# COMP30670
# Software Engineering (Conversion)

## 16 Flask/WebApp
## 31/03/2017

*Dr. Aonghus Lawlor*

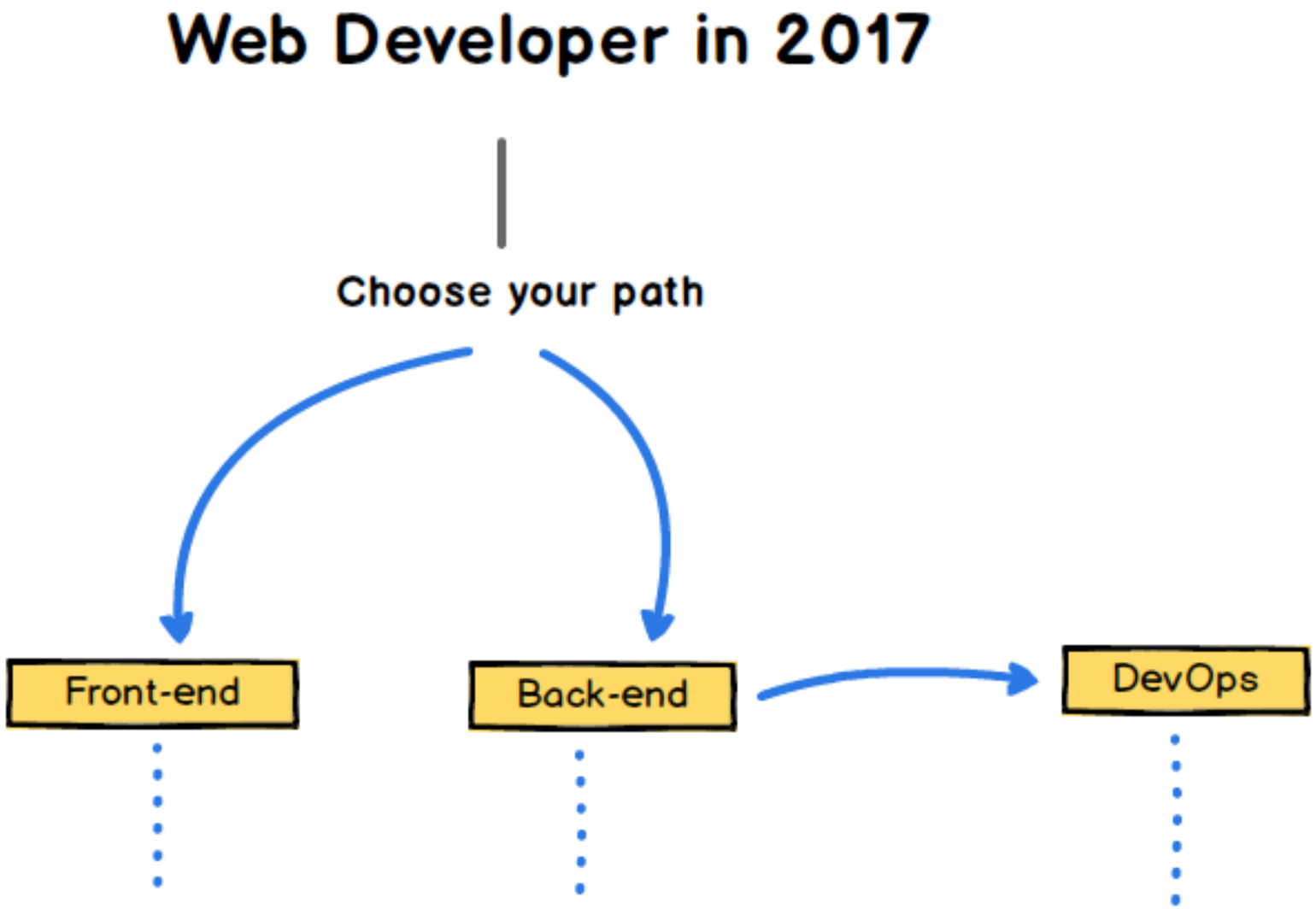
aonghus.lawlor@insight-centre.org

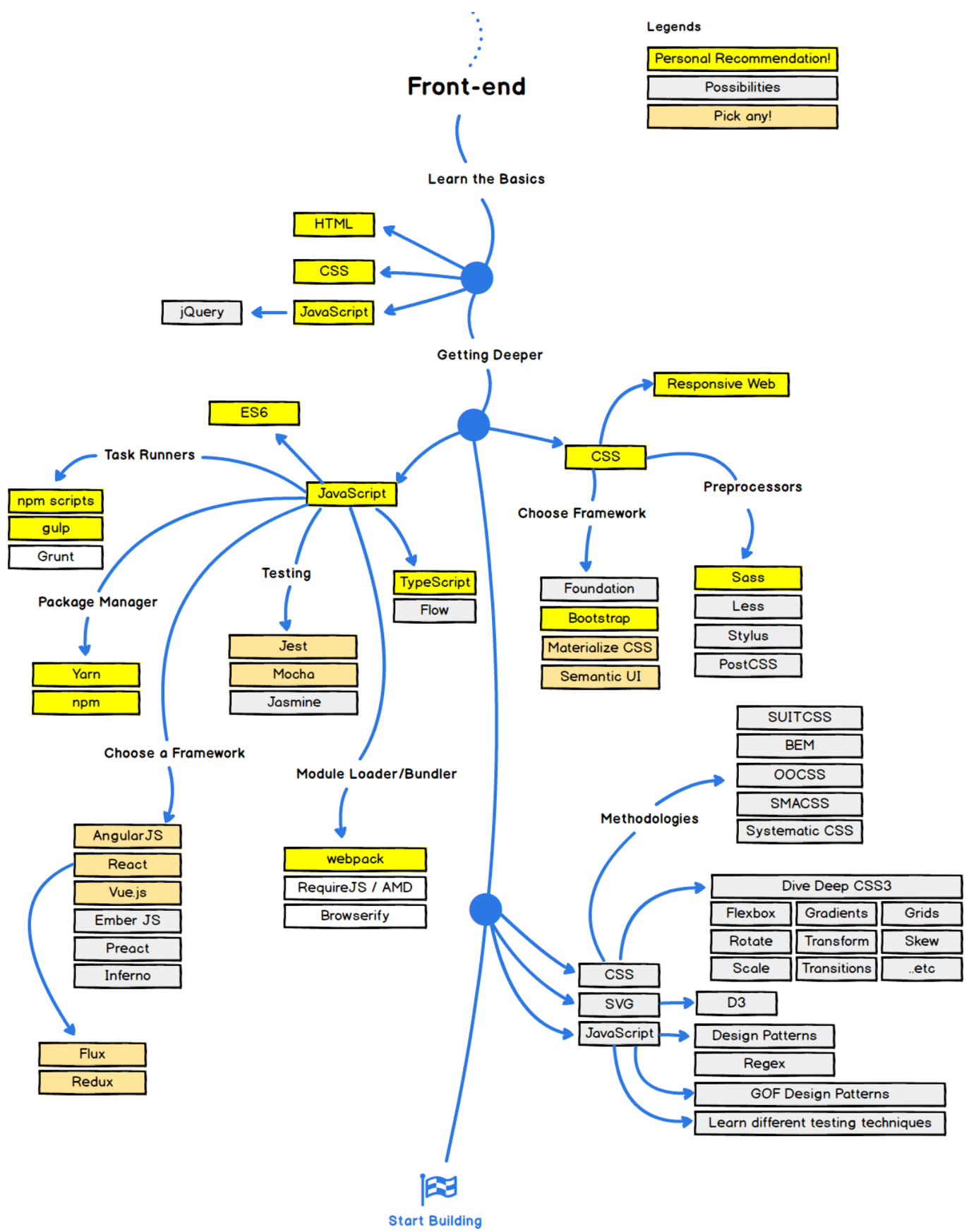
# Web Development 2017

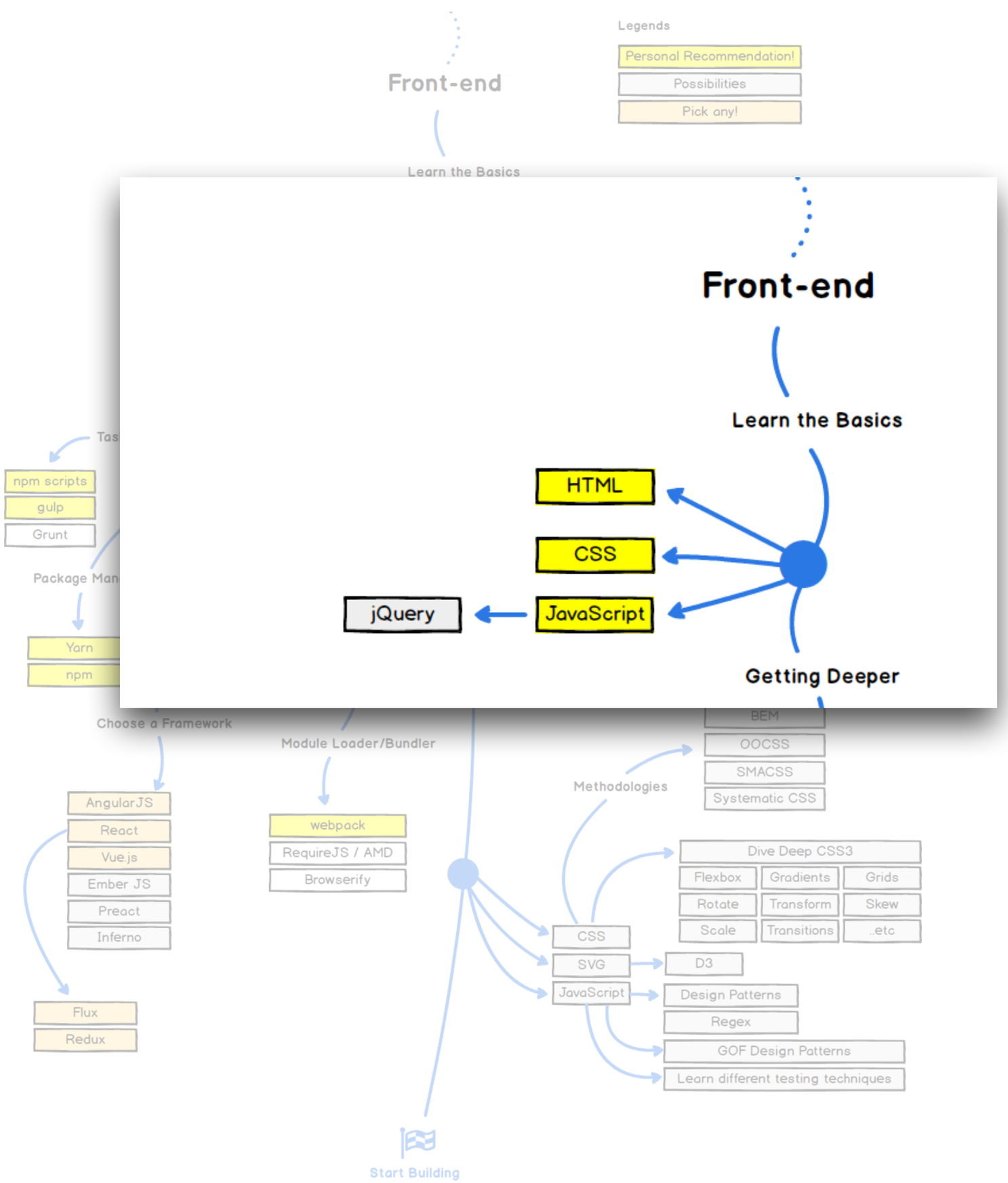
Required for any path

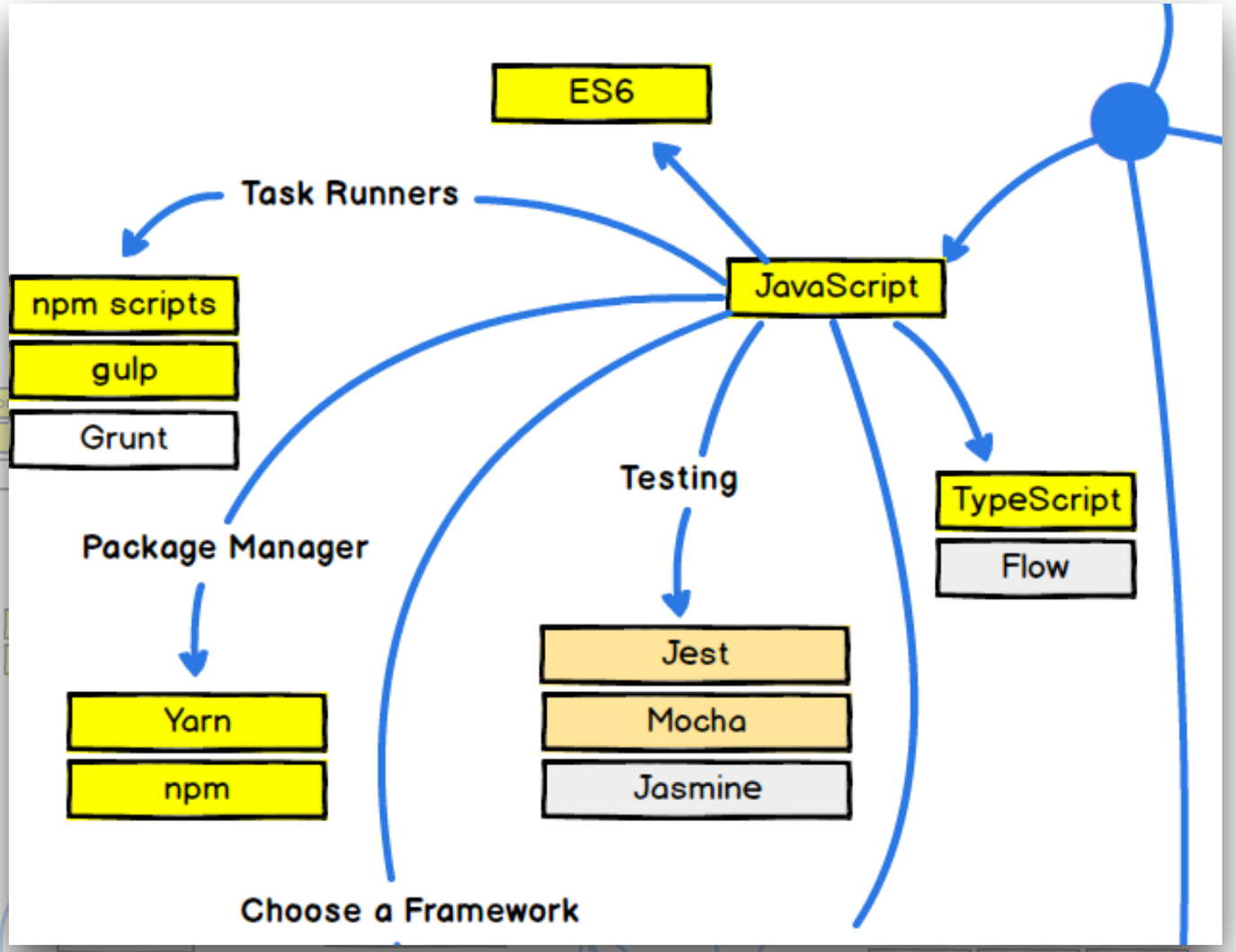
- Git - Version Control
- SSH
- HTTP/HTTPs and APIs
- Basic Terminal Usage
- Learn to Research
- Datastructures & Algorithms
- Character Encodings
- Github

Create your profile. Explore the relevant opensource projects. Make your habbit to look under the hood for the projects you like. Create and contribute to opensource projects.




Legends
Personal Recommendation!
Possibilities
Pick any!
Front-end
Learn the Basics
HTML
CSS
JavaScript
jQuery
Getting Deeper
ES6
Task Runners
npm scripts
gulp
Grunt
JavaScript
Package Manager
Yarn
npm
Testing
Jest
Mocha
Jasmine
TypeScript
Flow
Choose a Framework
AngularJS
React
Vue.js
Ember JS
Preact
Inferno
Flux
Redux
Module Loader/Bundler
webpack
RequireJS / AMD
Browserify
Responsive Web
CSS
Preprocessors
Choose Framework
Foundation
Bootstrap
Materialize CSS
Semantic UI
Sass
Less
Stylus
PostCSS
SUITCSS
BEM
OOCSS
SMACSS
Systematic CSS
Methodologies
Dive Deep CSS3
Flexbox
Gradients
Grids
Rotate
Transform
Skew
Scale
Transitions
..etc
CSS
SVG
D3
JavaScript
Design Patterns
Regex
GOF Design Patterns
Learn different testing techniques
Start Building


Front-end
Learn the Basics
HTML
CSS
JavaScript
jQuery
Getting Deeper
Legends
Personal Recommendation!
Possibilities
Pick any!
Front-end
Learn the Basics
Tas
npm scripts
gulp
Grunt
Package Man
Yarn
npm
Choose a Framework
AngularJS
React
Vue.js
Ember JS
Preact
Inferno
Flux
Redux
Module Loader/Bundler
webpack
RequireJS / AMD
Browserify
BEM
OOCSS
SMACSS
Systematic CSS
Methodologies
Dive Deep CSS3
Flexbox
Gradients
Grids
Rotate
Transform
Skew
Scale
Transitions
..etc
CSS
SVG
D3
JavaScript
Design Patterns
Regex
GOF Design Patterns
Learn different testing techniques
Start Building


ES6
Task Runners
npm scripts
gulp
Grunt
JavaScript
TypeScript
Flow
Testing
Package Manager
Jest
Yarn
npm
Mocha
Jasmine
Choose a Framework

Front-end

Legends

Personal Recommendation!

Possibilities

Pick any!

Learn the Basics

tting Deeper

Responsive Web

CSS

Preprocessors

Choose Framework

pt

Foundation

Bootstrap

Materialize CSS

Semantic UI

Sass

Less

Stylus

PostCSS

SUITCSS

Learn different testing techniques

Start Building

Legends

Personal Recommendation!

Possibilities

Pick any!

