

Iowa State University
COM S 319: Construction of User Interfaces
December 13, 2023

ISU Student Job Board
Documentation

Prepared By
Ryan Huellen
rhuellen@iastate.edu

Jayden Luse
jcluse@iastate.edu

Prepared For
Dr. Abraham Aldaco

Table of Contents

Project Description.....	1
Software Functionality Diagram.....	2
Files and Directory Architecture.....	4
Client – Server Architecture.....	5
Logical Architecture.....	6
Database Design.....	7
API Design.....	9
Web Views.....	11
Installation Manual.....	16

Project Description

With Iowa State University's transition off AccessPlus, students and staff are experiencing the transition of class registration, financial reporting, offers of admission, among other features will be moving to Workday. However, one key feature, the student job board, has not seen any plans to transition.

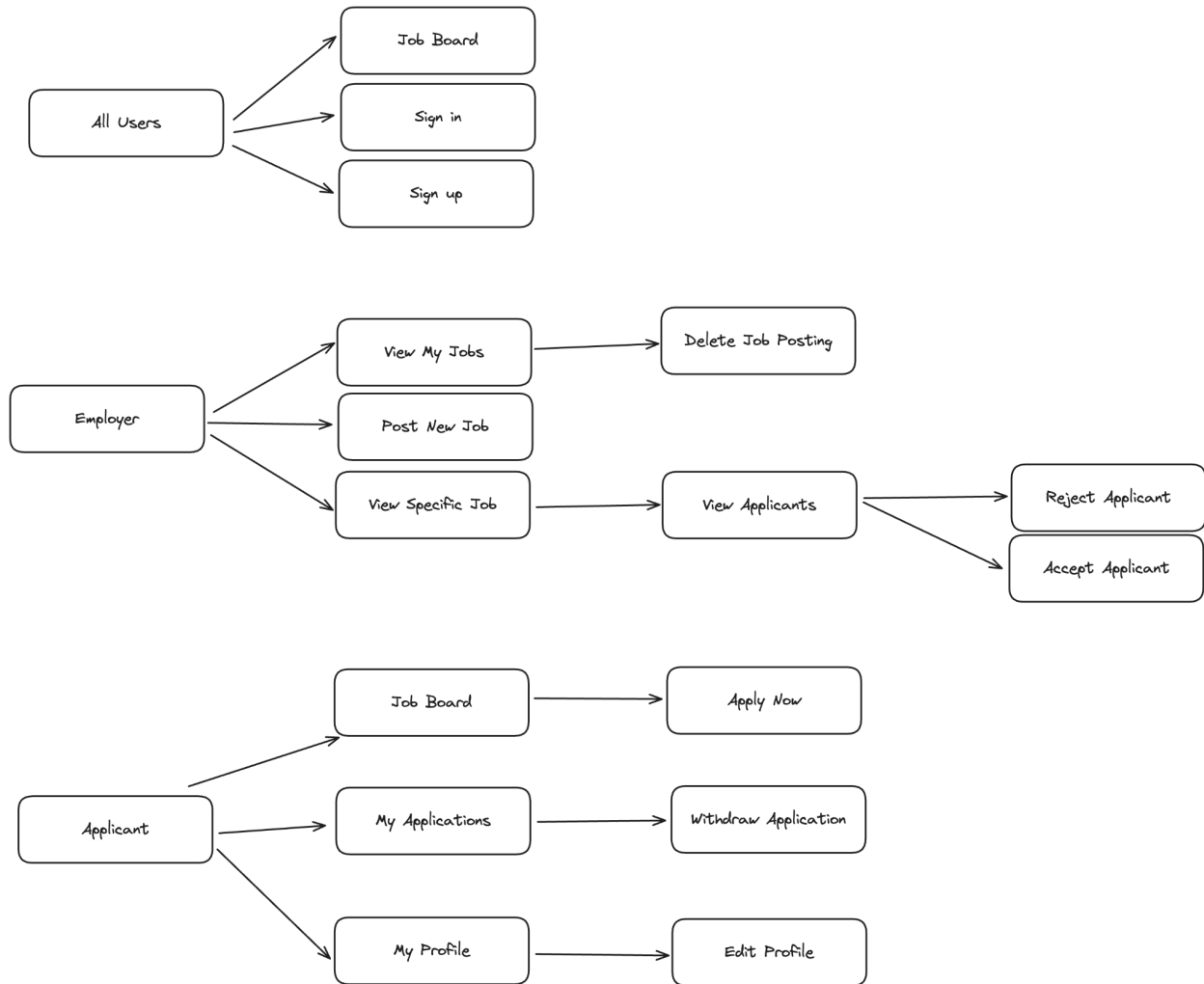
Given the need and prevalence of a student job board, both Jayden Luse, and myself, Ryan Huellen, saw a unique opportunity to create a mock job board as part of our coursework in COM S 309: Construction of User Interfaces.

