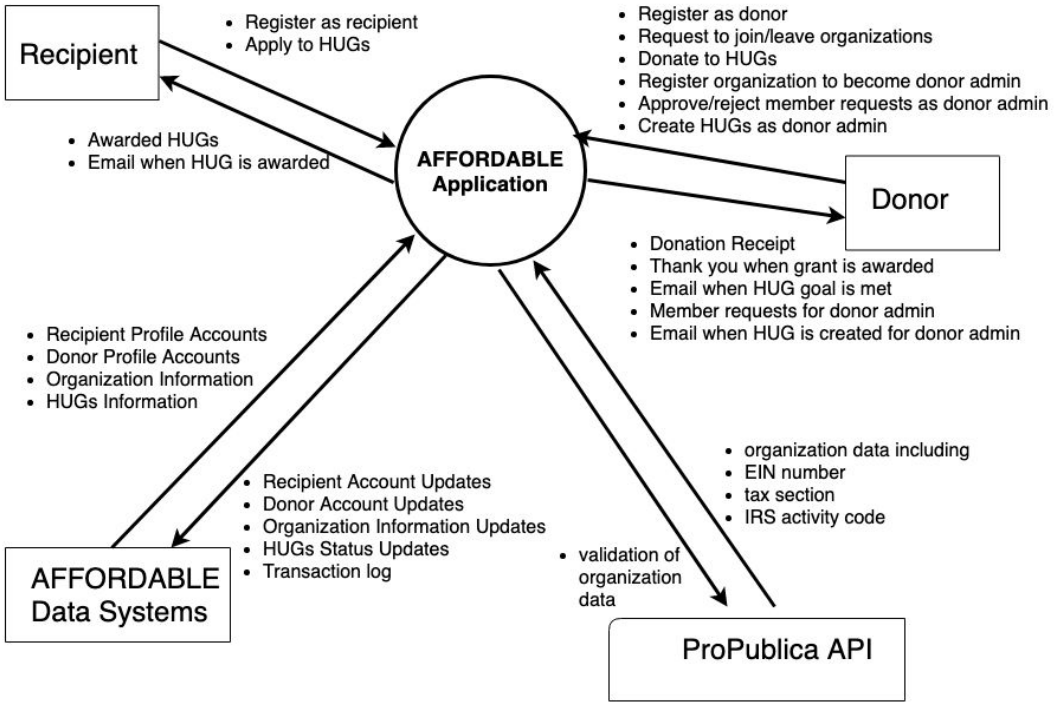
AFFORDABLE
Technical Documentation

Ahmir Robinson, Ariana Marean, Isha Satpalkar, Nikitha Kovvuri
CSE 5911 / Autumn 2019 / Dr. Keith Shafer

Table of Contents

Context Diagram



Actor	Workflow	Data In	Data Out
Recipient	Apply to grants and potentially receive grant	Recipient registration data	Awarded HUG(s), Email when HUG is awarded
Donor (Regular)	Donate to organization grants, join/leave an organization	Donor registration data, bank account information, donation	Donation receipt, Thank you when grant is awarded, Email when HUG goal is met
Donor (Admin)	Approve new organizations, members and HUGs	Authorization data, registered organization data	Approved organizations, member requests and creation of HUGs
Data System	Maintain user data and organization information, manage HUGS	Recipient profile accounts, Donor profile accounts, Organization information, HUGs information	Recipient Account Updates, Donor Account Updates, Organization Information Updates, HUGs Status Updates, Transaction log
ProPublica API	Validate organizations	Organization EIN number, tax section, IRS activity code	Validation of organization data

General Suggestions for Learning Technologies

React

- A large amount of your work will be understanding React
- Much of the backend functionality is tied to the front end working correctly
- Terms you should understand before coding:
    - State: what is it, how do you use it, what are its limitations, how do you update it
    - Props: what is it, how do you use it, what are its limitations, how do you update it
    - Render: what goes in here, what does it mean
    - Component: creating your own, built into react, built into bootstrap, affordable components

