

Asyncio Stack & React.js

or Development on the Edge

Intro

I am...

- **Igor Davydenko**
- Python & React.js developer
- From Kyiv, Ukraine
- Works on Ezhome Inc.
- Primarily designs & develops backend API

- [Personal Site](#)
- [@ GitHub](#)
- [@ Twitter](#)

My Path

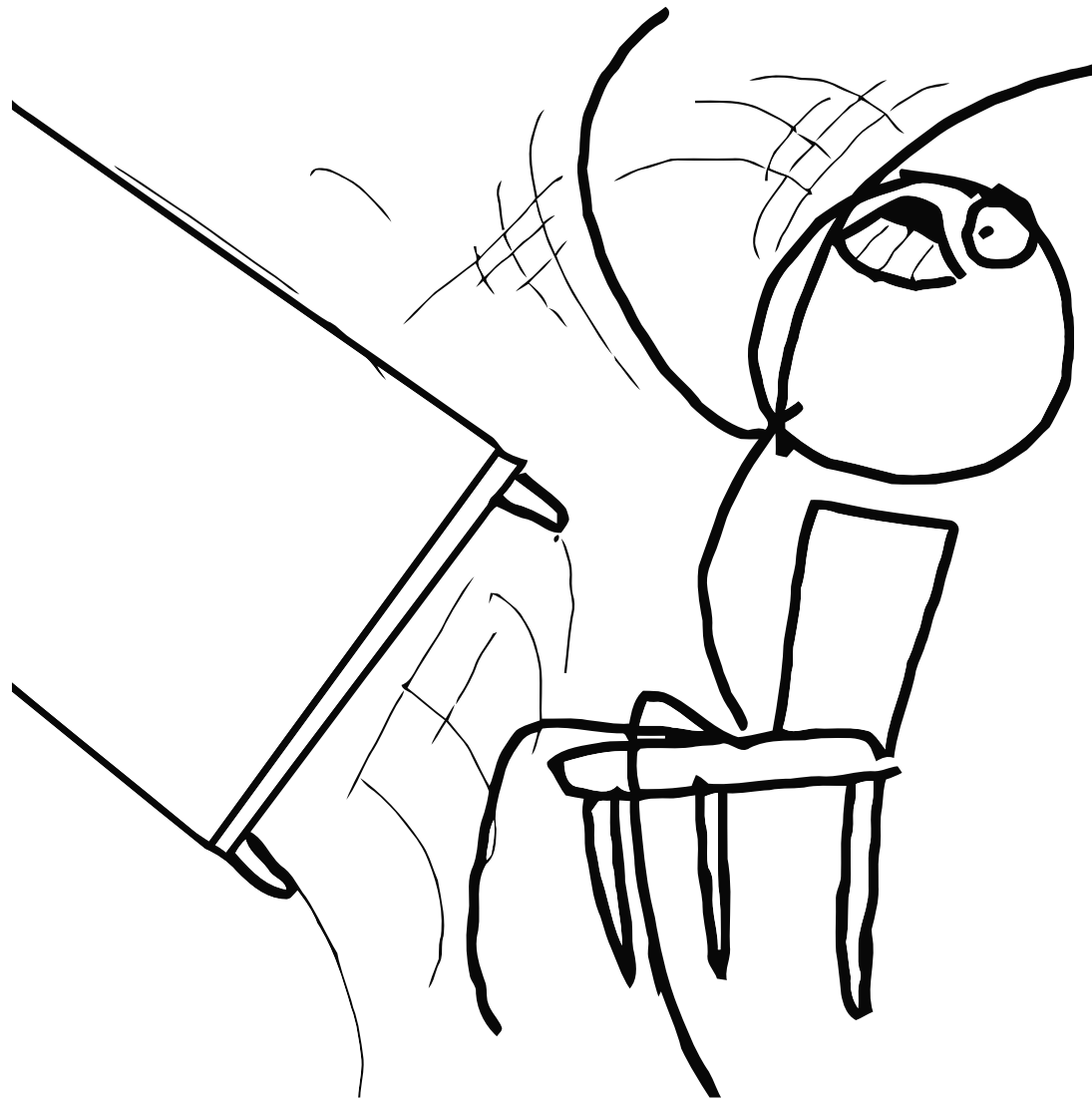
- Everything started from Microsoft FrontPage, **early 2002**
- Met PHP4, HTML4, **early 2003**
- First school projects, sites, **2003-2005**
- First work, **2006**
- Met PHP5, late **2006**
- Own web design studio, **2007**
- Switch to Python & Django, **late 2007**

