# Web Application A to Z

Moshe Zadka – https://cobordism.com

PyBay 2018

# Goal

- Understand high-level
- Get needed skills
- ...but perfecting them takes a lifetime

# Web applications

- Front-end (JavaScript)
- Back-end (Server)
- Storage (Database)

# Web applications - Front-End

React – JavaScript framework

# Web applications - Back-End

- Python
- WSGI
- Pyramid

# Web applications - Storage

- SQLite
- SQLalchemy

# Web applications - Platform

- Twisted
- Linux

# Pyramid: Imports

```
from pyramid import config, response
```

# Pyramid: Logic

```
def hello_world(request):
    return response.Response('Hello World!')
```

# Pyramid: Routing

```
with config.Configurator() as cfg:
    cfg.add_route('hello', '/')
    cfg.add_view(hello_world, route_name='hello')
    app = cfg.make_wsgi_app()
```

# Pyramid: Running

```
$ python -m twisted web --wsgi hello.app
```

# Returning Posts: Hardcoding

```
POSTS = [
    "Just chillin'",
    "Writing code",
    "Being awesome",
]
```

# Returning Posts: Logic

```
def posts(request):
    body = json.dumps(POSTS).encode('utf-8')
    return response.Response(
                content_type='application/json',
                body=body)
```

# Adding Post: Logic

```
def add_post(request):
    POSTS.append(request.json_body['content'])
    return response.Response('ok')
```

# Returning Posts: Routing

```
with config.Configurator() as cfg:
    cfg.add_route('posts', '/posts',
                  request_method='GET')
    cfg.add_view(posts, route_name='posts')
    cfg.add_route('add_post', '/posts',
                  request_method='POST')
    cfg.add_view(add_post, route_name='add_post')
    app = cfg.make_wsgi_app()
```

# Hands on: Add Author

15 minutes :25-:40

- Get application to run as-is
- Add an author field to post
- Finished? Add a date field
- Finished? Allow updating a post

# Using React: Header

```
<!DOCTYPE html>
<html>
<head>
<script src="https://fb.me/react-0.14.1.js">
</script>
<script src="https://fb.me/react-dom-0.14.1.js">
</script>
<script src=
"https://unpkg.com/babel-core@5.8.3/browser.min.js"
></script>
</head>
```

# Using React: Body

```
<body>
<div id="content"></div>
```

# Using React: Code

```
<script type="text/babel">
const element = (
  <h1>
    Hello
  </h1>
);
ReactDOM.render(
  element,
  document.getElementById('content')
);
</script>
```

# Static and Dynamic

- HTML, JavaScript are "static"
- Other URLs are dynamic
- Proxy in front of WSGI

# Static and Dynamic

- HTML, JavaScript are "static"
- Other URLs are dynamic
- Proxy in front of WSGI
- Custom Twisted plugin

# Static Files: Imports

```
import os

from zope import interface

from twisted.python import (usage, reflect,
                            threadpool, filepath)
from twisted import plugin
from twisted.application import (service,
                                 strports,
                                 internet)
from twisted.web import (wsgi, server,
                         static, resource)
from twisted.internet import reactor

import blog.app
```

# Static Files: Resource

```
class DelegatingResource(resource.Resource):

    def __init__(self, wsgi_resource):
        resource.Resource.__init__(self)
        self.wsgi_resource = wsgi_resource

    def getChild(self, name, request):
        print("Got getChild", name, request)
        request.prepath = []
        request.postpath.insert(0, name)
        return self.wsgi_resource
```

# Static Files: Pool

```
def getPool(reactor):
    pool = threadpool.ThreadPool()
    reactor.callWhenRunning(pool.start)
    reactor.addSystemEventTrigger('after',
                                  'shutdown',
                                  pool.stop)
    return pool
```

# Static Files: Root

```
def getRoot(pool):
    application = blog.app.app
    wsgi_resource = wsgi.WSGIResource(reactor,
                                      pool,
                                      application)
    root = DelegatingResource(wsgi_resource)
    static_resource = static.File('index.html')
    root.putChild(b'', static_resource)
    static_resource = static.File('static')
    root.putChild(b'static', static_resource)
    return root
```