Node.js

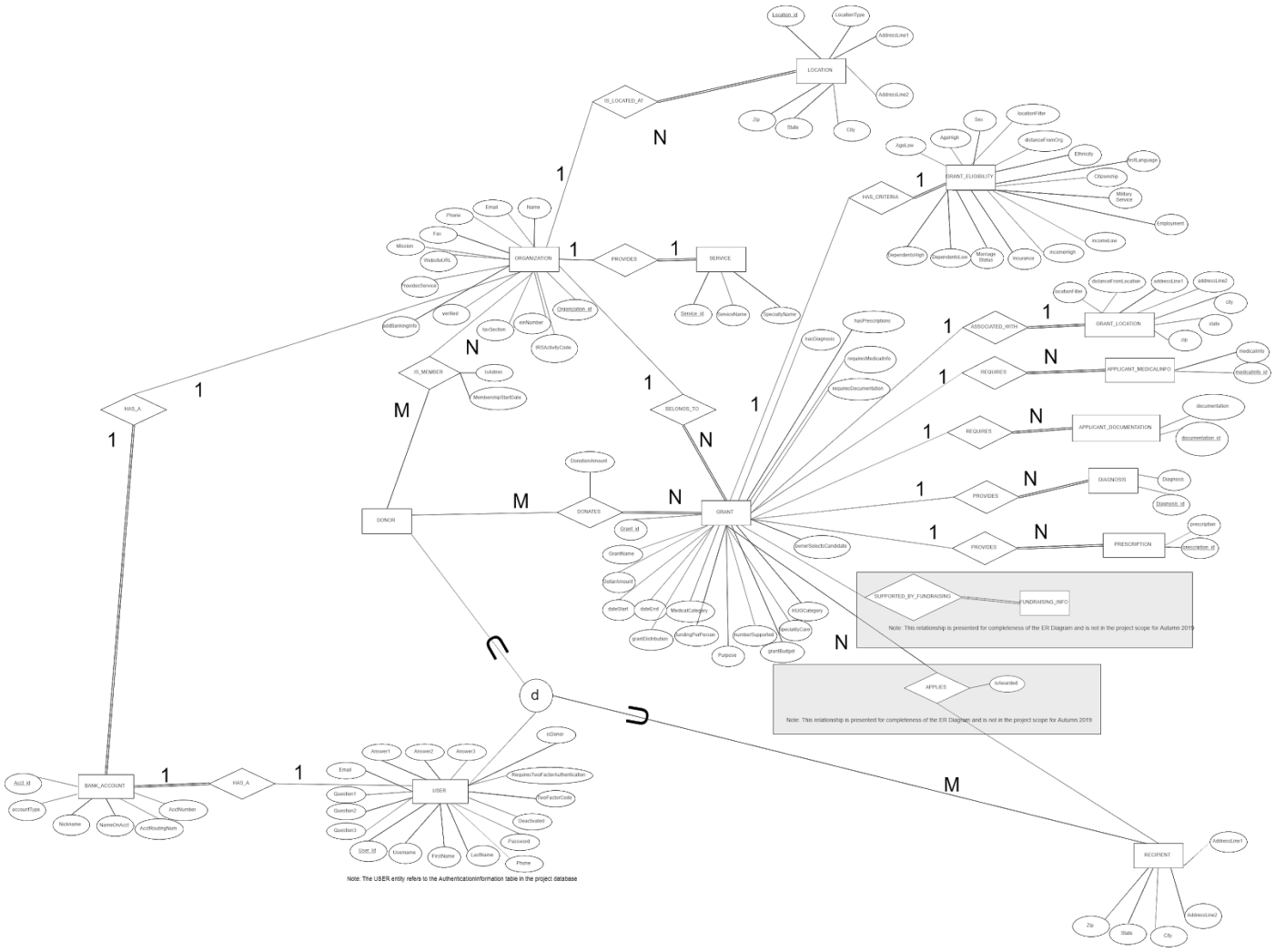
- Uses basic HTTP protocols
    - The syntax for writing methods for each of these have slight nuances you need to pay attention too
    - Reading the network logs in your console
    - Reading the server logs in your terminal
    - Utilizing Postman for efficient testing of REST API calls
- Routing
    - How do you route from your react components to the server files written in Node.js

Creating User Types in Database

ER Diagram and Schema

The ER diagram can be found here:
docs/2019-fall/ER Diagram.pdf or
https://www.x3xoj4kpf0ql1dwzact9uu8v01ic81pd.net:5443/sean/AffordableServerOSU/blob/Trust-1/docs/2019-fall/ER%20Diagram.pdf

