COMP 7214

Design Assignment

Assignment 2, Semester 2, 2019

Hamish Wadsworth (9835077)

Outline

1. Full Stack Development
2. REST API
3. Document Databases (NoSQL)
4. Single Page Applications (SPAs)
5. Design Patterns
6. References

Full Stack Development

Development Model

The final solution will be produced using the MERN stack. The main reason I have chosen this is that I have greater experience using React over Angular. Both of these front-end technologies would be suitable for the final solution however and both have ample support available for those situations when problems arise. One advantage of using React is that it is a library, as opposed to a framework, and therefore could be considered to be more flexible in terms of creating a custom project. This was not a major factor for my decision-making process but could be considered to be an advantage that React has over using the Angular framework. As learning React has been given greater focus as part of this course, in terms of classroom learning time, I have decided to go with it on that basis.

The overall design of this project includes a server-side that includes an ExpressJS server and MongoDB database which all runs within a NodeJS runtime environment. On the client side, React will be used to render views to the user.

REST API

API Paths (Routes)

The API will define routes relating to ‘albums’, ‘artists’ and ‘users’, as outlined below:

ALBUMS			
METHOD	**PATH**	**DESCRIPTION**	**JSON DATA BEING RETURNED**
GET	albums/	Get all albums (an array of album objects)	[{ _id, title, artist, releaseYear, recordLabel, albumArt }, { _id, title, artist, releaseYear, recordLabel, albumArt }]
GET	albums/:albumId	Get single album (a single album object)	{ _id, title, artist, releaseYear, recordLabel, albumArt }
POST	albums/	Create album	{ message: ‘Successfully created album’, createdAlbum: { title, artist, releaseYear, recordLabel, albumArt, _id } }
PUT	albums/:albumId	Update album	{ message: ‘Album updated’ }
DELETE	albums/:albumId	Delete album	{ message: ‘Album deleted }

ARTISTS			
METHOD	**PATH**	**DESCRIPTION**	**JSON DATA BEING RETURNED**
GET	artists/	Get all artists (an array of artist objects)	[{ _id, name, bio, }, { _id, name, bio, }]
GET	artists/:artistId	Get single artist (a single artist object)	{ _id, name, bio, }
POST	artists/	Create artist	{ message: ‘Artist created’, createdArtist: { _id, name, bio } }
PUT	artists/:artistId	Update artist	{ message: ‘Artist updated’ }
DELETE	artists/:artistId	Delete artist	{ message: ‘Artist deleted’ }

USERS			
METHOD	**PATH**	**DESCRIPTION**	**JSON DATA BEING RETURNED**
POST	/signup	New user signup	{ message: 'User created', result: { _id, email, password, isAdmin } } OR { message: 'Email already exists' }
POST	/login	Existing user login	{ message: 'Authorisation successful', token } OR { message: 'Authorisation failed' }
DELETE	/:userId	Delete user	{ message: 'User deleted' }

Document Databases (NoSQL)

A MongoDB database will be used to store data in an application-wide database, named 'grunge-music-app'. The three specific collections required within the app are 'Album', 'Artist' and 'User'. The respective properties of each are outlined below.

DATABASE COLLECTIONS					
ALBUMS		**ARTISTS**		**USERS**	
_id:	ObjectId	_id:	ObjectId	_id:	ObjectId
title:	String!	name:	String!	name:	String!
artist:	Artist!	bio:	String!	email:	String!
releaseYear:	Number!			password:	String!
recordLabel:	String!			isAdmin:	Boolean
albumArt:	String!				

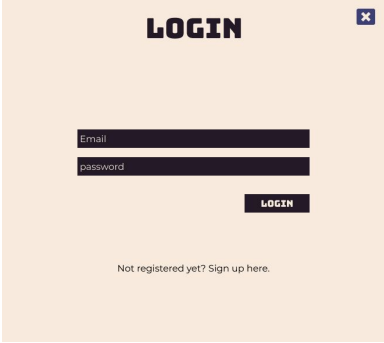
All registered users will be managed in the 'users' collection. They will enter a name, email and password (which will be hashed, using the 'bcrypt' package) and will be assigned a value of 'false' to the 'isAdmin' property. Only another user with admin rights will be able to change this value for non-admin users. The main benefit of this is to stop any registered user from being able to delete or change details of other users.

