Assignment Cover Sheet

EE417 – Group Assignment Report

Student Names:	Mark McGrath - 21210034
	Remi Bertogliati - 20108362
	Rui Fan - 20214169
	Michael O'Hara - 16414554
	Ciaran McCormac - 87198584
	Daire Cooney - 18108784
	Matthew Leboeuf - 20106963
	Corentin LeStanc - 20108362

Assignment:	Group Assignment
Project Name:	Group F
Date Submitted:	25/04/2021

Table of Contents

1. **Executive Summary**

The aim of the project was to build a complete web application incorporating full stack development (front, middle and backend tiers) and to provide comprehensive project documentation.

A banking type application was chosen by the team. The application combines static and dynamic pages at the front end and servlets in the middle tier to action requests from front end forms. The back end consists of a MySql database which holds customer information in a customer table and transaction information in a transaction table.

The team successfully developed an integrated front to back end application which incorporated a lot of the basic functionality one would find associated with a typical banking application such as static information pages about the bank, dynamic user registration pages, user login functions, user account pages, administration pages and admin functions.

Ten students were allocated to the team although only eight students engaged in the project. Two roles were allocated to the non-participating members of the team, Testing and Cloud Deployment. The team absorbed the testing role across team members but were unable to fulfil the cloud deployment requirement due to the resource constraints of missing two of the ten allocated team members.

The team made a decision not to do the Cloud deployment function as it was not essential to the overall goal of the project which was to develop a web application front to back end which of course can be run on a Tomcat Apache server without the need for cloud deployment.

The successful completion of this project was by no means a small achievement. To have eight practical strangers to come together and work in such a unified fashion to create a working integrated web application was in fact a huge achievement. It must also be said that each team member fully participated in all project meetings and also in a number of cross functional activities not initially allocated to them

The spirit of cooperation combined with the technical expertise learned on this module by the students on this project was exemplary.

2. Introduction to the Application Domain

The application domain is a banking type application. Of course, it was not possible within the project's four week timeframe to create an application of AIB or Bank or Ireland proportions.

The application does however contain the basic functions one would expect of the application domain.

- Information about the bank
- User registration
- User Login
- User account functionality: Statements of transactions, bill payment, transfer between accounts and display user details (overdraft and withdrawal limits)
- Admin login
- Admin functionality: Display users, simulate deposits and withdrawals to user accounts, delete users, set overdraft and withdrawal limits.

This project report is also accompanied by a full screen cast walkthrough of the application in mp4 format. The screen cast video was recorded through Zoom.

3. Technology Selection

The group decided to use Eclipse as the main software platform for this project. Eclipse was chosen as it was the technology used in the previous EE417 assignments and every group member was comfortable using it. In addition, Eclipse can be easily linked to a GitHub repository, making it efficient for multiple group members to work on the same project at the same time. Eclipse is also an IDE that is compatible with Windows, Linux, and Mac devices, making it suitable for every computer user.

Tomcat was used as the primary server and MySQL was used as the database technology. These two technologies were chosen as all users had these software packages installed and linking the tomcat server to eclipse and communicating to the database is straightforward in the Eclipse program. Tomcat is a relatively lightweight application, only providing the necessary information required to start running a server. This in turn makes it very quick to load up and start a server for the application, making it ideal for the numerous testing the group would be doing on the application. Tomcat also runs separately from the installed Apache instances, making it a very stable application and if a failure does occur, the server would still run at a high level. MySQL is an incredibly secure database application and has an incredibly reliable management scheme implemented in it, while also having the full capability to handle large numbers of queries from the Server-side applications, making it ideal for the banking application.

For Testing purposes, the Selenium software program was used. Selenium is used to create robust, browser-based regression automation suits and test, scale and distribute scripts across many environments. For this reason, the group decided to use it to run Automated UI tests on the web application in order to ensure that every aspect of the application that the user interacts

with is functioning correctly, such as navigating to other pages or completing the registration section of the application.

4. Code Management Decisions

GitHub was chosen has the code management repository for the project. For further detail on the code management process, please see Michael O'Hara's write up in section 8 of this report, Team Members' Contributions as Michael was responsible for this aspect of the project.

5. Roles and Responsibilities

	Bus. Req.s Analysis - Wed Application Architect	Front End HTML & CSS	Client Side Javascript	Server side Java Servlets & JSP	DB and JDBC connectivity	Testing & QA	Functional/ User Acceptance Testing	Cloud Deployment	GitHub	Project Lead	Project Docs	Project Demo
Mark	✓						✓			✓	✓	✓
Michael	✓					✓	✓		✓		✓	
Ciaran	✓			✓			✓				✓	
Daire	✓		✓				✓				✓	
Rui	✓	✓					✓				✓	
Mathieu	✓				✓		✓				✓	
Remi	✓	✓					✓				✓	
Corentin	✓			✓			✓				✓	
Saima (not engaged)								✓				
Sipeng (not engaged)						✓						

Each team member had a primary role on the project as denoted by a single tick in the column against the role. As the application was front end heavy, it was decided to allocate two team members to the front end HTML/JSP and CSS front end. As the front end was generating a significant number of actions, not the least with the back end database then it was also decided that two team members would work on the servlet middle tier.