Front-end

Learn the Basics

er

Methodologies

SUITCSS

BEM

OOCSS

SMACSS

Systematic CSS

Dive Deep CSS3

Flexbox

Gradients

Grids

Rotate

Transform

Skew

Scale

Transitions

..etc

CSS

SVG

D3

JavaScript

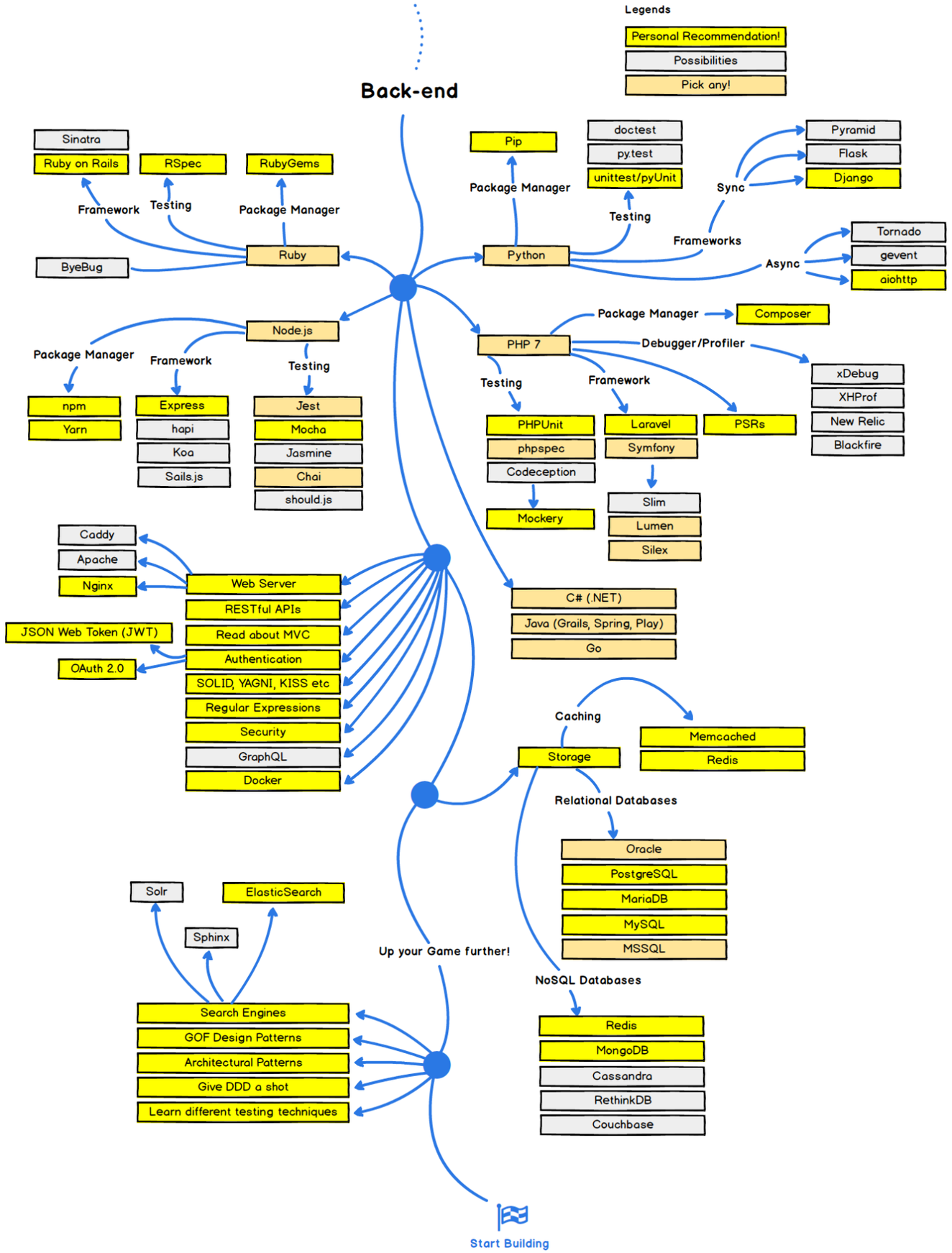
Design Patterns

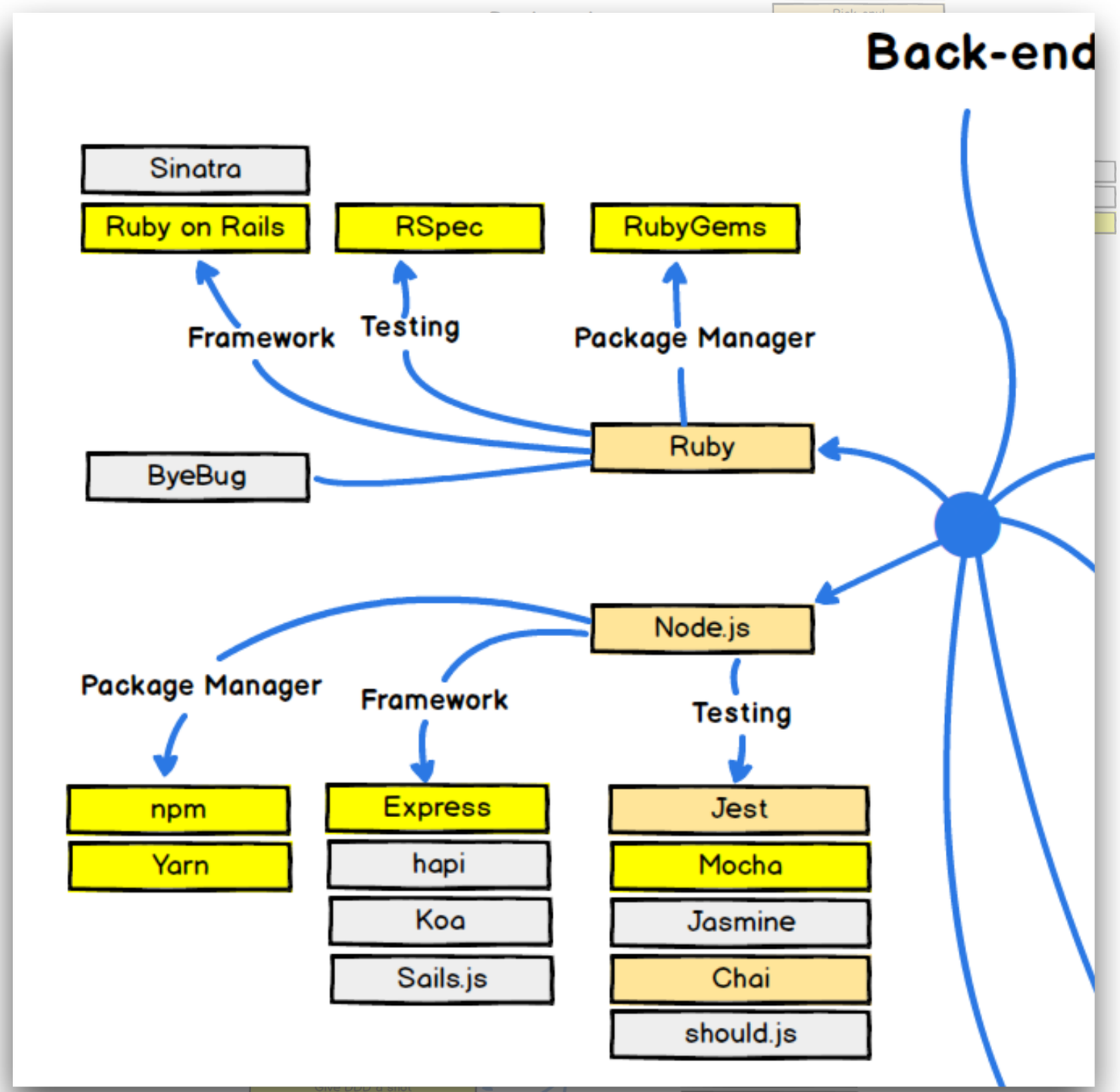
Regex

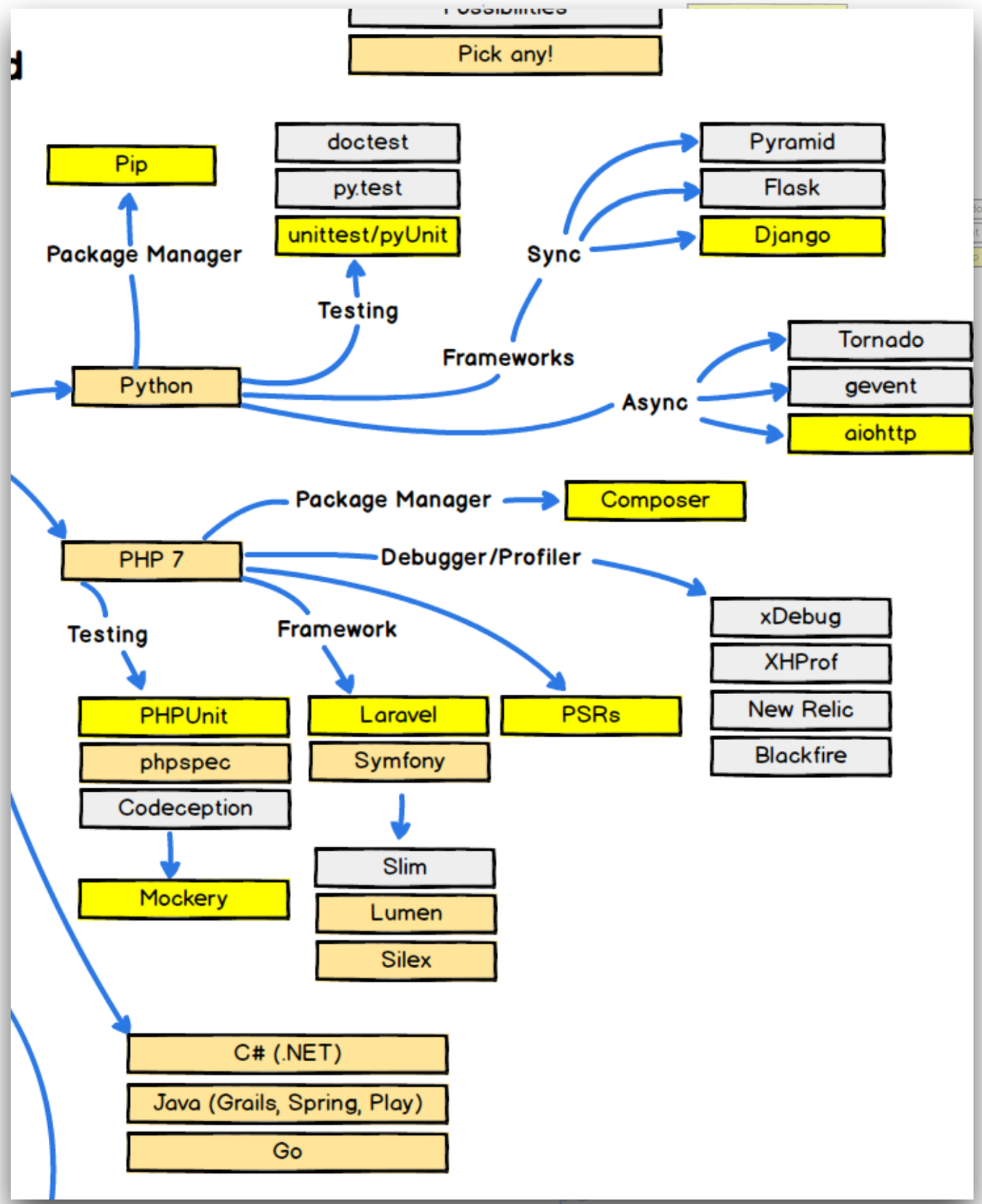
GOF Design Patterns

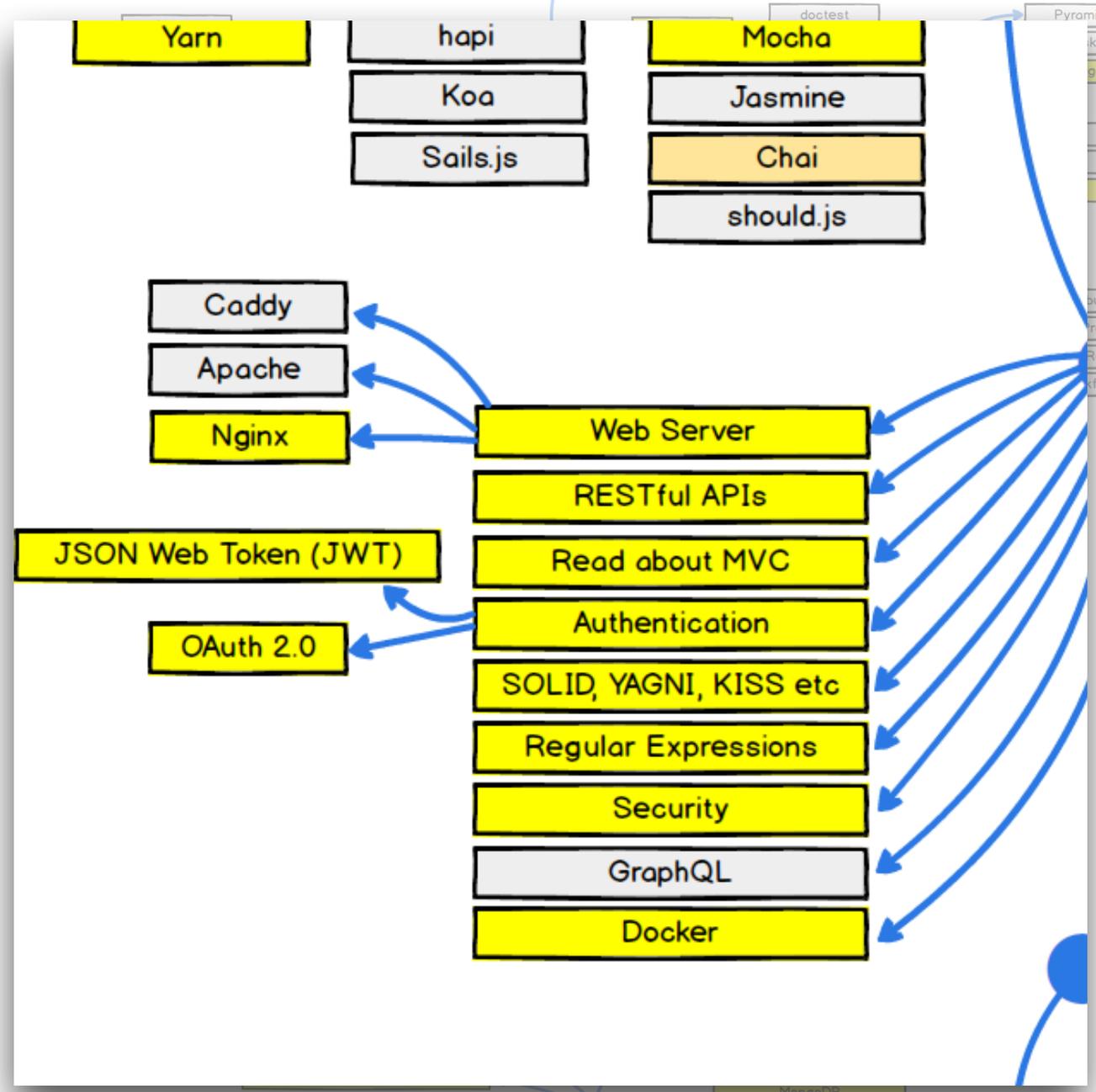
Learn different testing techniques

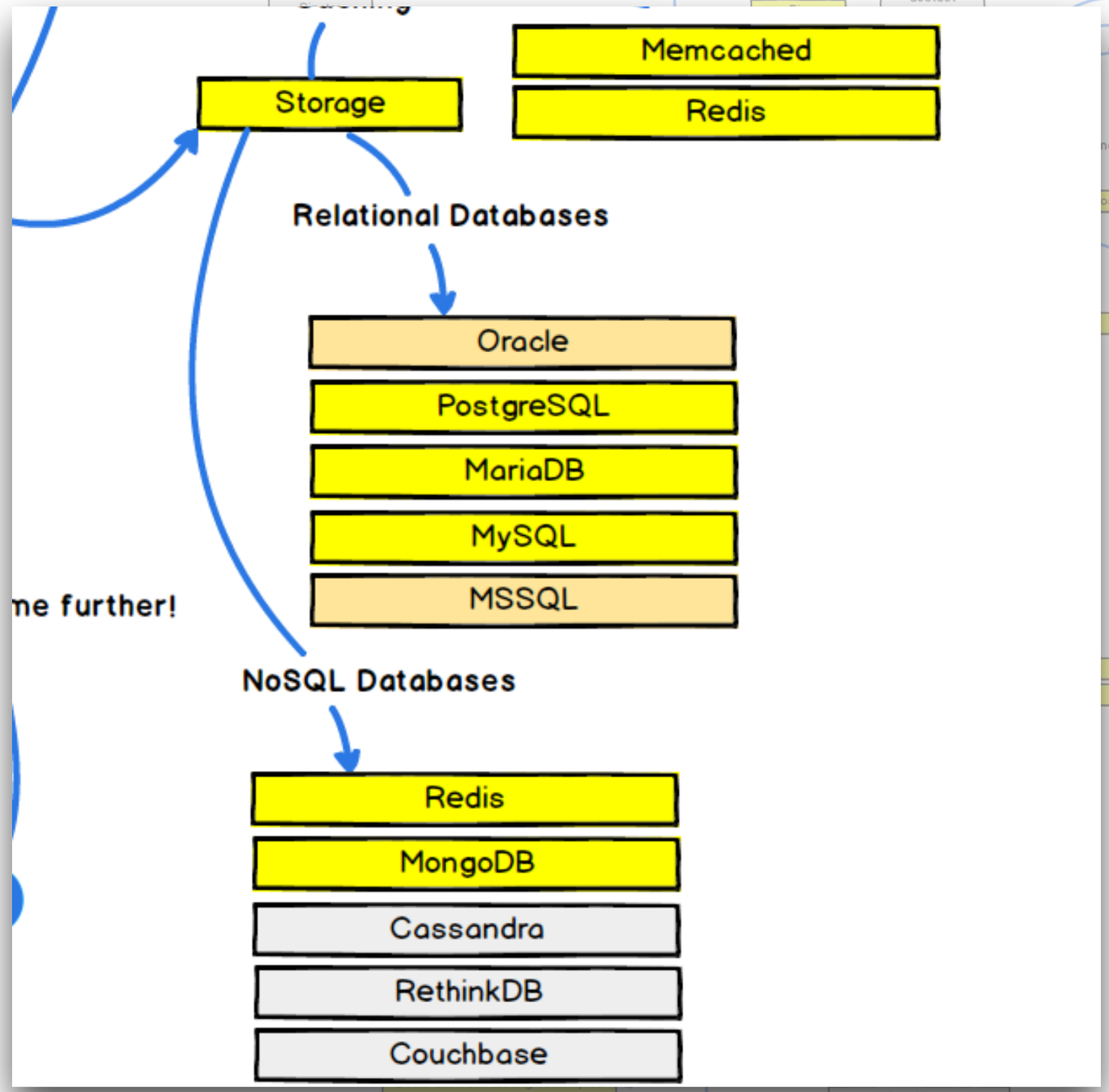
Start Building


Legends
Personal Recommendation!
Possibilities
Pick any!
Back-end
Sinatra
Ruby on Rails
RSpec
RubyGems
Framework
Testing
Package Manager
Ruby
ByeBug
Pip
Package Manager
doctest
pytest
unittest/pyUnit
Testing
Python
Sync
Pyramid
Flask
Django
Frameworks
Async
Tornado
gevent
aiohttp
Node.js
Package Manager
Framework
Testing
npm
Yarn
Express
hapi
Koa
Sails.js
Jest
Mocha
Jasmine
Chai
should.js
PHP 7
Package Manager
