Project Archive - Job Tracker Final Report

CS 497: Capstone Project

Winter 2023

Oregon State University

Rex Fagin, Patrycjusz Bachleda, Philip Peiffer

Table Of Contents

Introduction

What follows is the final report for the Job Tracker application. The team members of this project were Rex Fagin, Patrycjusz Bachleda, and Philip Peiffer. The team put in a tremendous effort and developed a user friendly, adaptable, and scalable application that will allow job seekers the opportunity to track their applications in a centralized location. Overall, the team worked well together and each member contributed meaningfully to the end goal. While we were not able to complete the stretch goal of integrating the Glassdoor or Reddit API set out in our project plan, we believe that we set the project up for seamless integration of that stretch goal.

User’s Perspective

From a user’s perspective, our project will allow them to track their job application efforts through a web based UI. The user can create an account and add jobs that they’ve applied for or are interested in to their account. They will then be able to track the status of those applications through various categories.

On top of job tracking, the user will be able to create skills, tie skills to certain jobs, and rate their comfort level with said skills. They will then be able to view insights into their skills (such as frequency needed vs. comfort level), with the goal of the insights being to improve their comfort level for their top needed skills. Finally, the user will be able to create networking contacts and tie them to the jobs as well, so that they have a central place to maintain contact info.

Development Efforts vs Plan

True to our original development plan, we decided to focus first on the backend API that would service all of our CRUD operations from the frontend. From there, we developed a rough frontend that acted as a sort of placeholder so that we could experiment with the UI and figure out how best to display the data. After that, we integrated authentication and authorization and tied all of our entities to users. So from a high level perspective, our development efforts matched our plan fairly well. Where we deviated from the plan is hidden in the details of our ERD, when and how we integrated authorization, and initial UI designs.

We’ll focus on the ERD first. Our original ERD is given below. You’ll notice that skills are related to users in a many to many relationship. This means that skills are shared amongst users. However, from the user’s perspective we know that we must be able to rate comfort level with skills. Therefore, skills must also be user specific. We overcame this by storing an array of skills under the user. Originally, this array held maps that stored the skill ID and the proficiency. This is where the problems started. When displaying a user’s skills, we needed to display the description of the skill along with their rating. Since the skills *entity* doesn’t have a link to the user, this means that we just look through the skills *array* under the user for this information, but the skills array doesn’t contain a description. This meant we’d have to loop through all of the skills in the array and perform database queries on each skill to get the description. This is not the most efficient, even if the skill ID is indexed. To overcome this, we started enforcing that the skill description had to be unique, and then added the description to the user's skills array. This made it so that everything was in one place.

Continuing along with the skills entity. You'll notice that originally we had an array of application IDs that were tied to each skill. We thought this would be necessary to see how often a skill is linked to an application. However, remember skills are shared amongst users. This means that *all* applications in the database could be in this array (if all applications were tied to a certain skill), not just a certain user's applications. So to find out how often a skill is referenced in a user's applications, you'd have to iterate through the applications array and figure out which ones were tied to the user. Once again, this is not the most efficient way. We decided to remove this array and instead implement a mapping algorithm for this information.

