Software Design Specification

for

*FlexAdvisor*

Version 1.0

Prepared by Max Garces

04/22/2020

Table of Contents

1. Introduction 1

1.1 Purpose 1

1.2 Intended Audience and Reading Suggestions 1

1.3 Product Scope 1

1.4 Overview 1

2. Design Considerations 2

2.1 Assumptions and Dependencies 2

2.2 General Constraints 2

2.3 Development Methods 2

3. System Architecture 2

3.1 High level Overview of API Micro Architecture 2

4 Component Definition 4

4.1 Login Component 4

4.2 New User 5

4.3 Dashboard 6

4.4 Program 7

4.5 User List 8

4.6 Messaging 9

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Max Garces | 04/22/2020 | Initial SDS version | 1.0.0 |
| Juan Cruz | 04/24/2020 | Cleanup, grammar, headings | 1.0.1 |
|  |  |  |  |

# 1. Introduction

## 1.1 Purpose

This is the Software Design Specification for FlexAdvisor. This document will define the design of the web application, by breaking down the project into components to describe in detail what the purpose of each component is and how it will be implemented. It will contain specific information about the expected input, output, and functionality. It will explain system constraints, assumptions, dependencies and interface.

## 1.2 Intended Audience and Reading Suggestions

This Software Design Specification is intended for:

* Development team who will develop the first version of the system.
* Developers who can review project’s capabilities and more easily understand where their efforts should be targeted to improve or add more features to it.
* Product testers can use this document as a base for their testing strategy. It might be used as a tool for verification and validation of the final product.

## 1.3 Product Scope

FlexAdvior will be a native react-based web application, built on the MERN stack using the microservices approach. It will allow a school’s system admin to register for an account – Once the user is created by the FlexAdvisor client service team, and email is sent with login details to the admin. Faculty accounts can be created through the system admin portal and student accounts can be created through the new user log in page. Once a new student is created, they are paired to an advisor and an IEP is created. The Current Scope of the application is limited to user creation, and IEP creation. A more robust messaging system, schedule and payments functionalities along with viewable profiles are all planned for future releases.

## 1.4 Overview

The remainder of this document includes 4 chapters.

* The second one introduces system constraints and assumptions about the product.
* The third chapter gives high level overview of system architecture.
* The fourth chapter provides the description of the system interface.
* The final fifth chapter provides detailed system design including main data structures, attributes, and methods used in each class.

# 2. Design Considerations

## 2.1 Assumptions and Dependencies

The client will be hosting the application on a cloud base solution. The recommended Heroku cloud has capabilities for importing all necessary JavaScript libraries

## 2.2 General Constraints

**Implementation environment**

Project will be written using the MERN stack (MongoDB, Express, React and Node.js).

**Hardware**

None.

## 2.3 Development Methods

For this project a Microservices approach will be taken. This is a software development technique —a variant of the service-oriented architecture (SOA) structural style— that arranges an application as a collection of loosely coupled services. In a microservices architecture, services are fine-grained, and the protocols are lightweight.

# 3. System Architecture

## 3.1 High level Overview of API Micro Architecture

The web application will use 4 core APIs as the base for the microservice architecture. These API endpoints are: RegisteredInstitutions, Users, Programs, School Details. REST API’s can be circumstanced by allowing the API Gateway to detect what input parameters are sent together with the POST/GET request and can also be specified through the calling method. This allows for a streamlined data model that is easy for FlexAdvisor Developers to manage and maintain. Since MongoDB does not store blank variables, if the POST request does not specify, we should end up seeing a natural uniformity of which variable align to which roles.

**RegisteredInsititutions** - Primarily for FlexAdvisor staff use to create SystemAdmin accounts based off whether an email already exist for a school.  
 **RegisteredIntitutions**{

InstitutionName:String,

AcademicID:String,

Users[]

}

**Users** - Will save all user information on the cloud platform – most methods will branch from this main API.  
 **Users**{

Username:String,

FullName:String,

Email:String,

Password:String, hashed using bcrypt open source library

InstitutionID:String,

Role:String,Protected access, only a FlexAdvisor Employee can create a SysAdmin, etc.

Program:String,

AssignedDirector:String,

AssignedAdvisor:String,

AssignedStudent[],

ClassPerSemester:String,

IEP[],

IEPApproved:Boolean

classPerSemester:Int,

classesTaken:Int

messageThread[]

}

**Programs** - Allows for directors to directly reference and update this collection, as well has new user to access for IEP creation without needing to open director credentials.  
 **Programs**{

ProgramDirector:String,

ProgramName:String,

AcademicID:String,

ClassList[]

}

**SchoolDetails** - Primarily for System administrator to monitor the current registered institution which they are system owners of.

**SchoolDetails**{

AcademicID:String,

SystemAdminEmail:String,

SchoolAddress:String,

}

# 4 Component Definition

## 4.1 Login Component

This class is responsible for:

* Displaying the Login Screen
* Checking for invalid input
* Calling the New User component
* Calling the create user module with the appropriate parameters
* Logs in a user if they are present in the database collection

**Attributes**email, password

**Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| Login(email, password) | Logs in a user if the user email is present in the data collection |
| newUserDisplay(user) | Component for the user to fill out the new use form |

**Uses/Interactions**

Logs users and validates user information provided.

**Constraints**

* Before the login method can be called, the user inputs must be validated
* Any user methods are POSTS, which encrypt the message before sending.
* User data constraints:
  + Each user must have a unique email.
  + Each user must have an academic ID

**Resources**

The only resource that is used by this class the native react package.

