Software Architecture

Specification

for

Group Testing Environment

v1.0

15/04/2022

Prepared by

Filipe Lagrenade,

Sanjay Williams,

Jia Son Pow, and

Isaac Ellis

Capstone Group I3

**Table of Contents:**

[**Introduction  
Contributors: Jia Son Pow**](#_yhowolea0j7d) **5**

[**Purpose of this Document**](#_xz16esheokk2) **5**

[**Document Scope**](#_j3q8voc31st2) **5**

[**Terminology**](#_96dno21lb1hb) **5**

[Acronyms (if needed)](#_pszuj9kd194p) 5

[**Architecture Overview  
Contributors: Jia Son Pow, Filipe Lagrenade, Isaac Ellis**](#_jxomcd5y0pn4) **6**

[**Identification**](#_4eimjukflpa6) **6**

[**Goals and Objectives**](#_e8uc2zjge51x) **6**

[Compatibility and Interoperability](#_4lk65u3xhtsf) 6

[Extensibility](#_s8xqe6pyz033) 7

[Portability and Platform Independence](#_b97rbfeeod9n) 7

[**The Architectural Model**](#_cz84vjcfu1ne) **8**

[Architectural Concepts and Representations](#_85gn4oazxgc1) 8

[**Software Architectural Overview**](#_t0z1el2s2yvz) **9**

[Component Interaction Diagram](#_nrbe7zjestsd) 9

[Component Interaction Diagram Description](#_tg0jra7f7u94) 10

[**Architectural Specification - UC Client Framework   
Contributors: Jia Son Pow**](#_vbi8rx8izxvt) **12**

[**Introduction**](#_5pp42nwjblkq) **12**

[**Design Methodology**](#_bxfsxnktjmt3) **12**

[Vue MVC Architecture](#_o4wj4p50dj0h) 12

[**System Structural Overview**](#_hrdy70ja8npb) **12**

[Software Component View](#_f5xge9d0p0du) 12

[Component Details](#_ys3j24gr1joe) 14

[List of Product Deliverables](#_nccuzxcia99z) 14

[**Theory of Operation**](#_a9nugil8b7wv) **15**

[Installation](#_sevca0axlqy9) 15

[Background processes](#_2b05ujgb979q) 15

[Vue.js](#_owaqu5xilyjg) 15

[Global notifications](#_nuhied7y8ukb) 15

[Error Handling](#_rijtp8ebchnb) 16

[**Architectural Specification - Student Client Framework  
Contributors: Filipe Lagrenade**](#_4rkhicweut6k) **17**

[**Introduction**](#_2btd7sytdqv3) **17**

[**Design Methodology**](#_edf14u9302cg) **17**

[Vue MVC Architecture](#_z3gmy1glfdva) 17

[**System Structural Overview**](#_w0aun2u5krxw) **18**

[Software Component View](#_62joiveusqlh) 18

[Component Details](#_1fpbupdzi3g6) 19

[List of Product Deliverables](#_1ax7qm58anr2) 19

[**Theory of Operation**](#_on9fn2xl7k9m) **20**

[Installation](#_um1758mf9kfn) 20

[Background Process](#_qw753yko8355) 20

[RSync](#_n4h5vl4htttb) 20

[Vue.js](#_5ehy2tiyibx1) 20

[Notifications](#_8uh3v3obozdc) 21

[Error Handling](#_y0cdxcikvrzg) 21

[**Architectural Specification - The Backend Application  
Contributors: Isaac Ellis**](#_so4zb4ic96lz) **22**

[**Introduction**](#_gs2n5tm6g128) **22**

[**Design Methodology**](#_2m51wauvqvm) **22**

[System Structural Overview](#_um1natx1iowh) 23

[Software Component View](#_rwkpdx2gc61l) 23

[Data Model](#_fd71p9a192j4) 23

[Component Details](#_dlequca2xyzi) 25

[List of External Interfaces](#_2hfuix5pvhsc) 25

[List of Product Deliverables](#_395sq5l3ehkf) 27

[**Theory of Operation**](#_rb3l4o8v1nq8) **27**

[Installation](#_yc2m0va2qdmr) 27

[Identity and Access Management](#_839u1usbj3av) 28

[Rsync Management](#_scd44npd5x2p) 28

[Error Handling](#_as6ursbldsye) 28

[**6. Architectural Specification - Database  
Contributors: Sanjay Williams, Isaac Ellis**](#_amyro3wj1q4h) **29**

[**Introduction**](#_ec6hiqu1pvre) **29**

[**System Structural Overview**](#_ir4co3chm4tw) **29**

[Software Component View](#_e072h6bbi9f4) 29

[Component Details](#_lrb6gp6ivkmq) 30

[**Theory of Operation**](#_wfd8ljvcxrqn) **31**

[Migration and Initiation](#_dak2k1yclsof) 31

[SQL Commands](#_1g5pj13y7dpp) 31

[**Security and Ethical Implications Summary  
Contributors: Jia Son Pow**](#_dnryzxgi8iy9) **32**

[**Debugging, Error Logging, and Performance Monitoring  
Contributors: Sanjay Williams**](#_xo1gy189k4ev) **33**

[Performance Monitoring](#_1gbk02owgj3c) 33

[Performance Counters](#_24fxczppdguo) 33

[**Debug**](#_b2cem4kgtzdh) **34**

[Debug Output](#_x64qvzimt9wh) 34

[Assertions](#_z6rwpi30v46) 34

[**Error Logging**](#_op50j1uhofks) **35**

[Error Handling](#_ckv77op3ry96) 35

# 

# Introduction Contributors: Jia Son Pow

## Purpose of this Document

This document provides a full-scale overview of the exam invigilation application. It outlines the major structural components that are used to support the application in various operating systems. Furthermore, it includes an overview of error handling, performance monitoring and security/ethical implications that will be used to maintain the product.