To follow, we will present the entirety of our final project dubbed "ISU Student Job Board." The application is complete with views for users to sign in to an existing account, sign up for an account, and view the job board. At this time, viewing the available job postings does not require an account.

To continue, employers can post a job of their choosing by entering a job title and company name. Once applicants hit apply, a feature we will describe in the next paragraph, employers will be able to see applicants and their information in its entirety.

From an applicant perspective, once I sign in and fill out my profile, which includes information such as name, desired position title, salary expectation, and a file upload of a resume, you can begin applying for different jobs. By hitting "Apply Now," employers will be able to see your information and update your application status as they see fit. Moreover, you can also view your applications that are under review.

Software Functionality Diagram



Depending on your user type, different portions of the software’s functionality will change in importance. As such, we describe our three user groups: all users, employers, and applicants (students), separately.

All Users

As a user of the application, typically one who is not signed in, you have the ability to perform three actions: view the job board, including all the active job postings, sign in to your account, and sign up for an account. All of these functions are relatively rudimentary and function as expected.

Employer

As an employer, your scope of functionality expands dramatically. Upon signing in, you can do one of the following:

1. View My Jobs: See all of the current job postings you have as an employer.
 - a. Delete Job Posting: Delete a job posting if you found the right candidate.

Software Functionality Diagram

2. Post New Job: Create a new job on the job board when you have an opening.
3. View Specific Job: Navigate to a specific job you'd like to view details about.
 - a. View Applicants: See all the applicants for a job posting and their details.
 - i. Reject Applicant: Reject an applicant from the job posting.
 - ii. Accept Applicant: Accept an applicant to fulfill the job posting's open position. Please note, this does not delete the job posting as you can accept multiple applicants.

Applicant

As an applicant, you have the ability to do the following:

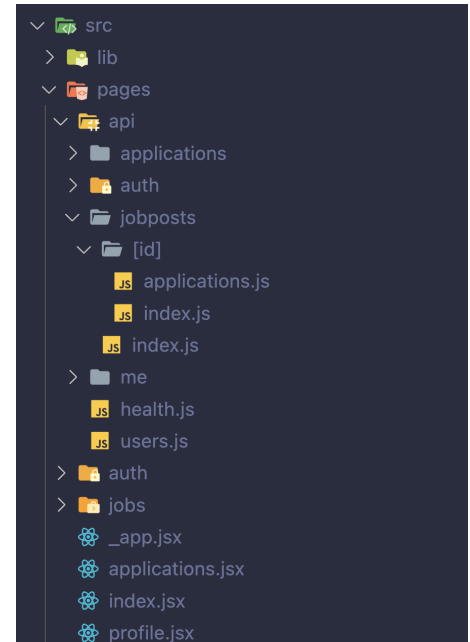
1. View Job Board: Though all applicants can view the job board, you have additional functionality as a signed in applicant.
 - a. Apply Now: Immediately submit the information on your profile to your prospective employer.
2. My Applications: View all pending applicants that you have.
 - a. Withdraw Application: Should you find a position elsewhere, you can withdraw applications that are currently under review from other companies.
3. My Profile: View your profile and your current preferences.
 - a. Edit Profile: Edit fields such as salary expectation, desired job position title, and update your resume as you see fit.

Files and Directory Architecture

This project uses a modern production-ready React framework dubbed Next.js. This framework leverages a paradigm called “file based routing.” Essentially, all files created under a ``/pages`` directory automatically become single page routes. As such, from a development perspective, you don’t need to configure a React router whatsoever.

In the graphic on the right, you can see various files. Namely, the ``index.jsx`` file. When exporting a component from this file, it will automatically represent what’s rendered when someone visits the root directory ``/`` of our website. Likewise, for our ``applications.jsx`` file, the exported component will be rendered when the ``/applications`` directory is visited.

To continue, you may notice we don’t have a separate backend directory as seen in many fullstack web applications. This is because Next.js is a batteries included framework. Like the ``/pages`` directory, we have a ``/api`` directory which holds all of the Express.js handlers for our various routes. For instance, take ``/api/users.js``. Here, we export a default handler that will automatically be called when a user makes a request to the ``/api/users`` path. Please note, however, despite these files being in the same directory as ``/pages``, these functions are indeed run on the server and are entirely hidden from the client view. As such, we can include database credentials or make calls from these routes in a secure manner.



Finally, you may notice some folders wrapped with brackets. For instance, we have an ``/api/jobposts/[id]`` folder. Files within this folder will match a query parameter in the request. For example, ``/api/jobposts/[id]/applications.js`` will match the following route: ``/api/jobposts/42/applications``. In this case, 42 is a randomly chosen id of a job post. In this way, we can follow REST conventions when dealing with CRUD operations, specifically READ, UPDATE, and DELETE.