All team members participated in the business analysis and requirements gathering phase, the testing phase to a certain degree and also the project documentation phase to a certain degree. Michael O'Hara was the principal team member on the testing phase and Mark McGrath on the project documentation phase. Mike performed automated Selenium testing and Mark compiled the Project documentation. The other team members contributed to the system and user acceptance testing functions and also assisted with the project documentation.

6. Project Communications

Formal project communications were scheduled weekly through group Zoom chats hosted my Mark. These increased as needed and as the project progressed. Full details of where all the minutes of these meetings can be found is in the appendix section of this project report.

They are also a good way to see the development progress of the project and really highlight the excellent team contributions and cooperation on the project.

Informal, day to day communications were done through Whatsapp group. Full details of where all the chats between team members in the group can be found are also in the appendix section of this document.

All supporting documentation, minutes, group chats, individual team members contributions, interim design documents, final design documents and so on are also attached to the project upload to loop as part of the zipped GIT repository.

7. Project Lifecycle

Project Milestones

Week 1: 18/03/2021 – 21/03/2021:

Agree project roles and responsibilities

Agree web application (online banking preferred)

Week 2: 22/03/221 – 28/03/2021:

Create Business Requirements and Analysis and Web application structure document.

Week's 3 & 4: 29/03/2021 - 11/04/2021

Development of web site

Week 5: 12/04/2021 – 25/04/21

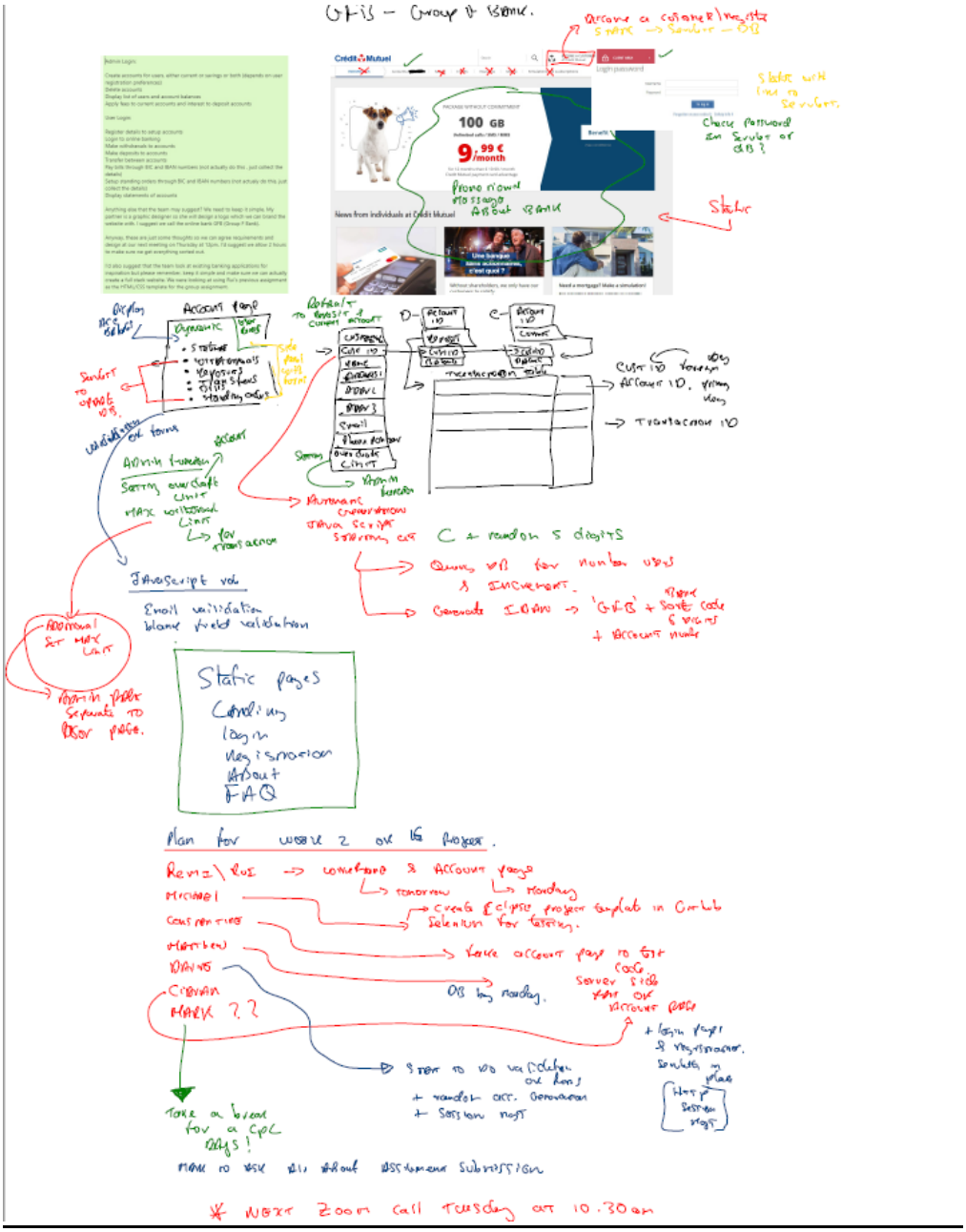
Testing and fixes

Requirements Gathering Phase

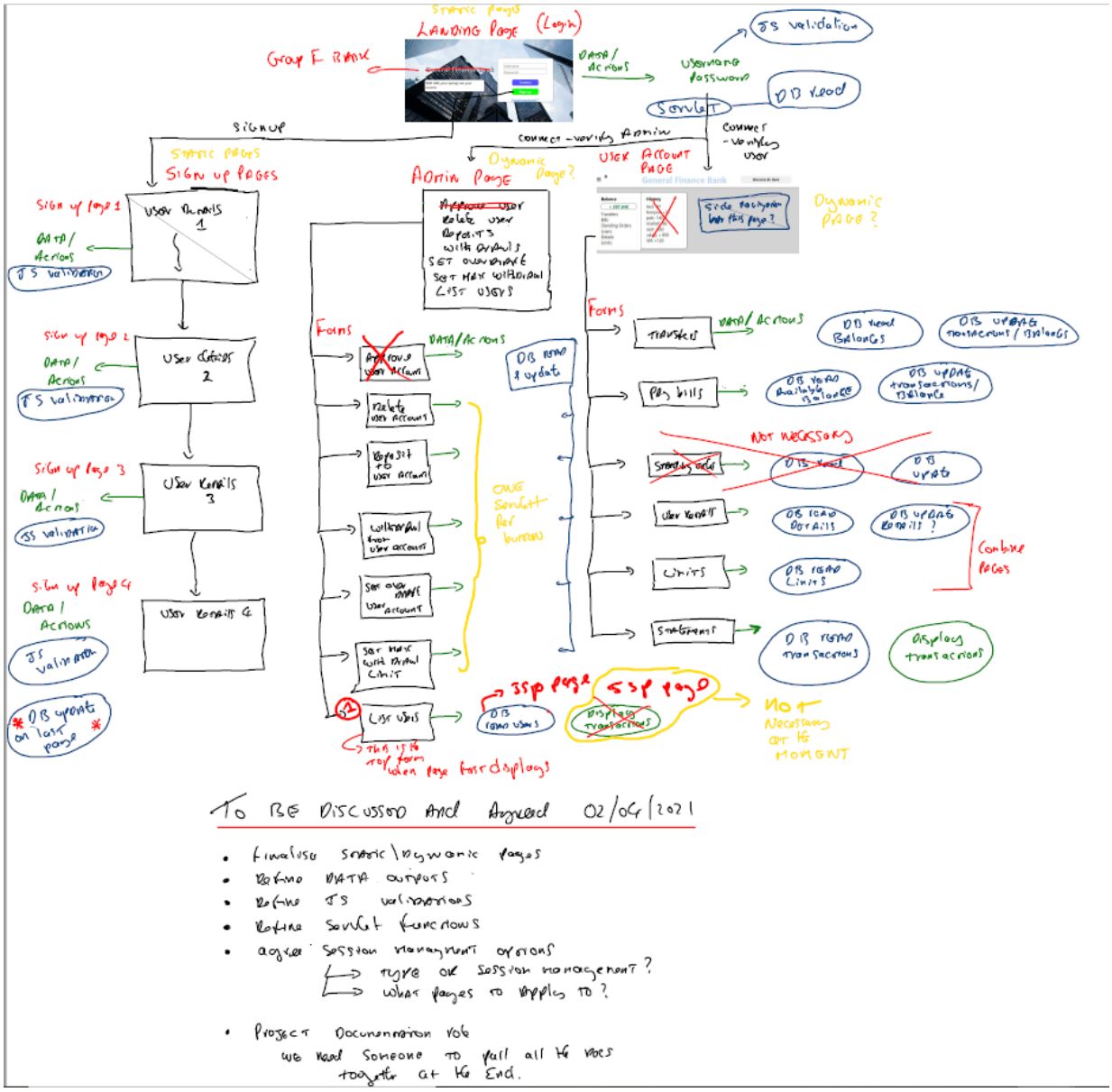
The team held two zoom call meetings at the start of the project to agree the requirements for the application. The evolution of the requirements for the application can be seen through the following web application diagrams and also through the meeting minutes of 18th and 25th March 2021.

