Structure of Files:

- Frontend
    - Public
        - index.html
    - Src
        - Demo.js, Index.js, Piechart.js, SharedContext.js, style.css
    - node_modules (git_ignore)
    - package. json
- Backend
    - Src
        - App.js, Db.js, Server.js, table_data.js
    - Node_modules
    - Api

Backend Scripts

App.js:

- Paths from the front end gets sent to here, and calls the specified query function for that path. The query functions are located and defined in table_data.js
    - '/v0/data/:s/:t'
        - Will take 2 parameters, demand + job title and call getDemandWithJob

Db.js

- Contains aws credentials for connecting to the database
- Connects to database
- dbGet(query)
    - Takes query and retrieve from database, returns result

Server.js

- Starts up localhost on port 3010 to test different queries of the database and see formatted results, expected input and output, and statuses.

Table_data.js:

- callQuery(res,query)
    - Waits for passed in statement to be queried from the database using db.js's dbGet function. Depending on the result, returns a 200 status if found, and 404 if not found.
- getDemandWithJob(req,res)
    - Uses switch statements to determine whether to query with demand being Skills, Languages, Degrees, Education, Industries, or Most Popular Fields, in combination which job title gets passed in.
    - Specific skill and job tables will join on ids.
    - Calls callQuery to execute.
- getPopularFields(req,res)

- Contains 2 queries
    - 1 for grabbing all tables and their row counts
    - 1 for grabbing all job tables and their row counts
- Calls callQuery to execute.

Frontend Scripts

Demo.js

- Using JSX, useStates, functions, piechart.js
- handleSubmit()
    - Gets called on any button click with passed in values and fetches from database by calling query()
- query()
    - Takes passed in demand and job to format an url that gets fetched, which is handled by backend components. If the response is valid (json), then the useState for data is updated, allowing for piechart's values to be updated as well.
- handleJobChange
    - Called when the user selects a new job from the dropdown list. Updates the selectedJobs state and calls a new query.
- newCard
    - Called when a new card is added.
    - Returns a card that contains a drop down value of different job titles and 5 different buttons (demands)
    - Contains a pie chart id
    - Determines which button has been clicked
- addCard
    - Creates a new card and updates the relevant states
- useEffect
    - Updates pie chart without having to re-query the database
- useEffectWhen
    - Referenced: *https://stackoverflow.com/questions/55724642/react-useeffect-hook-when-only-one-of-the-effects-deps-changes-but-not-the-oth*
    - Draws a new pie chart if the number of data entries has changed
- updateDemands
    - Updates the selectedDemands state by replacing the demand at the specified ID if it exists, else creates a new entry.
- updateData
    - Updates the data state by replacing the data at the specified ID if it exists, else creates a new entry.

frontend.test.js

- Test cases for validating if the initial web app is loaded properly ie:
    - Title of page is loaded, empty card set, empty pie chart

globalSetup.js

- Starts dev server before all tests start

globalTeardown.js

- Stops the dev server after tests finish

index.js

piechart.js

- Referenced: https://codepen.io/thecraftycoderpdx/pen/jZyzKo
- processLabel()
    - Formats data values back into user readable format
- pieChart()
    - Draws a pie chart given id and data
- Event listener - chartUpdate
    - Transitions chart to new incoming data

Style.css

Testing:

- frontend/src/frontend.test.js
    - Using Jest to create and close a new browser tab for each test case, with a timeout set to 10000 (10 seconds). Each test case is independent from the previous ones, but all rely on using localhost port 3000 as the source url.
- backend
    - The user is only allowed to interact with buttons and dropdowns, so the margin for error is slim when it comes to querying from the database. There is no unit test for the connection from user interface to database.
- Tests performed:
    - Page title of web app to be *PathFinder*
    - Initial empty pie as null
    - Initial empty card set as null

Known bugs:

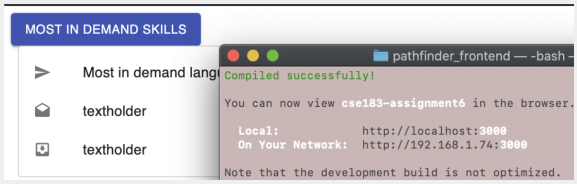
- When displaying fewer than 20 results in the chart, there will be a significantly darker line from the center of the chart to its top.
    - This could potentially be fixed by redoing the chart transition code to properly deal with changing amounts of data rather than filling the chart with empty data so that the chart would always have 20 data entries.

