Dev Notes

Django + React

———————————————————————————————————

Django-admin - main commands Django-admin startproject (project name)
CMD + W - close tab in pycharm
CMD + SHFT + . - show hidden files Mac
CMD + SHIFT + [(move a block of text left) CMD + SHIFT +] (move a block of text left)
Linux: cd - (goes back one directory)
rm -r (directory name) - remove a directory
mv (file/dir 1) (file/dir 2) move around files/directories
manage.py - file enables to run command line commands __init__.py - tells python that this is a python package settings.py - settings of the project
urls.py - path pattern where to send the user
wigs.py - how our python web application and web server communicate
python manage.py runserver - command run inside project directory to run server
Ctrl + C - stop server
Ctrl + L - clear terminal
CMD + Shift + R - hard reset of page, clearing cache
Ctrl + D - exit shell
SHIFT + Reload button in chrome - bypass the cache - reload static css in python dev mode
CMD + / - comment out a block of code
Mv folername/* . (Moves contents up to current directory)
Within the Django website project there can be multiple apps within that project. Eg a blog app within a larger project. You can then take that app and use it in multiple projects.
python manage.py startapp (app name) - create an app within the directory of a project
{% injects python/django syntax into html files %}

———————————————————————————————————

Django admin
Django Admin Controls: djangourl/admin - login (LeonR - Martialx12?)
python manage.py migrate - runs through migrations - migrations are useful s they allow us to make changes to our database even after its created and there is data In our database. If no way to run migrations, then have to run complex SQL code to update our database structure so it doesnt mess with the current data. With migrations we can simply make whatever changes we need, the run makemigrations then run migrate then it will make those changes for us
python manage.py createsuperuser - creates a django user
python manage.py makemigrations - detects the changes and prepares django to update the database, but doesn't run those changes - migration file created - 'from what I understand it shows the latest database models changes in django'
Django has its own built in ORM (object relational mapper) enables us to access

our database in an object oriented way, you can use different databases without changing your code. All the code to query any data base is the same. Can represent our database structure as classes, aka models.
Using two databases in blog app - SQLlight database for development, PostgreSQL database for production
python manage.py sqlmigrate (django app name) (migration number) - takes the simple class that we created in django and it writes out the SQL for all of the fields that will be compatible for the database that we are using - saves a tonne of time by not having to write out the SQL, didn't even need to know SQL to work with the database, just used the python model class in our models.py file and it wrote the backend SQL for use. That is why ORM's are so convenient, don't have to go deep into SQL code most of the time.
Everytime you add or change your models - you have to make migrations then migrate
———————————————————————————————
python manage.py shell - Django/Python Shell: Query the database using these models. Django ORM lets us do this using the classes as well. The shell allows us to work with these models line by line. Can run python code in here and work with our django objects. Interactive python (ipython). In the shell type:
python manage.py test (Django app name) - run tests in test.py file
from blog.models import Post
from django.contrib.auth.models import User
User.objects.all() - query our user table, will return a query set result of our users
User.objects.first() - first user
User.objects.last() -last user
User.objects.filter(username='LeonR') - filters for users name
User.objects.filter(username='LeonR').first() - returns just the user without the query set data after being filtered for a certain user User.objects.get(id=1) - returns the user with the id 1
Run: user = User.objects.filter(username='LeonR').first() to set the specific user to the variable user to perform more database functions in the python/django (iPython) shell.
user.id - id number of the user user.pk - primary key
Post.objects.all() - check for posts in the blog app
post_1 = Post(title='Blog1', content='First Post Content!', author=user) - creates a post_1 object, can also use author.id=user.id
post_1.save() - save the object
post = Post.objects.first() - grabs first post
post.content - blog content of the post
post.author - author of the post
post.author.email - the email of the user gathered from the database by just using their user name from a post
user = post.author
user.post_set - get all the posts a user has created, can run queries against this
user.post_set.all() - see all the posts the user has posted
user.post_set.create(title='Blog 3', content='Third Post Content!') - can create

a post by the user from here. Didn't have to specify the author for this post as django knows your setting the post to this author. Also don't need to run .save()
profile model in shell:
user.profile.image user.profile.image.url user.profile.image.width
———————————————————————————————————
Json in shell:
Import json file into ipython shell to load 25 blog posts, in the shell:
import json
from blog.models import Post with open('posts.json') as f:
posts_json = json.load(f)
for post in posts_json:
post = Post(title=post['title'], content=post['content'],
author=post['user_id'] post.save()
– Didn'twork ————————————————————————————————————
Pagination:
from django.core.paginator import Paginator posts = ['1', '2', '3', '4', '5']
p = Paginator(posts, 2)
p.num_pages
pages
for page in p.page_range: print(page)
- dummy post object
- paginate list 2 per page
- test the amount of - loops through pages
p1 = p.page(1)
p1.number
p1.object.list
p1.has_previous() - boolean if has a previous page p1.has_next() - boolean if has a next page
- set the variable of page 1 - prints page number out of total - shows object of p1
p1.next_page_number() - next page number
———————————————————————————————————
Environment Variables: - set environment variables on your machine for use in your project without having to show sensitive information in your script
Terminal - cd - (root directory of computer)
nano .bash_profile
sudo nano .bash_profile (if permission denied error)
export DB_USER ="username" - environment variables set
export DB_PASS="password"
Ctrl + X - Y to save - enter to exit terminal
Reset text editor
You can now access these environment variables inside your code
———————————————————————————————————