Schema: database/CreateDatabase.sql



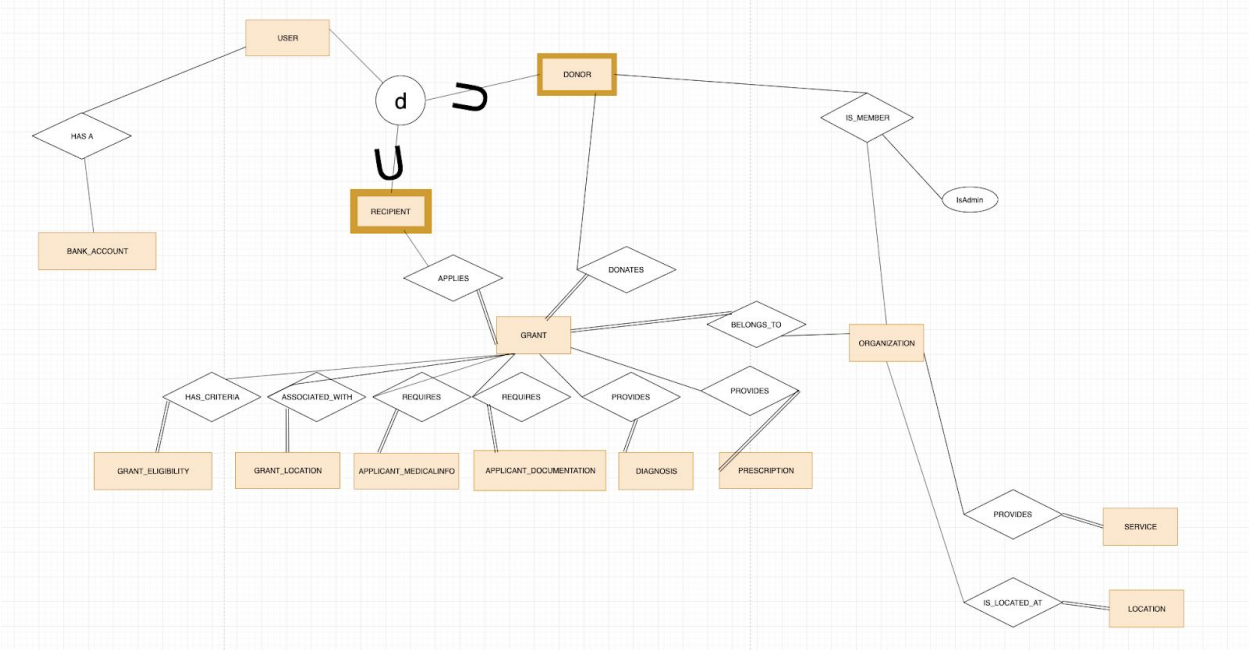
The relationship between HUGs and fundraising(refers to bank section in HUG fundraising page) for HUG Creation is currently not supported in the database.

Integration into legacy database

To maintain the user authentication system that was implemented in previous semesters, the USER entity is defined as a superclass and DONOR and RECIPIENT entities are its subclasses. A USER can be either a DONOR or RECIPIENT, and therefore the subclasses are disjoint.

The USER entity is defined in the schema as a table called `AuthenticationInformation` and contains all attributes related to the login and registration portal.

The ER snapshot can be found here:
docs/2019-fall/ERDiagramSnapshot.png or
https://www.x3xoj4kpf0ql1dwzact9uu8v01ic81pd.net:5443/sean/AffordableServerOSU/blob/Trust-1/docs/2019-fall/ERDiagramSnapshot.png



Creating Different Toolbar for Donors

All logic for the toolbar is done on the front end using Javascript and contexts. The Javascript handles the logic and the contexts hold the data.

Sidebar

Purpose: Donors and applicants will only have options that correspond to their permissions. Donors can also choose which organization they want to represent while navigating the application. All below components are referenced from the sidebar file.

Front End: app/src/pages/MainPagesRoutingContainer/Sidebar.js

Organization Options

Purpose: Allow donors to create, search, join and manage organizations within the Affordable application. All of these options have been implemented. These options are hidden for applicants.

Front End: app/src/pages/MainPagesRoutingContainer/OrganizationOptions.js

Application Options

Purpose: Allow applicants to apply, search, message and manage HUG applications within the Affordable. Our team has not created any of these associated pages. These options are hidden for donors.

Front End: app/src/pages/MainPagesRoutingContainer/ApplicationOptions.js

HUG Options

Purpose: Allow donors to create, search, and manage HUGs within the Affordable. These pages may be extended to serve applicants in the future.

