**CIOBrain Deployment**

Final Report

Team #6

Anton Horvath

Anuja Sahu

Scott Lorance

Jose Colina Salas

Saige Wright

Kevin Lieng

**Sponsor**: Tom Hill - Fellows Consulting Group, LLC

# Executive Summary

This report details the CIOBrain Deployment Project including the plan, requirements specifications, architecture, design, and testing plan. All features implemented throughout the project are described, along with explanations behind decisions made when approaching solutions. The report also describes CIOBrain’s purpose and the application’s main capabilities to give some background on what is being deployed.

The project encompassed two types of deployments: local deployment on a Windows machine and cloud deployment to Azure.

Local deployment on a Windows machine was made easier by creating a one-click MSI installation process that created a desktop shortcut that the end user could click to launch the application offline on their machine. Prior to this solution, both the CIOBrain API and React applications would need to be started on separate ports on localhost and viewed in a browser.

Cloud deployment on Azure was made easier by creating a script that will ask the user for simple input information before handling the entire deployment process to their Azure account. Prior to this solution, users would need to know how to create the web.config files for both projects, create repositories that the deployment process would use as the reference, and deploy both applications successfully before finding the URL where the API application was launched, configuring the React application to communicate with it, and then redeploying.

Deployment (launching the application) was made far easier for chief information officers — the main end users for this application — as they do not commonly have development backgrounds.

# Table of Contents

[**Executive Summary**](#_gn9u7vj3mow6) **2**

[**Table of Contents**](#_etb1mo8d9lyc) **3**

[**List of Figures**](#_w20nhmecgu1y) **5**

[**List of Tables**](#_1wjvr27bhs4n) **6**

[**1. Introduction**](#_d5584laijx2n) **6**

[1.1. Purpose and Scope](#_rzkekuuq7onz) 6

[1.2. Product Overview (including capabilities, scenarios for using the product, etc.)](#_maue4fxugkiv) 7

[1.3. Structure of the Document](#_oyttoefee17z) 7

[1.4. Terms, Acronyms, and Abbreviations](#_tyzowaj0ketw) 8

[**2. Project Management Plan**](#_io7yskemyqw0) **8**

[2.1. Project Organization](#_qpni7isja9ga) 8

[2.2. Lifecycle Model Used](#_7mv8szm98n4z) 8

[2.3. Risk Analysis](#_yxtblxp7fn9z) 9

[Table 1.1 - Risk Chart](#_51tlroq6ewfq) 9

[2.4. Hardware and Software Resource Requirements](#_9q39f9d8ez4z) 9

[2.5. Deliverables and Schedule](#_ozrtomw6zblv) 10

[Figure 1.1 - Desktop Icon](#_ea2yqmb4mmsn) 10

[2.5.5. Schedule](#_9hwex7k20ueo) 11

[Table 1.2 - Schedule of Document Deliverables](#_kqmzjircetm1) 12

[Figure 1.2 - List Schedule](#_aqktl46vz5s) 12

[Figure 1.3 - Timeline Schedule](#_2ykk9f51orl4) 12

[2.6. Monitoring, Reporting, and Controlling Mechanisms](#_4zusgqi2vt1v) 13

[Figure 1.4 - Completion Indication](#_rtlvgkgd6xj5) 13

[2.7. Professional Standards](#_gi82ohrj9j13) 13

[2.8. Impact of the project on individuals and organizations](#_ii49wgkvogbl) 13

[**3. Requirements Specifications**](#_e578isu14np9) **14**

[3.1. Stakeholders for the system](#_jjqme9fuipa4) 14

[3.2. Use case diagrams](#_b8nphi2bj1ui) 15

[Figure 2.1 - Native App Installer Use Case Diagram](#_nvafymv38o4c) 15

[Figure 2.2 One Click Transfer Use Case Diagram](#_xaqakqqwnrs9) 16

[Figure 2.3 Login Use Case Diagram](#_5zojdqhc7ak) 17

[Figure 2.4 Multiple User Use Case Diagram](#_wa1cz110845h) 18

[3.2.1. Graphic use case model](#_b9jx4puihh2s) 19

[3.2.2. Textual Description for each use case](#_lk6h6gjgc2xy) 19

[3.3. Rationale for your use case model](#_sqs3ht1g3fj5) 19

[3.4. Non-functional requirements](#_ax3fy2w9ely) 20

[**4. Architecture**](#_y5ehagkbmypk) **21**

[4.1. Architectural style(s) used](#_m8uwvee65fqq) 21

[4.2. Architectural model](#_nrarr8ir21n5) 23

[Figure 3.1: Architectural Client-Server Model for Base CIOBrain](#_z2u0vzqqz7o0) 23

[Figure 3.2: Authentication Architecture](#_nv9wr1tejayx) 23

[Figure 3.3: Event Driven Architecture for Data Uploads between Desktop App and Server](#_2sljutpoggfz) 24

[Figure 3.4: Model-View Controller Structure of CIOBrain between changes in data](#_um53zgdn7oxg) 25

[Figure 3.5: Data transfer from desktop app architecture](#_sj3wnkizc14m) 26

[4.3. Technology, software, and hardware used](#_tgohox2i1qzg) 27

[4.4. Rationale for your architectural style and model](#_v6a6a7val0nf) 27

[**5. Design**](#_hnqc8bmo8q5x) **29**

[5.1. GUI (Graphical User Interface) design](#_i2jow3h8ddoc) 29

[Figure 4.1: CIOBrain Trial v3 Desktop Shortcut](#_fzdsr33mg2f) 29

[Figure 4.2: CIOBrain Trial v3 Running on Native Process](#_tu4i31cs66f) 29

[Figure 4.3: Login Function Visualization](#_3luu70sds0lu) 30

[Figure 4.4: Data transfer capability GUI](#_7pdtrtuwii3d) 30

[5.2. Static model – class diagrams](#_wpcpu4hc0y83) 31

[Figure 4.5: Class diagram pre-build for MSI Installer Wrapper](#_ejz95dxa5qc1) 31

32

[Figure 4.6: Class diagram pre-build for Desktop Application Login Authentication](#_r52b18rba98y) 32

[Figure 4.7: Class diagram post-build for MSI Installer Wrapper](#_7460yvupdpv) 32

[5.3. Dynamic model – sequence diagrams](#_gvhin6r0m4q0) 33

[Figure 4.8: Sequence diagram for MSI-Installed Application Run](#_kh0txyrf5a25) 33

[Figure 4.9: Login Authentication Diagram](#_39tyvs7dl96u) 34

[Figure 4.10: Azure Deployment Communication Diagram](#_ansjardwkq68) 34

34

[Figure 4.11: Sequence Diagram for data transfer from desktop app to Azure](#_qiro54gfq8lu) 35

[5.4. Rationale for your detailed design model](#_6kzmtqlszd6d) 35

[5.5. Traceability from requirements to detailed design model](#_bgt18p5pv8f2) 36

[Table 2.1: Forward traceability](#_en424wdhdogf) 36

[Table 2.2: Backward traceability](#_gec3ikstk5w3) 36

[**6. Test Plan**](#_5kcttzd1h55b) **36**

[6.1. Requirements/specifications-based system level test cases](#_sonou5pd683x) 36

[6.2. Traceability of test cases to use cases](#_f56q9olb5w6d) 40

[Table 3.1: Test Cases and Their Associated Use Cases](#_gfn66y4313uu) 41

[6.3. Techniques used for test generation](#_p4pzxkgeeon4) 41

[6.4. Assessment of the goodness of your test suite  
(Which metrics were used for such assessment?)](#_ykwh1vjxy92e) 41

[**References**](#_70glw3ypk92v) **43**

# List of Figures

[Figure 1.1 - Desktop Icon](#_ea2yqmb4mmsn) 9

[Figure 1.2 - List Schedule](#_aqktl46vz5s) 11

[Figure 1.3 - Timeline Schedule](#_2ykk9f51orl4) 11

[Figure 1.4 - Completion Indication](#_rtlvgkgd6xj5) 12

[Figure 2.1 - Native App Installer Use Case Diagram](#_nvafymv38o4c) 14

[Figure 2.2 One Click Transfer Use Case Diagram](#_xaqakqqwnrs9) 15

[Figure 2.3 Login Use Case Diagram](#_5zojdqhc7ak) 16

[Figure 2.4 Multiple User Use Case Diagram](#_wa1cz110845h) 17

[Figure 3.1: Architectural Client-Server Model for Base CIOBrain](#_z2u0vzqqz7o0) 22

[Figure 3.2: Authentication Architecture](#_nv9wr1tejayx) 22

[Figure 3.3: Event Driven Architecture for Data Uploads between Desktop App and Server](#_2sljutpoggfz) 23

[Figure 3.4: Model-View Controller Structure of CIOBrain between changes in data](#_um53zgdn7oxg) 24

[Figure 3.5: Data transfer from desktop app architecture](#_sj3wnkizc14m) 25

[Figure 4.1: CIOBrain Trial v3 Desktop Shortcut](#_fzdsr33mg2f) 29

[Figure 4.2: CIOBrain Trial v3 Running on Native Process](#_tu4i31cs66f) 29

[Figure 4.3: Login Function Visualization](#_3luu70sds0lu) 30

[Figure 4.4: Data transfer capability GUI](#_7pdtrtuwii3d) 30

[Figure 4.5: Class diagram pre-build for MSI Installer Wrapper](#_ejz95dxa5qc1) 31

[Figure 4.6: Class diagram pre-build for Desktop Application Login Authentication](#_r52b18rba98y) 31

[Figure 4.7: Class diagram post-build for MSI Installer Wrapper](#_7460yvupdpv) 31

[Figure 4.8: Sequence diagram for MSI-Installed Application Run](#_kh0txyrf5a25) 32

[Figure 4.9: Login Authentication Diagram](#_39tyvs7dl96u) 33

[Figure 4.10: Azure Deployment Communication Diagram](#_ansjardwkq68) 33

[Figure 4.11: Sequence Diagram for data transfer from the desktop app to Azure](#_qiro54gfq8lu) 34

# List of Tables

[Table 1.1 - Risk Chart](#_51tlroq6ewfq) 8

[Table 1.2 - Schedule of Document Deliverables](#_1ksv4uv) 10

[Table 2.1: Forward traceability](#_en424wdhdogf) 35

[Table 2.2: Backward traceability](#_gec3ikstk5w3) 35

[Table 3.1: Test Cases and Their Associated Use Cases](#_gfn66y4313uu) 39

# 1. Introduction

## 1.1. Purpose and Scope

The CIOBrain application is split into two separate applications: a Node.js API backend and a React frontend. End users of the application, mainly chief information officers (CIOs), would need to obtain both projects. They would then need to know how to start both applications on separate ports, and navigate to the correct localhost URL on a browser to actually use the application on a local machine. This process is complex and impractical for non-developers that will be using the application.

Additionally, chief information officers would need to know how to deploy the application onto Azure if they wanted it hosted on the cloud. This would mean creating web.config files within the projects, knowing what properties to set, creating repositories for the projects that would be used as a reference for deployment, and deploying the applications successfully. This process is even more complex and would require a substantial amount of knowledge to implement.

The CIOBrain deployment project aims to solve both issues by making deployment options approachable for users with non-developer backgrounds. The project strives to establish one-click installation capabilities on a local Windows machine and simple script input deployment to Azure. Both capabilities will allow for easier application startup for end users.

The scope of the project was two main deliverables: MSI installation file and a cloud deployment script.

