

CS 157A Final Project Report

Kevin Tsoi
016999879
kevin.tsoi@sjsu.edu

Samson Xu
016959371
samson.xu@sjsu.edu

Huy Duong
017083716
huy.n.duong@sjsu.edu

Marvin Zhai
017076930
marvin.zhai@sjsu.edu

Goals & Overview

Our goal is to facilitate meaningful connections among San Jose State University (SJSU) students, alumni, and professors. University is a difficult endeavor for many people, and a place for professional networking, mentorship, and collaboration would benefit both students and graduates by helping them meet others. Alumni and staff have connections and opportunities that are helpful to students, but these are often difficult to find on existing platforms like Handshake which are flooded with outside opportunities that have a low response rate. Our website, Spartan Outreach, will provide a centralized and SJSU-exclusive social media platform to solve these problems.

Users on our website can sign up using an SJSU email address and password and filling out information such as graduation year and current status. Currently, only students can sign up without approval to keep the site secure. After logging in with their credentials, students can view an alumni wall to connect and view contact information. On the posts page, anyone can share their stories or experiences, similar to a social media website. There will also be pages for events, job opportunities, and fundraisers. Here, alumni and staff can post jobs they are hiring for, advertise research positions, and invite others to events. Anyone can donate, RSVP, and apply externally to these listings. With all these new tools available, we hope that students, alumni, and staff can make meaningful connections and advance their careers.

Functional Requirements

Users must sign up and login with an email and password. They must enter a first and last name and graduation year. Optionally, they can include a major, degree, and graduation month. By default, only students with a SJSU email address can sign up, and alumni and staff need approved accounts with higher permission levels. We decided to remove admin accounts as they are the same as staff. We also did not implement a self account deletion feature as we thought an application would be more safe if potential scammers and hacked accounts can't instantly delete their own account and all associated information.

For the alumni wall, we let alumni accounts optionally add themselves to be displayed on the alumni wall with their current company, industry, and external contact links. This is better since not all alumni want to be publicly visible. We added additional fields so that the alumni wall can be filtered by company, industry, graduation year, and major. Any user can click on their profile and send a connection request and further contact them off-site through contact links if desired. Connections can be pending, accepted, or not exist. Rather than having a chat feature, we opted for external links because using existing message platforms like LinkedIn would result in more active replies than having another site to check on. To let alumni view connections, there is a My Connections page showing all incoming or accepted connections.

Initially, posts would be a superset of jobs, events, and fundraisers, but we decided to separate them because of varying permission levels and their unique uses. Posts can be created by anyone, and contain title, text, and possible media URLs (URL, photo, or video). We made media displayed as a clickable link rather than stored on our machine, as it would make the GitHub file size too large and unscalable. All posts will be displayed and

are able to be filtered by substrings in the title. Users can like or unlike a post at most once and add multiple textual comments. A total like count will be shown and the full list of comments will be accessible in a dropdown.

Staff can create fundraisers to raise money. They can set a name, goal, description, and end date. Donations are not validated but are tracked on the website to show total donations and the names of all the donors. Users can donate multiple times. Staff can also create events for others to join. They can input a name, date, description, and an optional address. Anyone can RSVP or UnRSVP, and the list of attendees will be publicly visible in a toggle dropdown alongside a total attendee count. Both alumni and staff can post job postings. These will contain a job title that can be searched, a description, and an external URL to apply. The forms to create fundraisers, events, and jobs are only visible to those with the appropriate permissions.

Other features we felt were not as useful were post sharing, Spartan of the Month, alumni donations, and admin activity logs. Because of likes and comments, having post sharing was not really needed to show the popularity of a post. Alumni donations were combined with normal donations. Spartan of the Month and the admin panel felt out of scope of this project because we do not have an accurate way to track student achievements and activities accurately besides potentially the number of posts they make. As mentioned above, we included connections with detailed statuses, implemented external contact links, and added more details to events, jobs, and fundraisers.