Composer
Debugger/Profiler
xDebug
XHProf
New Relic
Blackfire
Testing
Framework
PHPUnit
phpspec
Codeception
Mockery
Laravel
Symfony
PSRs
Slim
Lumen
Silex
Caddy
Apache
Nginx
Web Server
RESTful APIs
Read about MVC
Authentication
SOLID, YAGNI, KISS etc
Regular Expressions
Security
GraphQL
Docker
JSON Web Token (JWT)
OAuth 2.0
C# (.NET)
Java (Grails, Spring, Play)
Go
Caching
Memcached
Redis
Storage
Relational Databases
Oracle
PostgreSQL
MariaDB
MySQL
MSSQL
NoSQL Databases
Redis
MongoDB
Cassandra
RethinkDB
Couchbase
Solr
ElasticSearch
Sphinx
Search Engines
GOF Design Patterns
Architectural Patterns
Give DDD a shot
Learn different testing techniques
Up your Game further!
Start Building

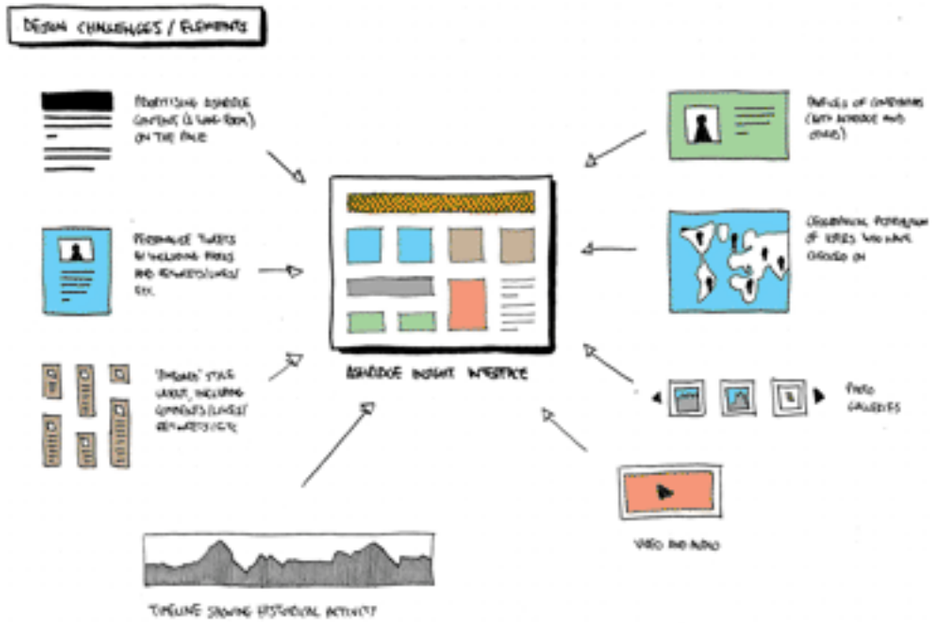

Back-end
Sinatra
Ruby on Rails
RSpec
RubyGems
Framework
Testing
Package Manager
Ruby
ByeBug
Node.js
Package Manager
Framework
Testing
npm
Yarn
Express
hapi
Koa
Sails.js
Jest
Mocha
Jasmine
Chai
should.js

