Outfox 2021 Documentation

Web Development Team
Matthew Moldawsky, Shannon Cordoni, Kaylin Moss, Ikponwonsa Asaolu, and Melissa Aguilar
Professor Gormanly

Table of Contents

Outfox Icon

Overview - What is Outfox?:

Outfox is designed to be an open-source, learning management system application. When we graduate, finish an online course, complete a training seminar, attend a conference, or even just switch to a new method of managing our notes and documents we lose something in the process. Often years later we find ourselves frustrated and retracing our steps though a past learning or research journey only to realize a crucial piece cannot be located or reconstructed. Outfox is an environment where we can conduct our learning, research our ideas, and remember all the lessons we learn, forever. Outfox is not owned by an academic institution or corporation but by everyone. This means that everything associated with you stays with you for as long as you want. In addition, users' data will never be used against them. It will be only for them and it will not be sold or used for marketing. The eventual goal is that students and instructors would be using this environment to give assignments, share lessons, and share resources. A student would then be able to submit an assignment and be returned a grade just like in most learning management systems. Opening a safe central hub of learning means collaboration and sharing of information can be easier than ever. Eventually, schools could consider using Outfox as their primary LMS. This would ensure the lessons students are taught would never be lost.

The front-end of this application was built using ReactJS while the backend was built using NodeJS. The database of Outfox is hosted by PostgreSQL. This application also uses redux in order to connect the front-end and the back-end together. This project is a continuation from the same one during CS Project Fall 2020. Some parts of this document will cover changes from the previous project and mention parts of the project that were unchanged. There is also another team that worked on Outfox called **OutfoxAI**. We combined our work into one whole project. However, this documentation will cover the information that pertains to the WebDev team.

User Guide:

A user would create an account using the “Sign Up” button on the top navigation bar of the Home Screen. If the user already has an account, then they would use the “Login” button on the top navigation bar of the Home Screen. Once the user logs in, they will see their dashboard. The top navigation bar and side navigation bar will appear on every page of Outfox. The top navigation bar is used for extra features such as a search bar, settings page, and profile page. The right side of the dashboard will show friend activity. However, it is currently just fake data.The side navigation bar is how the user will move throughout the various pages of Outfox.

These pages include:

- Dashboard
- Groups
- Resources
- Assignments
- Explore
- Friends
- Lessons
- Help

Dashboard:

The dashboard has dropdown containers that display some information about each of the user’s items in each category. There is one for Groups, Resources, Assignments, Lessons, and items shared with the user. The “Create” button in each dropdown container takes the user to the respective page where they can create an item in that category.

Groups:

A group is used for grouping resources together. For example, if you have a Calculus group, you could add calculus related resources to the group. You have to create a group first before adding a resource to it.

Resources:

A resource is used for links, files, and text that can be added to a group, assignment, or lesson. They can be used for information about a subject. Related and helpful links can be attached. A file can be attached which may or may not be related to a class.

Assignments:

An assignment is how a user can assign a task to another user. It has a way to send the recipient attachments (a Resource). Users will be able to submit an assignment they receive by using a text editor or by attaching another resource. An assignment has an open date, a due date, and a close date. The user that shares the assignment can grade the assignment and the recipient will be able to see the change. The page itself will show the user's created assignments and their shared assignments. A user can edit, delete, or share an assignment.

Explore:

The explore page is used to view other users and send them friend requests. You can click on another user to view their profile. If you click "Connect," you will send a friend request to that user.

Friends:

The friends page shows the user their current and pending friends. A user is able to share Groups, Resources, Assignments, and Lessons with a friend. If you click on a friend's profile, you can remove them from your friends list.

Lessons:

The lessons page shows the user their created lessons and any shared lessons. A user can view their lesson and edit or delete them. A user cannot edit their shared lessons. Lessons are used for sharing any lectures or notes with another user. A lesson has a built in text editor which can be used to format how the lesson looks.

User Set-Up:

To set up an Outfox account, click the Sign Up button on the top navigation bar on the home page. You will be asked to input your first name, last name, username, email, and password. There are options to sign-up using a Google account, Microsoft account, or Facebook account. However, these have not been implemented yet.

Set Up Guide (Developer Version):

NOTE: This set-up guide is only specific for the version of Outfox that the Webdev team worked on. The Set-Up guide of OutfoxAI contains the combined setup guide which includes the following information.

Installing PostgreSQL Database

Windows

For Windows, the best way to install PostgreSQL is through a website called Enterprise Database. It will ask you to create a free account, please do that because it's easier to install this all pre-packaged. The reason being is the company bundles PostgreSQL, pgAdmin and the company's own stack builder, which is only a few of the tools. Select the 64-bit version unless you have a 32-bit version of Windows. We are using PostgreSQL version 14 (13 also works).

1. Install the interactive installer from Enterprise Database - PostgreSQL.
2. Run the installer as administrator and answer yes to allowing the program to make changes to your computer. When it's asking to make changes to your computer, it will be editing your path variable meaning when you open Windows Powershell or Command Line you can run the command psql.
3. Accept the default install location.

4. Click through until you hit the select components screen. At the select components screen select the boxes for pgAdmin, Stack Builder, and Command Line Tools.
5. It is then going to ask where to store the data. We would **heavily** suggest using the default location unless you know what you are doing.
6. It is going to ask to set a password. Set it to something you can remember because this will be the password for the postgres user which is the root user.
7. Next, it is going to ask you to select the port number. We would suggest using the default of **5432** unless you have another software running on that port. (Like an older version of PostgreSQL or MySQL. In this case use 5433.)
8. Select your locale. We suggest using the default one. Then, click your way through the screens until the installation begins.
9. Once the installation is done, it will ask you if you would like to obtain additional packages through Enterprise DB's stack builder. Select the box and hit finish.
10. Once the stack builder launches, hit the PostgreSQL installation on the drop down menu and click **next**.
11. Expand the spatial extensions menu and select either the 32-bit or 64-bit version of the PostGIS bundle for the version of PostgreSQL you installed. Then, expand the add-ons, tools and utilities menu and select EDB Language Pack. Click though several times until all software is installed.
12. When the install for the files is finished, click **next** to install both components. Agree to the license for PostGIS. Click through until you are asked to choose components. Make sure "**PostGIS and Create spatial database"** are selected. Click **next** and accept the default database location.
13. Enter the password you made for the Postgres user that was prompted earlier in the installation. Then continue through the prompts until PostGIS is installed.
14. Answer **yes** when asked to register GDAL. Also **yes** to setting **POSTGIS_ENABLE_DRIVERS and POSTGIS_ENABLE_OUTDB_RASTERS**.
15. Installation is complete. Please view the portion on how to get the Outfoxdb created.

MacOS

For MacOS the best place to install PostgreSQL is from the PostgreSQL App website. This includes all the same drivers that the windows install has but it's compiled into an easier to install form. We are going to be installing version v12.4 or v13. In our case, they both work.

1. Go to the PostgreSQL App website and install the app image which will end in .dmg.
2. Double click the .dmg file to open it, and then drag and drop the app icon into your applications folder.

3. Open PostgreSQL by double clicking the app icon. When it opens click Initialize to create and start a PostgreSQL database.

Hint - A small elephant icon will be in your menu bar indicating that you now have a database running. If you would like CLI tools follow the instructions at PostgreSQL App CLI Tools.

Now we are going to install pgAdmin in order to manage our postgreSQL. This is why the command line tools are necessary. (Command line tools are more convenient if you are used to working with them). Since Postgres App does not include pgAdmin we will do that now.

1. Visit the pgAdmin website. This link will bring you to the macOS downloads automatically.
2. Select the latest version and download the installer. (The file extension is .dmg)
3. Double click the .dmg file and click through the prompt accepting the terms. Then drag pgAdmin's elephant app icon into your Applications folder.
4. Double click the app icon to launch pgAdmin. (May be called pgAdmin4 - the same thing)

Hint - Upon first opening pgAdmin a dialog might appear saying "pgAdmin.app can;t be opened because it is from an unidentified developer.". Right click the icon and select Open. The next dialog should give you the option to open the app. Your Mac will remember you've granted it permission.

Linux

The best way to install postgresql on linux is to check your distributions documentation on it. So, I will only be covering the install for Ubuntu.

Ubuntu

1. By default Ubuntu ships with PostgreSQL in the aptitude package manager.
2. Open up your terminal and run `sudo apt-get install postgresql postgresql-contrib`.
3. Restart your system after the install and the service will be run automatically.

Note - On Ubuntu pgAdmin typically isn't available. If you would like to try, use pgAdmin4 Apt Install.

Creating the User and Database

Windows and Mac (and linux if you installed pgAdmin)

1. Open pgAdmin and if asked to put in the postgres user password you created during the install.
2. Hit the plus on servers. This will show you the PostgreSQL database currently running.
3. Right click on Login/Group Rules. Then hit create new role the name will be sqlize.
4. Go to the privileges tab and click can login, superuser, create database, update catalog, and inherit rights from parent roles. Hit save.
5. Now right click the database called PostgreSQL and click create database.
6. A prompt will appear and where it says database write outfoxdb.
7. Underneath there will be a dropdown for the owner and set the owner to sqlize.

Linux (Only terminal)

1. Open the terminal and type `sudo -i -u postgres`. This will switch you over to the postgres user which is in charge of postgreSQL.
2. Create the sqlize role in terminal -> `psql -c "CREATE ROLE sqlize WITH CREATEDB CREATEROLE LOGIN INHERIT PASSWORD 'yourpasswordhere';"`
3. In the terminal type `createdb -O sqlize outfoxdb`. This will create a database called outfox with sqlize as the owner.

Allow logging into the database without a password.

This part is strictly for your local machine. In production this would not be a standard thing.

Windows

1. Go to *`C:/Program Files/postgresql/13/data`* and open *pg_hba.conf.*
2. When opening the pg_hba.conf scroll to the bottom and you should see two lines. One will say `*host all all 127.0.0.1/32 md5*` and the other will be `*host all all ::1/128 md5*`.
3. Where it says md5 you will change them to **trust**. This allows connections from your local machine to your database to not require a password.

Mac

1. Open the terminal and type `*ps aux | grep postgres*`. This only works if a postgreSQL server is running.
2. The output will look like `*user 3391 0.0 0.0 2615032 804 ?? S 18Dec15 1:01.04 /usr/local/opt/postgresql/bin/postgres -D /usr/local/var/postgres -r*

/usr/local/var/postgres/server.log`. Most of this does not matter, the only important part is the part after `-D` so in this case `/usr/local/var/postgres`. This is where postgres is installed on your mac machine.
3. Navigate to where your Mac has installed postgres using `*cd [folder name here]*` and `cd ..`.
4. Once you have reached the postgres folder you should be able to see the pg_hba.conf file.
5. Open it in a text editor and scroll down to the bottom. You will see `*host all all 127.0.0.1/32 md5*` and the other will be `*host all all ::1/128 md5*`. Change the md5 part to **trust**. This will allow you to connect your local machine to your local database with no password,

Linux

1. Open the terminal and use postgres user to open up psql by running `*sudo -u postgres psq*l`. This will make you enter the psql shell as postgres.
2. Run SHOW `hba_file;` in the psql terminal. This will show you the directory in which the pg_hba.conf is installed.
3. Switch to the postgres user by running `sudo -i -u postgres` and navigate to the directory that the pg_hba.conf is located. (In order to get there you need to be the user postgres)
4. Open the pg_hba.conf using vim, emacs, or nano(easiest) and scroll down until you see `*host all all 127.0.0.1/32 md5*` and the other will be `*host all all ::1/128 md5*`.
5. Switch where it says md5 to **trust**. This will allow our local machine to connect to the local database without a password.