Once users are registered and have signed in using their email and password information, they have the ability to add albums to the database (albums collection) through the 'add album' functionality in the UI. As well as this, users will also be able to update and delete any albums from the database collections that they were responsible for adding. Users won't be able to edit or delete albums if they weren't the user that added them in the first instance. Within the albums collection is a custom object called 'artist'. This has two required properties in order to add this additional information.

Single Page Applications

Front End Components & Assets

COMPONENT / ASSET	DESCRIPTION
GRUNGE	**Logo (asset)** Static logo
	**Hero Image (asset)** Static image, blur value changes when another element is layered on top of it (to bring focus to the topmost element).
	**Top 5 Albums (component)** Will be populated dynamically with thumbnail images of 5 most played albums.
10:45 PM MON, 12 MAY	**Date Time (component)** Dynamically populated based on current time and date - logic carried out on server-side.
ALBUMS	**Albums Button (component)** Static - loads ‘albums’ page.
ARTISTS	**Artists Button (component)** Static - loads ‘artists’ page.
	Login Signup Button (component) Static - loads ‘login / signup’ form.



Login Signup Form (component)
Allows user to enter email and password to either signup for the first time, or login if already registered. Values from the form will be handled by the API and saved to the database as part of the 'users' collection.

Album Art (asset)
This will be loaded from the 'albums' collection in the database. It will be called via a GET request when a user wants to view this particular single album.

Media Buttons (component)
These buttons will be handled by 'onClick' events which will access the music album file selected and behave accordingly depending on which button has been clicked.

Album Info (component)
This info will come via the 'add album' form that has been completed by users. The album data itself will be stored within the 'albums' collection in the database.

Edit Button (component)
When clicked by user it will open the 'edit album' component, allowing the user to access the 'PUT' request method to change details about the album. Updates will be saved back to the database. This button will only be visible to the person who originally uploaded the album.

Delete Button (component)
When clicked it will trigger the 'DELETE' method from within the api and delete the album from the database. This button will only be visible to the person who originally uploaded the album.

Search Bar (component)
Will be managed by an ‘onTextChanged’ event that will update the users search results as they enter text into the search bar.

Add Button (component)
Static button that loads the page/component to either add an album or artist.

Album Button (component)
Clicking on this button will load the ‘view album’ page. The button itself will be dynamically loaded with the ‘album title’ and ‘artist name’ properties from the ‘albums’ collection in the database.

Pagination Buttons (component)
Static buttons that will load the next page of albums or artists from their respective pages.

Artist Button (component)
Clicking on this button will load the ‘view artist page. The button itself will be dynamically loaded with the artist name property from the ‘artists’ collection in the database.

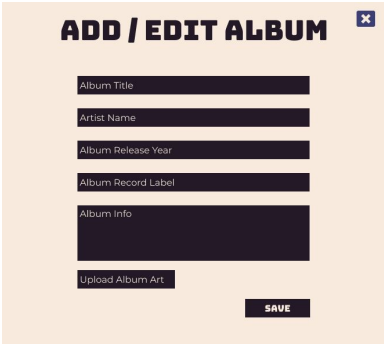
Artist Image (asset)
This is populated from a stored file in the ‘artists’ collection in the database

BUFFALO TOM

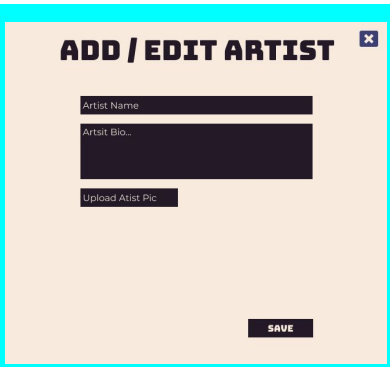
Buffalo Tom is an American alternative rock band from Boston, Massachusetts, formed in 1986. Its principal members are guitarist Bill Janovitz, bassist Chris Colbourn, and drummer Tom Maginnis. The band's name is derived from the band Buffalo Springfield and the first name of the drummer.