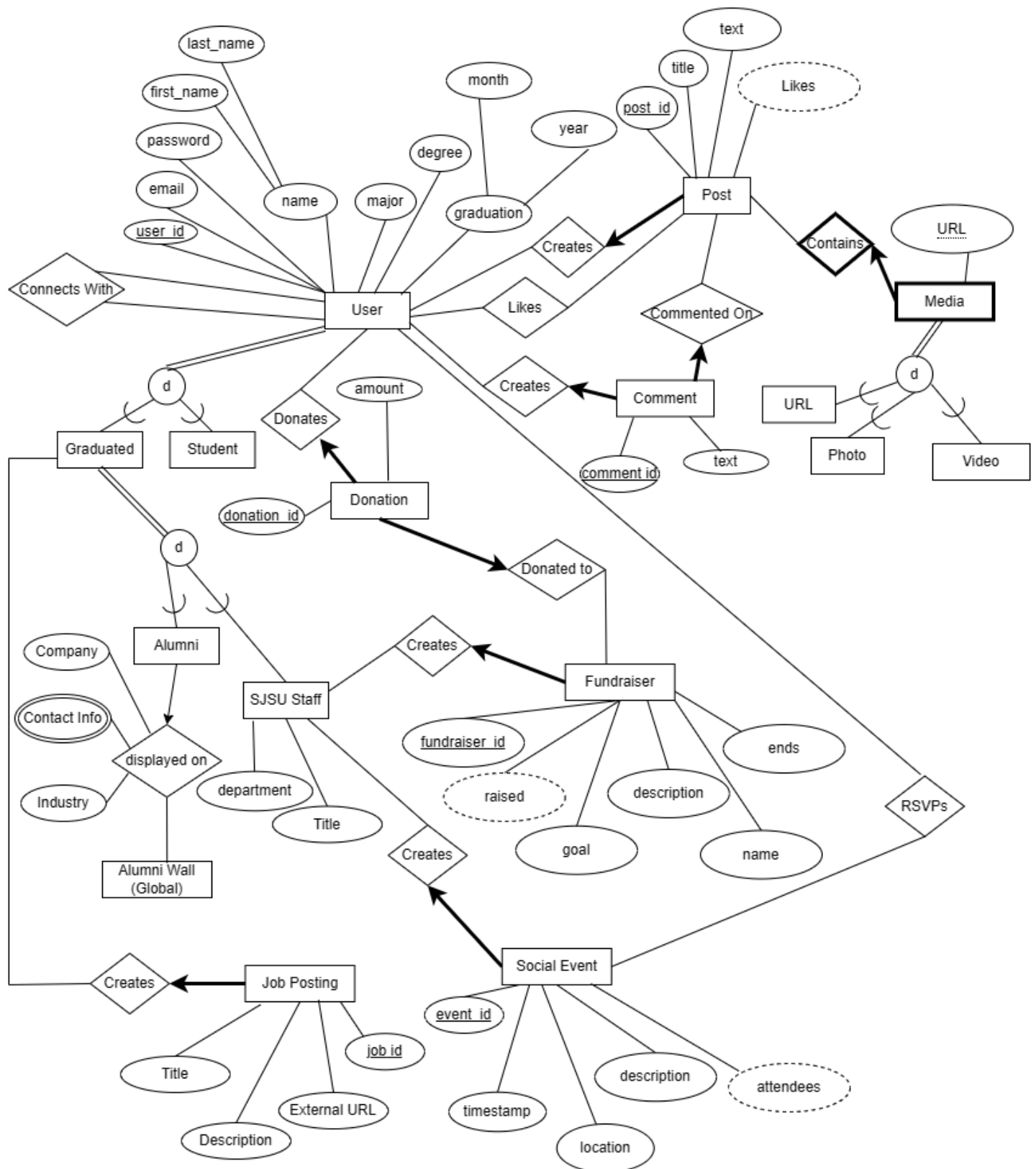
Architecture

For our project, we aimed to follow a three-tier architecture with MVC (Model-View-Controller) principles. Our model is MySQL, which we used as our database. The view is the React frontend. The controller is the Django (Python) backend where we serve our API endpoints. This separated the responsibilities cleanly and allowed for an easier way to manage the many pieces of our website. To facilitate collaboration across our different OS and to make running the application easier, we used Docker to have isolated containers for React, MySQL, and Django.

Although Django is known for being a monolithic solution with its built in ORM and ability to render pages, we did not use these functionalities. We wrote our own SQL statements for all database manipulations. Users interact with frontend React components, which use APIs to send data to Django. The Django API endpoints process the data and run SQL queries in the MySQL database and return result messages back to the frontend to update the display.

SQL table creation statements and initial data insertion can be found in `/db/init.sql`. The rest of our query statements and manipulation statements are in `/server/api/views.py`. In general, `/client` contains all the frontend code while `/server` contains the backend.

ER Design



DB Design

Initial Relational Schema

We created our initial tables based off of the ER diagram and used underlines to show primary keys. Foreign key relationships are described in bullet points. Data types will be defined in the SQL. We ended up with 13 tables. To conform to relational design, we already split some atomic values that will be discussed in 1NF.

User(userID, email, password, first, last, major, degree, gradMonth, gradYear, type, department, title)

Connection(user1, user2)

- user1 is FK to User: userID
- user2 is FK to User: userID

AlumniWall(user, company, industry)

- user is FK to User: userID

AlumniContact(user, url)

- user is FK to User: userID

Post(postID, user, title, text)

- user is FK to User: userID

Media(post, URL, type)

- post is FK to Post: postID

Like(user, post)

- user is FK to User: userID
- post is FK to Post: postID

Comment(commentID, user, post, comment)

- user is FK to User: userID
- post is FK to Post: postID

Fundraiser(fundraiserID, creator, goal, description, ends, name)

- creator is FK to User: userID

Donation(donationID, fundraiser, user, amount)

- user is FK to User: userID
- fundraiser is FK to Fundraiser: fundraiserID

SocialEvent(eventID, name, creator, timestamp, street, state, city, ZIP, description)

- creator is FK to User: userID

RSVP(user, event)

- user is FK to User: userID
- event is FK to SocialEvent: eventID

JobPosting(jobID, creator, URL, description, title)

- creator is FK to User: userID

Functional Dependencies:

We omitted 4 tables such as Connection(user1, user2) that only have trivial or redundant dependencies like $\text{user1, user2} \rightarrow \text{user1, user2}$.

