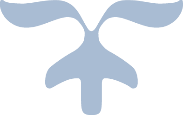


**College for me**

Implement Report



**SWYM Team**

Siyong Liu

Maciej Wlodek

WeiXiang Mei

Yousef Khan

## **Implementation Status Report**

### Phase1

Siyong Liu

**Task**

1. FrontEnd Structure
2. Frontend profile Page
3. Backend Structure
4. Backend credential Part ( Log and JWT credential)

**Implement Details**

1. Design of FrontEnd Structure (only show folders under src folder)

* App
* Index
* Config
  + I18
  + Index ( environment )
* Action
  + ActionType ( action constant )
  + Index (export all actions )
  + ProfileAction
  + AdminAction
  + CollegeAction
  + HighSchoolAction
  + StatusAction
  + UserAction
* Components
  + AdminScreenFolder
  + HomeScreenFolder
  + CredentialScreenFolder
  + LoginProfileScreenFolder
  + RegisterProfileScreenFolder
  + SearchScreenFolder
  + CollegeScreenFolder
  + LoadingPage
  + NavBar
* CSS
* Reducer
  + AdminPageareducerFolder
  + ProfileReducer
  + StatusReducer
  + ApplicationReducer
  + CollegeReucer
  + HighSchoolReducer
  + Index (combine reducers)
* Utils
  + ValidationCheck
  + HandlerTemplate

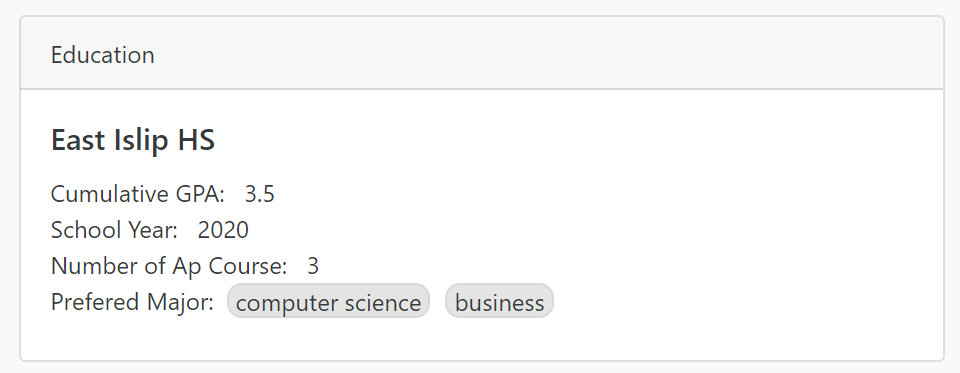
1. Design of Backend Structure (only show folders under src folder)

* Com.c4me.server
  + Config
    - AdviceFolder
    - CustomAnnotationFolder
    - AspectFolder
    - ConstantFolder
    - ExceptionFolder
    - FilterFolder
    - HandlerFolder
    - InterceptorFolder
    - JpaConfig
    - SecurityConfig
  + Core
    - User ( this is the sample of all folder under core )
      * Controller
      * Repository
      * Domain
      * Service
    - Profile
    - College
    - HighSchool
    - Log
  + Entities
  + Utils

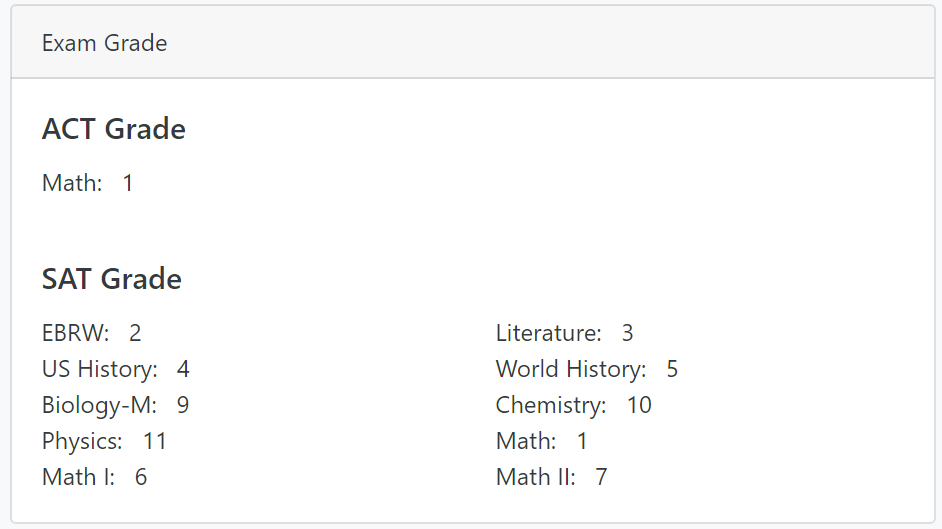
1. FrontEnd Profile Page implement detail (Profile Page)
2. Structure

* Saga
  + profile.js
  + user.js
  + application.js
* Reducer
  + applicationReducer.js
  + profileReducer.js
  + statusReducer.js
* Components
* Card
  + EducationCard.js
  + ApplicationCard.js
  + SATCard.js
  + ACTCard.js
* EditingCard
  + AddingApplicationCard.js
  + EditingEducationCard.js
  + EditingApplicationCard.js
  + EditingSATCard.js
  + EditingACTCard.js
* InfoCompleteCard.js
* UserInfoCard.js
* Index.js