# Static Files: Service

```
class Options(usage.Options):
    pass

def makeService(options):
    pool = getPool(reactor)
    root = getRoot(pool)
    site = server.Site(root)
    ret = strports.service('tcp:8080', site)
    return ret
```

# Static Files: Running

```
$ python -m twisted blog
```

# Hands On: Custom Plugin

:50-65
15 minutes

- Get it to run
- Finished? Add a favicon. (Hint: `/favicon.ico`)
- Finished? Add posts with requests
- Finished? Add parameter to control the port

# Listing Posts: Constructor

```
class Posts extends React.Component {
  constructor(props) {
    super(props);
    this.state = { posts: [] };
    this.handleSubmit =
      this.handleSubmit.bind(this)
    this.handleInputChanged =
      this.handleInputChanged.bind(this)
  }
```

# Listing Posts: Data fetch

```
componentDidMount() {
  fetch("/posts").then(
    response => response.json()).then(
    response => this.setState({posts: response}))
}
```

# Listing Posts: Rendering

```
render() {
  return <div>
         <ul>
         {this.state.posts.map(post =>
           <li>{post[0]}: {post[1]}</li>
          )}
         </ul>
         {this.getForm()}
         </div>
}
```

# Adding Post: Form

```
getForm() {
  return <form onSubmit={this.handleSubmit}>
    <p><label>Post:</label>
    <input type="text" name="post"
      onChange={this.handleInputChanged}/></p>
    <p><label>Author:</label>
    <input type="text" name="author"
      onChange={this.handleInputChanged}/></p>
    <p><input type="submit" value="Post"/></p>
    </form>
}
```

# Adding Post: Manage Input

```
handleInputChanged(event) {
  this.setState({[event.target.name]:
                 event.target.value});
  event.preventDefault();
}
```

# Adding Post: Sending Data

```
handleSubmit(event) {
  fetch("/posts",
  {
    headers: {"Content-Type":
              "application/json"},
    method: "POST",
    body: JSON.stringify(
            {author: this.state.author,
             content: this.state.post})
```

# Adding Post: Refreshing

```
    }).then(fetch("/posts").then(
            response => response.json()).then(
            response =>
              this.setState({posts: response})));
    event.preventDefault();
  }
```

# Hands On: Refreshing Posts

:70-90 (20m)

- Get it to run as is
- Add a button to refresh posts
- Finished? Add a loop to refresh posts every 5 seconds
- Finished? Allow turning the loop off and on
- Finished? Allow setting the interval time

# Backing Database: Imports

```
import contextlib
import os

import sqlalchemy
from sqlalchemy import sql
```

# Backing Database: Configuration

```
metadata = sqlalchemy.MetaData()

create_all = metadata.create_all
```

# Backing Database: Location

```
def get_engine():
    db = os.environ['BLOG_DATABASE']
    return sqlalchemy.create_engine(db)
```

# Backing Database: Schema

```
posts = sqlalchemy.Table('posts', metadata,
                         sqlalchemy.Column('content',
                                           sqlalchemy.String),
)
```

# Backing Database: Context

```
def context_connect(engine):
    return contextlib.closing(engine.connect())
```

# Backing Database: Retrieval

```
def get_posts(engine):
    with context_connect(engine) as conn:
        select = sql.select([posts])
        cursor = conn.execute(select)
        return list(cursor)
```

# Backing Database: Storing

```
def add_post(engine, content):
    with context_connect(engine) as conn:
        insert = posts.insert()
        ins_content = insert.values(
                                content=content)
        conn.execute(ins_content)
```

# Backing Database: App Changes: Configuration

```
with config.Configurator() as cfg:
    engine = storage.get_engine()
```

# Backing Database: App Changes: Retrieval

```
engine = request.registry.settings['engine']
posts = [(content, 'anonymous')
         for content,
             in storage.get_posts(engine)]
```

# Backing Database: App Changes: Storage

```
engine = request.registry.settings['engine']
content = request.json_body['content']
storage.add_post(engine, content)
```

# Backing Database: Initialization

```
$ BLOG_DATABASE=sqlite:///blog.db \
  python -c \
  'from blog.storage import *
  create_all(get_engine())'
```

# Backing Database: Hands-On

:125-:140 (15m)

- Get it to run
- Add author
- Finished? Normalize author to different table
- Finished? Add auto-calculated date

# Webpack

Real JavaScript developers minify and transpile and...