## 1.2. Product Overview (including capabilities, scenarios for using the product, etc.)

CIOBrain enables users to track assets and the relationships they have with others assets. Chief information officers of organizations need to know the assets the company owns to better gauge impact when an asset failure occurs or know what new products or assets will affect other assets.

Tracking these assets is difficult. With hundreds to thousands of assets controlled by companies, having a visual representation is crucial for easily understanding everything. CIOBrain offers this visual representation to aid CIOs.

Capabilities include:

* Importing Excel spreadsheets following a specific format containing asset information
* Choosing assets sorted by type
* Viewing a graph depicting the asset and all relationships to other assets
* Hovering over assets to view additional details
* Uploading asset data to a CIOBrain cloud instance\*
* Deployment-ready to Azure

## 1.3. Structure of the Document

The document is split into different parts:

* Introduction
* Project Management Plan
* Requirements Specifications
* Architecture
* Design
* Test Plan

The report starts with the plan and approach to the project, followed by the actual implementation including the architecture and design of it. Testing plans correspond to the objectives that were accomplished by the end of the project.

## 1.4. Terms, Acronyms, and Abbreviations

CIO - Chief information officer

API - Application Programming Interface

MSI - Microsoft Windows Installer

Auth - Authentication functionality (signup and login capabilities)

JSON - JavaScript Object Notation

# 

# 

# 2. Project Management Plan

## 2.1. Project Organization

This project is organized into a single team responsible for the deployment phase. Previous teams have provided the base functionality of the CIO Brain application.

## 2.2. Lifecycle Model Used

The lifecycle model used for this project will be an Agile type model. This is because the personnel in this group adapted well to this model and due to the short timeframe that the project has, which is only 4 months, this was objectively the best option compared to iterative or waterfall methods since we need to pump this out in such a short amount of time. We are utilizing a scrum type Agile model in which we have sprints of developments to follow. In our project management software we have set up Gantt chart-like schedules to follow these sprints.

## 2.3. Risk Analysis

| Risk Type | Probability | Impact |
| --- | --- | --- |
| Documentation Confusion From Previous Project Work | High | Tolerable |
| Team Scheduling Conflicts | Moderate | Moderate |
| Specification Miscommunication | Low | Serious |
| Change/Enhancement Request Configuration Conflicts | Moderate | Tolerable |
| Inconsistencies Throughout Additions | Low | Tolerable |

##### Table 1.1 - Risk Chart

## 2.4. Hardware and Software Resource Requirements

**R1**. CIOBrain must be able to be run on Microsoft Azure

**R2.** CIOBrain can be run on a personal computer

**R3**. CIOBrain will have its database taken from an Excel spreadsheet.

**R4.** CIOBrain will be running on Windows systems primarily.

**R5.** CIOBrain can be compatible with MacOS

**R6.** CIOBrain will have a one click installation msi file

**R7.** CIOBrain will have a one click transfer capability from desktop app to Microsoft Azure

**R8.** CIOBrain will have production-ready features to be able to operate in Microsoft Azure

**R9.** The Azure system will be able to handle multiple users running on CIOBrain at once

**R10.** CIOBrain will support login functionality utilizing a password system.

**R11.** The msi installation file will be a one touch installation

## 2.5. Deliverables and Schedule

**D.0** Install and run CIO application on local Windows machine and on Azure, testing current functionality.

**D.1** Build easy one-click installation file with run desktop icon called ciobrainTrialv3.msi.

* CIO’s should be able to launch the application by simply running through an installation process, which will add the CIOBrain application icon to the desktop. They will then be able to launch the application by simply clicking on the icon.



##### Figure 1.1 - Desktop Icon

**D.2** Build easy one-click data transfer capability from local CIOBrain app to Azure-hosted app.

* CIO’s will be able to click on a **Sync** button that will save data to a CIOBrain project in the Azure-hosted web app version. They will then be able to view their project online through any device.

**D.3** Build authentication (login/password access) for Azure-hosted app

* CIO’s should be able to log in or sign up for a CIOBrain account. When they click the **Sync** button in the local application, it will ask them to log in to their CIOBrain account before transferring the project data to their account data
* Possible auto-sync capabilities while the user is editing within the local app (Internet available)?

## 

## 

## 

## 2.5.5. Schedule

| Assignment | Members | Due Date |
| --- | --- | --- |
| Project Management Plan | All members | September 9 |
| Requirements Documentation | All members | September 23 |
| Architecture Documentation | All members | October 7 |
| Detailed Design Documentation | All members | October 21 |
| Testing Plan | All members | November 11 |
| Final Project Report | All members | December 2 |
| Final Project Demonstration | All members | December 2 |

##### Table 1.2 - Schedule of Document Deliverables

##### 

##### Figure 1.2 - List Schedule

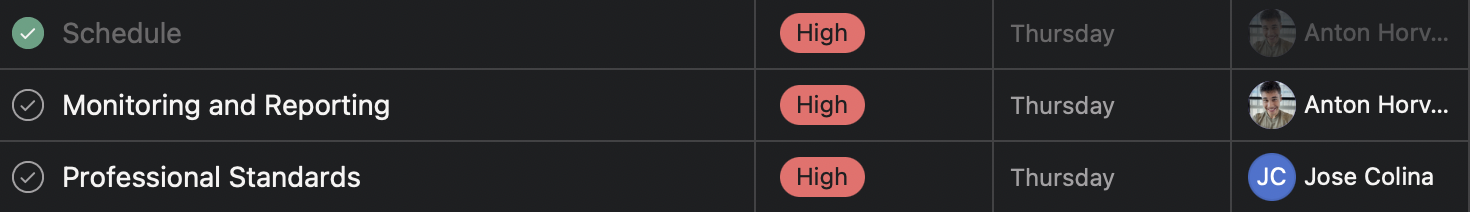
##### Figure 1.3 - Timeline Schedule

## 

## 2.6. Monitoring, Reporting, and Controlling Mechanisms

Our team has weekly sprints beginning on Monday and ending on Friday. Sprint progress will be communicated to the sponsor along with the next sprint’s tasks at the end of each week on Friday at 2:00pm CT over Microsoft Teams. Project management, including tracking assignments and tasks for all team members, will be conducted using [Asana](https://asana.com/?noredirect). The team will also report and communicate through a private Discord group chat that has been created.

Progress on documentation deliverables will be apparent at all stages of construction as Google Docs will be used for collaboration. If for any reason a section assigned to a team member on Asana is not being completed, the assignee will be contacted on Discord privately to work things out. Completion of tasks by team members can be indicated on Asana.



##### Figure 1.4 - Completion Indication

Progress on software development/coding will be tracked using GitHub. Repositories will be created as needed with review requirements for any pull requests added so at least one member will need to look over code being added.

## 2.7. Professional Standards

* The team meets every Friday at 1:00pm to discuss project issues and progression. Each member is expected to attend and participate during the meeting
* The team meets with Tom Hill every Friday at 2:00pm to present project progression, and ask questions
* Team members must complete work assigned every Thursday at 12:00pm
* Team members shall maintain independence and integrity in their judgment
* Team members shall be fair to and supportive with each other
* Team members are expected to communicate any problems encountered
* Team members shall be respectful with each other at all times
* Team members shall ensure the product meets the highest quality standard possible

## 2.8. Impact of the project on individuals and organizations

The project will allow chief information officers of varying organizations using the application to more easily launch it on their local machine. Prior to building the solutions this semester, the only way to launch the application on a local machine was to start both the API application and React application on separate ports and navigate to the correct localhost URL in a browser to view it. This process is complex for non-developers (such as CIOs) who would not know how to do most of these steps. With the new MSI installer, CIOs simply need to double click the installer file, wait until the application is installed, and then double click the desktop shortcut.

Additionally, these users will benefit from an expedited and far easier deployment process to Azure if they want to host the application on the cloud. Prior to the implementation of the Azure script, CIOs and other users would need to know how to deploy the applications themselves. This would mean creating the web.config files and other configuration requirements within both CIOBrain projects, creating repositories hosting the code that would be used as a reference for deployment, and then starting the application.

With the script, CIOs simply need to follow the on-screen prompts and input information. Automatic repository creation, web.config configuration, and deployment are then executed. This saves users of CIOBrain an immense amount of time especially if they have never dealt with launching on cloud providers.

The new authentication functionality for the API adds an extra layer of security that wasn’t present before. Since data transfer capabilities were implemented to transfer data from a native instance to a cloud instance, this security was necessary to protect cloud-hosted CIOBrain applications from receiving undesired data.

# 3. Requirements Specifications

## 3.1. Stakeholders for the system

Owner:

* Dr.Tom Hill
* Fellows Consulting Group, LLC

Development Team (2021 Fall Team):

* Rosie Wang
* Lena Mubarak
* Gabriel Medina
* Ritvik Divanji
* Anh Nguyen

Deployment Team (2022 Fall Team):

* Anton Horvath
* Anuja Sahu
* Scott Lorance
* Jose Colina Salas
* Saige Wright
* Kevin Lieng

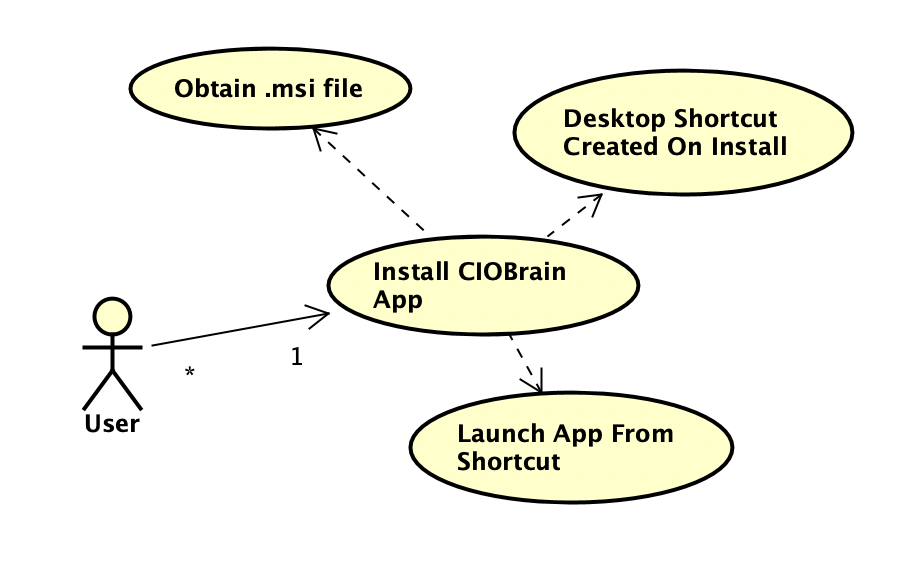
Customers:

* Business Customers and Users
* Contacts of Dr.Tom Hill

## 3.2. Use case diagrams

Requirement 1

CIOBrain shall be packaged into a .msi installer file and support one-click installation.