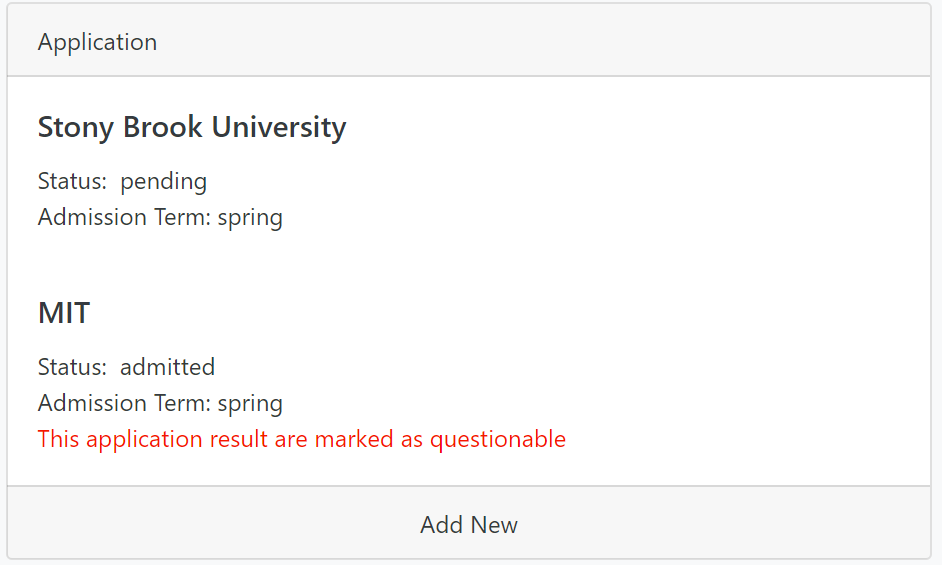
1. Product show
   1. EducationCard



* 1. ExamGradeCard



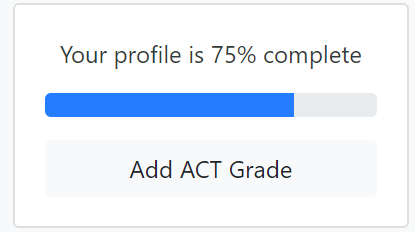
* 1. ApplicationCard

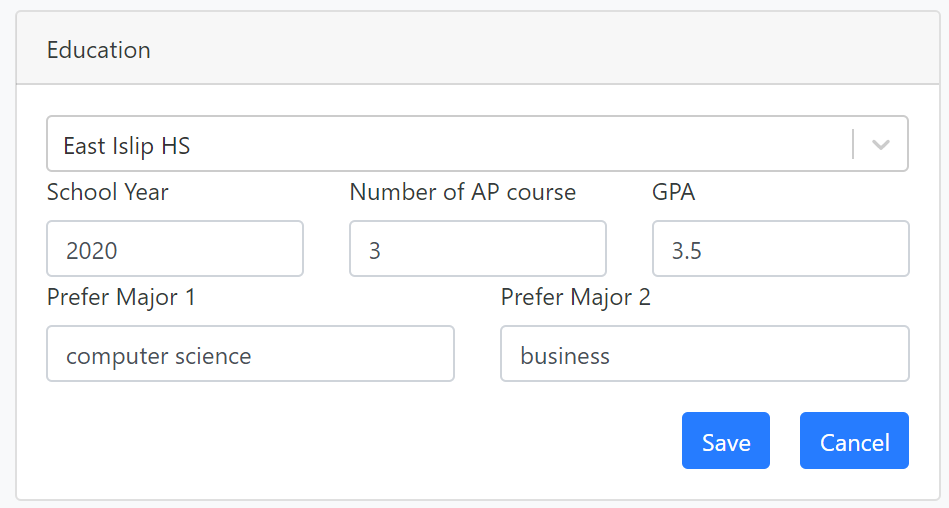
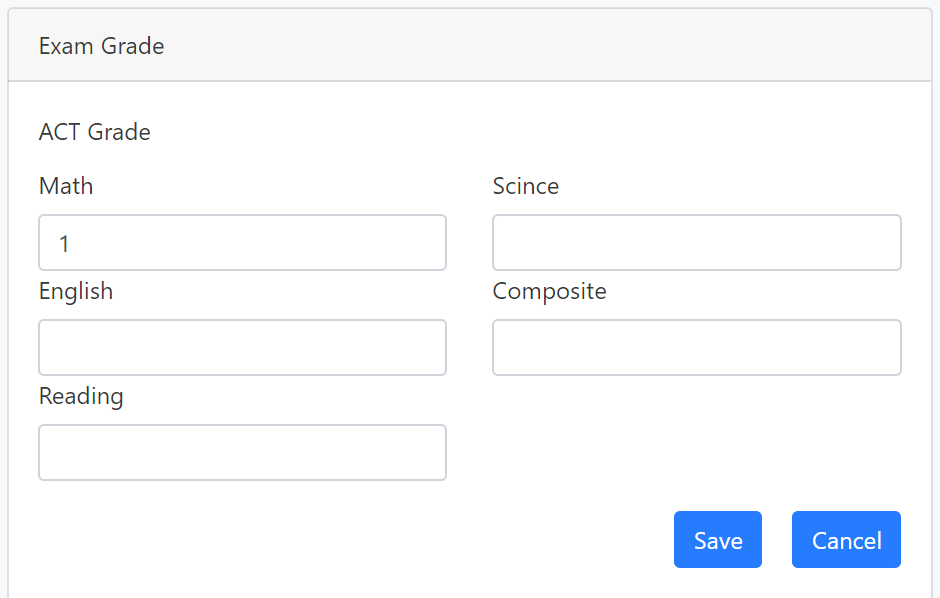
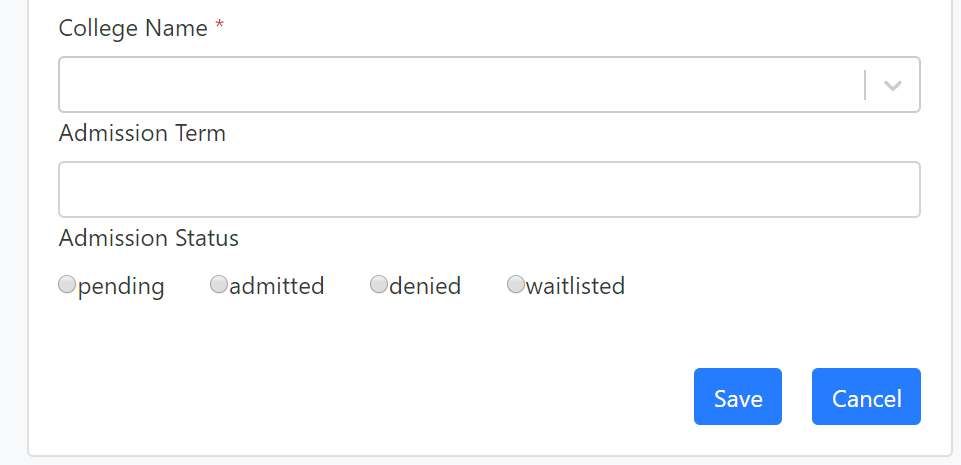
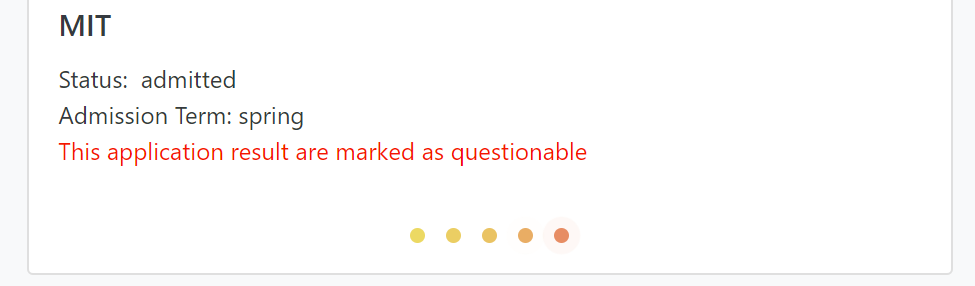


* 1. UserInfoCard



* 1. InfoCompleteCard



* 1. editingEducationCard
  2. editingExamGradeCard
  3. editingApplicationCard & addingApplicationCard
  4. LoadingPage

1. Actions

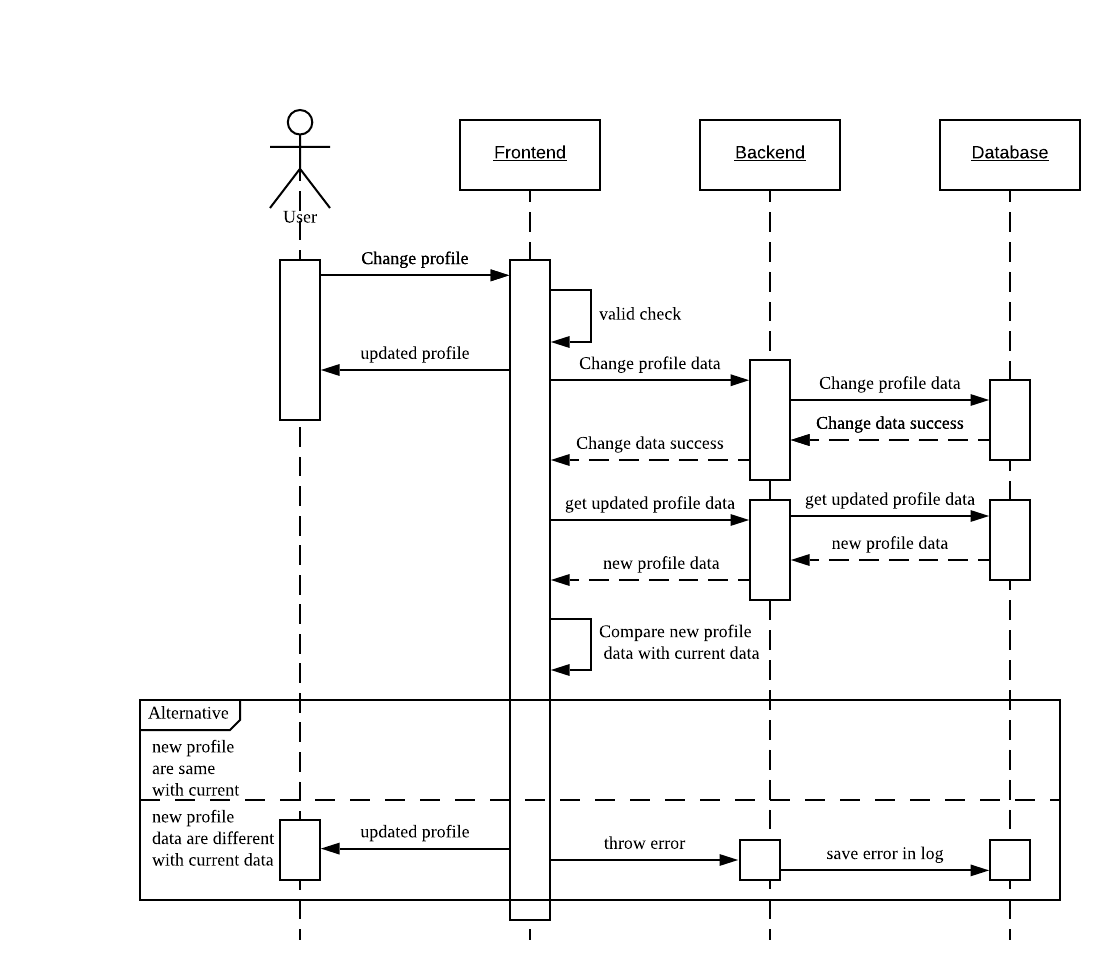
Action in profile page includes:

