**CIOBrain Deployment**

Architecture Documentation

Anton Horvath

Anuja Sahu

Scott Lorance

Kevin Lieng

Jose Colina Salas

Saige Wright

# 

# Abstract

This document details architectural models and styles used in the deployment enhancement of the CIOBrain project. Architecture describing portions of the application not applicable to the current team’s objectives will not be detailed. Existing architecture created by previous teams will not be described. The document will also present technology being used to satisfy current deployment requirements including software and hardware. Rationale behind each style or model used is also included.

# Table of Contents

[**Abstract**](#_4v8tbippnkd1) **2**

[**Table of Contents**](#_ila7beigog9s) **3**

[**List of Figures**](#_421o1p2momgb) **4**

[**List of Tables**](#_50j3oncgjo0) **5**

[**Introduction**](#_7l8jc64lps5h) **6**

[**Architectural Style(s) Used**](#_prpb6aiwhwgt) **7**

[**Architectural Model**](#_te452246pu3t) **8**

[**Technology, Software, and Hardware Used**](#_pok7ysnmlb6) **11**

[**Rationale for Architectural Style and Model**](#_jau0amcjwx0) **12**

[**Evidence the Document Has Been Placed Under Configuration Management**](#_r9jje5h3kudz) **13**

[**References**](#_xmechgf7qktm) **14**

# List of Figures

Figure 1: Client-Server Architectural Model 9

Figure 2: Authentication Architecture 9

Figure 3: Event Driven Architectural Model 10

Figure 4: Model-View Architectural Model 11

Figure 5: Data Transfer from Desktop App Architecture 12

Figure 5: Screenshot of Shared Document 15

# List of Tables

N/A

# Introduction

The purpose of this software architecture document for CIOBrain is to show the architecture used for the deployment enhancements. The software architecture determines how the software will be structured and what programming conventions will be followed.

This document describes the architectural styles used in the CIOBrain deployment enhancements, an architectural model of the systems, the technologies used in implementing this model, and the rationale for the architecture. The document includes the following:

* Architectural Style(s) Used  
  This section describes which architectural styles were used for CIOBrain deployment enhancements.
* Architectural Models  
  This section showcases the models of the CIOBrain deployment enhancements under the architectural style.
* Technology, Software, and Hardware Used  
  This section lists and describes the technology, software, and hardware used to implement this project.
* Rationale for Architectural Style and Model  
  This section gives the reasoning and rationale behind why the architectural style and model were chosen.
* Evidence the Document has been placed under Configuration Management