Front End: app/src/pages/MainPagesRoutingContainer/HUGOptions.js

Representing Organization

Purpose: Allow only donor admins to choose which organization they want to complete actions under. This is the organization that will be associated with HUGs created and managed. This feature allows only verified organizations with ProPublica to be a part of the representing organization menu.

Front End: app/src/pages/MainPagesRoutingContainer/RepresentingOrganization.js

Donor Organization Creation

Add Organization

Purpose: A Donor Admin can add their organization with Affordable by completing the Add Organization form registration.

Front End: app/src/pages/MainPages/OrganizationPages/AddOrganization/index.js
Back End: server/routes/addOrganization.js
Testing back end: server/test/addOrganization_Test.test.js
Associated AWS email request: POST addOrg/email/notifyAddOrg

Upon submitting this form, the organization information is stored in the following database tables: `Organization, Services, AccountInfo, OrgMembers, OrgLocations`

POST addOrg/
Sample POST HTTP Request

```
{
    "orgName": "My Organization",
    "email": "user@gmail.com",
    "phone": "123-456-7890",
    "fax": "123-456-7890",
    "WebsiteUrl": "mywebsite.com",
    "userID": "12",
    "Mission": "My mission",
    "ProvidesService": "0"
}
```

POST addOrg/addInfo
Sample POST HTTP Request

```
{
    "orgID": 1,
    "service": "Physician",
    "specialityService": "PrimaryCare",
    "accountType": "Primary",
    "accountNickname": "Savings",
    "accountRouting": 123456789,
    "userID": 12,
    "date": "11-18-2019",
    "addressLine1": "234",
    "addressLine2": "Apt H",
    "city": "Columbus",
```

```
    "State": "Ohio,
    "zip": 43210
}
```

Challenges Faced

- React Dynamic Rendering for Multiple Addresses https://itnext.io/building-a-dynamic-controlled-form-in-react-together-794a44ee552c
- React Conditional Rendering for Services Provided and Bank Account Info https://www.robinwieruch.de/conditional-rendering-react

Verify Organization

PREREQUISITE: Before an organization can be verified with Affordable, it MUST be registered/added to Affordable first.

Purpose: A Donor Admin can verify their organization with Affordable by providing their nonprofit organization details as filed with the IRS by completing the Verify Organization form. This is step 1 of 2 in verifying an organization with Affordable. Step 2 entails Affordable mailing a verification code to the organization for confirmation and was determined Out of Scope for Autumn 2019.

Front End: app/src/pages/MainPages/OrganizationPages/VerifyOrganization/index.js
Back End: server/routes/verifyOrganization.js
Testing back end: server/test/verifyOrganization_Test.test.js
Associated AWS email request: POST verifyOrg/email/notifyVerifyOrg

Technologies Used	
ProPublica API - https://projects.propublica.org/nonprofits/api	ProPublica provides information about nonprofit organizations that have filed tax returns with the IRS.

We are interested in verifying an organization's data based on its EIN so this ProPublica API call was used: `GET /organizations/:ein.json`

Example API call for GREATER WASHINGTON COALITION FOR CANCER SURVIVORS INC with EIN 311591553:
`https://projects.propublica.org/nonprofits/api/v2/organizations/311591553.json`

POST verifyOrg/verifyOrganization
Sample POST HTTP Request

```
{
    "orgName": "Mountain Ambulance and Health Care Foundation",
```

```
    "addressLine1": "PO BOX 2491",
    "addressLine2": "",
    "city": "Banner Elk",
    "state": "NC",
    "zip": "28604",
    "ein": "311682791",
    "taxSection": "3",
    "irsActivityCode": "E62"
}
```

Upon successful submission of this form, the organization information is stored in the following database tables: `Organization, OrgLocations`

Possible HTTP Responses	
`{ "status": "OK", "valid": "true" }`	Means that organization exists in Affordable database, organization's verification data matches ProPublica exactly, and has been successfully verified with Affordable. `OrgLocations` table will have verified address labeled as a "Match" that an admin may review.
`{ "status": "OK", "valid": "false" }`	Means that organization exists in Affordable database, organization's verification data does NOT match ProPublica exactly and is not verified with Affordable. `OrgLocations` table will have address labeled as "Different" that an admin may review. This label means that some aspect of the verification submission is inconsistent with ProPublica.
`{ "status": "NOT OK", "valid": "false" }`	Means that organization does NOT exist in Affordable, and therefore cannot be verified. A prerequisite to verifying an organization is to add the organization with Affordable first.
`{ "status": "404 Not Found", "valid": "false" }`	Means that the EIN that was entered does not exist with the ProPublica API, and therefore cannot be verified.