Git

git init - at base directory of our project - initialise an empty git repo github/gitignore/Python.gitignore - a file of common python files that don't need to be tracked by git for version control

git add -A
git reset
git status
git commit -m “Initial Commit” - commit files to git locally
- adds all the changes so far to the staging area - removes all files from staging area
- shows all the new files to be committed
You may want to add multiple commits after many changed to your code and therefore only state certain parts of your code and commit then repeat.
git log (view history of commits) git diff (see changes)
q (exit these git shells)
git checkout . (revert back to last committed version)
git remote command lets you create, view, and delete connections to other repositories.
git remote -v command lists the URLs of the remote connections you have to other repositories.
git push command pushes the changes in your local repository up to the remote repository you specified as the origin.
git (command) —help (details of each command) ls -la (show all files in directory, hidden also)
rm -rf .git (remove git from directory)
———————————————————————————————

Heroku
Goto your project directory
Terminal: heroku login - authenticate heroku with project
heroku git:remote -a (heroku app name) - initialise a git repo
heroku config:set DISABLE_COLLECTSTATIC=1 (fix option if not deploying)
Heroku local - runs commands in Profile locally
Git checkout -b main
Git branch -D master
(This created a main branch, deleted master, so we are now on main, ready for heroku deployment post initial deployment)
pip install gunicorn -
pip freeze - lists out all the pip installations we have made - need to add these to a requirements.txt file - requirements file I show heroku knows its a python application
pip freeze > requirements.txt - on Mac creates a requirements.txt file directly into the root directory
heroku addons:create heroku-postgresql:hobby-dev
heroku create (application name)
https://git.heroku.com/djangohelloblog.git
heroku open - open the app
git push heroku master - push our code up to heroku - if error it won’t push the app up to the server so site doesn’t break heroku open - after git push heroku master ran
heroku logs—tail
Error: ‘no web processes running’ - means has no proc file

Config variables
heroku config:set ALLOWED_HOSTS=APP_NAME.herokuapp.com heroku config:set SECRET_KEY=DJANGO_SECRET_KEY
heroku config:set WEB_CONCURRENCY=1
———————————————————————————————————
virtualenv for each project so your pip dependencies are specific to each project
Procfile - web: gunicorn django_project.wsgi
web - declares that this process type will receive web traffic when deployed
wsgi - web service gateway interface
gunicorn - the command
django_project - the directory name that holds your settings.py file
pip install pipreqs
pipreqs - run inside python project directory to get relevant dependencies for the project ———————————————————————————————————
python shell: - get a secret key in python
import secrets
secrets.token_hex(24)
2abc2216c1c0ebedd194a66982f9afcc0dc82f5f0e1f6992
sudo nano .bash_profile - set these in base profile
SECRET_KEY="2abc2216c1c0ebedd194a66982f9afcc0dc82f5f0e1f6992"
DEBUG_VALUE="True"
heroku config:set (the bash variable and its value as printed in bash profile, for all the hidden variables)
——————————————————————————————————— Terminal:
heroku addons - can see what add ons you have on your app - can see if heroku has created a PostgreSQL database for you
heroku add ons: create heroku-postgresql:hobby-dev - if you don't have progresql running - the free version
heroku pg - shows info on the database (10000 rows)
pip install django-heroku - running this in project directory - auto configure database url - take care of connecting our static assets tog unicorn using a package called white noise - also set up things for login - also take care of allowed hosts and secret key which we have already set up
heroku run python manage.py migrate - run python files on heroku machine - run the migrations for the application and create the tables for the app
heroku run bash - connect to the heroku machine (dynos - name of the heroku machines, so you have to run linux commands as its a Linux server (eg ls))
Create a superuser - last super user was only for local environment - have to create a live superuser on heroku machine
python manage.py createsuperuser - go and create a super user name, email and pass
exit - exit the heroku machine back to local machine
Superuser: LeonJMR
Email: leonjmroe@gmail.com Pass: Martialx12?
Django deployment checklist documentation is important to go through before fully deploying

heroku releases - shows all deployments
heroku rollback v7 - automatically rollback to v7 deployment - very efficient if mistakes made
———————————————————————————— AWS
S3 - simple storage service
Create a new bucket - what holds your files - have to be universally unique
Buckets - permissions - CORS - paste into CORS file (obtainable from heroku) - json format:
[
{
anywhere],
"AllowedMethods": ["PUT",
"POST",
"GET"],
"AllowedOrigins": ["*"
],
"ExposeHeaders": [] }
]
IAM - identity access management:
User - create a new user
Programmatic access - check - use this user to access AWS with a secret access key
Attatch existing policies directly - amazons3fullaccess - check - next - next create user - this provides this user with full access to the s3 bucket just created - maybe gives abit more permission than needed but the secret key adds security
Access key id and secret access key are provided after this user is created
———————————————————————————— .bash_profile
environmental variable data for the project:
export DB_USER="leonjmroe@gmail.com"
export DB_PASS="Roeincarnation12?"
export
SECRET_KEY="2abc2216c1c0ebedd194a66982f9afcc0dc82f5f0e1f6992"
export DEBUG_VALUE="True"
export AWS_ACCESS_KEY_ID="AKIAZQ7AQAWAA5IRRG5B" export
"AllowedHeaders": [
"*" - allow the origin from
AWS_SECRET_ACCESS_KEY="ImqF4b5IWRhnBaG41ZMj0dfT42UZJNJP2/
2KUiip"
export AWS_STORAGE_BUCKET_NAME="django-hello-blog-files"
————————————————————————————