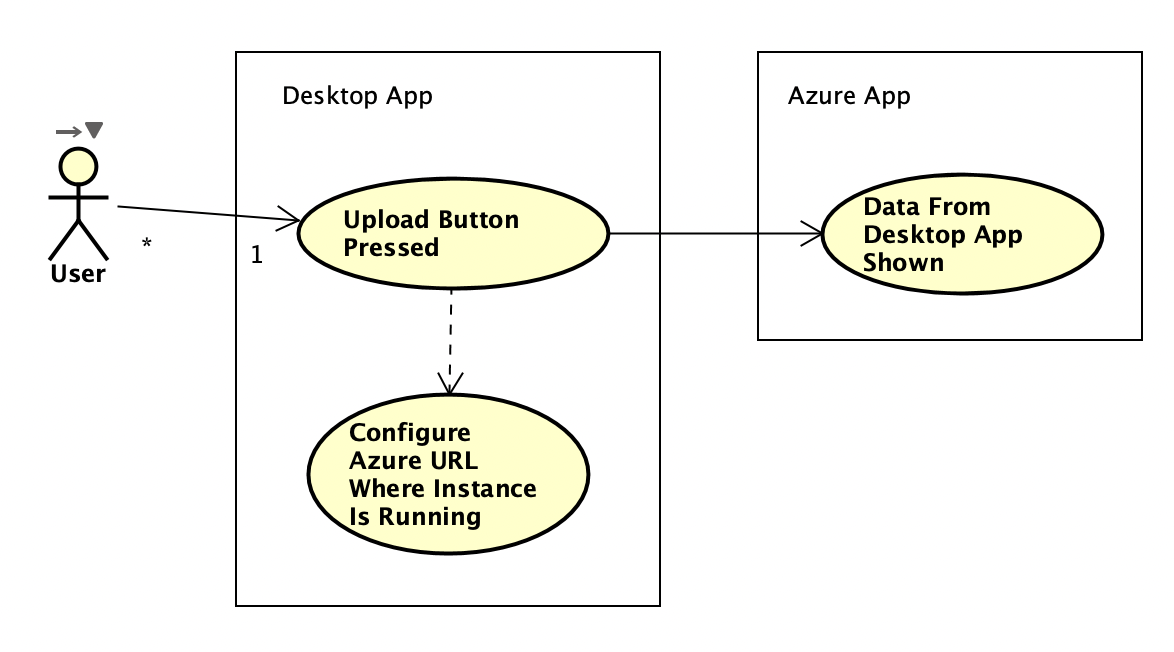


##### Figure 2.1 - Native App Installer Use Case Diagram

1. The desktop app should have a .msi installer file that can be run to instantly install the application to the Windows machine. A desktop shortcut should be created where the user can double-click to launch the application.
   1. Actors: User
   2. Entry condition: User obtains the .msi installer file.
   3. Normal Flow: User obtains .msi installer and runs it. Installer will install the desktop application and load a shortcut on the desktop. Users will be able to launch the application by double-clicking the shortcut.
   4. Exit Condition: User does not run the .msi file after obtaining it. User completes installation with desktop app installed and shortcut located on desktop.
   5. Exceptions: None
   6. Special Requirements: .msi installer is strictly Windows-compatible and cannot be run on Mac or Linux. Dedicated installers for those operating systems can be built using the ciobrain-native installer builder.

Requirement 2

CIOBrain shall have a one click transfer capability from desktop app to Microsoft Azure

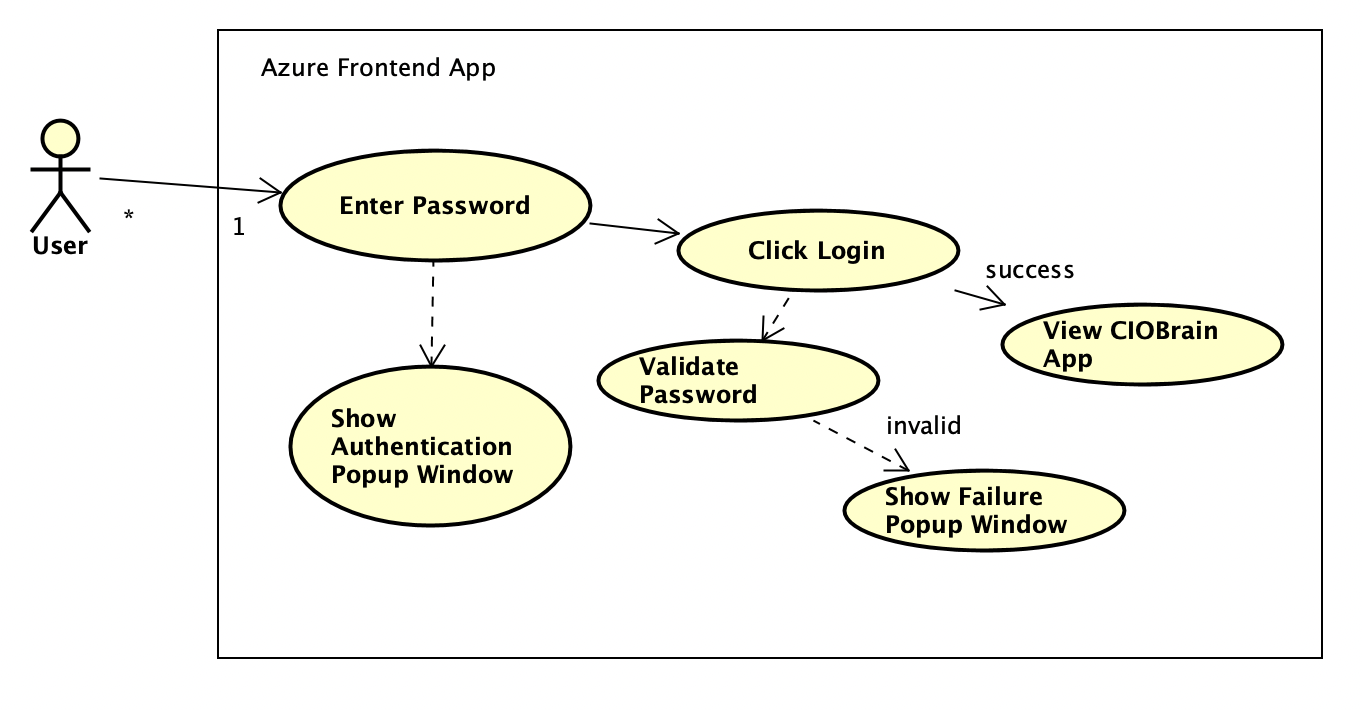


##### Figure 2.2 One Click Transfer Use Case Diagram

1. The desktop app should have an upload button to send the current relationships and data to the running app instance on Azure.
   1. Actors: User
   2. Entry condition: The user presses the upload button
   3. Normal Flow: The desktop CIOBrain takes the data currently in use for the desktop application and sends it to Azure. The CIOBrain API running on Azure will receive that data and add the assets to the existing dataset. CIOBrain running on Azure will show the newly received assets.
   4. Exit Condition: Data is successfully transferred to Azure CIOBrain API instance.
   5. Exceptions:
      1. Exception #1 - CIOBrain API not found at configured URL
         1. The user is prompted from a popup window when pressing the **Upload** button on the desktop app to input the URL where the API instance is running on Azure.
         2. The user selects **Upload** from the popup window after configuring the URL.
         3. A popup window will be displayed indicating the API could not be contacted or accessed at the given URL if not found.
   6. Special requirements: None.

Requirement 3

CIOBrain shall support login functionality utilizing a password system.

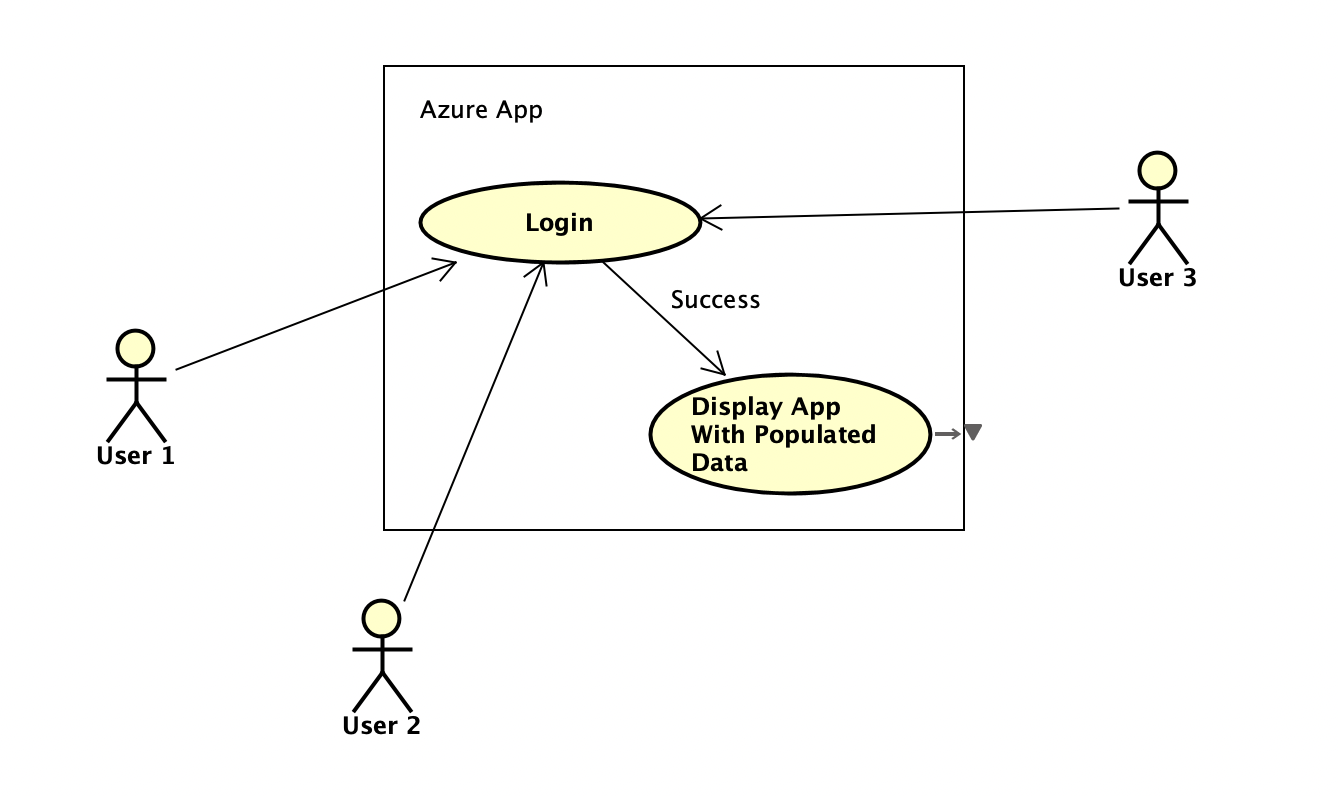


##### Figure 2.3 Login Use Case Diagram

1. A user who attempts to access the CIOBrain application running on Azure will need to input a password to access the application and its data.
   1. Actors: User
   2. Entry condition: User enters login credentials on login screen
   3. Normal Flow: User enters login credential, CIOBrain authenticates login, redirects to main CIOBrain interface that displays all assets data
   4. Exit Condition: User either log outs or the program rejects login information.
   5. Exceptions:
      1. Exception #1: Password failure
         1. User is prompted to enter a password to access the application.
         2. Password is not validated successfully and a popup window appears indicating this to the user.
         3. User is able to submit another password attempt.
   6. Special Requirements: None

Requirement 4

The instance of the app running on Azure shall be able to handle multiple users logging in and running CIOBrain at once.



##### Figure 2.4 Multiple User Use Case Diagram

1. The app running on Azure shall permit multiple users to log in individually to the application before being presented with the application if authenticated successfully.
   1. Actors: User
   2. Entry condition: User or Users login at once
   3. Normal Flow: CIOBrain on the cloud takes in a login from a user, validates the password, and permits the user on that browser to view the CIOBrain application if the credentials were verified.
   4. Exit Condition: System goes offline.
   5. Exceptions: None
   6. Special Requirements: None

## 3.2.1. Graphic use case model

## 3.2.2. Textual Description for each use case

## 3.3. Rationale for your use case model

The use cases are all derived from functional requirements that we will be implementing this semester and they do not cover requirements that have already been implemented.