Representing Verified Organization

PREREQUISITE: The intended organization to represent must be added to the database and verified with ProPublica.

Purpose: A Donor Admin can represent their verified organization with Affordable by selecting it from the representing organization dropdown in the sidebar.

Front End:
app/src/pages/MainPages/MainPagesRoutingContainer/RepresentingOrganization/index.js
Back End: server/routes/representOrganization.js
Testing front end: app/src/pages/MainPages/MainPagesRoutingContainer/index.spec.js

Challenges Faced:

1. The orgID is expected to be pulled from the current organization that the Donor is representing and passed to the various sidebar options. This value is set in the representing organization component, but needs to be passed to the different pages through the sidebar component.
    a. Solution: Use contexts and callback functions to pass the representing orgID from representing organization file to sidebar component.
        i. Contexts require that the provider be a parent component - so we used callbacks(repOrgIdFunction) to pass the orgID back to the Main Pages Routing Container (index.js) and then created the context here (RepresentingOrgIDContext)

Donor Organization Membership

Join Group

Purpose: This page includes three sections.

1. Searching & Joining - A search bar and the corresponding results. The user can select from the results the organization they wish to join.
2. Membership Requests - A list of organizations that the user has requested to join, but has not been approved for yet.

Front End: app/src/pages/MainPages/OrganizationPages/JoinGroup/index.js
Back End: server/routes/joinGroup.js
Testing back end: server/test/joinGroup.test.js

Search Results Section - Joining a new group

PREREQUISITE: Before a donor can join an organization, the organization must be verified with Affordable, it MUST be registered/added to Affordable first.

Purpose: A Donor Admin can join an existing organization with Affordable by searching the existing organization database.

POST joinGroup/searchOrg

Sample POST HTTP Request

```
{
    "orgName": "New Organization",
}
```

POST joinGroup/addMember

Sample POST HTTP Request

```
{
    "orgID": "1",
    "userID": "23
}
```

Membership Requests - Outstanding Requests for Membership

PREREQUISITE: A donor has applied to an organization and has not been approved or rejected.

Purpose: A donor can join an organization so that they may be an affiliate of the organization and contribute to that organization's HUGs.

POST joinGroup/addMember
Sample POST HTTP Request

```
{
    "orgID": "1",
    "userID": "23
}
```

Affiliation Section - Viewing Current Membership

PREREQUISITE: A donor must have applied and been approved to be listed as an affiliate of the organization.

Purpose: A Donor Admin can view the organizations where they are registered members

POST joinGroup/affiliations
Sample POST HTTP Request

```
{
    "orgID": "1"
}
```

HUG Creation

Create

Purpose: This page includes four sections for creating a valid HUG. They are modeled as five separate react components.

1. HUG Creation - this is the parent component that makes calls to generate each of the page components separately, all backend calls are made from this component
2. HUG Purpose - basic details about the HUG
3. HUG Eligibility - requirements for applicants to qualify for the HUG
4. HUG Information - information to be requested from applicants
5. HUG Fundraising - how funds will be distributed and payment information

HUG Purpose

Front End: app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/purpose.js

Front End Diagnosis Autosuggest Component:
app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/DiagnosisAutosuggest.js
Front End Diagnosis Data:
app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/diagnosis.js

Front End Prescription Autosuggest Component:
app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/PrescriptionAutosuggest.js
Front End Prescription Data:
app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/prescription.js

Testing front end:
app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/index.spec.js

Technologies Used	
React Autosuggest	A standard React component that allows a user to select a suggested input field value based on the text they enter. This component was used for specifying a diagnosis and prescription for HUG Creation. http://react-autosuggest.js.org/

No HTTP calls to the backend are made on completion of the page.
Associated AWS email request: POST hug/email/notifyHugCreation

HUG Eligibility

Front End: app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/eligibility.js
Testing front end:
app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/index.spec.js

No HTTP calls to the backend are made on completion of the page.

HUG Information

Front End: app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/information.js
Testing front end:
app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/index.spec.js

No HTTP calls to the backend are made on completion of the page.

HUG Fundraising

Front End: app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/fundraising.js
Testing front end:
app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/index.spec.js

No HTTP calls to the backend are made on completion of the page.