virtualenv

multiple python environments:
You don’t put your project files inside your environments they are just used to install the packages and dependencies and version that your project will require
pip install virtualenv - in root directory on Mac
pip list - see all pip installed packages

mkdir Environments - make a directory called environments at root directory of mac to hold all your environments
virtualenv (environment name) - django_managementapp
source (environment name)/bin/activate - activate environment - will show that you are in the environment with name in brackets on prompt
pip freeze —local > requirements.txt - requirements file of all packages cat requirements.txt - shows list of requirements and versions (didn't work for me)
deactivate - exits environment
pip install -r requirements.txt - if you close an environment and start a new one you can load the .txt file to load the packages when in a new environment
Pycharm creates its own virtual environment very easily when you create a new project. Go to preferences - project - project interpreter - can see all the packages. Click gear icon - add - to add a new virtual environment
Can install packages right into virtual environment in pycharm, from the project interpreter window
————————————————————————————
pip list -o (or pip list —outdated) - checks what packages need updating
pip install -U (package name) - update package
pip freeze —local | grep -v '^\-e' | cut -d = -f 1 | xargs -n1 pip install -U
(this command above will go through all packages and update them)
————————————————————————————— Function
Method - a function that belongs to an object
dbsqlite3 - in django file system - your database
—————————————————————————————

React

Need to download node.js and npm - node.js is a cross-platform javascript run-time environment that executes JS code outside of the browser. Ode package manager is a dependency management tool for JS applications.
node -v (version of node installed) npm -v
usr/local/bin/node - location usr/local/bin/npm
node (type In terminal to enter node machine)
Create a react app: npx create-react-app (appname) cd into directory
npm start
To integrate react with django - drag react folder into root directory of django app. Cd into react folder within django directory and run:
npm run build - need to run this every time you edit react
To work with react and django, you need two terminals. One in django dir and one in react dir, which is inside django dir.
Package.json - react (version of software) react-dom (allows to build within the browser, no react-dom in mobile apps) react-scrips (dev server, compile the app, tests etc)
public - index - div root - output for the application
src - index.js - entry point to react - importing the library, react-dom and the main app component - this file gets element by id 'root' from index.html in public file and injects code into it.
<app /> is being loaded in src-index.js this is app.js - uses JSX - looks like html but can have JS inside - have to use className over Class in JSX

——————————————————————————————————————— Django Rest Framework
pip install djangorestframework
Add this to installed apps in django settings file: 'rest_framework',
———————————————————————————————————————

Django + React Application

mkdir (project & environment name)
cd into it
pip3 install pipenv (globally - virtual environment)
pipenv shell (make virtual environment) - creates a pip file - where all packages go
pipenv install django djangorestframework django-rest-knox (for authentication)
(Virtual environment) (directory) - can navigate directories whilst in the virtual env, use pip list to check
django-admin startproject (project name)
cd djangoapp
python manage.py startapp (django app name)
npx create-react-app (react app name) npm start
npm run build
pip freeze —local > requirements.txt (dependencies inside virtualenv)
python manage.py createsuperuser (LeonR, leonjmroe@gmail.com, Martialx12???)
python manage.py makemigrations
python manage.py migrate
Don’t forget to makemigrations and migrate for each django app, particularly when working with DRF:
python manage.py makemigrations (Django_app)
python manage.py migrate (Django_app)
——————————————————————————————————————— Virtual Environment Simplified
Goto root directory of django-react project
pipenv shell (generates a Pipfile - guessing its the virtual env file) pip list to check
pip install any dependencies/packages
pip list to check they installed
pip freeze —local > requirements.txt (Cretes list of dependencies that were loaded in virtualenv)
Ctrl-D (exit virtualenv shell)
pip install -r requirements.txt (to reverse and load pipenv with dependencies from file) ———————————————————————————————————————
Sublime Text - Package Installation
CMD + Shift + P
Select Package Control: Install Package
Installed “Babel”
Installed “A File Icon”
Installed “Materialized CSS Snippets”

Installed “HTML-CSS-JS Prettify” (don’t know if it does anything) Installed “All Autocomplete”
Installed “Css Colors”
Installed “Auto Close”
Installed “Color picker” (CMD + SHFT + C)
Installed “HTML5” “HTML Snippets”
Previously:
Installed “Javascript Enhancements” Installed “Terminal View”
Installed “React ES6 Snippets” Installed “Sublime Linter”
Installed “Jedi”
———————————————————————————————————

Further Django-React study
CORS: in django_react_app project middleware hashed out for now that should be included after running: pip install django-cors-headers
Package-lock.json - identifies this project as unique if in multiple locations if one gets updated
———————————————————————————————————

React
Component - self contained piece of code with html, JS and styling in one. npm install react-router-dom (in root dir of react app - reat routing)
npm install @material-ui/core (good react library)
React Routing:
<Switch> tag specifies react to only render out the one component that matches the path and ignore the rest
exact attribute inside a <Route> tag says that to only render out component if the paths exactly match
e.preventDefault - stops page refreshing on event
[...previousdata] - three dots in react on a. State variable just gets all previous data in the array so its easy to add on
React state management: Context API or Redux - redux sounds more complicated but industry standard it seems.. Context api seems to be able to do it all as well as props... - state management is essentially passing global variables around through components in react. Seems to be props + contest api v redux ——————————————————————————————————— Redux Setup:
npm install redux react-redux redux-thunk connected-react-router Add Reducer.js file to react src
Add Root.js file to react src
Replace Rooter component with Root in App.js
Remove browser router form app.js and import root.js
———————————————————————————————————