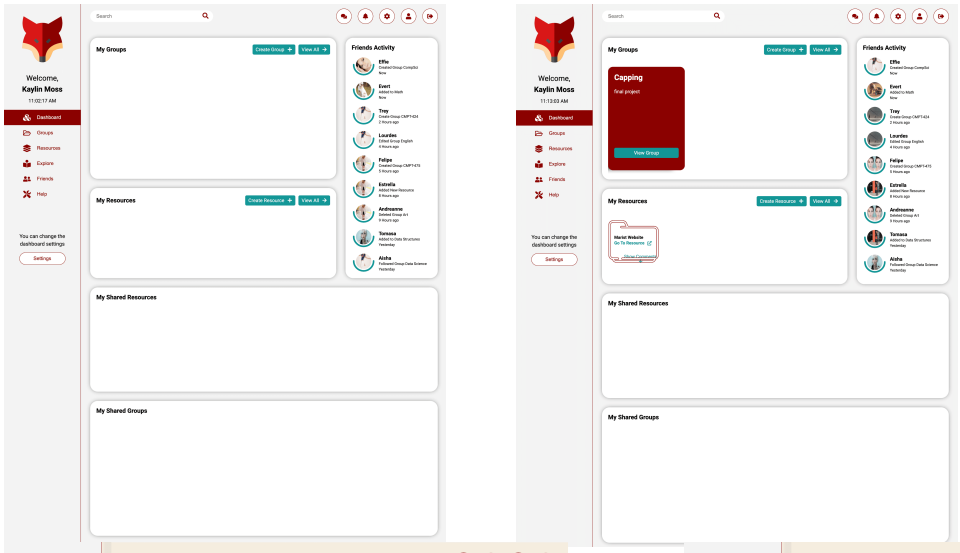
Testing the database by running the project

This part applies to every operating system.

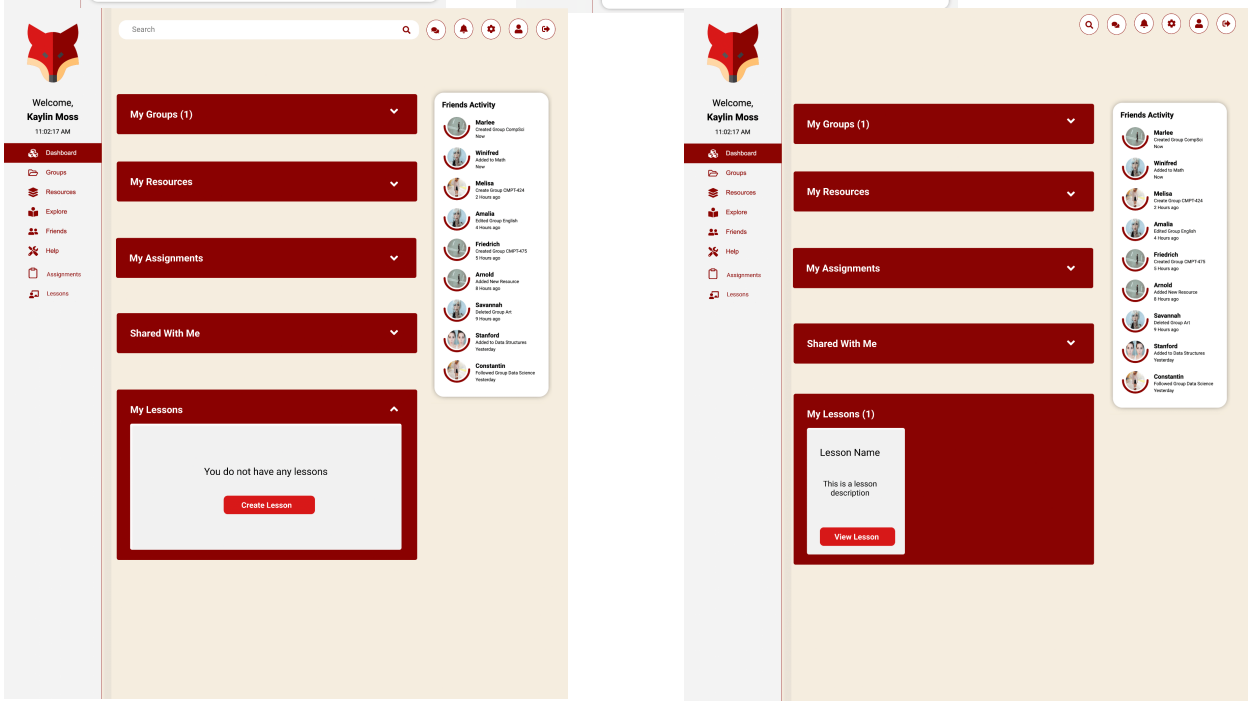
1. Pull from master. Master will always work. So, in order to make sure our project works we will pull/clone from master.
2. Double check that postgres is running. Check however you would for your respective operating system.
3. In the outfox directory run `npm i`.
4. In the outfox/server directory run `npm i`.
5. In the outfox/client directory run `npm i`.
6. In the outfox directory run ‘npm run dev’.

Design Process:

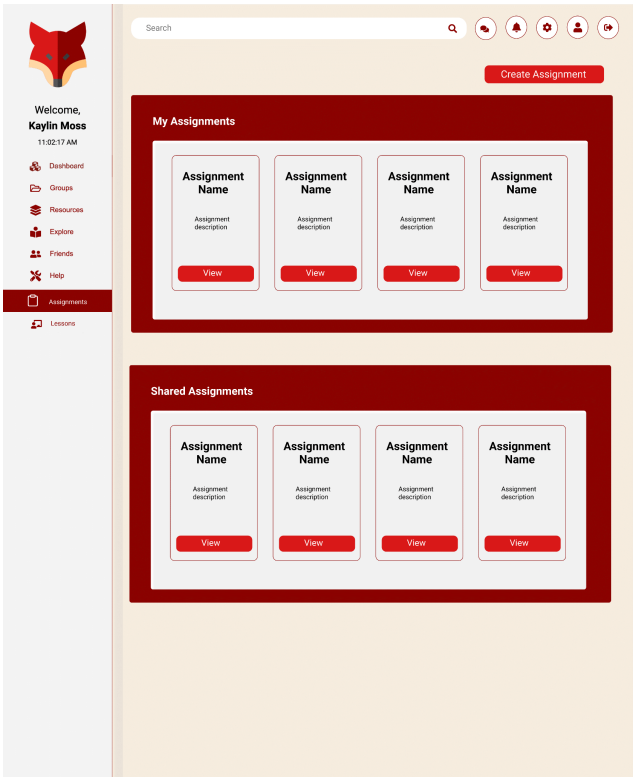
During the design phase, we drafted up new UI designs for existing pages for Outfox as well as new pages for the new features we added. Some of these changes to existing pages include a more consistent color scheme and indicating that the user does not have a given object. We also added dropdown menus to the dashboard to avoid clutter from the new features added to the site. We first brainstormed using a whiteboard to come up with basic design ideas. Afterwards, we used Figma to make wireframe models of new and existing pages for Outfox. We received feedback on certain parts of the designs and made the changes accordingly. The following images are the final drafts of the Figma models for the design changes to Outfox.