Requirement 1

Users currently need to launch both the CIOBrain frontend React application and API application separately on a machine to view the application. Creating an easy to install .msi installer will allow users to install the application, have a desktop shortcut created, and click on the shortcut to run the application without having to run both frontend and backend apps separately.

Other installers for Mac and Linux should optionally also be supported alongside the .msi installer for Windows. This option will be available in the ciobrain-native installer builder application that is being developed.

Requirement 2

Users who are maintaining the application on Azure would not have the option to transfer data from the desktop application to this running instance on the cloud. They would need to instead manually transfer the Excel spreadsheets to the Azure instance by importing. To improve convenience, an **Upload** button should be added to the desktop app to send data to a running CIOBrain Azure instance with a given URL where the API is hosted.

Requirement 3

Users can access a running CIOBrain instance hosted on Azure without any authentication. When supporting multiple users that may visit the hosted application, a password system is needed to ensure unauthorized users are not accessing sensitive data. A password system when accessing the application will ensure unauthorized users are restricted from viewing the data.

Requirement 4

A single log-in workflow would result in other users accessing the application to not have to input a password since verification was already permitted to a single user. The authentication system must be dependent on the client’s browser so multiple users attempting to access the application will all need to enter credentials before being permitted in.

## 3.4. Non-functional requirements

**Reliability**

* The native application installer shall run to completion without errors 99% of the time.
* The CIO brain application hosted on Azure shall be operational as long as Azure is operational.

**Compatibility**

* The backend API shall be compatible with any PC environment.
* The backend API shall be able to be deployed onto Microsoft Azure.
* The frontend shall be able to be deployed onto Microsoft Azure.
* The frontend shall be viewable using Internet Explorer, Microsoft Edge, Mozilla Firefox, Google Chrome, and Safari Web Browser.
* The native application installer shall be an .msi installer running on Windows.
* The native application shall run on Windows.

**Portability**

* The application hosted on Azure shall be accessible in full feature form on PC and mobile devices.

**Capacity**

* The application shall handle at least 20 concurrent users in its cloud deployed form.

**Robustness**

* The native installer shall provide an error message if there is an issue with installation.
* The system shall inform the user if the native application’s API process could not be run.
* The system shall inform the user if the native application’s React process could not be run.
* The system shall inform the user if the native application could not transfer data to the running cloud application
* The system shall inform the user if a password could not be set for the cloud application
* The system shall inform the user if the password inputted was not valid
* The system shall obscure source code from users and impede reverse engineering
* The Azure hosted version shall inform the user to the causes of any errors (eg. Server outage, improper format, spreadsheet not found)

**Online User Documentation and Help**

* There shall be a sample file provided to the end-user to be able to have a correctly-formatted blank file for assets to be imported.
* A link to this sample file shall be found immediately below the Import button.
* Documentation for Azure installation and run-time instructions shall be created so users can easily deploy the application to the cloud.
* Documentation for desktop app installation using the .msi file shall be created

**Performance**

* The front end application shall be run client side to reduce server usage.
* The Azure hosted version shall not slow down within acceptable traffic volumes (<40 concurrent users)

**Usability**

* The native installer shall be able to be ran without reading any documentation
* The native installer shall produce a desktop icon that launches the application
* The native application should have simple one-click data transfer capabilities to a running instance on Azure

# 4. Architecture

## 4.1. Architectural style(s) used

Client-Server Architecture

CIOBrain was built with a client-server architecture in mind. The deployment enhancements build on top of the existing client React application requesting information from the API server. The native application bundles together both the React application and the API server and launches the API server on a separate process upon startup. The React app can then request information from the API server running on that separate process.

The cloud-based application relies on the client-server architecture heavily as CIOBrain will not be functional if the API server is not responding with the necessary data. The React client run on the browser will request data from the API server. The API server responds with all data needed to show the visualization map.

Event-Driven Architecture

Events drive the program features. These events dictate when the view of CIOBrain needs to be updated. For CIOBrain deployment enhancements, login functionality within the Azure application will rely on a user inputting information before selecting submit. This event will request access from the API server using the HTTP POST method.

Data transfer capability from the native application to an Azure-deployed instance will require an upload button that will trigger a component to display a form. Once the user submits the form, the event will request access to the API server using the provided credentials before calling another endpoint to transfer all data.

Model-View-Controller Architecture

The modelsare the assets of the CIOBrain application which are parsed from Excel spreadsheets. Each asset is represented as a JSON object. Assets are used in the data transfer functionality. The views are the components of the React application that display information and input fields. The login component and data transfer component are part of the deployment enhancements. The controllersof CIOBrain’s React and API applications handle all actions that need to be conducted which are then delegated to services. An authentication and transfer controller, along with their respective services, have been created for the deployment enhancements.

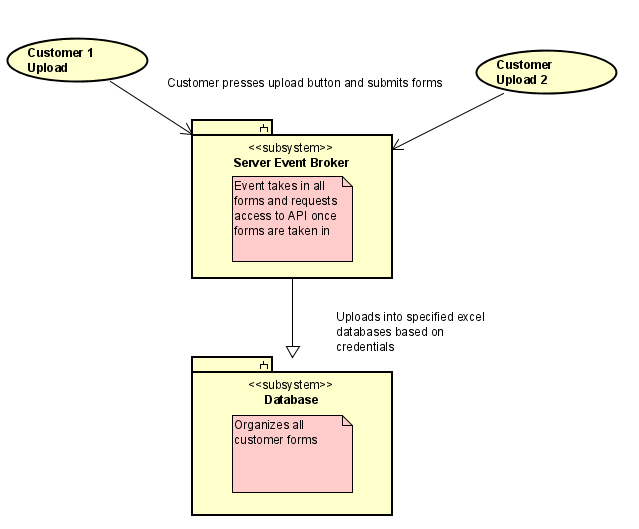
By structuring CIOBrain into three logical components, we are able to achieve low coupling. This is desirable as it allows each component to be changed independently without having to make major changes to other components. This also allows for extensibility in future iterations of CIOBrain. Future groups will find that they can add new features without having to make major changes to the architecture.

Another benefit is that CIOBrain is able to show multiple presentations of assets data. This is because MVC separates assets state from its presentation. For example, our data is currently represented in a graph that shows our different assets and their relationships with each other. Separating assets from presentation makes data transfer from the native app to an Azure instance incredibly easy. Assets can be sent as data to the API backend, and any frontend React applications connected to the API will display the data when prompted.

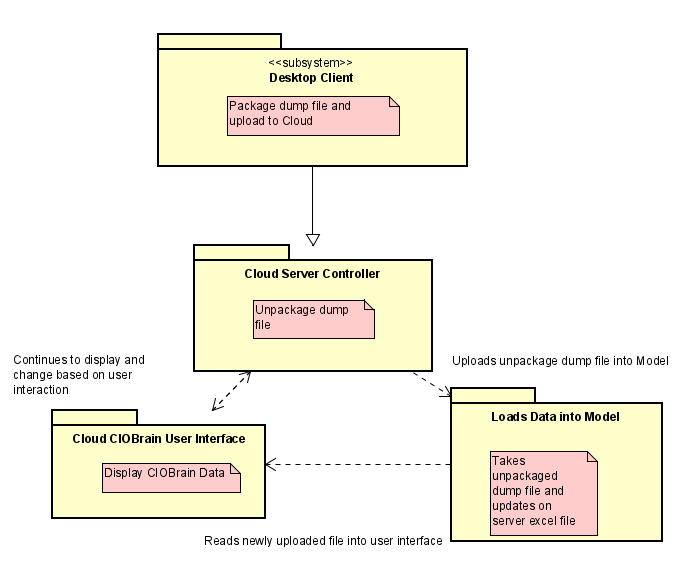
## 4.2. Architectural model

##### Figure 3.1: Architectural Client-Server Model for Base CIOBrain

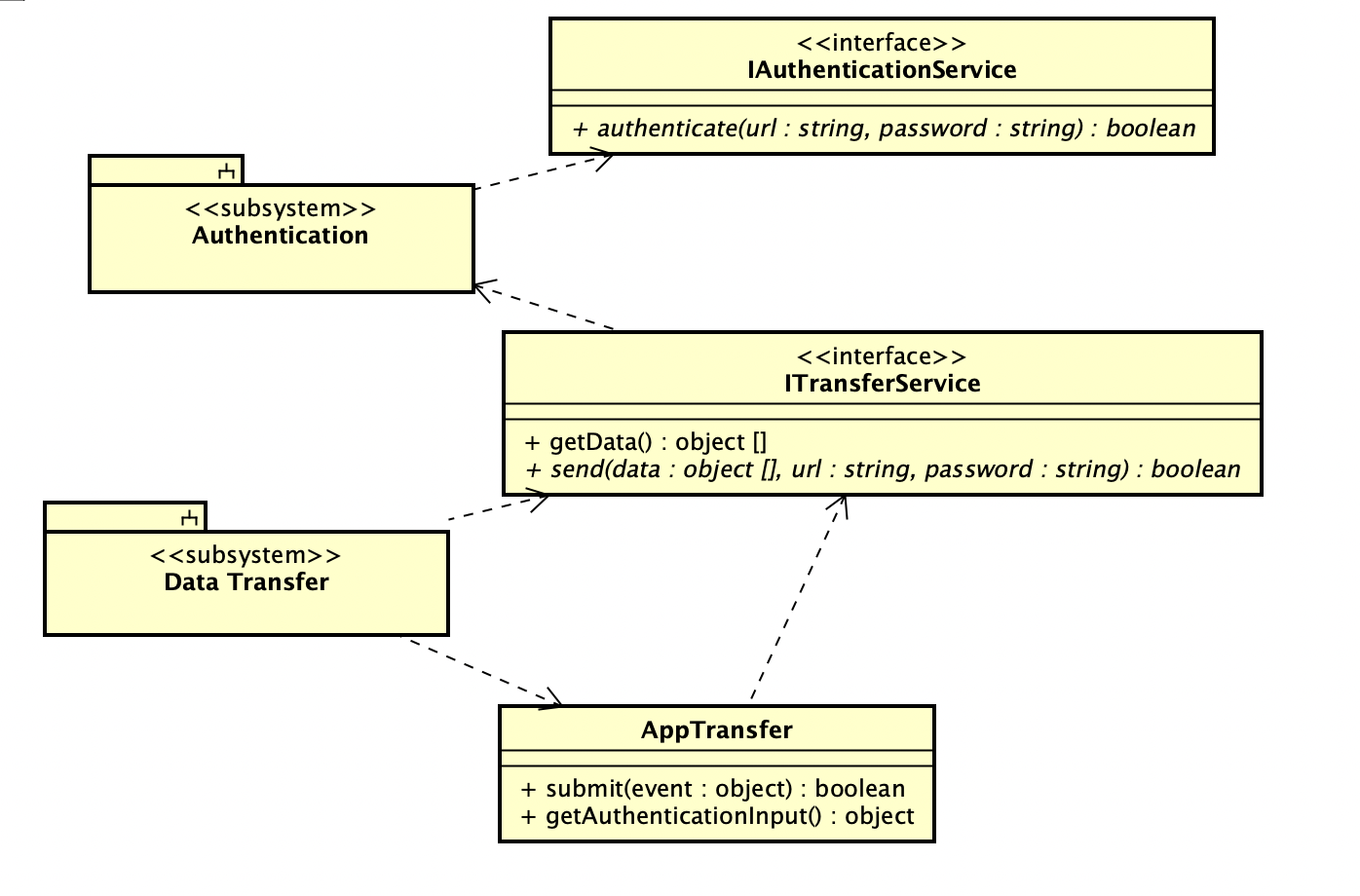
##### Figure 3.2: Authentication Architecture

* 

##### Figure 3.3: Event Driven Architecture for Data Uploads between Desktop App and Server

* 

##### Figure 3.4: Model-View Controller Structure of CIOBrain between changes in data

* 

##### Figure 3.5: Data transfer from desktop app architecture

## 4.3. Technology, software, and hardware used

Software

The deployment enhancements rely on Node.js for backend functionality. Node.js powers the backend API of the CIOBrain application and it handles the new authentication and data transfer capabilities added [1]. React powers the frontend of the CIOBrain application, handling all user interface and visual functionality in both the Azure deployed system and native application [2]. D3.js powers the visual graphing and data visualization capability of the app — creating the asset map displayed to users in the React application [3].

