CS348 - Final Report

Aaron Choo, Jason Chu, Kallen Tu, Charles Zhang, Jack Zhang

April 14, 2020

Note: Final Changes are highlighted in RED

Description

The application we want to build is a website named Bread that fuses the services of Tinder and WaterlooWorks. Students looking for internships, new graduates looking for entry-jobs, and job-hunters looking for full-time jobs postings are the users of Bread.

When the user first signs in, they will see job cards that describe the job tasks, the hiring company, the location of the job, and other data related to the job posting. Users will be able to swipe right if they are interested in applying to the job or swipe left if they are uninterested in the job posting.

We will keep track of what jobs the user has viewed already and what jobs they are interested in. This data will be used for interview matching with the employer. Employers belong to one or more companies. Employers will be able to post their own jobs under a company and do the same swiping process for interested candidates. Employers will not be able to see candidate profiles until the candidate has already swiped on that company.

Planned Features

We (tentatively) plan to have the following features.

1. Candidates can register and login with accounts and see the Candidate view
2. Employers can register and login with accounts and see the Employer view
3. Candidates can see Jobs postings one by one and swipe left or right to indicate if they like the job or not
4. Employers can post Jobs under a Company. After candidates swipe on this job, Employers can see who liked this job and swipe to indicate if they like the candidate

5. After both sides indicate they like each other, there is a match. Matches are posted to a dedicated page which Candidates and Employers can both see each other's contact info.
6. We plan to allow Candidates to filter by job location and tags.
7. Company selection during employer registration. Employers can select which company they work for by searching companies by name, then selecting the correct company from a drop down list.

Implemented Features

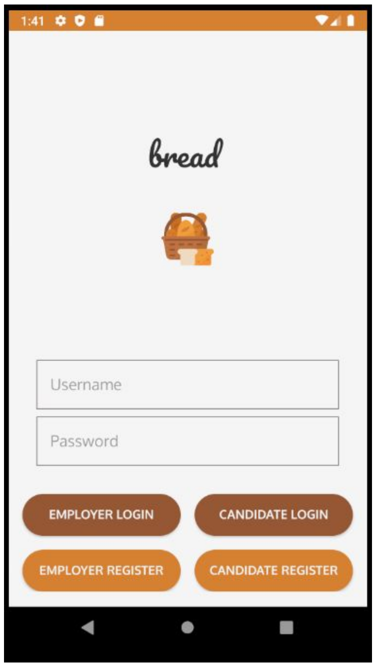
The features which we have implemented currently are:

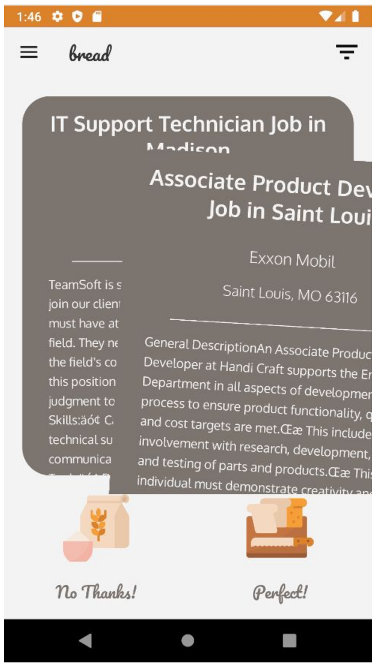
1. Login for candidates frontend & backend
2. Login for employers frontend & backend
3. Registration for candidates backend
4. Registration for employers backend
5. Tag assignment for jobs (search jobs by tags like frontend, accounting, law etc).
6. Get companies, searchable by name (does substring matching) backend
7. Get available jobs for a candidate and jobs could be filtered by tags
8. Candidates swipe left (dislike) on jobs
9. Candidates swipe right (like) on jobs
10. Get open jobs for an employer's company
11. Employers swipe left (dislike) on candidates
12. Employers swipe right (like) on candidates
13. Matches view for candidates
14. Matches view for employers

Fancy Features

1. Get open jobs for an employer's company. We built an index on company_id in the jobs table to improve the performance of this query.
2. Candidate swiping and employer swiping. We added animation in the Android app to deliver an exceptional user experience.
3. Employer matches. This is a complex query involving a 5-way table join, in order to query all necessary data. We also built an index on the company ID in matches to improve the performance of this query.
4. Filter jobs by tags. This enhances the user experience and it requires a more complex query and extra tables to store the job tags.
5. Clean, usable user interface!

