NEO Deployment Document

Deployment platform - Heroku
Deployed Application Link - https://teamup-neo-ba74edfc326f.herokuapp.com/
Storage Media - PostgreSQL

Steps for Deployment

1) Google OAuth Setup

This section will walk you through setting up Google OAuth for your application in the Google Developer Console. Here are the steps and important information:

Step 1: Create a New Project in Google Developer Console

1) Go to the Google Developer Console.
2) Select or create a project for your application, giving it a name like "teamup-neo-backend."

Step 2: Set Up the OAuth Consent Screen

1) In your project, navigate to "APIs & Services" and click on "OAuth consent screen."
2) Set the user type to “External” in testing mode.
3) Fill out the required information on the consent screen. You need to provide the app name, user support email, and developer contact information.
4) Save your changes.

Step 3: Add Scopes

1) Add userinfo.email, user info.profile, /auth/calendar.events.public.readonly, and /auth/calendar.calendars.readonly
2) Click "Save and Continue."

(Optional) Step 4: Add Test Users

You can add test users who are allowed to log in to your application. This means that only the email addresses you add here can access your application. If you choose to add test users, do so on this page and click "Save and Continue."

Step 5: Create OAuth Client ID

1) On the dashboard, click on "Credentials," then "Create Credentials."
2) Select "OAuth client ID" and choose "Web application" as the application type.
3) Give your application a name.
4) Under "Authorized redirect URIs," add the following:
    a) http://localhost:3000/auth/google_oauth2/callback
    b) http://localhost:3000/callback
    c) http://localhost:3000/users/google_oauth2/callback

    Note: Replace the below with your deployed application url
    d) https://teamup-neo-ba74edfc326f.herokuapp.com/users/google_oauth2/callback
    e) https://teamup-neo-ba74edfc326f.herokuapp.com/auth/google_oauth2/callback
    f) https://teamup-neo-ba74edfc326f.herokuapp.com/callback
5) Click "Create."
6) You will receive a client ID and client secret. Save this information.
7) Ensure that your client ID is enabled.

By following these steps, you have set up the necessary configurations in the Google Developer Console to enable Google OAuth for your application. This allows your app to authenticate users using their Google accounts.

Step 5: Add OAuth ID and Secret to Rails Credentials Edit the Credentials

Run the below command
`EDITOR=nano rails credentials:edit`

The credentials file will open in the editor.

Add your Google OAuth credentials to the file in the below format. Make sure to maintain the correct indentation and spacing as shown. There should be 2 spaces before client_id and client_secret, and a space after the colon:

google:
  client_id: your_client_id
  client_secret: your_client_secret

Note: Replace your_client_id and your_client_secret with your own Google OAuth credentials. Do not include any quotes around the actual credentials.

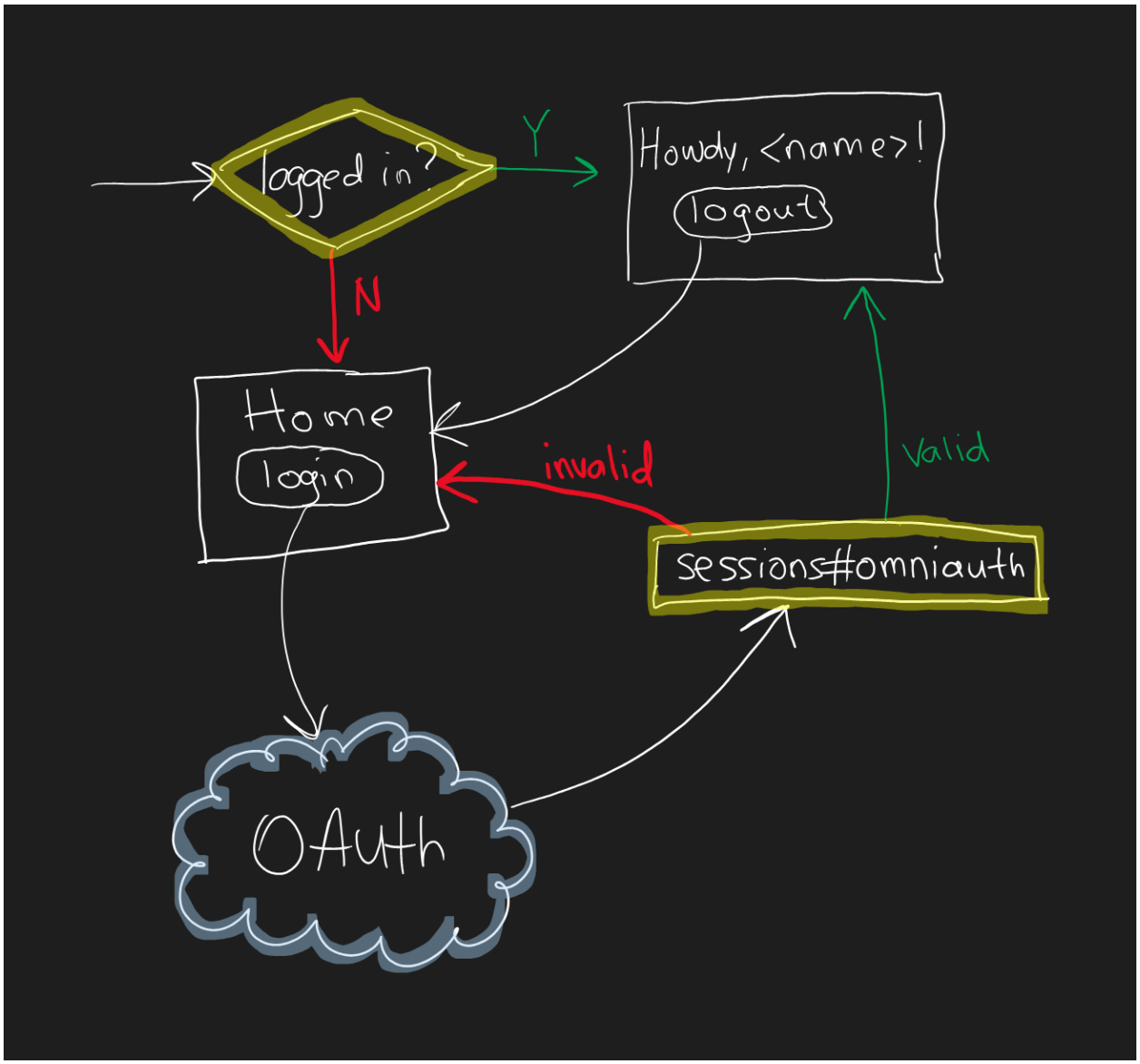
After adding your credentials, save the changes and exit the editor.

