

FullStack Development

CS571: Building User Interfaces

Cole Nelson

Today's Warmup

1. Clone `today's starter code` and run `npm install` in `starter/frontend`, in `starter/backend`, in `solution/frontend`, in `solution/frontend`, and in `deployment`.
 - a. Consider using `pnpm` for future development :)
2. Download & install `Docker`

Please Note

Today's content *will be* on the final exam, but there is *not* an associated HW assignment.

Bonus Quiz + CTF points added *outside* of Canvas.

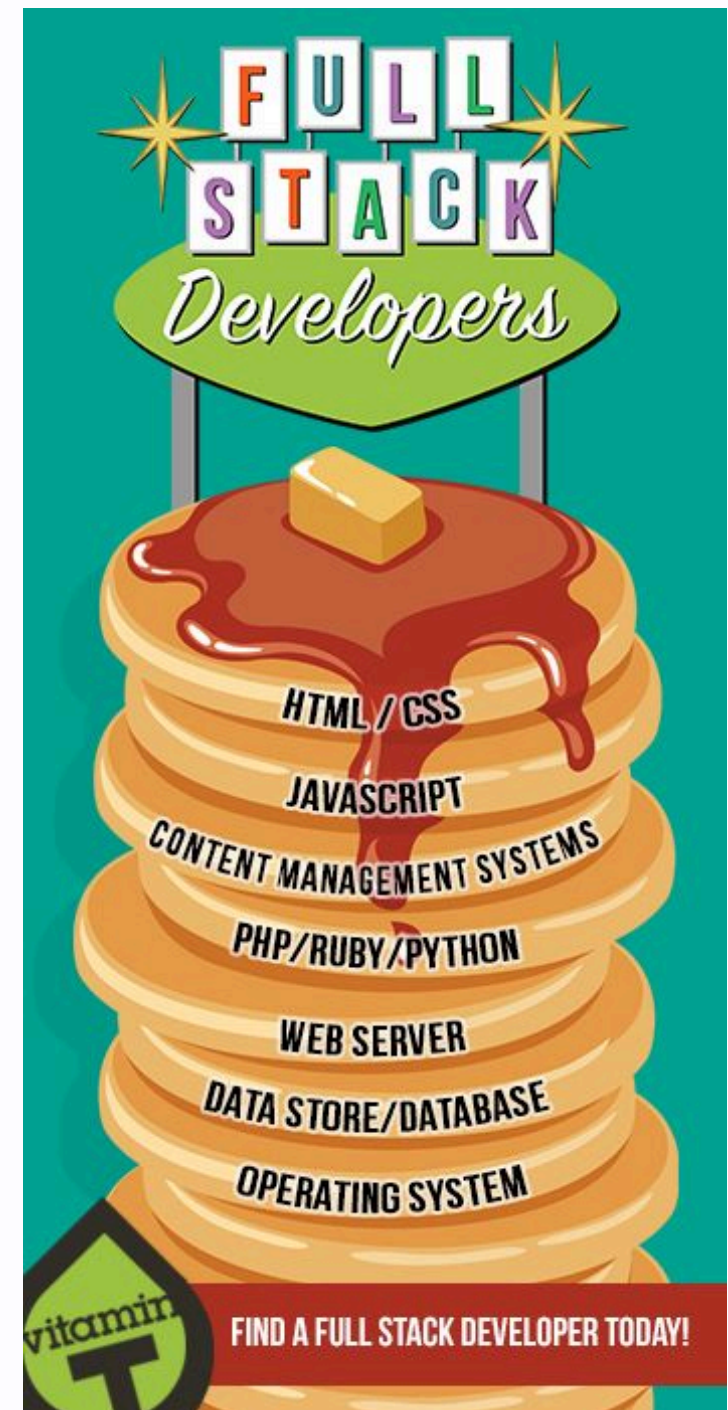
Learning Objectives

1. Be able to define how backend development fits into the software stack.
2. Be able to develop a backend.
3. Be able to persist data.
4. Be able to make other considerations such as containerization.

