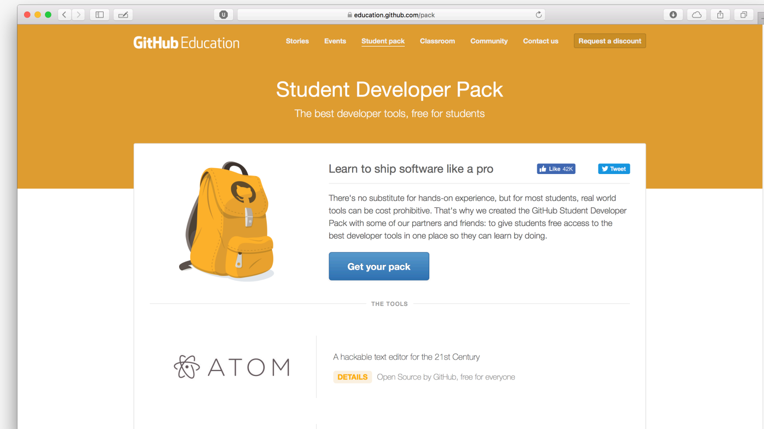
Back-end development

Tiberiu Vilcu
Prepared for EECS 411
Sugih Jamin
13 September 2017



https://education.github.com/pack

Outline

1. Design an example API for “Chatter”
2. Create a DigitalOcean Droplet to serve the API
3. Set up a database to store our data
4. Create web API endpoints to connect from anywhere
5. Test endpoints to verify and check for errors
6. Discuss Firebase and Heroku as alternatives

Example API: Chatter

```
/registeruser/
<– username, name, email
–> {} 200 OK

/addchatt/
<– username, message
–> {} 200 OK

/getchatts/
<– Nothing
–> list of chatts 200 OK
```

DigitalOcean

- Is a simple cloud computing service
- Deploys “Droplets” to run machines
- Has various easy-to-install applications (Django, MongoDB, Node.js)
- Is not free but gives $50 credit with GitHub developer pack

https://www.digitalocean.com

Why DigitalOcean?

- Simple to code, in Python
- Nothing to install locally
- Little work necessary for bare minimum product
- Can also create dynamic webpages
- Most flexible back-end option, can serve all platforms
- Custom API generally easiest to use among team

Outline

1. Design an example API for “Chatter”
2. Create a DigitalOcean Droplet to serve the API
3. Set up a database to store our data
4. Create web API endpoints to connect from anywhere
5. Test endpoints to verify and check for errors
6. Discuss Firebase and Heroku as alternatives

Creating a Droplet

Use these options:

- Django 1.8.7 on 16.04
- $5 per month size (free with student developer pack)
- New York datacenter 1 or 3

https://cloud.digitalocean.com/droplets

Accessing the Droplet

1. IP address, username and password are emailed to you
2. `ssh root@[IP]`
3. Change default password upon login
4. Can access IP address's webpage in browser

https://www.digitalocean.com/community/tutorials/how-to-use-the-django-one-click-install-image-for-ubuntu-16-04

How the server works

nginx: web server that listens on port 80
configuration: `/etc/nginx/sites-enabled/django`

gunicorn: serves Django project on port 9000
configuration: `/etc/init/gunicorn.conf`

django: Python code framework that is run to do the work
location: `/home/django/django_project`

Outline

1. Design an example API for "Chatter"
2. Create a DigitalOcean Droplet to serve the API
3. Set up a database to store our data
4. Create web API endpoints to connect from anywhere
5. Test endpoints to verify and check for errors
6. Discuss Firebase and Heroku as alternatives

Setting up the database

1. `sudo -u postgres psql` (use PSQL as user postgres)
2. `\connect django` (connect to the database)
3. `GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO django;`
4. `\dt` (list tables)
5. Use SQL commands from here to do what you need
6. Control+D to exit

https://www.w3schools.com/sql/

Creating tables

```
CREATE TABLE users (username varchar(255), name
varchar(255), email varchar(255));

CREATE TABLE chatts (username varchar(255), message
varchar(255), time timestamp DEFAULT CURRENT_TIMESTAMP);
```