# 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.

# 

# Architectural Model

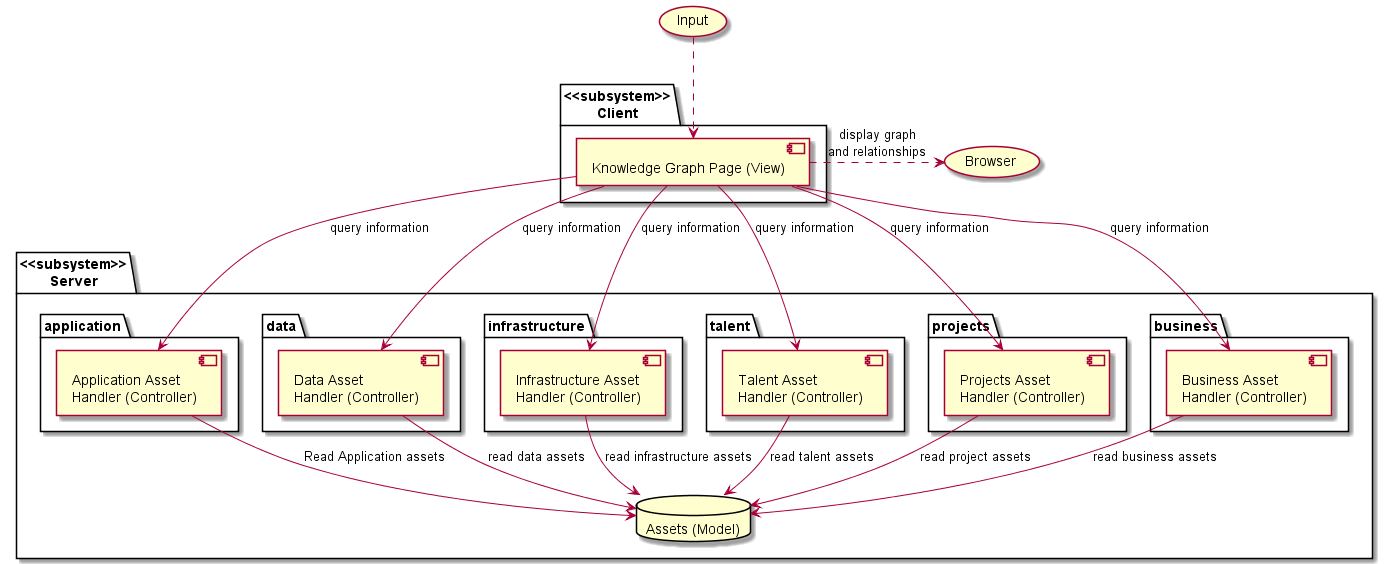


Figure 1: Architectural Client-Server Model for Base CIOBrain

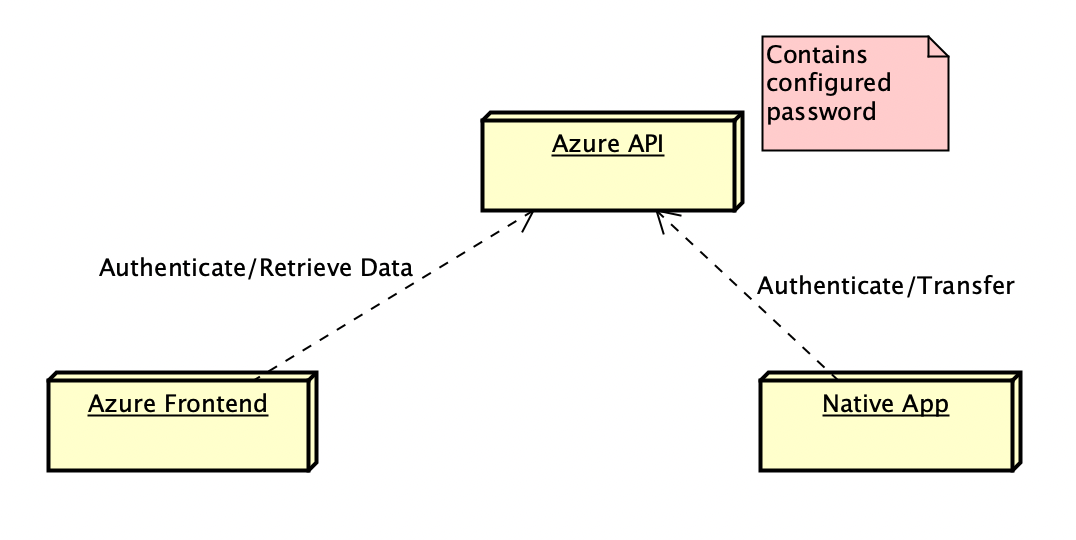


Figure 2: Authentication Architecture

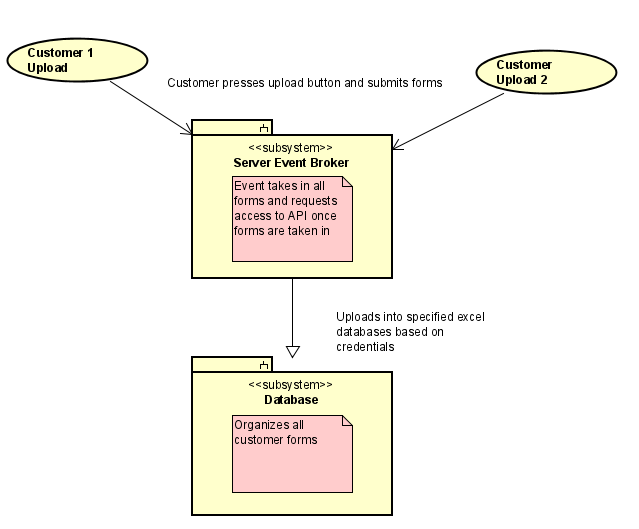


Figure 3: Event Driven Architecture for Data Uploads between Desktop App and Server