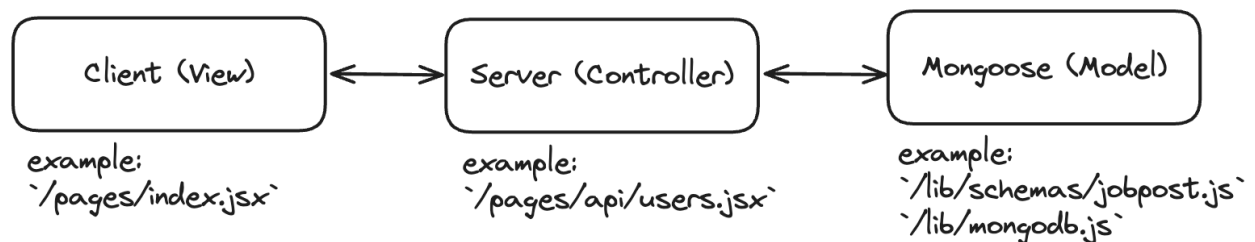
Since the introduction of Next.js, many other JavaScript frameworks in the ecosystem have followed suit in providing file based routing out of the box to developers. In our opinion, this allows us to not only move quicker, but to ensure all of our routes are included, without messing with a React router. Pretty cool.

Client – Server Architecture

The software has three main components: the model, view, and controller. The model represents our data, the view represents what the end user can actually see in the browser, and the controller represents how a user can interact with that data. This is also known as an MVC architecture.

In our project, the Model is our Mongoose schema for Users, Job Posts, and Applications. Our view is our React components inside our ``/pages``, ``/lib/layouts``, and ``/lib/components`` directories. Finally, our controllers are our Express.js API routes within our ``/pages/api`` directory.

To get a better idea of how these components talk to each other, along with example file paths in which they do, see the following graphic:

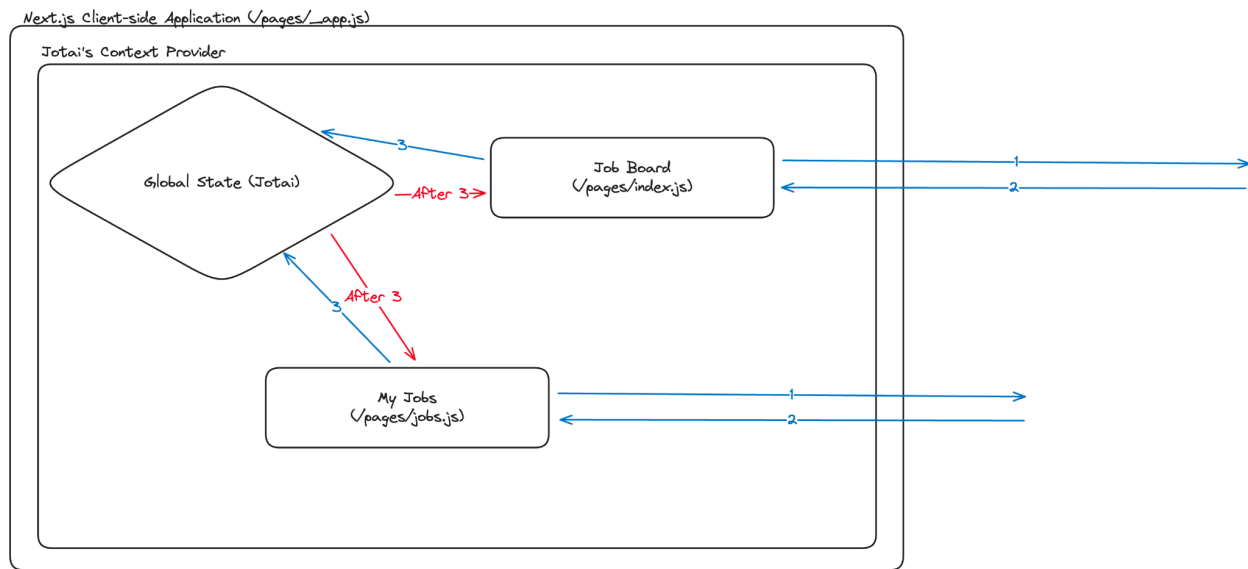


Notice how all of our database communication occurs on the server. The server is responsible for verifying all requests from the client are both valid and contain the necessary information to avoid situations like data corruption or unauthorized access to data. In this way, our database stays secure and our application has predictable functionality.

Logical Architecture

In order to prevent page flickering, data in our application is handled in a global state. In other words, we don't have individual `useState` annotations for storing data from our database. Instead, we use a global state management library `jotai`. This is a library that is both widely accepted in the Next.js ecosystem, and is easy to use. Jotai's functionality is comparable to that of React's built-in context functionality. Essentially, we wrap our application in Jotai's `ContextProvider`, then we can define variables using the library's special `atom` function. Likewise, we can use atoms throughout our application with the `useAtom` function.