Frontend Screenshots





IT Support Technician Job in Madison

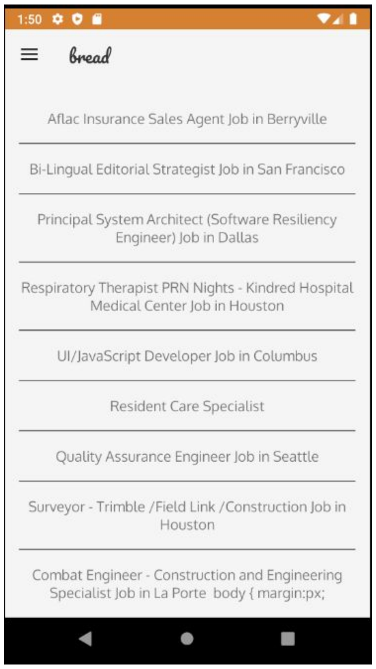
TeamSoft is s join our clien must have at field. They ne the field's co this position judgment to Skills:â€¢ C technical su communica

Associate Product Dev Job in Saint Loui

Exxon Mobil

Saint Louis, MO 63116

General DescriptionAn Associate Produc Developer at Handi Craft supports the Er Department in all aspects of developmer process to ensure product functionality, q and cost targets are met.Œæ This include involvement with research, development, and testing of parts and products.Œæ This individual must demonstrate creativity an



Aflac Insurance Sales Agent Job in Berryville

Bi-Lingual Editorial Strategist Job in San Francisco

Principal System Architect (Software Resiliency Engineer) Job in Dallas

Respiratory Therapist PRN Nights - Kindred Hospital Medical Center Job in Houston

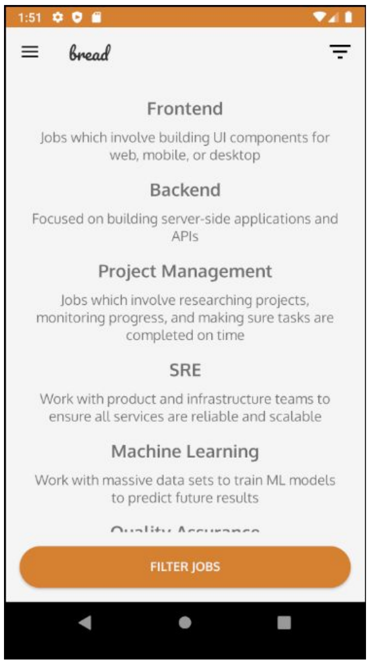
UI/JavaScript Developer Job in Columbus

Resident Care Specialist

Quality Assurance Engineer Job in Seattle

Surveyor - Trimble /Field Link /Construction Job in Houston

Combat Engineer - Construction and Engineering Specialist Job in La Porte body { margin:px;



Frontend

Jobs which involve building UI components for web, mobile, or desktop

Backend

Focused on building server-side applications and APIs

Project Management

Jobs which involve researching projects, monitoring progress, and making sure tasks are completed on time

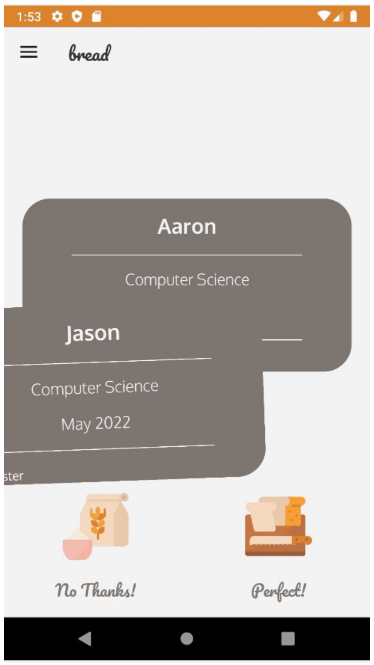
SRE

Work with product and infrastructure teams to ensure all services are reliable and scalable

Machine Learning

Work with massive data sets to train ML models to predict future results

Quality Assurance



Aaron

Computer Science

Jason

Computer Science

May 2022

ster

Technical Design

Our database uses MySQL and is hosted on AWS. In addition to .sql files, we use python scripts to read csv datasets to populate our database tables.

