Team Wowzer

Documentation

(3 marks out of 30)

Documentation
At the end of week 12, you will submit documentation to support your completed application. This is a reflective documentation which should be written for an audience of other developers.

Your documentation should feature the following components at a minimum:

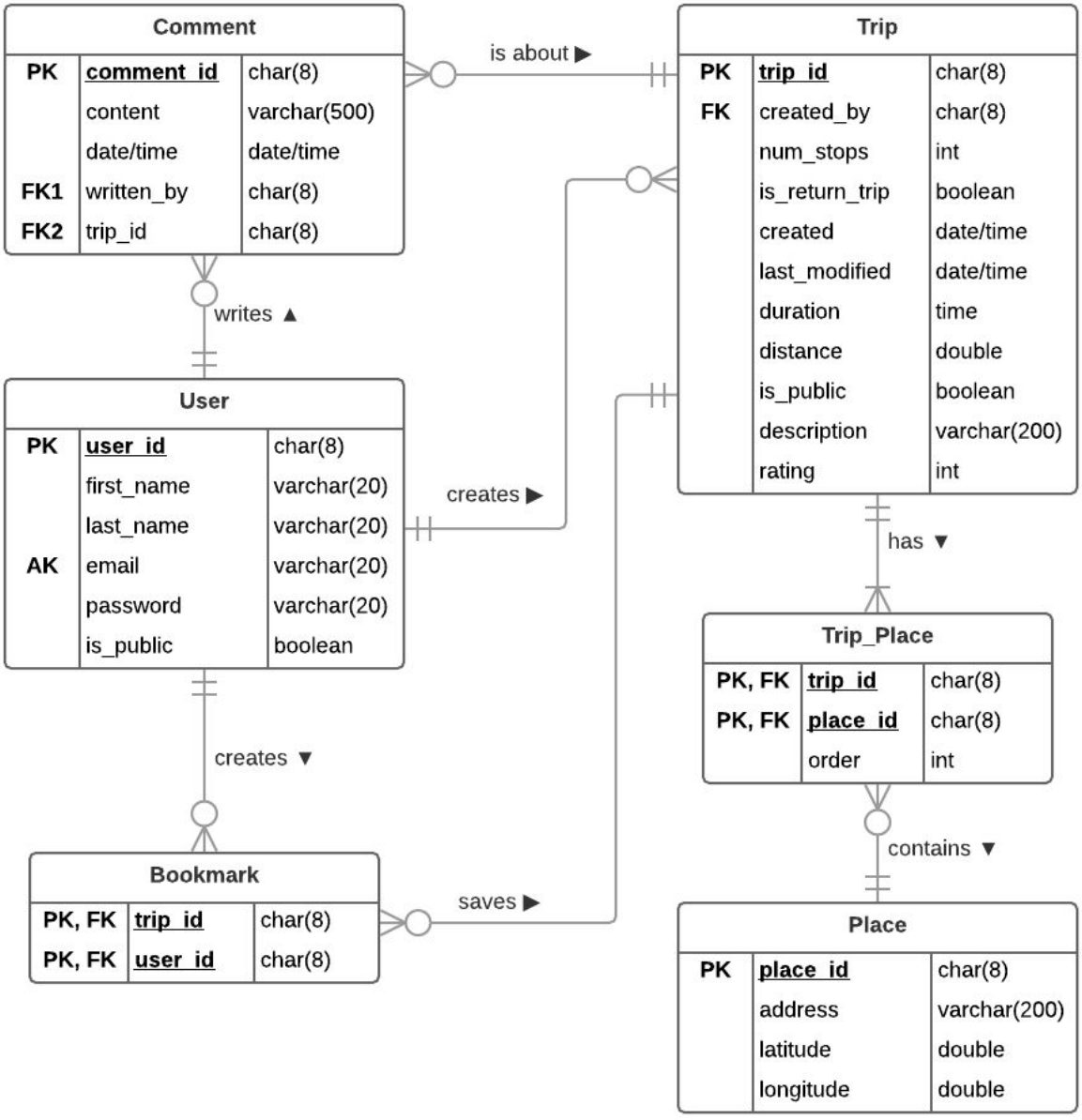
- **Data Model**: Include the data models provided in your SRS featuring any changes made during the implementation process.

  SRS Data Model

- **Code Structure**: Include the documentation on code structure you provided in your SRS featuring any changes made during the implementation process.
- A comparison of your finished project to the SRS
- Describe how your finished project compares to the product you defined in your SRS.
    - Did you fail to fully meet your specification? Did you exceed it? Why?