Here's an example of how we handle our user data across multiple screens, such as the Job Board and My Jobs pages.



In this example, we have four steps. All of the steps can be performed from any page that contains the global data.

1. Upon initial load of any page, we fetch the user's data from the server. In our application, this is located at the `/api/me` endpoint.
2. Once the data is retrieved from step 1, we feed it back to the respective component.
3. We store the data in Jotai's Global State. From this point onward, we don't need to refetch the data as all pages will have access to it.

Once the data is in the global state, upon navigation on our single page application, because the page doesn't reload, by design, the data in memory does not get erased. Moreover, any page can pull from our data store. In this way, we only fetch data once. This saves our server computing power, prevents unnecessary loading screens for the client, and makes the overall application feel fast! All of our application's critical data is handled this way, especially if caching isn't an option. To make this concrete, we do not do this for our job posts, because that data is changing very often and it probably makes more sense to refresh it when we navigate!

Database Design

The database is made up of three core tables: Users, Job Posts, and Applications. These tables are defined by Mongoose schemas in the `/lib/models` folder in our project. For the sake of clarity, I will show the exact code snippets from these files.

Job Post Schema:

```
const jobPostSchema = new Schema({
  position: { type: String, required: true },
  company: { type: String, required: true },
  ownerId: { type: String, required: true },
});
```

Position: The title of the position the job post is for.

Company: The name of the company the job post is for.

Owner Id: The User Id of the individual who created the job post.

User Schema:

```
const userSchema = new Schema({
  firstName: { type: String, required: true },
  lastName: { type: String, required: true },
  desiredPosition: { type: String, required: true },
  salaryExpectation: { type: Number, required: true },
  resume: { type: String, required: false },
  email: { type: String, required: true },
  passwordHash: { type: String, required: true },
});
```

First Name: The user's first name.

Last Name: The user's last name.

Desired Position: The user's desired position title.

Salary Expectation: The user's salary expectation, in dollars.

Resume: The link to the user's resume, if it has been uploaded.

Email: The user's email.

Password Hash: A bcrypt hash of the user's password which is verified at every sign in.

Database Design

Application Schema:

```
const applicationSchema = new Schema({
  jobPostId: { type: String, required: true },
  ownerId: { type: String, required: true },
  status: { type: String, required: true }, // under_review, accepted, rejected, withdrawn
  company: { type: String, required: true },
  position: { type: String, required: true },
});
```

Job Post Id: The Id of the Job Post the application is for.

Owner Id: The applicant's User Id.

Status: The application's status. Thus far, we consider `under_review`, `accepted`, `rejected`, and `withdrawn` valid application statuses. In the future, we should consider creating an enum.

Company *: The title of the company the application is for.

Position *: The position title the application is for.

* These fields are technically duplicates. We copy the data from the original job post because if the original employer deletes the post, the applicant can still see their applications and the companies they were for, despite other data being missing.

API Design

The Application Programming Interface, implemented in the `/pages/api` directory of Next.js, follows the REST standard for APIs. We take advantage of the regular CRUD operations and their respective REST HTTP methods. See the following table.

CRUD	HTTP Method
Create	POST
Read	GET
Update	PUT
Delete	DELETE

Due to Next.js's file based routing, all endpoints are accessed via. the `/api/` path. For conciseness, this prefix is not included in the following table describing each endpoint.

Endpoint	Method	Description
/health	GET	View whether or not the API is online. This endpoint will always return a success message.
/users	POST	Create a new user. This endpoint is called when a user signs up for a new account.
/me	GET	Fetch the logged in user's data. Using the JWT token from the request's headers, the user's profile data is retrieved and returned to the client.
/me	POST	Update the user's data. Namely, this endpoint is called from the edit profile view.
/me/applications	GET	Retrieve the logged in user's job applications.
/me/jobposts	GET	Retrieve the logged in user's job posts.
/jobposts	GET	Retrieve all of the job posts.
/jobposts	POST	Create a new job post.
/jobposts/[id]	GET	Get a specific job post by id.
/jobposts/[id]	DELETE	Delete a specific job post by id. This functionality is locked down to the employer who originally created the job post.