Profile: SaveProfile, UpadateProfile

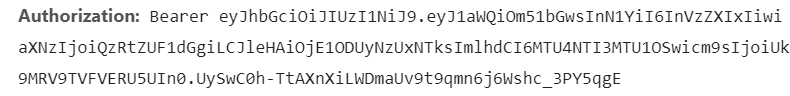
Application: Save, Update, Add, Delete for Application

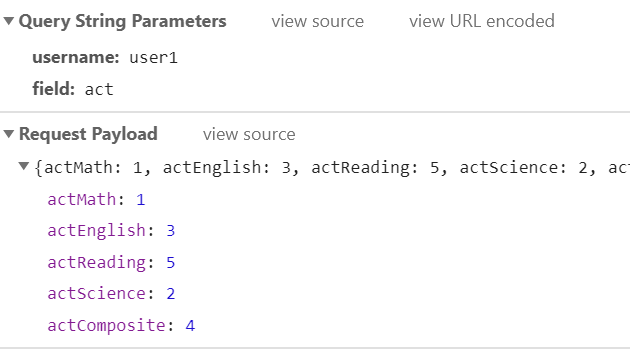
1. Async Request

Because the request is expensive and cost too much of time, I used Async request, the middleware I use is redux-sage. So the frontend will directly change the memory data after the user saves the changes. At the same time, the browser will also request the backend to change the data, then get the updated data in the database, the whole progress in my laptop cost 1s. Because React will compare the data to be modified, it means the screen will not change after the frontend gets updated data. Here is the sequence diagram to explain the async operation



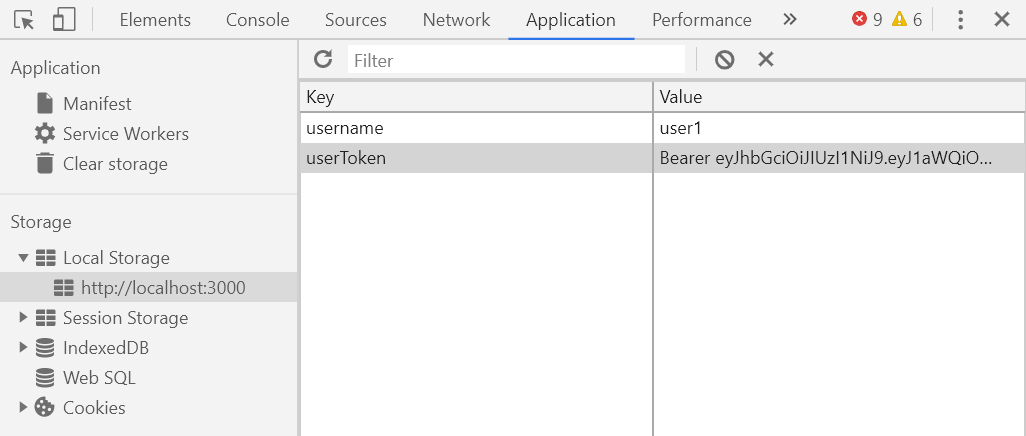
Example: Request update act grade (profile)





1. Store username and token

I stored the token and username(id) in storage



1. Backend credential implement detail ( Log and JWT credential)
2. structure

* Config
  + Advice
    - ControllerAdvice
    - ErrorAdvice
  + Annotation
    - LogAndWrap
  + Aspect
    - LogInfoAspect
  + Exception
    - DuplicateUsernameInterception
  + Filter
    - JWTAuthorizationFilter
    - JWTAuthenticationFilter
  + Handler
    - JWTAuthenticationHandler
    - JWTAccessDeniedHandler
  + Interceptor
    - LogInterceptor
  + JpaConfig
  + SecurityConfig
* Core
* Domain
* Entities
* Utils

1. JWT Authentication and Authorization

I will show detail in phase 2 because we are not going to declare this part.

1. Log Flow
   1. LogInterceprot

* Store Log into the database
* Record the error happen during the backend handle request
* Record the user info and resource user want to access
  1. LogAspect
* Print the resource that user want to access in backend console

1. LogandWrap Annotation and Controller Advice
   1. Store the annotation variable *log*’s value in the database
   2. Wrap the response data to BaseResponse format and assigned log’s value in the message field
2. ErrorAdvice and Handler
   1. Get the global error and in the server and handle it in the handler
3. Persistence
   1. Use Hibernate to map database

### Phase2

Siyong Liu

Task

* Frontend Filter Panel
* Frontend College Page
* Authentication

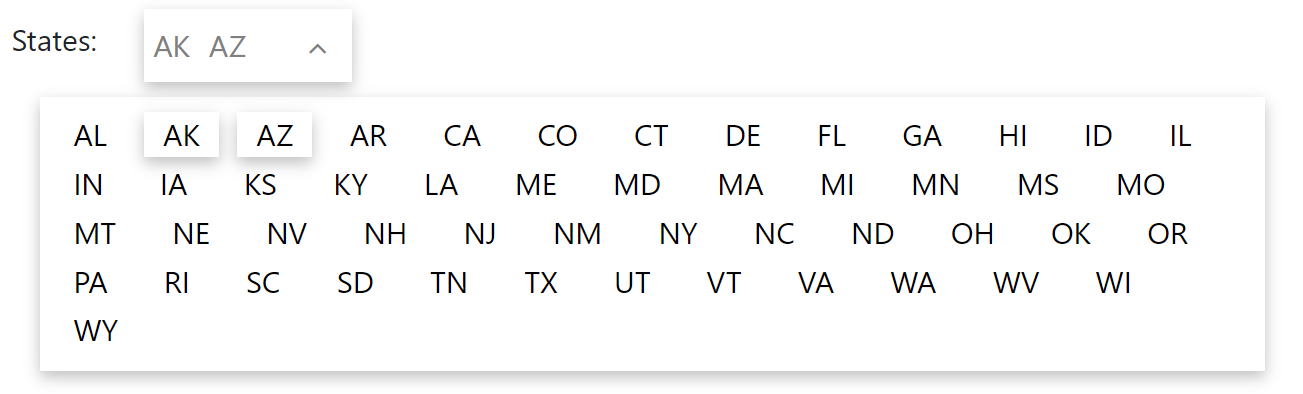
Detail:

Filter Structure:

1. Limit Filter

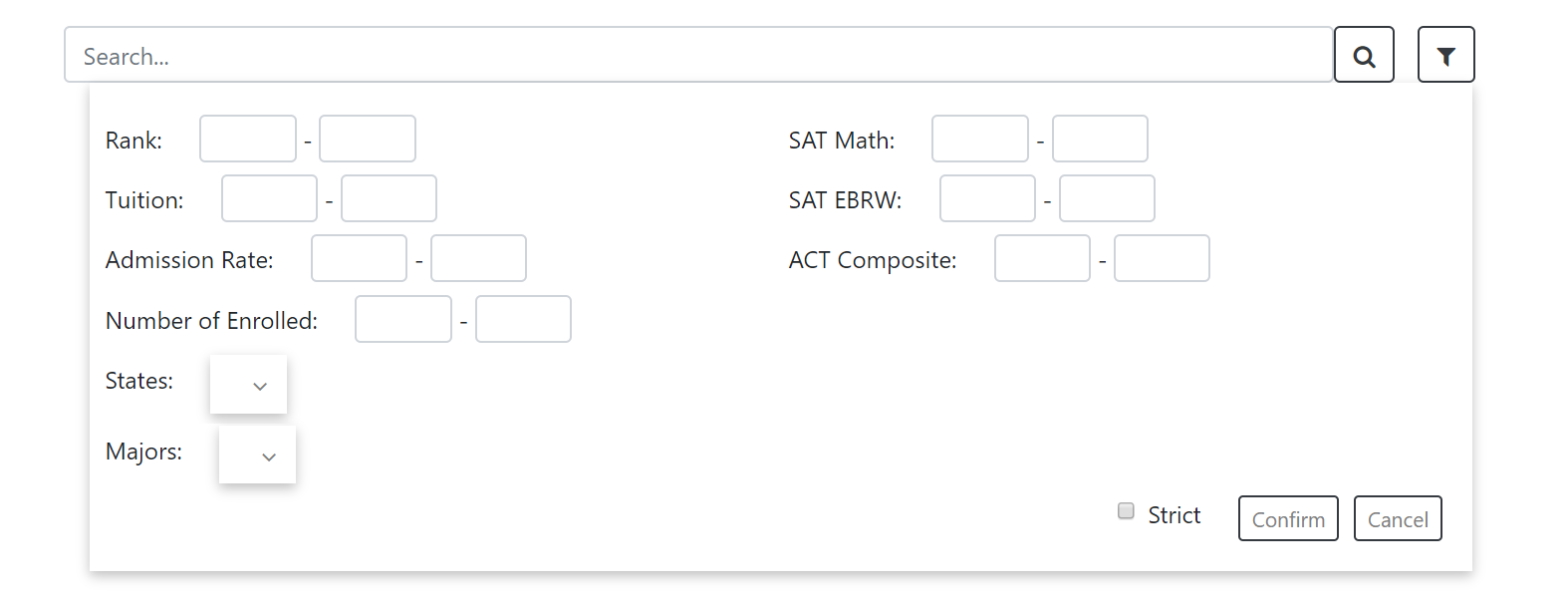


1. TagFilter



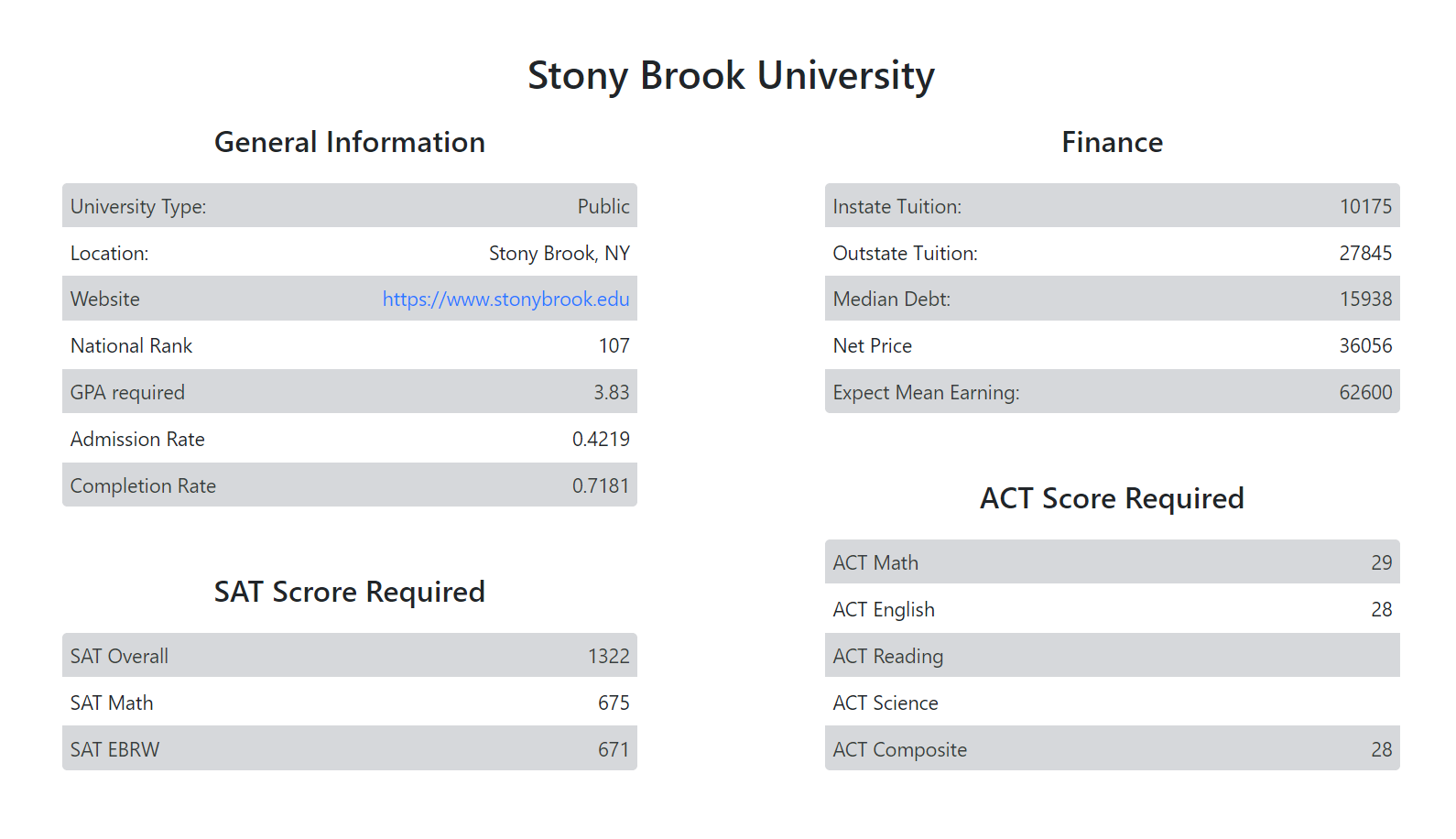
* Shows what the user choose on the bar and could be collapsed at anytime

1. FilterPanel



* The new filter could be added at anytime because I have made the limitfilter and tag filter as module

1. College Page



* College include All information about a college, and we will add profile tracker here in phase 3

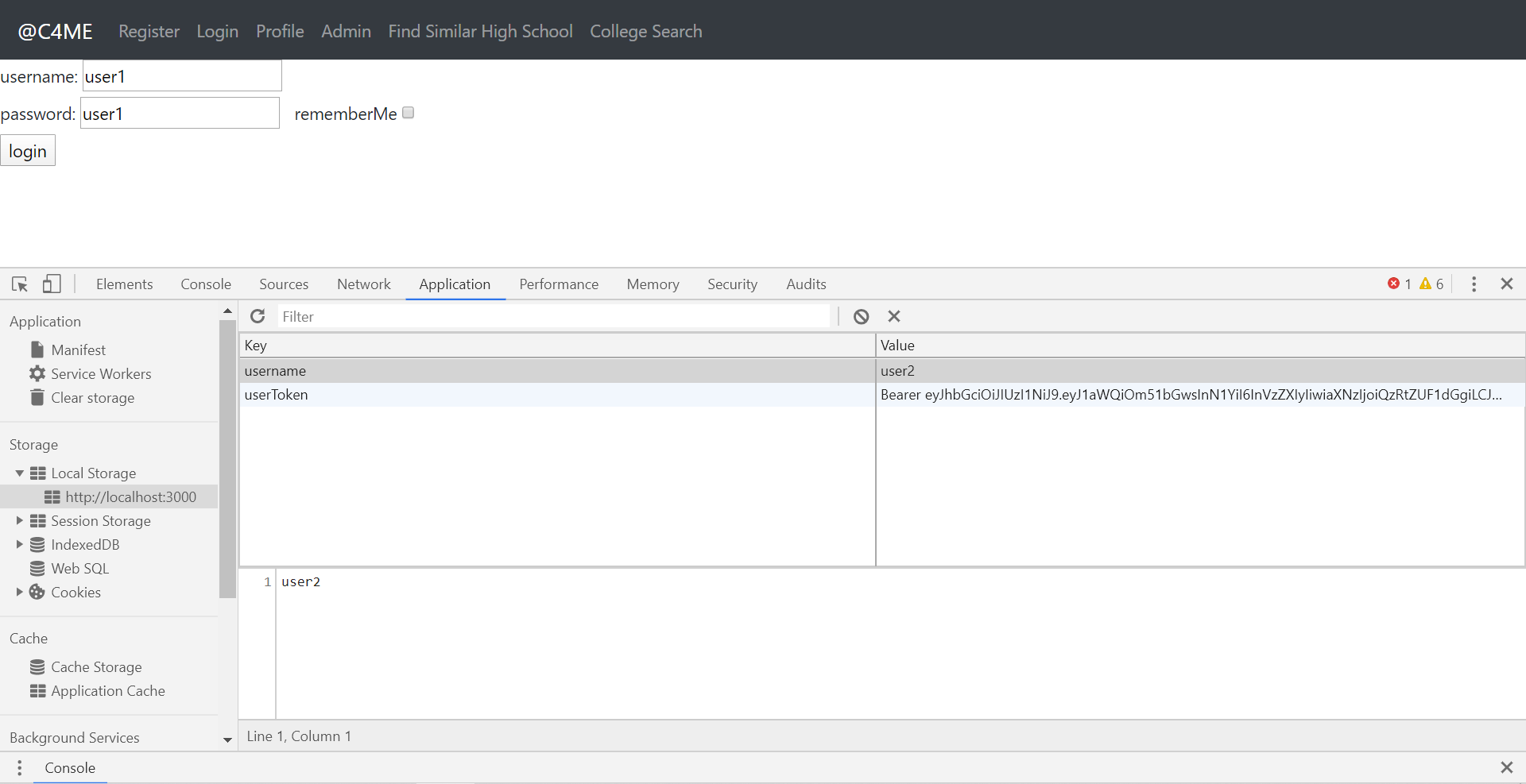
Authentication

Authentication will be seperated to 2 parts:

1. User did not authenticate cannot access admin or main page on browser
2. User did not authenticate cannot access resources

( Here, I only did the second part)

Frontend save userToken in localStorage



Every request access resource must have a user token or it will be rejected by backend.