Software Stack

Think of software like a stack of pancakes...

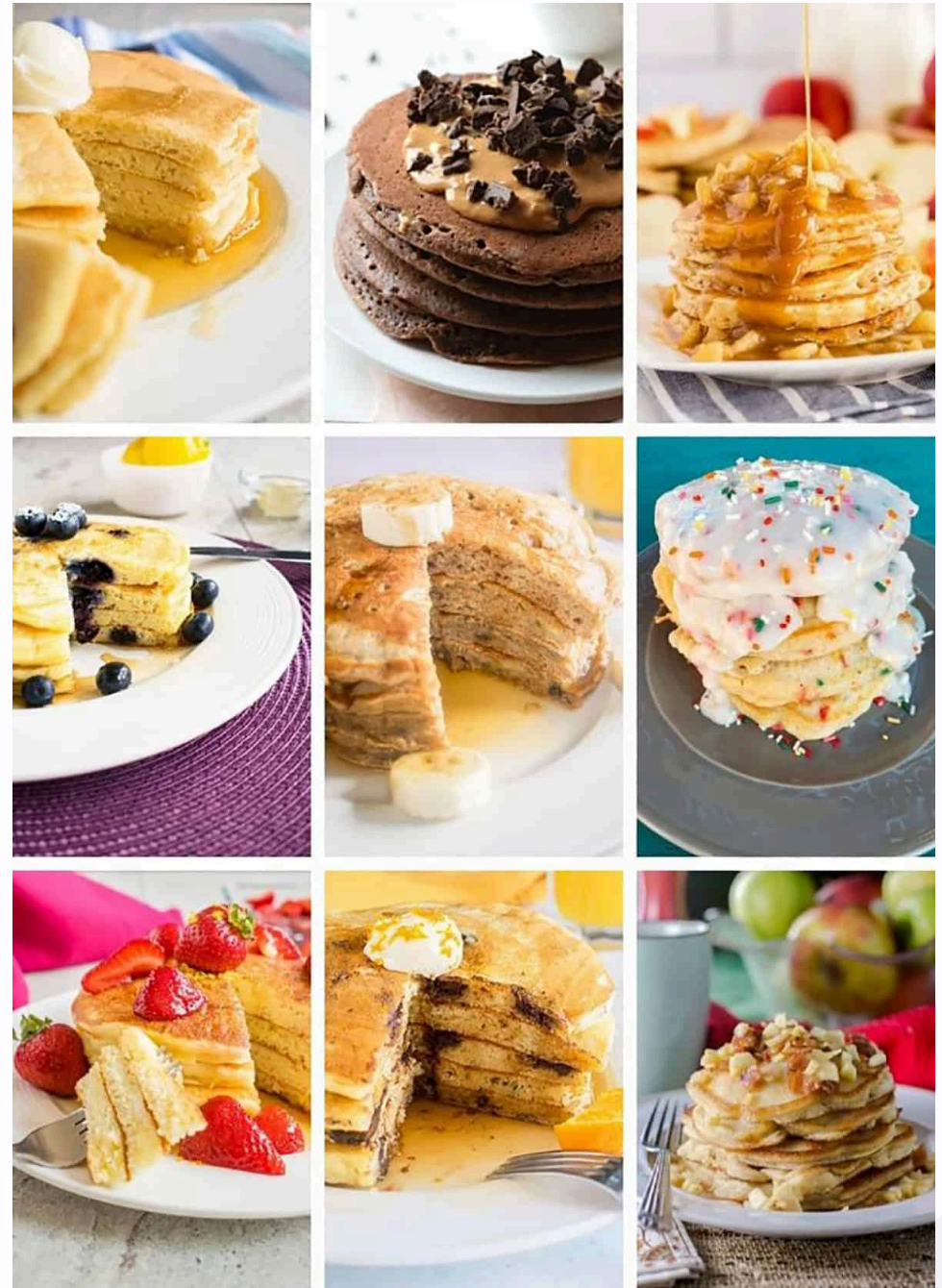
[Image Source](#)



Software Stack

... where each pancake
can be its own flavor...