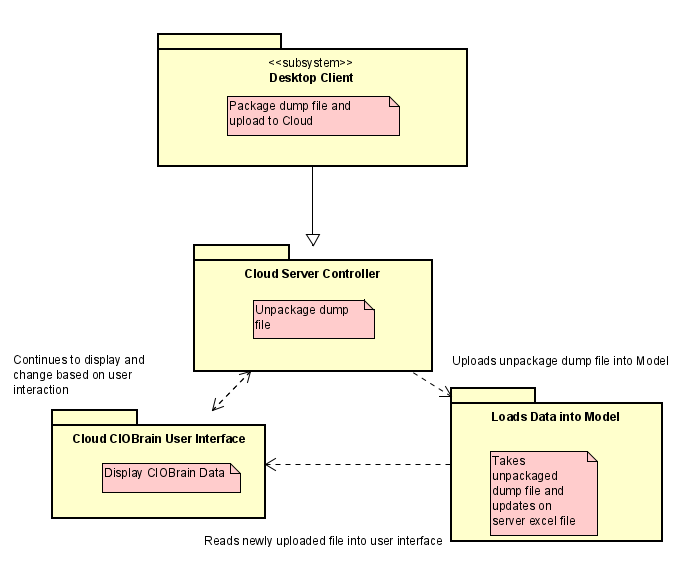


Figure 4: Model-View Controller Structure of CIOBrain between changes in data

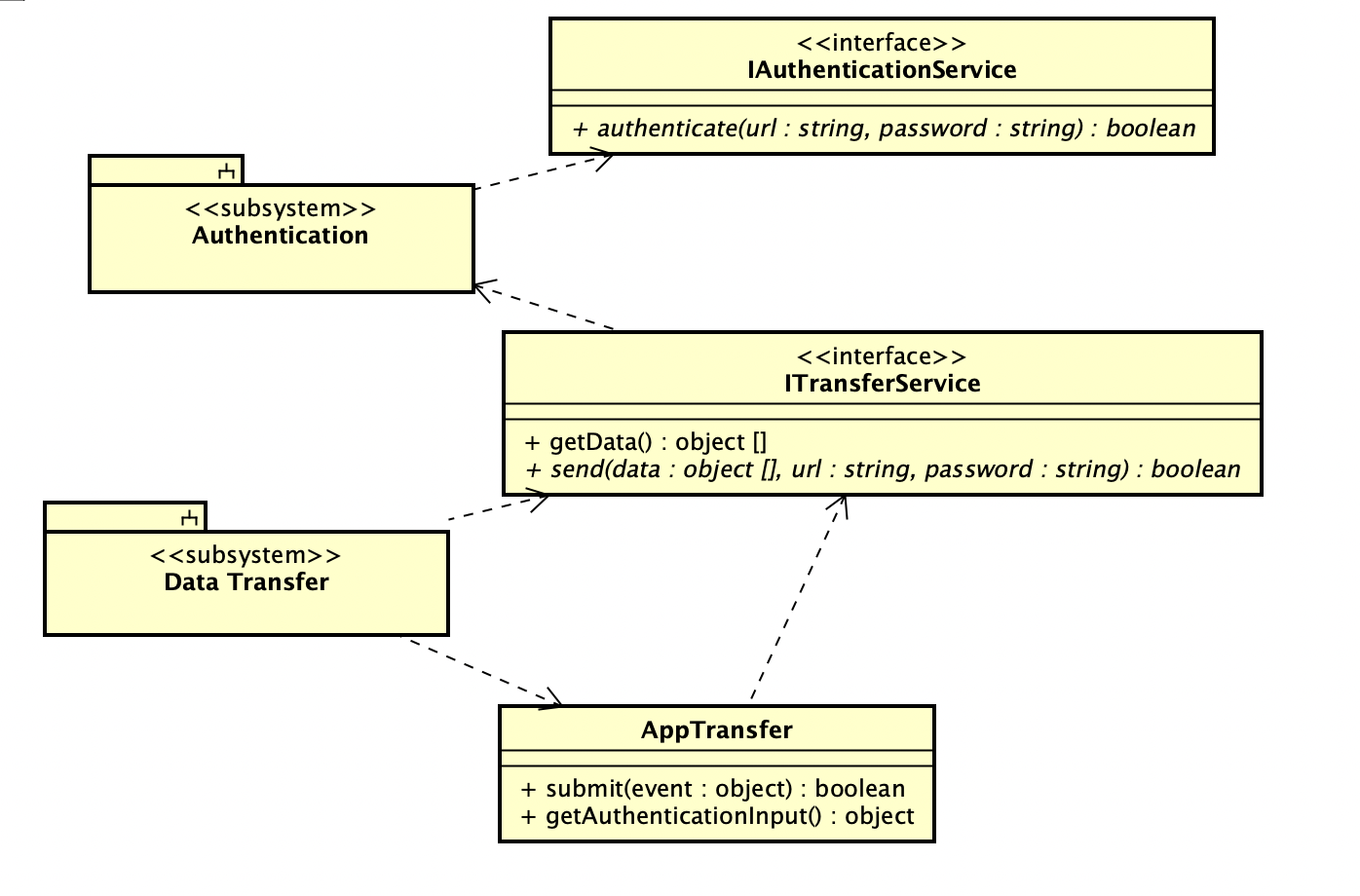


Figure 5: Data transfer from desktop app architecture

# 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. React powers the frontend of the CIOBrain application, handling all user interface and visual functionality in both the Azure deployed system and native application. D3.js powers the visual graphing and data visualization capability of the app — creating the asset map displayed to users in the React application.