My Path

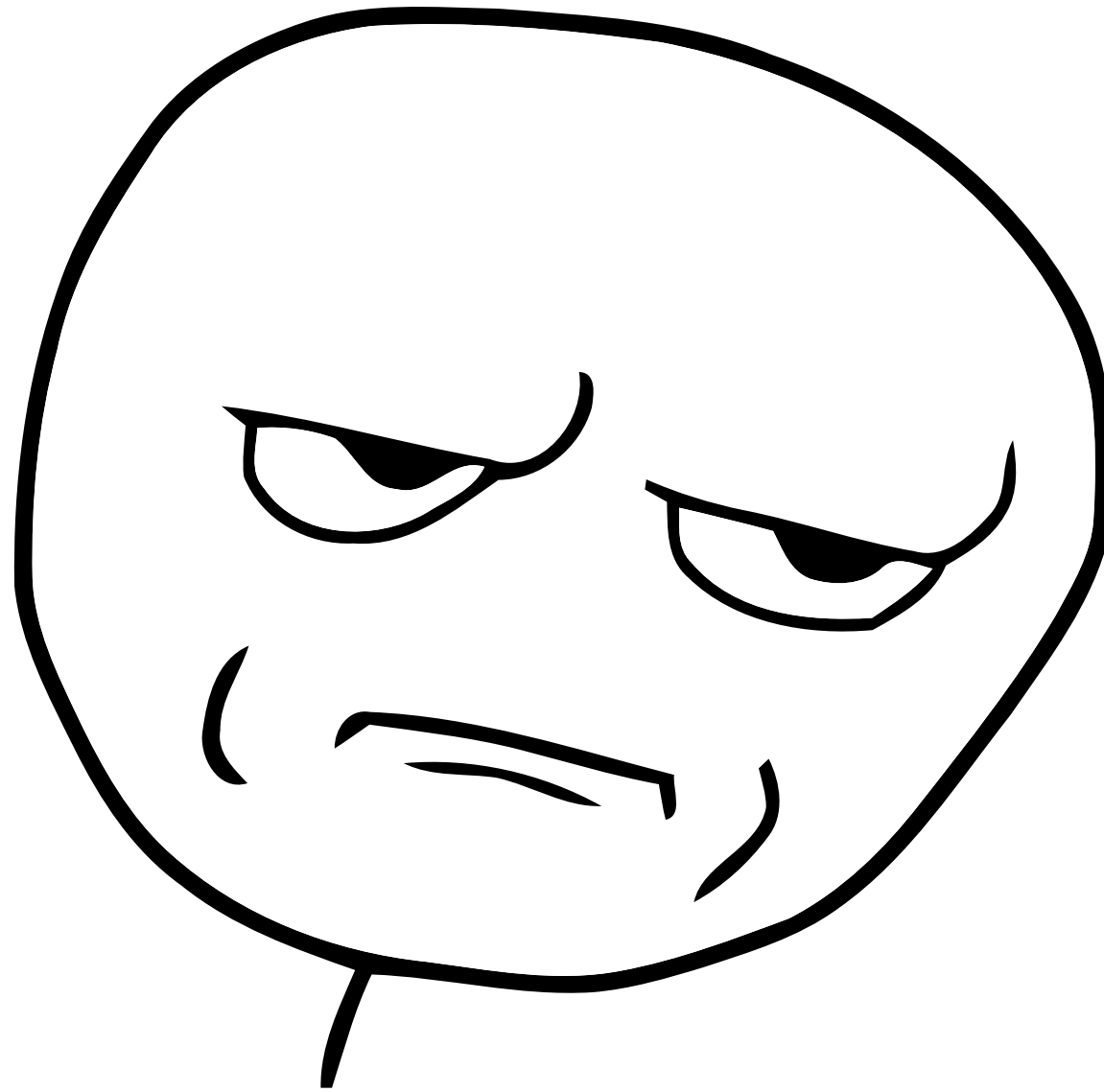
- Outsourcing career, **2008-2011**
Django, Django, Django
- oDesk PS, **2011-2012**
Django is good, but Flask is better
- GetGoing Inc., **2012-2015**
Flask, Flask, Flask. Oh no, Django again
- Ezhome Inc., **early 2015**
Django REST Framework, okay

In Total

- Python developer for past **8** years
- Using Django from **0.96**
- Using Flask from **0.7**
- **Still not satisfied...**



Hello, JavaScript!



JavaScript was bad

- **Prototype** was big and hard
- **jQuery** was small and easy. *Somewhen, decades ago*
- **jQuery UI** :(
- **jQuery plugins** :(:(

JavaScript was bad

- My motto was: *Let someone else make frontend Tasks*
- **Backbone?** Okay, but not in big teams
- **Angular?** No, no, no. Just no
- **Vanilla JS?** Cool, but not in big teams

JavaScript problems

- No standart modules
 <script>-hell in templates
- Hard to maintain across distributed team
 One developer -> one code style, X developers -> X code styles
- Hard to maintain between projects
 Same vendor / directories, no dependencies management

But then...