Artist Info (component)
This detail is provided by the user when adding an artist (via the ‘add artist’ form) and is stored in the ‘artists’ collection in the database.

Artists Other Albums (component)
In the ‘artists’ collection in the database, their albums are stored in an array. If an artist has multiple albums these will be loaded from this array into this component.

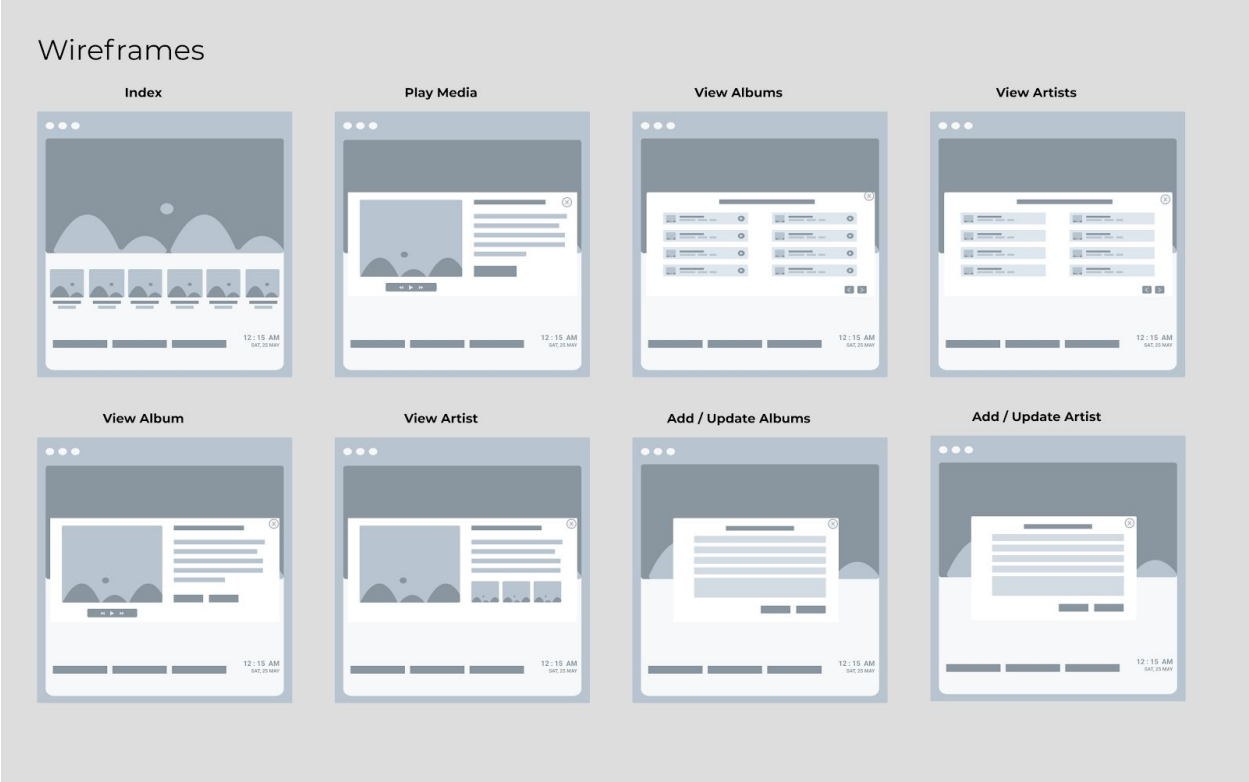


Add / Edit Album Form (component)
This form will take album data from the user and store it in the 'albums' collection in the database. This data will then be accessed by the 'play media', 'view albums' and 'view album' pages.



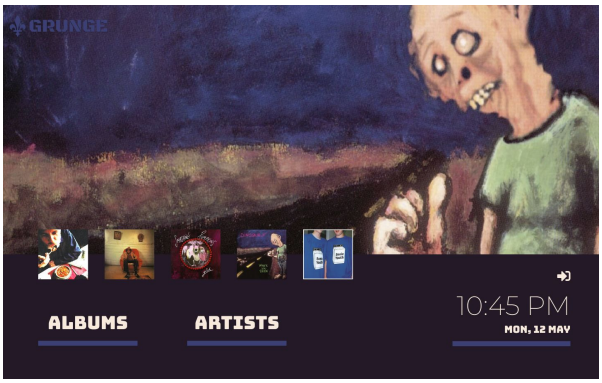
Add / Edit Artist Form (component)
This form takes artist data from the user and stores it in 'artists' collection in the database. The data is then used to populate the 'view artists' and 'view artist' pages.