User

- $\text{userID} \rightarrow \text{email, password, name, major, degree, graduation, type, department, title}$
- $\text{email} \rightarrow \text{userID, password, name, major, degree, graduation, type, department, title}$

AlumniWall

- $\text{user} \rightarrow \text{company, industry}$

Post

- $\text{postID} \rightarrow \text{user, title, text}$

Media

- $\text{post, URL} \rightarrow \text{type}$

Comment

- $\text{commentID} \rightarrow \text{user, post, comment}$

Fundraiser

- $\text{fundraiserID} \rightarrow \text{creator, goal, description, ends, name}$

Donation

- $\text{donationID} \rightarrow \text{fundraiser, user, amount}$

SocialEvent

- $\text{eventID} \rightarrow \text{name, creator, timestamp, location, description}$

JobPosting

- $\text{jobID} \rightarrow \text{creator, URL, description, title}$

Candidate Keys:

User

- userID
- email

Connection

- user1, user2

AlumniWall

- user

AlumniContact

- user, url

Post

- postID

Media

- post, url

Like

- user, post

Comment

- commentID

Fundraiser

- fundraiserID

Donation

- donationID

SocialEvent

- eventID

RSVP

- user, event

JobPosting

- jobID

1NF

- To be in 1NF, all attributes have to be atomic (indivisible) values. We made sure this was satisfied before making the relation design by splitting User's graduation to gradMonth and gradYear. We had also split User's name to first and last. Finally, we also previously split SocialEvent's location to Street, City, State, and ZIP.

User(userID, email, password, first, last, major, degree, gradMonth, gradYear, type, department, title)

Connection(user1, user2)

AlumniWall(user, company, industry)

AlumniContact(user, url)

Post(postID, user, title, text)

Media(post, URL, type)

Like(user, post)

Comment(commentID, user, post, comment)

Fundraiser(fundraiserID, creator, goal, description, ends, name)

Donation(donationID, fundraiser, user, amount)

SocialEvent(eventID, name, creator, timestamp, street, state, city, ZIP, description)

RSVP(user, event)

JobPosting(jobID, creator, URL, description, title)

2NF

- To be in 2NF, every non-candidate key attribute has to be fully functional dependent on the whole candidate key on top of satisfying 1NF. All our tables satisfy this already because all of our candidate keys are singular, so no non-candidate key attribute can be partially dependent on a subset. The only

exception is post, URL \rightarrow type in Media, but a post or URL alone can't determine the type of Media accurately (a URL could be used as a URL type in one post and as a Photo type in another).

There is no change from the 1NF design.

3NF

- To be in 3NF, no non-candidate key attributes are transitively dependent on a candidate key on top of satisfying 2NF. All our tables satisfy this already because all of the left hand sides of the FDs are candidate keys. There are no transitive dependencies originating from a non-candidate key to another non-candidate key.

There is no change from the 1NF design.

BCNF

- To be in BCNF, every functional dependency $X \rightarrow Y$ must have X as a candidate key, on top of satisfying 3NF. Since our schema already satisfies 3NF, and every X is a candidate key, this means that Boyce-Codd Normal Form is satisfied for our schema as well.

There is no change from the 1NF design.

Major Design Decisions

In our database design, we started with having auto-incrementing primary keys on most tables to ensure minimal conflicts. For example, post ID 1 and 2 can separate two posts with the same title and text. This also led to faster queries on joins when fetching something like the number of likes on a PostID due to simple and unique foreign key references. MySQL automatically indexed our PKs and FKs, but we added additional indexes on search fields like graduation year. We used FULLTEXT indexes on text search fields like job titles because normal b-tree indexes do not work well with wildcard (%) selectors at the beginning of queries like %intern%. Additionally, we used transactions when inserting data to multiple tables in one logical unit. For example, inserting a Post and its list of Media had to be in a transaction to prevent any Media from being created if another Media or the Post itself did not properly insert.

In terms of architecture, our first major design decision was using Docker containers because it facilitated standardized development across Mac and Windows and made running the program easier when grading because there is only one dependency. Next, we set up the MySQL, Django, and React containers. We included a init.sql script to automatically populate the database with the tables and sample data whenever MySQL creates a new one. We started coding with the three-tier MVC architecture in mind for flexibility and separation of concerns for the frontend, backend, and database. One obstacle was knowing if users were logged in and the status of their authentication. We added authentication middleware in Django to fetch their login status from