- **node.js** happens
- **npm** installs dependencies
- **CommonJS** allows reuse your code
- **browserify** builds bundles from your code
- **jshint/jslint/jsrc** lints your code
- *Even some people starts write backends in node.js*

Before

In **template.html**,

```
<script src="/path/to/underscore.js" type="text/javascript"></script>
<script type="text/javascript">
  _.each(document.getElementsByTagName("a"), function(item) {
    if (item.href.indexOf("http://") || item.href.indexOf("https://")) {
      item.href = "/go?" + item.href;
      item.onclick = function() {
        item.classList.add("visited-link");
        return False;
      }
    }
  });
  ...
</script>
```

Now

In **template.html**,

```
<script src="/path/to/bundle.js" type="text/javascript"></script>
<script type="text/javascript">
  processExternalLinks(document.getElementsByTagName("a"));
</script>
```

Now

In **js/external-links.js**,

```
import {each} from "underscore";

function handleExternalLinkClick(evt) {
  evt.preventDefault();
  evt.target.classList.all("visited-link");
}

export function processExternalLinks(elements) {
  each(elements, item => {
    if (/^https?:\/\/\//.test(item.href)) {
      item.href = "/go?" + item.href;
      item.addEventListener("click", handleExteranlLinkClick);
    }
  });
}

export default {
  processExternalLinks: processExternalLinks
};
```


Welcome ECMAScript 2015

Previously known as ES6

ES2015. Modules

./lib/myLibrary.js

```
export function myFunction() { ... }  
  
export default {  
  myFunction: myFunction  
};
```

./app.js

```
import myLibrary from "./lib/myLibrary";  
import {myFunction} from "./lib/myLibrary";
```

ES2015. Let + Const

```
const unchangeable = true;

let changeable;
changeable = true;

unchangeable = false; // Error
```

ES2015. Arrows

```
const episodes = [...];
const aNewHope = episodes.filter(item => item.episode === "IV");

const episodeTitles = episodes.map(item => {
  return item.title.toUpperCase();
})
```

ES2015. Classes

```
class ANewHope extends Episode {
  id: "IV",
  name: "A New Hope",

  constructor() {
    super();
    this.directedBy = "George Lucas";
  }

  render() {
    ...
  }
}

const aNewHope = new ANewHope();
```

ES2015. Template Strings

```
const episode = 'IV';  
const title = 'A New Hope';  
`A long time ago in a galaxy far,  
far away...`
```

```
STAR  
WARS
```

```
Episode ${episode}  
${title.toUpperCase()}`
```

```
It is a period of civil war.  
Rebel spaceships, striking  
from a hidden base, have won  
their first victory against  
the evil Galactic Empire.`
```

ES2015. Destructuring

```
const [first, , third] = [1, 2, 3, 4, 5, 6];
const {episode, title} = {
  episode: "IV",
  title: "A New Hope",
  opening: "..."};
const {episode: id, title: name} = {...};
```