Wireframes

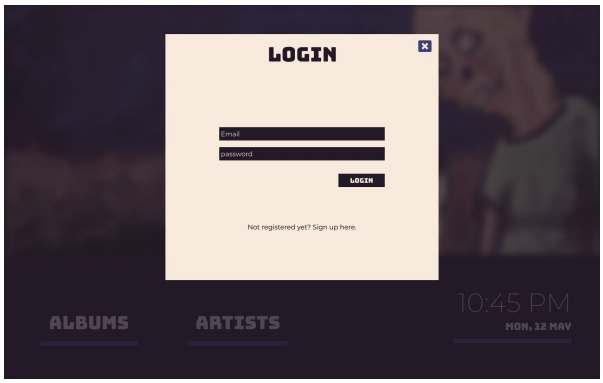


Moodboard / Storyboard

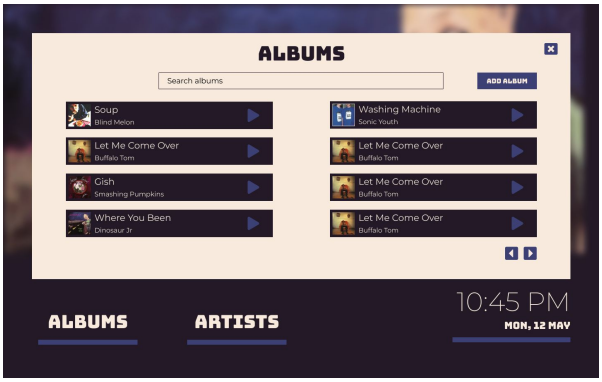
Click here to view storyboard as a functioning prototype (press the play button in the top right of the screen). This demonstrates how navigation will be handled across all pages of the application.