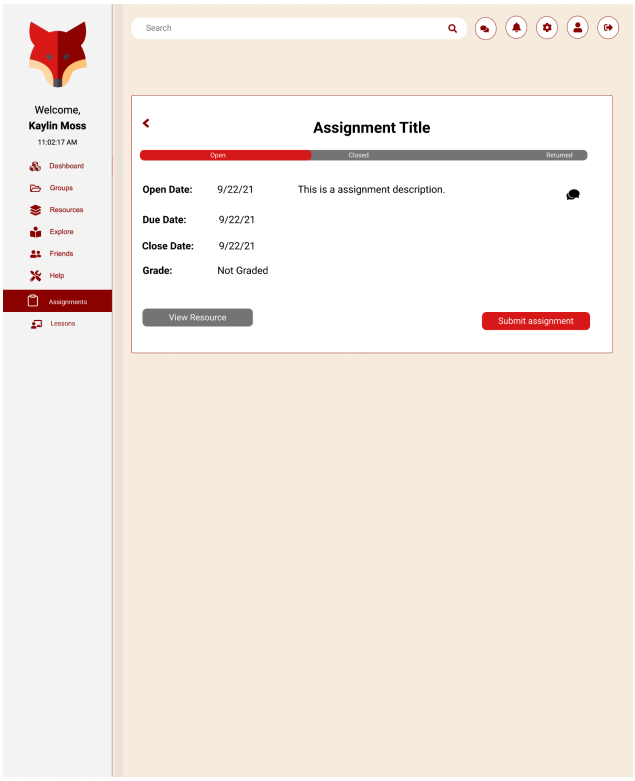
Old Dashboard



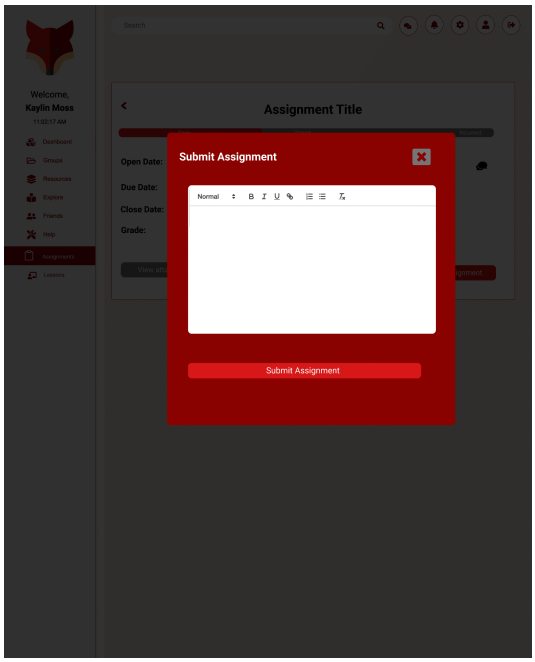
New Dashboard

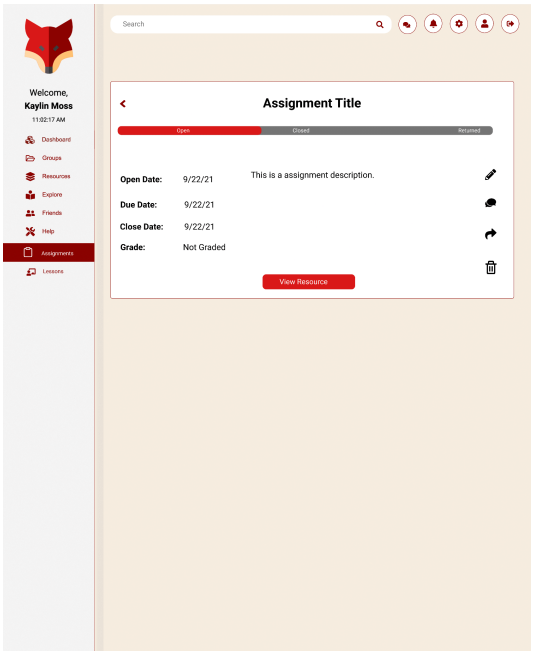


Assignment Page

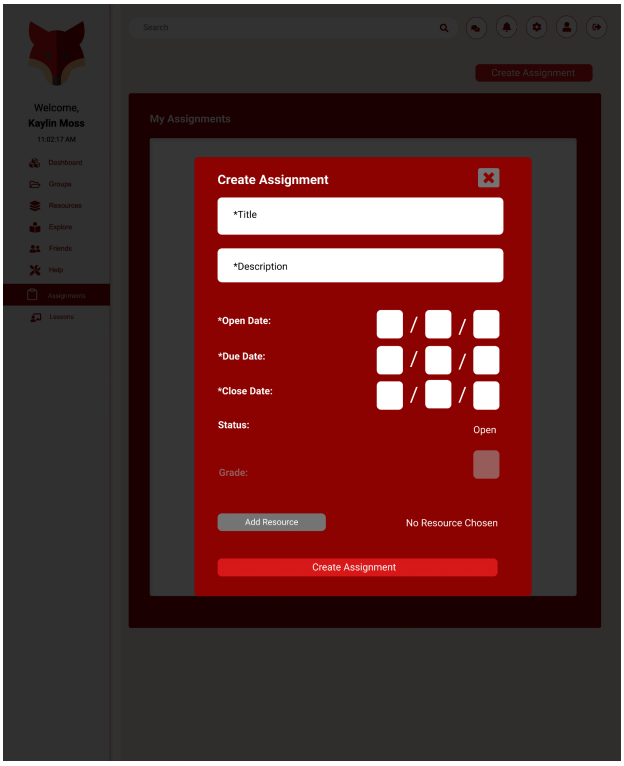


Viewing a Shared Assignment

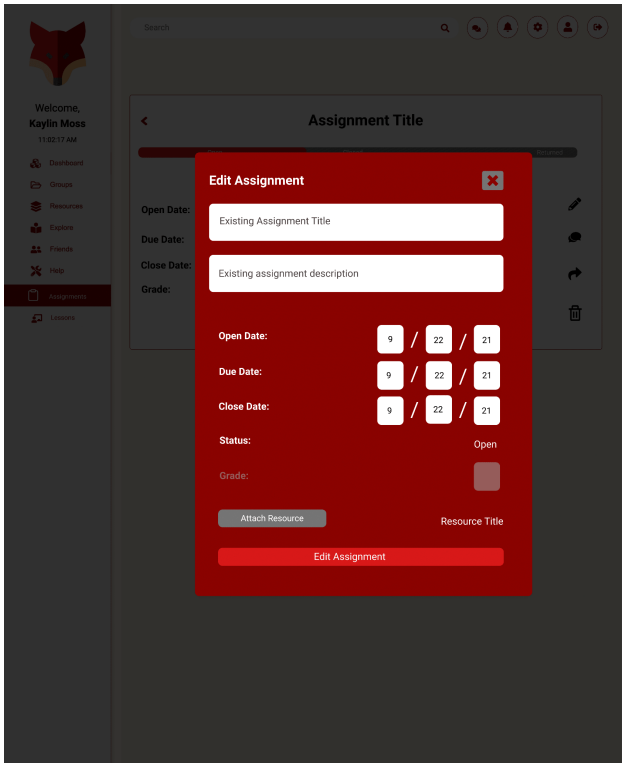




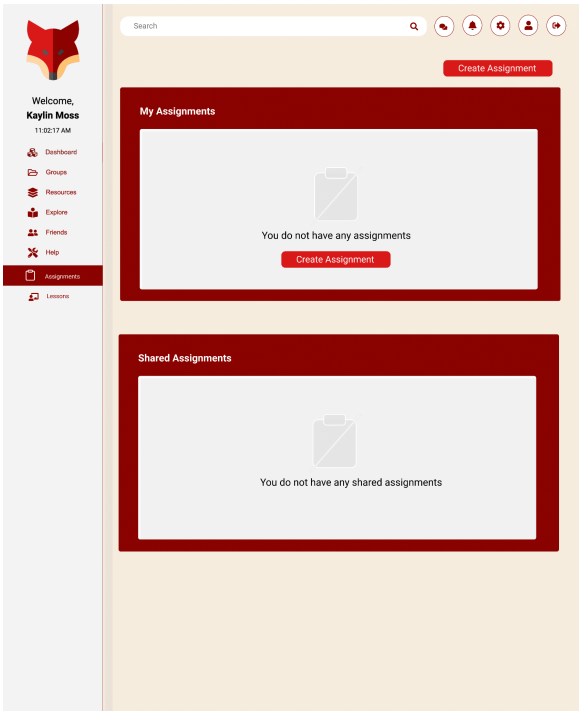
Submit Assignment



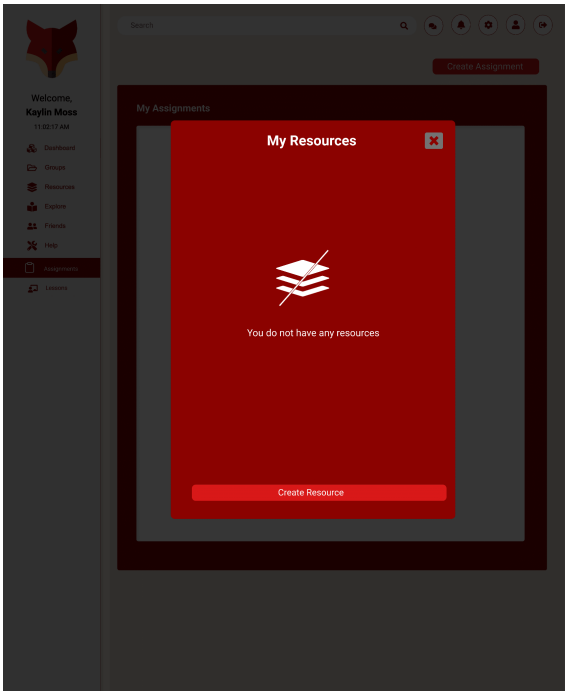
View Assignment



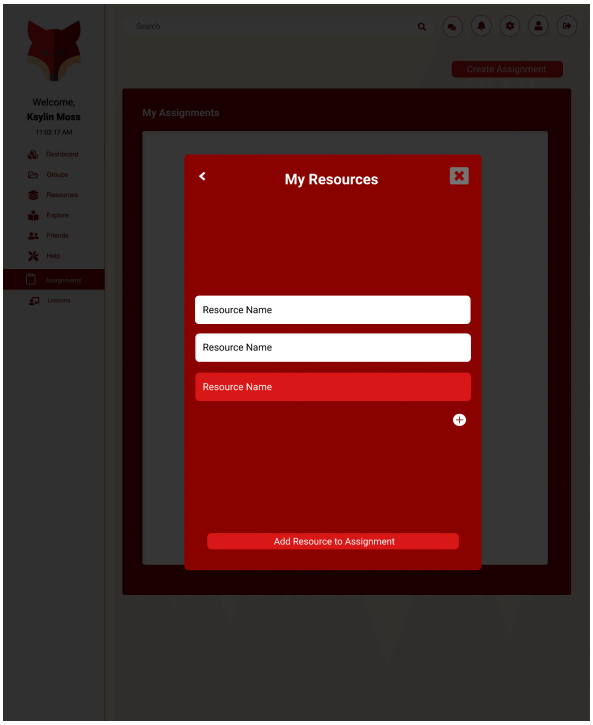
Create Assignment



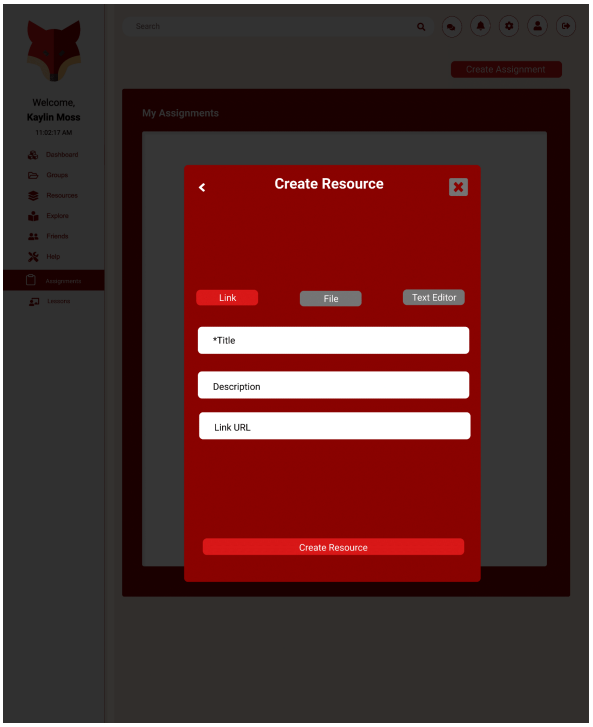
Edit Assignment



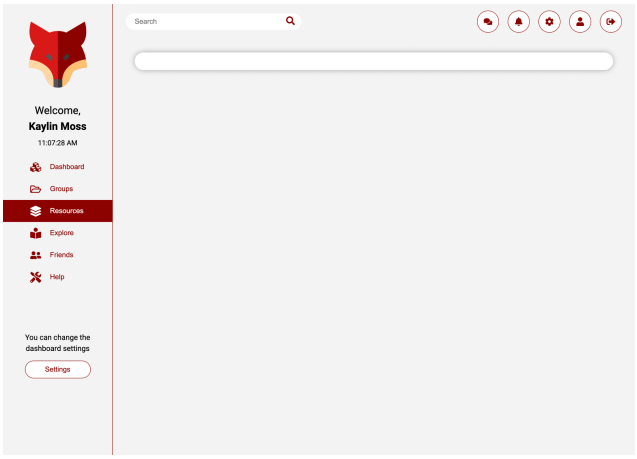
Assignment Page (None)