ES2015. Map + Set

```
var mapping = new Map();
map.set("IV", "A New Hope");
map.get("IV") === "A New Hope";

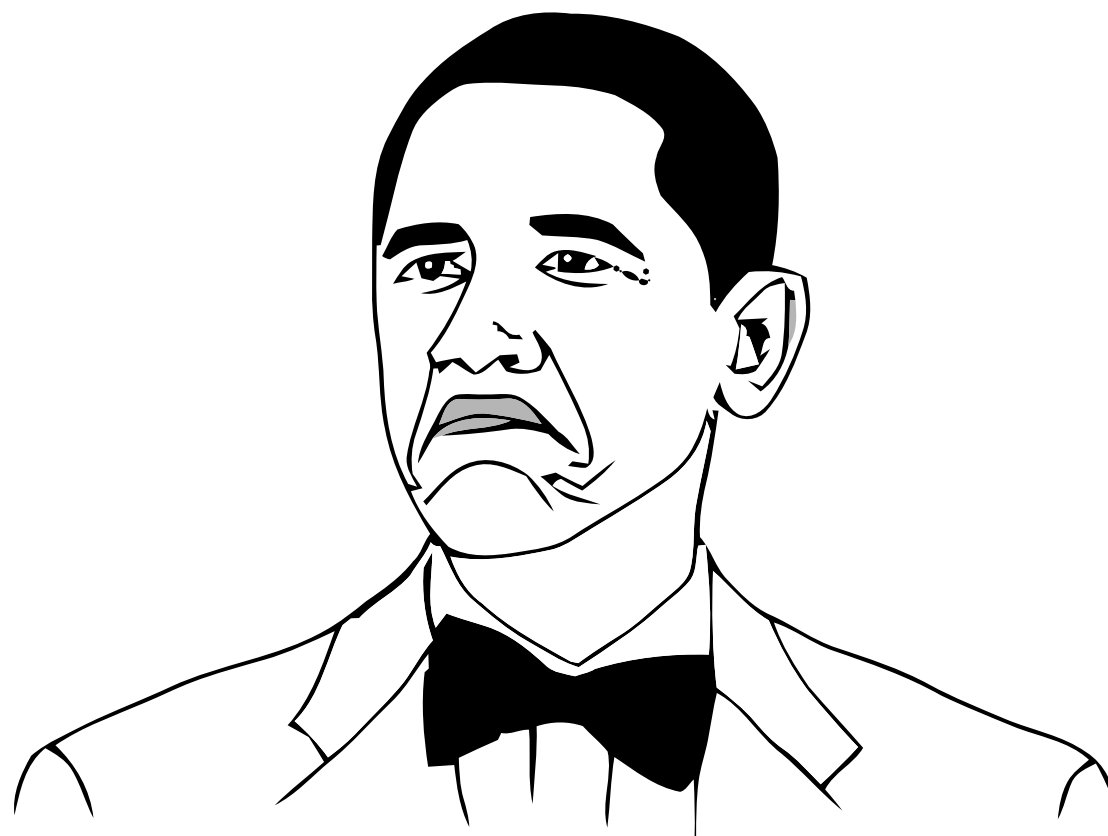
var episodes = new Set();
episodes
  .add("A New Hope")
  .add("The Empire Strikes Back")
  .add("Return of the Jedi")
  .add("A New Hope");
episodes.has("A New Hope");
episodes.size === 3;
```


And More

- **Learn ES2015**
- Default + Rest + Spread
- Iterators + For..Of
- Generators
- Improved Unicode
- Module Loaders
- Proxies
- Symbols
- ...

And More

- **ES2016, ES2017, ...**
- Comprehensions
- Class Properties
- Function Bind
- Async Functions
- Decorators
- ...



NOT BAD

How to Use?

- Use **Babel** for all good things!
- Supports browserify, webpack, ...
- Emerges eslint as your one and only JS linter

So here comes **React.js**

React.js

- **Painless UI Framework**
- Virtual DOM
- One-way reactive data flow
- Has a strong community around

React.js. A Simple Component

```
import React, {Component} from "react";

class EuroPython extends Component {
  render() {
    return (
      <div>Hello, EuroPython 2015!</div>
    );
  }
}

React.render(<EuroPython />, document.getElementById("react-container"));
```

React.js. A Stateful Component

```
class EuroPython extends Component {
  state = {clicks: 0},

  handleClick = () => {
    this.setState({clicks: this.state.clicks + 1});
  },

  render() {
    const {clicks} = this.state;
    return (
      <div>
        This button clicked {clicks} time(s).<br />
        <button onClick={this.handleClick}>
          Click Me
        </button>
      </div>
    )
  }
}

React.render(<EuroPython />, document.getElementById("react-container"));
```


React.js. Using Components

```
import Markdown from "../components/Markdown";

class CommentForm extends Component {
  state = {comment: "", preview: false},
  handleCommentChange = (evt) => {
    this.setState({comment: evt.target.value});
  },
  handlePreviewClick = (evt) => {
    this.setState({preview: true});
  },
  render() {
    const {comment, preview} = this.state;
    if (preview) return <Markdown>{comment}</Markdown>;
    return (
      <div>
        <textarea onChange={this.handleCommentChange} placeholder="Your Comment" value={comment} />
        <button onClick={this.handlePreviewClick}>Preview</button>
      </div>
    );
  }
}
```

React.js. Fetching data

```
class Comments extends Component {
  state = {data: [], status: null},
  componentDidMount() {
    this.loadData();
  },
  loadData = () => {
    fetch(apiUrl)
      .then(response => {
        if (response.ok()) {
          return Promise.resolve(response.json());
        }
        return Promise.reject(new Error(response.statusText || response.status));
      })
      .then(
        json => {
          this.setState({data: json, status: true});
        },
        () => {
          this.setState({data: [], status: false});
        }
      )
  };
},
...