Image Source



Software Stack

... and can be cooked its own way...

Image Source

PANCAKES RECIPE

1. EGGS

2. FLOUR

3. MILK

4. SUGAR

5. COOKING OIL

6. SALT

7. BUTTER

8. FRYING PAN

9. WHISK

10. SPATULA

11. BOWL

12. CUP

13. SPOON



Software Stack

... with as many or as few as we want!

Image Source



Our Software Stack

JavaScript and React for frontend development.

JavaScript and Express for backend development.

When you build your project, you get to choose your software stack!

Creating a Backend Server

Many, many, many options!

- Google Cloud Functions
- AWS Lambdas
- C# & .NET
- Java & Spring
- Python & Flask
- **JavaScript & Express**

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

Taken from [ExpressJS](#)

ExpressJS

Define every *endpoint* with a *callback handler*.

```
app.get('/messages', (req, res) => {  
  res.send('I should get the messages.')  
})
```

```
app.post('/messages', (req, res) => {  
  res.send('I should create a message.')  
})
```

Notice! We use `res` rather than `return`.

ExpressJS

We prefer JSON! :)

```
app.get('/messages', (req, res) => {  
  res.send({msg: 'I should get the messages.'})  
})
```

```
app.post('/messages', (req, res) => {  
  res.send({msg: 'I should create a message.'})  
})
```


ExpressJS

Parameters in `req` , response in `res` .

```
app.get('/messages', (req, res) => {  
  const chatroom = req.query.chatroom;  
  if (chatrooms.includes(chatroom)) {  
    res.status(200).send({msg: `I should get the messages.`})  
  } else {  
    res.status(404).send({msg: 'Could not find specified chatroom.'})  
  }  
})
```

ExpressJS

Parameters in `req`, response in `res`.

```
app.post('/messages', (req, res) => {  
  const comment = req.body.comment;  
  if (comment) {  
    res.status(200).send({msg: `I should create this comment.`})  
  } else {  
    res.status(400).send({msg: 'You must specify a comment.'})  
  }  
})
```

ExpressJS Middleware

use some middleware that can read the req and modify the res before proceeding to the next

```
app.use((req, res, next) => {  
  res.header("Access-Control-Allow-Origin", req.headers.origin);  
  res.header("Access-Control-Allow-Headers", req.headers["access-control-request-headers"]);  
  res.header('Access-Control-Allow-Methods', req.headers["access-control-request-method"]);  
  res.header('Access-Control-Allow-Credentials', 'true');  
  res.header('Access-Control-Expose-Headers', 'Set-Cookie');  
  res.header('Vary', 'Origin, Access-Control-Allow-Headers, Access-Control-Allow-Methods')  
  next();  
});
```

Applied in the order of use

Your Turn!

Build the BadgerChat Nano API.

How to persist data?

Let's use SQLite

SQLite

- SQL, but lite!
- Creates a `.db` file on your machine
- Is not a "hosted" database, but is good for quick projects and hacks!

```
const db = await open({  
  filename: "./db.db",  
  driver: sqlite3.Database  
});
```

SQL 101

Interact with the database via string queries.

```
-- Get all comments.  
SELECT * FROM BadgerComment;  
  
-- Get a specific comment.  
SELECT * FROM BadgerComment WHERE id = ?;  
  
-- Create a comment and return its id.  
INSERT INTO BadgerComment(comment, created) VALUES (?, ?) RETURNING id;  
  
-- Delete a specific comment.  
DELETE FROM BadgerComment WHERE id = ?;
```

SQL w/ ExpressJS

Common functions incl. `exec` , `run` , `get` , and `all` .

```
// Run arbitrary queries, disregarding the results.  
await db.exec('CREATE TABLE BadgerComment');  
  
// Run a query, disregarding the results.  
await db.run('DELETE FROM BadgerComment WHERE id = 4;');  
  
// Get the first row back as an object.  
const datum = await db.get('SELECT * FROM BadgerComment WHERE id = 7');  
  
// Get all rows back as a list of objects.  
const data = await db.all('SELECT * FROM BadgerComment');
```

Your Turn!

