Criterion B – Design

Contents

[Use Cases 4](#_Toc162374082)

[Flowchart Overview 5](#_Toc162374083)

[Prototype Design 6](#_Toc162374084)

[Final Design 8](#_Toc162374085)

[Data Types Used 10](#_Toc162374086)

[Unified Modeling Language (UML) Diagrams 11](#_Toc162374087)

[Hierarchical Chart 14](#_Toc162374088)

[Connection Chart 15](#_Toc162374089)

[Data Flow 16](#_Toc162374090)

[Flowcharts 18](#_Toc162374091)

[Pseudocode 24](#_Toc162374092)

[Data Linear Extrapolation 24](#_Toc162374093)

[Preconditions 24](#_Toc162374094)

[Postconditions 25](#_Toc162374095)

[Getting the Latest Complete Year 25](#_Toc162374096)

[Preconditions 26](#_Toc162374097)

[Postconditions 26](#_Toc162374098)

[Data Combination 27](#_Toc162374099)

[Preconditions 28](#_Toc162374100)

[Postconditions 28](#_Toc162374101)

[Testing Plan 29](#_Toc162374102)

# Use Cases

A diagram of a person's data flow

Description automatically generated

Figure 1 All use cases which the client should be able to do.

# Flowchart Overview

A diagram of a flowchart

Description automatically generated

Flowchart 1 Main dashboard.

# Prototype Design

Designs created after the initial meeting with the client.[[1]](#footnote-1)

A screenshot of a computer screen

Description automatically generated

Figure 2 Login, signup, and user dashboard.

A screenshot of a computer

Description automatically generated

Figure 3 Company window.

A screenshot of a computer

Description automatically generated

Figure 4 Company merging.

# Final Design

Designs created after second meeting with the client.[[2]](#footnote-2)

A screenshot of a computer

Description automatically generated

Figure 5 Login, signup, and user dashboard.

A screenshot of a computer

Description automatically generated

Figure 6 Company dashboard.

A screenshot of a computer

Description automatically generated

Figure 7 Merging interface.

# Data Types Used

Table 1 Data types used.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Variable** | **Reasoning** |
| String | username | * Strings allowed for usernames to have any characters. |
| password | * More choice of characters means more secure password. |
| File | file | * File object from java.io.File library[[3]](#footnote-3). * File object is used by various libraries. * Company or user file can be loaded as File object. |
| User | currentUser | * Current user represented as variable of User type.   + Descriptive variable.   + Serves purpose of constructing user as an object.   + User object has useful methods I created. * Current user object has data which needs to be accessed. |
| Company | head | * Used in CompanyList class as beginning of linked list. |
| current | * Current company in linked list. * Useful when looping through linked list. * Allows for checking if there is a next company with methods. |

# Unified Modeling Language (UML) Diagrams

Models of each proposed class.

|  |  |
| --- | --- |
| **Authentication** | |
| -  -  - | usersFile: String  currentUser: User  debug: boolean |
| +  -  -  -  -  +  - | Authentication()  signUp(username: String, password: String): User  logIn(username: String, password: String): User  writeString(file: RandomAccessFile, string: String): void  readString(file: RandomAccessFile, string: String): void  getUser(): User  toggleDebug(toggle: boolean): void |

UML Diagram 1 Authentication class.

|  |  |
| --- | --- |
| **User** | |
| -  -  -  -  - | username: String  isAdmin: boolean  file: File  companyList: CompanyList  debug: boolean |
| +  +  +  +  +  - | User(username: String, isAdmin: Boolean, filePath: String)  User()  getUsername(): String  getIsAdmin(): boolean  getCompanyList(): CompanyList  toggleDebug(toggle: boolean) |

UML Diagram 2 User class.

|  |  |
| --- | --- |
| **CompanyList** | |
| -  -  +  + | head: Company  file: File  debug: boolean  companyListLoaded: boolean |
| +  +  +  +  +  +  +  + | CompanyList(file: File)  add(company: Company)  save()  toArray(): Company[]  length(): int  isEmpty(): boolean  exists(fileName: String): boolean.  getHead(): Company |

UML Diagram 3 CompanyList class

|  |  |
| --- | --- |
| **Company** | |
| -  -  -  -  -  -  -  +  + | next: Company  name: String  description: String  country: String  tickerSymbol: String  revenues: ArrayList<Statistic>  costs: ArrayList<Statistic>  debug: boolean  companyLoaded: boolean |
| +  +  +  + | Company(filePath: String)  Company()  getNext(): Company  setNext(next: Company) |

UML Diagram 4 Company class

|  |  |
| --- | --- |
| **Statistic** | |
| --  - | name: String  filePath: String  data: ArrayList<Data> |
| +  +  +  +  +  + + | Statistic(name: String, filePath: String)  Statistic(name: String, data: ArrayList<Data>)  getName(): String  readData()  getData(): ArrayList<data>  toString(): String  extrapolateData(monthsToExtrapolate: int): ArrayList<Data> |

UML Diagram 5 Statistic class

|  |  |
| --- | --- |
| **Data** | |
| -  -  - | year: int  month: int  value: int |
| +  +  +  + + | Data(year: int, month: int, value: int)  getYear(): int  getMonth(): int  getValue(): int  toString(): String |

UML Diagram 6 Data class

# Hierarchical Chart

Illustration of how proposed windows and interfaces could be linked together.