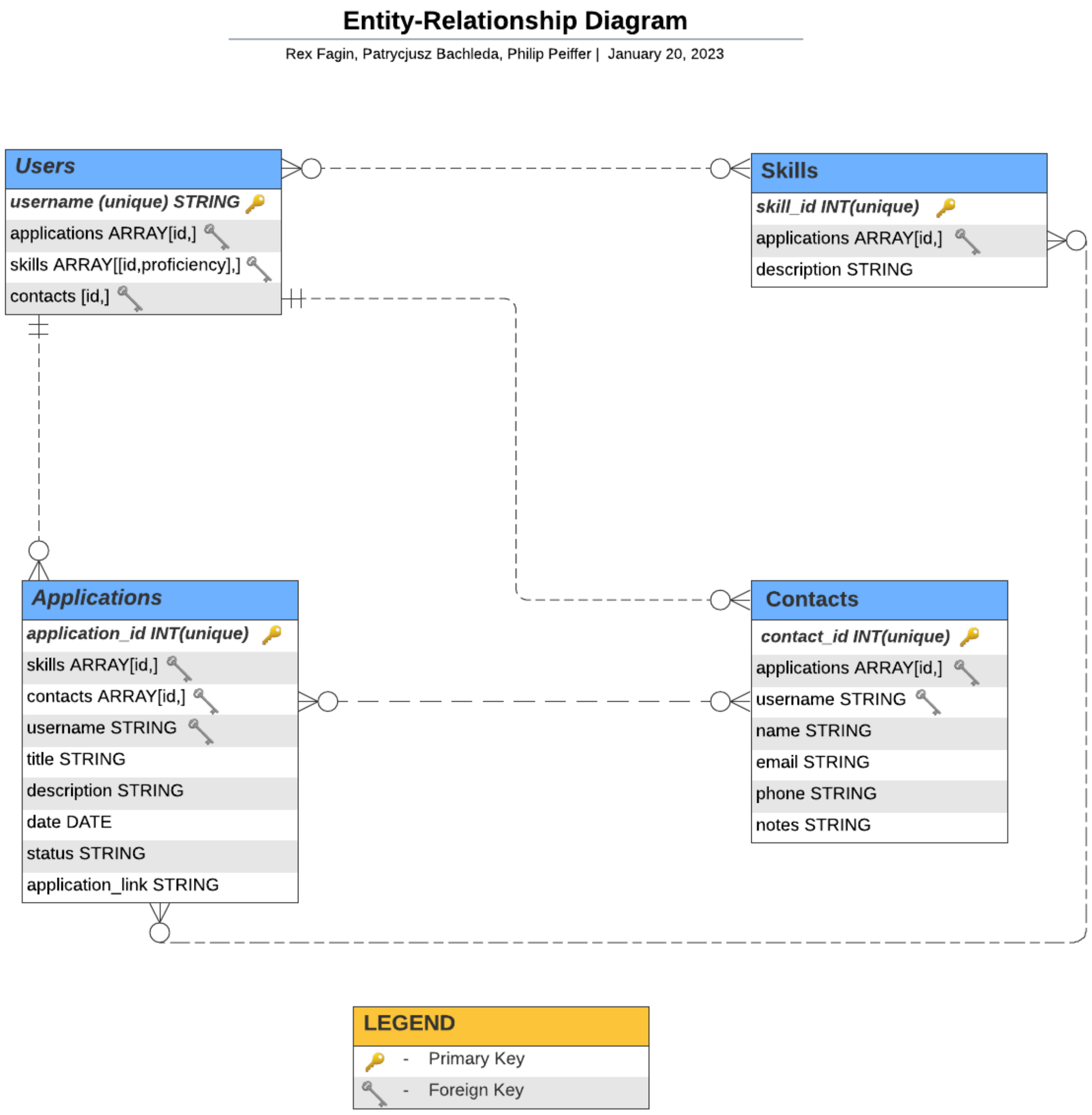
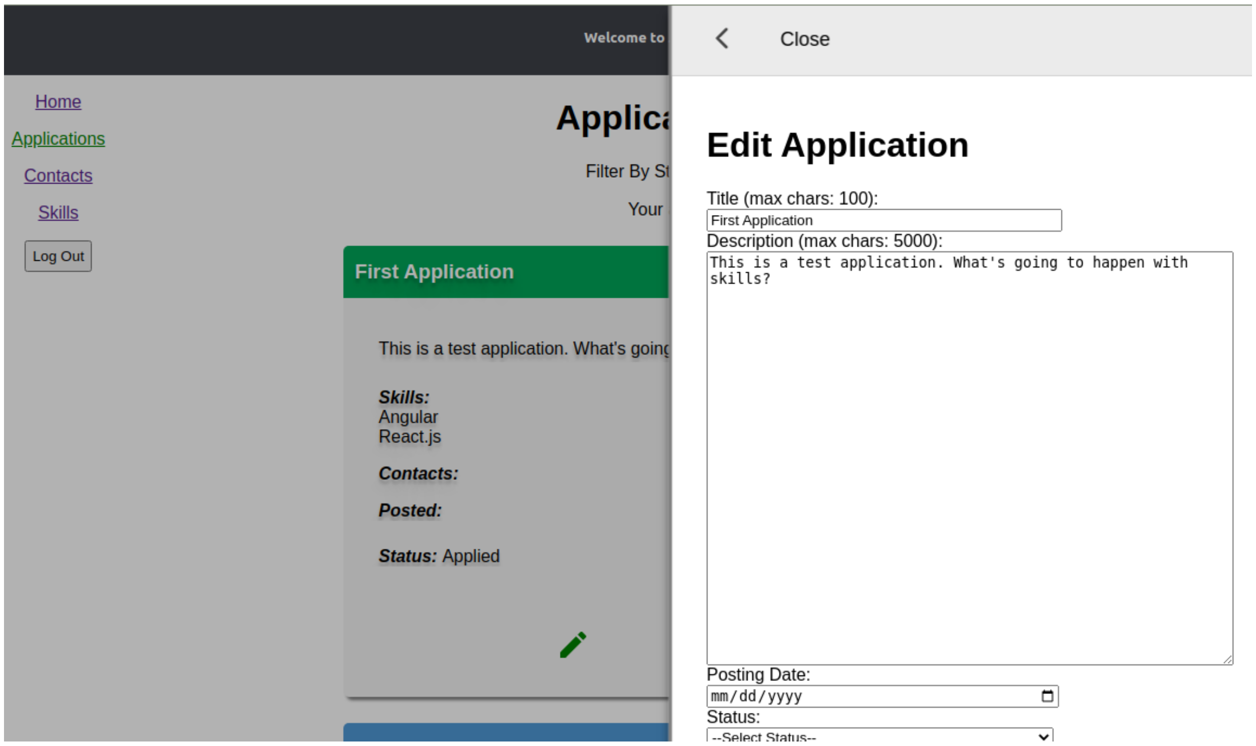