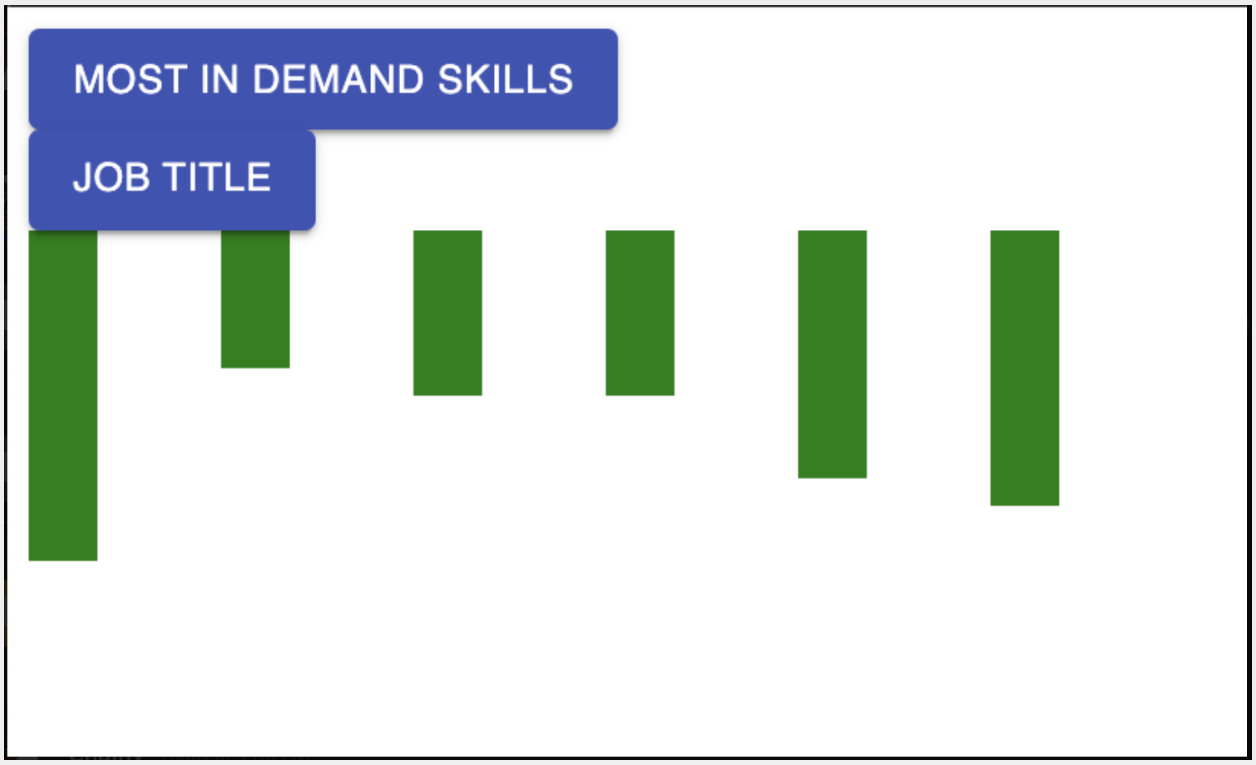
General Notes:

Front end is currently running on React with D3 library, connection for front end to database supported by Pool, and validation on url and paths uses Express and yaml-api. The aws credentials for connecting to the database is currently not hidden, but is okay for now, since our repo is private. The structure of folders and codebase is based off of CSE183 Web Applications with Professor `David Harrison` , but restructured for our own use.

