1. Database
   1. Tables
      1. Include a job table
         1. Include an id column
            1. This should be a UUID
            2. Ensure this is the primary key

This cannot be null and must be unique

* + - 1. Include a title column
         1. This should be text
         2. This may be null and need not be unique
      2. Include a location column
         1. This should be text
         2. This may be null and need not be unique
      3. Include a status column
         1. This should be text
         2. This may be null and need not be unique
      4. Include a user\_id
         1. This should be a UUID

This should match the UUID of the user as in user.id who created the job

* + - * 1. This may not be null and need not be unique
      1. Include a minsalary column
         1. This should be a 32-bit signed integer
         2. This may be null and need not be unique
      2. Include a maxsalary column
         1. This should be a 32-bit signed integer
         2. This may be null and need not be unique
    1. Include a profile table
       1. Include an id column
          1. This should be a UUID
          2. Ensure this is the primary key

This cannot be null and must be unique

* + - 1. Include a name column
         1. This is text
         2. This can be null
      2. Include a title column
         1. This is text
         2. This can be null and does not need to be unique
      3. Include a bio column
         1. This is text
         2. This can be null and does not need to be unique
      4. Include a location column
         1. This is text
         2. This can be null and does not need to be unique
    1. Include an interview table
       1. Include an id column
          1. This should be a UUID
          2. Ensure this is the primary key

This cannot be null and must be unique

* + - 1. Include a user\_id
         1. This should be a UUID

This should match the UUID of the user as in user.id who created the interview

* + - * 1. This may not be null and need not be unique
      1. Include a format
         1. This is a string
         2. This can be null and does not need to be unique
      2. Include a date
         1. This is a string
         2. This can be null and does not need to be unique
      3. Include a time
         1. This is a string
         2. This can be null and does not need to be unique
  1. Security
     1. Use appropriate RLS policies to restrict access to information (or to read or write) about a user’s profile to only that user
     2. Use appropriate RLS policies to restrict access to information (or to read or write) about a user’s jobs to only that user
     3. Use appropriate RLS policies to restrict access to information (or to read or write) about a user’s interviews to only that user

1. Spring Boot
   1. Profile
      1. Controller
         1. GET endpoint, current
            1. Return a profile of the currently logged-in user
         2. GET endpoint, id
            1. Validates the ID

If the ID is not of the currently logged-in user, throw an error

If the ID is a match, return the profile

* + - * 1. If there is another issue, throw an error
      1. PUT endpoint
         1. Require the id of the profile before changes can be made
         2. If ids match, change information appropriately
         3. If the profile cannot be found, throw an error
         4. If another problem occurs, throw an error
      2. POST endpoint
         1. Creates a new profile
         2. Attaches the id of the currently logged-in user to the profile

Throws an error if a profile for that id already exists

* + - * 1. Throws an error if a problem occurs creating a profile
    1. Service layer
       1. Includes business logic
       2. Includes no HTTP requests
    2. Model
       1. Includes Java elements corresponding to the profile table’s schema 1:1
    3. Repository
       1. Include SQL queries as annotations
       2. For every required SQL query operation in the service layer, include a method that Spring will translate into a query
  1. Job
     1. Controller
        1. GET endpoint
           1. Returns a list of all jobs associated with the user
           2. Returns a 2xx response even if the user has no jobs
           3. Throws an error if something goes wrong fetching a user’s jobs
        2. GET endpoint, id
           1. Returns a specific job whose id matches the parameter, and a 2xx response
           2. Throws an error if no job is found with that id
           3. Throws an error if something else goes wrong
        3. GET endpoint, search
           1. Takes in any subset of the optional parameters, including none of them
           2. Returns a 2xx response either with jobs or with no jobs, depending on how many match the provided parameters
           3. Throws an error if something goes wrong during the search
        4. GET endpoint, status-counts
           1. Returns key-value pairs of statuses a job may have, and how many of a user’s job have that status, for every status
           2. Displays an appropriate message if every count is 0
           3. Throws an error if something goes wrong
        5. POST endpoint
           1. Sets the user id of a job record to be that of the current logged-in user
           2. Creates the record
           3. Returns a 2xx response if creation succeeds
           4. Throw an error if creation fails
        6. PUT endpoint, id
           1. Require the id of the job as a parameter
           2. Returns a 2xx response if the attempt to update succeeds
           3. Throws an error if bad data is passed or if the id is nonexistent
           4. Throws an error if something else goes wrong
        7. DELETE endpoint, id
           1. Requires the id of the job as a parameter

Throws an error if a job with that id cannot be found

* + - * 1. Sends a query that deletes the job from the table
        2. Returns a 2xx if deletion from the job table is successful
        3. Throws an error if something else goes wrong
    1. Service layer
       1. Includes business logic
       2. Includes no HTTP requests
    2. Model
       1. Includes Java elements corresponding to the job table’s schema 1:1
    3. Repository
       1. Include SQL queries as annotations
       2. For every required SQL query operation in the service layer, include a method that Spring will translate into a query
  1. Interview
     1. Controller
        1. GET endpoint
           1. Return a 2xx with all interviews a user has, or with an acknowledgment that they have no interviews
        2. GET endpoint, id
           1. Get a specific interview whose id is the parameter

Throw an error if it does not exists

* + - 1. GET endpoint, search
         1. Take in 0 or more of the optional parameters
         2. Return all interviews that match the subset of optional parameters given by the user

If there are matches, return 2xx

If there are no interviews, return that with 2xx

* + - * 1. If something goes wrong, throw an error
      1. POST endpoint
         1. Validate the input

Ensure there is a time and a date

If either is missing, throw an error

If both are present, create the interview

Throw an error if something goes wrong

* + - 1. PUT endpoint, id
         1. Ensure the id parameter is not null

If so, throw an error

* + - * 1. Ensure the id parameter matches that of the currently logged in user

If not, throw an error

* + - * 1. Attempt to update an interview

If the update is successful, return with 2xx

If the update fails, throw an error

* + - * 1. If a problem occurs, throw an error
      1. DELETE endpoint, id
         1. Attempts to delete an interview with the given id

If deletion is successful, return with 2xx

Otherwise, throw an error

* + - * 1. If a problem occurs, throw an error
    1. Service layer
       1. Includes business logic
       2. Includes no HTTP requests
    2. Model
       1. Includes Java elements corresponding to the interview table’s schema 1:1
    3. Repository
       1. Include SQL queries as annotations
       2. For every required SQL query operation in the service layer, include a method that Spring will translate into a query
  1. Starter
     1. Starter
        1. Active profile should be local
        2. Environment variables should be set
           1. supabase.jwt.secret
           2. db\_password
        3. should successfully start the backend of the application

1. UI