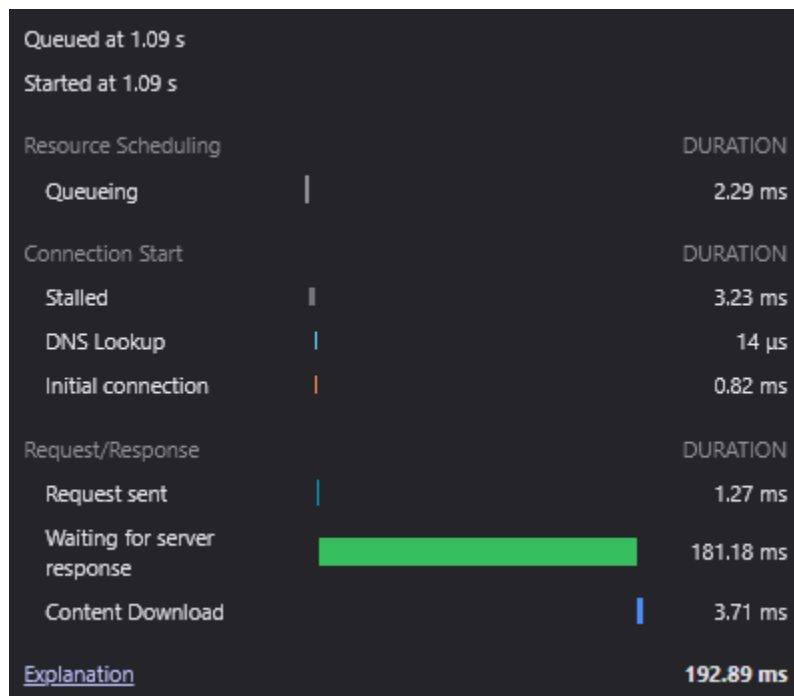
their headers and issue JWT tokens on login. These were stored in the browser's local storage so their session persisted without having to log in frequently.

Non-functional Requirements

Our first non-functional requirement was security. We satisfied this by limiting signups to @sjsu.edu emails and making alumni and staff accounts require approval or manual creation by staff. Passwords were stored securely with bcrypt hashing (shown below) so any data breach would not have plain text passwords. We did not have chats but allowed users to contact each other securely off-site through external links.

```
('student6@sjsu.edu', '$2y$10$pNq5F5ZRyWwk6Ecb9lIYEuQrJ/7Co.m0YfQEzNXzC6nviSfkVpZdm',  
'alumni6@sjsu.edu', '$2y$10$pNq5F5ZRyWwk6Ecb9lIYEuQrJ/7Co.m0YfQEzNXzC6nviSfkVpZdm',  
'staff6@sjsu.edu', '$2y$10$pNq5F5ZRyWwk6Ecb9lIYEuQrJ/7Co.m0YfQEzNXzC6nviSfkVpZdm', '
```

Next, we met the requirement of performance by reducing redundant API calls. Rather than fetching the comments for every single post, we made them only make the API call to fetch when the dropdown was opened. This prevented a spam of 100s of API calls every time you switch a page. All our API responses also take an average of less than 1 second (Post fetching time is shown below). We did not add caching as it is out of the scope of this course.



In terms of usability, our UI was designed intuitively so that all users can clearly log in, sign up, and navigate through pages using the navbar. Non-technical users can easily understand it and use the search features as well as alumni connections. We did not add progress bars since it is not in the scope of this course but we included descriptive errors. More can be seen in the Demonstration section.

Two non-functional requirements, backup & recovery and payment gateway integration, were completely removed. We had to use MySQL so we felt additionally integrating cloud backups was out of the scope of this course. Docker data is persisted from each startup and backups and recovery are handled by the MySQL DBMS. Payment integration was also out of scope because it leaned much more towards web development than database queries.

As we continued with the application, we encountered more complex queries and created the new non-functional requirement of scalability and fast query execution time. To handle scalability, we used transactions for multiple inserts in a row like a post having multiple media. If the API timeouts, the transaction prevents one media being committed while its corresponding post was rolled back. We also effectively used FKs and PKs to facilitate the relation design, and these are auto indexed by MySQL. Additionally, we created our own indexes on the fields like User graduation year, Fundraiser ends, Donation amount, and Event timestamp. This made the sorting through ORDER BY fast. We used fulltext indexes on the text search fields Post title, User major, Job title, Alumni company, and Alumni industry. This facilitated search filtering through WHERE. In the picture below, you can see the use of EXPLAIN shows the first query using the auto FK index on field user. The second uses the fulltext index on post title to ensure quick inserts, especially if the application scales to thousands of posts. The second picture shows key=NULL when searching for Post text, which we did not implement and therefore did not create and index. As you can see, it does not use an index and will take longer.

```
44 CREATE TABLE IF NOT EXISTS Post (  
45     postID INT PRIMARY KEY AUTO_INCREMENT,  
46     user INT NOT NULL,  
47     title VARCHAR(255) NOT NULL,  
48     text TEXT,  
49     FOREIGN KEY (user) REFERENCES User(userID) ON DELETE CASCADE  
50 );  
51  
52 CREATE FULLTEXT INDEX idx_post_title ON Post(title);  
53
```