```

React.js. Fetching data

```
...
render() {
  const {data, status} = this.state;
  if (status === null) {
    ... // Loading
  } else if (status === false) {
    ... // Server error
  } else if (!data.length) {
    ... // Empty data
  } else {
    ... // Valid data
  }
}
}
```

React.js. One-way data binding

```
class Comments extends Component {  
  ...  
  render() {  
    const content = this.state.data.map(item => (  
      <Comment data={item} key={"comment-" + item.id} />  
    ));  
  }  
}  
  
class Comment extends Component {  
  static defaultProps = {data: {}},  
  propTypes = {  
    data: PropTypes.shape({...})  
  },  
  render() {  
    return (...)  
  }  
}
```

React.js. One-way data binding

- **Comments** is Higher Order Component
 - **Comment** is Dumb Component
-

Higher Order Components

- Data loading
- Events handling

Dumb Components

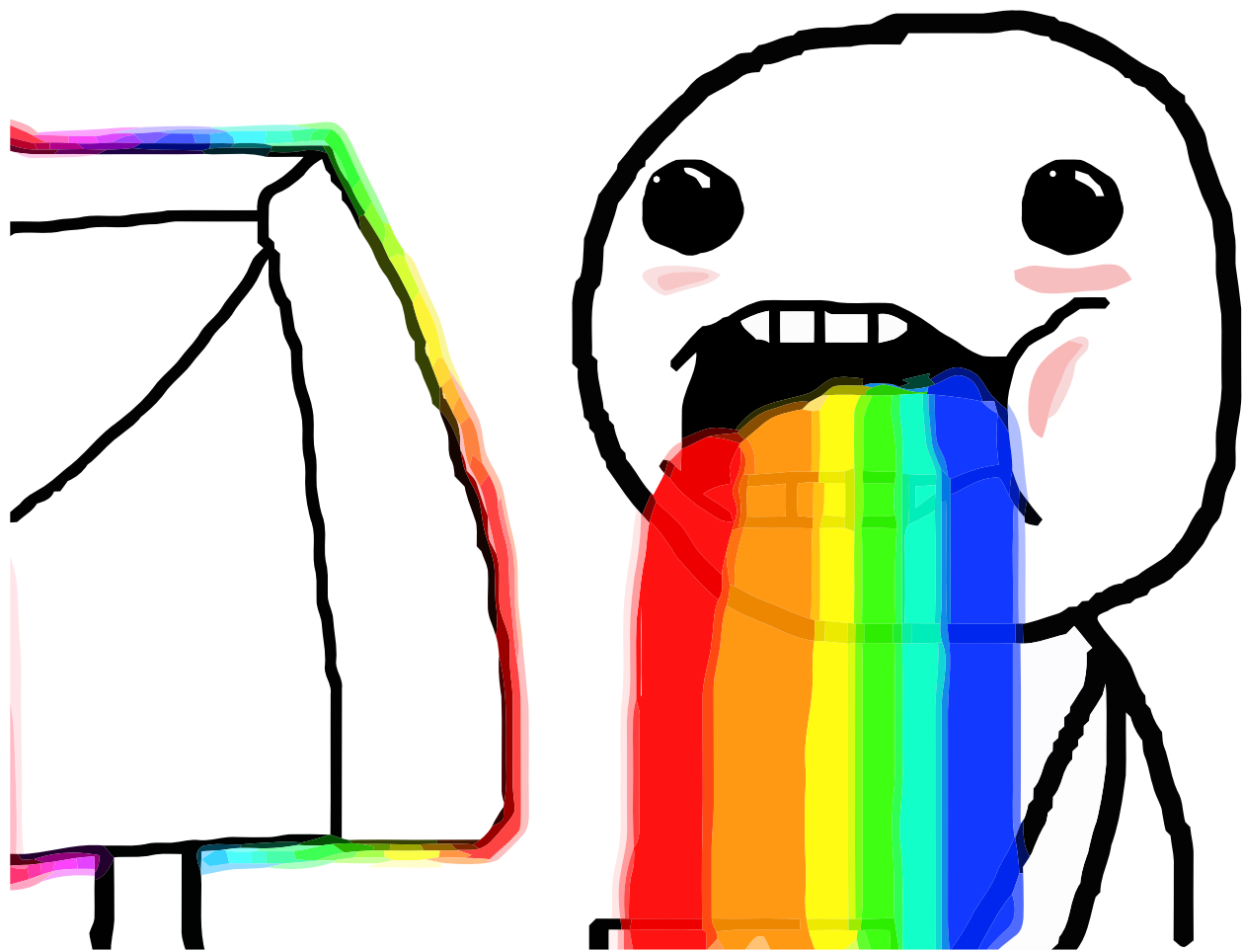
- Set of reusable Components
- Shareable across one and many projects

And More

- **React DOM** is separate package from 0.14
 - **React Native, React Canvas**
-