Git Commands:

Terminal commands:
touch (filename) creates a file
ls - lists all of the folders
ls -la - lists all of the files
cd .. - returns one dir back

cd - enters a directory
. - just install in the current directory
On initial install:
git --version - checks the version of the installed locally git
git config --global user.name "Your Name" - sets up the name of the user
git config --global user.email "yourname@somemail.eu" - sets up the mail of the user
git config --list - lists all the git configurations
For help on commands:
git help <verb> (e.g. git help config) OR git <verb> --help
For initializing the project:
git init - initializes the git repo in the current folder
touch .gitignore - creates a git ignore file
git status - check working tree - both on the git and on local
Add files:
git add -A - adds all of the files for commiting remember - git status - to check the state of the repo
Remove files:
git reset - removes files to be commited
git reset somefile.js - removes somefile.js from the commit preparation
Committing:
git commit -m "This is the commit message" - -m is used to add message
Check log:
git log - renders commit ids, authors, dates
Clone a remote repo:
git clone <url> <where to clone>
View info about the repo:
git remote -v - lists info about the repo git branch -a - lists all of the branches
View changes:
git diff - shows the difference made in the files
Pull before push ALWAYS: git pull origin master
THEN PUSH:
git push origin master - <origin> name of remote repo <master> the branch that we push to
First time push of the branch:
git push -u origin <name of the branch> - -u coordinates the two branches (local and on server)
Create a branch:
git branch <name of the branch>
Checkout a branch:
git checkout <name of the branch>
Merge a branch:
git checkout master
git pull origin master

git branch --merged - see which branches are merged git merge <name of the branch you want to merge> git push origin master
Git merge —abort
Delete a branch:
git branch -d <name of the branch> - this deletes it locally!!!
git branch -a - check the repo branches
git push origin --delete <name of the branch> - this deletes it from
the repo!
Move branches:
git branch (see all the branches, * is the one you are on) git switch (branch name) (switch to another branch)

Commit control:
git log to see all the commits (enter to go further back, q to exit this
shell) - copy the commit ID
git checkout (commit ID) - reverts back to a different commit - this will then be called “HEAD” - some form of branch (current branch) - says “head detached from commit ID”
-just experimented and git checkout (initial commit) - I then run git log and I can only see this initial commit, so all code is gone?!? - when I go into sublime merge the commit tree is there to be seen.
-creating a head in git then committing on that head branch then reverting back to a previous commit will loose your commits, I guess you have to merge. Committing back to master from head deletes the head.
-you only checkout if you’ve got a bug or something and need to go back
so you don’t plan on keeping the commits in front of the head. However master branch is now where you made a head from
Revert back to an old commit:
git log (to find the commit ID)
git checkout (commit ID)
git checkout -b (new master name)
git branch -D master
git branch -mv (new master name) master
git push (after committing to push to private GitHub repo)

Sublime text git (next to file):
-uncoloured circle
-blue circle (edited file)
-blue arrow (added to staging)
-no circle (committed/un-edited file)
-orange prongs (two merged files) - this will come with syntax filling in on your code to show what code was just merged to master
Simple Virtual Environment:
python3 -m venv myenv
cd (django backend dir) virtualenv venv

source venv/bin/activate
Superuser for luma_algo_lab: (leonjmroe-admin) (leonjmroe@gmail.com) (Consciousrealm12.0)
——

Bootstrap CSS

Its all about class names, no css just learn how to use the short hand class names. (Code-value)
Values: 1-5
Codes:
ms - margin left
mt - margin top
mb - margin bottom
p - padding all around (very handy)
property = m for margin and p for padding
Following are sides shorthand meanings:
● l=definestheleft-marginorleft-padding
● r=definestheright-marginorright-padding
● t=definesthetop-marginortop-padding
● b=definesthebottom-marginorright-padding
● x=Forsettingleftandrightpaddingandmarginsbythesinglecall ●
y=Forsettingtopandbottommargins
● blank=marginandpaddingforallsides
The size can be from value 0 to 5 and auto. I will show you examples for seeing the difference.
The breakpoint = sm, md, lg, and xl.
Combining all the above, the left padding complete code can be (for example):
For left padding in extra small devices:
pl-2
or for medium to extra large:
pl-md-2
Now, let me show you the usage of these padding and margin shorthand utility classes in action in Bootstrap 4.