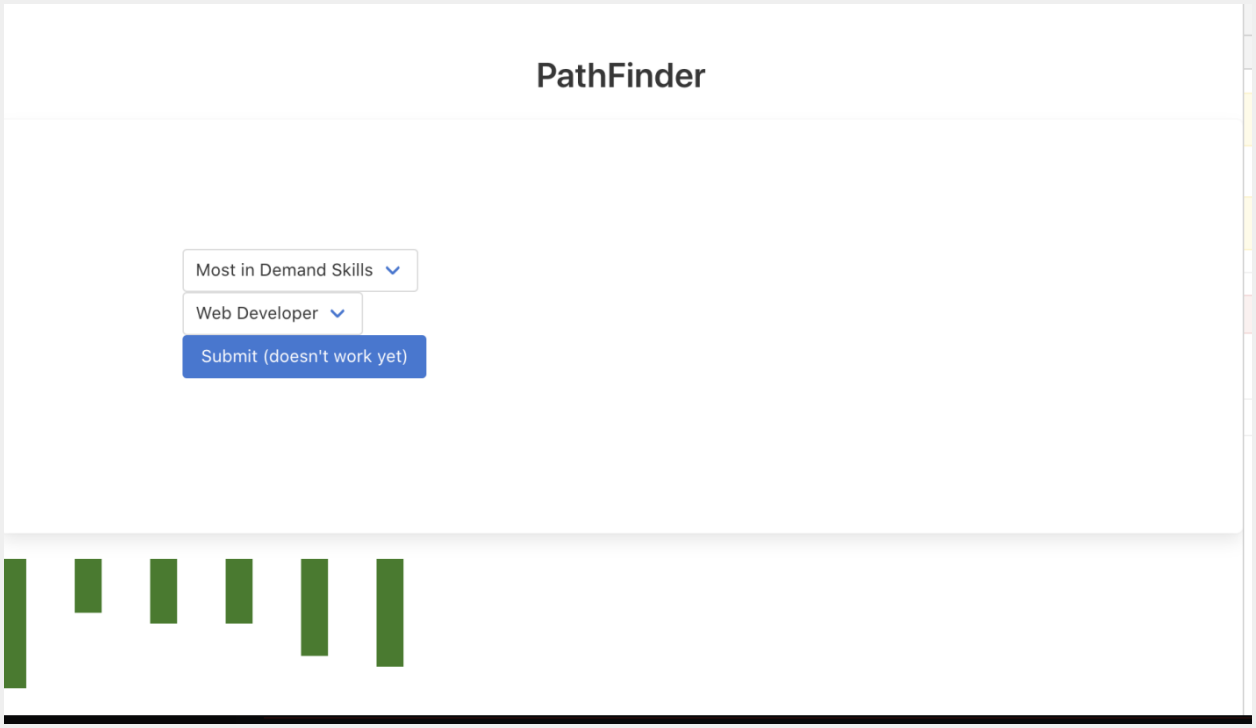
Iterations of front end design
React → Vue.js → React

May 14 → May 16 (React)