The target audience of this document includes software design and personnel involved for developing the exam invigilation software. Anyone who has an interest in learning about the application’s high-level software structure should read this document.

## Document Scope

This SAS introduces the structure and behaviour of the entire software stack. The exam invigilation system is being developed by us to support coding-based assessments that will take place in the newly-built Curtin Super Lab.

## Terminology

“Work Folder” refers to the Linux directory where the student creates and edits files during an assessment.

### Acronyms (if needed)

## 

# Architecture Overview Contributors: Jia Son Pow, Filipe Lagrenade, Isaac Ellis

## Identification

This document demonstrates the structural overview of the technology stack that will be used to support the project “Curtin Super Lab Exam Invigilation Application”.

The project will be developed by the Capstone Group I3 and aims to better the experience of coding-based assessments for students.

## Goals and Objectives

The goal and objective of the software architecture are to describe all the software components in comprehensive detail. This document aims to provide a good idea to future software developers of how the software architecture for a project looks and how the different components interact with each other. It is also a guide for any further extensions to be implemented on the Exam Invigilation Application in the future.

### Compatibility and Interoperability

Compatibility and Interoperability were both taken into careful consideration when the software architecture for this project is designed.

Compatibility: Components which include various frameworks and libraries across the entire technology stack were selected carefully so that they are congruent with one another when communication occurs.   
  
Interoperability: To ensure the interoperability between the frameworks/libraries in the entire technology stack, Javascript-based frameworks have been chosen so that they will not have any problems sharing important information. Vue.js will be used by both the student and UC clients so that the PostgreSQL-based backend server is able to respond to their respective API requests without running into specification problems. For the middleware, Apache Tomcat is used as it is able to deploy Vue-based applications very well whilst also being lightweight and able to provide dynamic content by employing Java-based logic.

### Extensibility

Extensibility is a measure of how simple it is to extend/add extra functionality into the system. To ensure this, each major section of the program will be split up into different modules so that any changes made to a certain module will not affect the structure of a different module. This will allow for new modules to be added later on with minimal difficulties.

### Portability and Platform Independence

Portability: A software is portable when there are no installations needed. The UC Client is being created with portability in mind. This means that the software will most likely be accessed through a web browser, accessible through a tablet, laptop or computer without needing any software installed.

Platform Independent: A software is platform independent when it is not limited to a single platform. The student client is being made to be platform independent. Through the use of Electron.js, the software can be bundled into any native binary which means that if the client would like to expand to having the application working on Windows or MacOS, it will be simple to do so.

## The Architectural Model

The architectural model is described using a component interaction diagram.

### Architectural Concepts and Representations

The component interaction diagram details the architecture of several software components.

* Labelled boxes represent different software components. These boxes encapsulate groups of other components.
* Software components may be separate pieces of software, important interfaces within a piece of software or processes managed by software components.
* Labelled boxes with dashed lines represent hardware that the components run on.
* Arrowed lines describe interfaces where software components interact with each other
* Stick figures represent users
* Arrows between stick figures and software component represent software interfaces where users can interact with the software component.

## Software Architectural Overview

### Component Interaction Diagram

### Component Interaction Diagram Description

**Users**

* **Student**: Represents a University student sitting an assessment
* **UC**: represents a Unit Coordinator
* **Invigilator**: represents an assessment Invigilator who helps in the running of assessment

**Hardware**

* **Device with web browser**: any device with access to a modern web browser
* **Curtin super lab computer**: any computer in the new Curtin super lab
* **Backend Server**: The server used to host the backend architecture

**Software Components**

* **Vue.js:** A frontend javascript framework
* **Notification Handler**: An operating systems notification handler
* **UC Client Assets:** Static assets(js/html/css files) generated by the build of a vue.js UC client application.
* **Backend Application:** The backend spring boot application
* **Controller Layer**: a software layer of the **Backend Application** that provides API’s
* **Service Layer**: a software layer of the **Backend Application** where b business logic is conducted.
* **DAO Layer**: a software layer of the **DAO Application** where data persistence is managed.
* **KeyCloak**: a software layer providing Identity and Access Management(IAM)
* **PostgreSQL database**: A relational database
* **File System**: The file system layer of an Operating System, provides access to a disk.
* **Email Server**: a server used for the sending of emails
  + 1. **Dependency Matrix**

|  | **UC Client** | **Student Client** | **Database** | **Backend** |
| --- | --- | --- | --- | --- |
| **UC Client** |  |  |  |  |
| **Student Client** |  |  |  |  |
| **Database** |  |  |  |  |
| **Backend** |  |  |  |  |

# Architectural Specification - UC Client Framework Contributors: Jia Son Pow

## Introduction

The UC Client framework provides a simple and interactive interface that allows new users to easily become familiar with the application. The Vue-based framework will be used mainly to make API calls to the backend server to interact with the data model. This framework will be utilising a Vue-js user interface

## Design Methodology

### Vue MVC Architecture

The design of the UC client follows an MVC architecture with three sections:

* **View:** The user interface interacted to by the UC/invigilator, created through the use of Vue.js
* **Model:** This is the UC Client Assets module which will sync the database with the updated files from the computer’s file system.
* **Controller:** Vue.js will call API’s to pass the test materials from the computer’s file system to the database.