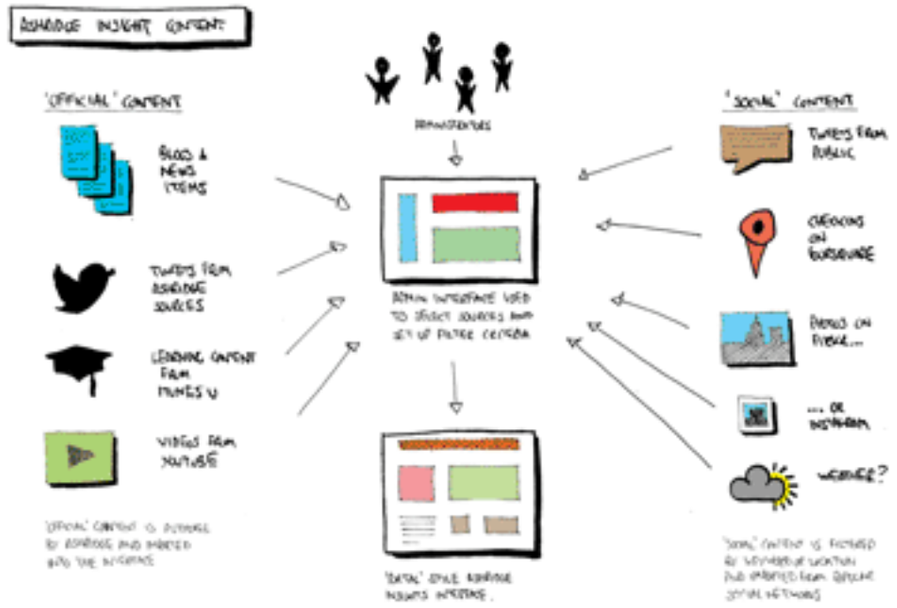

Pick any!
Pip
doctest
py.test
unittest/pyUnit
Pyramid
Flask
Django
Package Manager
Testing
Sync
Frameworks
Python
Tornado
gevent
aiohttp
Async
Package Manager
Composer
PHP 7
Debugger/Profiler
xDebug
XHProf
New Relic
Blackfire
Testing
Framework
PHPUnit
phpspec
Codeception
Mockery
Laravel
Symfony
Slim
Lumen
Silex
PSRs
C# (.NET)
Java (Grails, Spring, Play)
Go


Back-end
Legends
Personal Recommendation!
Possibilities
Pick any!
Yarn
hapi
Koa
Sails.js
Mocha
Jasmine
Chai
should.js
Caddy
Apache
Nginx
Web Server
RESTful APIs
JSON Web Token (JWT)
Read about MVC
Authentication
OAuth 2.0
SOLID, YAGNI, KISS etc
Regular Expressions
Security
GraphQL
Docker
Pyramid
Tornado
gevent
aiohttp
Architectural Patterns
Give DDD a shot
Learn different testing techniques
MongoDB
Cassandra
RethinkDB
Couchbase
Start Building


Storage
Memcached
Redis
Relational Databases
Oracle
PostgreSQL
MariaDB
MySQL
MSSQL
me further!
NoSQL Databases
Redis
MongoDB
Cassandra
RethinkDB
Couchbase

- Body Level One
  - Body Level Two
    - Body Level Three
      - Body Level Four
        - Body Level Five



# DBikes Project

# Flask

# Flask

- Flask is a web framework.
- Flask comes with tools, libraries and technologies that allow you to build a web app.


- This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.
- Flask is a micro-framework, with little to no dependencies to external libraries
- Used by Pinterest, LinkedIn, others

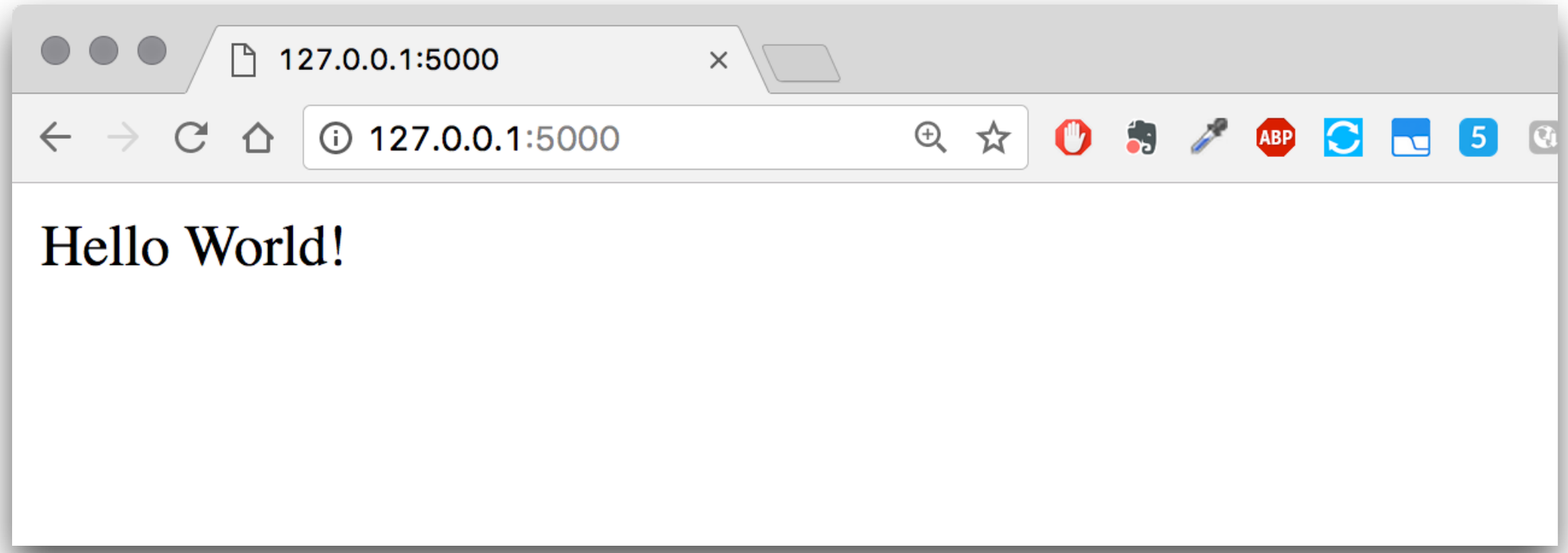
```python
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

Flask hello world app

```
$ python app.py
```

or

```
$ export FLASK_DEBUG=1
$ flask run
```



very basic flask app, which simply serves our main index.html

Access it like this:

$ python app.py

and then browse to:

http://127.0.0.1:5000/

```
#!/usr/bin/env python
from flask import Flask

# Create our flask app. Static files are served from
'static' directory
app = Flask(__name__, static_url_path='')


# this route simply serves 'static/index.html'
@app.route('/')
def root():
    return app.send_static_file('index.html')


if __name__ == "__main__":
    app.run(debug=True)
```

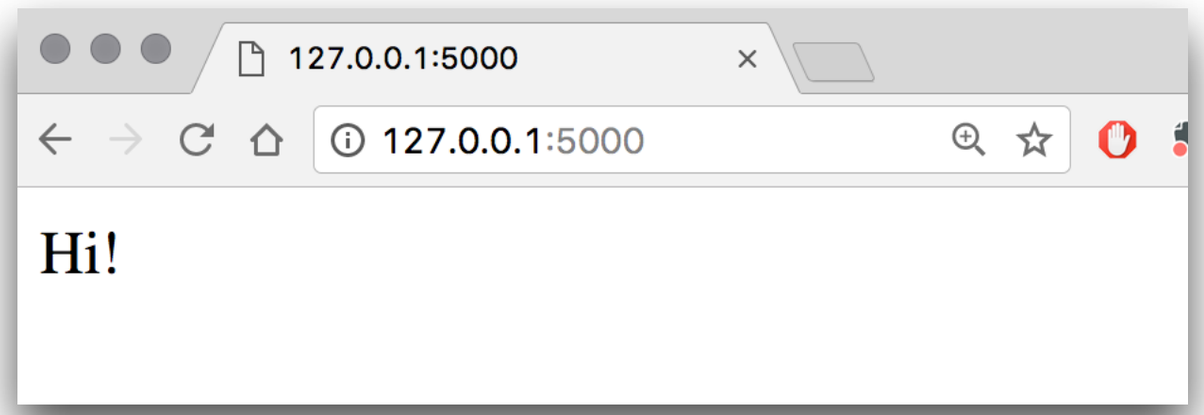
```
|--
/project_dir
    |--
    |-- /web
        |-- app.py
        |-- config.py
        |--
        |-- ...

    |-- /static
        |-- d3.min.js
        |-- ...

    |-- /templates
        |-- index.html
        |-- ...
```

flask file template

# Flask Routing

- in Flask the *route()* decorator is used to bind a function to a URL
- You can make certain parts of the URL dynamic and attach multiple rules to a function

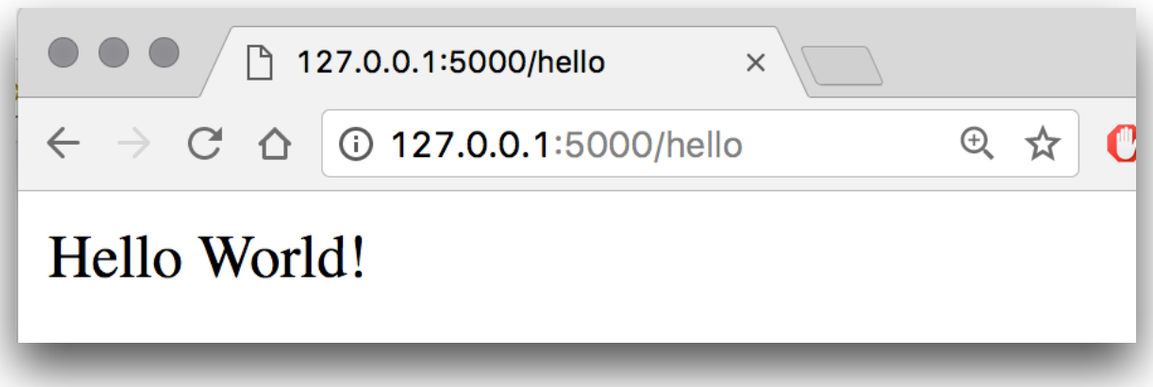
```
@app.route('/')
def index():
    return 'Index Page'
```



# Flask Routing

- in Flask the *route()* decorator is used to bind a function to a URL
- You can make certain parts of the URL dynamic and attach multiple rules to a function

```
@app.route('/hello')
def hello():
    return 'Hello World’
```

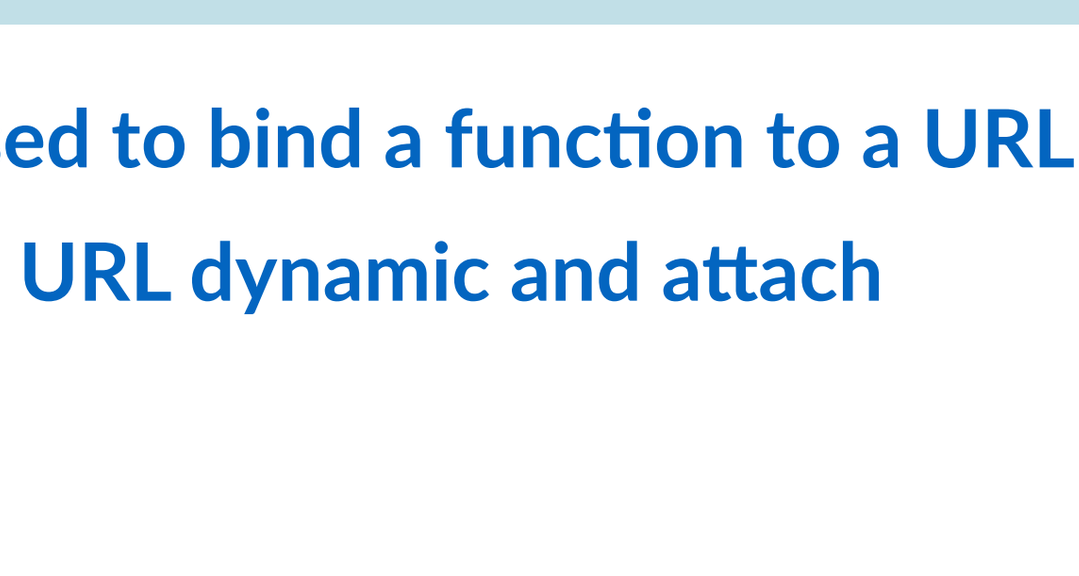


# Flask Routing

- in Flask the *route()* decorator is used to bind a function to a URL
- You can make certain parts of the URL dynamic and attach multiple rules to a function