——

React- Redux

React state management library. It follows a functional (as in functional programming) style, with a heavy reliance on immutability. You'll create a single global store to hold all of the app's state. A reducer function will receive actions that you dispatch from your components, and respond by returning a new copy of state.
Because changes only occur through actions, it's possible to save and replay those actions and arrive at the same state. You can also take advantage of this to debug errors in production, and services
like LogRocket exist to make this easy by recording actions on the server.

functional programming. - Objects are constantly being created and destroyed. We do not change Bob; we create a clone, modify his clone, and then replace Bob with his clone.
immutable, then, is something that *cannot be changed*. - React prefers immutability.
React.PureComponent instead of React.Component. This way, the component will only re-render if its state is changed or if its *props have changed*. if you're passing props into a PureComponent, you have to make sure that those props are updated in an immutable way.
when you compare two objects or arrays with the === operator, JavaScript is actually comparing the *addresses* they point to – a.k.a. their *references*.
JS Memory Theory:
// This creates a variable, `crayon`, that points to a box (unnamed), // which holds the object `{ color: 'red' }`
let crayon = { color: 'red' };
// Changing a property of `crayon` does NOT change the box it points to
crayon.color = 'blue';
// Assigning an object or array to another variable merely points
// that new variable at the old variable's box in memory
let crayon2 = crayon;
console.log(crayon2 === crayon); // true. both point to the same box.
// Niw, any further changes to `crayon2` will also affect `crayon1` crayon2.color = 'green';
console.log(crayon.color); // changed to green! console.log(crayon2.color); // also green!
// ...because these two variables refer to the same object in memory
console.log(crayon2 === crayon);
const - will only prevent you from reassigning the reference. It doesn't stop you from changing the object. If I'm writing code where I know for certain I'll be mutating an array or object, I'll declare it with let
Redux: Update an Object
Redux requires that its reducers be *pure functions*. This means you can't modify the state directly – you have to create a new state based on the old one When you want to update the top-level properties in the Redux state object, copy the existing state with ...state and then list out the properties you want to change, with their new values.
function reducer(state, action) { // State looks like:
state = { clicks: 0, count: 0 }
return {
...state,
clicks: state.clicks + 1, count: state.count - 1
}}
Spread operator [...]:
the spread operator makes it easy to create a new object or array that contains the exact same contents as another one. This is useful for creating a copy of an object/array, and then overwriting specific properties that you need to change:
// Internal properties are left alone: let company = {

name: "Foo Corp",
people: [{name: "Joe"}, {name: "Alice"}]}
let newCompany = {...company};
newCompany === company // => false! not the same object
newCompany.people === company.people // => true!
Reducers
by updating state in an immutable way, Redux is able to tell which pieces of state changed, and optimize how your app is re-rendered. This function is nice because it is *predictable*. If you call it with the same arguments, you get the same outputs, every single time. It doesn't matter what else has changed in your app – this function will always act the same way. The store maintains an internal state variable. When an action is dispatched, the store calls the reducer, and replaces its internal state with whatever the reducer returned. Every time the store calls the reducer, it passes in the last-known state.
Redux: We dispatch an action. An action is a function that returns an object. Once we dispatch this, our reducer will take a look to see what action dispatched by looking at the name.
***FONT {font-sans-serif); font-size: 1rem; font-weight: 400;} ***

Django-React Heroku Deployment

==
Create .env
As mentioned above, the local version of the Django app is using db.sqlite3 as its database. However, when we visit the Heroku version, APP_NAME.herokuapp.com, Heroku will need to use a PostgreSQL database instead.
What we want to do is to get our app running with SQLite whenever we're working on it locally, and with Postgres whenever it's in production. This can be done using the installed python-dotenv library.
We will then use a file called .env to tell Django to use SQLite when running locally. To create .env and have it point Django to your SQLite database:
$ echo 'DATABASE_URL=sqlite:///db.sqlite3' > .env
Include the .env file inside our .gitignore when pushing to Heroku by running the following command:
$ echo '.env' >> .gitignore
STATIC_ROOT points to the directory containing all the static files, while STATICFILES_DIRS refers to other directories where Django will collect the static files as well. In this case, it is pointing to React's 'build/ static' directory which contains the static files for frontend when Heroku builds the React app using npm run build during deployment.
C. Static files MIME Type issue
Upon deploying the web app in Heroku, one of the common issues that occur is the static files failing to load due to MIME type limitations. The particular MIME type (text/html) problem is related to your Django configuration.
The views.py in your React frontend needs a content_type argument in the HttpResponse.

Heroku needs to know where the static files are.
The "refused to execute script ... MIME type ('text/html')" problem stems from Django's default content_type setting for an HttpResponse, which
is text/html.
This can be fixed by including a content_type='application/
javascript' argument in the return statement of a new class-based view called Assets(View) inside
What I’ve discovered is that STATIC_URL and STATIC_ROOT are actually overwritten by heroku to STATIC_URL = '/static/' and STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles'). Even if you have a
different STATIC_ROOT in your settings if you were to run heroku run python manage.py collectstatic, they will use staticfiles as
the STATIC_ROOT.

Superuser for djangp-react-boilerplate-app

Leonjmroe
leonjmroe@gmail.com
Roeincarnation12???

1nceuk superuser

Louisroe
louisroe33@gmail.com
Skateordie42

Git/Heroku

Git log (find your commit code)
Git checkout 8932489237493872
Git branch master -d (delete master)
Git branch master (creates master at HEAD)
Git checkout master (switches to new master and auto removes HEAD) Git push -f heroku master (force pushes new master to override heroic remote reopening that is ahead)

Django/Heroku

Your local database and your production database will differ!!
Heroku run python manage.py makemigrations Heroku run python manage.py migrate
Heroku run python manage.py createsuperuser

React

Repeat the calling of functions useCallback()
useMemo()