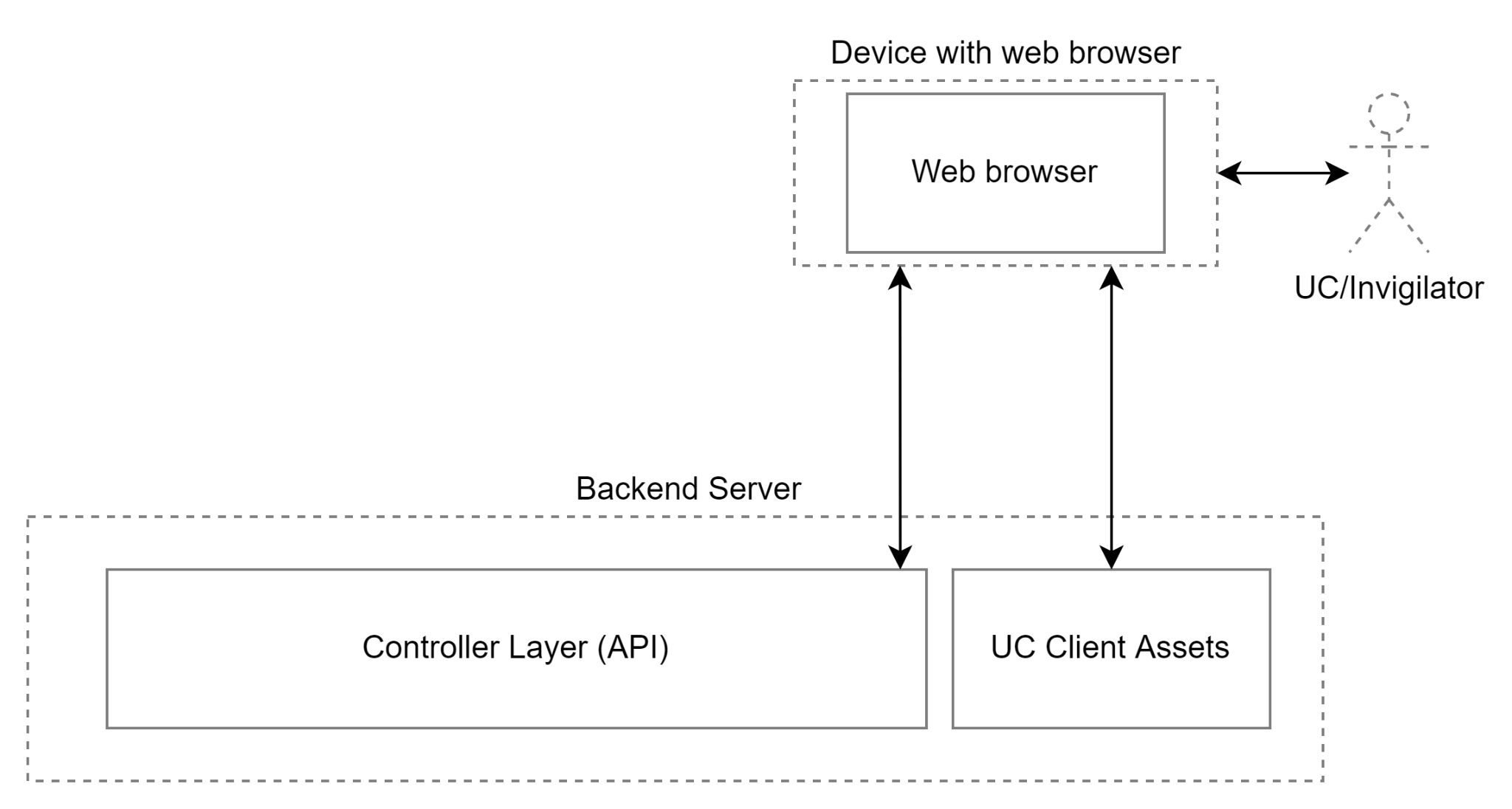
The view layer will have an option for uploading/updating files which UC can do at any point. This will call the controller layer which will notify the UC Client Assets module that there are updates wanting to be made to the database. The UC Client Assets module will then create a copy of the files from the computer’s file system onto the server.

## System Structural Overview

### Software Component View

The below diagram represents the architecture of the UC client framework.

* The web browser will contain the Vue.js modules.
* Vue.js will control the communication between the backend server and the web application.
* Vue.js will update the backend server with any changes made to an assessment periodically.
* Vue.js will communicate with the UC Client Assets in the backend to handle authentication/sign-in as well as the API’s found in the backend.



### Component Details

| **Component Type** | UC Client Framework |
| --- | --- |
| **Purpose** | Provides a way for the application to interact with the computer’s resources (file system, push notifications etc.) as well as giving a way to connect the Web browser with the Controller API module which will facilitate the downloading and uploading of files. It will compile the application into its executable file. |
| **Functionality** | * Call Controller API’s * Provide system API’s * Provide access to computer’s file system * Provides ability to run application as a background app |
| **Externally resources required** | * Spring Boot backend application   + RSync module   + Keycloak module |
| **Dependencies and Interrelationships with other major components** | * The UC Client manages Rsync instances that connect to Rsync instances that are managed by the backend application * The Student Client and the UC client both use API’s provided by the Backend Application * Relies on the front-end Vue.js module |
| **Development Verification/Validation/**  **Analysis/Test strategies** | * Unit testing, low-level testing of methods and classes * Integration testing, testing that different classes work together. * Functional testings, testing that system against requirements |

### List of Product Deliverables

This will be delivered as a web browser application package containing RSync and Vue.js that will work on various devices and across different operating systems.