```
@app.route('/user')
def user():
    return app.send_static_file('user.html')
```

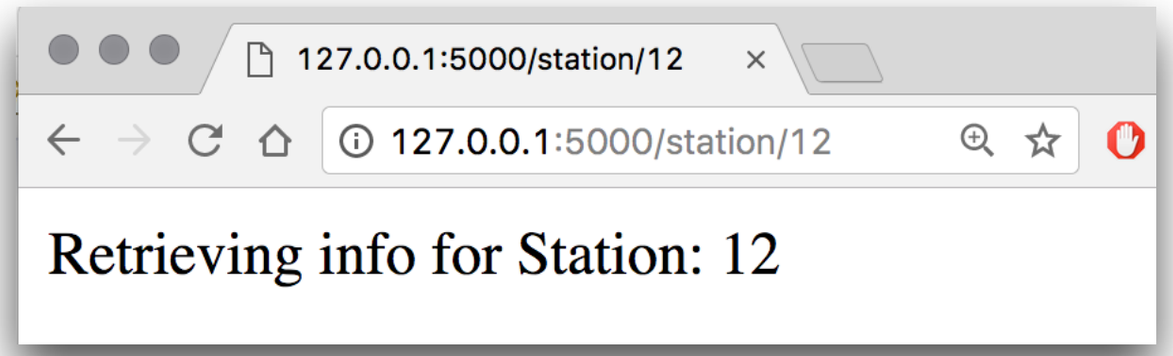
```
<!DOCTYPE html>
<html>
<body>
<h1>Hi!</h1>
<p>This is the User page</p>
</body>
</html>
```



# Flask Routing

- in Flask the *route()* decorator is used to bind a function to a URL
- You can make certain parts of the URL dynamic and attach multiple rules to a function

```
@app.route('/station/<int:station_id>')
def station(station_id):
    # show the station with the given id, the id is an integer
    return 'Retrieving info for Station: {}'.format(station_id)
```



# Flask Routing

- When the application receives a request from a client, it needs to figure out what function to invoke to service it.
- to do this, Flask looks up the URL given in the request in the application's **URL map**,
- this dictionary contains a mapping of URLs to the functions that handle them.
- Flask builds this map using the `app.route` decorators

# Flask Routing

- What is a decorator?
- a function which takes in a function (the one which you decorated with the "@" symbol) and returns a new function.
- When you decorate a function, you're telling Python to call the new function returned by your decorator, instead of just running the body of your function directly.

# Flask Routing

```
# This is our decorator
def simple_decorator(f):
    # This is the new function we're going to return
    # This function will be used in place of our original definition
    def wrapper():
        print "Entering Function"
        f()
        print "Exited Function"

    return wrapper
```

1

3

```
@simple_decorator
def hello():
    print "Hello World"
```

2

```
hello()

>>> Entering Function
Hello World
Exited Function
```

# Web App Architecture

Browser
HTML rendering

**Client**

*html*

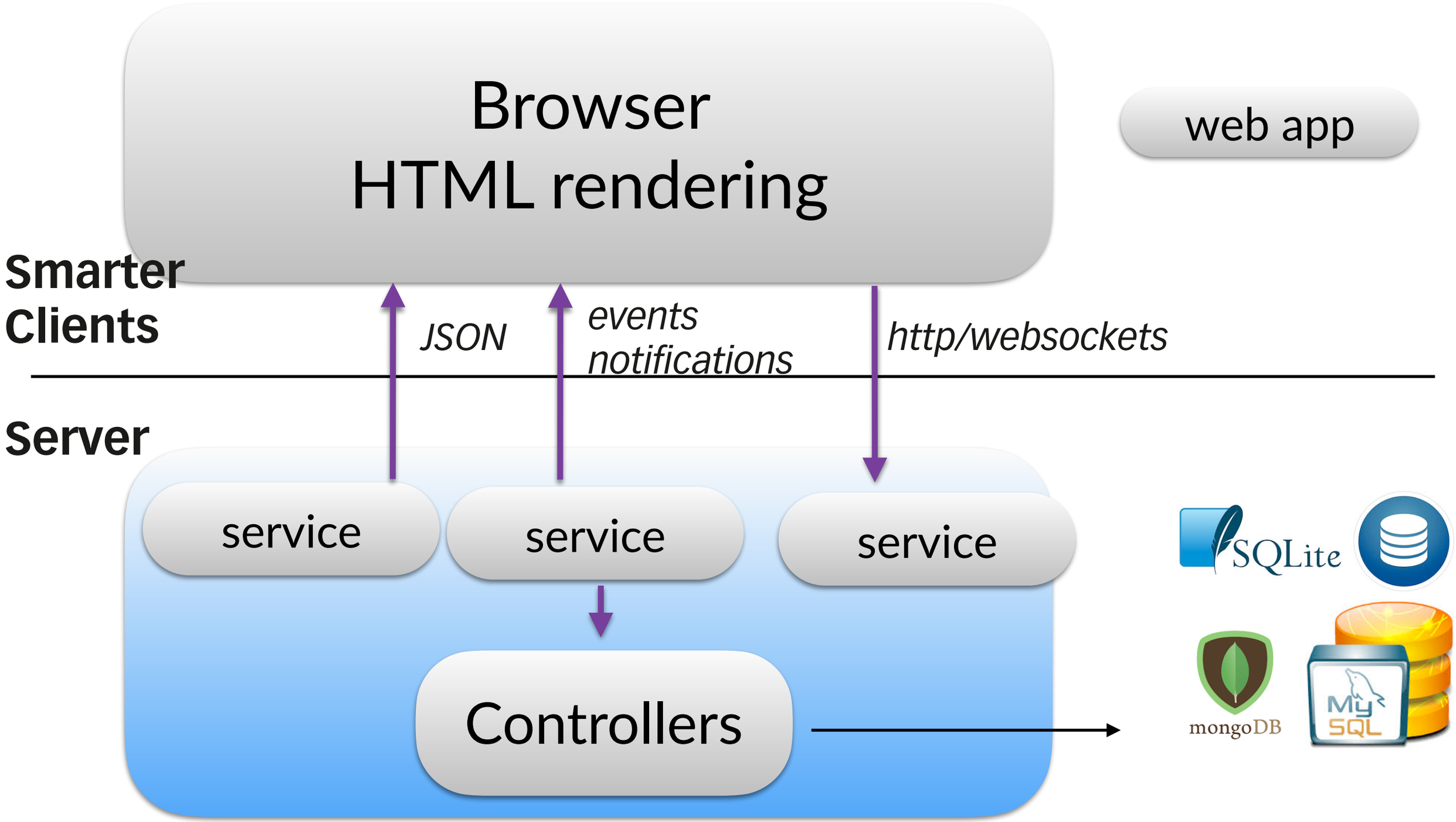
*http*

**Server**

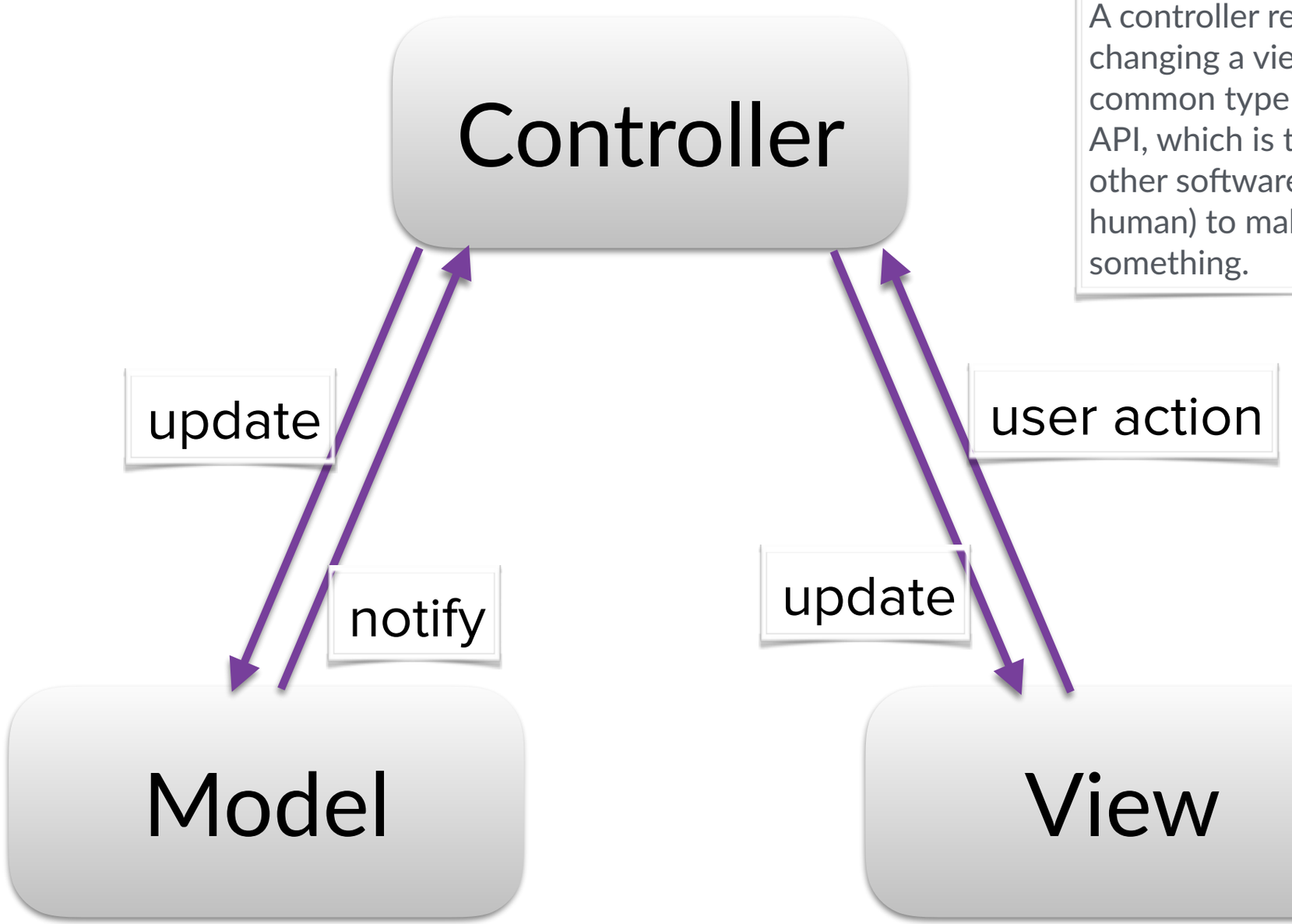
View generation

Controllers

# Web App Architecture



# Architecture

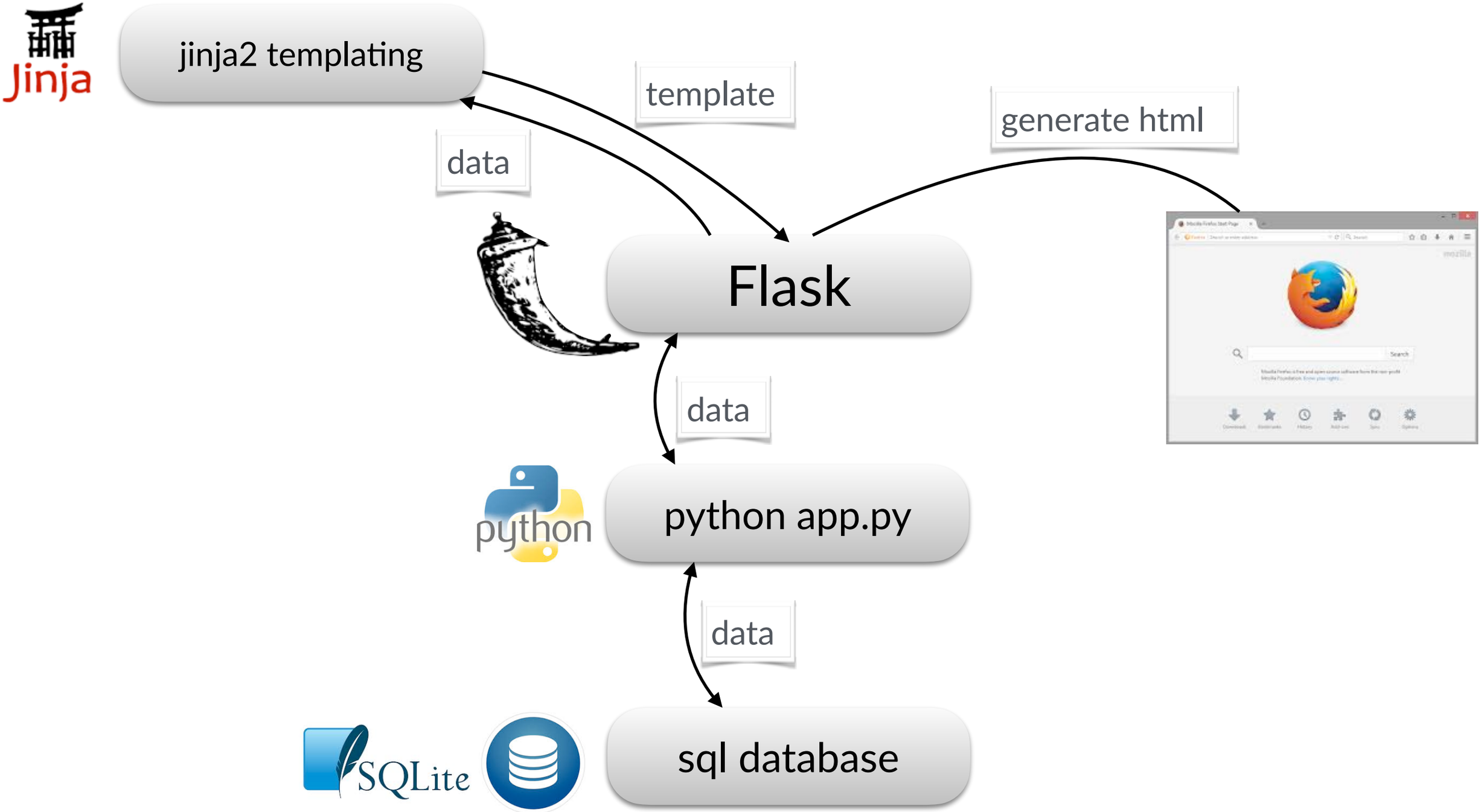



A controller responds to input by changing a view or model. A common type of controller is an API, which is typically used by other software (rather than a human) to make the application do something.

A model might be a very simple representation of a real thing, or very detailed. A model is like an ideal representation of the actual domain being modelled. Usually, models are stored in a database, and all of the model attributes can be mapped to rows and columns.

responsible for display of data in your applications

# Architecture