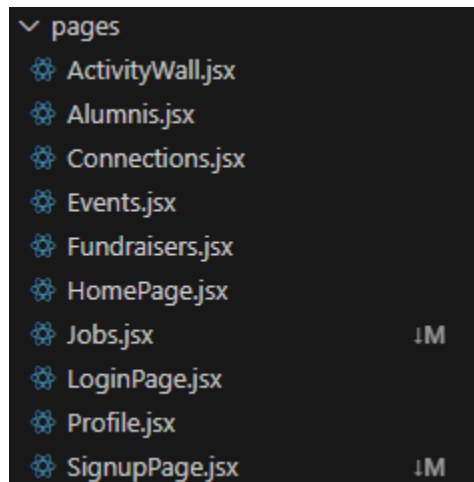
PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE COMMENTS

```
mysql> EXPLAIN SELECT * FROM Post WHERE user = 1;  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
| 1 | SIMPLE | Post | NULL | ref | user | user | 4 | const | 1 | 100.00 | NULL |  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
1 row in set, 1 warning (0.00 sec)
```

```
mysql> EXPLAIN SELECT * FROM Post WHERE MATCH(title) AGAINST('job*');  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
| 1 | SIMPLE | Post | NULL | fulltext | idx_post_title | idx_post_title | 0 | const | 1 | 100.00 | Using where; Ft_hints: sorted |  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
1 row in set, 1 warning (0.01 sec)
```

```
mysql> EXPLAIN SELECT * FROM Post WHERE text LIKE '%maintenance%';  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
| 1 | SIMPLE | Post | NULL | ALL | NULL | NULL | NULL | NULL | 1 | 100.00 | Using where |  
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+  
1 row in set, 1 warning (0.01 sec)
```

Another main non-functional requirement we added was maintainability. To make sure the application was easy to update, we have the client, db, and server folders to clearly separate functionality. Each page was divided into its own React component (shown below) for easy readability so people will know where to look for each component based on clear file names. We also made use of functions to separate code cleanly. For the init.sql, the create table statements are matched with their indexes and the insertions are towards the bottom so they are easy to find and edit. In the backend, all the API routes are placed in one file to get an easy overview of all routes (shown below). The endpoint controllers are stored in one file for easy access to view all SQL queries. By organizing our code well, we made sure it's easy to edit and maintainable for updates to both queries and frontend in the future



```
urlpatterns = [
    path('login/', login_view, name='login'),
    path('register/', register_view, name='register'),

    path('users/', UsersView.as_view(), name='all users'),
    path('users/<int:id>/', user_view, name='user'),

    path('connections/', connections_view, name='all connections'),
    path('connections/<int:id>/', ConnectionView.as_view(), name='connection'),

    path('alumni/', AlumniView.as_view(), name='alumni'),
    path('contacts/<int:id>/', contact_view, name='get contacts'),
```

Implementation Details

To start, we created docker-compose.yml to have containers for the database, client, and server. Docker volumes allow for persisted MySQL data between restarts. The React and MySQL containers automatically start when you run the Docker command. We used a healthcheck to make sure MySQL is accepting connections before initializing Django.

MySQL runs the /db/init.sql file when it starts for the first time. This runs all the CREATE and INSERT statements with initial data such as testing accounts and posts. Here, we made use of PK, FK, INDEX, FULLTEXT INDEX, and more constraints to create a database that satisfies BCNF. Indexes ensured future queries would be optimized and relational constraints ensure unique entries. The picture below shows example insertions and a table created.

```
125 CREATE TABLE IF NOT EXISTS JobPosting (  
126     jobID INT PRIMARY KEY AUTO_INCREMENT,  
127     creator INT NOT NULL,  
128     URL VARCHAR(255) NOT NULL,  
129     description TEXT,  
130     title VARCHAR(255) NOT NULL,  
131     FOREIGN KEY (creator) REFERENCES User(userID) ON DELETE CASCADE  
132 );  
133  
134 CREATE FULLTEXT INDEX idx_job_title ON JobPosting(title);  
135  
136 DELIMITER //  
137  
138 CREATE PROCEDURE PopulateInitialData()  
139 BEGIN  
140     START TRANSACTION;  
141  
142     IF (SELECT COUNT(*) FROM User) = 0 THEN  
143         INSERT INTO User (email, password, first, last, major, degree, gradMonth, gradYear, type, department, title)  
144         VALUES  
145         ('student1@sjsu.edu', '$2y$10$pNq5F5ZRYWmk6Ecb91IYEuQrJ/7Co.mOYfQEzNXzC6nviSfkVpZdm', 'John', 'Adams', 'Computer Science', 'MS', 12, 2028, 'student', NULL, NULL),  
146         ('student2@sjsu.edu', '$2y$10$pNq5F5ZRYWmk6Ecb91IYEuQrJ/7Co.mOYfQEzNXzC6nviSfkVpZdm', 'Bob', 'Carter', 'Biology', 'BS', 12, 2025, 'student', NULL, NULL),  
147         ('student3@sjsu.edu', '$2y$10$pNq5F5ZRYWmk6Ecb91IYEuQrJ/7Co.mOYfQEzNXzC6nviSfkVpZdm', 'Katie', 'Evans', 'Mathematics', 'PhD', 5, 2028, 'student', NULL, NULL),  
148         ('student4@sjsu.edu', '$2y$10$pNq5F5ZRYWmk6Ecb91IYEuQrJ/7Co.mOYfQEzNXzC6nviSfkVpZdm', 'Elsa', 'Perez', 'Mechanical Engineering', 'BS', 12, 2025, 'student', NULL, NULL),  
149         ('student5@sjsu.edu', '$2y$10$pNq5F5ZRYWmk6Ecb91IYEuQrJ/7Co.mOYfQEzNXzC6nviSfkVpZdm', 'Tiffany', 'Hill', 'Mathematics', 'PhD', 12, 2029, 'student', NULL, NULL),  
150     ;  
151     COMMIT;
```