## Theory of Operation

### Installation

The application will be installed as a package which is compiled by the Vue.js module. This application will have to be installed with permissions given from the admin for the application to access and save files to the computer’s file system, lock files from the student and run as a background process. It will be installed on the UC/invigilators’ computers as well as their mobile devices if desired.

### Background processes

The application will need to perform the below actions whether its open or closed:

* Must be able to retrieve assessment data and documents at any time.
* Must check for notifications and messages that might have come through from both the student and UC clients.
* Needs to still be running a countdown timer if there is an ongoing assessment.

### Vue.js

For the front-end user interface, Vue.js will be used to make up the view section of the MVC architecture. It will be how the end user (in this case, the UCs/invigilators) interacts with the system.

### Global notifications

To send out global notification pop-ups to the student clients, we will be using VueUp that provides this functionality in a simple and lightweight manner. The Vue module will push the notifications to the controller layer in the backend server where it will then send push notifications to the students, using HTML5 Notification API which utilises the currently running operating system’s native notification API’s to display the message.

### Error Handling

* + - 1. **Vue.js errors**

Errors in Vue.js will be managed by utilising either the global error handler or the component local error handler built into Vue.js

# Architectural Specification - Student Client Framework Contributors: Filipe Lagrenade

## Introduction

The Student Client framework will use Electron to communicate with the backend application. It will receive API calls from the Rsync module to pass updated files to the repository as well as receiving assessment files from the repository. It also will compile the application into a native binary for usage on the linux operating system that is defined by the client. This framework will be packaging a Vue.js user interface following an Model-View-Controller (MVC) Architecture.

Electron also provides the necessary access to the computer to allow for file system access, the ability for push notifications and running the application as a background process, which a normal Vue.js application is unable to do.

## Design Methodology

### Vue MVC Architecture

The design of the user client follows an MVC architecture with three sections:

* **View:** The user interface interacted to by the student, created through the use of Vue.js
* **Model:** This is the RSync module which will sync the database with the updated files from the computer’s file system.
* **Controller:** Electron will call API’s to pass the updated test materials from the computer’s file system to the database.

The view layer will have an option for manual saving of files which students can do at any point during the assessment. This will call the controller layer which will notify the RSync module that there are updates wanting to be made to the database. The RSync module will then, with the permissions granted by Electron, create a copy of the files from the computer’s file system onto the server. This also happens in reverse. When the student starts the assessment, the view layer will call the controller layer to get the necessary files from the server. This will call the RSync module to push files to the computer’s file system.

## System Structural Overview

### Software Component View

The below diagram represents the architecture of the student client framework.

* The electron package will contain the RSync and Vue.js modules
* Electron will control the communication between the file system/notification handler and the application
* RSync will update the backend server with any changes made to an assessment periodically
* Electron will communicate with Keycloak in the backend to handle authentication/sign-in as well as the API’s found in the backend

### 

### Component Details

| **Component Type** | Client Framework |
| --- | --- |
| **Purpose** | Provides a way for the application to interact with the computer’s resources (file system, push notifications etc.) as well as giving a way to connect the front end module with the RSync module which will facilitate the downloading and uploading of files. It will compile the application into its executable file. |
| **Functionality** | * Call RSync API’s * Provide system API’s * Provide access to computer’s file system * Provides ability to run application as a background app |
| **Externally resources required** | * Spring Boot backend application   + RSync module   + Keycloak module |
| **Dependencies and Interrelationships with other major components** | * The Student Client manages rsync instances that connect to rsync instances that are managed by the backend application * The Student Client and the UC client both use API’s provided by the Backend Application * Relies on the front-end Vue.js module |
| **Development Verification/Validation/**  **Analysis/Test strategies** | * Unit testing, low-level testing of methods and classes * Integration testing, testing that different classes work together. * Functional testings, testing that system against requirements |

### List of Product Deliverables

This will be delivered in a full package bundled by Electron.js containing RSync and Vue.js.

## Theory of Operation

### Installation

The application will be installed as a package which is compiled by the electron.js module. This application will have to be installed with permissions given from the admin for the application to access and save files to the computer’s file system, lock files from the student and run as a background process. It will be installed on the laboratory computers.

### Background Process

The application will need to perform the below actions whether its open or closed:

* Must be able to retrieve assessment data and documents before an assessment.
* Must check for notifications and messages that might have come through.
* Needs to still be running a countdown and file syncing services.

### RSync

The RSync module will be managed by a controller as part of the MVC model mentioned above. Electron will call RSync through API’s to both push data to the database (the updated student tests) and pull data from the database into the student computers (The assessment information).

An RSync process will be created every time a test is started and will run throughout the duration of the test, making periodic updates from the student’s file system (with permissions given from Electron.js) to the database.

### Vue.js

Vue.js will be used in the creation of the front-end interface and makes up the view section of the MVC architecture. It will be how the end user (in this case, the students) interacts with the system.

### Notifications

To send push notifications to the user, Electron.js uses HTML5 Notification API which utilises the currently running operating system’s native notification API’s to display the message.

It is easily added in by using electrons notification module which will show notifications in the main process.

### Error Handling

* + - 1. **RSync Errors**
* If RSync fails it will automatically attempt reconnection
* If it is unable to establish a reconnection the backend service will supply the user client with an API which can be used to save the data
  + - 1. **Electron Errors**
* Any electron errors will be handled by the electron-unhandled package. This package will produce custom error dialog boxes.
  + - 1. **Vue.js Errors**
* Errors in Vue.js will be managed by utilising either the global error handler or the component local error handler built into Vue.js

# Architectural Specification - The Backend Application Contributors: Isaac Ellis

## Introduction

The backend application provides a set of api for the student client and UC client to interact with the data model. The backend interacts with a database, the file system, rsync, and an email server. It will be primarily built using SpringBoot, an Open Source Java application framework.