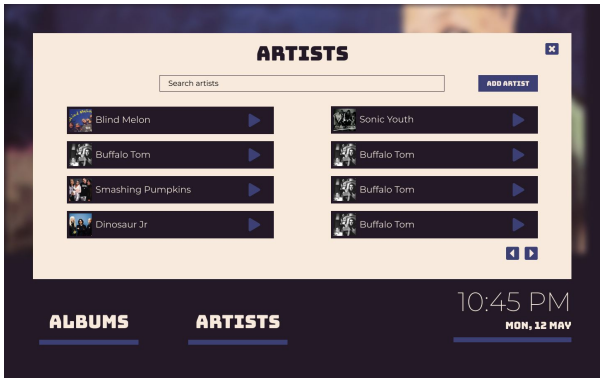
index



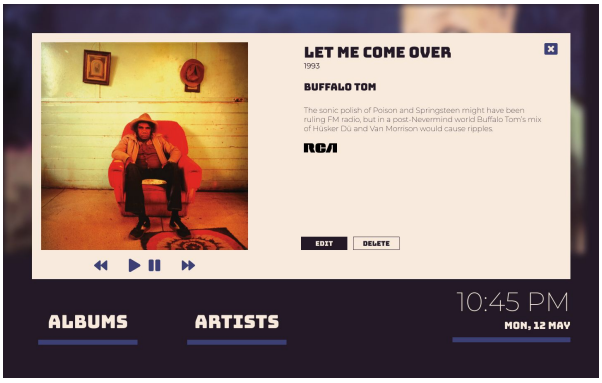
login / signup



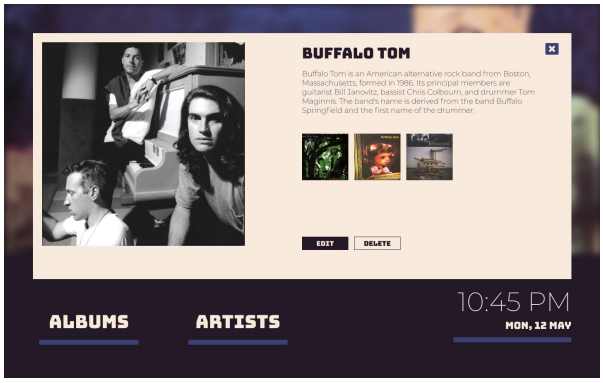
view albums



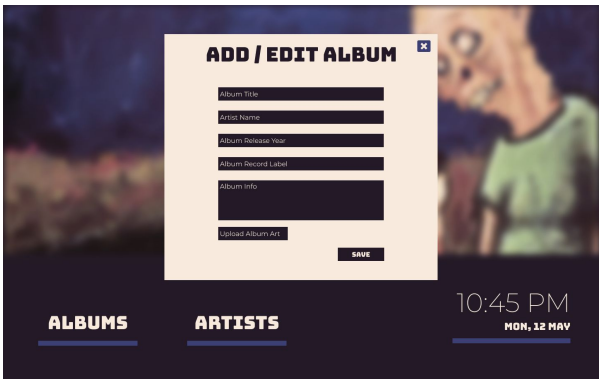
view artists



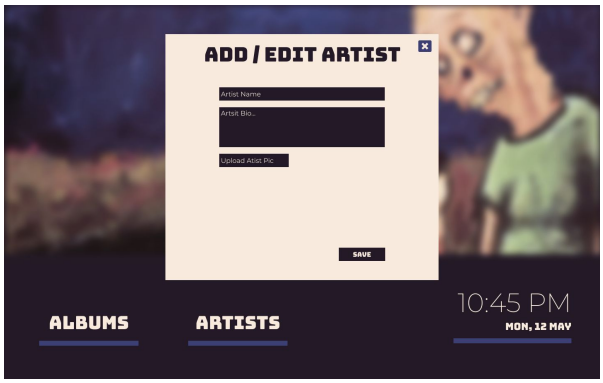
view album / play media



view artist



add / edit album



add / edit artist

Fonts

Heading Text
Bungee

Body Text
Montserrat Extra Light

LOREM IPSUM DOLOR

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Colors

Based on colors from iconic album.

Design Patterns

Client Design Pattern: Identified and Employed

This application uses an MVC (Model, View, Controller) design philosophy. On the client side the React app takes care of and renders all views. The React app has no access to logic on the server side (the ‘controller’ files) and simply renders the JSX code from within it’s own component files. This approach ensures that each part is only responsible for carrying out a single task - in the case of React it is to render the views. The components are rendered through the App.js file, which is passed to the index.js file and is in turn rendered inside the div element with the id of ‘root’ in the index.html file.

Server API Design Pattern : Identified and Employed

The server side also follows the MVC design philosophy. It consists of NodeJS, Express and a cloud-based MongoDB database (MongoDB Atlas) for storing music files and all other data. The ‘Express’ folder contains all files relating to server setup (server.js) and routing for albums, artists and users (‘routes’ folder). As well as this, the models for the database are also contained within the express folder and equally importantly is the ‘controllers’ folder - responsible for server-side logic - providing the functionality for the GET, PUT, POST and DELETE requests.

References

Academind. (2018, July 25). SQL vs NoSQL or MySQL vs MongoDB [YouTube]. Retrieved September 26, 2019, from https://www.youtube.com/watch?v=ZS_kXvOeQ5Y

Ganguly, S. (2019, February 27). Most Popular Technology Stack To Choose From: Full Stack Vs. MEAN Stack Vs. MERN Stack In 2019. Retrieved September 26, 2019, from https://hackernoon.com/most-popular-technology-stack-to-choose-from-full-stack-vs-mean-stack-vs-mern-stack-in-2019-d12c0a17439a

Academind. (2018a, January 17). Building a RESTful API. Retrieved September 26, 2019, from https://www.academind.com/learn/node-js/building-a-restful-api-with/

Sherman, P. (2018, April 11). How Single-Page Applications Work. Retrieved September 26, 2019, from https://blog.pshrmn.com/how-single-page-applications-work/