Initial Application Architecture Designs

25/03/2021



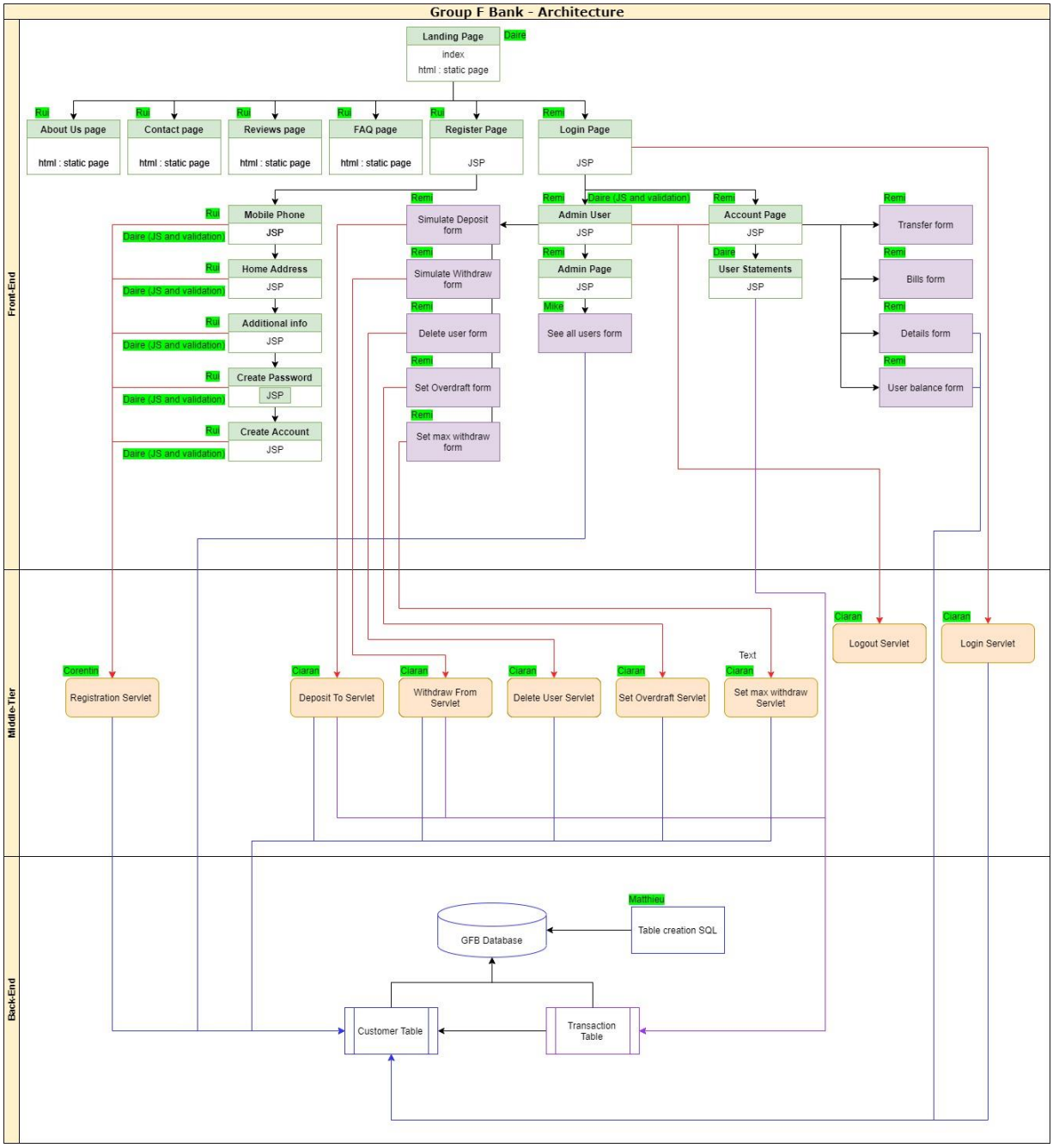
02/042021



To be discussed and Agreed 02/06/2021

- Finalise static/dynamic pages
- Define DATA outputs
- Define JS validations
- Define Servlet functions
- Agree Session management options
  - ⟶ type of session management ?
  - ⟶ what pages to apply to ?
- Project Documentation role
  We need someone to pull all the docs together at the End.

Final Application Architecture Design



Development Phase

Initially each team member worked on their own in each of their allocated roles. The front end developers put together an example front end, the middle tier servlet developers investigated servlet interaction with the back end database and other required servlet functionality. Generic Java script validation routines were written. An initial database with customer and transaction tables was created. The Github code repository was created and instructions on use of the repository issued to all team members. Selenium was investigated as a potential automated testing tool.

Once the team had completed their initial investigations in their respective areas and had some concrete work to show, the process of integration of the development to a complete web application began.

This was an iterative cycle and was quite Agile focussed really. Through extended zoom calls on a frequent basis the application evolved to its final working state.

The entire development process can be seen sequentially through the meeting minutes on the zipped Github repository.

Testing & Acceptance Phase

Mikael O'Hara developed and ran automated testing routines to check the inherent structural soundness of the application. Mike has detailed this in his description of his project contribution in the following section of this document, section 8.

Each team member completed their own unit tests of their work.

System testing was carried out as part of the agile development process through the regular team progress review calls on Zoom.

UAT was also carried out in a similar fashion but all the group members together.

Carrying out system and user acceptance testing in this fashion as part of the development process allowed the team to effect changes immediately depending on the results of the tests and what in fact could be done at that time. Anything that could not be fixed during the meetings was allocated to team members in the meeting minutes as actions.

8. Team Members Contributions

The following team member contribution statements have been written by each individual team member.

Mark McGrath

Mark fulfilled three main roles on the project, Project Leader, Project Documentation and Project Demonstration. Mark also participated quite actively in the requirements gathering phase and developed the web application architecture design from inception through to the final design with assistance from Remi.

As project leader, Mark organised the group's activities, hosted, chaired and minuted the formal zoom progress meetings and ensured the project ran according to plan and that the final project was delivered with the requisite assignment functionality.

In addition to compiling the project documentation, Mark wrote the executive summary, introduction to the application domain, roles and responsibilities, project communications and project life cycle sections.

Mark also carried out the full walk through screen cast project demonstration and actively participated in the group testing and development meetings through leading the application walk throughs at every meeting.