Now, your Google OAuth credentials are securely stored in the Rails credentials file and your application will be able to use them for authentication. Make sure to keep your credentials safe and secret.

Note: The application integrates Google Calendar which requires sensitive scopes to be added while setting up the credentials (Step 3). However, the addition of sensitive scopes restricts the app to be only used in testing mode. The .tamu accounts will have access to the calendar, however, if we want the .gmail accounts to also have access then the application should be switched to production mode. This switchover will

require you to get the application verified by Google and the process should be done in approximately 1 week. Currently, the deployed application is not verified by Google.

Temporary fix - to get the .gmail accounts access to the application, we will need to add their gmail accounts as test users. This will give them access to the application.



2) Amazon S3 Bucket Setup

Create an S3 Bucket in AWS

1. Log in to the AWS Management Console:
    - Go to AWS S3 Console.
2. Create a Bucket:
    - Click on Create bucket.
    - Enter a Bucket name (e.g., my-rails-app-bucket).
    - Select a region.
3. Configure Permissions:
    - Disable public access by default for sensitive data.
    - For public files (e.g., images), later configure permissions accordingly.
4. Complete Creation:
    - Click Create bucket.

Install Necessary Gems

Add the following gems to your Rails application:

```
gem 'aws-sdk-s3', '~> 1.48' # AWS SDK for Ruby
gem 'dotenv-rails', groups: [:development, :test]
```

Set Up AWS Credentials

1. Create an IAM user in AWS with the necessary S3 permissions:

    - Go to IAM Management Console → Users → Add users.
    - Provide programmatic access and attach the policy: AmazonS3FullAccess or a custom policy with limited access to your bucket.
    - Save the Access Key ID and Secret Access Key.

2. Store the credentials in the credentials.yml.enc file.

3. Configure Rails to Use S3

```
rails active_storage:install
```

```
rails db:migrate

# config/storage.yml

amazon:
  service: S3
  access_key_id: <%= ENV['AWS_ACCESS_KEY_ID'] %>
  secret_access_key: <%= ENV['AWS_SECRET_ACCESS_KEY'] %>
  region: <%= ENV['AWS_REGION'] %>
  bucket: <%= ENV['AWS_BUCKET_NAME'] %>
```

4. Set Active Storage to use S3 in production (or other environments):

3) Installing Required Dependencies

Execute the below command to install all the required gems

```
bundle install
```

4) Database Setup

1) **Create database**

   ```
   rails db:create
   ```

2) **Execute the DB migration to create the user's table in the local database using the below command**

   ```
   rails db:migrate
   ```

3) Create seed data into the database, use the below command to prepopulate the database with the list of authorized users

   ```
   rails db:seed
   ```

Note: Add at least one admin role in the seed database, so that other users can be uploaded using the import feature.

5) Running Application

Launch the rails application using the below command, the application will be hosted on http://localhost or http://127.0.0.0 with the default port as 3000

```
rails server
```

6) Running Tests

Note: Google OAuth Configuration is required for the application to start up smoothly and run tests.

Execute the below commands to run RSpec

```
bundle exec rspec
```

Execute the below command to run cucumber scenarios

```
bundle exec cucumber
```

7) Heroku Deployment

Login to Heroku CLI

If you have not already done so, install the
[Heroku CLI](https://devcenter.heroku.com/articles/heroku-cli#install-the-heroku-cli).

1) Heroku login
2) Create a Procfile

```
echo "web: bundle exec rails server -p $PORT" > Procfile
```

3) This file tells Heroku what to do when the application is deployed. In this case: spin up a web process to host the application.

4) Create a Heroku App

```
heroku create <app_name>
```

5) Verify that the git remote is set

```
git config --list --local | grep heroku
```

6) Make Master Key Available on Heroku

```
heroku config:set RAILS_MASTER_KEY=`cat config/master.key`
```

This enables Rails on Heroku to decrypt the credentials.yml.enc file, which contains the OAuth credentials (and any other secrets your app needs).

7) Provision a Database

```
heroku addons:create heroku-postgresql:essential-0
```

An essential-0 PostgreSQL database costs $5 a month, prorated to the minute.

8) Add PostgreSQL to Gemfile

```
group :production do
  gem 'pg'
end
```

Run this so that your app won't try to install production gems (like Postgres) locally:

```
bundle config set --local without 'production' && bundle install
```

9) Deploy the App to Heroku

```
git push heroku main
```

10) Migrate the Database

```
heroku run rails db:create
heroku run rails db:migrate
```

11) Seed the Database

```
heroku run rails db:seed
```

12) Launch the application

```
heroku open
```

Your app is now successfully deployed to the production environment.