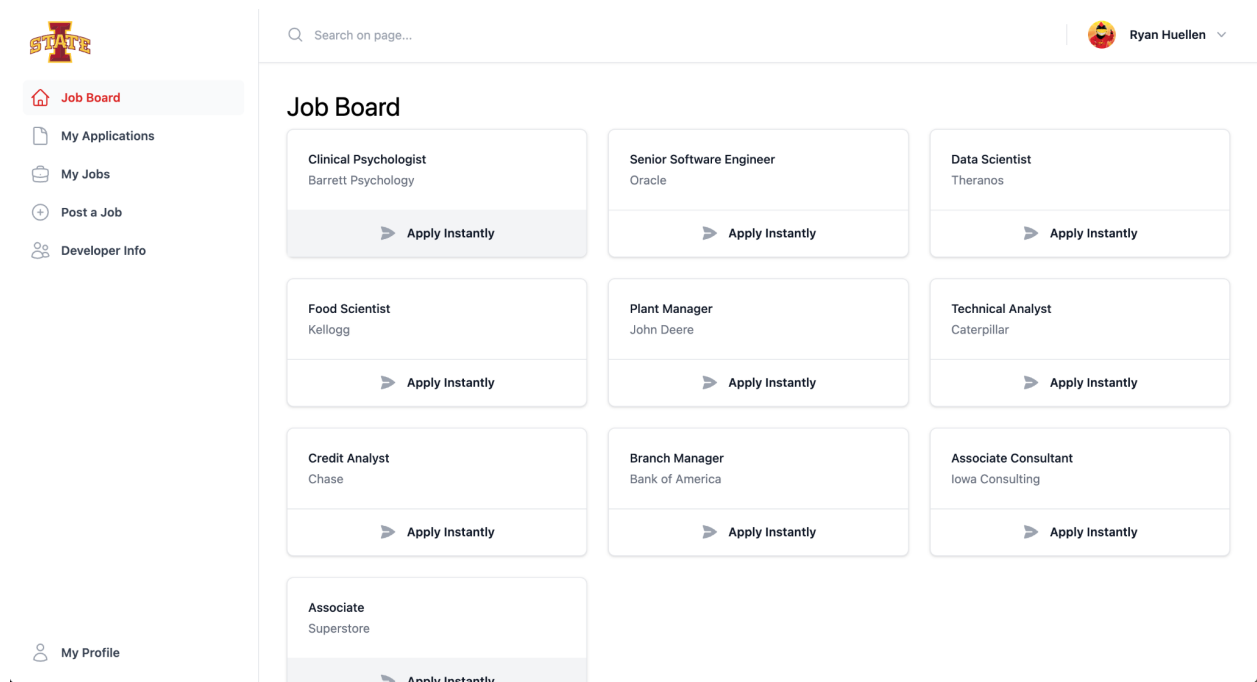
API Design

/jobposts/[id]/applications	GET	Get a specific job post's applicants. This functionality is locked down to the employer who originally created the job post.
/jobposts/[id]/applications	POST	Apply for a specific job.
/auth/signin	POST	Sign in to your account. This endpoint will create a JWT token for a session that is good for 24 hours.
/applications/[id]	PUT	Update a specific application. This is called by applicants when they withdraw their application and by employers when they update an application's status.
/applications/[id]	DELETE	Delete an application in its entirety, withdrawing it. This endpoint is included for completeness, but not actually accessible from the user interface.

Web Views

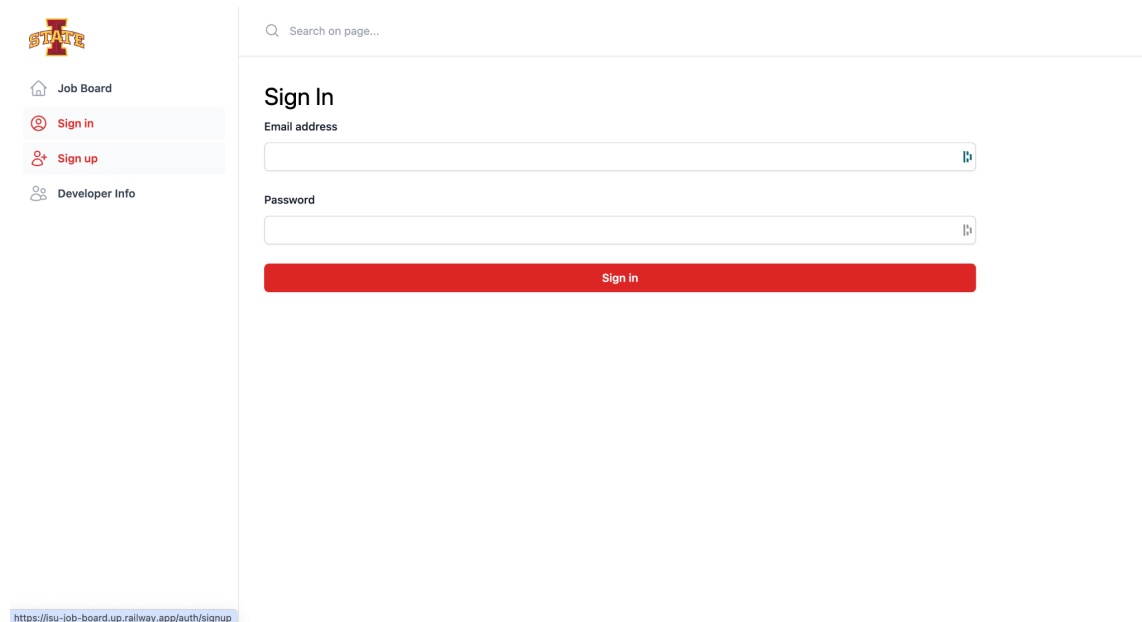
The application contains eight different views, each bolstering differing functionality.