Remi Bertogliati

The team started to design the Application's design and architecture as a group. Remi proposed to take inspiration from the N26 bank and the French bank Crédit mutuel to determine our target for the project.

For this assignment, Remi was tasked to realize, in collaboration with Rui, the front end of the project. This included work on HTML, CSS and JSP pages. He worked on the layout of the account pages, login pages and the different admin pages. He also got to implement the navbar and the different pop-up modals. These features can be found on most pages to navigate through the project. To make these modals animated and interactive, he reused some JavaScript work from his previous assignments which can be found in the app.js file and in the different JSP pages.

He also created most of the CSS files to insure a coherent style between pages.

Although we had a person to ensure quality and testing, we planned several meetings to go through the project and debug together. Finally, Remi worked helped to work on the project's documentation by doing the web application architecture design with Mark.

Rui Fan

Front-End and CSS:

For this EE417 Web Application Development Group F Rui did these following pages:

- registerPage.jsp
- mobilePhoneNumber.jsp
- homeAddress.jsp
- additionalInformation.jsp
- createPassword.jsp
- createAccount.jsp

JSP pages are dynamically compiled into servlets, so it is easy to update the presentation code. JSP pages can be easily combined with static templates, including HTML or XML, or with code that generates dynamic content.

- aboutUs.html
- contact.html
- reviews.html
- faq.html

HTML is a static page, which can contain text, image, sound, flash animation, client script and so on. The file can be read directly without data processing.

- registerPage_style.css

CSS can effectively control the layout, font, color, background and other effects of the page more accurately.

Daire Cooney

JavaScript

It was Daire's Role was to provide some JavaScript functionality to the Group F Banking Application. Different JavaScript functionality had to be implemented across the Application so that the project functions as proposed.

Validation and events

Throughout this application, A lot of JavaScript validation functionality was implemented in order to make sure the information included in the forms was correct. This can be easily done if the form calls on a JavaScript function. This was done on all of the pages in the *register pages* folder. The script would read in the values from the IDs in the form and check to see that they contained the right information.

```
var fname = document.getElementById("fname").value;
var lname = document.getElementById("lname").value;
var email = document.getElementById("mail").value;
var DoB = document.getElementById("dateBirth").value;
var digits =  /^\d+$/;
var DoB_test = /(((0|1)[0-9]|2[0-9]|3[0-1])\/(0[1-9]|1[0-2])\/((19|20)\d\d))$/; <!-- Makes ure the format is DD/MM/YYYY --

if (fname == "" || fname == " "|| digits.test(fname) == true) <!-- checks that the name contains no digits -->
{
   alert("First name must be filled out and contain no numbers!");
   return false;
}
```

An alert would be returned displaying the error to the user so that they can correct it. This method was applied across all the "Register Pages".

There was also validation carried out on the *accountPage.jsp* and *adminUser.jsp* pages. For every form in the application, JavaScript validation was used. The "alert()" event was used to inform the user that they had entered in the incorrect syntax in the form field.

Animations

JavaScript also has the capability to implement some animations. The anime.js script (template from https://tobiasahlin.com/moving-letters/) was used to animate the motto of the GFB ("Fluent in Finance") across the screen in the *index.html, login.jsp* and *accountPage.jsp* pages. The letters were originally set to be invisible using the opacity setting and then were faded in gradually using the delay function.

```
var wrap_text = document.querySelector('.AnimationHeader .letters');
wrap_text.innerHTML = wrap_text.textContent.replace(/\S/g, "<span class='letter'>$&</span>"
anime.timeline({loop: true})
  .add({
    targets: '.AnimationHeader .letter',
    scale: [0,1],
    opacity: [0,1], // sets the characters to be invisible and then gradually make them vis
    easing: "easeOutExpo", //eases out the letters
    duration: 5000, // sets the transition time to be 5000ms -> 5 seconds
    delay: (el, n) => 125 * (n+1) // allows the characters to ripple in using the (n+1) to
  });
```

There was also a slideshow done on the index.html page. The basis of this slideshow was taken from https://www.w3schools.com/w3css/w3css_slideshow.asp and it was modified to suit the application. There were 3 slides, coupled with captions that showed on screen for 5 seconds and then move onto the next photo.

```
function showSlides() {
  var slideshowslides = document.getElementsByClassName("mySlides");
  var squares = document.getElementsByClassName("square");
  for (var i = 0; i < slideshowslides.length; i++) {
      slideshowslides[i].style.display = "none";  // hides the current slide
  }
  index++;
  if (index > slideshowslides.length) {index = 1}   // go back to the first slide
  for (i = 0; i < squares.length; i++) {
      squares[i].className = squares[i].className.replace(" .active_slides", ""); //r
  }
  slideshowslides[index-1].style.display = "block";  // display the slide
  squares[index-1].className += " .active_slides";
  setTimeout(showSlides, 5000); // Wait 5 seconds for the next slide to ease in
}
</script>
```

On the *accountPage.jsp*, there was also some JavaScript which took the id and balance of a user passed by the LoginServlet.java page and stored it in local storage.