Electron.js powers the native application. The framework is able to build desktop applications for Mac, Windows, and Linux from existing web applications. 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.

# Rationale for 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.

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.

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.

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.

# Evidence the Document Has Been Placed Under Configuration Management

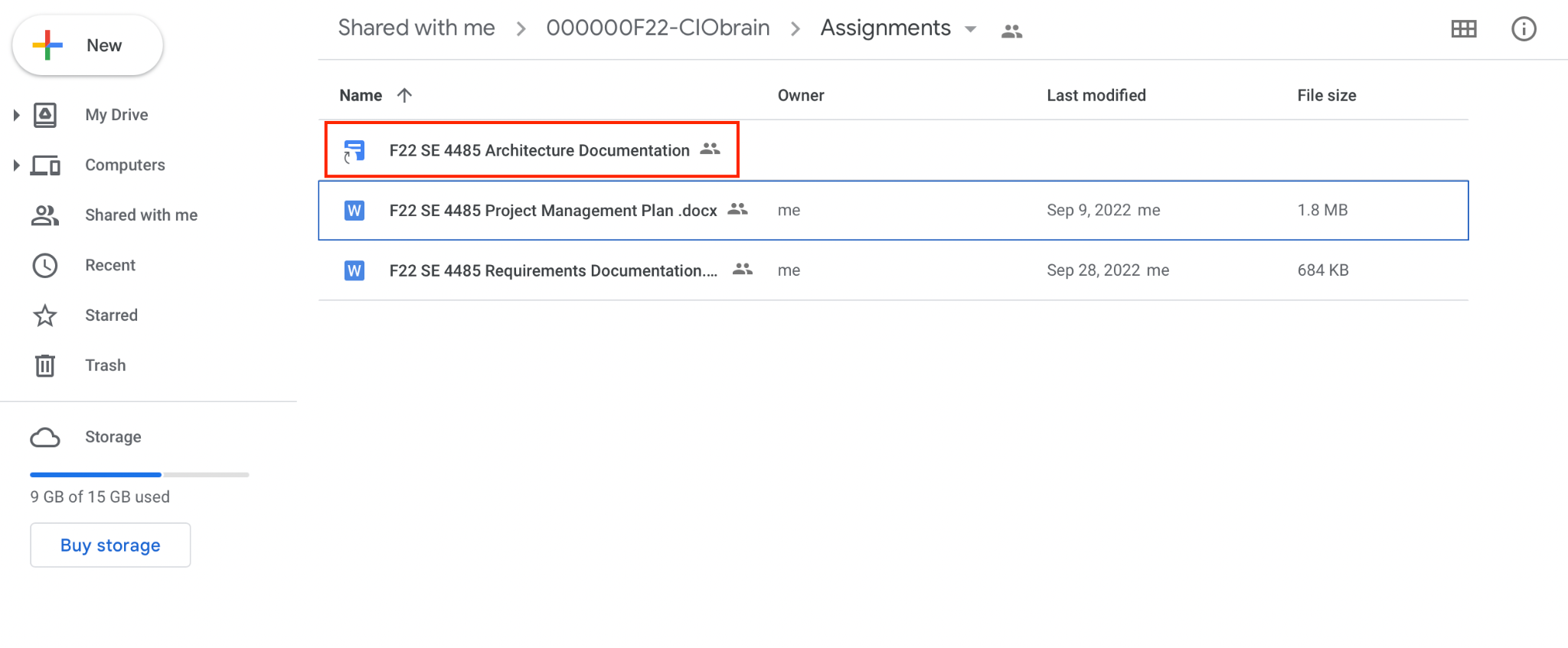


Figure 5: Screenshot of Shared Document

# References

N/A