## Design Methodology

The design of the backend application makes use of a three layer architecture consisting of:

* **Controller Layer**: where api handlers ares defined
* **Service Layer**: where business logic is performed. Such as interacting with the filesystem, sending emails, managing rsync connections and managing users.
* **DAO**: the data access layer interacts with the database via an object relational mapping(ORM) library.

The Controller layer handles api requests by calling functions from the service layer. The service layer functions by interacting with the dao layer, other services and performs business logic on data. The DAO layer provides functions that allow interaction with databases, such as reading, searching, deleting, updating and creating.

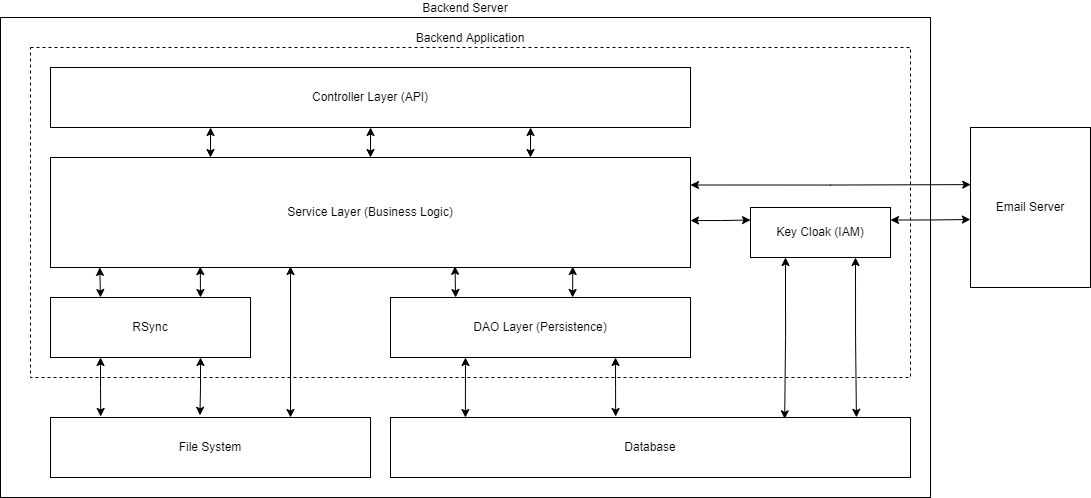
User management is handled by embedding a KeyCloak server inside the backend application. This embedded KeyCloak is configurable whilst providing java libraries and an admin interface.

## 

## System Structural Overview

### Software Component View

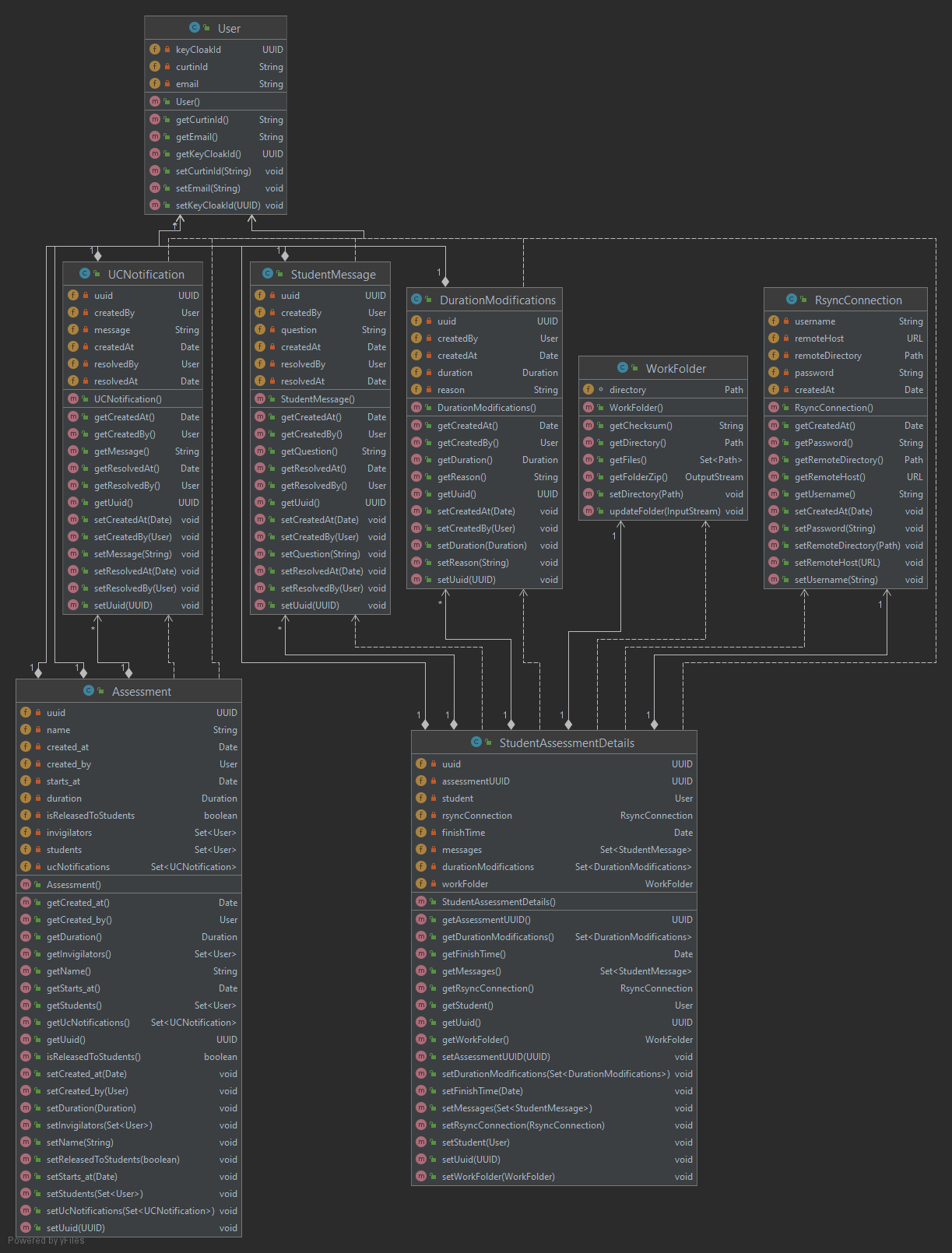
The diagram below represents the architecture of the backend application.