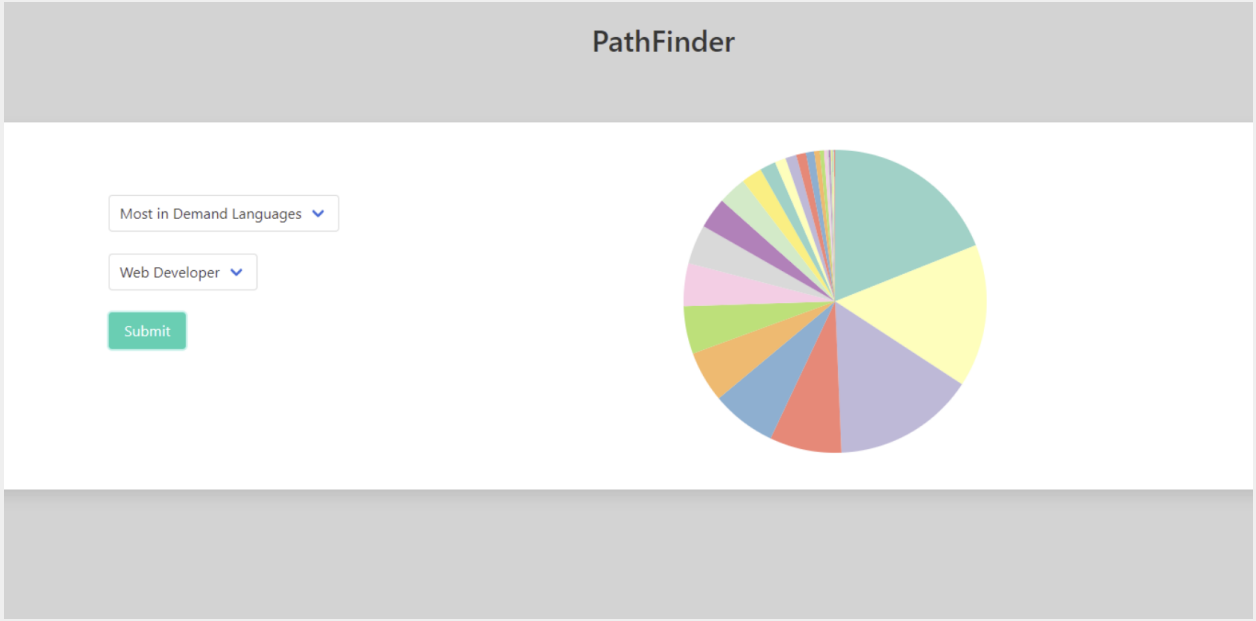




May 16 → May 18 (Vue.js)



May 18 → May 26 (React)

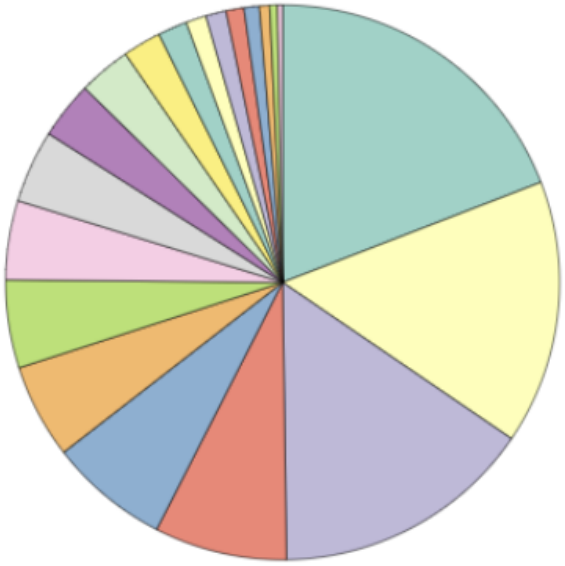


May 26 → June 1 (React)