# Webpack: Breaking Out JS

```
const ReactDOM = require('react-dom');
const React = require('react');

class Posts extends React.Component {
  constructor(props) {
```

# Webpack: Yarn

Yarn manages JavaScript environments

# Webpack: Yarn: Configuration

```
{
  "devDependencies": {
    "webpack": "^4.12.0",
    "webpack-cli": "^3.0.8"
  },
  "dependencies": {
    "@material-ui/core": "^1.3.0",
    "babel-core": "^6.26.3",
    "babel-loader": "^7.1.4",
    "babel-preset-react": "^6.24.1",
    "react": "^16.4.1",
    "react-dom": "^16.4.1"
  },
  "private": true
}
```

# Webpack: Yarn: Install

```
$ yarn install
```

# Webpack: Configuration

```
const path = require('path');
const webpack = require('webpack');

module.exports = {
  entry: './src/index.jsx',
  output: {
    path: path.resolve('dist'),
    filename: 'index_bundle.js'
  },
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        exclude: /node_modules/,
        use: "babel-loader",
      }
    ]
  }
```

# Webpack: Babel configuration

```
{
  "presets": ["react"]
}
```

# Webpack: Running

```
$ yarn webpack
```

# Webpack: Layout

```
$ ls -l static/dist
... static/dist -> ../dist/
```

# Webpack: Bootstrap

```
<!DOCTYPE html><html>
<head><meta charset="utf-8"/></head>
<body><div id="content"></div>
<script
src="/static/dist/index_bundle.js"
></script>
</body></html>
```

# Webpack: Hands-On

:160-:180

- Get it to run
- Finished? Generate source maps.

# Styling

Look and feel

# CSS

- Classes
- Selectors
- Styles

# CSS and JavaScript/React

- CSS styles
- JavaScript makes components
- Interactions can be surprising...
- ...and hurt modularity

# Material UI 1.0

- JavaScript +
- CSS +
- Design

# Material UI: Imports

```
const React = require('react');
import Button from
       '@material-ui/core/Button';
import TextField from
       '@material-ui/core/TextField';
import List from
       '@material-ui/core/List';
import ListItem from
       '@material-ui/core/ListItem';
import ListItemText from
       '@material-ui/core/ListItemText';

class Posts extends React.Component {
  constructor(props) {
```

# Material UI: Form

```
getForm() {
   return <form onSubmit={this.handleSubmit}>
    <p><TextField inputProps={{id: "post"}}
          label="Post"
          onChange={this.handleInputChanged}/></p>
    <p><TextField inputProps={{id: "author"}}
          label="Author"
          onChange={this.handleInputChanged}/></p>
    <p><Button variant="contained"
          onClick={this.handleSubmit}>
          Post</Button></p>
      </form>
}
```

# Material UI: Posts

```
render() {
  return <div>
         <List>
         {this.state.posts.map(post =>
           <ListItem><ListItemText
                          primary={post[0]+
                                   ": " +
                                   post[1]}
           /></ListItem>
         )}
         </List>
         {this.getForm()}
         </div>
}
```

# Hands On: Add elements

:200-:215 (15m)

- Get it to run
- Add a title
- Finished? Add a bar
- Finished? Play with alignments
- Finished? Play with colors

# Web application, A to..what?

What's missing?

- Packaging (with Docker)
- Testing (with Selenium)
- Continuous integration (CircleCI, Jenkins, etc.)
- Deployment (K8s)
- Continuous Deployment (CI + K8s)
- TLS – certificate management and rotation
- Monitoring – Pinging, logging, metrics, stack traces, alerting
- Redundancy
- Data migration
- Backup

# Summary

- SQLalchemy
- Pyramid
- Twisted
- React
- Webpack
- Material UI

# Questions