- **Routing** via [react-router](#)
 - **Reusable Components**
 - [react-bootstrap](#)
 - [react-dnd](#)
-

- **Flux**
- **Relay**
- **GraphQL**



**JavaScript is good now,
but what about Python?**

Let me introduce **Asyncio Stack**

asyncio

- Asynchronous I/O, event loop, coroutines, and tasks
- <https://docs.python.org/3/library/asyncio.html>

```
import asyncio

@asyncio.coroutine
def hello():
    return "Hello, world!"

loop = asyncio.get_event_loop()
content = loop.run_until_complete(hello())
print(content)
loop.close()
```

- Included in Python 3.4 and later
- Available in Python 3.3 with `$ pip install asyncio`
- Backported to Python 2.7 as [trollius](#) (*I don't recommend to use it anyway*)

aiohttp

- HTTP client/server for asyncio
- Latest version: 0.16.5
- <http://aiohttp.readthedocs.org/>

```
import asyncio
import aiohttp

@asyncio.coroutine
def fetch_page(url):
    response = yield from aiohttp.request('GET', url)
    assert response.status == 200
    return (yield from response.read())

loop = asyncio.get_event_loop()
content = loop.run_until_complete(fetch_page('http://ep2015.europython.eu/'))
print(content)
loop.close()
```

aiohttp.web

- Web framework for asyncio
- <http://aiohttp.readthedocs.org/en/v0.16.5/web.html>

```
import asyncio
from aiohttp import web

@asyncio.coroutine
def hello(request):
    return web.Response(body='Hello, world!', content_type='text/plain')

app = web.Application()
app.router.add_route('GET', '/', hello)
```

```
$ gunicorn -k aiohttp.worker.GunicornWebWorker -w 9 -t 60 app:app
```

aiopg

- Accessing PostgreSQL database from the asyncio
- Latest version: 0.7.0
- <http://aiopg.readthedocs.org/>

```
import asyncio
from aiopg import create_pool

@asyncio.coroutine
def hello(dsn):
    pool = yield from create_pool(dsn)
    with (yield from pool.cursor()) as cursor:
        yield from cursor.execute('SELECT 1')
        selected = yield from cursor.fetchone()
        assert selected == (1, )

loop = asyncio.get_event_loop()
loop.run_until_complete(hello('dbname=aiopg user=... password=... host=...'))
loop.close()
```

aioredis / asyncio_redis

- Accessing Redis dataset from the asyncio
- aioredis from aio-libs
- Latest version: 0.1.5
- <http://aioredis.readthedocs.org/>
- asyncio_redis from third-party developers
- Latest version: 0.13.4
- <http://asyncio-redis.readthedocs.org/>

And many others

- [Python Asyncio Resources](#)
- MySQL: [aiomysql](#)
- Mongo: [asyncio_mongo](#)
- CouchDB: [aiocouchdb](#)
- ElasticSearch: [aios](#)
- Memcached: [aiomcache](#)
- AMQP: [aiomqp](#)
- ØMQ: [aiomq](#)

And many others

- All Asyncio projects @ PyPI
- S3: aio-s3
- SSH: asyncssh

- Autobahn, WebSocket & WAMP
- RxPY, Reactive Extensions for Python
- Pulsar, Concurrent framework for Python

Even web-frameworks available

- [muffin](#)
- [Induction](#)
- [Spanner.py](#)
- [Growler](#)

aiohttp.web

Architecture

- All starts from view functions (handlers)
- View functions should be a coroutine, and return `web.Response`

```
import asyncio
from aiohttp import web

@asyncio.coroutine
def index(request):
    return web.Response(body='Hello, world!', content_type='text/plain')
```

- It's good idea to put all view functions to `views.py` module

Architecture

- Next you need to create an `web.Application`
- And register handler for a request

```
from aiohttp import web

from . import views

app = web.Application()
app.router.add_route('GET', '/', views.index)
```

- Obvious to put application code to `app.py` module

Architecture

- Now you ready to serve your application
- I recommend to use [Gunicorn](#)

```
$ gunicorn -b 0.0.0.0:8000 -k aiohttp.worker.GunicornWebWorker -w 9 -t 60 project.app:app
```

- Add `--reload` flag to automatically reload Gunicorn server on code change

Handling GET/POST data

In **views.py**,

```
@asyncio.coroutine
def search(request):
    """Search by query from GET params."""
    query = request.GET['query']
    locale = request.GET.get('locale', 'uk-UA')
    ...
    return web.Response(...)
```

Handling GET/POST data

In **views.py**,

```
@asyncio.coroutine
def submit(request):
    """Submit form POST data."""
    data = yield from request.post()
    # Now POST data available as ``request.POST``
    ...
    return web.Response(...)
```