Electron.js powers the native application. The framework is able to build desktop applications for Mac, Windows, and Linux from existing web applications [4]. Since CIOBrain was built as a web application initially with a React frontend and Node.js backend, this framework expedited the process of converting it into a full native application.

Excel and its files are used to store data and import said data into the CIOBrain API backend. Each Excel spreadsheet imported needs to have specific column names that will be used when parsing the data into the application. Data is parsed into JSON data structures that are saved in memory when the application is running.

GitHub is being used for version control. Forked repositories have been created specifically for deployment enhancements including one for native application development, another for React enhancements, and another for API enhancements. Visual Studio Code is being used for all development.

A browser is used to display the running React application and provide an interface for the API server (Firefox, Chrome, Safari, Edge). API testing is done using Postman and terminal to ensure all endpoints are operating successfully. The cloud-based application is deployed and hosted on Azure, while the native application is run on a local machine.

Hardware

The native application requires a computer running Mac, Linux, or Windows to run successfully.

## 4.4. Rationale for your architectural style and model

The client-server architectural style decouples the backend data manipulation from the frontend components displayed to the user. This allows the application to be deployed at scale using load balancers. It also helps make the application easier to maintain as two codebases are working independently of each other and can be modified individually. Client-server can also be used for the native application by running the API application on a separate process on startup, simulating the backend being deployed [5].

The architectural style also permits authentication processing on a separate, secure server. Important credentials will not be accessible by the client in the browser, as authentication requests will need to be passed to the API server running elsewhere [6].

The event-driven architectural style is used because the entire program is reactive, and only does anything when user input prompts it to. Thinking of the architecture in a reactive way allows us to develop appropriate events to respond to user input. The event-driven approach is useful in authentication, as the request can be triggered and sent to the API server for validation. It also ensures data transfer functionality is possible when the user requests it. Using the client-server architecture, events power when requests are sent to the backend API [7].

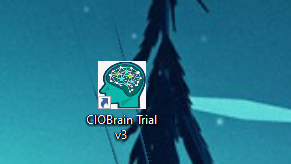
Similarly, the Model-View-Controller style is appropriate for this project’s architecture because the view of the user is influenced by the model (assets) and the controller’s manipulation and presentation of that model. The architectural model itself is designed using the aforementioned styles. In the model, the Node.js backend application is responsible for handling and storing the asset data while the client React application presents the response from the server based on the user input. MVC is how the API operates and is how the authentication and data transfer capabilities were handled [8].

# 5. Design

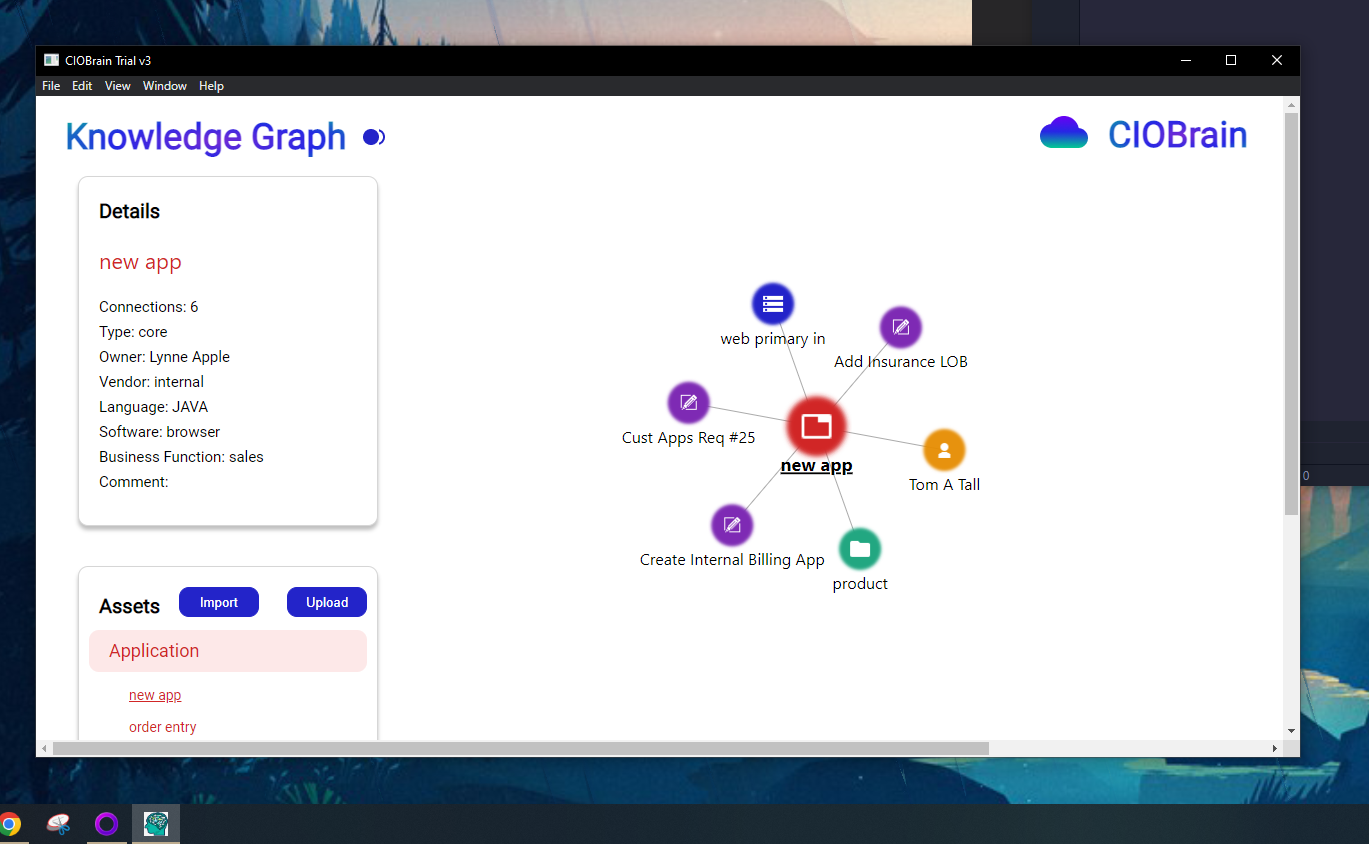
## 5.1. GUI (Graphical User Interface) design

MSI Installer

Requirement 1: The desktop app should have a .msi installer file that can be run to instantly install the application to the Windows machine. A desktop shortcut should be created where the user can double-click to launch the application.