PathFinder

Web Developer

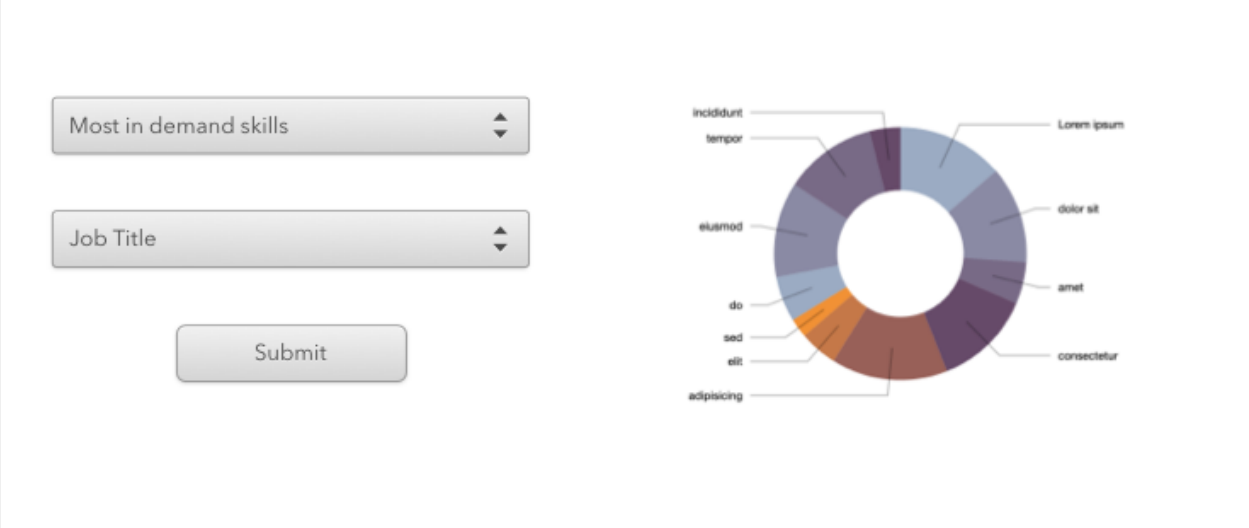
Top 20 Languages



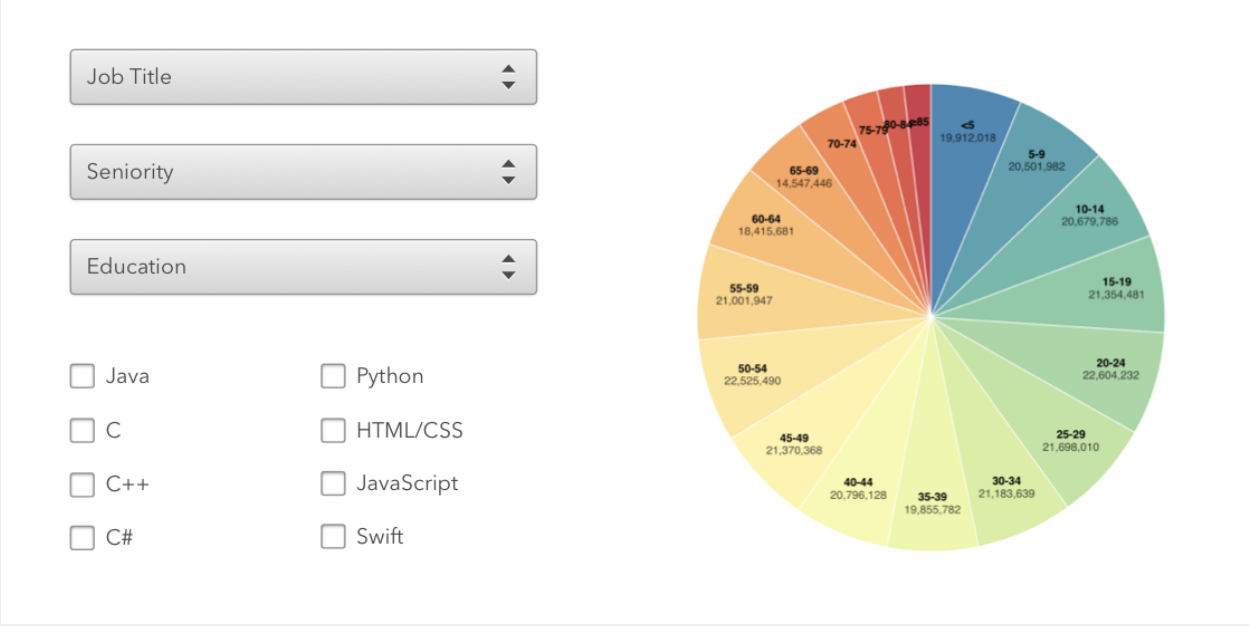
Languages | Skills | Degrees | Education | Industries

Prototype Ideas

As a user, I want to see the most in-demand skills from [insert job title]



- On submit
    - 2 Queries:
        - Most in demand skills + job title
        - Most in demand skills