Figure 1: Original ERD

The final thing that we changed about the ERD is that we decided to use an ID instead of username to uniquely identify each user. The ID was issued by Auth0 when their account was created and provided in the access token. This allowed us to map Auth0 users to the database if ever needed.

Speaking of authentication, there was a slight change in when/how we decided to authenticate users. Originally we planned on implementing authentication in Sprint 3. We ended up not implementing true authentication until Sprint 4. However, we did stub out this process in Sprint 2 so that we mimicked how it was going to work and fixed most of the bugs before integrating Auth0. This allowed us to continue development at a rapid pace and also seamlessly integrate Auth0 when the time came to truly authenticate.

Moving to the front-end side of things, there were some changes in how we decided to implement the UI. Our original plan was mostly table based and involved navigating between separate screens to view and edit each entity. We felt like this approach could be improved and decided to implement a sliding pane. When an entity is clicked, a sliding window opens over the current window and edits can be made without navigating away from the current screen. This made the interaction feel more seamless and intuitive.



First Application

This is a test application. What's going

Skills:
Angular
React.js

Contacts:

Posted:

Status: Applied

Edit Application

Title (max chars: 100): First Application
Description (max chars: 5000): This is a test application. What's going to happen with skills?
Posting Date: mm/dd/yyyy
Status: --Select Status--

Figure 2: Sliding window in action.

Finally, there were some changes in how we decided to version control and deploy our project. We originally planned to have a Git repository for each “service” in the project. We’d have a repository for the back-end and one for the front-end. This would have allowed us to have more control over the CI/CD process and provided better commit history for each domain. However, after discussion, we decided to put all of our code in one repository. This allowed us to only focus on one repository, but it did sacrifice the other two considerations above.

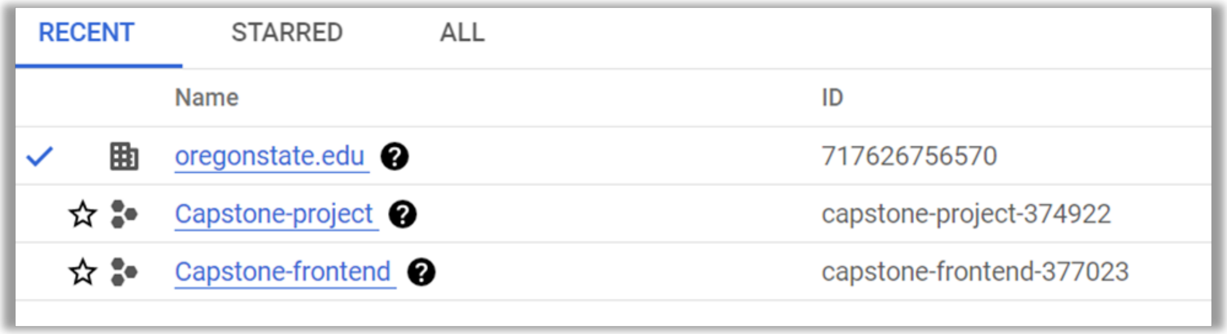
For deployment, after our midpoint project archive we determined that it would be beneficial to have two different deployment environments. We created another deployment environment for “dev” so that we could make changes and test out our application before pushing to production. This aligns with what a lot of companies do with modern CI/CD practices.