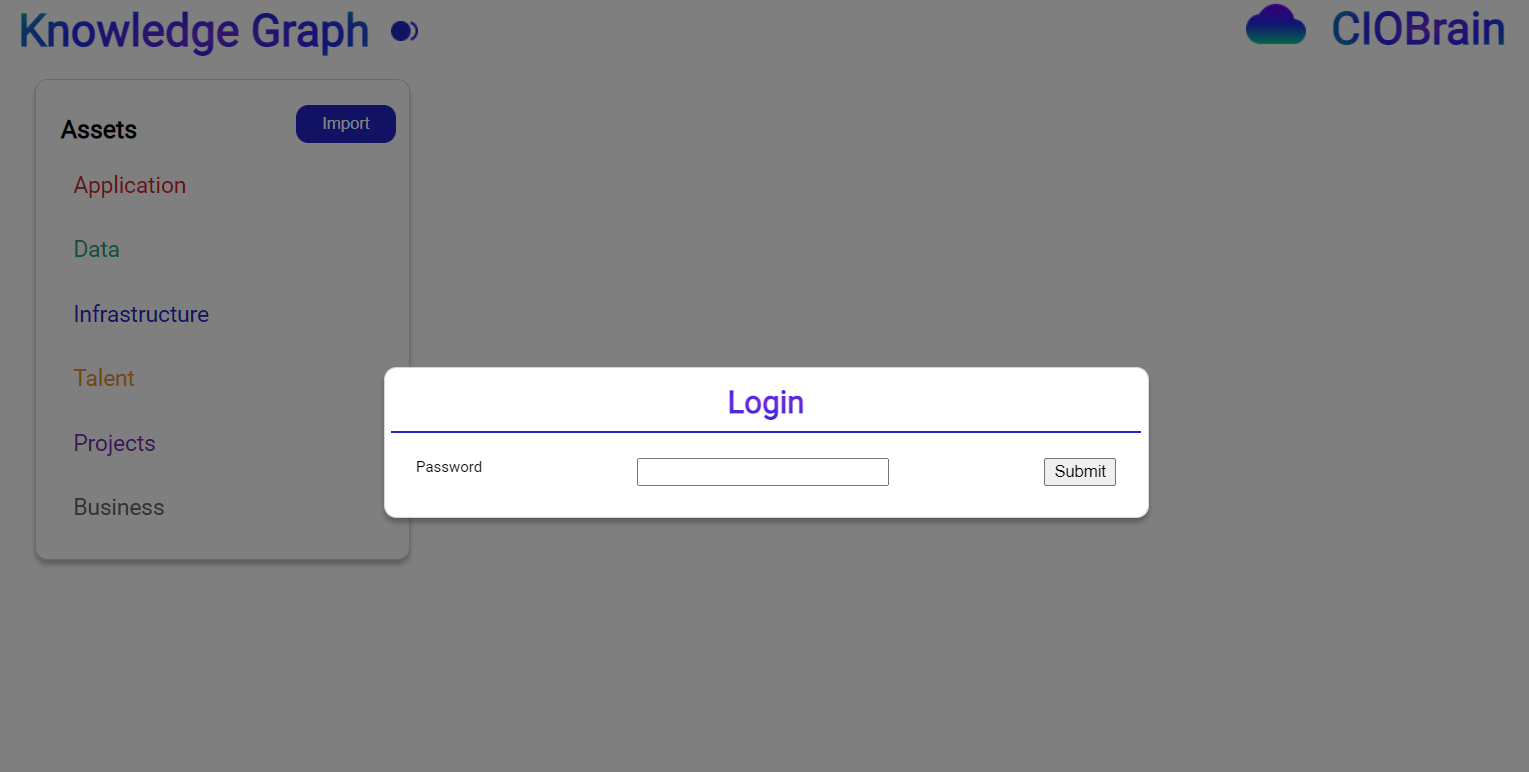
* 

##### Figure 4.1: CIOBrain Trial v3 Desktop Shortcut

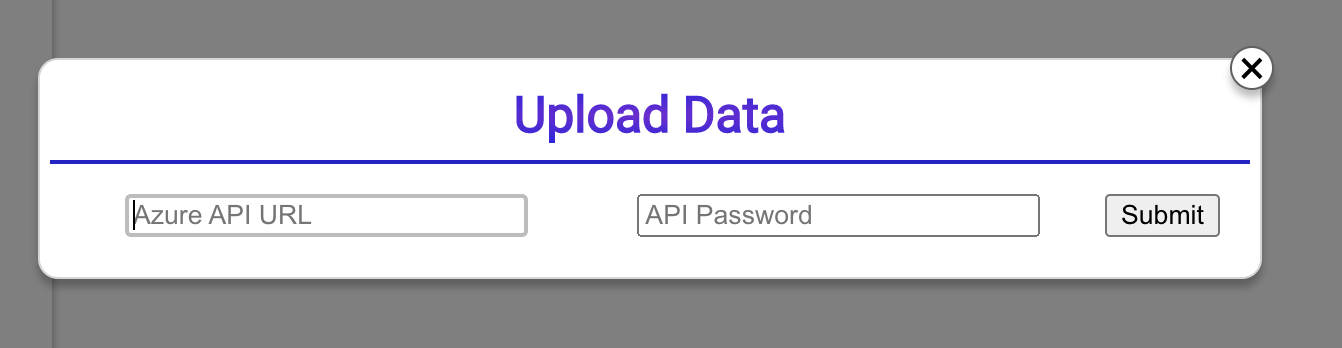
* 

##### Figure 4.2: CIOBrain Trial v3 Running on Native Process

CIOBrain Desktop Application Login Authentication

* 

##### Figure 4.3: Login Function Visualization

* 

##### Figure 4.4: Data transfer capability GUI

## 

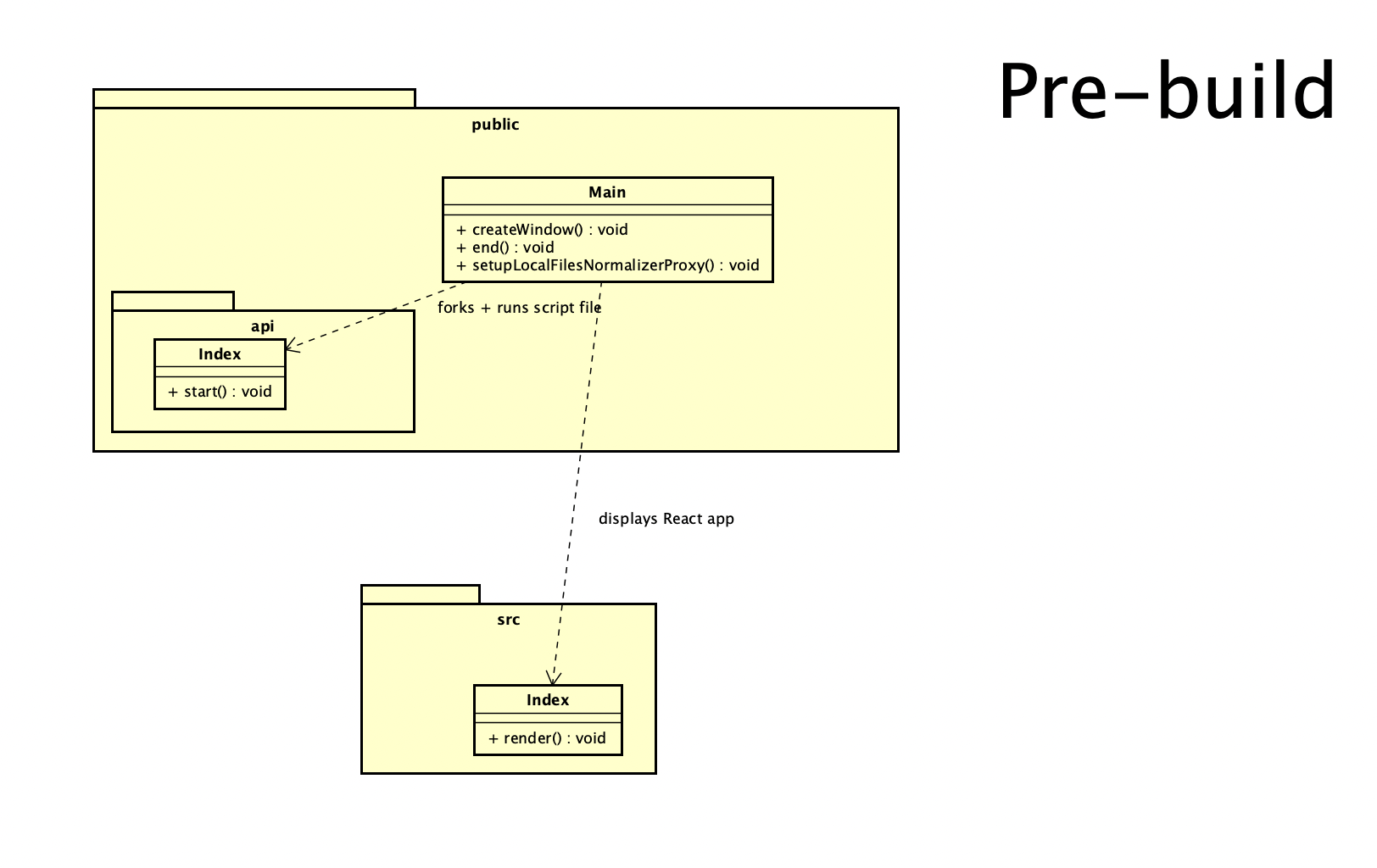
## 

## 

## 

## 5.2. Static model – class diagrams

MSI Installer

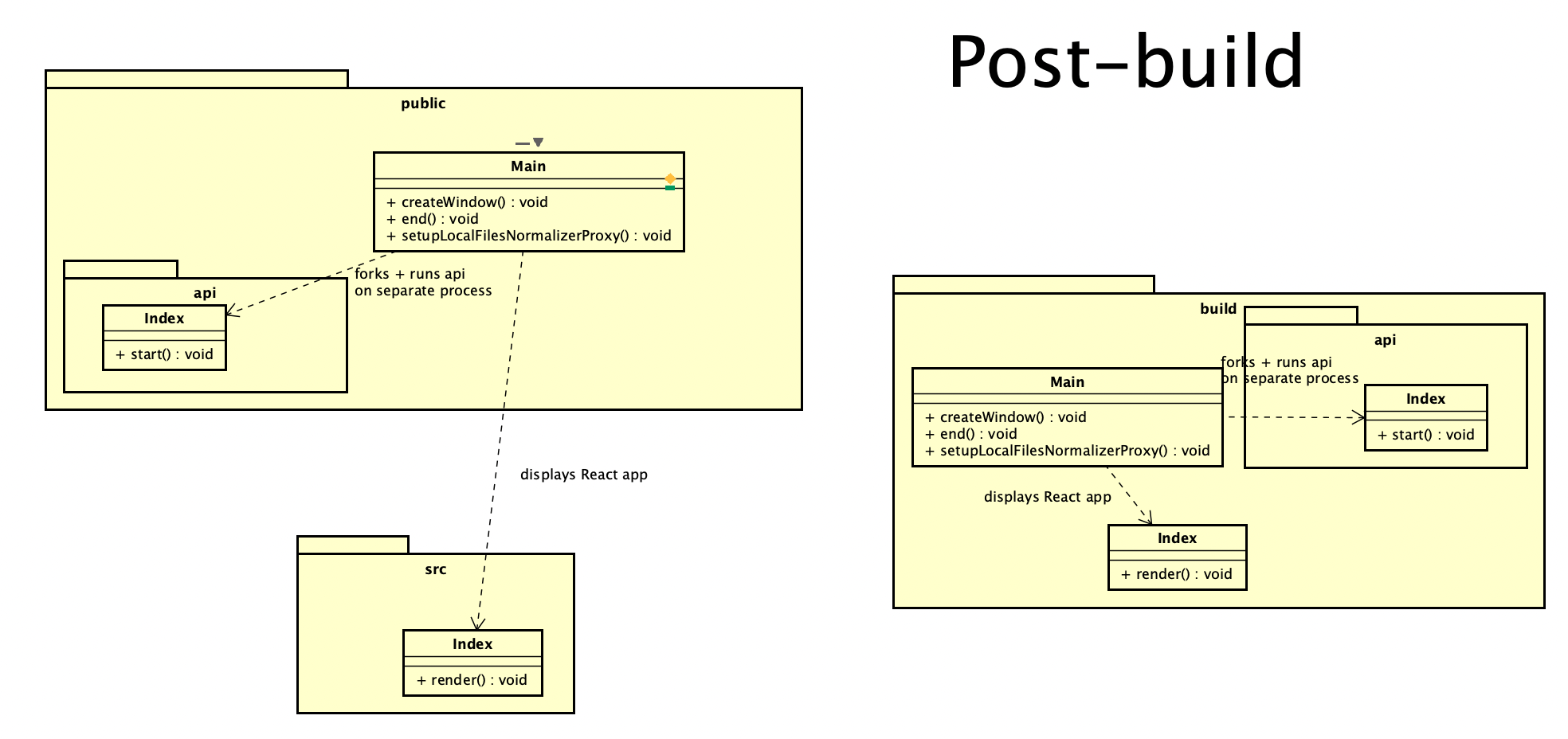
* 

##### Figure 4.5: Class diagram pre-build for MSI Installer Wrapper

CIOBrain Desktop Application Login Authentication

## 

##### Figure 4.6: Class diagram pre-build for Desktop Application Login Authentication

* 

##### Figure 4.7: Class diagram post-build for MSI Installer Wrapper

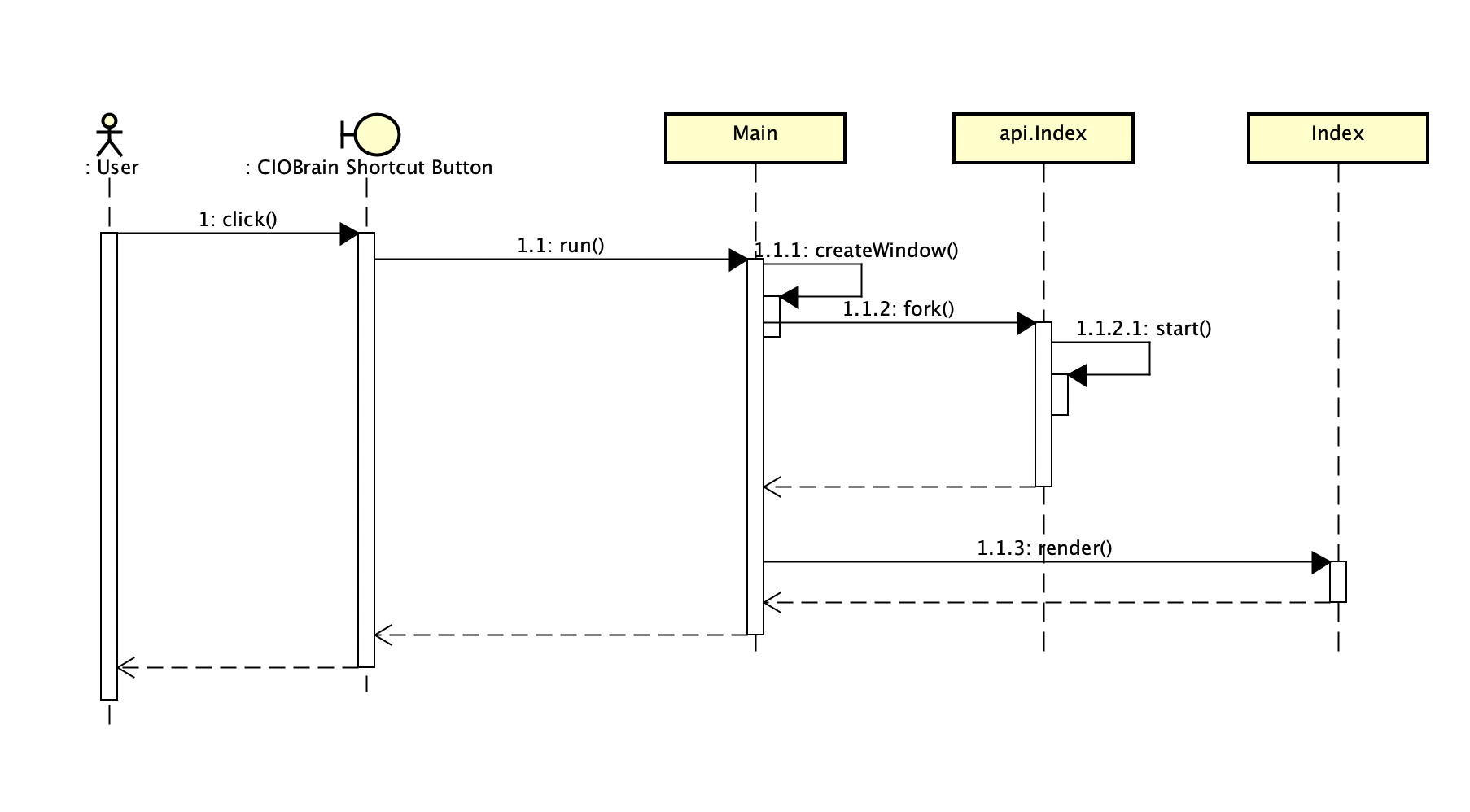
## 

## 

## 

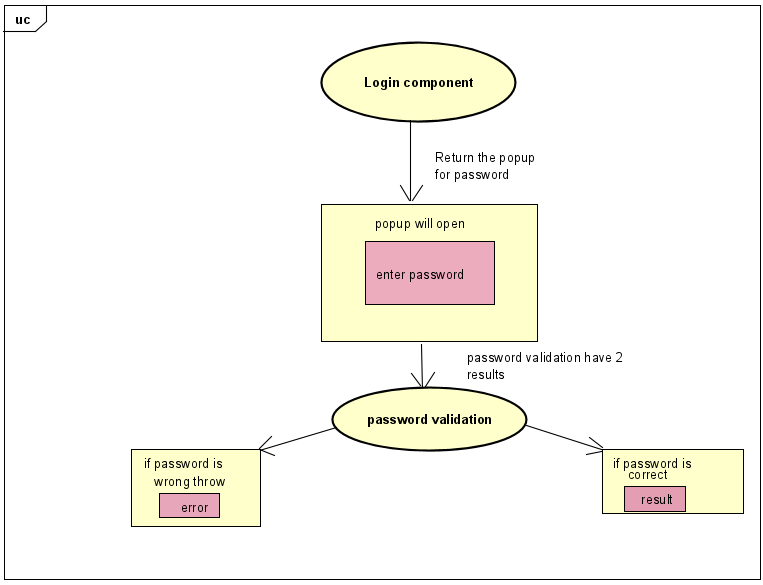
## 5.3. Dynamic model – sequence diagrams

MSI Installer

* 

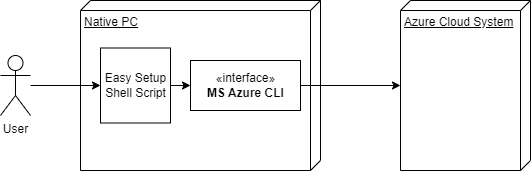
##### Figure 4.8: Sequence diagram for MSI-Installed Application Run

Azure Authentication

* 

##### Figure 4.9: Login Authentication Diagram

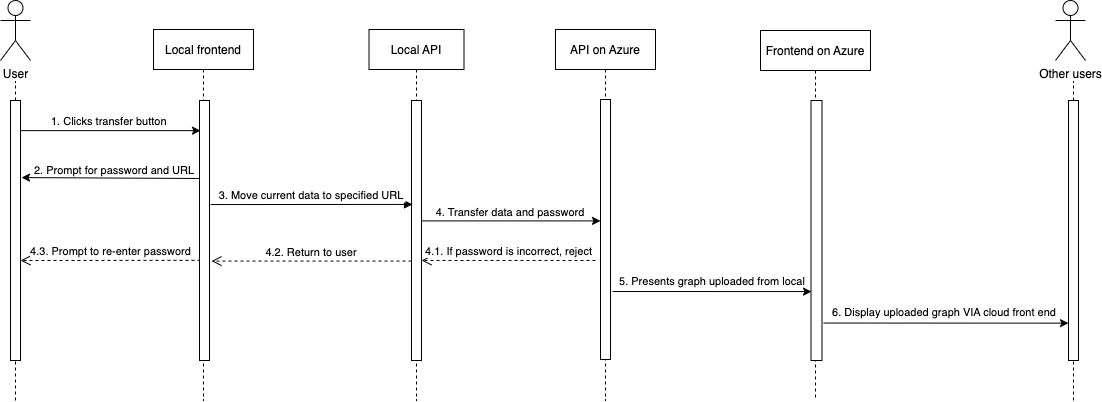
Azure Deployment Script

* 

##### Figure 4.10: Azure Deployment Communication Diagram

## 

Data Transfer

* 

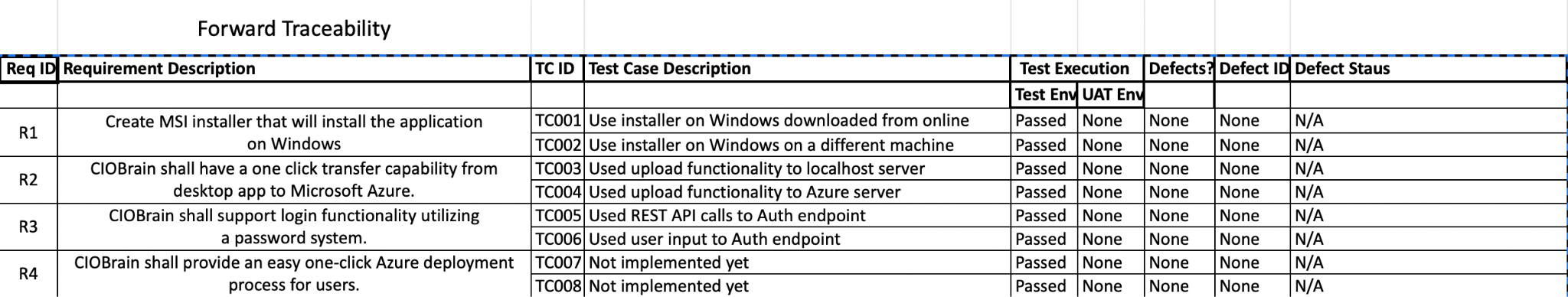
##### Figure 4.11: Sequence Diagram for data transfer from desktop app to Azure

## 5.4. Rationale for your detailed design model

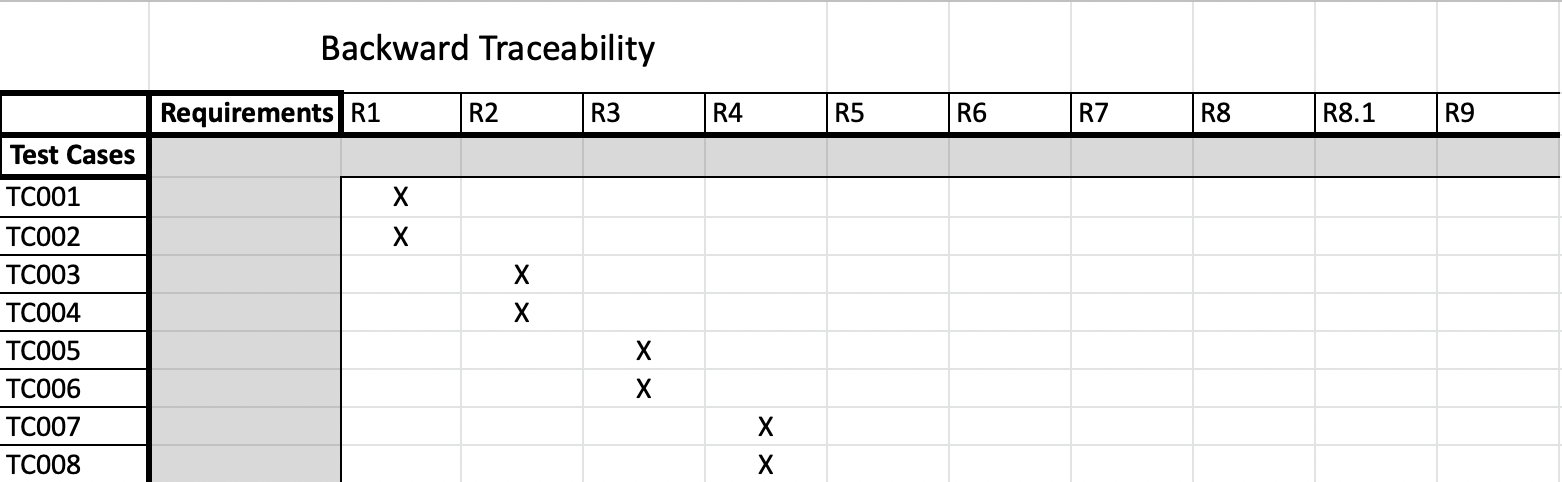
The MSI installer wrapper had to combine both existing web applications for CIOBrain and create a new application that would launch both at the same time to operate offline on a local machine. This could be done using Electron.js, a framework that uses a Main class to launch a web application running locally [9]. Since a client-server architecture was still necessary to keep the app’s