View #1: Job Board:



The Job Board is the main page of the web application. Here, users can navigate between pages, view their information (if signed in), view active job postings, and instantly apply to jobs they're interested in.

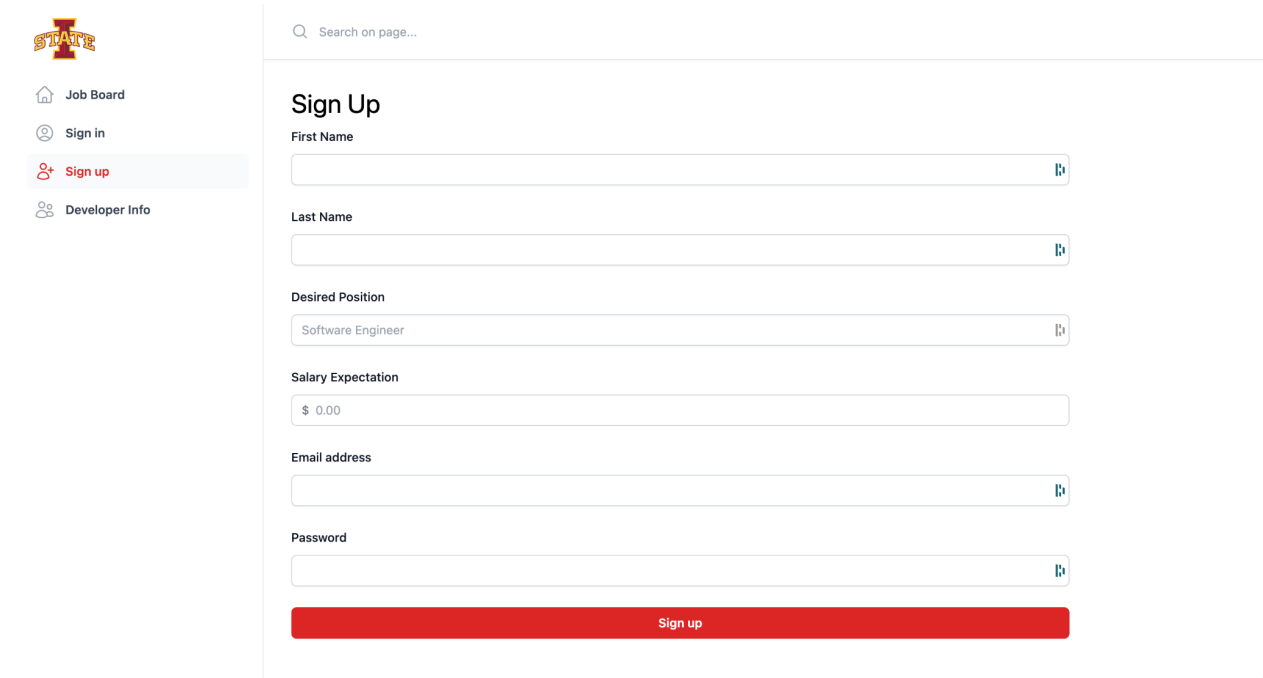
View #2 Sign In:



Web Views

On the sign in page, users can enter their credentials for an existing account. Upon sign in, users will be redirected to the job board.