Adding a dummy entry

```
INSERT INTO chatts values('testuser1', 'Hello world');

SELECT * from chatts;
```

Outline

1. Design an example API for “Chatter”
2. Create a DigitalOcean Droplet to serve the API
3. Set up a database to store our data
4. Create web API endpoints to connect from anywhere
5. Test endpoints to verify and check for errors
6. Discuss Firebase and Heroku as alternatives

Setting up Django

```
python manage.py startapp chatter
```

Creates a directory “chatter” and the necessary files

Adding a view

In `chatter/views.py`

```
from django.http import JsonResponse, HttpResponse

def getchatts(request):
    if request.method != 'GET':
        return HttpResponse(status=404)
    response = {}
    response['chatts'] = ['Hello', 'World']
    return JsonResponse(response)
```

Updating the URLs

In `django_project/urls.py`

```
from chatter import views

urlpatterns = [
    url(r'^getchatts/$', views.getchatts,
name='getchatts'),
    url(r'^admin/', include(admin.site.urls)),
]
```

Running the application

- `service gunicorn restart`
  restart the application completely
- `python manage.py runserver localhost:9000`
  let it restart every time a change is detected
  not good for production

Creating the rest of the views

- getchatts
  Query the database and return all found chatts
- registeruser
  Parse JSON parameters and insert a user into the database
- addchatt
  Parse JSON parameters and insert a chatt into the database

Connecting to the database

In django_project/settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'django',
   'USER': 'django',
   'PASSWORD': '[given password]',
   'HOST': 'localhost',
   'PORT': '',
    }
}
```

Using database cursors

```
from django.db import connection


cursor = connection.cursor()
cursor.execute('SELECT * FROM chatts;')
rows = cursor.fetchall()
```

Parsing URL parameters

`http://website.com/path/`**`?key=val&key2=val`** are parameters

Do not need to specify them in the URLs file

```
numentries = request.GET.get('numentries', None)
```

Retrieves the argument with key ‘numentries’, or None if does not exist

Parsing JSON parameters

```
import json

json_data = json.loads(request.body)
username = json_data['username']
```

Preventing CSRF errors

Django wants CSRF (cross-site request forgery) cookies by default

```
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def addchatt(request):
    pass
```

Finishing addchat

```
@csrf_exempt
def addchatt(request):
    if request.method != 'POST':
        return HttpResponse(status=404)
    json_data = json.loads(request.body)
    username = json_data['username']
    message = json_data['message']
    cursor = connection.cursor()
    cursor.execute('INSERT INTO chatts (username, message) VALUES '
                   '(%s, %s);', (username, message))
    return JsonResponse({})
```

Adding adduser

```
@csrf_exempt
def adduser(request):
    if request.method != 'POST':
        return HttpResponse(status=404)
    json_data = json.loads(request.body)
    username = json_data['username']
    name = json_data['name']
    email = json_data['email']
    cursor = connection.cursor()
    cursor.execute('INSERT INTO users (username, name, email) VALUES '
                   '(%s, %s, %s);', (username, name, email))
    return JsonResponse({})
```

Finishing the URLs

```
urlpatterns = [
    url(r'^getchatts/$', views.getchatts, name='getchatts'),
    url(r'^addchatt/$', views.addchatt, name='addchatt'),
    url(r'^adduser/$', views.adduser, name='adduser'),
    url(r'^admin/', include(admin.site.urls)),
]
```

Outline

1. Design an example API for “Chatter”
2. Create a DigitalOcean Droplet to serve the API
3. Set up a database to store our data
4. Create web API endpoints to connect from anywhere
5. Test endpoints to verify and check for errors
6. Discuss Firebase and Heroku as alternatives

Testing the API

- Can test GET requests through the browser URL
- Use Postman to test POST and PUT (and other) requests
- Free app / Chrome extension to create custom requests

https://www.getpostman.com

Adding JSON to request body

1. Set request type to POST
2. In “Body” select raw form
3. Change type to “application/json”
4. Type in valid JSON