**Processing**

**validateEmail (): Boolean**

1. Checks user email input and validates that it is an email
2. Call the GET🡪 user and compares to current registered users list makes sure the email isn’t already present in the data collection

**validatePassword():**

1. Checks that it is a valid password
   1. Password restraints can be set by the business use case.
   2. Call the crypt hashcode(password) method to encrypt password when storing user.

**createUser(user):**

1. Creates an entry in the data collection using the current user object that is present in the session.
2. Returns a validation mail notification once the user is created.
3. Send an email to the user session that created the account.

## 4.2 New User

This class is responsible for:

* Displaying the new user screen
* Calls the “Register your institution method”
* If a new user is created, it refreshes the UI to capture additional details from the student (which program they are enrolled in, how many classes per semester)
* Calls assignAdvisor to dynamically assign advisor upon account creation.
* Calls the createIEP method to dynamically create and send IEP to the assigned advisor.

**Class Attributes**

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| user | The entire user object with all the data elements are accessed and edited through this class. (AssignedAdvisor for student, AssignStudent[] for advisor) |
|  |  |
|  |  |
|  |  |

**Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| createUser(user) | Passes the new user object to the data collection |
| sendConfirmationEmail(email,password) | Once a user is created and inserted into the data collection a confirmation with the username and password is sent to the email address that created the account. |
| createIEP(user) | Grabs the required preferences from the student user profile and seeks through the programs data collection to the grab the ClassList. |
| registerInstitutionScreen() | Refreshes screen with the register an institution module |

**Uses/Interactions**

Creates a new user instance in the users data collection

**Constraints**

Preconditions – all necessary values are passed to the create user.

## 4.3 Dashboard

This class is responsible for storing and returning (when it is requested) for the user information.

**Attributes**

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| user | Passes the new user object to the data collection |
|  |  |
|  |  |
|  |  |
|  |  |

**Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| populateDashboard(user) | The user data collection has all the fields necessary to display the dashboard |

**Uses/Interactions**

Once user logs in successfully, the dashboard component is the visible display shown.

**Constraints**

User must be logged in with one of the 4 account types to display proper elements in the dashboard.

**Processing**

This class is responsible for:

* Displaying the user, the permission is needed for the type of user using the “role” field as the logic piece and the rest of the user items are available directly in the collection

## 4.4 Program

This class is responsible for storing and returning (when it is requested) program information. Only a user with the according program director role can edit the program list.

**Attributes**

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| user | Passes the new user object to the data collection |
| classList[] | Saves the classes in index order into an array within the data collection |
|  |  |
|  |  |
|  |  |

**Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| editProgram(prorgam) | A program director can access the program data collection if the Class definition matches the title for which the program director is the owner of. |
|  |  |

**Uses/Interactions**

A program director will click on “program” and define the order in which classes should be taken in order to graduate from the program.

**Constraints**

Only a program director can write to this field.

**Processing**

**displayDashboard(this.user.role): String**

1.Using the role parameter, will show the sidebar options for which user list a user has access to.

**grabbingUser(this.user.role,assignedStudents,assignedadvisors): String**

1.This will set the array fields for the assigned students, and then reference the students from the data collection to populate the information into a react data grid.

## 4.5 User List

This class is responsible for storing and returning (when it is requested) for the user information.

**Attributes**

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| user | Passes the new user object to the data collection |
|  |  |
|  |  |
|  |  |
|  |  |

**Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| displayUserList(user, RequestedList) | The user data collection has all the fields necessary to display the dashboard. It will send the requested list from the current user to verify they have permission to view the  list. |
|  |  |

**Uses/Interactions**

A user will click on the program or IEP feature if they are a student list that they are trying to access. From a UI standpoint, depending on the logged in user’s roles, the proper display will show on the sidebar. The same “ClassList” for a program within an institution is circumstanced based off the variable of the user when populated for an IEP if they have one available.

**Constraints**

All the user fields will be available for validation through the did.

**Processing**

**displayDashboard(this.user.role): String**

1.Using the role parameter, will show the sidebar options for which user list a user has access to.

**getProgram(this.usere.program): String**

1.This will set the array fields for the assigned students, and then reference the program from the data collection to populate the information into a react data grid.

## 4.6 Messaging

In this release, messaging will be a simple string array that can me only accessed once it is verified that the student account is assign to the advisors, and that the student is also assign to the corresponding advisor is their data collection.

**Attributes**

|  |  |
| --- | --- |
| **Attribute** | **Description** |
| user | Passes the new user object to the data collection (two user objects will be compared in this instance) |
|  |  |
|  |  |
|  |  |
|  |  |

**Methods**

|  |  |
| --- | --- |
| **Method** | **Description** |
| displayUserList(user, RequestedList) | The user data collection has all the fields necessary to display the dashboard. It will send the requested list from the current user to verify they have permission to view the  list. |
|  |  |

**Uses/Interactions**

The first message string is initiate through the createUser activity if a student advisor is created.

**Constraints**The user and the advisor’s users must both be present in each other’s user field data collection.

**Processing**

**checkAssignment(this.user.role, this.user.AssignedAdvisor, target.user.AssignedStudent): String**

1.Validates that assigned user, and assigned student correspond to each other’s data collection. This will give access to modify the Message[] string array within the student user data collection.

**messagingViewer(this.user.role)**

1.This will allow for editing and access to the message string. A save button will allow to save into the string and provide a notification if the array it displaying a new index.

2.From here, advisors can director update the IEP Approved Boolean value using the UI buttons to allow the student to view their created IEP.