CSCI 187

Final Design + Documentation + File Structure

Group Name → BlockBusters: Sam Barba, Henry Chen, Kobe Tran & Vincent Yu

December 4, 2020

The website is linked through firebase at:
https://blockbusterproject.web.app
Technologies:

- React JS
- Github/Git
- Firebase (realtime database and hosting)
- Movie Database technology from themoviedb.org (tmdb)

We ended up using React JS framework to deploy our website that can recommend movies or tv shows to a person interested in watching a movie or tv show. We used a Movie Database api from themoviedb.org to get recent movie releases so that our recommender can get movies and tv shows that people want to see. As a team (Henry, Sam, Kobe, and Vincent) we learned React JS and used Git/GitHub for project collaboration and sharing code. Sam focused on the backend function between the system and firebase as well as hosting on firebase while the rest of the team (Henry Kobe and Vincent) worked on organization, and Kobe implemented the firebase authentication so that login and signup work properly. Our Project is a chrome web app, but it almost works on mobile and other devices (there are minor css problems on non-chrome or mobile). We supply the user with several different lists of movies to choose from on the content pages (for both tv and movies)so there should be a movie somewhere that was recommended by us that will be a good pick for the user. Our design does not discriminate between movie services so our app is a good way to look for movies on a wide variety of services. Movies can be added to the list from 3 different places, the search window, the movie page, the tv page.

(descriptions of files in parenthesis and Folder names are followed by a colon)

File structure:

```
    public:
        favicon.ico (icon in browser tab)
        index.html (html main file)
        ...logos and manifests as well...
    src:
        Components:
            Banner:
                Banner.js (top movie in movie or tv tab)
            Content:
                Content.js (movie tab)
                ContentTV.js (tv tab)
            Footer:
                Footer.js (bottom text bar including our names)
            Home:
                Home.js (landing page)
            Login:
                Login.js (handles login, sets correct firebase auth info like userid)
                LoginPage.js (login form)
            MyList:
                MyList.js (retrieves data from firebase based on user)
            Navbar:
```

Logout.js (top logout button)

MenuItems.js (instantiates list of clickable menu items for the bar)

Navbar.js (renders the menu items and handles linking button)

Rows:

Row.js (rows displayed in content)

RowTV.js (rows displayed in contentv)

Search:

Result_Movie.js (handles movie adding search result to mylist)

Result_TVShow.js (handles tv adding search result to mylist

Search.js (handles both movies and tv show searching

User: (user tab with forms)

User.js

inlineEdit.js

Hooks:

useKeypress.js

useOnClickOutside.js

App.js (defines routing and all the different pages that are available)

Axios.js (web request)

firebaseConfig.js (includes all the info needed to communicate with firebase)

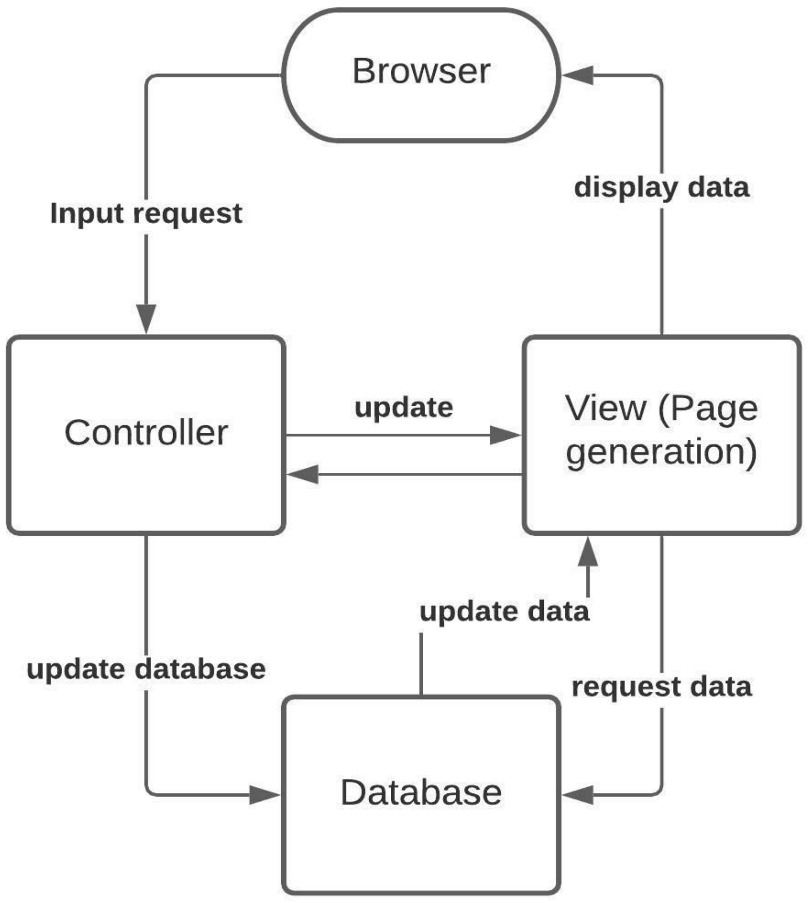
index.css (compilation of all of our css, all in one file)