A screenshot of a computer screen

Description automatically generated

Figure 8 Hierarchal chart.

# Connection Chart

A representation of the proposed class relationships.

A diagram with black text

Description automatically generated

Figure 9 Connection chart.

# Data Flow

A diagram of a login

Description automatically generated

Data Flow 1 Process of user log in.

A diagram of a company

Description automatically generated

Data Flow 2 Process of merging companies.

A diagram of a company

Description automatically generated

Data Flow 3 Process of user adding a company.

A diagram of a company data processing

Description automatically generated

Data Flow 4 Process of creating company valuation.

# Flowcharts

A diagram of a flowchart

Description automatically generated

Flowchart 2 Initial screen.

A diagram of a flowchart

Description automatically generated

Flowchart 3 User dashboard.

A screenshot of a diagram

Description automatically generated

Flowchart 4 Company addition and removal.

A diagram of a company

Description automatically generated

Flowchart 5 Company dashboard flowchart.

A diagram of a flowchart

Description automatically generated

Flowchart 6 Company statistics interface.

# Pseudocode

## Data Linear Extrapolation

Pseudocode 1 Data linear extrapolation.

|  |
| --- |
| calculateLinearExtrapolationData(dataLastYear, month, nextYear)  n = LENGTH(dataLastYear)  sumX = 0  sumY = 0  sumXY = 0  sumX2 = 0    LOOP i FROM 0 TO n-1  IF dataLastYear[i].month = month + 1 THEN  x = dataLastYear[i].year  y = dataLastYear[i].value  sumX = sumX + x  sumY = sumí + y  sumXY = sumXY + x \* y  sumX2 = sumX2 + x^2  ENDIF  END LOOP    exaggeratedFactor = 2.0  slope = ((n \* sumXY - sumX \* sumY) / (n \* sumX2 - sumX^2)) \* exaggeratedFactor  intercept = (sumY - slope \* sumX) / n    projectedValue = intercept + slope \* (nextYear + 1)    RETURN NEW Data(month + 1, nextYear + 1, ROUND(projectedValue))  END calculateLinearExtrapolationData |

### Preconditions

1. “dataLastYear” must be a non-empty ArrayList of Data objects.
2. Each data object in the “dataLastYear” ArrayList must have the attributes “month,” “year,” and “value” initialized.
3. “month” should be an integer between 1 and 12, where 1 corresponds to January and 12 corresponds to December.
4. “nextYear” should be a positive integer representing the future year for which the projection is to be calculated.

### Postconditions

1. The function returns a new “Data” object representing the data point that was extrapolated.
2. The “value” attribute from the returned “Data” object holds the rounded result of the linear extrapolation.
3. The “value” is determined through linear regression and is exaggerated by a scale factor of 2.0 to project the future value of the data point.

## Getting the Latest Complete Year

Pseudocode 2 Getting latest complete year.

|  |
| --- |
| getLatestCompleteYear()  CALL readData()    IF LENGTH(data) >= 12 THEN    lastYear = data[LENGTH(data) - 1].year    dataLastYear = NEW LIST FROM data[LENGTH(data) - 12 TO LENGTH(data)]    FOR EACH dataPoint IN dataLastYear  IF dataPoint.year != lastYear THEN  LOG name + " is not a full year."  RETURN -1  ENDIF  END LOOP    RETURN lastYear  ENDIF    RETURN -1 // If there are less than 12 datapoints  END getLatestCompleteYear |

### Preconditions

1. The call to “readData()” must be successful, meaning that the data for the Statistic object must be successfully read.
2. The “data” list needs to hold data in a chronological order. Additionally, each Data object must have the “month” and “year” attributes properly initialized.
3. For the function to perform as expected, it requires that there are at least 12 datapoints (a full year’s worth) to consider the latest year as complete.

### Postconditions

1. If the function identifies 12 datapoints in “data” list that belong to the same year, the last full year that it finds is the year that is returned.
2. If the function returns -1 it means that it cannot verify a complete year as the latest one in the dataset. This scenario occurs if the dataset contains less than 12 datapoints in total or if there is not a single consecutive year within the dataset.