Your documentation will be assessed from the perspective of another developer, who has been brought in to take over your project. Your documentation should include all details necessary for another developer to get up-to-speed on how your system works, the code layout, and any dependencies or issues that they may face.

1. Data Model



1.1 Data Model Modification

For place model, the latitude and longitude attributes are replaced with google map string id for coordinate representation as it has better google map api support in rendering and fetching.

2. Code Structure

2.1 Style Guides

The project requires the use of various programming to build the finish product. Each of those languages has guidelines to be followed in the coding process to ensure consistency and clear communication of code across the different languages. The languages used are mainly:

- JavaScript
- Python
- CSS
- HTML

JavaScript

JavaScript is used in delivering content on the page of the website using AJAX and React Native to build the product or app. The JavaScript style guide that will be used is the Google JavaScript Style Guide and is available at https://google.github.io/styleguide/jsguide.html. The front end and the backend are completely modularised in the sense that API routes are made in the backend through which the React framework fetches data through distinct AJAX calls directly for HTML requests such as GET and POST.

Python

Python is used mainly on the back-end in Django framework. The style guide is outlined in the PEP 8 documentation at https://www.python.org/dev/peps/pep-0008/ for further details. Serializers were implemented in the backend using the Django REST framework to be able to set up API for the front end to fetch all the data that is requested from the database to be rendered in the frontend

HTML and CSS

HTML and CSS are essential for the presentation of the various pages of the website. The style guide is explained in the Google HTML/CSS Style Guide guideline available at https://google.github.io/styleguide/htmlcssguide.html for further information.

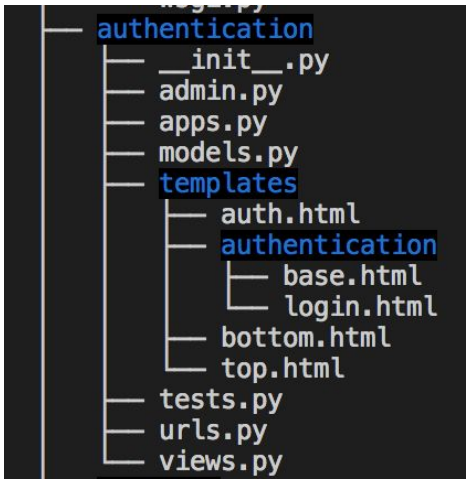
2.2 Code structure modification

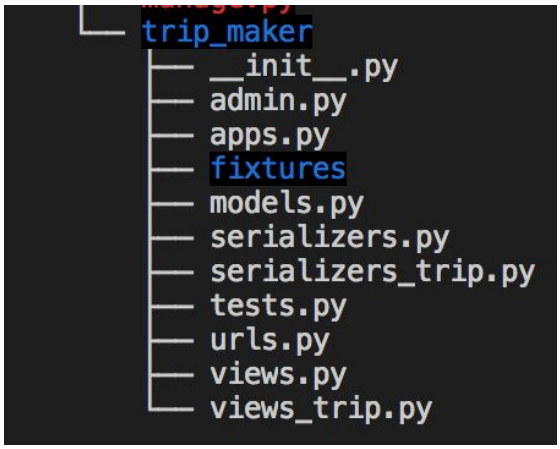
React is a JavaScript framework that intends to achieve declarative, efficient and flexible creation of user interfaces. This is why the responsibility of front-end rendering was given completely to React. This is achieved by creating a seperate app for frontend rendering.