* + - The controller/service/DAO layers consist of java beans.
    - KeyCloak is embedded within the application and is accessed by the libraries that it provides
    - Rsync processes are managed by a bean in the services layer

### Data Model

The following UML diagram represents the java data model used by the Controller, Service and DAO layers. The model roughly maps to the entity diagram allowing for an object relational mapping(ORM) library to be used.



### Component Details

| **Component Type** | Backend Application |
| --- | --- |
| **Purpose** | Provides a set of api for the student client and UC client to interact with the data model. |
| **Functionality** | * Provide REST APIs * Authentication and user management * Sync assessment work folders with Rsync * Send emails via an external email server * Manage data in a database |
| **Externally resources required** | * Email Server * PostgreSQL Database |
| **Dependencies and Interrelationships with other major components** | * The Student Client manages rsync instances that connect to rsync instances that are managed by the backend application * The Student Client and the UC client both use apis provided by the Backend Application |
| **Development Verification/Validation/**  **Analysis/Test strategies** | * Unit testing, low-level testing of methods and classes * Integration testing, testing that different classes work together. * Functional testings, testing that system against requirements |

### 

### List of External Interfaces

* + - 1. **Rest APIs via Controller Layer**

| **UCAssessmentController (Api prefix= “/api/1/uc/assessment/”)** | | |
| --- | --- | --- |
| **METHOD** | **API PATH** | **Functionality** |
| POST | /create | Creates an assessment |
| UPDATE | /update/{assessmentId} | Updates an assessment |
| DELETE | /delete/{assessmentIid} | Deletes an assessment |
| POST | /time/{assessmentIid} | Adds time to all student sitting an assessment |
| POST | /time/{assessmentIid}/{studentId} | Adds time to a student sitting an assessment |

| **AssessmentController (Api prefix= “/api/1/assessment/”)** | | |
| --- | --- | --- |
| **METHOD** | **API PATH** | **Functionality** |
| GET | /read/{assessmentIId} | Returns an assessment object |
| GET | /read/{assessmentIId}/{studentId} | Returns an assessment details object |
| GET | /document/{assessmentId} | Returns all of an assessment tests documents. Such as pdfs or word documents as zip file. These files are test papers and resource required to sit the test |
| GET | /document/{assessmentId}/{filename} | Returns a document from relate to an assessment such as a test paper |
| POST | /rsync/{assessmentId} | Initiates an rsync connection on the backend by informing the backend what rsync service to connect to. |
| POST | /{assessmentId}/submit | Lets a student end their assessment |

| **WorkFolder Controller (Api prefix= “/api/1/work”)** | | |
| --- | --- | --- |
| **METHOD** | **API PATH** | **Functionality** |
| GET | /list/student/{studentId} | Returns a list of work folder objects related to a student |
| GET | /list/assessment/{assessmentId} | Returns a list of work folder objects related to an assessment |
| GET | /download/{assessmentId} | Returns a zip of all the files related to a work folder object |
| GET | /download/{assessmentId}/all | Returns a zip of all the work folder related to an assessment |
| POST | /save/{assessmentId} | Upload a a work folder and saves it |

| **MessageController (Api prefix= “/api/1/message”)** | | |
| --- | --- | --- |
| **METHOD** | **API PATH** | **Functionality** |
| GET | /notification/{assessmentId}/list | Lists notifications sent by UC’s |
| POST | /notification/{assessmentId}/send | Send a notification to all student (used by UC’s) |
| GET | /message/{assessmentId}/list | List messages sent by all students during an assessment |
| GET | /message/{assementId}/{studentId}/list | List messages sent by a student during an assessment |
| POST | /message/{assessmentIid}/send | Send a message to the UC (used by students) |
| UPDATE | /message/{messageId}/resolve | Marks a students message as resolved |

| **AuthController (Api prefix= “/api/1/auth”)** | | |
| --- | --- | --- |
| **METHOD** | **API PATH** | **Functionality** |
| POST | /login | Authenticates the user |
| POST | /password/reset | Resets a users password |
| POST | /logout | Logs a user out |
| POST | /create/UC | Creates UC account |
| POST | /verify/account | Verifies an account during account creation |

### List of Product Deliverables

* The backend application will be built as a .jar file

## Theory of Operation

### Installation

The linux installation involves the creation of a linux service that uses the .jar file.