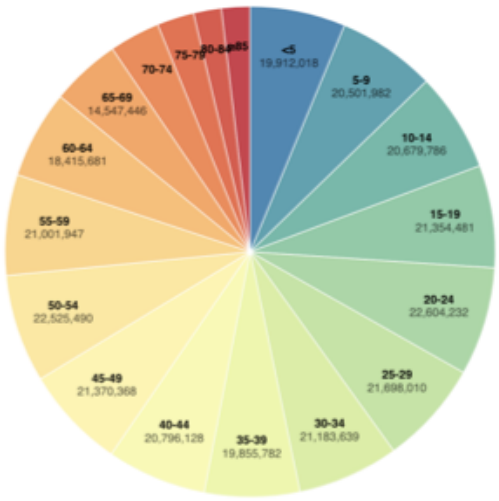
Job Title

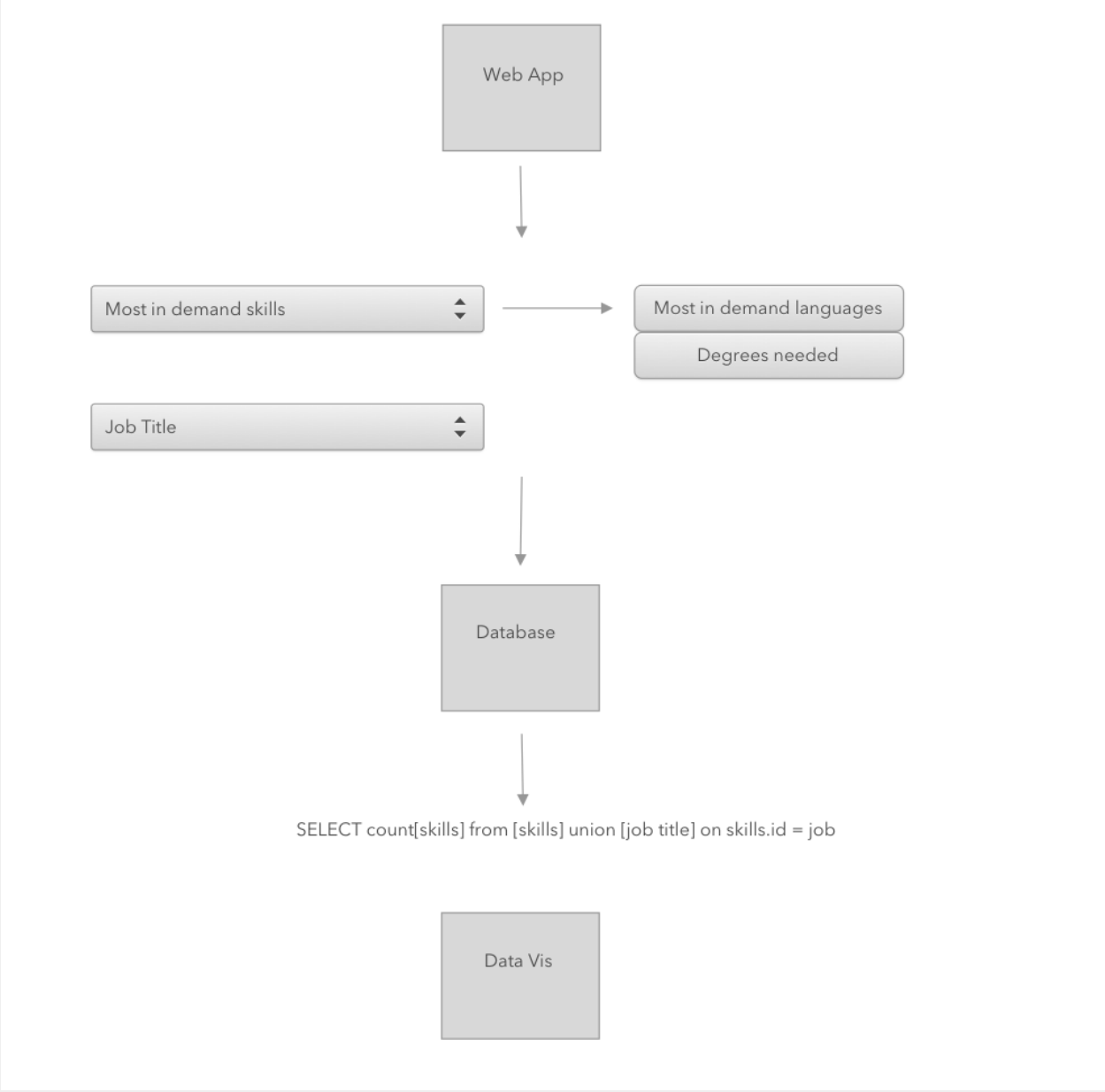
Seniority

Education