To connect to MySQL, Django uses connection details such as the user, password, and port to maintain a connection. After the connection is made, it starts accepting API requests through the endpoints found in /server/api/urls.py. It only handles valid HTTP requests at valid paths with supported methods like GET, POST, and DELETE. Routing is done in this file, and the handling is left for /server/api/views.py. Each function or class here takes the request and possibly parameters or request bodies and headers. Using the provided data, it can execute SQL statements by plugging in variable values and getting the result through a cursor managed by Django. The first two pictures below show a SELECT and INSERT statement example. Authentication for these endpoints is handled through a middleware. Login is verified by using bcrypt to compare a plaintext password with the hash. The third picture shows the creation of JWT tokens to store sessions.

```
with connection.cursor() as cursor:  
    cursor.execute("""  
        SELECT f.fundraiserID, f.name, f.goal, f.description, f.ends,  
        COALESCE(SUM(d.amount), 0) AS raised  
        FROM Fundraiser f  
        LEFT JOIN Donation d ON f.fundraiserID = d.fundraiser  
        GROUP BY f.fundraiserID  
        ORDER BY f.ends  
    """)  
    rows = cursor.fetchall()
```

```

with connection.cursor() as cursor:
    cursor.execute("BEGIN;")

    cursor.execute("""
        INSERT INTO AlumniWall (user, company, industry)
        VALUES (%s, %s, %s)
    """, [user, company, industry])

    for url in contacts:
        if url:
            cursor.execute("""
                INSERT INTO AlumniContact (user, url)
                VALUES (%s, %s)
            """, [user, url])

    cursor.execute("COMMIT;")

```

```

user_id, password_hash, permission_level = user

if not bcrypt.checkpw(password.encode(), password_hash.encode()):
    return JsonResponse({"error": "Invalid credentials"}, status=401)

payload = {
    "user_id": user_id,
    "permission_level": permission_level,
    "iat": datetime.datetime.utcnow(), # Issued at
}
token = jwt.encode(payload, SECRET_KEY, algorithm="HS256")

return JsonResponse({"token": token, "permission_level": permission_level})

```

The last component is the React frontend. In `/client/src/App.jsx`, you can find all the frontend routes that are managed by React Router. The component codes are found in `/client/src/pages`. Each page uses the JavaScript `fetch()` API to send HTTP requests to the Django backend. It takes in the result in JSON format and parses it for either an error or the desired response. The first three pictures below show an example fetch request from React and two example JSON responses from the backend.


```
18 const fetchJobs = async () => {
19   try {
20     const response = await fetch(
21       `http://localhost:8000/jobs/?searchQuery=${encodeURIComponent(
22         searchQuery
23       )}`,
24       {
25         method: "GET",
26         headers: { Authorization: `${token}` },
27       }
28     );
29
30     const data = await response.json();
31
32     console.log(data);
33     if (data.error) {
34       setWallError(data.error);
35       if (data.error === "Invalid token") {
36         setWallError("You must be logged in.");
37       }
38     } else {
39       setJobs(data.jobs);
40     }
41   } catch (error) {
42     console.error("Error fetching jobs data:", error);
43   }
44 };
```

X Headers Payload Preview Response

1 |{"error": "Invalid token"}

```
1 {
-   "jobs": [
-       {
-           "jobID": 1,
-           "title": "Software Engineer Intern",
-           "URL": "https://jobs.cisco.com/jobs/ProjectDetail",
-           "description": "Looking for a software engineer",
-       },
-       {
-           "jobID": 2,
-           "title": "Marketing Manager",
-           "URL": "https://wellfound.com/jobs/3138923-growt",
-           "description": "Marketing manager position avail",
-       }
-   ],
-   "query": ""
- }
```

Demonstration

 SpartanOutreach_Demo.mp4

This is a zoomed out **Sign Up** page where users can enter email, password, and more details.

Activity WallJobsEventsFundraisersAlumniMy ConnectionsLogin

Sign Up

Email

Enter your email

Password

Enter your password

First Name

Enter your first name

Last Name

Enter your last name

Major

Enter your major

Degree Type

BS

Graduation Month

January

Graduation Year

2024


User Type

Student

Sign Up

Already have an account? Login

This is the **login** page where users enter their email and password.

Activity WallJobsEventsFundraisersAlumniMy ConnectionsLogin

Login

Email

Enter your email


Password

Enter your password

Login

Don't have an account? [Sign up](#)

This is the **Activity Wall**, which users will be redirected to once they have logged in. Users can visit other locations through the navigation bar. They can create posts with a title, description, and media. They can search by title. They can like, unlike, and comment on posts.