functionality intact, the API application instance was packaged into this Electron application and run on a separate process upon startup [10]. The main process where the Electron app was running would display the React application, achieving the client-server architecture packaged into a single app. This design enabled requirement 1’s implementation.

The rationale for the easy Azure deployment script is to fulfill requirement requirement 2 for easy transfer to Azure, by allowing individuals without Azure experience to upload and run the service by inserting a few credentials and running the script [11]. The script would use a GitHub repository as its source for deployment, ensuring CI/CD was possible [12]. The sequence diagram that shows the process for data transfer from a user’s local machine to the Azure application starts with the user clicking on the transfer button on their local machine, which prompts them to specify the URL at which the data needs to be received at, and the password set by the user. All the currently opened graphs and password are pushed to the Azure server, but if the password entered is incorrect, then the user is prompted to re-enter the password. Otherwise, the server on Azure accepts the files and rejects any duplicates. Once all the data is transferred successfully, Azure API will display the uploaded graph via the cloud front end to any users who use the cloud hosted version.

## 5.5. Traceability from requirements to detailed design model

* 

##### Table 2.1: Forward traceability

* 

##### Table 2.2: Backward traceability

# 6. Test Plan

## 6.1. Requirements/specifications-based system level test cases

**Test Case #:** 1

**Title:** Secure Login Successfully

**Description:** On startup and refresh of the application, the user should be prompted for a login that should lock out all features of CIOBrain until login has been successfully done.