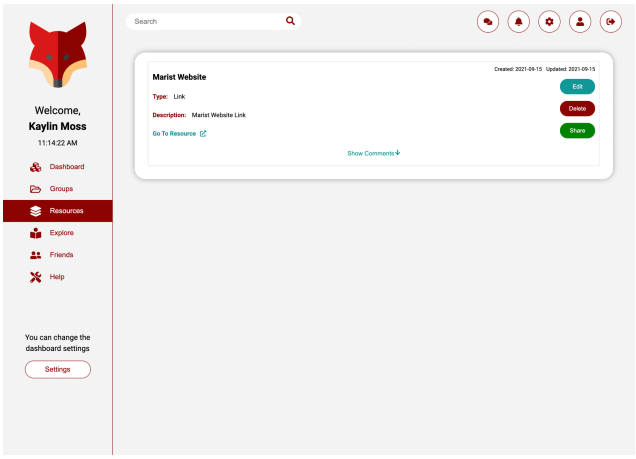
Add a Resource



Adding Your Resource



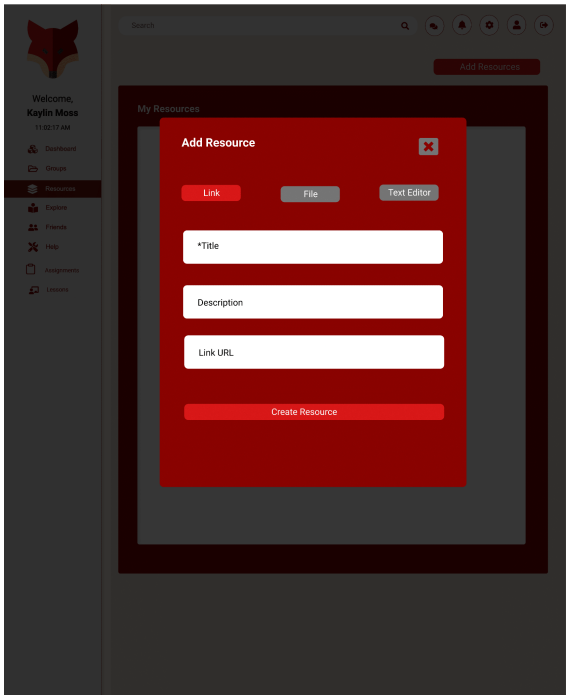
Creating a new Resource



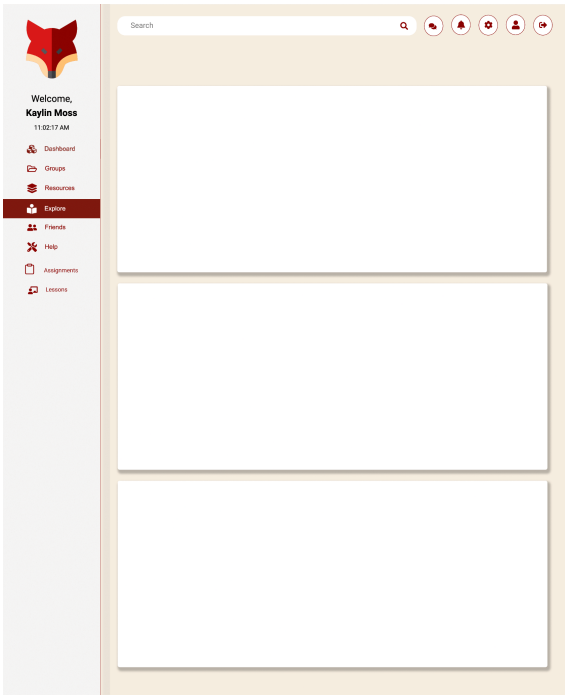
Old Resource Page (None)

Old Resource Page

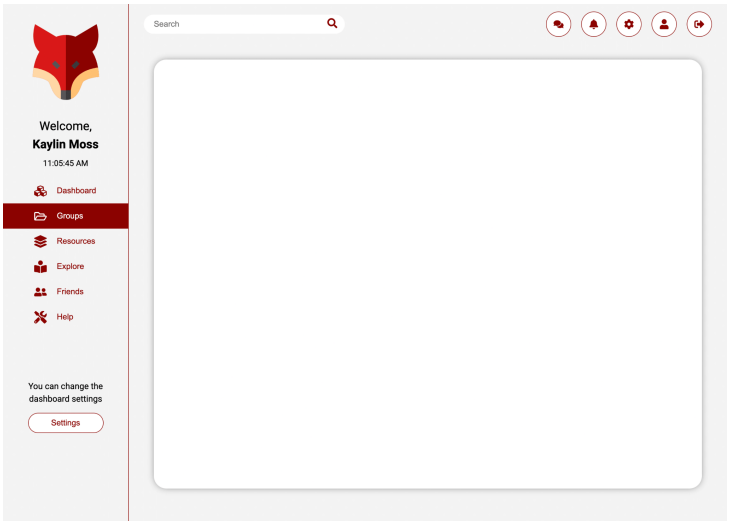
New Resource Page (None)