```python
#!/usr/bin/env python
from flask import Flask, g, render_template, jsonify
import json
import sqlite3

# Create our flask app. Static files are served from 'static' directory
app = Flask(__name__, static_url_path='')
app.config.from_object('config')


def connect_to_database():
    return sqlite3.connect(app.config["DATABASE"])


def get_db():
    db = getattr(g, '_database', None)
    if db is None:
        db = g._database = connect_to_database()
    return db


@app.teardown_appcontext
def close_connection(exception):
    db = getattr(g, '_database', None)
    if db is not None:
        db.close()


# this route simply serves 'static/index.html'
@app.route('/')
def root():
    return render_template('index.html',
MAPS_APIKEY=app.config["MAPS_APIKEY"])


@app.route("/stations")
def get_stations():
    conn = get_db()
    # https://docs.python.org/2/library/sqlite3.html#sqlite3.Row
    conn.row_factory = sqlite3.Row
    cur = conn.cursor()
    stations = []
    rows = cur.execute("SELECT * from stations;")
    for row in rows:
        stations.append(dict(row))

    return jsonify(stations=stations)

if __name__ == "__main__":
    app.run(debug=True)
```

```python
@app.route("/stations")
def get_stations():
    conn = get_db()
    # https://docs.python.org/2/library/sqlite3.html#sqlite3.Row
    conn.row_factory = sqlite3.Row
    cur = conn.cursor()
    stations = []
    rows = cur.execute("SELECT * from stations;")
    for row in rows:
        stations.append(dict(row))

    return jsonify(stations=stations)
```

The jsonify() function in flask returns flask.Response() object that already has the appropriate content-type header 'application/json' for use with json responses, whereas the json.dumps() will just return an encoded string, which would require manually adding the mime type header.

```python
from flask import Flask, g, jsonify
app = Flask(__name__)

def connect_to_database():
    return engine = create_engine("mysql+mysqldb://{}:{}@{}:{}/{}".format(config.USER,
                                                                          config.PASSWORD,
                                                                          config.URI,
                                                                          config.PORT,
                                                                          config.DB), echo=True)



def get_db():
    db = getattr(g, '_database', None)
    if db is None:
        db = g._database = connect_to_database()
    return db
```

```python
@app.teardown_appcontext
def close_connection(exception):
    db = getattr(g, '_database', None)
    if db is not None:
        db.close()

@app.route("/available/<int:station_id>")
def get_stations():
    engine = get_db()
    data = []
    rows = engine.execute("SELECT available_bikes from stations where number = {};".format(station_id))
    for row in rows:
        data.append(dict(row))

    return jsonify(available=data)
```

```python
@app.route("/available/<int:station_id>")
def get_stations(station_iod):
    engine = get_db()
    data = []
    rows = engine.execute("SELECT available_bikes from stations
where number = {};".format(station_id))
    for row in rows:
        data.append(dict(row))
    return jsonify(available=data)
```

# Templates

# Templates

- template are mostly standard HTML
- the only difference is that there are some placeholders for the dynamic content enclosed in {{ ... }} sections
- templates allow you to keep the logic of your app separate from its layout and style

```
from jinja2 import Template
template = Template('Hello {{ name }}!')

print template.render(name='John Doe')
>>> Hello John Doe!
```

# Template rendering

```
tpl = """
<html>
  <head>
    <title>{{ title }} - microblog</title>
  </head>
  <body>
      <h1>Hello, {{ user.nickname }}!</h1>
  </body>
</html>
"""

user = {'nickname': 'Aonghus'}  # fake user
title = "Welcome to my Blog!"
print Template(tpl).render(user=user, title=title)
```

```
<html>
  <head>
    <title>Welcome to my Blog! - microblog</title>
  </head>
  <body>
      <h1>Hello, Aonghus!</h1>
  </body>
</html>
```

# Template control flow

templates support control statements, given inside {%...%} blocks

```
<html>
  <head>
    {% if title %}
    <title>{{ title }} - microblog</title>
    {% else %}
    <title>Welcome to microblog</title>
    {% endif %}
  </head>
  <body>
      <h1>Hello, {{ user.nickname }}!</h1>
  </body>
</html>
```

```
if
    ...
else
    ...
endif
```

# Template loops

Templates support simple list iteration. It is up to the view function to decide how many elements are presented.
The template cannot make any assumptions about the number of elements, so it needs to be prepared to render as many as the view sends.

```
{% for elem in elements %}
        ...
{% endfor %}
```

```
<html>
  <head>
    {% if title %}
    <title>{{ title }} - microblog</title>
    {% else %}
    <title>Welcome to microblog</title>
    {% endif %}
  </head>
  <body>
    <h1>Hi, {{ user.nickname }}!</h1>
    {% for post in posts %}
    <div><p>{{ post.author.nickname }} says: <b>{{ post.body }}</b></p></div>
    {% endfor %}
  </body>
</html>
```

# Template Inheritance

base.html

```
<html>
  <head>
    {% if title %}
    <title>{{ title }} - microblog</title>
    {% else %}
    <title>Welcome to microblog</title>
    {% endif %}
  </head>
  <body>
    <div>Microblog: <a href="/index">Home</a></div>
    <hr>
    {% block content %}{% endblock %}
  </body>
</html>
```

index.html

```
{% extends "base.html" %}
{% block content %}
    <h1>Hi, {{ user.nickname }}!</h1>
    {% for post in posts %}
    <div><p>{{ post.author.nickname }} says: <b>{{ post.body }}</b></p></div>
    {% endfor %}
{% endblock %}
```

- template inheritance allows us to move the parts of the page layout that are common to all templates and put them in a base template from which other templates can be derived.
- the **block** control statement defines the place where the derived templates can insert themselves. Blocks are given a unique name, and their content can be replaced or enhanced in derived templates.

# Linking Static Files

```
url_for('static', filename='path/to/file')
```

## file.html

```
<link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='bootstrap/bootstrap.min.css') }}">

<link rel="stylesheet" type="text/css" href="static/bootstrap/
bootstrap.min.css">
```

```
function showStationMarkers(data) {
            $.getJSON("http://127.0.0.1:5000/stations", null, function(data) {
                if ('stations' in data) {
                    var stations = data.stations;
                    console.log('stations', stations);
                    _.forEach(stations, function(station) {
                        // console.log(station.name, station.number);
                        var marker = new google.maps.Marker({
                            position : {
                                lat : station.position_lat,
                                lng : station.position_lng
                            },
                            map : map,
                            title : station.name,
                            station_number : station.number
                        });
                        marker.addListener("click", function() {
                            //drawStationCharts(this);
                            drawStationChartsWeekly(this);
                        });
                    })
                }
            });
        }
```

# Architecture

$.getJSON(“http://127.0.0.1/stations”)

Flask

```
@app.route(“/stations”)
    def get_station():
        ...
        return jsonify(data=...)
```

data

sql database

```
[{“name”: “Grand Canal Dock”,
“number”: 42,
“position_lat”: ...},
{...}
]
```

# Git notes

# How to protect your google API key?

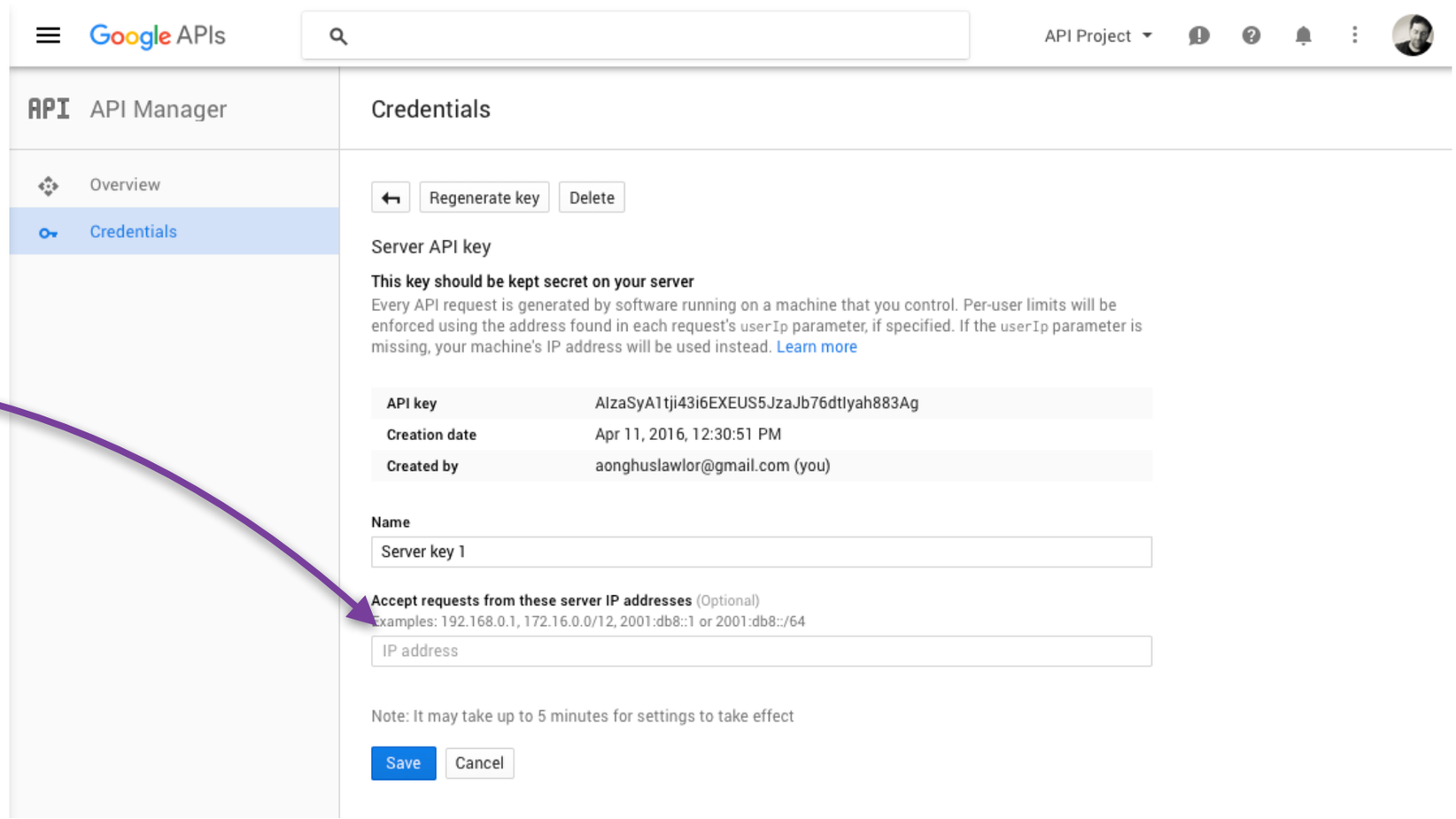
```
        };
      }
    </script>
    <script async defer
    src="https://maps.googleapis.com/maps/api/js?key=AIzaSyA1tji43i6EXEUS5JzaJb76dtIyah883Ag&callback=initMap">
    </script>
  </body>
</html>
```

In the HTML source the API key is visible to anyone who loads the page. Because the key is sent as a URL parameter, there is no simple way to hide this from everyone.

A possible solution is to restrict the referrer URLs which are allowed to use this key.

Google API Console

add the set of IP addresses which you use in here.




But unless you notice someone else is using your API key, it shouldn't be necessary to do this (since the API keys are free for small scale use).

# gitignore

- *.gitignore* tells git which files (or patterns) it should ignore
- files already tracked by Git are not affected
- usually used to avoid committing transient files from your working directory that aren't useful to other collaborators, such as images, db files, compilation products, temporary files IDEs create, etc.

GitIgnore Docs

Lots of useful gitignore templates

**sample .gitignore for python**

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
env/
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
*.egg-info/
.installed.cfg
*.egg

# Django stuff:
*.log
local_settings.py

# Sphinx documentation
docs/_build/

# IPython Notebook
.ipynb_checkpoints

# virtualenv
.python-version
venv/
ENV/
```

# gitignore

- **Each line in a gitignore file specifies a pattern:**
  - foo/ will match a directory foo and paths underneath it, but will not match a regular file or a symbolic link foo
  - If the pattern does not contain a slash /, Git treats it as a shell glob pattern and checks for a match against the pathname relative to the location of the .gitignore file
  - wildcards in the pattern will not match a / in the pathname. For example, "Documentation/*.html" matches "Documentation/git.html" but not "Documentation/ppc/ppc.html" or "tools/perf/Documentation/perf.html".
  - A leading slash matches the beginning of the pathname. For example, "/*.c" matches "cat-file.c" but not "mozilla-sha1/sha1.c".