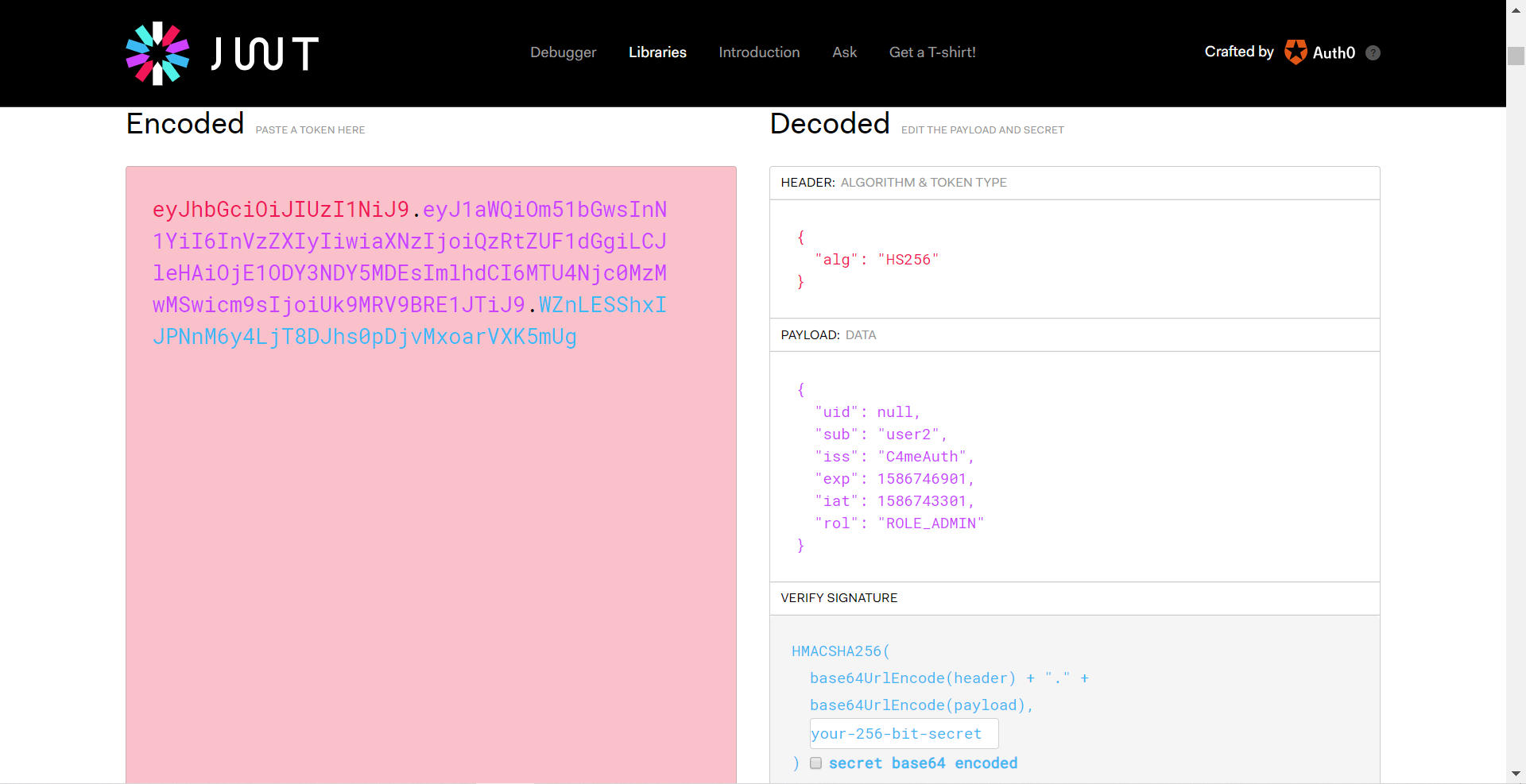
Backend Authorization and Authentication

* 1. Request which does not have a token will be rejected by resource.

Backend return full authentic request

* 1. Request which access admin resources will be rejected if it does not have admin mark, backend return access denied

JWT structure



* I added a field role in JWT, so that that backend and frontend could get this information about a user by decode JWT.

### Phase3

Siyong Liu

Task:

1. Reconsider the logic of whole program, make it easy reasonable and easy use
2. Reorganize and beautify all Screens

Detail:

1. Logic
   1. The browser should have feedback when fetching data synchronously
   2. The browser should have feedback when fetching data unsuccessfully.
   3. The browser should hide or disable the unusable buttons to prevent misuse.

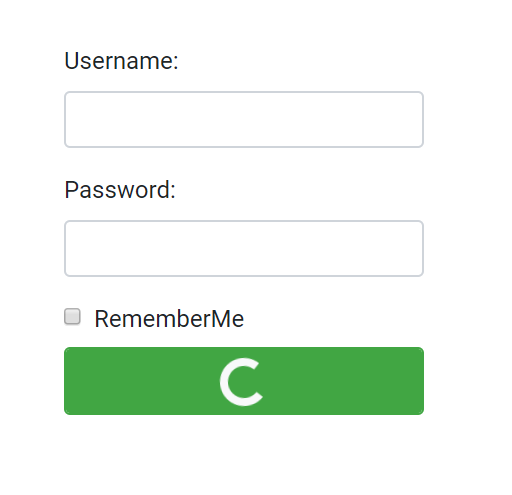
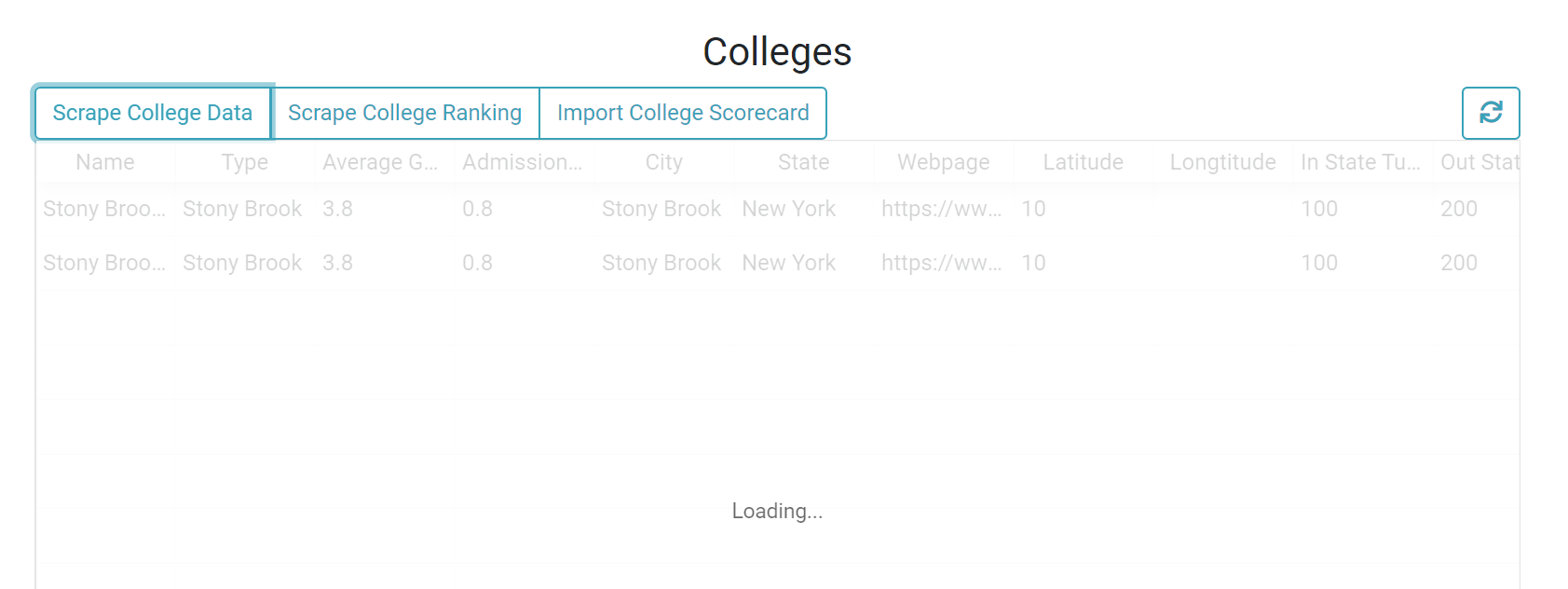
Edit button in profile