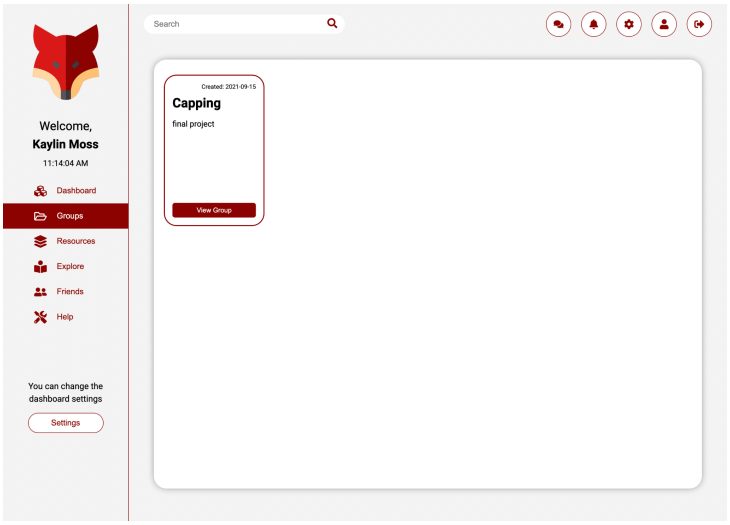
New Resource Page



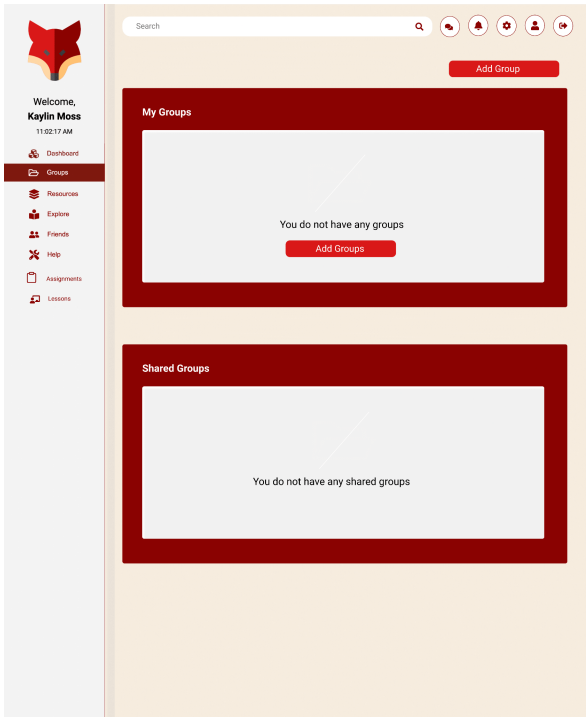
Create Resource Form

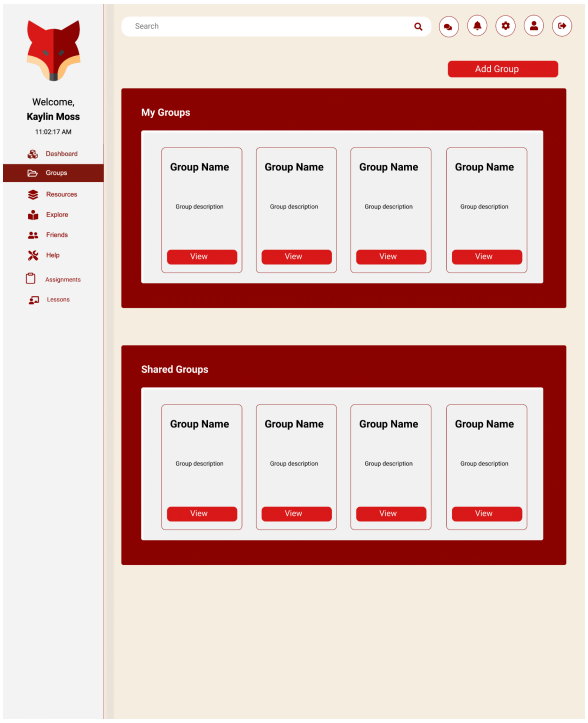


Explore Page

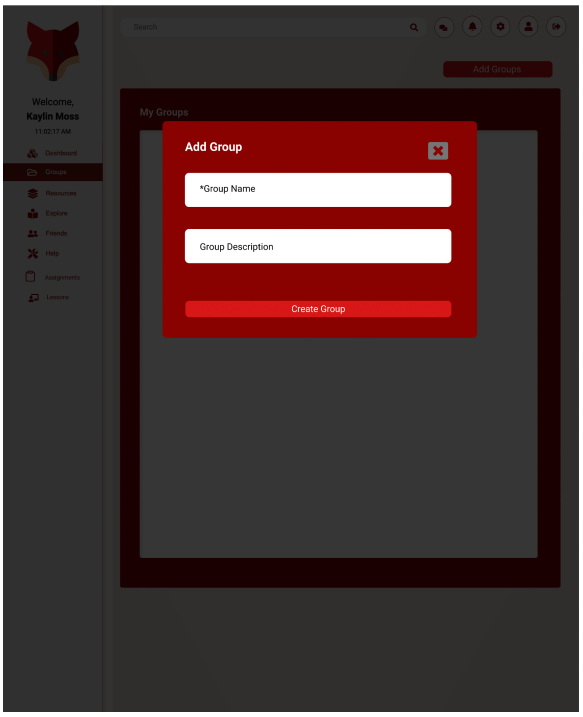


Old Groups Page

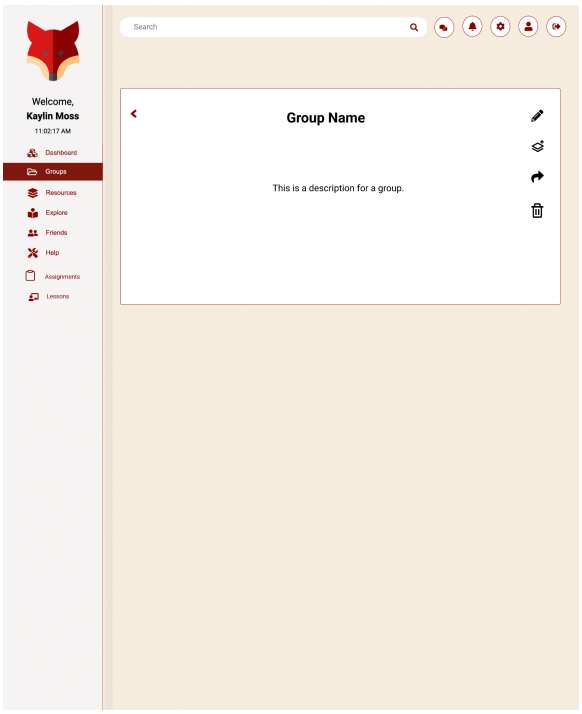




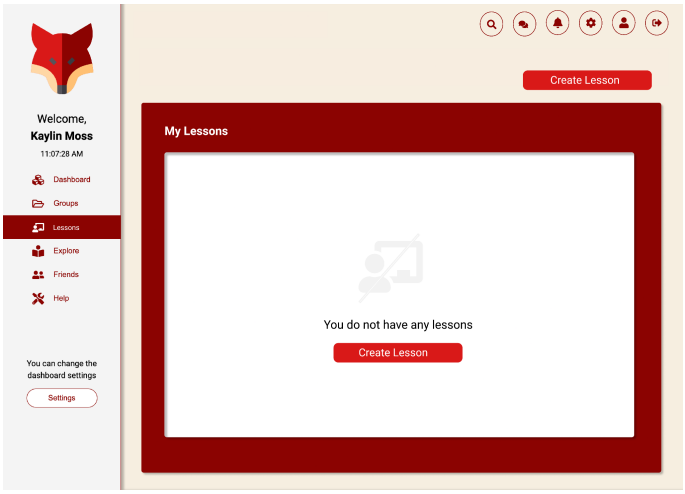
New Groups Page



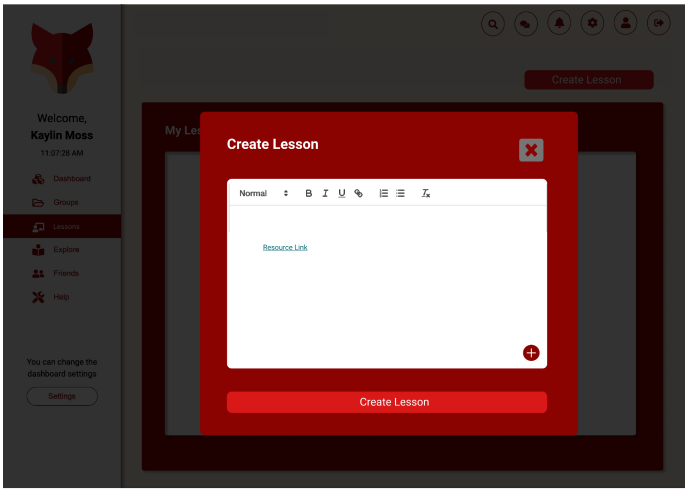
Create Group Form



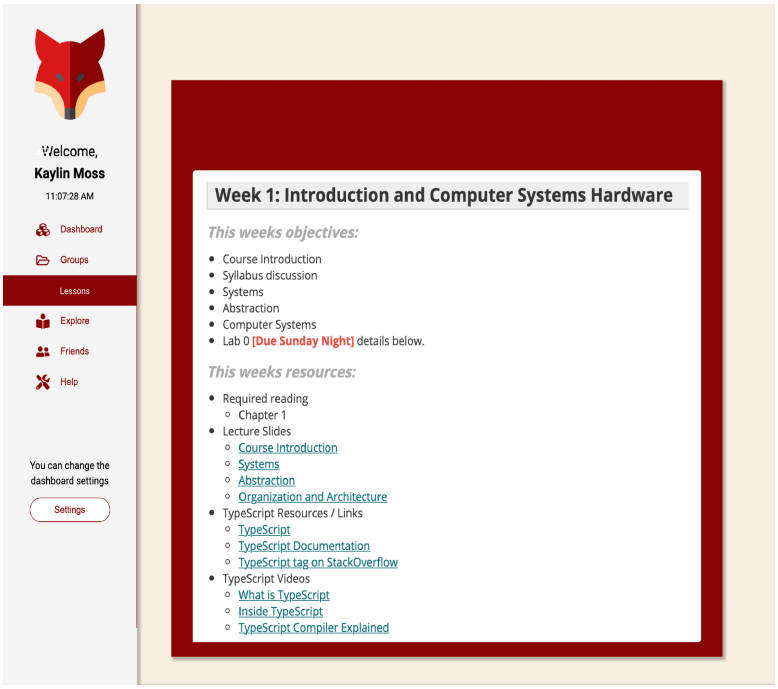
View a Group



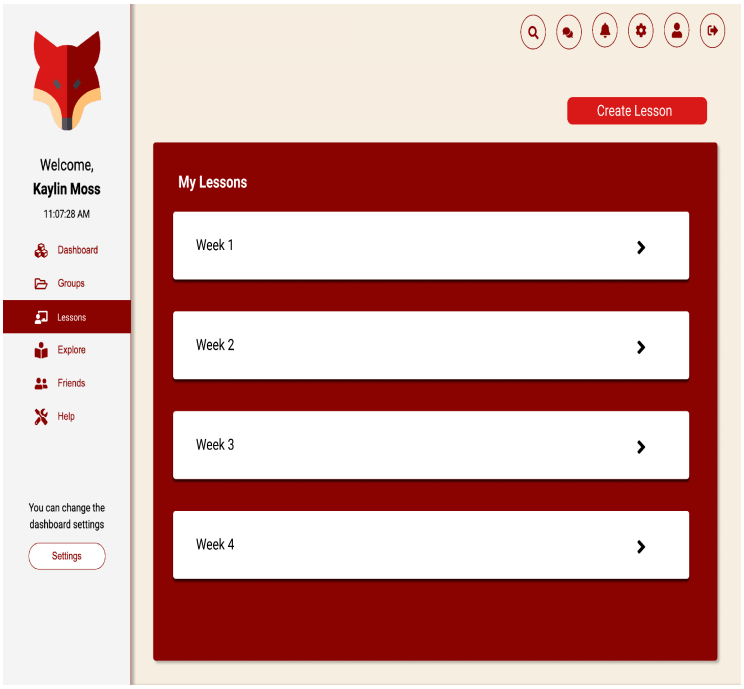
Lesson Page



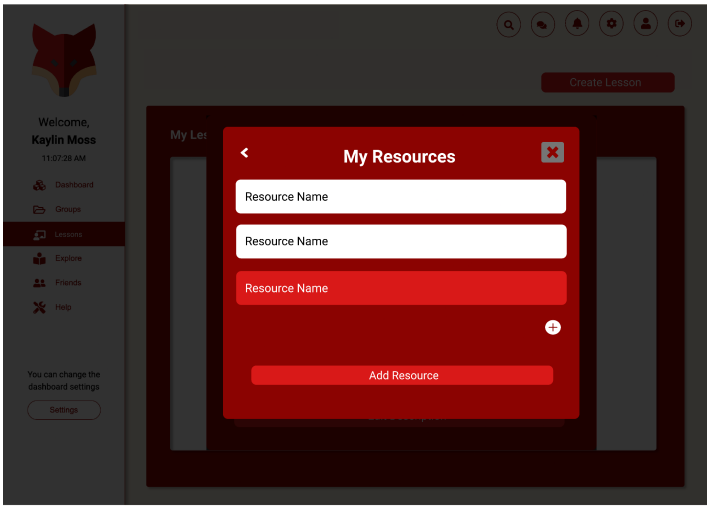
Create Lesson Form



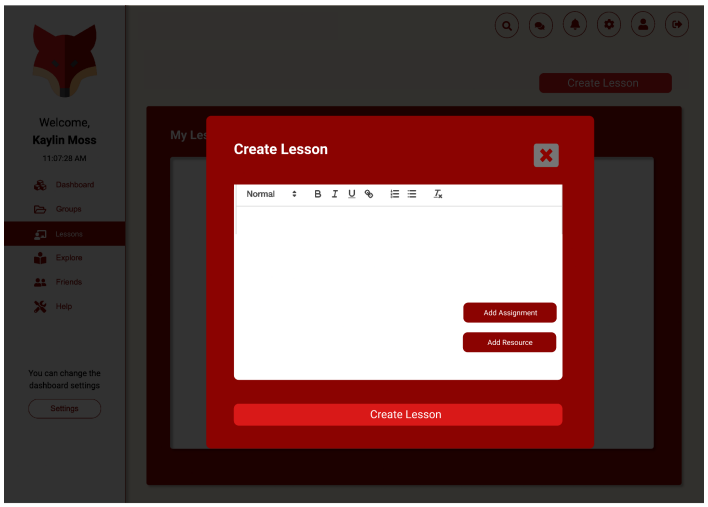
Viewing Lesson



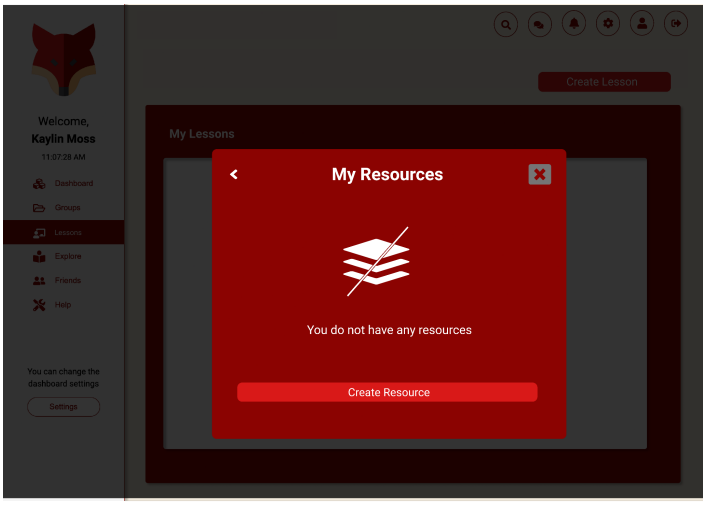
Lesson Page w/ Lessons



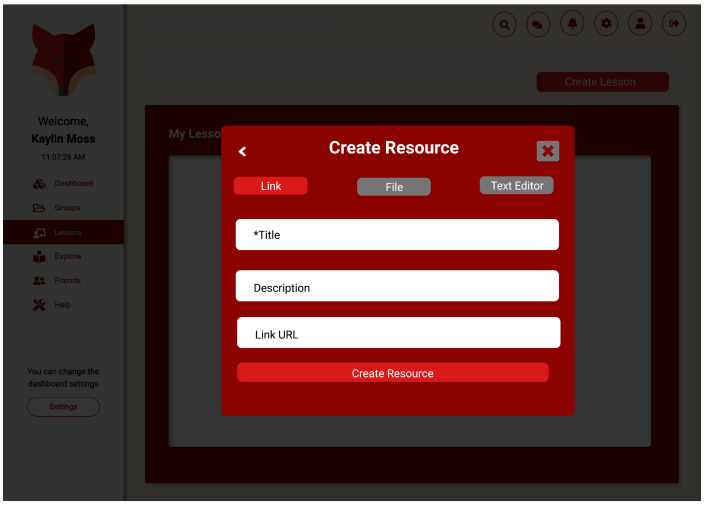
Viewing Resources



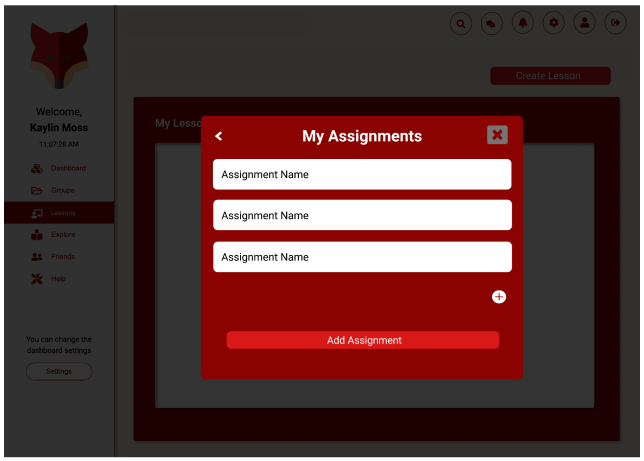
Plus Form



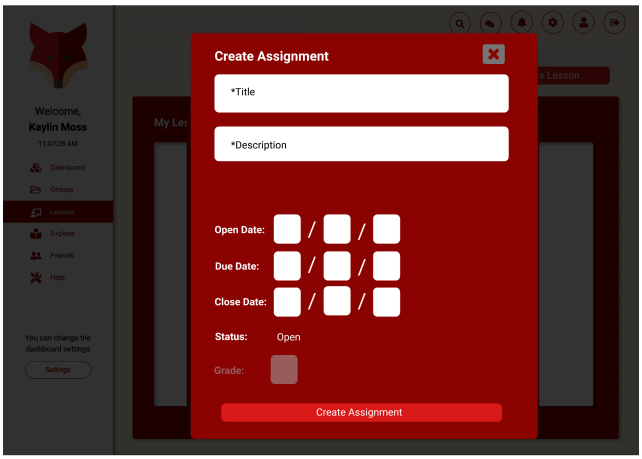
No Resources for Lessons



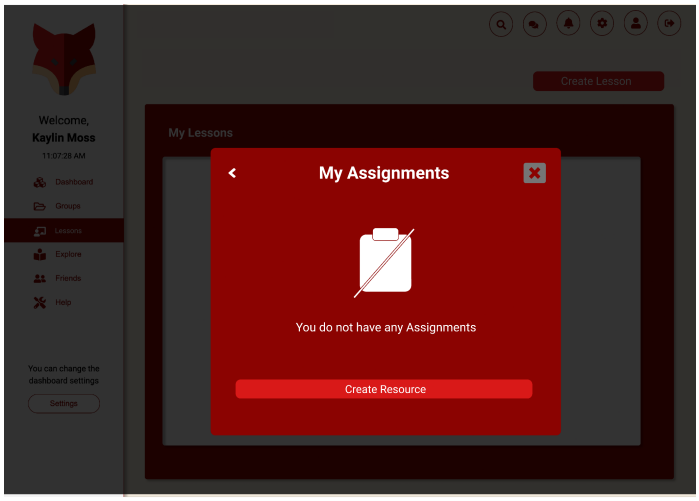
Creating a new Resource



Viewing Assignments



Creating a new Assignment



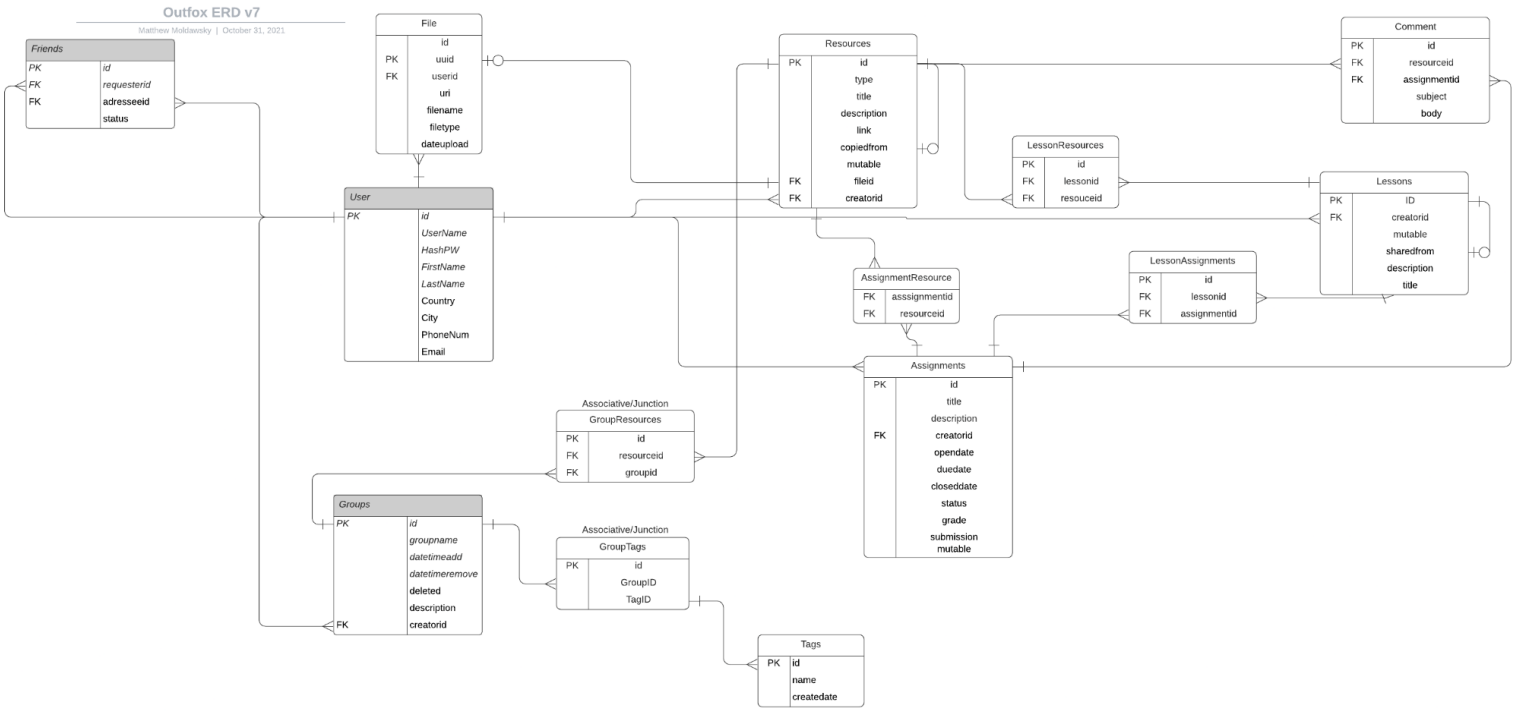
No Assignments shown

Explaining the Flow Behind the Lessons Page:

A user clicks the “Create Lesson” button and the form pops up. This form has a text editor and a plus form for adding Resources and Assignments. Based on which one the user clicks, a form will pop up to show the user’s current resources or assignments. If they do not have a resource or assignment, then they can create one right there. After adding a resource or

assignment to the lesson, the user can then click “Create Lesson” and then their lesson will be saved.

Entity Relationship Diagram:



Link to image for better quality:
https://drive.google.com/file/d/1DQ_tKSJjnDTLXRpACS9-rjWFqKXFt5vF/view?usp=sharing

ERD Documentation:

User:

The User table is for a user that can use our system. The user must have a username and password along with additional personal information for authenticity. This is one of the main tables because the user is who interacts with the system and can make changes to

other aspects of the system. Most of the attributes of the Users table allow the database to store general information about the users such as their name, phone number, email, the city they reside in, the country they reside in, and also credentials such as the username and the hashed password.

Resource:

The Resources table is the table responsible for dealing with all the different types of content that can be added to different user- created groups. Resources can come in a variety of different types such as word documents, photo and video files, pdf files, and more. Most of the attributes of the Resources table allow the database to store general information about resources such as their id, type, title, description, and linkurl. Some of the other attributes in this table are mutable and creatorid which have to do with who created the resource and if it can be changed after its initial creation.

GroupResource:

The GroupResources table is an associative table that connects Resources with the Groups they are a part of. ***NOTE: This is currently not being used to handle adding a Resource to a Group. That is currently handled by Redux. This would be something to ask a future group to work on.*

Group:

The Groups table is responsible for dealing with where different resources and notes are stored for each individual and collection of users. A group is created by one user and can either be a totally private collection of resources for that individual user, or it can be shared with other users which would allow them to view the resources in the group and pull resources into their own groups to edit and create separate resource versions. The 4 attributes of the Groups table all have

to do with identifying each group, general information such as the group name, and what time each group is created and deleted.

Comments:

The Comments table is a table that users are able to write whatever thoughts a user may have about a certain resource. These comments provide an insight into how other uses feel about the resource that was uploaded. Each comment must contain a name so that there is a general understanding of what it contains as well as a body to write out a full complete thought on the resource. **Note: This table could be updated to allow being able to be added to an assignment. Potential task idea for future group*

GroupCategories:

The GroupCategory table is an associative table that connects Groups with the Categories it belongs to.

Categories:

The Categories table is the table that deals with the overall category of a certain resource or the category of a certain group. For example, some categories could be History, Science, Math, Computer Science, etc. This table only has 2 attributes which are an id value associated with each category and the category name which will be preset and readily available on Outfox. Categories all need to be created manually by the developers and there can be an "Other" category allowing users to add resources or groups to categories that aren't official yet.

CategoryTag:

The CategoryTag table is an associative table that connects Tags with Categories in order to help identify certain aspects of a category by giving more information about the category using already created tags.

ResourceTags:

The ResourceTags table is an associative table between the Resources and Tags tables. The primary purpose of this table is to identify the tag that is put on a resource, what resource the tag is put on, an id to identify the tag and when it was made, and who it was made by (linked to UserID).

CommentTags:

The CommentTags is an associative table between the Comments and Tags tables. The primary purpose of this table is to identify the tag that is put on a comment, what comment the tag is put on, an id to identify the CommentTags, when it was made, and who it was made by (linked to UserID).

Tag:

The Tag table is the table that is used to further separate each resource, comment, or category by giving a single word that relates to that certain item. From there, other items that have the same tag can be collected and grouped in order to show similar items to the one that has the tag. The tag must contain a tag name or phrase that can be matched with others of the same name. In addition, a create date must also be included in order to show when and who created the tag for that item.

Friend:

The Friend table is the table that is used to store information about a User's friend. The primary purpose is to handle friend requests. It has two foriegn keys that point to the User who is the requestor and the User who is the addressee. There is also a status field in the table that can be

one of three options: ‘a’, ‘r’, or ‘p’. These values stand for “accepted”, “rejected”, and “pending” respectively. The default value of this field is ‘p’.

AssignmentResource:

The AssignmentResource table is an associative table that connects Resources with the Assignments they are a part of.

Assignment:

The Assignment table is a table that deals with assigning and submitting a user-created Assignment. An Assignment would be created by one user and then assigned to another user through the use of the share functionality. An Assignment can have a resource as an attachment and as a submission. Each Assignment will have an open date, due date, and close date. A user can view the resource attachment of a shared assignment. Each Assignment has a status which can be “open”, “submitted”, and “closed.”

LessonResource:

The LessonResource table is an associative table that connects Resources with the Lessons they are a part of.

LessonAssignment:

The LessonAssignment table is an associative table that connects Assignments with the Lessons they are a part of.