HUG Creation

Front End: app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/hugcreation.js
Back End: server/routes/addHUG.js
Testing back end: server/test/addHUG.test.js
Testing front end:
app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/index.spec.js

When a new HUG is added, a series of backend calls are made:

POST addHUG/
Sample POST HTTP Request

```
{
    "hugName": "Mothers helping mothers",
    "hugCat": "Prescriptions",
    "medCat": "Prenatal Care",
```

```
    “specialtyCare”: null,
    “Purpose”: “Help mothers get vitamins”,
    “hasPrescriptions”: “0”,
    “hasDiagnosis”: “0”,
    “dateStart”: “10/10/2019”,
    “dateEnd”: “03/04/2020”,
    “grantDistribution”: “Monthly”,
    “fundingPerPerson”: “100”,
    “grantBudget”: “1000”,
    “numberSupported”: “10”,
    "orgID": "1"
}
```

POST info/grantID

***this backend call is made to get the id of the grant just created so that it can be used as a foreign key for the following insertions**

Sample POST HTTP Request

```
{
     “hugName”: “Mothers helping mothers”,
    “hugCat”: “Prescriptions”,
    “medCat”: “Prenatal Care”,
    “specialtyCare”: null,
    “Purpose”: “Help mothers get vitamins”,
    “hasPrescriptions”: “0”,
    “hasDiagnosis”: “0”,
    “dateStart”: “10/10/2019”,
    “dateEnd”: “03/04/2020”,
    “grantDistribution”: “Monthly”,
    “fundingPerPerson”: “100”,
    “grantBudget”: “1000”,
    “numberSupported”: “10”,
    "orgID": "1"
}
```

POST addHUG/addEligibility

Sample POST HTTP Request

```
{
    “age”: “30”,
    “sex”: “female”,
    “location”: “State”,
    “distance”: “50”,
    “grantID”: “2”
```

```
    "orgID": "1"
}
```

POST addHUG/addLocationInfo

Sample POST HTTP Request

```
{
    “locationFilter”: “Distance from Location”,
    “distanceFromLocation”: “100 miles”,
    “addressLine1”: “132 Buckeye Lane”,
    “addressLine2”: “”,
    “city”: “Columbus”,
    “state”: “Ohio”,
    “zip”: “43210”,
    “grantID”: “3”
}
```

These following calls are optional and based on user input

POST addHUG/addDiagnosis

Sample POST HTTP Request

```
{
    “diagnosis”: “Pregnancy”,
    “grantID”: “3”
}
```

POST addHUG/addPrescription

Sample POST HTTP Request

```
{
    “prescription”: “Vitamins Prenatal with Zinc”,
    “grantID”: “3”
}
```

POST addHUG/addMedicalInfo

Sample POST HTTP Request

```
{
    “medicalInfo”: “Medical History”,
    “grantID”: “3”
}
```

POST addHUG/addDocumentation

Sample POST HTTP Request

```
{
    “documentation”: “Passport”,
    “grantID”: “3”
```

```
}
```

Challenges Faced

2. The orgID is expected to be pulled from the current organization that the Donor is representing. This value is set in the sidebar component, but needs to be passed to the hug creation component.
    a. Solution: Use contexts and callback functions to pass the orgID from the sidebar to hug creation.
        i. Contexts require that the provider be a parent component - so we used callbacks(repOrgIdFunction) to pass the orgID back to the Main Pages Routing Container (index.js) and then created the context here (RepresentingOrgIDContext)
3. Storing each page of the form as a separate page and route. This resulted in the data being broken up and difficult to handle.
    a. Solution: Deviate from current file pattern and store all the pages as components under the same route and folder.

HUG Search

Purpose: The HUG search enables donors and users to see all existing HUGs in Affordable. In addition, we enabled the donate feature which is accessed through the donate button on the search card results.