## Data Combination

Pseudocode 3 Data combination.

|  |
| --- |
| FUNCTION combineData(data1, data2)  combineData = NEW LIST    data1copy = COPY OF data1  data2copy = COPY OF data2    FOR EACH dataPoint1 IN data1  PRINT dataPoint1    FOR EACH dataPoint2 IN data2  PRINT dataPoint2    IF dataPoint1.year = dataPoint2.year AND dataPoint1.month = dataPoint2.month THEN  REMOVE dataPoint1 FROM data1copy  REMOVE dataPoint2 FROM data2copy    combined Value = dataPoint1.value + dataPoint2.value  ADD NEW Data(dataPoint1.year, dataPoint1.month, combinedValue) TO combinedData  ENDIF  END LOOP  END LOOP    ADD ALL data1copy TO combinedData  ADD ALL data2copy TO combinedData    SORT combinedData BY year, THEN month    PRINT combinedData    RETURN combinedData  END FUNCTION |

### Preconditions

1. Both the “data1” and “data2” lists are lists of “Data” object, each “Data” object containing the attributes of “year,” “month,” and “value”. These values must be initialized for the method to work.
2. The “data1” and “data2” lists must be initialized lists before the “combineData” method is called. They should not be “null”.
3. Although not strictly enforced by the method’s logic the lists should be in the correct order in ascending ordered first by “year” then by “month.”

### Postconditions

1. The function returns a new list called “combinedData” containing all the unique data from “data1” and “data2”. If a “Data” object exists in both lists, a single combined data point is added with the value being the sum of the values from “data1” and “data2”.
2. Any data points that were combined due to having the same year and month are not included in their original form in “combinedData” meaning that duplicates are removed.
3. The output list is sorted first by the “year” attribute then by “month” attribute, ensuring that the data points follow a chronological order.
4. Both initial lists remain the same after the method, this being achieved by operating on copies of the original lists.

# Testing Plan

Table 2 Testing plan outlining tests that will need to be performed on the program to ensure it meets success criteria.

|  |  |  |  |
| --- | --- | --- | --- |
| **Success Criterion Tested** | **Description of Test** | **Method of Test** | **Expected Outcome** |
| 1 | Check if company can be added to user file. | * Open the file before adding. * Check contents of file. * Run program and open dashboard. * Press “Add Company” button. * Close program and check contents of user file. | * User file should have new company file name added. |
| Test if companies can be deleted from user file. | * Open file before adding it, check contents. * Run program and open dashboard. * Open company that has been added. * Click “Delete” button. * Close program. * Open user file. | * User file should not have name of removed company file. |
| 2 | Check if all company data is stored on file. | * Open company file. * Check data. * Open program. * Perform action that would change data. | * Contents of company file altered. |
| 3 | Check if user inputs are valid. | * Input abnormal values into input forms in program. | * Program should output data is invalid. * You should be prompted to enter data again. * Or kicked out of input process. |
| Check if existing username check is performed. | * Open signup interface. * Input existing username. | * Program should output that username already exists. |
| Check if company file added by user is CSV file. | * Open main dashboard. * Go to add company. * Try to add a file of other file type. | * Program should output that the file is not CSV. * File should be rejected. |
| 4 | Check if interface is easy-to-use. | * Give a program to someone unfamiliar with the program. * Command them to perform simple task. | * They should be able to perform it without needing help. * Completion in a reasonable amount of time. |
| 5 | Check if documentation is embedded. | * Check if “i” button exists. * Check if it exists on all interfaces. * Click the button. | * Information should pop up. * Information for almost every page should exist. |
| Check if documentation is easily accessible. | * Check if the “i” button appears on every interface. * Open all main program interfaces. * Check top left corner for button. | * “i” button should be in top left corner. * True for all main interfaces. |
| * Give program to user unfamiliar. * Ask them to open documentation. | * They should intuitively click “i” button to get documentation. |
| 7 | Check if company data is retrievable from opening file. | * Open company file. | * Company files should be readable by the user. * User should be able to edit file as well. |
| Check if company data is retrievable by program from company file. | * Open program dashboard. * Add company. | * Company should be added. * Name should appear properly. * Name indicates data was loaded. |
| * Open company dashboard. * Open statistics. | * If statistics are listed, it means data was loaded. |
| 7 | Check if statistics can be graphed. | * Open company panel. * Click on “Statistics” button. * Click on a statistic button. | * Popup with a graph of the data for that statistic should popup. |
| 8 | Check if user can edit basic company details. | * Open company dashboard. * Input new name, description, and country. * Click the button that updates the details. | * Open company file. * Company file should have new values. |
| 9 | Test if user can use program to estimate value of company. | * Calculate company value beforehand. * Open company dashboard. * Press the “Calculate Value” button. * Input number of years to predict for (enter 0). | * A popup should appear with estimated value. * The company value that you calculated earlier should somewhat align. * Or estimate should be reasonable. |
| 10 | Test if program can project company value. | * Open company dashboard. * Press the “Calculate Value” button. * Input random number of years. * Repeat. | * Every time different value should be predicted. |
| 11 | Test if merging works. | * Open company dashboard. * Click “Merge” button. * Select two companies to be merged. * Perform merge. | * New CSV file for merged company should be created. * New file should have combined statistics data. |

1. Please see Appendix A: 1st Meeting with the Client. [↑](#footnote-ref-1)
2. Please see Appendix B: 2nd Interview with the Client. [↑](#footnote-ref-2)
3. See Libraries section of this document. [↑](#footnote-ref-3)