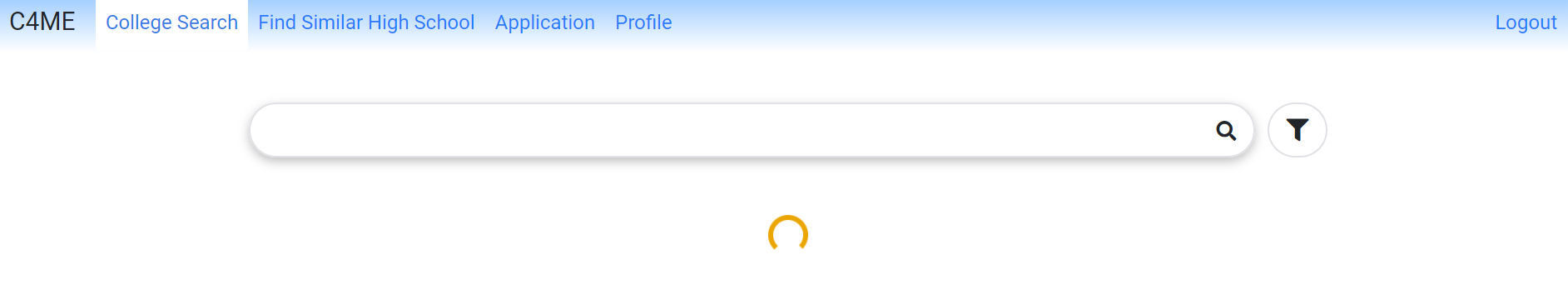
* 1. The browser should have notification if the user input invalid information
  2. The browser should restore the result list that the user searched when they want to go back from the detail page.
  3. The browser should not lose the information and login state if they refresh pages
  4. The browser should help user do some operation if predictable
  5. The browser should show default value in the page if it need
  6. The browser should have hint if it is using abbreviation
  7. The browser should hide the information and its label if it does not exist

1. Beautify
   1. Every Screen should use the same page outline layout
   2. The component with same function should have same appearance
   3. The text and background should have specific coloring scheme
   4. Use more icon the explain the information rather than text

Implementation

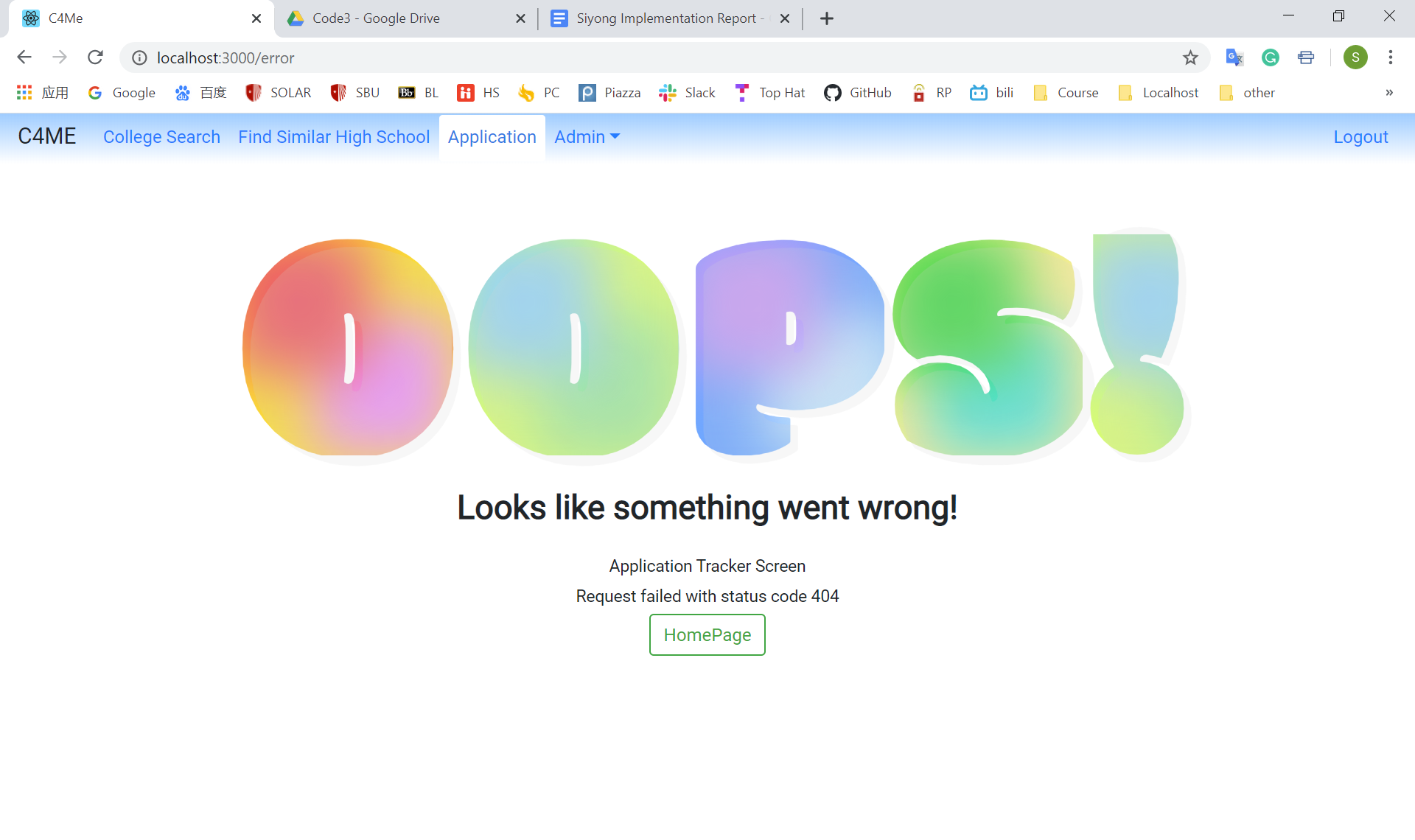
1. The browser should have feedback when fetching data synchronously

* Added loading page when searching data
* Added loading sign in table when fetching data
* Added loading on button when sign in and register



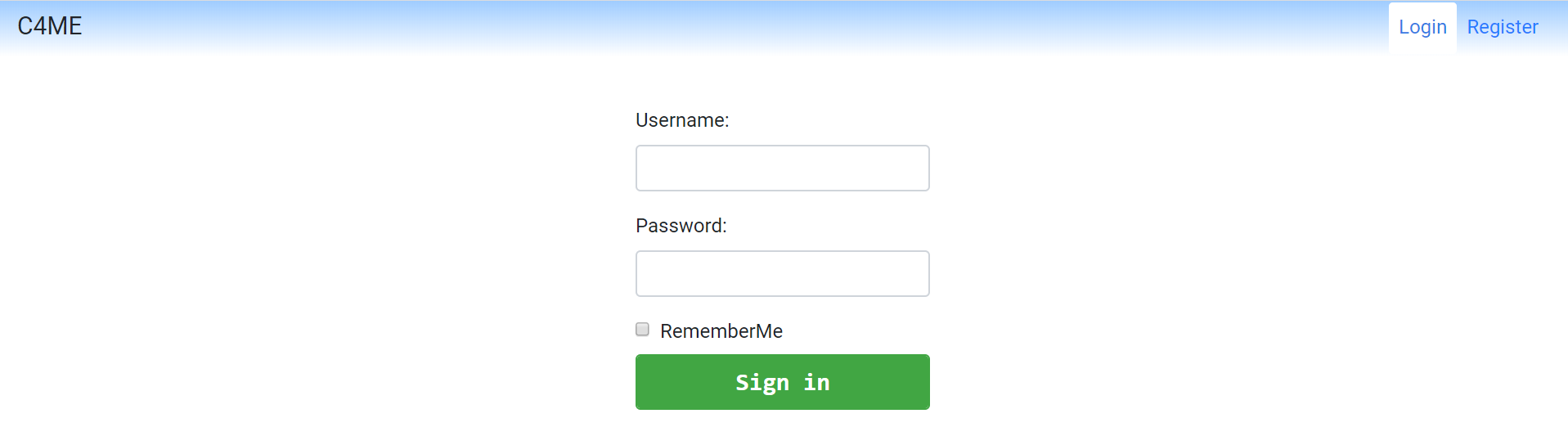
1. The browser should have feedback when fetching data unsuccessfully