Front End: app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/HUGSearch.js
Back End: server/routes/search.js
Testing back end: server/test/search_Test.test.js
Testing front end:
app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/index.spec.js

Searching

POST search/grants
Sample POST HTTP Request

```
{
    "grantName": "Ohio State Grant"
}
```

POST search/eligibility
Sample POST HTTP Request

```
{
    "grantIDs": "[1000,1001]"
}
```

POST search/location
Sample POST HTTP Request

```
{
    "grantIDs": "[1000,1001]"
}
```

POST search/hugOrg
Sample POST HTTP Request

```
{
    "orgID": "1001"
}
```

Donating

Associated AWS Email Request: POST hug/email/notifyHugFunding

POST search/donate *used when a donor is donating to a HUG for the first time
Sample POST HTTP Request

```
{
    "donateAmount": "20",
    "grantID": "3",
    "donorID": "5"
}
```

POST search/donateAgain *used when a donor is donating to a HUG repeatedly
Sample POST HTTP Request

```
{
    "donateAmount": "20",
    "grantID": "3",
    "donorID": "5"
}
```

GET search/fundingStatus?grantID
Sample GET HTTP Request
search/fundingStatus?grantID=3

GET search/donationStatus?grantID&donorID
Sample GET HTTP Request
search/fundingStatus?grantID=3&donorID=2

Technologies Used	
React Popup	A simple React popup component that is easy to use. This component was used to construct the modal for donating to a HUG. https://react-popup.elazizi.com/react-modal/
React Progressbar	A standard React component used to provide up-to-date feedback on the progress of a workflow or action with a simple yet flexible progress bar. This component was used to show the progress of funding for a HUG. https://react-bootstrap.github.io/components/progress/

React Paginate	This is not a standard React component and must be installed as an additional library. It is encompassed in the npm install or yarn command. The component handles the functionality of paging through the results and the bar at the bottom displaying the number of pages. It does **not** generate the frontend results themselves.
Card from React-Bootstrap	Bootstrap's cards provide a flexible and extensible content container with multiple variants and options. These cards were used in the HUG Search and HUG Manage page to display the results in a simple container. https://react-bootstrap.github.io/components/cards/

Challenges Faced

1. Handling someone donating to the same HUG more than once
    a. Solution: Update the table entry in the Donates table with the total amount they donated to the HUG
    b. Note: This means there is no way to currently see their separate donation amounts - we are only storing the total amount they have donated to each HUG

HUG Management

Purpose: Allow donors to see the progress of HUGs that they are admins for already. Current data displayed includes the status of goal funding and the number of users who have donated.

Front End: app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/manage.js
Back End: server/routes/manage.js
Testing back end: server/test/manage.test.js
Testing front end: app/src/pages/HUGPages/HUGPagesRoutingContainer/index.spec.js

Manage

POST manage/managegrants
Sample POST HTTP Request

```
{
    "orgID":"100"
}
```

GET /manage/managedonates?grantID
Sample GET HTTP Request
/manage/managedonates?grantID=1001

Note: Backend calls from search were reused on this page to get information for location, eligibility and funding status

Technologies Used	
React Progressbar	A standard React component used to provide up-to-date feedback on the progress of a workflow or action with a simple yet flexible progress bar. This component was used to show the progress of funding for a HUG. https://react-bootstrap.github.io/components/progress/
React Paginate	This is not a standard React component and must be installed as an additional library. It is encompassed in the npm install or yarn command. The component handles the functionality of paging through the results and the bar at the bottom displaying the number

	of pages. It does **not** generate the frontend results themselves.
Card from React-Bootstrap	Bootstrap's cards provide a flexible and extensible content container with multiple variants and options. These cards were used in the HUG Search and HUG Manage page to display the results in a simple container. https://react-bootstrap.github.io/components/cards/

Challenges Faced:

1. Whether to count someone donating multiple times as multiple people.
    a. Solution: Count each person once no matter their amount of donations to the HUG

AWS Email Notifications

Technologies Used	
Beefree.io	Free and easy-to-use drag/drop tool to create HTML templates
Nodemailer and Nodemailer-SES-Transport	Easy-to-use Node package that is a wrapper around aws.SES from the AWS SDK package Involves initializing an SES Transport object with AWS access key and secret keys
AWS Simple Email Service	Email platform used to send informational emails using AffordHealth's domain

Secure Use of AWS Credentials with Gitlab
Note: A user called affordable_emailer has already been created and its access key ID and secret access key are used for all email notifications. This user only has permissions for AWS Simple Email Service. It is STRONGLY recommended to update this user's access key and secret access key regularly for security reasons.
AWS hackrs

User: affordable_emailer
Password: TeamTrust1

Using an admin account, to create a new user with only SES permissions:

1. On the AWS Console, navigate to:
   User Account dropdown → My Security Credentials → Users → Add User
2. Create a new user with Programmatic Access and AWS Management Console Access. Then, set permissions to ONLY have Full SES Access.
3. Upon user creation, save the access key ID and secret access key to use in the project.