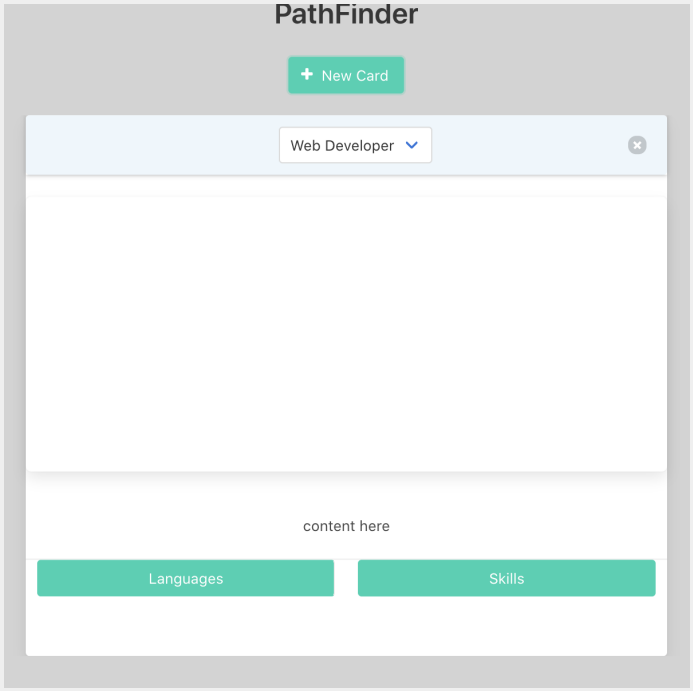
- [ ] Java
- [ ] Python
- [ ] C
- [ ] HTML/CSS
- [ ] C++
- [ ] JavaScript
- [ ] C#
- [ ] Swift