View #3: Sign up

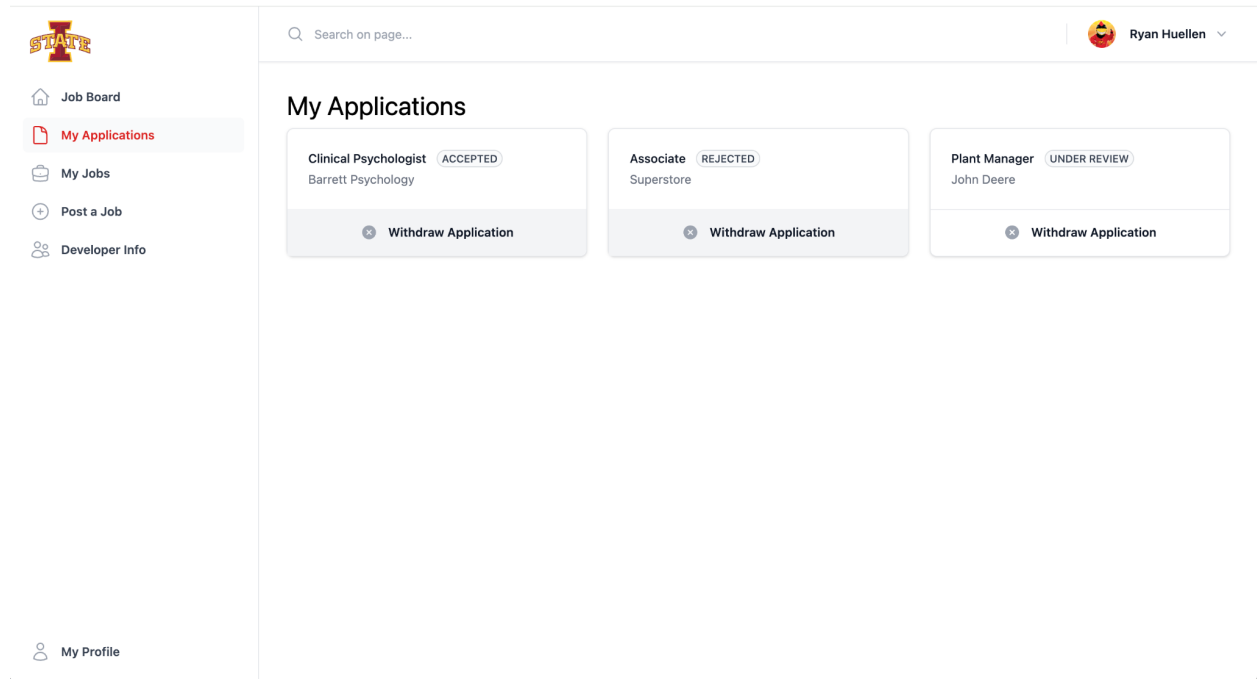


The screenshot displays a web application interface for signing up. On the left is a vertical sidebar with a logo at the top and four menu items: 'Job Board', 'Sign in', 'Sign up' (which is highlighted with a red background), and 'Developer Info'. The main content area has a search bar at the top. Below it, the 'Sign Up' section contains several form fields: 'First Name', 'Last Name', 'Desired Position' (with 'Software Engineer' entered), 'Salary Expectation' (with '\$ 0.00' entered), 'Email address', and 'Password'. Each text input field has a small icon on the right side. At the bottom of the form is a prominent red button labeled 'Sign up'.

Here, users have the option to create a new account by entering a first name, last name, desired position title, salary expectation, email address, and password. Upon signing up, the user's account is created and their password is hashed. From here, users can sign in as they wish.