Sample format to customize emails:

```
const mailOpt = {
 from: 'donotreply@affordhealth.org',
 to: user@gmail.com,
 subject: 'Affordable:: About Your HUG',
 html: utils.formatHugFundingEmail(orgName, hugName),
 attachments:[{
   filename : 'EmailsLogo.png',
   path: '../app/src/assets/images/EmailsLogo.png',
   cid : 'HugFundingEmailsLogo'
 }]

};
```

Note: Images in the AWS emails cannot be directly coded in the email template html. The image is passed to the template using attachments as shown in the sample format

Add Organization

Purpose: Notify Donor Admin by email when their organization has been added with Affordable.

Front End: app/src/pages/MainPages/OrganizationPages/AddOrganization/index.js
Email Template: server/utils.js `module.exports.formatAddOrgEmail` function
Back End: server/routes/addOrganization.js `sendVerificationEmailForAddOrg` function

POST addOrg/email/notifyAddOrg
Sample POST HTTP Request

```
{
    "email": "user@gmail.com"
}
```

Verify Organization

Purpose: Notify Donor Admin by email when their organization has been verified with Affordable.

Front End: app/src/pages/MainPages/OrganizationPages/VerifyOrganization/index.js
Email Template: server/utils.js `module.exports.formatVerifyOrgEmail` function
Back End: server/routes/verifyOrganization.js

POST verifyOrg/email/notifyVerifyOrg
Sample POST HTTP Request

```
{
```

```
    "email": "user@gmail.com"
}
```

HUG Creation

Purpose: Notify Donor Admin by email when their HUG has been created for the organization they are representing.

Front End: app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/hugcreation.js
Email Template: server/utils.js `module.exports.formatAddHugEmail` function
Back End: server/routes/addHug.js `sendConfirmationEmailForHugCreation` function

POST hug/email/notifyHugCreation

Sample POST HTTP Request

```
{
    "email": "user@gmail.com",
    "grantName": "My Grant Name"
}
```

HUG Status of Funding Goal Met

Purpose: Notify Donor Admin by email when their HUG funding goal has been met.

Front End: app/src/pages/MainPages/HUGPages/HUGPagesRoutingContainer/HugSearch.js
Email Template: server/utils.js `module.exports.formatHugFundingEmail` function
Back End: server/routes/search.js `sendVerificationEmailForHugFunding` function

POST search/email/notifyHugFunding

Sample POST HTTP Request

```
{
    "email": "user@gmail.com",
    "grantName": "My Grant Name",
    "orgName": "My Organization Name"
}
```

Resources

https://stackoverflow.com/questions/23982299/nodemailer-and-amazon-ses
https://medium.com/viithiisys/node-mailer-with-amazon-ses-6fb18bea568e
https://nodemailer.com/transports/ses/
https://yarnpkg.com/en/package/nodemailer-ses-transport
https://docs.aws.amazon.com/ses/latest/DeveloperGuide/using-credentials.html?icmpid=docs_ses_console

Testing

Technologies Used	
Jest	Jest is a delightful JavaScript Testing Framework with a focus on simplicity. Snapshot testing with jest helps keep track of large objects with ease. Snapshots live either alongside your tests, or embedded inline.
Selenium	Selenium is used with a web driver to imitate user interaction with the application. This is heavily dependent on the naming and organization of frontend components. Only one test can be run at a time from the selenium folder.

Generating Code Coverage

1. Navigate to server folder
2. Type ‘npx jest --coverage’

Viewing Code Coverage in Browser

1. Navigate to coverage/lcov-report folder through file explorer
2. Open index.html in a browser
3. Click server/routes to view the code coverage of the routes
4. Click on a route to view the lines of code that have not been covered

Running Selenium Tests

1. Navigate to selenium folder
2. Type ‘python nameoftest.py”
3. Wait for browser to automatically open and wait for test completion
    a. Any errors will be output to the terminal console
    b. Successful test is indicated by completion of the script without error messages

Tests Implemented:

- addHug_success.py - works
- verifyorg_add.py - works and email worked
- verifyorg_fail.py - works
- verifyorg_success

Outstanding Bugs

Email Bug

- This bug was preexisting to our time on the project
- Discoveries:
    - The username is blank when passed to the backend method
        - Always on the first login of a new instance of the application
    - When the username has a value, the pop-up does not happen
        - This only occurs if you are logging in again on the same running instance of the application