```
{
  "username": "tv",
  "message": "Hello world! Live demo right here!"
}
```

Takeaways

- Now have working API server
- Mobile app can interact with GET/POST requests
- Easy to extend database and endpoints
- Easy to add functionality / fix correctness

Stability improvements

Don't want 500 INTERNAL SERVER ERROR

- Validate URL/JSON parameters and values
- Validate entries in database before/after adding
- Return correct error codes for caught exceptions

Security improvements

- Remove the CSRF token workaround
- Prevent SQL injection
- Don't allow any user to POST on behalf of anyone else
- If using passwords: salt and hash
- Never write your own crypto/hashing code, use Python libs

Other improvements

- Document API for team!
- Verify user/item exists before posting
- Add option to limit number of chatts to return
- Integrate with other API's (perhaps Facebook for user accounts)
- Server scripts for database management

Outline

1. Design an example API for "Chatter"
2. Create a DigitalOcean Droplet to serve the API
3. Set up a database to store our data
4. Create web API endpoints to connect from anywhere
5. Test endpoints to verify and check for errors
6. Discuss Firebase and Heroku as alternatives

Firebase as an alternative

https://firebase.google.com

- Realtime database (NoSQL, use JSON)
- User authentication
- Cloud storage and functions
- Support for iOS, Android, web apps
- Online account management (also through Node.js)

Limits: don't have "console" control, stuck with their API

Using Firebase on mobile apps

1. Register app through bundle ID or package name
2. Download configuration files from Firebase
3. Add Firebase SDK to project
4. Use their referenced API to interact with services

Firebase database

- JSON formatted data (not structured in tables)
- Allows rules for authentications and permissions
- Automatically backs up (with paid plan)
- Can select and insert items by keys and sub-keys

Firebase database JSON

```
{
    "users": [{
            "username": "testuser1",
            "email": "test1@user.com",
            "password": "fakepass"
        },
        {
            "username": "testuser2",
            "email": "test2@user.com",
            "password": "fakepass"
        }
    ],
    "chatts": [{
            "username": "testuser1",
            "message:": "This is my first chatt",
            "timestamp": "right now"
        },
        {
            "username": "testuser2",
            "message:": "This is also my first chatt",
            "timestamp": "a little while ago"
        }
    ]
}
```

Heroku as an alternative

https://www.heroku.com/home

Myriad of products:

- Platform: managed containers with runtime environments and ecosystems
- Postgres: SQL database service
- Redis: key-value data store
- Kafka: distributed data streams (not very relevant)

Heroku as an alternative

Things to note:

- Offers analogous features to Firebase and DigitalOcean
- Harder to get started with for the less experienced
- Confusing to know which product works best
- Requires local development setup (deploy from git)

Using Heroku on mobile apps

Has "Sinatra" API for iOS but requires attaching SDK

Can create a REST API server on their stack with Node.js and MongoDB:

- Set up app environment
- Provision a database and connect to the server
- Create API endpoints with Node.js and Express
- Create client-side contract to interact with API

AWS alternatives

- AWS Lambda: run your own serverless code on AWS
- AWS EC2: cloud computing service

Various pricing tiers per service; free or paid with GitHub credit

Resources

- Project sample repo: https://github.com/UM-EECS-441/django-project-sample-f17
- GitHub student pack: https://education.github.com/pack/
- DigitalOcean: https://www.digitalocean.com
- Django setup: https://www.digitalocean.com/community/tutorials/how-to-use-the-django-one-click-install-image-for-ubuntu-16-04
- SQL W3 tutorials: https://www.w3schools.com/sql/
- Postman: https://www.getpostman.com
- Firebase: https://firebase.google.com
- Firebase API reference: https://firebase.google.com/docs/reference/
- Heroku: https://www.heroku.com/home
- Heroku iOS tutorial: https://devcenter.heroku.com/articles/getting-started-ios-development-sinatra-cedar
- Heroku REST API tutorial: https://devcenter.heroku.com/articles/mean-apps-restful-api
- AWS Lambda: https://aws.amazon.com/lambda/
- AWS EC2: https://aws.amazon.com/ec2/