Handling GET/POST data

In **app.py**,

```
app.router.add_route('GET', '/search', views.search)
app.router.add_route('POST', '/submit', views.submit)
```


Handling variable routes

In **views.py**,

```
@asyncio.coroutine
def project(request):
    project_id = request.match_info['project_id']
    ...
    return web.Response(...)
```

Handling variable routes

In **app.py**,

```
app.router.add_route('GET', '/projects/{project_id}', views.project)
```

Or even,

```
app.router.add_route('GET', '/projects/{project_id:\d+}', views.project)
```

Named routes, reverse constructing, and redirect

In **app.py**,

```
app.router.add_route('GET', '/projects', views.projects, name='projects')
app.router.add_route('POST', '/projects', views.add_project)
```

Named routes, reverse constructing, and redirect

In **views.py**,

```
@asyncio.coroutine
def add_project(request):
    data = yield from request.post()
    ...
    url = request.app.router['projects'].url()
    return web.HTTPFound(url)

@asyncio.coroutine
def projects(request):
    ...
```

You don't need to `from app import app`

- Or `from flask import current_app` either
- Request contains app instance for all your needs

In **app.py**,

```
from . import settings
...
app['settings'] = settings
```

You don't need to `from app import app`

In **views.py**,

```
@asyncio.coroutine
def search(request):
    settings = request.app['settings']
    query = request.GET['query']
    locale = request.GET.get('locale', settings.DEFAULT_LOCALE)
    ...
    return web.Response(...)
```

Middlewares

- `web.Application` accepts optional middlewares factories sequence
- Middleware Factory should be a coroutine and returns a coroutine

In **app.py**,

```
@asyncio.coroutine
def trivial_middleware(app, handler):
    @asyncio.coroutine
    def middleware(request):
        return (yield from handler(request))
    return middleware

...

app = web.Application(middlewares=[trivial_middleware])
```

Middlewares

Ready to use Middlewares

- User Sessions, [aiohttp_session](#)
- Debug Toolbar, [aiohttp_debugtoolbar](#)

In **app.py**,

```
import aiohttp_debugtoolbar
from aiohttp_debugtoolbar import toolbar_middleware_factory
from aiohttp_session import session_middleware
from aiohttp_session.cookie_storage import EncryptedCookieStorage

app = web.Application(middlewares=[
    toolbar_middleware_factory,
    session_middleware(EncryptedCookieStorage(b'1234567890123456'))
])
aiohttp_debugtoolbar.setup(app)
```


Middlewares

Handling Exceptions

In `app.py`,

```
@asyncio.coroutine
def errorhandler_middleware(app, handler):
    @asyncio.coroutine
    def middleware(request):
        try:
            return (yield from handler(request))
        except web.HTTPError as err:
            # As it special case we could pass ``err`` as second argument to
            # error handler
            return (yield from views.error(request, err))
    return middleware
```

User Sessions

- [aiohttp_session](#)
- Latest version: 0.1.1
- Supports storing session data in encrypted cookie or redis
- Enabled by passing `session_middleware` to middleware factories sequence

In **views.py**,

```
from aiohttp_session import get_session

@asyncio.coroutine
def login(request):
    ...
    session = yield from get_session(request)
    session['user_id'] = user_id
    return web.Response(...)
```

Rendering Templates

[Jinja2](#) & [Mako](#) supported via [aiohttp_jinja2](#) & [aiohttp_mako](#)

Jinja2 Support

In `app.py`,

```
import aiohttp_jinja2
import jinja2

...

aiohttp_jinja2.setup(
    app,
    loader=jinja2.FileSystemLoader('/path/to/templates')
)
```

Rendering Templates

Jinja2 Support

In **views.py**,

```
import aiohttp_jinja2

@aiohttp_jinja2.template('index.html')
def index(request):
    return {'is_index': True}
```

Rendering Templates

Jinja2 Support

In **views.py**,

```
from aiohttp_jinja2 import render_template

@asyncio.coroutine
def index(request):
    return render_template('index.html', request, {'is_index': True})
```

Rendering JSON

In **utils.py**,

```
import ujson

from aiohttp import web

def json_response(data, **kwargs):
    # Sometimes user needs to override default content type for JSON
    kwargs.setdefault('content_type', 'application/json')
    return web.Response(ujson.dumps(data), **kwargs)
```

Note: I recommend to use [ujson](#) for work with JSON data in Python, cause of speed.