Lessons:

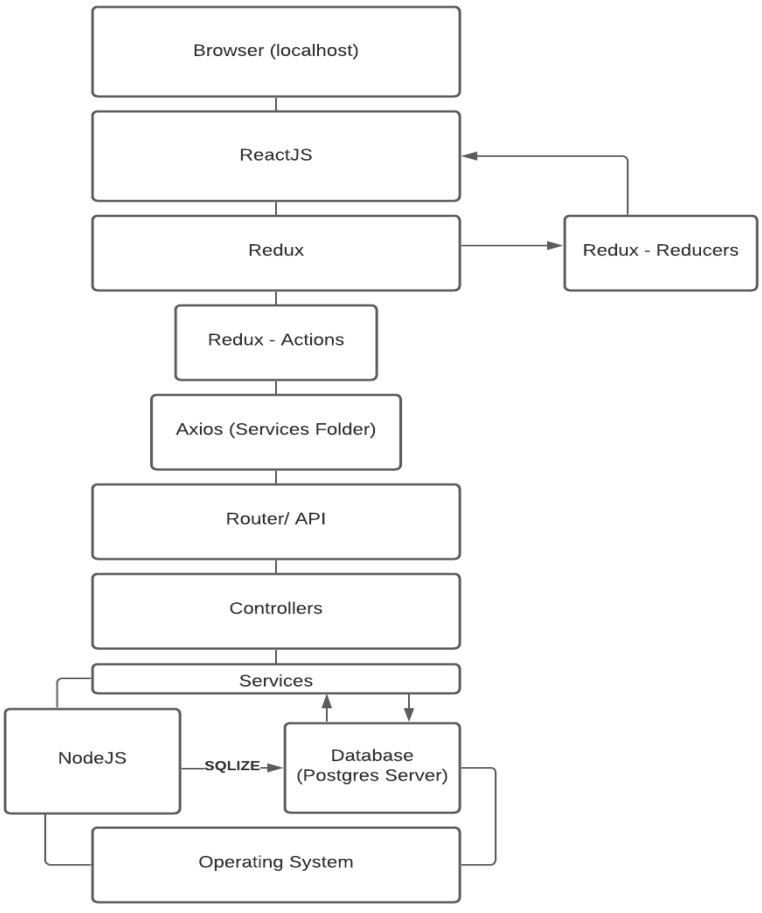
The Lessons table is the table that deals with storing information about user-created lessons. A Lesson has a title and description. The description holds the content created by the text-editor in the creation form. Through the use of the previously mentioned associative tables, a Lesson can

have resources and assignments as attachments. A Lesson can be shared to another user and the recipient will not be able to edit the Lesson.

File:

The File table is the table used to store information about the files uploaded by the User. This model is to be used in place of files being stored using the ResourceController. It will create a unique identifier each time a file is stored. This UUID will then be used when a Resource is created with a file. The file has other fields such as file name, file type, and the URI. *NOTE: This model is currently not implemented in the code of the project even though the table exists in the database. This is an aspect that a future group could implement into the project. The reason for this table is to have more scalability. For instance, you could implement files for Assignments and Lessons instead of Resources.*

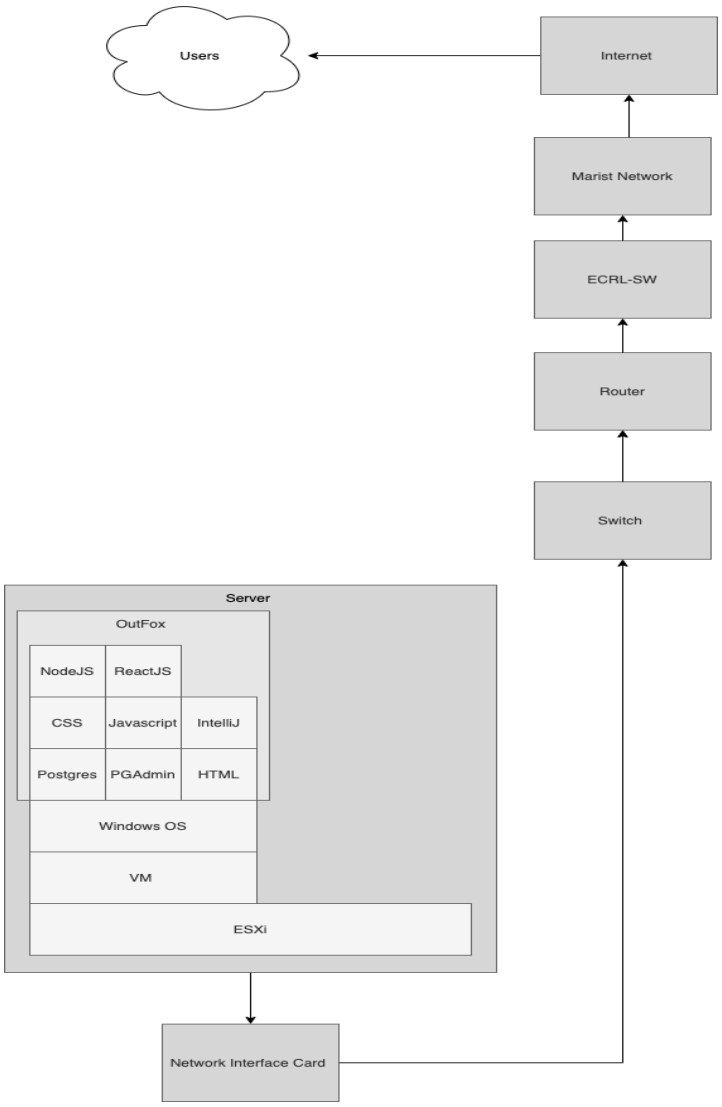
System Architecture Diagram:



*Note: Redux is an add-on to ReactJS that handles front-end storage. It is also used to make API requests to the back-end in conjunction with Axios. Therefore, the flow of an API request would be:

- ReactJS component to Redux action.
- It then uses Redux reducers to store the data in the front end
- It then uses Axios to make the API call (in the services folder under client)
- Once it goes to the controller, it then handles the API request and gives a response based on the result.

Network Architecture Diagram:



Code Documentation:

There are two applications within Outfox as a whole: the front end and the back end. These are two separate applications that interact with each to make Outfox work properly.

The following is the folder structure of the Outfox project:

- **Client:** This folder represents the front end application
    - **Node_modules:** This is the folder that is created when you run ‘npm i’ in the client folder. It holds the data of the dependencies used for the front end.
    - **Public**
    - **Src:** This folder has the files used to build the front end application
        - **Assets:** This folders has images to be used on the front end
        - **Components:** This folder holds each of the ReactJS components used in Outfox.
        - **Pages:** This folder holds each ReactJS page used in Outfox.
        - **Redux:** This folder is used for handling front end storage through Redux and making calls to the Services folder.
            - **Actions:** This folder holds files used to send API requests to services and make calls to redux to store changes in the front end state.
            - **Constants:** This folder holds files used to reduce complexity when making calls to the Reducers in Actions.
            - **Reducers:** This folder holds the Redux reducers which are used for front end storage when changes are made to the database. Changes to the state need to be made so that they can be seen on the front end.

        - **Store.js:** This file builds the redux store.
    - **Services:** This folder holds files that use Axios to make API calls to the backend controllers.
    - **Styles:** This folder contains files that handle styling for the front end.
    - **Index.js:**
- **Node_modules:** This is a folder that is created when you run ‘npm i’ in the root folder of Outfox. It holds data about dependencies used for the entire project.
- **Resources:** This folder is used to store miscellaneous files associated with the Outfox application.
    - Images
    - Promo-text
    - Social Media Artwork
    - Social Media bio
- **Server:** This folder represents the back end of the Outfox application.
    - **Dist:** This folder is a recreation of the src folder writing in Javascript. The files in Src are written in typescript. At runtime, the ‘tsc’ command is used to create this folder.
    - **Node_modules:** This is a folder that is created when you run ‘npm i’ in the server folder. It holds data about dependencies used for the back end of Outfox.
    - **Src:** This folder has the files that build the back end application of Outfox. These files are written in Typescript.
        - **Controllers:** This folder holds the controllers that are responsible for handling the API requests sent to the backend.

- **Interfaces:** This folder holds files that define interfaces used in the rest of the back end.
- **Middleware:** This folder holds files that are used for helping with building the database, connecting to the database, and authenticating a login.
- **Models:** This folder holds files that build out models that are used to create tables in the database through the use of sequelize.
- **Storage:** This folder holds other folders that are used when the user stores a file in a resource.
- **Tools:** This folder holds a file that is used to copy the storage folder over to the Dist folder. This is necessary because the Dist folder is what is used at runtime.
- **App.ts:** This file defines the App class which holds a constructor and other methods used to build the back end application of Outfox.
- **Server.ts:** This file creates a new instance of App and passes in the controllers and the port number to the constructor. It then runs the code 'app.listen' to see if the server is running.

Quill:

There is a text editor, Quill, in the "CreateLesson" component and the "EditLesson" component. In "EditLesson," we were able to set the contents of the quill editor to be the value of the current lesson. However, we were not able to edit the text. An error popped up in the console saying "addrange(): the given range isn't in document." We tried implementing different text editors believing that this issue could just be with Quill itself. For example, we tried Draft.js, upon trying to place it in Create Lesson, since this component already worked, no progress was made.

It appeared as if the editor did not get implemented at all and no text editor window popped up. We also tried react-draft-wysiwyg draft-js, which is a combination of react and draft. Upon running there was a text editor window that popped up; however, it was just the toolbar and there was no box to enter text in.

API Documentation:

- User
- Group
- Resource
- Comments
- Tags
- Friends
- Assignment
- Lesson
- Share

All UPDATE and POST requests require a proper JSON object.

The following GET requests can be changed to GETALL by truncating the ID at the end of the request path.

- User
- Group
- Resource
- Friends
- Comments and ResourceComments
- Assignment
- Lesson

API Requests:

USER:

- Fetching user data
- Fetching users and their groups
- Fetching a user's friends
- Creating user data
- Deleting users
- Updating user data

GROUP:

- Fetching group data
- Creating group data
- Deleting groups
- Updating group data

RESOURCE:

- Fetching resource data
- Creating resource data
- Deleting resources
- Updating resources
- Downloading resources

COMMENTS:

- Fetching comment data
- Fetching resource-specific comments
- Creating comments
- Updating comments
- Deleting comments
- Adding a comment to a specific thread

TAGS:

- Fetching tag data
- Creating tags
- Updating tags
- Deleting tags

FRIENDS:

- Fetching friend data
- Creating friend relationships
- Updating friend relationships
- Deleting a friend relationship
- Fetching pending and accepted friend request data

SHARE:

- Fetching shared group data

- Creating a shared group
- Deleting a shared group
- Fetching shared resource data
- Creating a shared resource
- Creating a shared assignment
- Creating a shared lesson
- Deleting a shared assignment
- Deleting a shared lesson
- Fetching shared assignment data
- Fetching shared lesson data
- Editing a shared assignment (submitting)
- Deleting a shared resource

ASSIGNMENTS:

- Fetching assignment data
- Creating an assignment
- Deleting an assignment
- Editing an assignment

LESSONS:

- Fetching lesson data
- Creating a lesson
- Deleting a lesson
- Editing a lesson

API Details:

USER:

GET User

http://localhost:8080/api/users/{id}

Retrieve a user or users

Success will result in a return of the user information in the PostgreSQL DB; Status: 200

Failure will result in an error, could mean that the user does not exist; Status: 404 not found or 500.