Work with Terminal windows and tabs

Action	Shortcut
New window	Command-N

New window with same command	Control-Command-N
New tab	Command-T
New tab with same command	Control-Command-T
Show or hide tab bar	Shift-Command-T
Show all tabs or exit tab overview	Shift-Command-Backslash (\)
New command	Shift-Command-N
New remote connection	Shift-Command-K
Show or hide Inspector	Command-I
Edit title	Shift-Command-I
Edit background colour	Option-Command-I
Make fonts bigger	Command-Plus (+)
Make fonts smaller	Command-Minus (–)
Next window	Command-Grave Accent (`)
Previous window	Command-Shift-Tilde (~)
Next Tab	Control-Tab
Previous Tab	Control-Shift-Tab
Split window into two panes	Command-D
Close split pane	Shift-Command-D
Close tab	Command-W
Close window	Shift-Command-W
Close other tabs	Option-Command-W
Close all	Option-Shift-Command-W
Scroll to top	Command-Home
Scroll to bottom	Command-End
Page up	Command-Page Up
Page down	Command-Page Down
Line up	Option-Command-Page Up
Line down	Option-Command-Page Down

Edit a command line

Action	**Shortcut**
Reposition the insertion point	Press and hold the Option key while moving the pointer to a new insertion point
Move the insertion point to the beginning of the line	Control-A
Move the insertion point to the end of the line	Control-E

Move the insertion point forwards one character	Right Arrow
Move the insertion point backwards one character	Left Arrow
Move the insertion point forwards one word	Option-Right Arrow
Move the insertion point backwards one word	Option-Left Arrow
Delete the line	Control-U
Delete to the end of the line	Control-K
Delete forwards to the end of the word	Option-D (available when Use Option as Meta key is selected)
Delete backwards to the beginning of the word	Control-W
Delete one character	Delete
Forward-delete one character	Forward Delete (or use Fn-Delete)
Transpose two characters	Control-T

Select and find text in a Terminal window

Action	**Shortcut**
Select a complete file path	Press and hold the Shift and Command keys and double-click the path
Select a complete line of text	Triple-click the line
Select a word	Double-click the word
Select a URL	Press and hold the Shift and Command keys and double-click the URL
Select a rectangular block	Press and hold the Option key and drag to select text
Cut	Command-X
Copy	Command-C
Copy without background colour	Control-Shift-Command-C
Copy plain text	Option-Shift-Command-C
Paste	Command-V
Paste the selection	Shift-Command-V
Paste escaped text	Control-Command-V
Paste escaped selection	Control-Shift-Command-V
Find	Command-F
Find next	Command-G

Find previous	Command-Shift-G
Find using the selected text	Command-E
Jump to the selected text	Command-J
Select all	Command-A
Open the character viewer	Control-Command-Space

Work with marks and bookmarks

Action	**Shortcut**
Mark	Command-U
Mark as bookmark	Option-Command-U
Unmark	Shift-Command-U
Mark line and send return	Command-Return
Send return without marking	Shift-Command-Return
Insert bookmark	Shift-Command-M
Insert bookmark with name	Option-Shift-Command-M
Jump to previous mark	Command-Up Arrow
Jump to next mark	Command-Down Arrow
Jump to previous bookmark	Option-Command-Up Arrow
Jump to next bookmark	Option-Command-Down Arrow
Clear to previous mark	Command-L
Clear to previous bookmark	Option-Command-L
Clear to start	Command-K
Select between marks	Shift-Command-A

Other shortcuts

Action	**Shortcut**
Enter or exit full screen	Control-Command-F
Show or hide colours	Shift-Command-C
Open Terminal preferences	Command-Comma (,)
Break	Typing Command-Full Stop (.) is equivalent to entering Control-C on the command line
Print	Command-P
Soft reset terminal emulator state	Option-Command-R
Hard reset terminal emulator state	Control-Option-Command-R
Open a URL	Hold down the Command key and double-click the URL
Add the complete path to a file	Drag the file from the Finder into the Terminal window
Export text as	Command-S

Export selected text as	Shift-Command-S
Reverse search command history	Control-R
Toggle “Allow Mouse Reporting” option	Command-R
Toggle “Use Option as Meta Key” option	Command-Option-O
Show alternate screen	Shift-Command-Down Arrow
Hide alternate screen	Shift-Command-Up Arrow
Open man page for selection	Control-Shift-Command-Question Mark (?)
Search man page index for selection	Control-Option-Command-Slash (/)
Complete directory or file name	On a command line, type one or more characters, then press Tab
Display a list of possible directory or file name completions	On a command line, type one or more characters, then press Tab twice

---

Git Control

Checkout on a previous branch:
You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.
If you want to create a new branch to retain commits you create, you may do so (now or later) by using -c with the switch command. Example:

```
  git switch -c <new-branch-name>
```

Or undo this operation with:

```
  git switch -
```

Git log - see all commits
Git checkout log_code - creates a head at selected commit
Git restore . - clears directory, nothing to commit
Git add . - adds changes ready for commit
Git reset - remove added files to stash (before commit)
Git revert commit_id - Reverts that commit
Git branch branch_name - creates a branch
Git branch -a - shows all branches
Git branch -d/-D - deletes a branch -D is hard delete
Git checkout -b main -creates new branch and checkout same time
If just a head and no master - git branch master, git checkout master - this removes the head and switches to master - back to normal

Git merge branch_name - merges selected branch to current one you are on
Git merge —abort - reverts the merge attempt
Git branch -m new_branch_name - renames branch of the one you are on
Git reset - used to reset conflicted files on a git merge
Git reset —hard - clears f
Git clean -fdx - clears untracked files

