure	1. User Types in country name / selects country from dropdown list 2. User selects '-' button to remove country 2.1 The user selection is validated by checking that the country is in the list of selected countries 2.2 The user input is validated by checking that the country is in the country database 3. The country is removed from the list of selected countries for analysis
Expected Outcome	The country disappears from the list of countries to analyze
Notes	The user should only be able to remove a country that is already added

7.1.6 TC03b - Unsuccessfully adding a country

Test ID	**TC03b**

Category	Evaluation of user-added countries stored on file or DB
Requirements Coverage	UC2-Unsuccessful-Country-Removed
Initial Condition	The user has successfully added a country to the list of countries to analyze
Procedure	1. User Types in country name / selects country from dropdown list 2. User selects '-' button to remove country 2.1 The user selection is validated by checking that the country is in the list of selected countries 2.2 The user input is validated by checking that the country is in the country database 3. The country is removed from the list of selected countries for analysis
Expected Outcome	An error message appears indicating that the country could not be removed from the list
Notes	The user should only be able to remove a country that is already added

7.1.7 TC04 - Selecting an analysis type

Test ID	**TC04**
Category	Selecting an analysis type for a list of countries
Requirements Coverage	UC4-Selecting-Analysis-Type
Initial Condition	The user has added countries to the list of countries that analysis will be performed on
Procedure	1. User clicks on the dropdown list to view the available types of analysis 2. User selects the analysis they wish to perform from the dropdown 3. System checks that this analysis type is valid 4. The analysis type choice is complete
Expected Outcome	The user has selected a specific type of analysis from the dropdown and the "recalculate" button is highlighted.

Notes	The user can only select analyses shown in the dropdown.

7.1.8 TC05a - Successfully performing analysis

Test ID	**TC05a**
Category	Performing an analysis on a list of countries
Requirements Coverage	UC5-Successful-Performing-Analysis
Initial Condition	The user has selected an analysis type
Procedure	1. The user presses the "recalculate" button 1.1. Call to analysis system 1.2. Initiate associated analysis function 1.3. Analysis system accesses Covid19 API to retrieve data for the list of countries 1.3.1. System validates that data received can be used for analysis 1.4. Analysis System accesses the Country database to retrieve information regarding age/population/ect. 1.5. Analysis on data is performed by analysis function
Expected Outcome	Results of the analysis are returned from the analysis function.
Notes	

7.1.9 TC05b - Unsuccessfully performing analysis

Test ID	**TC05b**
Category	Performing an analysis on a list of countries
Requirements Coverage	UC5-Unsuccessful-Performing-Analysis
Initial Condition	The user has selected an analysis type

Procedure	1. The user presses the "recalculate" button 1.1. Call to analysis system 1.2. Initiate associated analysis function 1.3. Analysis system accesses Covid19 API to retrieve data for the list of countries 1.3.1. System validates the data received and returns that some or all data is invalid 1.4. Message displayed to user that some or all of the data required for the selected analysis is unavailable
Expected Outcome	User is notified by a pop up message that there was an error performing the analysis.
Notes	

7.1.10 TC06a - Successfully displaying results

Test ID	**TC06a**
Category	Displaying results of analysis
Requirements Coverage	UC6-Successful-Results-Displayed
Initial Condition	The calculations of the database information are compiled properly with accurate data
Procedure	1. UI displays results 1. 1 Visual map information is displayed regarding the chosen countries 1.2 Output menu is filled with logistic information regarding the chosen countries
Expected Outcome	The information retrieved is displayed on the map and information summary is displayed to the user
Notes	

7.1.11 TC06b - Unsuccessfully Displaying Results

Test ID	**TC06b**
Category	Displaying results of analysis

Requirements Coverage	UC6-Unsuccessful-Results-Displayed
Initial Condition	The calculations of the database information are compiled properly with accurate data
Procedure	1. 1UI displays results 1.1 Visual map information is displayed regarding the chosen countries 1.2 Output menu is filled with logistic information regarding the chosen countries
Expected Outcome	The information retrieved is not displayed in graphical or text format and an error message appears to the user
Notes	