Our backend is written in Go, which exposes the database to the client through HTTP REST APIs in JSON format.

Our frontend is an Android app written in Java, and clients will use this to interact with our data.

Plan for getting data

We will take job data sourced from Monster.com on kaggle to populate our jobs database. The CSV representation of this data can be found in data/monster-job-data.csv. We have a script (script/import_job_data.py) to read this csv data and insert into our database.

However, the Monster.com job dataset on kaggle does not contain company names. Therefore, we will use the list of Fortune 500 companies to populate our companies database, then randomly assign each job to one of the companies in the database.

For application specific data like user accounts, likes, matches etc we will manually create them.

Tags are automatically generated through a script which runs through the job data row by row, and seeing if any of the job titles/descriptions contains the keyword of the tag. For example, if a job is titled Production Engineering (SRE), then it will match tags Engineering and SRE. Output is automatically added to the Jobs Describes Tags table, while the tags themselves are manually created by us.

Assumptions

For the Jobs data which we imported from Kaggle, we are assuming that the id string which they provided is unique.

For candidate seen job (CSJ) and job seen candidate (JSC), we are assuming that a candidate always sees the job before the job (employer) sees the candidate. That is, a row must exist in CSJ before it can exist in JSC.

For a match, we are assuming that there are both CSJ and JSC entries before an entry can be written to the match table. That is, a match cannot exist until both the candidate and job (employer) has seen and swiped on each other

Schema Changes/Optimizations

One optimization that we will implement is to build an index on uid/jid in the matches table. With an index we can speed up the queries by reducing the amount of I/O's needed to find which jobs the candidates have been matched with. We think that it is important to have an index here because people tend to refresh the matches page a lot to see if they got a match, so we need the getMatch query to have as little performance overhead as possible.

We also plan to build a non-clustered index on tag_name in the tags table, as we anticipate that the number of tags will grow when we scale our data, and we want tag search to run as fast as possible. Having an index on tag_name will again reduce the number of I/O's needed to find the tag_id, so that we can quickly query which jobs the tag is associated to.

For M2 we made a schema change to create a new table named "TagsDescribesJobs" which stores pairs of (job_id, tag_id) in a many to many relationship. We plan to "pre fill" this data when a new job is posted by preprocessing the text in the job title and job description. This way, filtering/searching by predefined tags is very fast since we do not need to do any string matching on the fly.

We built a non-clustered index on company_id in the jobs table. There are around 22,000 rows in the jobs table and the query to get jobs for a specific company could take a while to run without an index on company_id, so this index improves the performance of this query.

We built a non-clustered index on jid in the matches table. The matches table had a primary key (uid, jid), this clustered index is useful for the query that gets matches for a specific candidate. However, the clustered index does not help the query that gets matches for a specific job, so we built this index in order to improve the performance of this query, as the numbers of rows in the matches table could get large.

E/R Diagram

List of Tables

Account

Column	Constraints
id	Primary key
username	
password	

Candidates

Column	Constraints
id	Primary key, also Foreign key to account.id
name	
program	
grad_date	
description	

Employers

Column	Constraints
id	Primary key, also foreign key to account.id
name	
works_at	Foreign key to companies

Companies

Column	Constraints
id	Primary key
name	
description	

Jobs

Column	Constraints
_id	Primary key
country_code	
job_description	
job_title	
sector	
company_id	Foreign key to Companies

Job Tag

Column	Constraints
tid	Primary key
tag_name	
tag_description	

Tags Describe Jobs

Column	Constraints
tid	Foreign key to Tags
jid	Foreign key to Jobs

Candidate Seen Job

Column	Constraints
uid	Foreign key to Candidate
jid	Foreign key to Jobs
liked	bool

Job Seen Candidate

Column	Constraints
jid	Foreign key to Jobs
uid	Foreign key to Candidate
liked	bool

Match

Column	Constraints
uid	Foreign key to Candidate
jid	Foreign key to Jobs