**Priority [Very High, High, Medium, Low, Very Low]:** High

**Precondition(s):** A data connection into the server exists and a password has been set

**Test Steps:**

1. Open up the CIOBrain application locally on the desktop
2. Confirm the Login screen has shown up and everything else is untouchable
3. Attempt a successful login with the correct password
4. Refresh the application after a successful login to show the login screen reappears

**Expected Result:** A login screen should show on the screen and be able to be logged in once a correct password is entered.

**Test Case #:** 2

**Title:** Incorrect Login Blocked Successfully

**Description:** On startup and refresh of the application, the user should be prompted for a login that should lock out all features of CIOBrain and should reject any and all incorrect attempts of password.

**Priority [Very High, High, Medium, Low, Very Low]:** Medium

**Precondition(s):** None

**Test Steps:**

1. Open up the CIOBrain application locally on the desktop
2. Confirm the Login screen has shown up and everything else is untouchable
3. Attempt a unsuccessful login with an incorrect password
4. Refresh the application after a successful login to show the login screen reappears

**Expected Result:** The login screen failed and has rejected the login password.

**Test Case #:** 3

**Title:** Data transfer from local machine to Azure application

**Description:** Upon clicking the ‘data transfer’ button, the files chosen for transfer from the user’s local machine should move to the Azure application’s database after the user enters the URL for destination and the correct password set for the account. However, if the data files contain duplicates, those files should be rejected.

**Priority [Very High, High, Medium, Low, Very Low]:** High

**Precondition(s):** User enters the URL and password for data transfer

**Test Steps:**

1. Import a chosen Excel database from the computer’s local machine onto the local CIOBrain application
2. Click on the Upload button
3. Login in using the correct Azure URL and password
4. Confirm that the application is starting data transfer to the database
5. Navigate to the Azure CIOBrain application
6. Reject any duplicate files during the data transfer
7. Login into the cloud application and confirm the files appear on the database of the application

**Expected Result:** The Azure application database should hold the Excel files that the user intended to transfer from their local machine.

**Test Case #:** 4

**Title:** Deploying from Powershell on a Windows system

**Description:** User authenticates github account while trying to deploy from powershell

**Priority [Very High, High, Medium, Low, Very Low]:** High

**Precondition(s):**

**Test Steps:**

1. The user opens powershell to deploy application
2. The user is prompted to input password to authenticate github account
3. The one time code generated is input incorrectly by user

**Expected Result:** Powershell displays message indicating incorrect code

**Test Case #:** 5

**Title:** Correct password for data transfer

**Description:** When the user enters the password during data transfer of files from the local machine to Azure’s database, the application must verify that it is the password set for the account. However, if the incorrect password is set, the data transfer is canceled and the user is prompted to re-enter the correct password.

**Priority [Very High, High, Medium, Low, Very Low]:** Low

**Precondition(s):** The user clicks the ‘data transfer’ button

**Test Steps:**

1. Open the CIOBrain application.
2. Click on the Upload button.
3. Login in using the correct Azure password.
4. Confirm that the password is correct.

**Expected Result:** The data transfer from the user’s local machine to the Azure application will begin.

**Test Case #:** 6

**Title:** URL user input while deploying on Windows

**Description:** A URL will be created during the deployment process, and the input needs to copy it into the github account

**Priority [Very High, High, Medium, Low, Very Low]:** Very High

**Precondition(s):**

**Test Steps:**

1. The user opens powershell to deploy application
2. A URL will be generated
3. The user needs to copy and paste URL on github

**Expected Result: URL is copied correctly and the deployment proceeds.**

**Test Case #:** 7

**Title:** Remove duplicate files in data transfer

**Description:** Upon clicking the ‘data transfer’ button, the files chosen for transfer from the user’s local machine to the Azure application should check for any duplicate files. If there exists any, then those files should be rejected.

**Priority [Very High, High, Medium, Low, Very Low]:** Low

**Precondition(s):** User enters the correct URL and password for Azure application

**Test Steps:**

1. Open the CIOBrain application.
2. Click on the Upload button.
3. Enter the URL for the data transfer destination.
4. Login in using the correct Azure password.
5. Confirm that the password is correct.
6. Confirm that the application is starting data transfer to the database

**Expected Result:** Any duplicate files should be rejected during data transfer.

**Test Case #:** 8

**Title:** Deployment attempts

**Description:** The user attempts to deploy the application more than once.

**Priority [Very High, High, Medium, Low, Very Low]:** Low

**Precondition(s):** User has already deployed application.

**Test Steps:**

1. Deploy the CIOBrain application successfully
2. User follows all steps to deploy the application again.
3. Azure already created resource groups, and the web app.

**Expected Result:** Once the user deploys the application correctly, second attempt will not work.

**Test Case #:** 9

**Title:** Native Program Run

**Description:** Once the MSI installer has successfully installed the program, the program should run successfully on the machine from the shortcut that was created on the desktop.

**Priority [Very High, High, Medium, Low, Very Low]:** Very High

**Precondition(s):** The CIOBrain installer has installed the program onto the Windows machine.

**Test Steps:**

1. Click on the CIOBrain application shortcut on the desktop.
2. Verify that the application works.
   1. Able to select assets.
   2. Able to view node information via long-press (hover equivalent).
   3. Able to see all relationships, implicit and explicit, between selected nodes.
   4. Able to import more Excel spreadsheets.
   5. Able to transfer data to an Azure instance.

**Expected Result:** The native application works correctly when launched on a Windows machine.

**Test Case #:** 10

**Title:** MSI Installer Builder

**Description:** The native application needs to be installed using an MSI installer. This MSI installer is built using a builder application that should successfully package together the API and React applications into a single native application.

**Priority [Very High, High, Medium, Low, Very Low]:** Very High

**Precondition(s):** None

**Test Steps:**

1. Clone the project onto a local machine.
2. Run ‘npm install’ to install all dependencies of the project.
3. Run ‘npm run electron:package:win’ to build the MSI installer for Windows.
4. Check the ‘dist’ folder once complete.

**Expected Result:** The MSI installer should be present within the ‘dist’ folder of the project.

**Test Case #:** 11

**Title:** Dynamic Fields On Excel Spreadsheets

**Description:** The Excel spreadsheets that are imported into the CIOBrain application should be able to support custom-named fields along with the preset fields needed to import.

**Priority [Very High, High, Medium, Low, Very Low]:** Medium

**Precondition(s):** CIOBrain application is running.

**Test Steps:**

1. Import Excel spreadsheet with new custom fields.
2. Check to make sure the fields do not break the graph.
3. Fields should be shown on the UI.

**Expected Result:** The custom fields should not break the application and should appear on the application UI.

## 6.2. Traceability of test cases to use cases

| **Test Case #** | **Associated Use Case** |
| --- | --- |
| 1,2 | CIOBrain shall support login functionality utilizing a password system. |
| 3,7 | CIOBrain shall have a one click transfer capability from desktop app to Microsoft Azure |
| 6 | The instance of the app running on Azure shall be able to handle multiple users logging in and running CIOBrain at once. |
| 4,5,8 | Deployment |
| 9, 10, 11 | There should be a one click MSI installer for smooth and easy installation |

##### Table 3.1: Test Cases and Their Associated Use Cases

## 6.3. Techniques used for test generation

**Boundary value analysis:** Testing the successful scenario, then the bordering unsuccessful scenarios (examples: T01, T02).

**Specification-based technique**: Test cases are generated in part from the formally defined requirements specification. (T03, T04, T05, T10)

**Black-box testing:** Test cases that will simply be tested from outside of the development environment (T06)

## 6.4. Assessment of the goodness of your test suite (Which metrics were used for such assessment?)

The test suite created was an integration-heavy collection that thoroughly tested the full functionality of all capabilities developed throughout the semester. Since most objectives were non-programming based, this integration heavy approach proved to be effective in our project.

**MSI Installer**

All test cases for the MSI Installer implementation passed successfully, with the CIOBrain application running locally on a Windows machine without issues. Building the MSI installer file, which would be conducted by Dr. Hill and other stakeholders responsible for distributing the application (not end users), was successful with no issues.

**API Authentication**

All test cases for login capabilities passed successfully. Users were unable to import data into a CIOBrain application hosted on Azure without correct login credentials. Accessing the CIOBrain application with an authentication-secured API requires login input as expected.

**Data Transfer**

All test cases for data transfer passed successfully. Duplicate files were not imported into the cloud instance. Password authentication was required to transfer the data to the cloud instance. Moving from a native application instance to a cloud instance was successful.

**Azure Script**

All test cases for the Azure script were successful. Multiple script deployments with the same information inputted in the file were unsuccessful as expected. Required unique fields that were inputted as not unique failed as expected. Multiple script deployments back-to-back were successful when unique fields were changed.

All objectives and deliverables corresponding to them for the project were tested successfully. The test suite was effective for our project goals as little programming was required during the course of our project — just scripting, documentation, and research.

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# References

[1] CIOBrain, “CIOBrain/ciobrain: The front-end of the CIOBrain application.,” GitHub. [Online]. Available: https://github.com/CIOBrain/ciobrain. [Accessed: 01-Dec-2022].

[2] CIOBrain, “CIOBrain/ciobrain-API: The back-end of the CIOBrain application.,” GitHub. [Online]. Available: https://github.com/CIOBrain/ciobrain-api. [Accessed: 01-Dec-2022].

[3] M. Bostock, “Data-driven documents,” D3.js. [Online]. Available: https://d3js.org/. [Accessed: 01-Dec-2022].

[4] “Introduction: Electron,” Electron RSS. [Online]. Available: https://www.electronjs.org/docs/latest. [Accessed: 01-Dec-2022].

[5] “What is client server architecture? - differences, types, example,” Intellipaat Blog, 30-Mar-2022. [Online]. Available: https://intellipaat.com/blog/what-is-client-server-architecture/. [Accessed: 01-Dec-2022].

[6] “Server-Side Authentication,” IIOP security. [Online]. Available: https://docs.oracle.com/cd/A97335\_02/apps.102/a83725/secure5.htm. [Accessed: 01-Dec-2022].

[7] EdPrice-MSFT, “Event-driven architecture style - azure architecture center,” Azure Architecture Center | Microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven. [Accessed: 01-Dec-2022].

[8] “MVC - MDN Web Docs glossary: definitions of web-related terms: MDN,” MDN Web Docs Glossary: Definitions of Web-related terms | MDN. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/MVC. [Accessed: 01-Dec-2022].

[9] C. Delgado, “How to create a MSI installer in windows for an electron framework application,” Our Code World, 19-Jul-2019. [Online]. Available: https://ourcodeworld.com/articles/read/927/how-to-create-a-msi-installer-in-windows-for-an-electron-framework-application. [Accessed: 01-Dec-2022].

[10] “Node.js V19.2.0 documentation,” Child process | Node.js v19.2.0 Documentation. [Online]. Available: https://nodejs.org/api/child\_process.html. [Accessed: 01-Dec-2022].

[11] Dbradish-Microsoft, “Azure Command-Line Interface (CLI) - overview,” Azure Command-Line Interface (CLI) - Overview | Microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/cli/azure/. [Accessed: 01-Dec-2022].

[12] Msangapu-Msft, “CLI: Deploy an app from GitHub - Azure App Service,” CLI: Deploy an app from GitHub - Azure App Service | Microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/azure/app-service/scripts/cli-deploy-github. [Accessed: 01-Dec-2022].