Index.js (points to app)

reportWebVitals.js

requests.js (defines all the search queries used to retrieve the movies)

Figure 1: Design for overall structure using the Model-View-Controller pattern



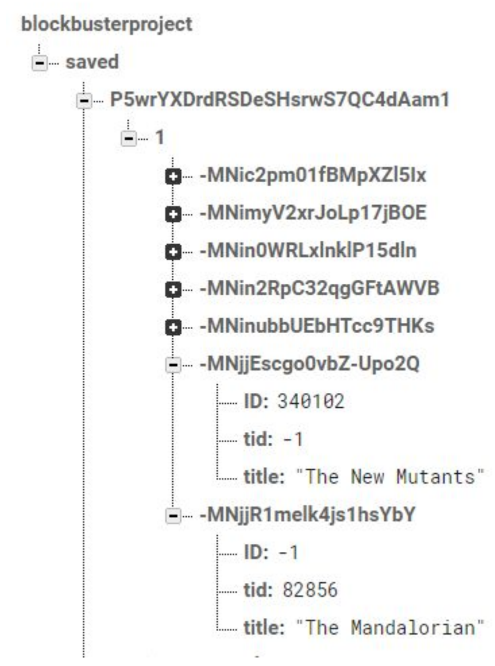
Architectural Patterns of the Model-View-Controller Pattern	
Description	Separates presentation and interaction from the system data. The Components that interact with each other: ● The database manages the system data and associated operations on that data. Also, incorporates the online movie database (themoviedb API) to update movie information on movie title, director, actors, genre, and release date. Our app contacts themoviedb to get all of the info related to movies

	that are then saved in a representation of MyList containing title and tmdb id that is in our firebase realtime database. ● The View (page generation) component defines and manages how the data is presented to the user which is sent to the browser for display. There should be an organized UI and generated recommended movie list. ● The controller component manages user interaction such as key presses and mouse clicks and passes these interactions to the View and database. So things like movie search, personal user list, clicking on movie info, etc. (check **Figure 1**)
When used	Used when the user wants to find information on recommended movies.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways, with changes made in one representation shown in all of them.
Disadvantages	May involve additional code and code complexity when the data model and interactions are simple.

Figure 2: MVP Database design (updated)

CHANGE: Due to our ability to retrieve movie information with a simple api call, instead of storing all of the information for each movie we can just store the movie lists that are lists of integers that represent the movies like they are retrieved using the API.

Screenshot from firebase



```
blockbusterproject
saved
  P5wrYXDrdRSDeSHsrwS7QC4dAam1
    1
      -MNic2pm01fBMpXZl5lx
      -MNimyV2xrJoLp17jBOE
      -MNin0WRLxlnklP15dln
      -MNin2RpC32qgGFtAWVB
      -MNinubbUEbHTcc9THKs
      -MNjjEscgo0vbZ-Upo2Q
        ID: 340102
        tid: -1
        title: "The New Mutants"
      -MNjjR1melk4js1hsYbY
        ID: -1
        tid: 82856
        title: "The Mandalorian"
```

- Root database folder

- saved folder

-userID representing user

- 1 representing that it is mylist

-series of 7 unique IDs for the posts

-first of the expanded entries represents a movie entry “the new mutants”

-second of the expanded entries represents a tv show entry “the mandalorian”

I had to use a separate field for tvID and ID because they have the same domain space (ie. 345 represents both a tv show and a different movie in the movie database api.)

User experience:

USER ENTERS => Login/Signup => Home

=>Navbar <=>Movies (Content.js)

OR<=>TV Shows (ContentTv.js

OR<=>MyList(MyList.js)

OR<=>Search(Search.js)

OR<=>Home(Home.js)

OR=>Logout(go back to login page)

(User can skip to use Navbar before logging in but then mylist doesn’t work as intended)