* Added Error Page to to show Error



1. The browser should hide the unusable buttons to prevent misuse.

* Before authenticate, user cannot go to any pages except register and login



* If it is the first time view Search page, client will hide filter or recommender button

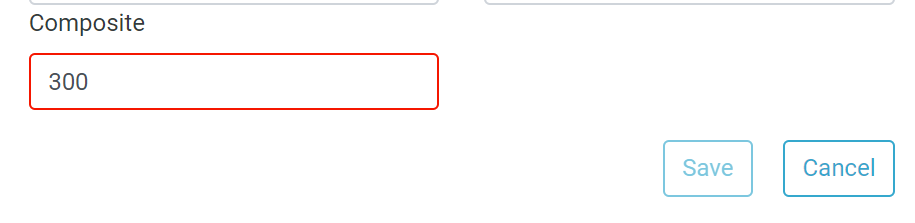


* Admin does not have profile, user cannot access admin



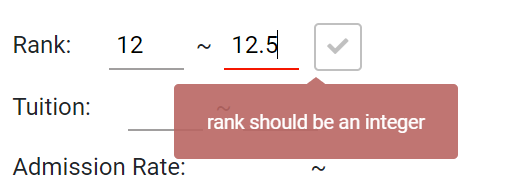
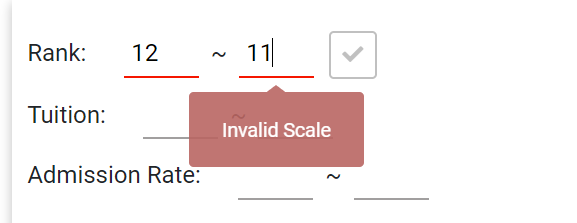


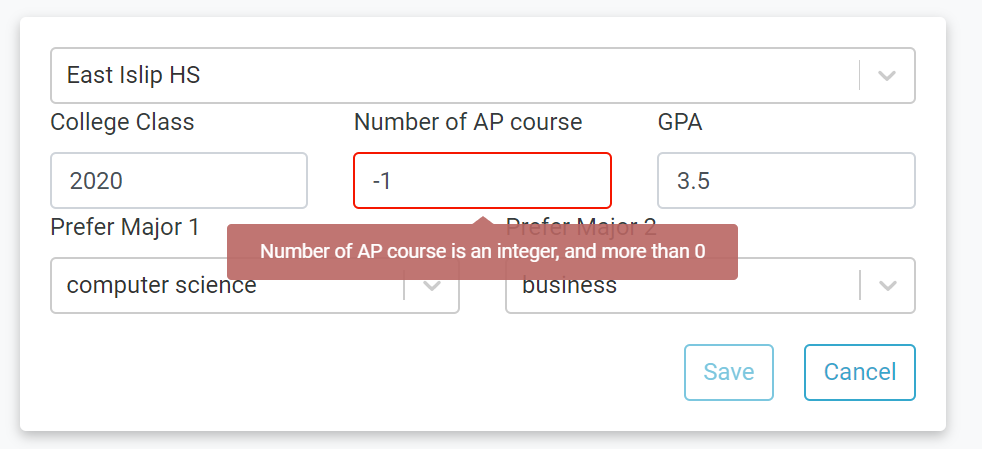
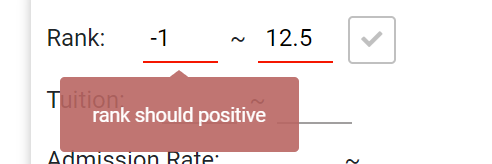
* The button will be diabled if user input some invalid information

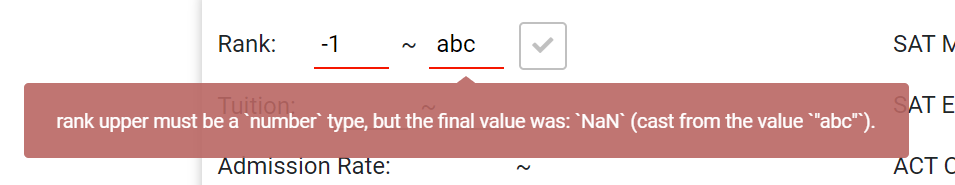


1. The browser should have notification if the user input invalid information

* Invalidation check, message will show when user hover the textfield







1. The browser should restore the result list that the user searched when they want to go back from the detail page.

workFlow:

1. The client show college List in college search screen
2. The user choose an university to see the detail of this college
3. The user clicked goback after viewing detail page
4. The user could still see his searching results

Implement:

* Save the states in the browser’s history, pop those states when user come back

1. The browser should not lose the status if they refresh pages

* Use react-persist preserve the user status, only 3 value need to preserve after optimization, which is loginState, registerState and error

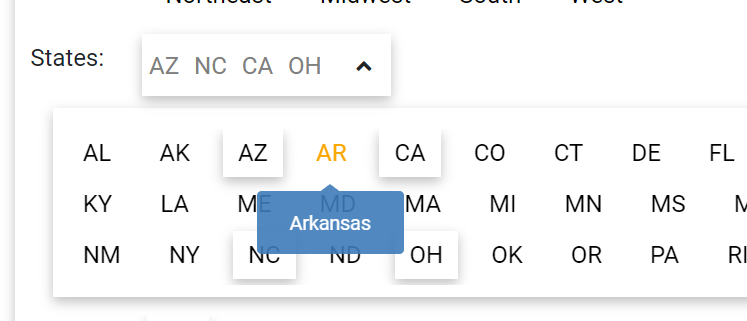
1. The browser should help user do some operation if predictable

* After user register, client login by using register username and password
* After the user clicks the application tracker in detail page, it will save that college to the application tracker textfield and search automatically.

1. The browser should show default value in the page if it need

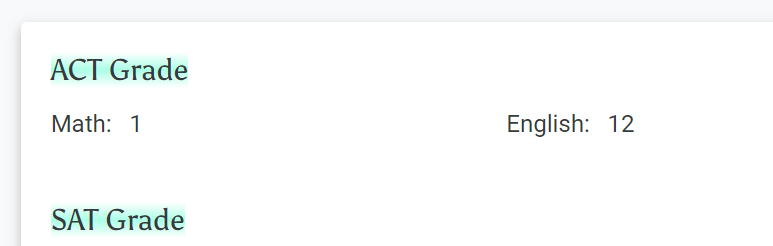
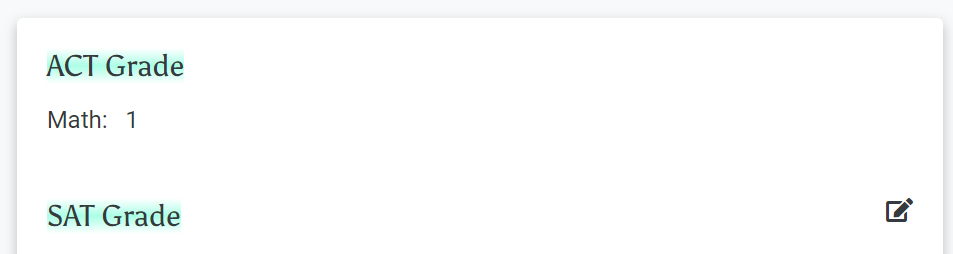
* The admin page will directly get the data in database after mounting