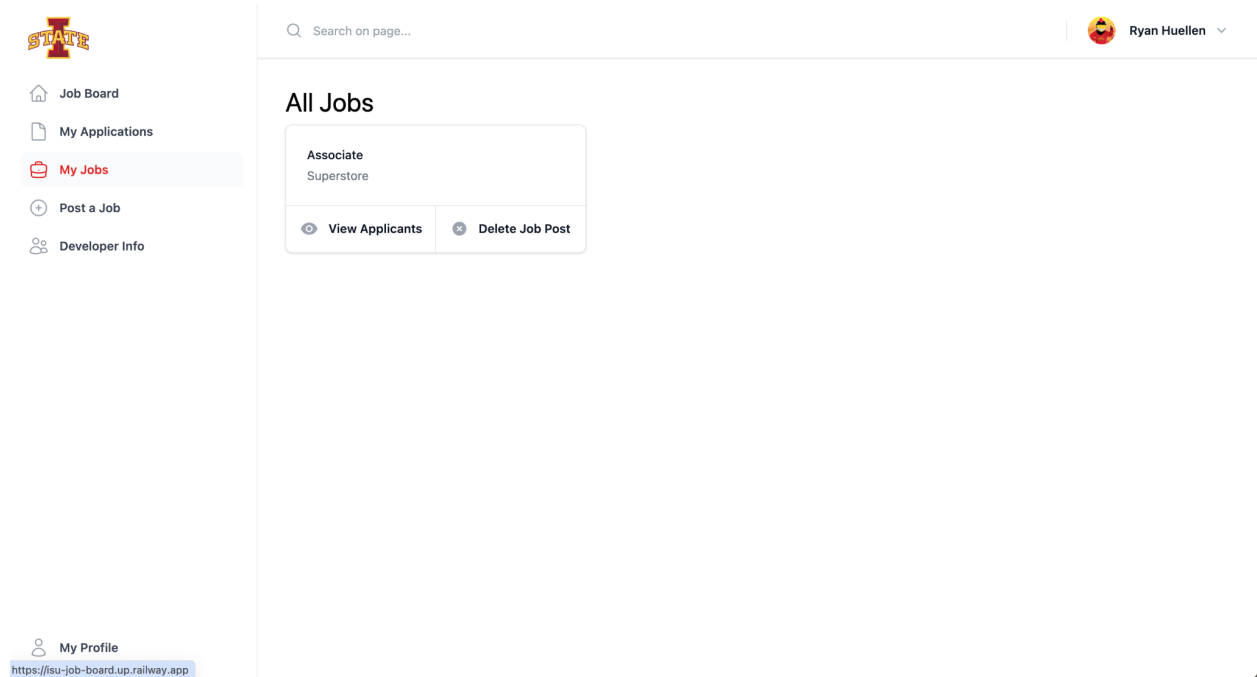
Web Views

View #4: My Applications



If a user has applied for a position on the main job board, that same user can view those applications on the My Applications page. Here, users have the option to see the status of their current applications, or withdraw their application if it's still under review.


View #5: My Jobs




Web Views

If a user is an employer and posted a job, they can see all their jobs on the “My Jobs” page. Moreover, they can view applicants for a specific job, or delete the job post if they have already found the right candidate to fill the position.


View #6: Job Applicants



- Job Board
- My Applications
- My Jobs
- Post a Job
- Developer Info

 My Profile

Search on page...

 Ryan Huellen

Job Applicants

Associate at Superstore

Name	Desired Position	Salary Expectation	Application Status	Available Actions
Ryan Huellen	Waterslide Tester	\$10222	REJECTED	
Jason Barrett	Psychologist	\$5	WITHDRAWN	

Once applicants have begun applying to your specific job posting, you can view their applications all on one screen. Here, you can view each applicant’s name, their desired position title, salary expectation, and you also have the ability to accept or reject specific applicants. On this page, you can also see if a specific applicant has withdrawn their application.

Web Views

View #7: Post a New Job

STATE

Job Board

My Applications

My Jobs

Post a Job

Developer Info

My Profile

Search on page...

Ryan Huellen

New Job

Company name

Microsoft

Position

Data Engineer

Post job

If you're an employer and have a new opening, you can post a new job by entering the company's name and position they're hiring for. Upon submission, the job post will automatically be available to all users of the job board.

View #8: Developer Information

STATE

Job Board

My Applications

My Jobs

Post a Job

Developer Info

My Profile

Search on page...

Ryan Huellen

Course Details

Course Name: SE/ComS319 Construction of User Interfaces, Fall 2023

Date: Nov 30, 2023

Professor Information

Dr. Abraham N. Aldaco Gastelum

Email: aaldaco@iastate.edu

Student Information

Name: Ryan Huellen

Email: rhuellen@iastate.edu

Name: Jayden Luse

Email: jcluse@iastate.edu

This page is available to all users and allows applicants and employers to see details regarding the application's development and attain support via. e-mail.

Installation Manual

The ISU Student Job Board's project code is available both in an attached .zip on Canvas, and on GitHub: <https://github.com/ryanhaticus/isu-job-board>. For the foreseeable future, the code will continue to be provided on this GitHub repository under an MIT license, meaning any developers in the future can copy the code, modify it, and redistribute it as they see fit.

As such, anyone can download our code and install it on their own machine. In order to run the application, however, a few preliminary installations are required.

First and foremost, our application heavily relies on Next.js. Next.js is powered by Node.js, the underlying server-side runtime for JavaScript. At this time, we have verified our application works on Node.js versions 18 and up. In order to install Node.js, you can visit their official website: <https://nodejs.org/en>. Upon doing this, you can run `npm install` inside the project's root directory. After 30 or so seconds (depending on your WiFi connection speed), you will have all the dependencies required to run the job application.

To continue, our application stores user information including profiles, job posts, and applications in MongoDB. MongoDB is a specialized NoSQL database engine that ensures our application loads quickly as it scales to thousands of users. To install and set up your own MongoDB server locally, you will need MongoDB Community Server, available here: <https://www.mongodb.com/try/download/community>. When setting up the database, you can use most of the defaults. However, please pay attention to your credentials! You will need this in the next step.

Luckily, because we're using Mongoose, the schemas will automatically be created by the Next.js application the first time you use them. However, you will need to tell the Next.js application how to connect to your database. You will do this by creating a `.env.local` file, or an environment file. Here, you will set two variables.

```
❏ .env.local
1  MONGO_URL=mongodb://username:password@127.0.0.1:27017
2  JWT_SECRET=mysecret|
```

The first variable, you should replace `username` and `password` with the respective credentials you created when setting up MongoDB Community Server. For the second variable, you may enter any unique passphrase. This passphrase will be responsible for ensuring user sessions stay secure.

After performing all the steps listed above, you can start the application's development server by running `npm run dev`. From here, navigate to <http://localhost:3000> in the browser to get started!