Technologies Used

The Job Tracker project is a full-stack development application, it includes both the front end and back end. The choice of technologies that we used to create the application was determined at the beginning of the process.

Deployment

The Job Tracker application is deployed on Google Cloud due to our experience with this service from previous OSU courses. The project is deployed via two different apps: the first app is the React front end; the second app is the Node/Express back end, which services data requests from the front end to the API connected to datastore.



	Name	ID
✓	oregonstate.edu	717626756570
☆	Capstone-project	capstone-project-374922
☆	Capstone-frontend	capstone-frontend-377023

Fig 3. Screenshot from Google Cloud: Capstone-project – the back end app, Capstone-frontend – the front end app.

Front End

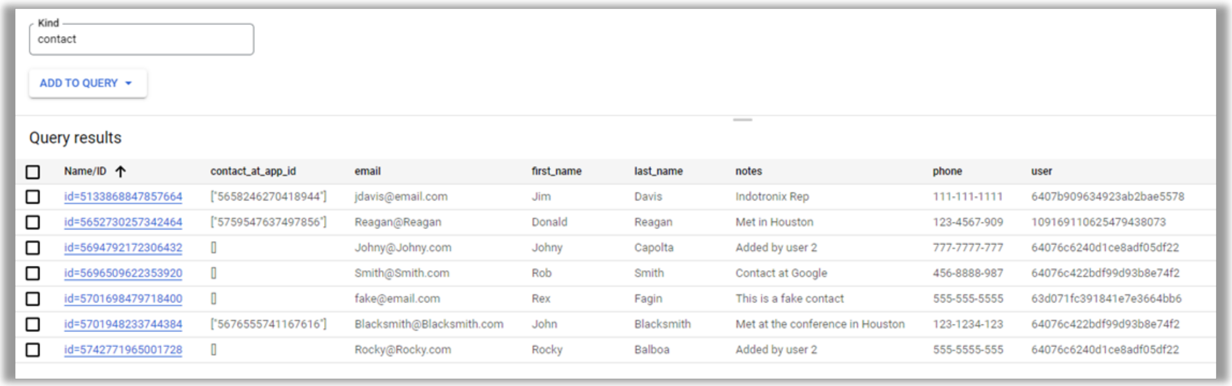
We chose React.js, which is a front end JavaScript library that is capable of making API calls to the back end. Using React allowed us to build components that could be reused for different pages of the application. One example of such a component is the sliding window component which is used for Applications, Contacts and Skills pages, it allows the user to add and edit data on these pages. Another thing about React is its rich UI component library that gave us a good selection of components we could choose from.

Back End

The back end uses Node.js that is a cross-platform JavaScript runtime environment and Express.js that is a back end web application framework for RESTful APIs. The back end connects to the database. Requests are sent from the front end via HTTP and are caught/serviced by express on the back end.

Database

We have decided to use the NoSQL Google Cloud Datastore. We chose this database due to the familiarity with it and the availability of gcloud documentation and support to troubleshoot issues.