Index.html

Daire was given the task to create the index.html page. This involved writing all the corresponding HTML, CSS and JavaScript code. The dynamic sidebar that was already coded by another group member was also implemented into this page.

adminPage.jsp

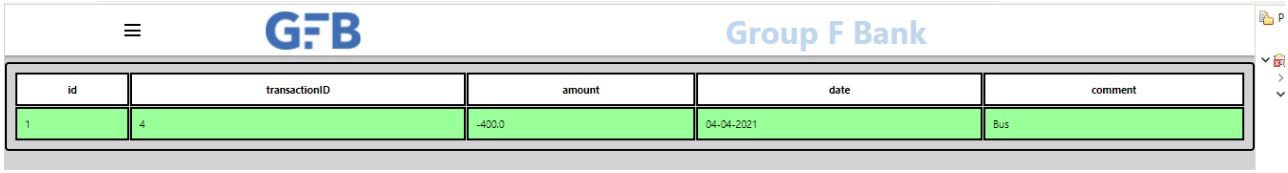
Daire also did the CSS styling on the *adminPage.jsp* for the display table.

UserStatements.jsp

Daire was also delegated the task of creating a script that returns the users transactions. The users ID was passed to this page through a form that had the ID as a hidden element that was triggered when the statements button was selected. The *UserStatements.jsp* page connects to the Database using the username and password of the user and the select statement is executed:

```
rs = stmt.executeQuery("SELECT * FROM mydb.transaction WHERE id = " + result);
```

This statement then accesses the database and retrieves the information while displaying the following page



Ciaran McCormac

Ciaran's primary role was to provide the non-registration servlets (listed below). These servlets provided the Tier-2 interface between the Tier-1 front-end and the Tier-3 database. Their use was ideal due to their ability to open and maintain database connections, rather than reconnecting on each page-load. This reduced the overall connection overheads, reduced database loading, and improved network efficiency. It also centralised the database access, making database issues easier to resolve, and the database itself easier to maintain (for example, when adding additional table entries).

Servlets were therefore written to provide the database access and straddle the interface between the HTML and the database. They included "business-logic" to provide some of the higher-level database operations (for example, the read-modify-writes in the WithdrawFromServlet) and to map it into the equivalent SQL. This allowed the database access to be abstracted, and private detail to be buried in the Servlets (for example, the database password - this ensured it was not visible to the user).

The list of servlets written by Ciaran, their roles ("Action") and interfaces ("Input"/"Output") are listed below.

DatabaseConnection.java

Support c*lass to provide the database connectivity of the servlets*

DeleteUserServlet.java

Servlet for deleting users

Input id

Action Delete user from the customer table and transaction table

Output None

DepositToServlet.java

Servlet for depositing an amount to a user account

Input id, amount

Action Read database balance, add amount and update database balance

Output Add to transaction table

Login Servlet.java

Servlet for Login Functionality

Input uname, password

Action Check the database for the specified username/password and login if applicable

Output Writes id, credentials,overdraft and withdrawal limits to session. Calls accountPage/adminPage/login depending on result.

Logout Servlet.java

Servlet for Logout Functionality

Input none

Action Invalidates session id and credentials

Output Calls login page

SetMaxWithdrawlServlet.java

Servlet for deleting users

Inputs id, maxwithdrawl

Actions Sets database maxwithdrawl for user

Outputs None

SetOverdraftServlet.java

Servlet for setting the overdraft for a user

Inputs id, overdraft

Actions Sets database overdraft for user

Outputs None

WithdrawFromServlet.java

Servlet for withdrawing an amount from a user account

Input id, amount

Action Read database balance, subtract amount and update database balance

Output Add to transaction table

Corentin LeStanc

The registration part of this website is accessible from the drop-down menu. By clicking on the button on the top left the drop-down menu will appear and you have to click on "sign up".

Then, you will arrive on the page (registerPage.jsp). This page contains text fields to be filled in to create an account. When you press the submit button you will be taken to the next registration page, this process has to be done six times each time with different information to fill in.

You will successively access the following pages:

• registerPage.jsp

• mobilePhoneNumber.jsp

• homeAddress.jsp

• additionalInformation.jsp

• additionalInformationStatus.jsp

• createPassword.jsp

• createAccount.jsp

Each of the following pages consists of a form that retrieves the information entered in the text fields and sends it to the "registrationServlet.java" servlet.

The servlet retrieves this information and processes it as follows:

Firstly, when the servlet is called by the first registration page (registerPage.jsp) the servlet stores the data coming from the form in variables (line 49 at 67 in registrationServlet.java). Variables are created to register any data from any registration pages. At each call of the servlet, the variables that do not correspond to the data entered on the page from which the servlet is called will therefore be null. For example, when registerPages.jsp is called, only the variables fname, lname, mail, and date birth will not be null.

Then, from line 70 to the end of the code, a succession of "if" allows to know which page has called the servlet in order to know which actions must be performed to update the database correctly.

The first “if” tell us that it is the first registering page (registerPage.jsp) that called the servlet (the “if” condition checks if the variables distributed by the other jsp pages are null). As it is the first record page a new row is created on the database customer with the information collected by this first page, the other information of the table not yet collected is initialized as null. Then, the servlet redirects us to the following registration page.

When this next page sends its data to the servlet, the servlet stores it again in variables and the "ifs" make it possible to know which page called the servlet in order to update the line of the database previously created but not containing all the information. We can know which line is to be updated thanks to the HTTP session. When calling the first page the mail entered by the user

is stored in the HTTP session (lines 93 and 94). As the mail cannot be duplicated in the database, it can work correctly. When record pages 2 to 6 call the servlet it can retrieve the mail from the HTTP session and so update the correct line.