Activity WallJobsEventsFundraisersAlumniMy ConnectionsLogout

Posts

Post Title

Post Description

Create Post

Enter media URL

URL

Add

Search by Post Title

Search

My Second Post

Author: Bob Carter

This is my second post!

photo: <https://picsum.photos/seed/def/600>

3 likes

Unlike

 Hide

John Adams: Hello again!

Enter Comment

Add

Maintenance

Author: Mary Johnson

There will be maintenance for Spartan Outreach this weekend!

This is the **Jobs Page**, where users are able to view the currently listed jobs. Jobs have a title, application link, and description. You can search by title and apply at the external URL. Only Alumni and Staff see the job adding form at the top.

Activity Wall

Jobs

Events

Fundraisers

Alumni

My Connections

Logout

Jobs

Job Title

Job URL

Job Description

Add Job

Search by Job Title

Search

Software Engineer Intern

<https://jobs.cisco.com/jobs/ProjectDetail/Software-Engineer-I-Intern-United-States/1427387>

Looking for a software engineer intern for summer 2025.

Marketing Manager

<https://wellfound.com/jobs/3138923-growth-marketing-manager>

Marketing manager position available.

Software Engineer Intern II

<https://jobs.cisco.com/jobs/ProjectDetail/Software-Engineer-I-Intern-United-States/1427387>

Looking for a software engineer intern for summer 2025.

Janitor

<https://google.com>

Cleaning up SJSU.

This is the **Events Page** where you can see the list of all events. Users can RSVP to an event once and undo the RSVP. Users can also see other users who have already registered for an event in a dropdown. Staff users can create events with these fields: name, date, street, city, state, zip code, and a description of the event.

Activity Wall

Jobs

Events

Fundraisers

Alumni

My Connections

Logout

Events

Event Name

mm/dd/yyyy, --:-- --

Street Address

City

State

ZIP Code

Event Description

Add Event

Board Meeting

Attendees: 3

Date & Time: 12/31/2024, 3:00:00 PM

Location: 21st Dorm St, San Jose, CA, 12345

Election for the next committee

RSVP

Hide

Lebron Green

Mary Johnson

David Hall

Picnic

Attendees: 1

Date & Time: 12/14/2024, 8:51:00 PM

Location: 123 Sesame St, San Jose, CA, 95112

Tower Lawn picnic.

Cancel RSVP

Show Attendees

This is the **Fundraisers Page**, where any user can donate to a specific fundraiser. Staff can create fundraisers with the following attributes: name, goal amount, description, and end date. The fundraiser card will have these attributes listed, along with the total amount that has been donated. The donation amount will update automatically with every donation. People who have donated will have their names below.

Activity Wall

Jobs

Events

Fundraisers

Alumni

My Connections

Logout

Fundraisers

Fundraiser Name

Goal Amount

Fundraiser Description

mm/dd/yyyy

Add Fundraiser

AI Research

Goal: \$1234.00

Total Raised: \$495.00

Ends: 2024-12-02

Raising funds for an AI video surveillance camera.

Donation Amount

Donate

Donations:

Tiffany Hill: \$200.00

Bob Carter: \$120.00

Kevin Tsoi: \$100.00

Alex Adams: \$75.00

Research on AI

Goal: \$500.00

Total Raised: \$90.00

Ends: 2024-12-19

Professor B needs funds

Donation Amount

Donate

Donations:

Lebron Green: \$90.00

Food Drive

Goal: \$750.00

This is the **Alumni Page**. If the user is an Alumni or Staff, they can choose to add themselves onto the page by filling out the following fields: company, industry, and a comma separated list of contact information. On the page, Alumni that have added themselves to the wall can be seen. Any user can query and filter alumni by searching by graduation year, major, company, industry, or multiple at a time. Users can also click on a specific alumni leading to their user profile.

Activity WallJobsEventsFundraisersAlumniMy ConnectionsLogout

Alumni Wall

CompanyIndustryContacts (csv)Add Yourself

Search by Grad Year

Search by Major

Search by Company

Search by Industry

Search

Joseph Hill

Graduation Year: 2023

Major: Mathematics

Company: Chase

Industry: Banking

Taylor Walker

Graduation Year: 2022

Major: Computer Science


Company: Google

Industry: Technology

Eddie Young

By clicking on any alumni on the alumni page, this will lead to **their User Profile Page**. This page lists out the contact information that was filled out by the alumni on the alumni wall and other information during signup. There is the option to send out a connection to the Alumni. If you already sent the connection or they have accepted it, the status will change.