— locally store changes for when you want to move between branches and not commit
— after you run git stash apply, you can then run git stash drop as your changes are back in your directory
Git stash
Git stash -u (include untracked)
Git stash save "saved stash"
Git stash list
Git stash pop
Git stash apply
Git stash drop
Git mergetool

Delete and recreate remote repo - Heroku
Git push —force Heroku main
git remote rm heroku - delete Heroku if you get into a mess with remote branches git push —force
heroku git:remote -a example-app
git push heroku master:main - override master branch on main remote Heroku repo
heroku repo:reset --app appname - reset remote repo
Git remote - see remote repos

————————————

Terminal Control

————————————

Navigate to a specific directory in a command:
cd ~/Projects/environments
~/Projects/environments
mv dir1 dir2 (renameing)

Control-Z - quits shell
Control-L - clears terminal

Django alternative server
Python manage.py runserver 0.0.0.0:8001

Heroku/Local Repo differences:
git fetch heroku
git diff main..heroku/main

Overwrite local to remote?
git push heroku main:main --force

/———————————/
Db.SQLlite3
/———————————/

See what's in the Django DB in command line:
python3 manage.py dbshell (in directory of db.sqlite3 file is)
.tables (to see all db tables in format app_table)
.dump ?OBJECTS? (app_table) - to see all contents in that table
DROP TABLE (app_table); - deletes a table
DELETE FROM django_migrations WHERE App='(app_name)'; -deletes a table
Make migrations and migrate again then

—— Killing ports ———
Kill a NPM port - n
npx kill-port 3000
Kill a Django port - sudo lsof -t -i tcp:8000 | xargs kill -9

———————————————
Bash Profile
———————————————

Python versions on your system can be set in multiple ways.

1. Your system has its default python version installed.
2. You can set a alias path to point to another python version that you have installed. Your BASH profile is edited when you use the alias command to do this
3. You can use pyenv to point to a python version, this is a package that makes it easier to install and manage python versions. You can set peens local or global. Global takes precedence. Local is for a certain directory. Advise not setting local.

Virtual environments load up with a set python version. Typically this does not change, so you need to deactivate and activate a new VE if you want to change the python version

BASH profile is a body of text that you can edit in your shell and it holds meta instructions for your shell. It's either called bash or zrc.

To access on you Mac run: **nano ~/.zshrc**
When you are in, you can edit the text, once finished, CTRL + X, Y (to save), then ENTER (to exit bash file)

In the bash file you can set alias's eg: **alias python="/opt/homebrew/bin/python3.11"**

Or you can can point your shell to pyenv via:

export PYENV_ROOT="$HOME/.pyenv"
export PATH="$PYENV_ROOT/bin:$PATH"
eval "$(pyenv init --path)"

Which python - Command to see the source of the python version being used
pyenv versions - To view

Create a virtual env with s specified python version, getting the path via 'which python': **virtualenv -p /Users/leon/.pyenv/shims/python xbtenv**

Jupyter notebook using a different python path to my VE, big bug that threw me off for hours…

Check the path of jupyter: import sys
print(sys.executable)
Check the path of your VE: which python
If they differ:

Go into your VE
pip install ipykernel
python -m ipykernel install --user --name=xbtenv
jupyter notebook
Open a notebook then select the new kernel created from the drop down

The steps I provided earlier help align the Python interpreter in Jupyter Notebook with the one in your virtual environment by explicitly creating a kernel that uses the virtual environment's Python. This ensures that when you're working in a Jupyter Notebook, you have access to the same Python interpreter and installed packages as you do in your virtual environment. It's a way to make the environment consistent and predictable, avoiding the issues caused by the factors listed above.

These are the reasons why this could happen:

- **Default Configuration**: By default, Jupyter Notebook uses the Python interpreter that was used to install it. If Jupyter was installed outside the virtual environment, it would use the system Python or another version, not the one inside the virtual environment.
- **Kernel Selection**: Jupyter Notebook operates with different kernels, which are essentially different runtime environments. If you have multiple Python versions installed on your system, you might have different kernels for each version. When you start a new notebook, Jupyter might select a kernel that doesn't correspond to your virtual environment.
- **Path Configuration**: The PATH environment variable determines the order in which directories are searched for executables. If the system Python comes before the virtual environment's Python in the PATH, then commands like python or jupyter might resolve to the system versions rather than the virtual environment versions.
- **Virtual Environment Activation**: If the virtual environment is not activated when starting Jupyter Notebook, the system will not know to use the Python interpreter from the virtual environment.
- **Inconsistent Environment Management**: Mixing different environment management tools like pyenv, virtualenv, conda, etc., can lead to conflicts and unexpected behavior in selecting the correct Python interpreter.

Find the root of the directory:

```
import os
print(os.getcwd())
```

Heorku run local - gunicorn your_app_name.wsgi

```
heroku run bash
```

heroku run bash - enter a shell on the server, so you can navigate the hosted file system - exit command to get out of shell

Django shell - python manage.py shell
Then run commands to see/delete users:

```
from django.contrib.auth.models import User
print(User.objects.all())
```

```
from django.contrib.auth.models import User
user = User.objects.get(username='username_to_delete')
user.delete()
```

Git Cache

If git too big, then unwanted files probably added and need to be removed from the git cache first.
Then add the directory to the .gitignore. Then commit

```
git rm -r --cached frontend/node_modules/
git rm --cached -r "Module 4 - Applications of AI/Week 7"
git rm --cached -r .
```