It is also important to note that one email and one phone number cannot be used for several different accounts.

Michael O'Hara

ADMINPAGE

Michael also did a bit of work on the Admin Page which displays all the user records that are stored in the database. This is done through the use of a JSP which has java code embedded into to it to make a connection to the database and query the customer table for all the records that are in it.

```
try
{
    Class.forName("com.mysql.cj.jdbc.Driver");                              // Load up the driver
    connection = DriverManager.getConnection(dbURL, dbUsername, dbPassword); // Connect
    statement  = connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
    String query = "SELECT * FROM customer";
    query_res = statement.executeQuery(query);
    while(query_res.next()){
%>
```

The results of the query are then added into a table with a table row each row entry in the database.

GITHUB

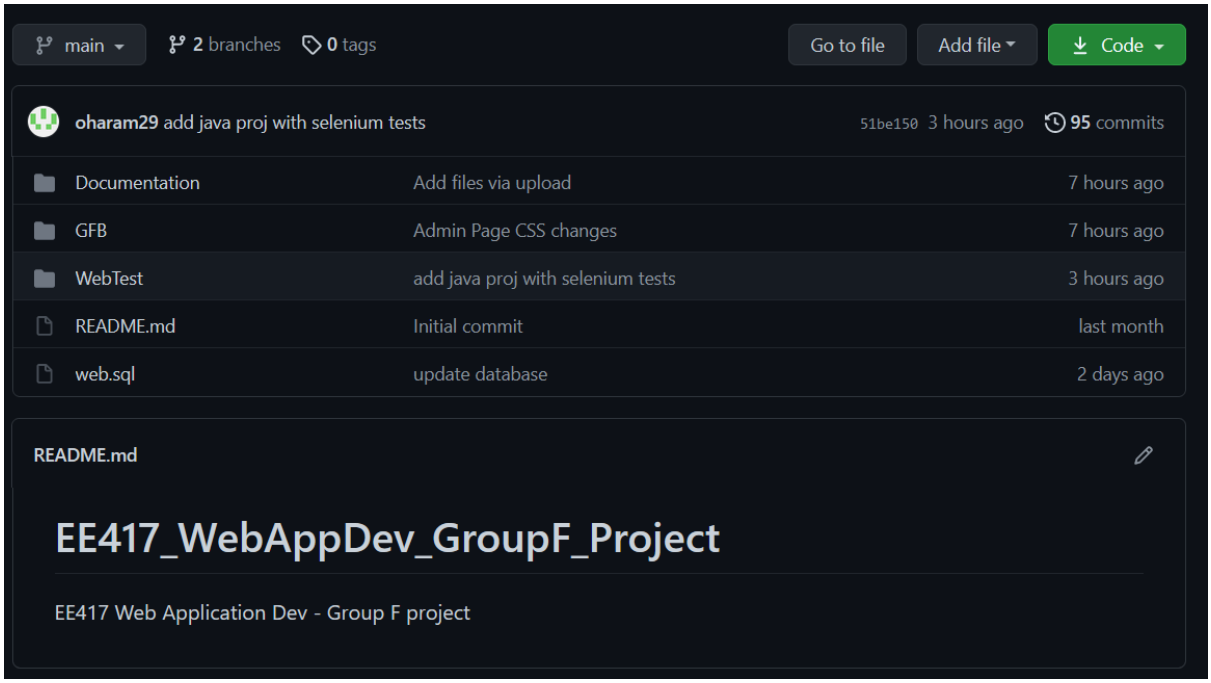
For the duration of the project Michael was in charge of managing the GitHub repository where the team would commit their code changes to . This involved initially creating the repo and inviting the all the other group members to be collaborators on the project. Once the collaboration invitation has been accepted then the other group members have full access to add and remove files or make changes to existing files

The repo can be found here:

https://github.com/oharam29/EE417_WebAppDev_GroupF_Project

A screenshot of the top level of the project can be seen below, containing:

- Documentation folder - for the report and minutes of meetings
- GFB – For the actual web application source code
- WebTest – Folder containing the selenium test scripts
- Web.sql – SQL script for creation of database



It also involves making sure everyone was able to link their version of the eclipse IDE or which ever IDE they were doing the development of the web application in and ensuring they could commit changes correctly and pull-down changes correctly. Inside of eclipse in order to commit or pull any changes, a group member must right click the project and open the team menu to have access to the git actions such as commit, push, pull and more.

We had a few mishaps along the way of code going missing or being overwritten which required a roll back to a previous version of the code base and manually adding changes back in one at a time to ensure we did not lose any progress.

This also involved merging in the front-end branch in with the main branch when we decided against doing branches. Different code branch did not make sense for this type of project as all the sections of the website worked in tandem with one another, so development was done in tandem.

When this merge took place there was a number of conflicts which were cause by two members working on the same file at the same time, to resolve this the files were combined, and minor adjustments were made to incorporate both sets of changes.

TESTING:

Due to the lack of participation from two group members Michael was also in charge of testing for the web application.