GET UserAndGroups

http://localhost:8080/api/users/userandgroups/{id}

Retrieve a user's information and all the groups, resources, lessons, and assignments associated with them.

Success will return the user's info and info about all of their groups, resources, lessons, and assignments; Status: 200

Failure results in Status 404 Not Found or a 500 error.

GET AllUserFriends

http://localhost:8080/api/users/userfriends/{id}

Retrieve a user's friends

Status 201 for success

Status 400 on failure or 404 for not found

POST User

http://localhost:8080/api/users

Create data for a new user

Success shows the info of the new user; Status: 201

Failure results in an error, possibly a data type exception, Status: 500

DELETE User

http://localhost:8080/api/users/{id}

Deletes a user by ID

Success returns Status: 204, no content, deletes user

Failure results in Status: 404 user not found or Status: 500

PUT - UPDATE User

http://localhost:8080/api/users/{id}

Updates a user's information by ID

Success displays the user info with updated information, Status: 200

Failure results in 404 - User not found or Status: 500 for other errors

GROUP:

GET Group

http://localhost:8080/api/groups/{id}

Retrieve a group or groups

Success will return the group's information; Status: 200

Failure will result in an error; Status: 404 if not found, Status: 500 for other errors

POST Group

http://localhost:8080/api/groups

Create data for a group

Success shows the info for the new group; Status: 200

Failure results in an error, possibly a data type exception; Status: 500

DELETE Group

http://localhost:8080/api/groups/{id}

Deletes a group by ID

Success yields no content; Status: 204

Failure returns Status: 404 group not found or Status: 500 for internal errors

PUT - UPDATE Group

http://localhost:8080/api/groups/{id}

Updates a group by ID

Success returns the group info with updated content; Status: 200

Failure returns Status: 404 - group not found - or Status: 500

RESOURCE:

GET Resource

http://localhost:8080/api/resources http://localhost:8080/api/resources/{id}

Retrieves a resource by ID or all resources without an ID

Success results in Status: 200 and returns the resource(s) info

Failure results in Status: 404 - resource not found or Status: 500

GET DownloadResource

http://localhost:8080/api/resources/download/{id}

Downloads a resource to the user's machine, Specified by ID

API should be moved to the File Controller

Downloads the resource, Status: 200 on success

Status: 500 if there is an error or if the resource is a link; Status: 404 if not found

POST Resource

http://localhost:8080/api/resources

Creates a resource

Success returns the created resource with Status: 201

Failure results in an error, possibly data type exception; Status: 500

DELETE Resource

http://localhost:8080/api/resources/{id}

Deletes a resource by id

Success results in Status: 204, no content

Failure results in Status: 404 not found or Status: 500

PUT - UPDATE Resource

http://localhost:8080/api/resources/{id}

Updates a resource's information by ID

Success results in Status: 200

Failure returns Status: 400 or Status: 404 not found

GET Comment

http://localhost:8080/api/comments/{id}

Retrieves a comment by its ID

Success returns Status: 200

Failure returns Status: 500 or Status: 404 not found

GET ResourceComments

http://localhost:8080/api/resources/resourcecomments/{id}

Retrieves the comments on a resource

Success returns Status: 201 and resource comments

Failure returns an error, Status: 400 or Status: 404 not found

COMMENT:

POST Comment

http://localhost:8080/api/comments

Creates a new comment

Success returns Status: 201

Failure returns Status: 500

DELETE Comment

http://localhost:8080/api/comments/{id}

Deletes a comment by its ID

Status: 204 “Comment successfully deleted”

Status 500 on failure or Status 404 not found

PUT - UPDATE Comment

http://localhost:8080/api/comments/{id}

Modifies one or more comment fields, requires proper JSON

Success returns Status 200 and the updated comment

Failure returns Status 500 or Status 404 if not found

PUT - UPDATE AddCommentToThread

http://localhost:8080/api/comments/addtothread/{id}/{id}

Adds a comment to a thread if sent the proper JSON

Status 201 on success

Status 500 on failure

GET Tag

http://localhost:8080/api/tags http://localhost:8080/api/tags/{id}

Retrieves full list of Tag(s) or one Tag by ID

Status 200 and tag(s) retrieved on success

Status 500 on failure or Status 404 if not found

TAG:

POST Tag

http://localhost:8080/api/tags

Creates a new Tag if sent the proper JSON

Returns Status 201 on success

Returns Status 500 on failure or Status 404 if not found

DELETE Tag

http://localhost:8080/api/tags/{id}

Deletes a tag with the specified ID

Status 204, tag is deleted on success

Status 500 on failure or Status 404 if not found

UPDATE Tag

http://localhost:8080/api/tags/{id}

Updates a tag given an ID and proper JSON

Status 200, tag modified on success

Status 500 on failure or Status 404 if not found

FRIEND:

GET Friend / GET GetAllFriends

http://localhost:8080/api/friends http://localhost:8080/api/friends/{id}

Retrieves a friend list or a friend if given an ID

Status 200 on success

Status 500 on failure or Status 404 if not found

POST Friend

http://localhost:8080/api/friends

Creates a friend relationship with the proper JSON

Status 201 on success

Status 500 on failure

PUT - UPDATE Friend

http://localhost:8080/api/friend/{id}

Modifies information about a friend relationship with proper JSON

Status 200 on success

Status 500 on failure or Status 404 if not found\

DELETE Friend

http://localhost:8080/api/friend/{id}

Deletes a friend relationship by ID

Status 204 no content on success, friend relationship deleted

Status 500 on failure or Status 404 if not found

GET PendingFriendRequests

http://localhost:8080/api/friends/pending/{id}

Retrieves all pending friend requests of a specific user

Status 200 on success and retrieves user's pending requests

Status 500 on failure or Status 404 if not found

GET AcceptedFriendRequests

http://localhost:8080/api/friends/accepted/{id}

Retrieves all accepted friend requests of a specific user

Status 200 on success and retrieves user's accepted requests

Status 500 on failure or Status 404 if not found

SHARE:

GET SharedGroups

http://localhost:8080/api/share/group/{id}

Retrieves the shared groups of a single user by ID

Status 200 and the shared group returned on success

Status 500 on failure or Status 404 if the group is not share with you

POST ShareGroup

http://localhost:8080/api/share/group

Creates a shared group with the proper JSON

Status 200 and the share group info on success

Status 500 on failure

DELETE ShareGroup

http://localhost:8080/api/share/group/{id}

Deletes a shared group by ID

Status 204 on success, the group has been deleted

Status 500 on failure or Status 404 if not found

GET SharedResources

http://localhost:8080/api/share/resource/{id}

Retrieves resources shared with a user by that user's id

Status 200 and shared resource on success

Status 500 on failure or Status 404 if not found

POST SharedResource

http://localhost:8080/api/share/resource

Creates a shared relationship on a resource between two users. Requires proper JSON

Status 201 and the shared resource on success

Status 500 on failure

DELETE SharedResource

http://localhost:8080/api/share/resource/{id}

Deletes a shared resource by ID

Status 404, no content on success

Status 500 on failure or Status 404 if not found

POST SharedAssignment

http://localhost:8080/api/share/assignment

Creates a shared assignment

Status 201 and the share assignment on success

Status 500 on failure

GET SharedAssignment

http://localhost:8080/api/share/assignment/{id}

Retrieves a shared assignment by ID

Status 200 and the share assignment on success

Status 500 on failure or Status 404 if not found

DELETE SharedAssignment

http://localhost:8080/api/share/assignment/{id}

Deletes a shared assignment by ID

Status 204 on success, no content returned

Status 500 on failure or Status 404 if not found

GET SharedLesson

http://localhost:8080/api/share/lesson/{id}

Retrieves a shared lesson by ID

Status 200 and the shared lesson on success

Status 500 on failure or Status 404 if not found

POST SharedLesson

http://localhost:8080/api/share/lesson

Creates a shared lesson

Status 201 and the shared lesson on success

Status 500 on failure

DELETE SharedLesson

http://localhost:8080/api/share/lesson/{id}

Deletes a shared lesson by ID

Status 204 on success, no content returned

Status 500 on failure or Status 404 if not found

ASSIGNMENT:

GET Assignment

http://localhost:8080/api/assignments/{id} http://localhost:8080/api/assignments

Retrieves all or one assignment if given an ID

Status 200 and the assignment(s) on success

Status 500 or Status 404 if not found

POST Assignment

http://localhost:8080/api/assignments

Creates a new assignment

Status 201 and the new assignment on success

Status 500 on failure

DELETE Assignment

http://localhost:8080/api/assignments/{id}

Deletes an assignment given an ID

Status 204 on success, no content returned

Status 500 on failure or Status 404 if not found

PUT - UPDATE Assignment

http://localhost:8080/api/assignments/{id}

Updates an assignment by an ID

Status 200 on success with the updated assignment

Status 500 on failure or Status 404 if not found

*NOTE: There is currently no API that handles adding a Resource to an assignment. This would be handled in this controller. This would be a task for a future group.

LESSONS:

GET Lesson

http://localhost:8080/api/lessons/{id} http://localhost:8080/api/lessons

Retrieves a lesson if given an ID or all lessons

Status 200 on success, returns lesson by ID or all lessons

Status 500 on failure or Status 404 if not found

POST Lesson

http://localhost:8080/api/lessons

Creates a new lesson if given proper JSON

Status 201 on success, returns new lesson

Status 500 on failure

PUT - DELETE Lesson

http://localhost:8080/api/lessons/{id}

Updates a lesson given an ID

Status 200 on success, returns the updated lesson

Status 500 on failure or Status 404 if not found

DELETE Lesson

http://localhost:8080/api/lessons/{id}

Deletes a lesson given an ID

Status 204 on success, no content on return

Status 500 on failure or Status 404 if not found

*NOTE: There is currently no API to handle adding Resources and Assignments to Lessons. This would be something to be included in a future version. This could be a task to be handled by a future group.

Future Update Ideas:

The following are some potential ideas for updates that future groups could implement to flesh out Outfox even more.

Ideas for future groups:

- Implementation of the File model to Resources, could be expanded to include Assignments and Lessons as well
- Implementation of some new API that includes the junction tables for the Many to Many relationships. Example: GroupsResouce and AssignmentResource.
- Fixing issues with Quill or using a different text editor. Check the Quill section for more details.
- Implementation of sharing Lessons and sharing assignments.
- Adding functionality for the top nav bar
    - Profile pages
    - Settings page
    - Message feature
    - Search feature
- Adding existing resources to Group and Assignments

- Fixing all issues on the GitHub page

Helpful Links:

https://github.com/briangormanly/outfox

https://github.com/briangormanly/outfox/wiki

https://github.com/briangormanly/outfox/issues

https://documenter.getpostman.com/view/12985314/TVYDeKfh

https://reactjs.org/docs/getting-started.html

https://redux.js.org/introduction/getting-started

https://nodejs.org/en/docs/

https://quilljs.com/

Known Bugs:

- Some items don't appear until a refresh of the page
- Able to send friend requests to the same person multiple times whether they are your friend or not. The recipient can accept the same request from the same person multiple times.
- URL manipulation security
    - You are able to switch to a different user by changing the user id in the URL.