Analyse repo size

```
Git gc
Git count-objects -vH
```

du -sh .git (see size of .git repo)
du -sh “Module 1 - Programming for AI/“ - this is just checking the size of the file, not what is being tracked by git

git gc --prune=now --aggressive (reachable commit. This is typically safe because it only removes Git objects that are not accessible by any branch, tag, or other references.
(The --aggressive option will more aggressively optimize the repository at the expense of taking more time to complete)

git clean -fdx (dangerous! Permanently deletes untracked files) run once of these first to see what will be deleted:
git clean -n / git clean --dry-run
git log --oneline --shortstat (show all commit changes by size/files)

(The whole below command - finds all large files sizes in entire git history, showing top 10):

```
git rev-list --objects --all | \
git cat-file --batch-check='%(objecttype) %(objectname) %(objectsize) %
(rest)' | \
sort -k 3 -n -r | head -n 10
```

(See the size of git items, top 10):

```
du -h .git | sort -hr | head -n 10
```

(Find which pack files are the largest):

```
ls -lh .git/objects/pack
```

(Find the items in the pack - hopeless command really - way to big of an output):
git verify-pack -v .git/objects/pack/pack-dbecd23cf977a66015cc87d7bb602851bd653fa1.idx

Git re-write history

du -sh .git   (see git size)

(Below removes all files in this directory frontend/node_modules/ from all of the git commits")
git filter-branch --force --index-filter \
"git rm --cached --ignore-unmatch -r frontend/node_modules/" \
--prune-empty --tag-name-filter cat -- --all

git gc --prune=now --aggressive (run garbage collection after removing files from git history)

Reverting a bad deployment

Git branch (create a branch name of latest deployment head that failed)
Git branch -D main
Git log to find last successful deployment commit ID
Git checkout commit_ID
Git branch main
Git checkout main

— Unsure if all the below is needed - deleting the remote repo from the last deployment and pushing the new main branch to the the remote to deploy
git remote remove heroku
heroku repo:reset --app app-1nceuk
heroku git:remote -a app-1nceuk
git push heroku main

Remote Branches
Git branch -r (see all remote branches)
Git branch -r -D branch_name

Heroku staging deployment

-Deploy to the staging app hosted on staging server to test deployment before going live

```
heroku create app-1nceuk-staging --remote staging
heroku config:set DISABLE_COLLECTSTATIC=1 -a your-staging-app-name
Git push staging main
heroku stack:set heroku-20 --app app-1nceuk
```

```
git push heroku nov_working_build:main
git push heroku nov_working_build:main --force
```

(This pushes a previous branch to the remote repo, overriding the main branch commit of heroku - heroic can only be pushed on main and master branches)

Package dependency issues

```
pip list --outdated
```

Git push -u origin main vs Git push origin main

The difference between git push origin main and git push -u origin main lies in the -u flag, which stands for --set-upstream. This flag tells Git to set the origin repository's main branch as the upstream (tracking) reference for the local main branch. Here's what each command does:

- git push origin main: Pushes commits from your local main branch to the main branch in the origin repository (which is typically your remote repository on platforms like GitHub, GitLab, or Bitbucket). This command simply performs the push action without altering any configuration or setting up tracking information.
- git push -u origin main: Does the same as the above but also sets the origin/main branch as the upstream for your local main branch. This has a couple of implications:
  - **Tracking Relationship**: By setting the upstream branch, you establish a tracking relationship between your local main branch and origin/main. This is useful for Git operations that rely on this information, such as git status, git pull, and git push, to understand the relationship between your local branch and its remote counterpart.
  - **Simplified Future Commands**: Once the upstream branch is set, you can simply use git push or git pull without specifying the remote name and branch for future operations, and Git will know

you're referring to origin/main. Git will use the tracking information to understand which remote branch corresponds to the branch you're currently on.

In summary, adding -u or --set-upstream when you first push a branch sets up a tracking relationship, making future commands more straightforward and providing helpful context to commands like git status. If you're working with a branch frequently or collaborating with others, setting the upstream branch can streamline your workflow and reduce the likelihood of pushing or pulling from the incorrect branch.

Linux

Rm -rf folder
Removes a folder by force

Git show/switch

Git show commit log code:path/to/file > saved_name_of_file
git ls-tree <commit-hash> <path-to-directory> - see the files in a previous commit
git checkout 844358d9543a7f3565b4ed5b7ce5550b0eccdc3b -- frontend/src/components
(Above brings in a whole directory from another commit into working director/current branch)
Git switch -c new-branch (this creates a new branch and checks out branch - good use when at a head after running git branch -D main, so you can checkout main on a new branch)
Git push heorku main —force (this may be needed if your remote branch is ahead of new local checkedout main branch from head)

Pip packages

Pip list —outdated (fund all packages that are not up to date)
pip list --outdated | grep -v 'Package' | awk '{print $1}' | xargs -n 1 pip install --upgrade
(Above updates all outdated packages)
pip freeze | grep -vE '^(pip|setuptools|wheel)==' | xargs pip uninstall -y
(Above removes all packages from venv)
Pip check - (This command will scan your current environment and report any inconsistencies between package requirements and the packages you have installed.)

Creating new origin repo (GitHub)

Create a new repo in GitHub
Copy the repo link

git remote add origin https://github.com/Leonjmroe/MSc-AI.git
Git push -u origin master

.gitignore

Module 4 - Applications of AI/Week 7/**/*.csv
(In git ignore, ignores all csvs in subfolders of week 7)