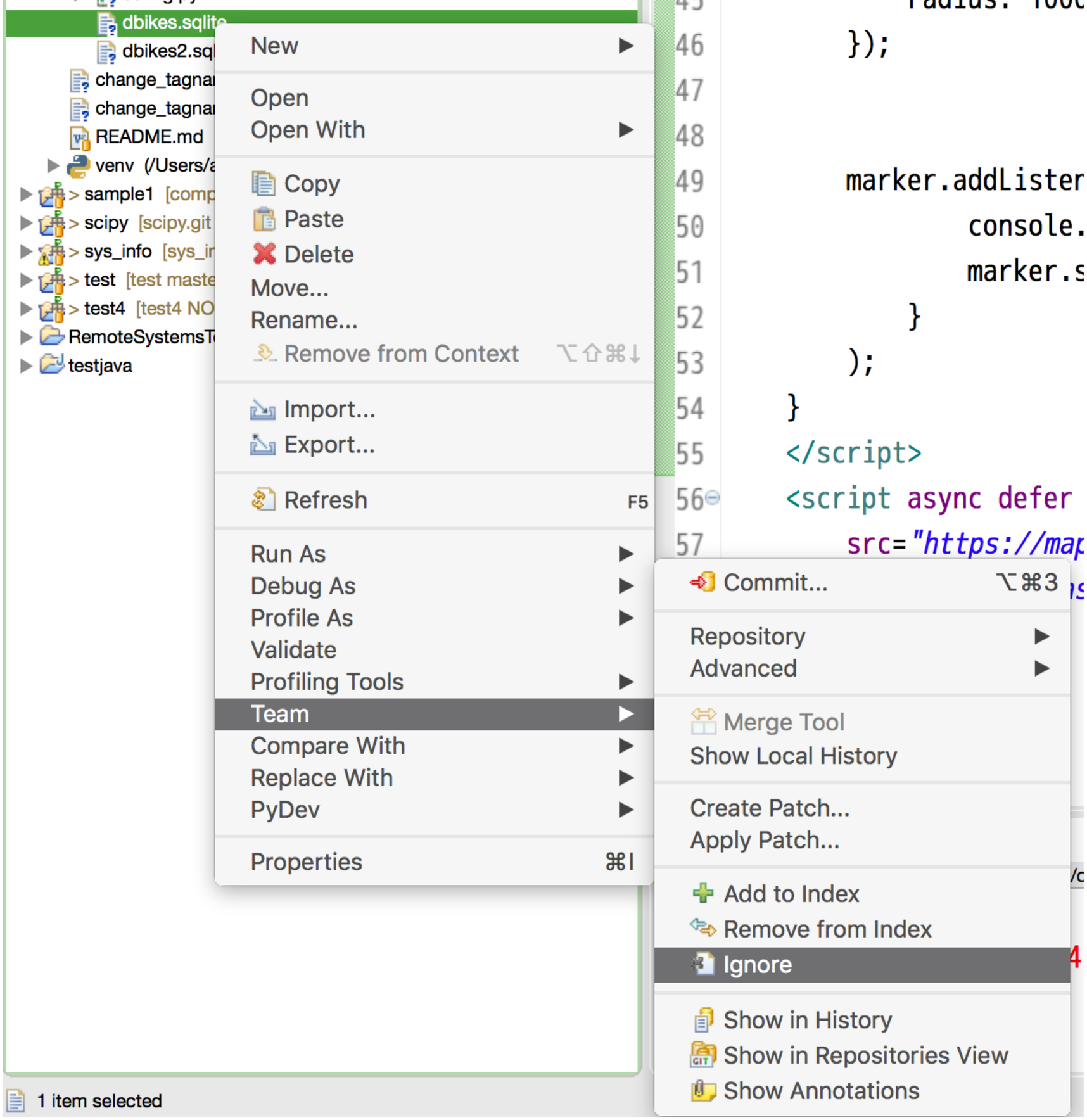
GitIgnore Docs

Lots of useful gitignore templates

**sample .gitignore for python**

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
env/
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
*.egg-info/
.installed.cfg
*.egg

# Django stuff:
*.log
local_settings.py

# Sphinx documentation
docs/_build/

# IPython Notebook
.ipynb_checkpoints

# virtualenv
.python-version
venv/
ENV/
```

Eclipse:
File -> Team -> Ignore



# .gitattributes

- you can use *attributes* to specify separate merge strategies for individual files or directories in your project
- tell git how to diff non-text files
- have Git filter content before you check it into or out of Git
- tell Git which files are binary and tell it how to handle those files (eg. pdf or images or sql db files)

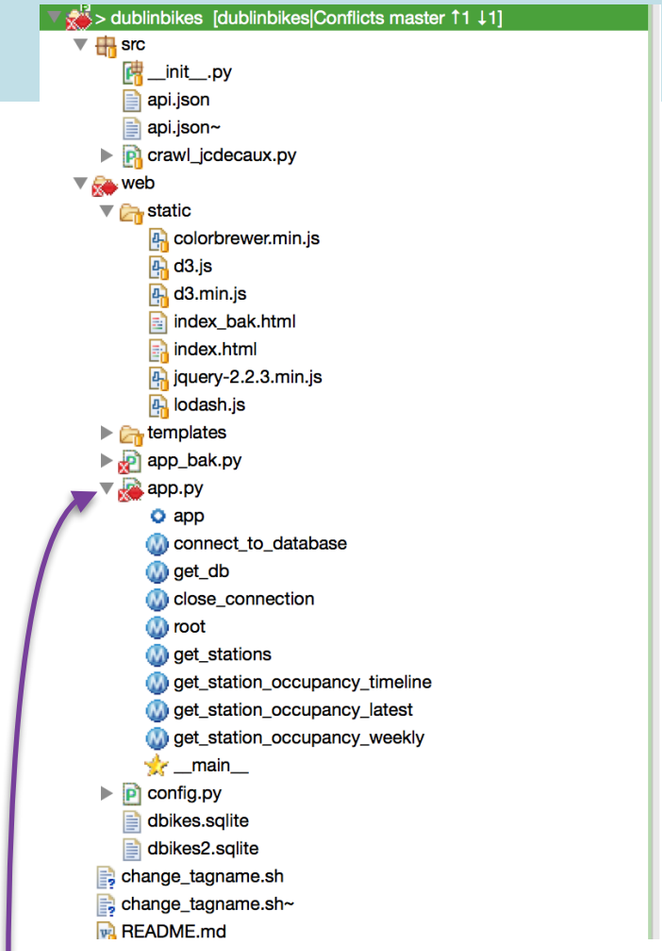
**sample ..gitattributes**

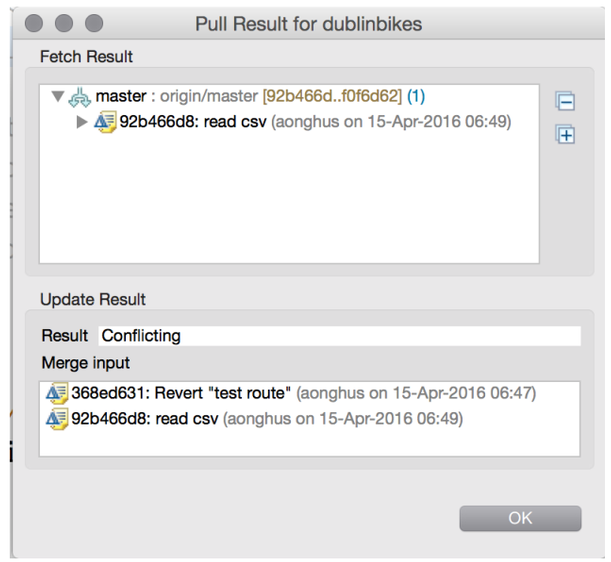
```
#
## These files are binary and should be left untouched
#
# (binary is a macro for -text -diff)
*.pdf binary
*.png binary
*.jpg binary
*.jpeg binary
*.gif binary
```

Git attributes templates

# Merge Conflicts

- conflicts can happen when two branches have changed the same part of the same file, and then those branches are merged together.
- For example, if you make a change on a particular line in a file, and your colleague working in a repository makes a change on the exact same line, a merge conflict occurs.
- Git has trouble understanding which change should be used, so it asks you to help out.
- When this sort of conflict occurs, Git writes a special block into the file that contains the contents of both versions where the conflict occurred.

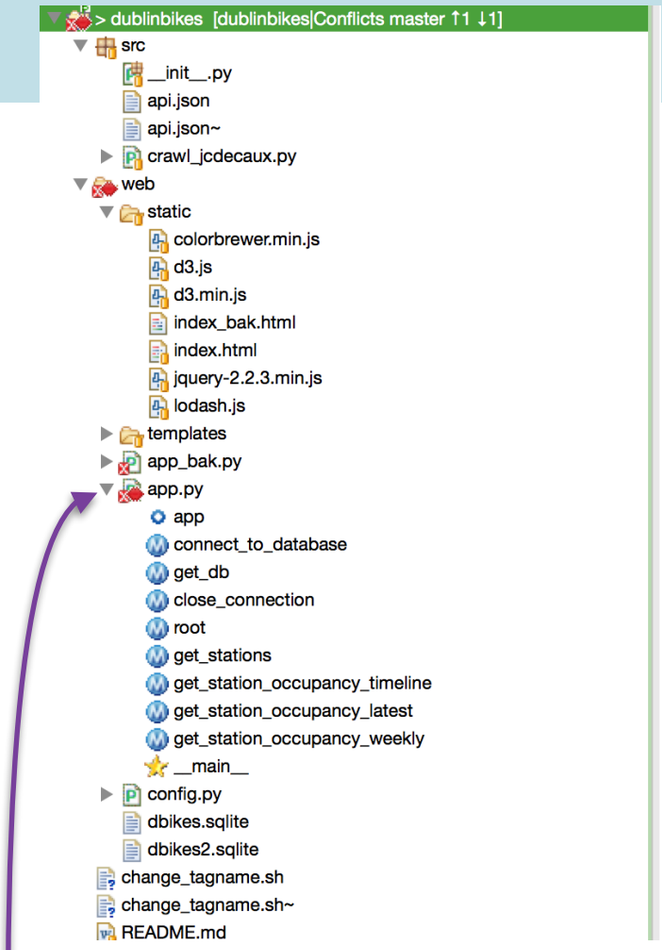


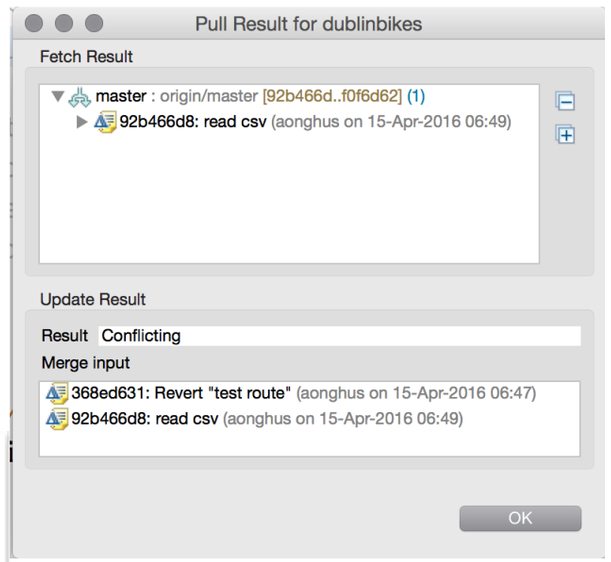




eclipse indicates which file has the conflict

# Merge Conflicts