1. Create system link with “sudo ln -s /path/to/backendapplication.jar /etc/init.d/AssementBackend
2. Start the service “sudo service AssementBackend start”

### Identity and Access Management

KeyCloak is and Identity and Access Management system that provides our application with

* User management
* Role and privilege management
* Password reset token management
* User creation, including verification token management
* Secure password storage
* Tracking of login attempts and lockout from too many attempts

### Rsync Management

Rsync is used by the backend service to sync a student's work folder from the rsync client managed by the student client.

* The students clients rsync process is connected to by an rsync process managed by the backend service
* The backend manages an rsync process for each student client rsync process
* An rsync process only runs while a student is sitting an assessment
* Rsync makes versions of a students work folder each 15 minutes

### Error Handling

* + - 1. **API**
* Error created when processing api requests will return an error message with appropriate HTTP status codes.
* Error will be logged throughout the application.
  + - 1. **Rsync**

1. Rsync connections that fail will attempt reconnection.
2. The rsync connection can also be updated if the student changes computers during the assessment.
3. If rsync is not working the backend service provides an api for a students work folder to be saved

# 6. Architectural Specification - Database Contributors: Sanjay Williams, Isaac Ellis

## Introduction

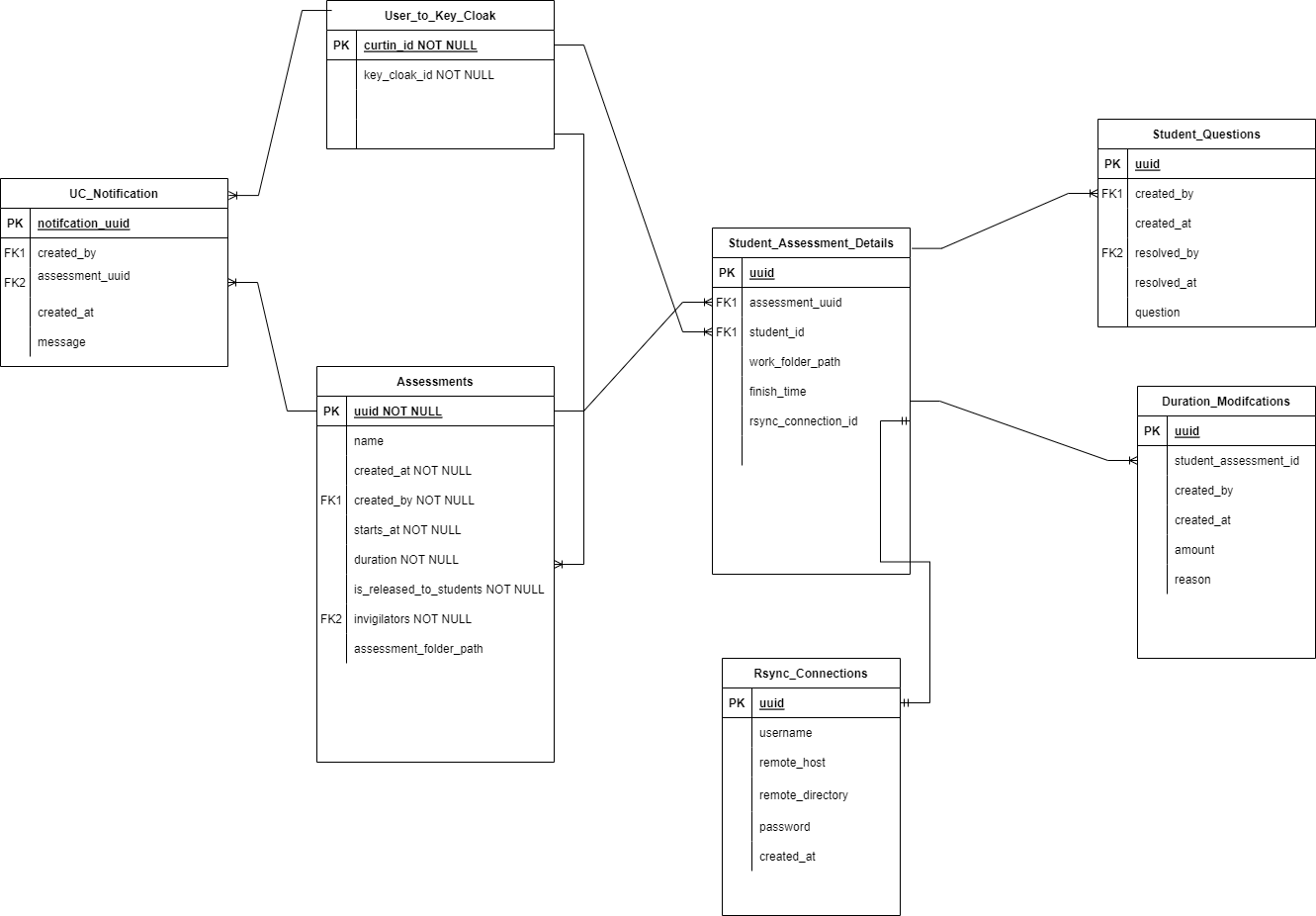
The database will be the storage location for multiple data types created and/or used by the application. This will include all appropriate student files, logs of save times, end times, student questions, etc., and any other assessment details including the list of students taking the assessment, the list of staff that are permitted access, and the list of apps and sites not allowed in the assessment. PostgreSQL will provide the interface to query, add, remove, or modify the data stored in the database. These options can be accessed through the pgAdmin GUI or the psql command-line interface. pgAdmin is a utility that comes with the Postgres installation and allows the admin to do regular database-related tasks in a GUI.