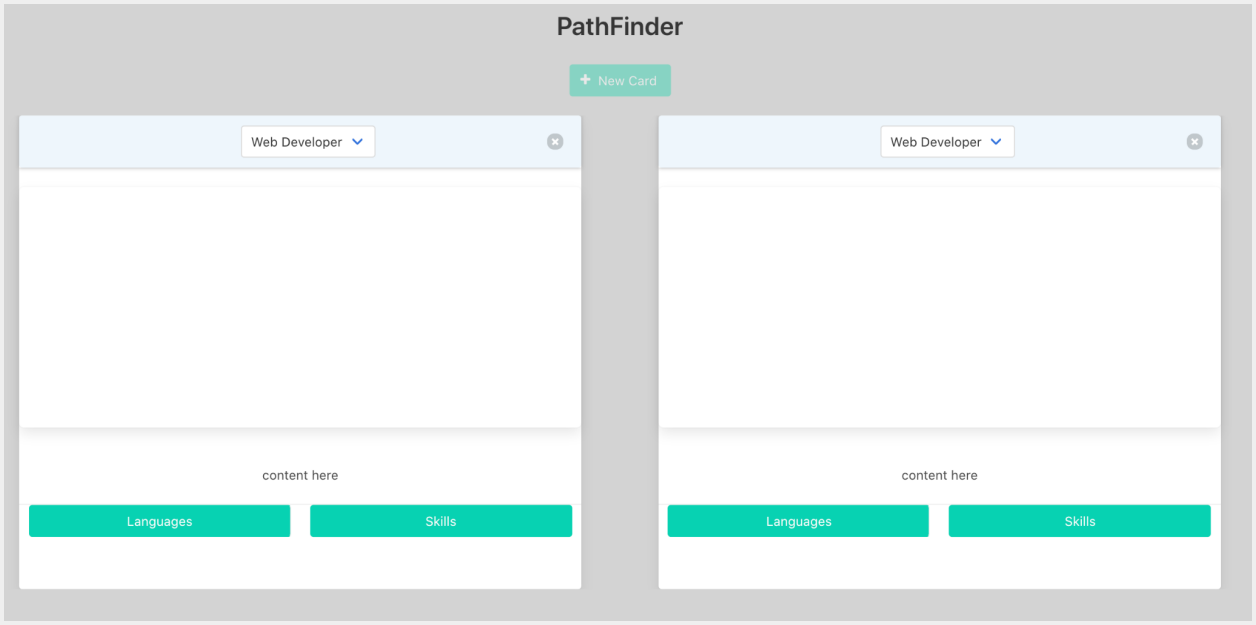
Submit

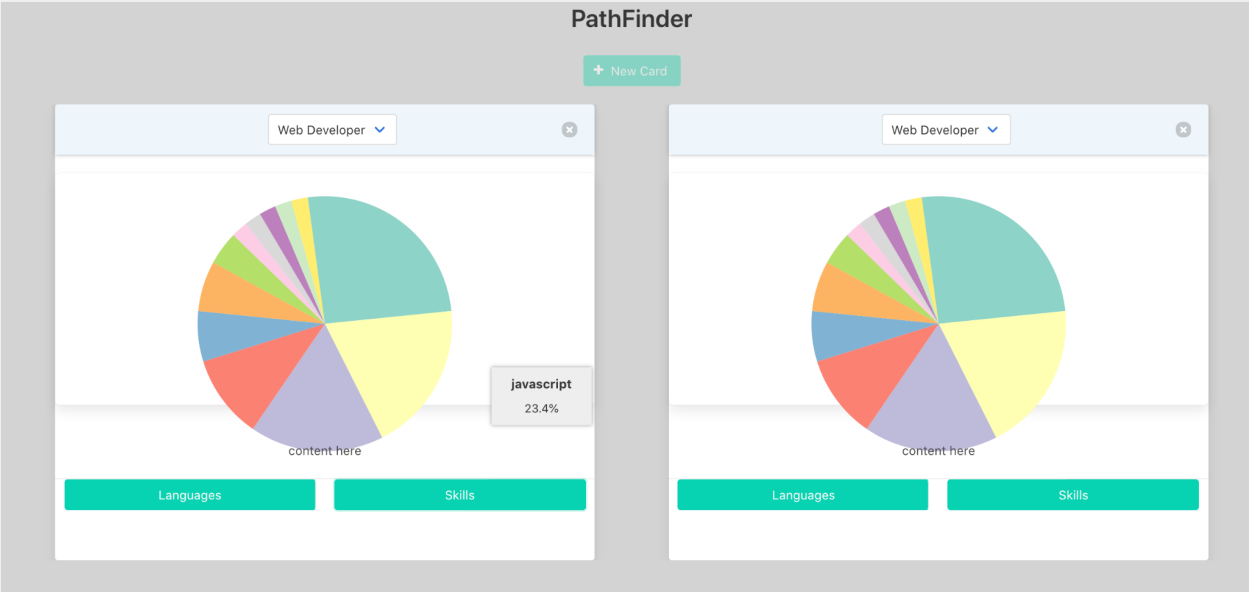



Web App
Most in demand skills
Most in demand languages
Degrees needed
Job Title
Database
SELECT count[skills] from [skills] union [job title] on skills.id = job
Data Vis

Progress:


PathFinder
New Card
Web Developer
content here
Languages
Skills


PathFinder
New Card
Web Developer
content here
Languages
Skills
Web Developer
content here
Languages
Skills



Changes:

- Changed from Vue.js → React
- Classes → functions
- Hooks/states → Shared Context