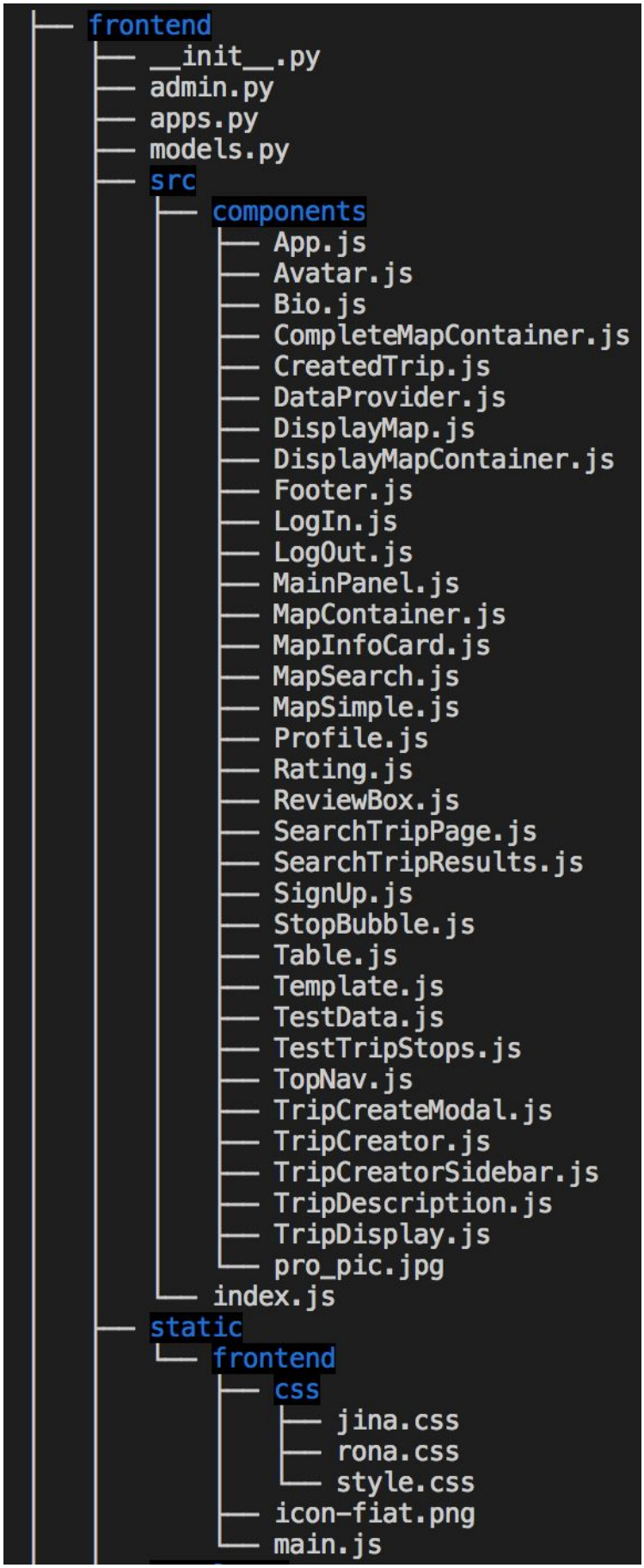
There were three apps installed:
authentication: Responsible for authenticating users when logging in, registering, and logging out - and also when user specific views are rendered

trip_maker: responsible for setting up the API calls made by React frontend via AJAX for HTTP verbs such as GET and POST

frontend: responsible for creation and managing of the user interface and making AJAX calls to the backend






frontend
__init__.py
admin.py
apps.py
models.py
src
components
App.js
Avatar.js
Bio.js
CompleteMapContainer.js
CreatedTrip.js
DataProvider.js
DisplayMap.js
DisplayMapContainer.js
Footer.js
LogIn.js
LogOut.js
MainPanel.js
MapContainer.js
MapInfoCard.js
MapSearch.js
MapSimple.js
Profile.js
Rating.js
ReviewBox.js
SearchTripPage.js
SearchTripResults.js
SignUp.js
StopBubble.js
Table.js
Template.js
TestData.js
TestTripStops.js
TopNav.js
TripCreateModal.js
TripCreator.js
TripCreatorSidebar.js
TripDescription.js
TripDisplay.js
pro_pic.jpg
index.js
static
frontend
css
jina.css
rona.css
style.css
icon-fiat.png
main.js

3. Implemented system comparison with SRS

For the system implementation we are able to implement moderate amount of features specified in SRS, in particular, we are able to achieve most of the backend oriented features such as account login, creation and sessioning.

Accomplished User story:
Create Account
Log Into Account
Log Out of Account
Create Trip
View Trip

Partial Implemented User story:
Post Comment
Rate Trip
Bookmark Trip

Unimplemented User story:

Edit Profile
Delete Account
Change Privacy of Account
Edit Trip
Delete Trip

4. Describe how your finished project compares to the product you defined in your SRS.

During the implementation of the product we constantly reviews on the specification outlined in SRS for coding structuring regarding the feature interactivity. This involves frontend UI design interconnections, data model design, routing design and function declaration.

For many user story implementation, we have been using API routing support for data transmission between front and back end, this include Creating trip record via front end,
Additionally we created API routing support for data transmission between front and back end with most user story implementation, with ajax support we achieves onsite data refreshing.

Though with the detailed specification outline in SRS, there are implementation decision we need to make during implementation which is not specified. These are more of technical oriented designs such as API routing methods.

In conclusion we failed in fully meeting our specifications, partially due to some technical obstacles we fail to overcome.