One method of testing that was incorporated into the projects was to use Selenium to run automated UI tests. These tests were run on the Login and Register sections of the web site. These tests are run using a web driver deriver server which is controlled by Java code which can be found in the project repo.

```
//change this to where project is stored on machine
String path ="C:\\Users\\Nova6\\eclipse-workspace\\java\\";
System.setProperty("webdriver.ie.driver", path+"\\WebTest\\drivers\\IEDriverServer.exe")

//local url for webapp + Instantiate a IEDriver class.
String localURL = "http://localhost:10037/";
WebDriver driver=new InternetExplorerDriver();

//Applied wait time + maximize window
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.manage().window().maximize();

//open browser with desried URL
driver.get(localURL+"GFB/index.html");
```

The above screen grab is a the set up for the test which defines a string path which is the path to where the project is stored on your local machine, the next line defines the type of driver the test uses and the path to the web driver file, which is made up of the variable path plus a path to where the web driver is in the project folder. It will then set the URL to localhost and the port the tomcat server is pointing at, in this case 10037, and instantiate a new web driver and maximize the window size.

It will then navigate to the home page web application. Now the tests are ready to be performed. The first test that was written was a basic login test which navigates to the login page and enters credentials for an account stored in the database and logs in to the website.

```
//open browser with desried URL
driver.get(localURL+"GFB/index.html");

//login test
driver.get(localURL+"/GFB/login.jsp");
WebElement username = driver.findElement(By.id("uname"));
username.click();
username.sendKeys("Adams");

WebElement password = driver.findElement(By.id("password"));
password.click();
password.sendKeys("test3");

WebElement login = driver.findElement(By.id("loginbtn"));
login.click();
```

The way this test is run is in it create web elements which are items on the page so for this test they are the username field, the password field and the login button, it gets this element by searching for them using the ID assigned to them in the html. For the texts it will call *"sendkeys"* to simulate a user entering their details, this will be the string pass to the call to that field. And then finally it will get the login button on the page by ID and simulate a user clicking on that button.

A similar test was done for the registering to the website, it will navigate through the pages associated with registering for the website and enter in values that a user could potentially enter.

```
//check new page is correct
URL = driver.getCurrentUrl();
if(URL == localURL+"/GFB/register_pages/homeAddress.jsp" ) {
    //mobile phone
    WebElement houseNum = driver.findElement(By.id("houseNum"));
    houseNum.click();
    houseNum.sendKeys("0");

    WebElement streetName = driver.findElement(By.id("streetName"));
    streetName.click();
    streetName.sendKeys("glasnevin avenue");

    WebElement address1 = driver.findElement(By.id("address1"));
    address1.click();
    address1.sendKeys("k12345");
```

As seen above it goes through similar steps to the previous test but with one difference, it will check the URL is correct upon clicking the continue button and moving to a new page.

The second type of testing that was done was manual QA testing. In order to carry this out the web application was opened up locally. Then the website was then tested by performing the actions the user would carry out such as navigating the links of the website to ensure the links work as intended.

It was also the actions of testing the servlets to make sure the do the intended actions such as simulate a deposit/withdrawal.

It also involved ensuring the database calls on the admin page to display a list of all the customers in the database at the time of loading that page.

Matthew LeBeouf

Mathieu created a database named gfb_database which contains 2 tables. The first one is called customer and contains all the useful information about a user such as name, email, phone number or address. The `status` parameter is an integer allowing to differentiate between a simple user and an administrator. For example, 1 is a user and 2 is an administrator. The primary key is the id. It is an integer which is automatically incremented by one when adding a new user to the table. Email and phone number are unique keys because we decided that a user can only have one bank account in order to facilitate the code. These can therefore only be associated with a single user. He then filled this table with five concrete examples.

The second one is called transaction and contains all the useful information about a transaction such as amount, date or subject of the transaction. The amount will be positive if the user is the creditor and negative if the user is the debtor of this transaction. The primary key is the transactionID. It is an integer which is automatically incremented by one when adding a new transaction to the table. The foreign key is id which is linked to the id of the customer table. He then filled this table with five concrete examples.

For the JDBC, Mathieu created a Java class named DatabaseConnection.java. Its attributes are an object of type Connection and another of type Statement. The constructor takes in parameters the url of the database as well as the user name and the password to access it. It allows to link the database and our code. This class also contains all the subroutines making it possible to update the database, to add a user or a transaction in the concerned table or to retrieve the information which one could need. When you want to connect to the database and then use it, you have to create a DatabaseConnection object and then call the appropriate subroutine of this class.

9. Application Installation Instructions

Installation instructions are included as part of the readme file in the zipped GIT repository which accompanies this documentation.

10. Appendix

Minutes of Meetings

There are minutes for nine Zoom progress calls.

1. 18-03-2021
2. 30-03-2021
3. 02-04-2021
4. 06-04-2021
5. 08-04-2021
6. 12-04-2021
7. 20-04-2021
8. 23-04-2021
9. 25-04-2021

There is just too much text to be included in this document. All meeting minutes can be found in the minutes sub-folder of the documentation folder in the accompanying zipped Github repository.

Whatsapp Group history chat for Group F

Similar to the minutes of meetings, there is just too much text from the whatsapp group messages to be included in this document. The full text of the Whatsapp Group F chat can also be found in the minutes sub-folder of the documentation folder in the accompanying zipped Github repository.