```
<<<<<<< HEAD
# @app.route("/station_occupancy_t/<int:station_id>")
# def get_station_occupancy(station_id):
#     conn = get_db()
#     df = pd.read_sql_query("select * from dbikes where number = :number", conn,
params={"number": station_id})
#     df['last_update_date'] = pd.to_datetime(df.last_update, unit='ms')
#     df.set_index('last_update_date', inplace=True)
#     res = df['available_bike_stands'].resample('1d').mean()
#     print station_id, len(res)
#     return res.to_json()
=======
@app.route("/station_occupancy_test/<int:station_id>")
def get_station_occupancy_test(station_id):
    conn = get_db()
#    df = pd.read_sql_query("select * from dbikes where number = :number", conn,
params={"number": station_id})
    df = pd.read_csv()
    df['last_update_date'] = pd.to_datetime(df.last_update, unit='ms')
    df.set_index('last_update_date', inplace=True)
    res = df['available_bike_stands'].resample('1d').mean()
    print station_id, len(res)
    return res.to_json()
>>>>>>> branch 'master' of https://git.ucd.ie/aonghuslawlor/comp30670_dublinbikes.git
```

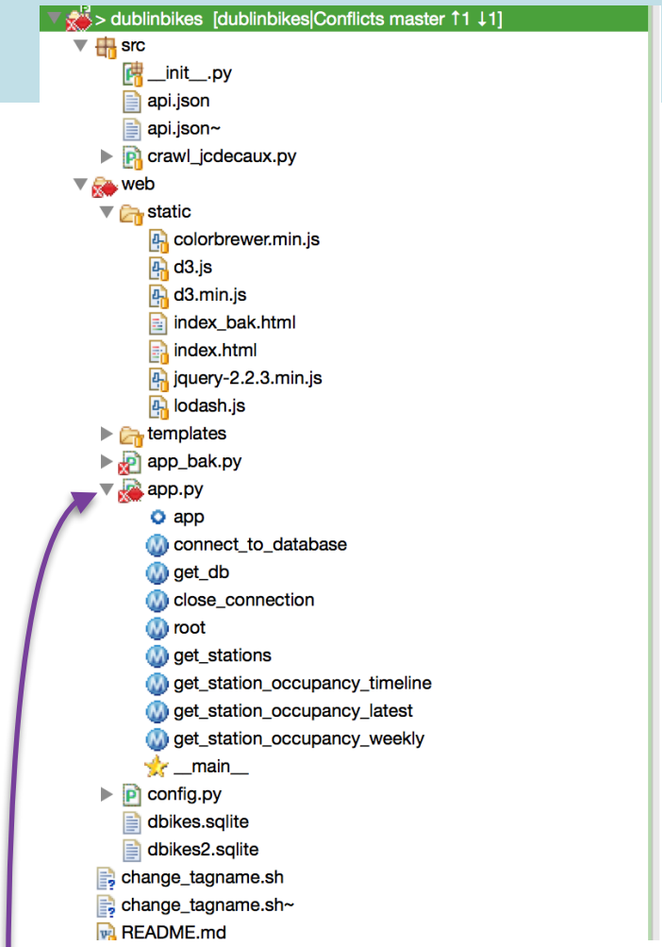


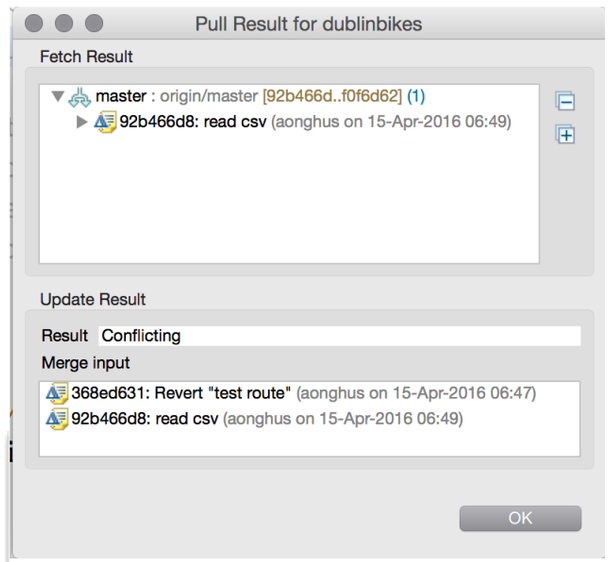




eclipse indicates which file has the conflict

# Merge Conflicts

```
<<<<<<< HEAD
# @app.route("/station_occupancy_t/<int:station_id>")
# def get_station_occupancy(station_id):
#     conn = get_db()
#     df = pd.read_sql_query("select * from dbikes where number = :number", conn,
params={"number": station_id})
#     df['last_update_date'] = pd.to_datetime(df.last_update, unit='ms')
#     df.set_index('last_update_date', inplace=True)
#     res = df['available_bike_stands'].resample('1d').mean()
#     print station_id, len(res)
#     return res.to_json()
=======
@app.route("/station_occupancy_test/<int:station_id>")
def get_station_occupancy_test(station_id):
    conn = get_db()
#    df = pd.read_sql_query("select * from dbikes where number = :number", conn,
params={"number": station_id})
    df = pd.read_csv()
    df['last_update_date'] = pd.to_datetime(df.last_update, unit='ms')
    df.set_index('last_update_date', inplace=True)
    res = df['available_bike_stands'].resample('1d').mean()
    print station_id, len(res)
    return res.to_json()
>>>>>>> branch 'master' of https://git.ucd.ie/aonghuslawlor/comp30670_dublinbikes.git
```

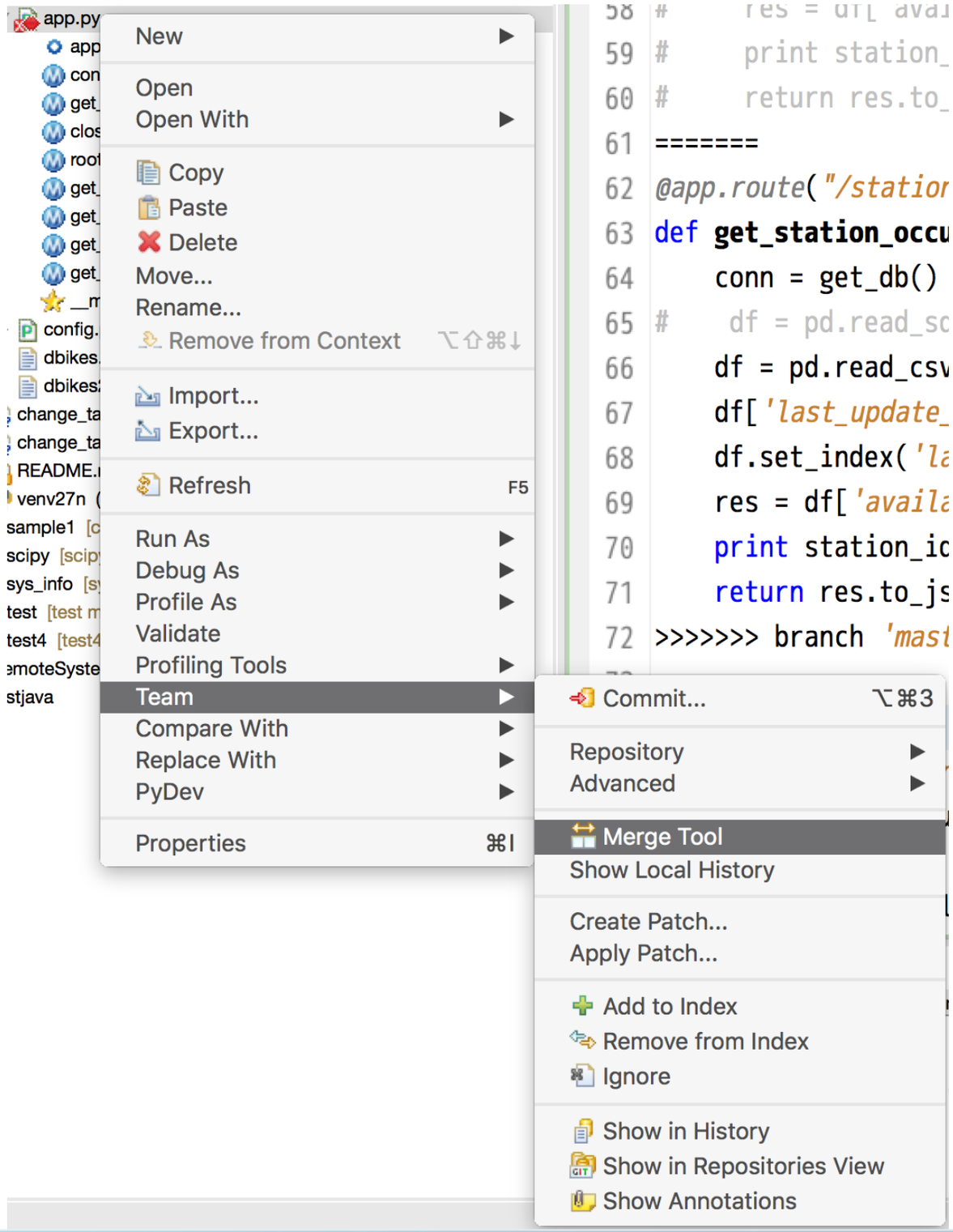



eclipse indicates which file has the conflict

# Merge Conflicts

Eclipse has a useful merge tool to help resolve conflicts



# Merge Conflicts

Structure Compare

web

app.py

Text Compare

Revert "test route" - 368ed63

```
48     return jsonify(stations=stations)
49
50
51 # @app.route("/station_occupancy_t/<int:station_id>")
52 # def get_station_occupancy(station_id):
53 #     conn = get_db()
54 #     df = pd.read_sql_query("select * from dbikes where number = :number", c
55 #     df['last_update_date'] = pd.to_datetime(df.last_update, unit='ms')
56 #     df.set_index('last_update_date', inplace=True)
57 #     res = df['available_bike_stands'].resample('1d').mean()
58 #     print station_id, len(res)
59 #     return res.to_json()
60
61
62 @app.route("/station_occupancy_timeline/<int:station_id>")
63 def get_station_occupancy_timeline(station_id):
64     conn = get_db()
65     df = pd.read_sql_query("select * from dbikes where number = :number", con
66     df['last_update_date'] = pd.to_datetime(df.last_update, unit='ms')
67     df.set_index('last_update_date', inplace=True)
68     sample = '1h'
69     occupancy = df['available_bike_stands'].resample(sample).mean()
70     availability = df['available_bikes'].resample(sample).mean()
71
72     return jsonify(occupancy=occupancy.to_json(), availability=availability.t
73
```

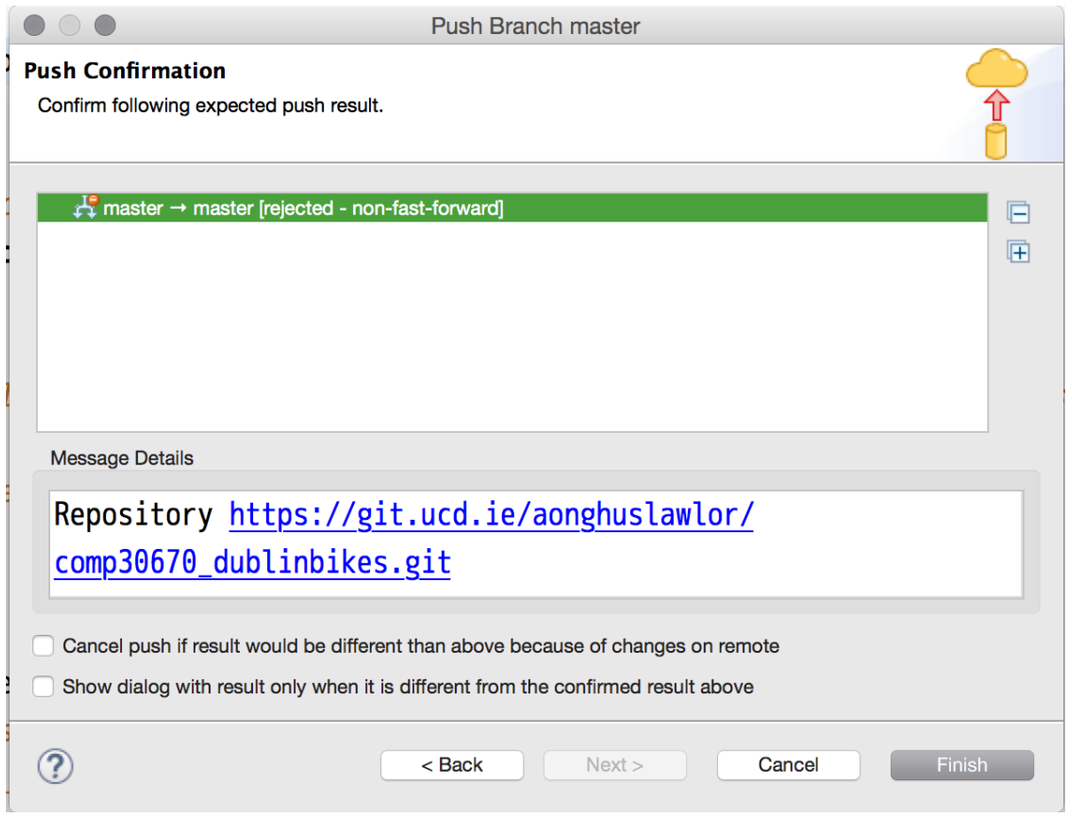
read csv - 92b466d

```
48     return jsonify(stations=stations)
49
50
51 @app.route("/station_occupancy_test/<int:station_id>")
52 def get_station_occupancy_test(station_id):
53     conn = get_db()
54 #     df = pd.read_sql_query("select * from dbikes where number = :number",
55     df = pd.read_csv()
56     df['last_update_date'] = pd.to_datetime(df.last_update, unit='ms')
57     df.set_index('last_update_date', inplace=True)
58     res = df['available_bike_stands'].resample('1d').mean()
59     print station_id, len(res)
60     return res.to_json()
61
62
63 @app.route("/station_occupancy_timeline/<int:station_id>")
64 def get_station_occupancy_timeline(station_id):
65     conn = get_db()
66     df = pd.read_sql_query("select * from dbikes where number = :number",
67     df['last_update_date'] = pd.to_datetime(df.last_update, unit='ms')
68     df.set_index('last_update_date', inplace=True)
69     sample = '1h'
70     occupancy = df['available_bike_stands'].resample(sample).mean()
71     availability = df['available_bikes'].resample(sample).mean()
72
73     return jsonify(occupancy=occupancy.to_json(), availability=availabilit
74
```

Eclipse merge tool gives some options to select changes from left (your version) or right (upstream version). Or you can manually edit the file. Save the changes and then continue with the merge.

# Merge Conflicts

You can now push your commit to upstream




Resolving a Merge Conflict (Egit)

Eclipse - Resolving Merge conflicts