## System Structural Overview

The PostgreSQL database will be a standalone component residing within the server-side of the application. Postgres provides third-party modules such as ‘uuid-ossp’ to generate our data types in its own form of UUIDs and uses its core capabilities to store and compare the UUID values.

### Software Component View

The data will be stored in the following structure. See section 5.3.1. for the architecture of the relationship between the Back End and the Database.



### Component Details

| **Component Type** | Database |
| --- | --- |
| **Purpose** | Storage space for all relevant data during and after use of the application.   * Queried by the backend for a particular piece of data * Added to, removed from, and modified by the backend. |
| **Functionality** | * Capable of handling user-defined types. * Features a sophisticated locking mechanism. * Can handle nested transactions. * Asynchronous replication. |
| **Externally resources required** | * Backend Application |
| **Dependencies and Interrelationships with other major components** | * The Postgres database will interact directly with the backend application. |

### 

## Theory of Operation

### Migration and Initiation

The database will be initialised and migrated using Flyway.

Flyway is a database tool that provides version based database migrations. This allows database schemas and procedures to be stored in code and applied to any postgresql instance.

### SQL Commands

* + - 1. Querying Entries: ‘SELECT’
* **Input**: table ID, column ID, and condition (row ID)
* **Output**: None, specified return value, or NULL if not found.
  + - 1. Adding Entries: ‘INSERT’
* **Input**: table ID, column ID(s), and data
* **Output**: None, specified return value, or NULL if not found.
  + - 1. Removing Entries: ‘DELETE’
  + **Input**: table ID and condition (row ID)
  + **Output**: None, specified return value, or NULL if not found.
    - 1. Modifying Entries: ‘UPDATE’
         * **Input**: table ID, column ID, condition (row ID), and new value
         * **Output**: None, specified return value, or NULL if not found.

# 

# Security and Ethical Implications Summary Contributors: Jia Son Pow

The system will not store any passwords in plaintext. Instead, we will be using JSON Web Token (JWT) as the choice of user authentication method. There are different ways that it can be used:

**Authentication:** When a user is able to login successfully to the client with their credentials, an ID token which contains the user’s cached profile information will be returned.

**Information Exchange:** The information contained within a JSON object can be verified and trusted. This is because the JWTs are commonly signed using either a secret hashed algorithm (HMAC) or a public/private key pair (RSA/DSA). This ensures data integrity as when a token is signed with a key pair, it signifies that the party with the private key is the one who signed it. Any alteration to the content will break the digital signature and the server will reject the token.

To prevent any possible SQL injection attacks, a number of prevention measures will be put into practice. They are:

**Query Parameterisation:** parameterized statements use stored queries that have markers, known as parameters, to represent the input data. Instead of parsing the query and the data as a single string, the database reads only the stored query as query language, allowing user inputs to be sent as a list of parameters that the database can treat solely as data.

**Database auditing:** regular auditing and vacuuming will help maintain the PostgreSQL environment. Extensions like [pg\_audit](https://www.pgaudit.org/) provide a deeper level of detailed session and object logging than the standard logging found in PostgreSQL. This level of detail can help pinpoint any unusual or irregular queries; for example, queries to system tables. System tables, like those found under the information\_schema, are not regularly accessed by users and should be treated with suspicion.

As for the ethical implications that will be enforced in this project, information of any form that is stored/provided to the system will be classified highly confidential and cannot be easily accessed unless it’s from a user of high hierarchy like a UC/invigilator or admin. The information will, under no circumstances, ever be used for targeted advertising for commercial purposes.

# Debugging, Error Logging, and Performance Monitoring Contributors: Sanjay Williams

## Performance Monitoring

Performance monitoring will analyse the system’s various areas and how well they are running. When performance monitoring is enabled, it will assess if the system is running as expected and log the necessary data if not. Areas that will be able to be assessed will be the front-end, server, and database and will include data such as the network speed, number of requests, the run time, response time, and CPU usage.

### Performance Counters

The library will provide performance counter functions that, when enabled, will allow the person monitoring the system to analyse the timing of the execution of functions, including both the timing within and between functions.

As performance monitoring will likely affect performance itself, the person debugging will be able to specify which areas of the system are being monitored and limit the output to those areas to minimise the impact on performance.

Similarly, performance monitoring should be disabled during normal use.

## Debug

Debugging will be provided by the library. Developer tools will be built into the application and will be used alongside assertions to ensure control over and analysis of the system. The application will have builds that will allow developers to output certain data for debugging and performance checking purposes. They will also allow the developer to assert if certain parameters are correct, halt the system, and halt execution.

### Debug Output

When the debug build is in use, the person debugging will be able to output specific data from the parts of the system that are being assessed. The output that isn’t being assessed will not be output to avoid impacting performance.

Similarly, during normal use, debugging will be disabled.

### Assertions

While using the debug build, assertions can be enabled to halt the system or abort the execution that follows the assertion statement while also giving feedback to the person debugging. Assertions should be used to assert any parameters that aren’t already provided by the runtime debugging.

While using the non-debug builds, assertions will have no effect.

## Error Logging

Error Logging will also be available in the debug builds when enabled. The logs will be categorised as either informational, warning, or error. These will be directed to the system log file.

### Error Handling

* + - 1. **Electron**

Electron can be made to log errors using third-party logging modules. An example of this is ‘electron-log’ which can log in the following log levels: error, warn, info, verbose, debug, silly.

* + - 1. **Apache Tomcat**

Tomcat has its own implementation of several key elements of the ‘java.util.logging’ API called JULI, the key component of which is a custom LogManager implementation. The Apache Tomcat startup scripts will enable the logging implementation if provided with certain system properties when starting Java.

# 

# 

# 