Activity WallJobsEventsFundraisersAlumniMy ConnectionsLogout



Joseph Hill

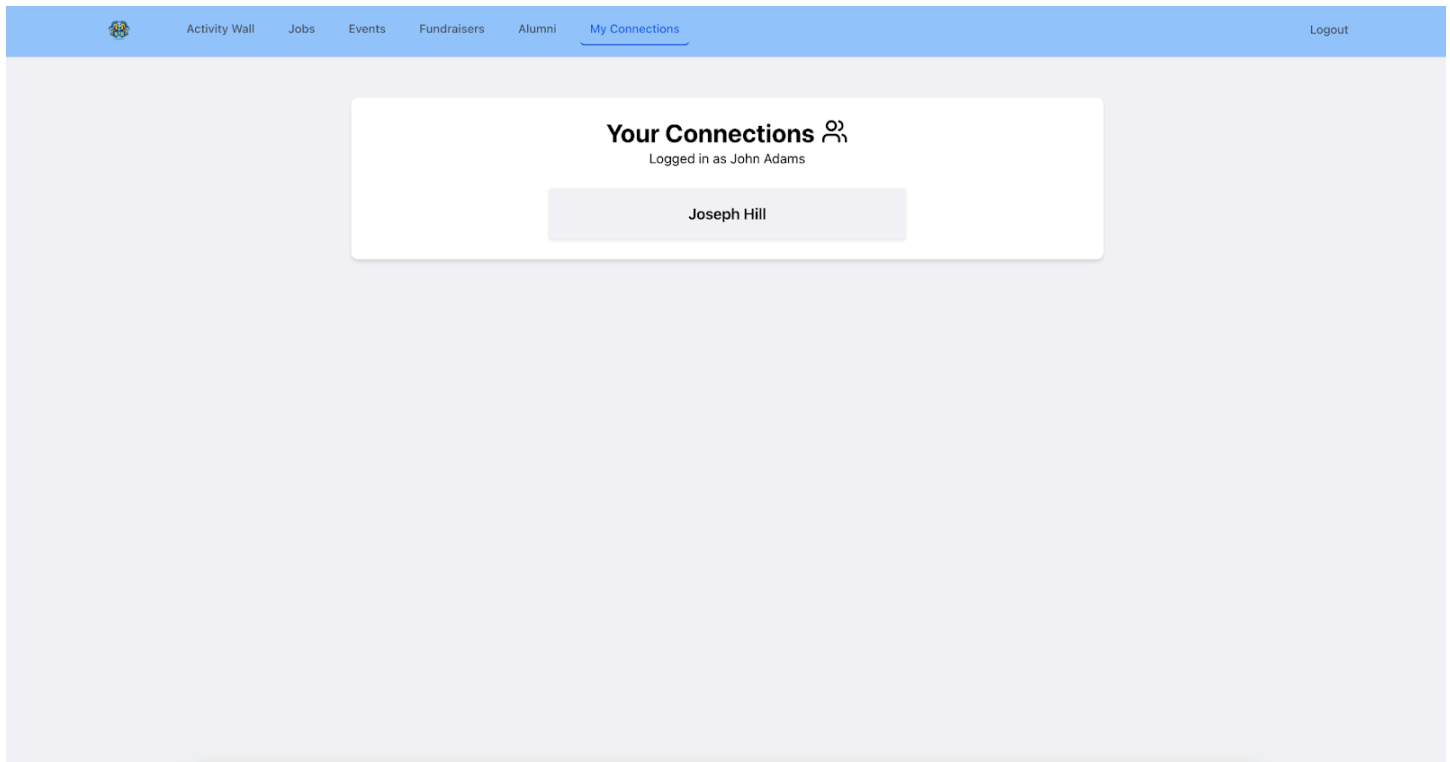
BS Mathematics • Graduated 5-2023

<https://instagram.com/josephhill>

<https://linkedin.com/in/josephhill>

Connect

All users can go to the **My Connections Page**. This shows all incoming connections meaning everyone who has sent you a connection or is currently connected with you. This lets you accept or reject their connections or unconnect if needed.



Conclusion

We successfully created a full-stack web application, Spartan Outreach, using MySQL, React, and Django. The platform meets the outlined functional and architectural requirements along with Boyce Codd Normal Form, providing a centralized hub for SJSU students, alumni, and staff to connect, collaborate, and share opportunities. Key features include secure sign-ups, an alumni wall, filtered searches, posts with media and comments, job postings, event creation, and fundraiser management. These tools collectively address the need for meaningful networking and professional development within the SJSU community.

Through this project, we learned valuable lessons about database normalization, different types of indexing, the importance of primary and foreign keys, and optimizing queries. Additionally, implementing a three-tier architecture with Docker-enabled isolated containers enhanced our understanding of managing scalable and modular web applications. Using version control software like Git also helped us learn how to collaborate effectively when working in larger groups.

Future improvements include embedding images and videos. This would require the use of cloud services to efficiently store large media because our devices can't handle high storage requirements. Additionally, it would be fun to figure out storing the uploaded image URLs in SQL efficiently. We would also like to add payment

processors to facilitate fundraiser donations. We would be able to verify the payments through Paypal or Stripe and have a challenge of storing current payment statuses in SQL. Additionally, we would like to try pagination to display only 50 alumni at a time if our website scales too large. This would be great since with thousands of posts, we can't render them all at once. A chat feature would also be good if many users requested it, and we would have to figure out high volume and real-time updates linked with SQL.

Overall, Spartan Outreach is a significant step toward bridging the gap between students, alumni, and staff at SJSU, fostering a supportive and resourceful university network.