1. The browser should have hint if it is using abbreviation

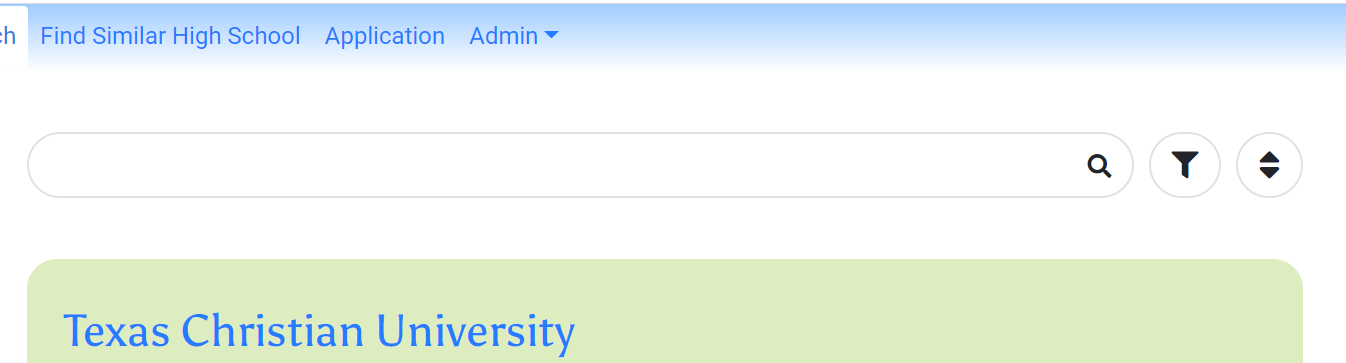


1. The browser should hide the information and its label if it does not exist

* Hide the information if it does not have

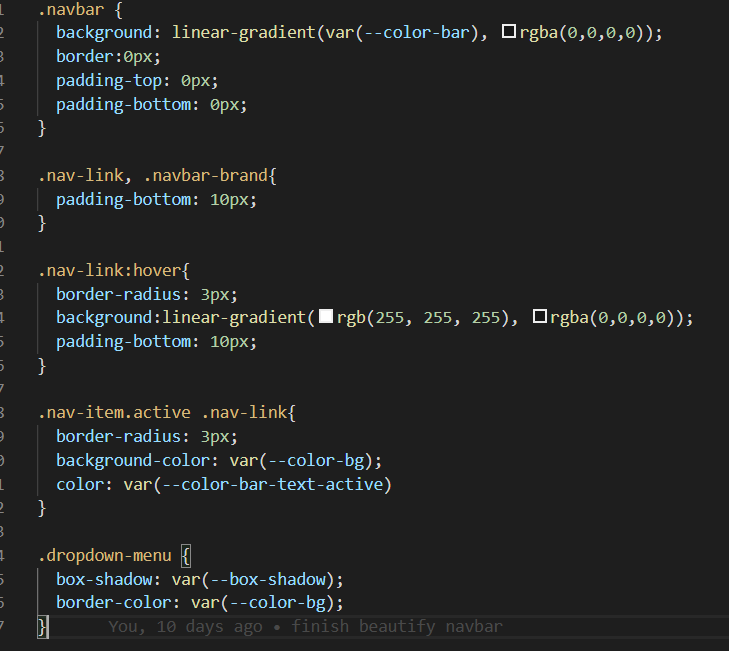
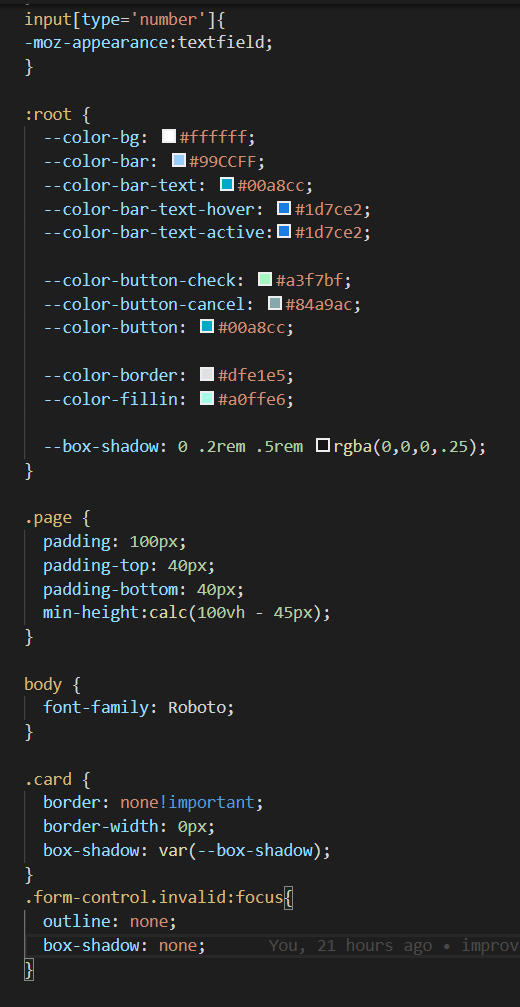


* Hide the button if user are not allowed to use ( Admin does not have recommender because they does not have profile )



1. Every Screen should use the same page outline layout

* Add Page layout
* Reorganize all css
* Define css globale box-shadow and color value

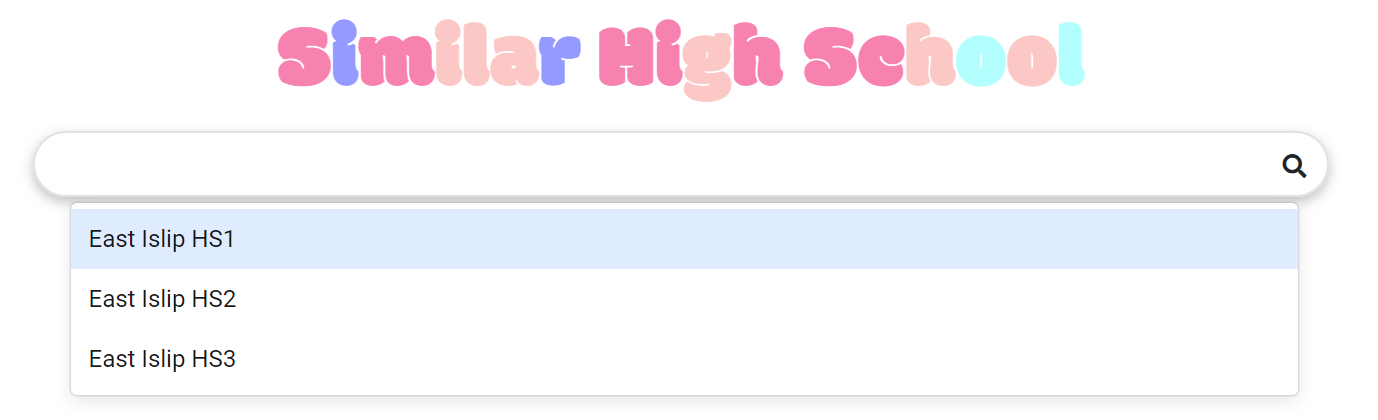


1. The component with same function should have same appearance

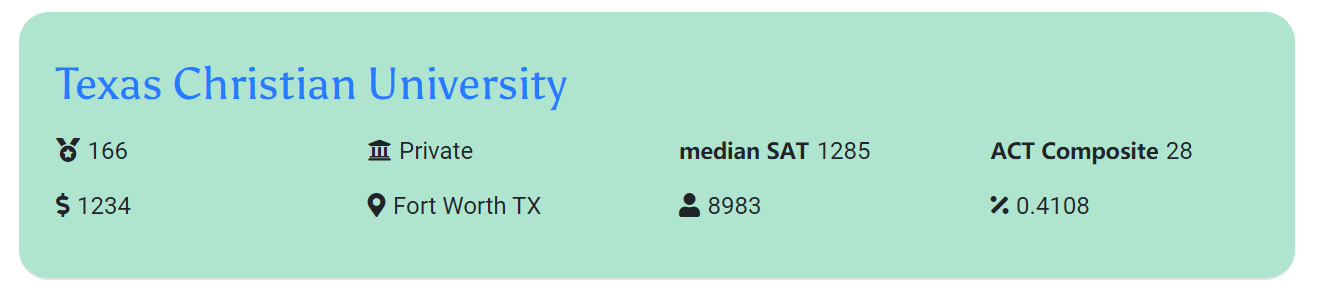
* This is a raw search component



* This is select search component



1. Use more icon the explain the information rather than text



1. Reorganize the Component

* Search Bar in 3 search pages use the same component now
* Search result card In 3 search pages use the same component now
* Validation Check are dynamic added in limit filter

Tools:

React-tooltip

Bootstrap + font-awesome

React-persist

Formik + Yup