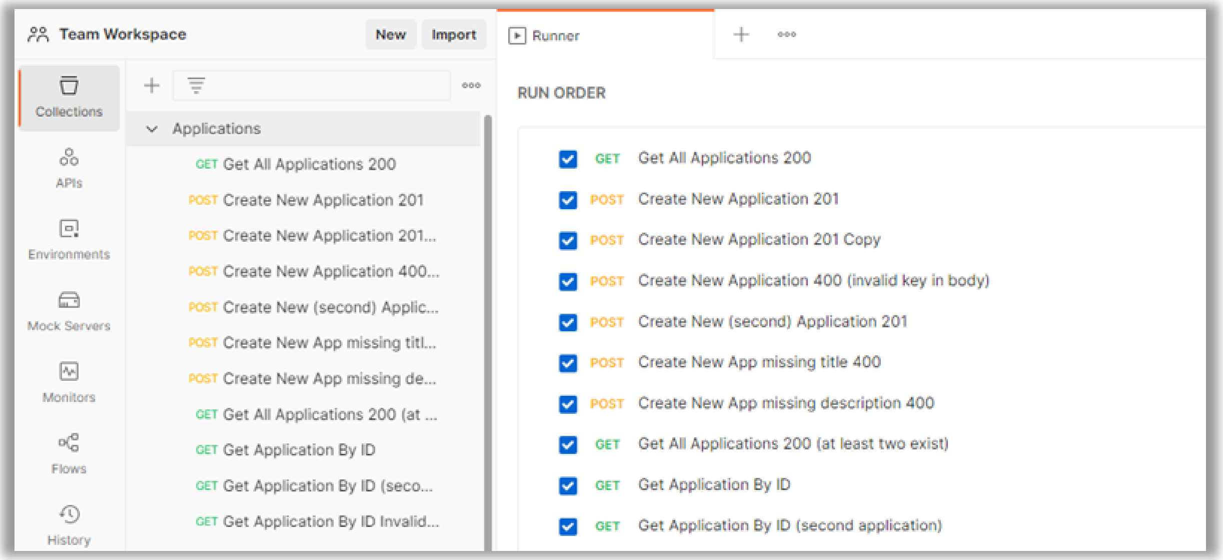
Query results

Name/ID ↑	contact_at_app_id	email	first_name	last_name	notes	phone	user
id=5133868847857664	["5658246270418944"]	jdavis@email.com	Jim	Davis	Indotronix Rep	111-111-1111	6407b909634923ab2bae5578
id=5652730257342464	["5759547637497856"]	Reagan@Reagan	Donald	Reagan	Met in Houston	123-4567-909	109169110625479438073
id=5694792172306432	[]	Johny@Johny.com	Johny	Capolta	Added by user 2	777-7777-777	64076c6240d1ce8adf05df22
id=5696509622353920	[]	Smith@Smith.com	Rob	Smith	Contact at Google	456-8888-987	64076c422bdf99d93b8e74f2
id=5701698479718400	[]	fake@email.com	Rex	Fagin	This is a fake contact	555-555-5555	63d071fc391841e7e3664bb6
id=5701948233744384	["5676555741167616"]	Blacksmith@Blacksmith.com	John	Blacksmith	Met at the conference in Houston	123-1234-123	64076c422bdf99d93b8e74f2
id=5742771965001728	[]	Rocky@Rocky.com	Rocky	Balboa	Added by user 2	555-5555-555	64076c6240d1ce8adf05df22

Fig 4. Screenshot from the Google Cloud Datastore.

Testing Tools

The team used Postman to test that backend requests are functioning correctly. This tool was intensely used especially during the back end development to automize our testing.



Applications
- GET Get All Applications 200
- POST Create New Application 201
- POST Create New Application 201...
- POST Create New Application 400...
- POST Create New (second) Applic...
- POST Create New App missing titl...
- POST Create New App missing de...
- GET Get All Applications 200 (at ...
- GET Get Application By ID
- GET Get Application By ID (seco...
- GET Get Application By ID Invalid...

RUN ORDER
- ✓ GET Get All Applications 200
- ✓ POST Create New Application 201
- ✓ POST Create New Application 201 Copy
- ✓ POST Create New Application 400 (invalid key in body)
- ✓ POST Create New (second) Application 201
- ✓ POST Create New App missing title 400
- ✓ POST Create New App missing description 400
- ✓ GET Get All Applications 200 (at least two exist)
- ✓ GET Get Application By ID
- ✓ GET Get Application By ID (second application)

Fig 5. Screenshot from Postman, shows requests for testing application calls.

Authentication/Authorization

Due to the hazards associated with creating and maintaining a custom authorization service, the team felt it best to use a reputable provider. Therefore, we used Auth0 for authentication purposes and JWTs for authorization. Auth0 provides pre-made React components to interact with their OAuth2.0 flow. Their free tier allows up to 7000 users on a single application.

<u>Development</u>

The team used a GitHub repository, it allowed each of the team members to have access to the most current code. If there were any major changes to the code, the merge had to be approved by another team member. There were in total 65 merges and 458 commits.

All of us were familiar with these technologies, with a small exception of React that was a new technology to one team member. Therefore, the process of using these tools went quite smoothly during the creation of our web application.

Conclusion

The team was created during the first week of the course. All team members wanted to work on a project that required full-stack development and the Job Tracker app was our first choice to work with. This project allowed us to get more familiar and more practice with all tools and technologies that we were taught during different OSU courses. Another aspect of this project was teamwork. Some of us will soon begin our careers in software development and this gave the opportunity to learn teamwork skills such as reviewing other team members code, practice communication skills and other valuable skills for working in industry.

One future direction we would like to implement for the project is a Glassdoor API which would allow searching within the app for jobs. The project will also be used as part of our portfolios.