Build the BadgerChat Nano API with a database.

SQL Prepared Statements

Concatenating SQL queries with user input is **very bad**.
Prefer to use prepared statements.

```
// Use prepared statements instead!  
await db.get('SELECT * FROM BadgerComment WHERE id = ?', 7)
```

`?` is sanitized and interpolated with the arguments following the SQL query.

Your Turn!

Build the BadgerChat Nano API with a database using prepared statements.

This is great but...

...how can we deploy this?

Deployment

Run the setup commands, then...

1. Open the ports on your machine & router (or use a reverse proxy tool like [ngrok](#)).
2. Open the ports on a remote machine.

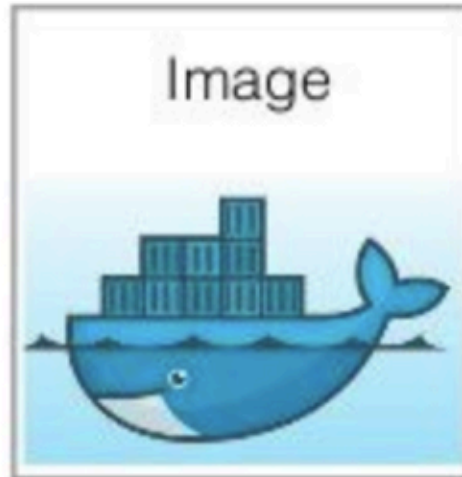
Still... how do we *isolate* ourselves? How do we make the environment *portable*? **Use a VM or a container!**



```
FROM ubuntu:14.04
MAINTAINER John Doe <john.doe@example.com>
RUN apt-get update
RUN apt-get install -y python-pip
RUN pip install Flask
RUN pip install gunicorn
RUN gunicorn -w 1 -b 0.0.0.0:8080 flask:app
CMD ["python", "flask.py"]
```

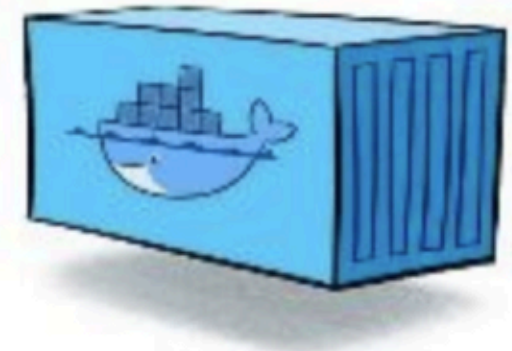
Dockerfile

build



Docker Image

run



Docker Container

Image Source

Docker Commands

`docker build` builds an image from a given Dockerfile

`docker run` runs a container from a given image

`docker stop` stops a running container

`docker rm` removes a stopped container

`docker push` pushes an image

`docker pull` pulls an image

Demo

Building, pushing, pulling, and running a Docker image as a container on a remote machine.

Backend Server Hosting



Amazon
EC2



DigitalOcean



Not an endorsement of any particular service.

Other Considerations

- Use [Jenkins](#) or some other CI/CD platform to create a [build and deploy pipeline](#).
 - Include testing as an automated step.
- Use HTTPS for a secure HTTP connection.
 - Consider [LetsEncrypt](#).
- Use TypeScript instead of JavaScript!
- Buy a domain name?
 - Completely optional!

Questions?