Rendering JSON

In **views.py**,

```
from .utils import json_response

@asyncio.coroutine
def api_browser(request):
    return json_response({
        'projects_url': request.app.router['projects'].url(),
    })
```

Serving Static Files

Important: It's highly recommend to use nginx, Apache, or other web server for serving static files in production.

In **app.py**,

```
app.router.add_static('/static', '/path/to/static', name='static')
```


Serving Static Files

In **views.py**,

```
@aiohttp_jinja2.template('index.html')
def index(request):
    return {'app': request.app, 'is_index': True}
```

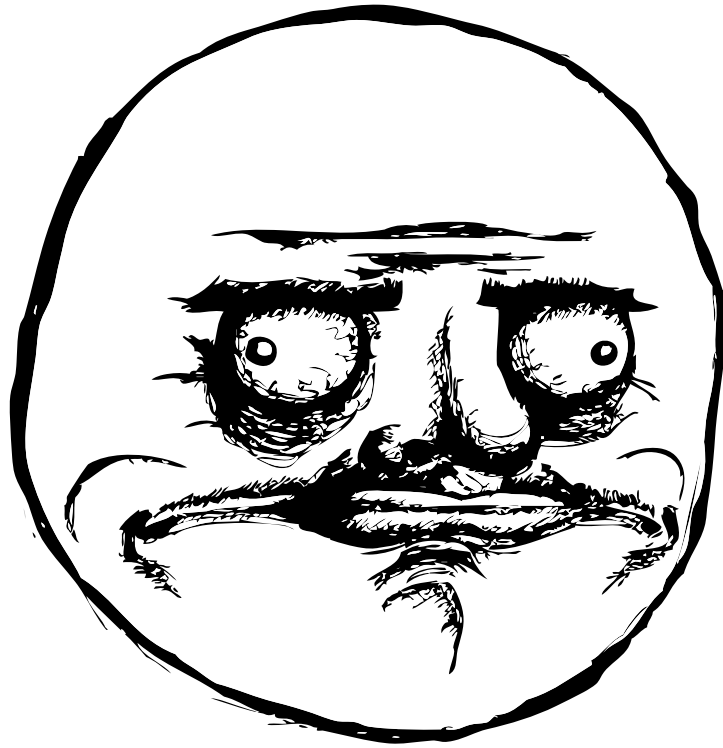
Serving Static Files

In **index.html**,

```
<script src="{{ app.router.static.url(filename="dist/js/project.js") }}"  
  type="text/javascript"></script>
```

And More

- WebSockets
- Expect Header
- Custom Conditions for Routes Lookup
- Class Based Handlers
- And I say it again, **WebSockets**



Real World Usage

Structure

yourproject

- **api**
 - storage.py
 - views.py
- **auth**
 - api.py
 - views.py
- **static**
- **templates**
- app.py
- settings.py
- storage.py
- views.py

Add Route Context Manager

```
from contextlib import contextmanager

@contextmanager
def add_route_context(app, views, url_prefix=None, name_prefix=None):
    def add_route(method, url, name):
        view = getattr(views, name)
        url = ('/'.join((url_prefix.rstrip('/'), url.lstrip('/'))
                        if url_prefix
                        else url)
              if name_prefix else name)
        return app.router.add_route(method, url, view, name=name)
    return add_route
```

Add Route Context Manager

In **app.py**,

```
from .api import views as api_views

with add_route_context(app, api_views, '/api', 'api') as add_route:
    add_route('GET', '/', 'index')
    add_route('GET', '/projects', 'projects')
    add_route('POST', '/projects', 'add_project')
    add_route('GET', '/project/{project_id:\d+}', 'project')
    add_route('PUT', '/project/{project_id:\d+}', 'edit_project')
    add_route('DELETE', '/project/{project_id:\d+}', 'delete_project')
```


Add Route Context Manager

In **api/views.py**,

```
import asyncio

from ..utils import json_response

@asyncio.coroutine
def index(request):
    router = request.app.router

    project_url = router['api.proejct'].url(parts={'project_id': 42})
    project_url = project_url.replace('42', '{project_id}')

    return json_response({
        'urls': {
            'add_project': router['api.add_project'].url(),
            'project': project_url,
            'projects': router['api.projects'].url(),
        }
    })
```

Immutable Settings

- Python 3.3+ has [MappingTypeProxy](#)
- Settings shouldn't be changed across the app
- Welcome, [rororo.settings](#)

In **app.py**,

```
from rororo.settings import immutable_settings

from . import settings

def create_app(**options):
    settings_dict = immutable_settings(settings, **options)
    app = web.Application()
    app['settings'] = settings_dict
    ...
    return app
```

Other Settings & Logging Helpers

rororo.settings

- inject_settings
- setup_locale
- setup_logging
- setup_timezone
- to_bool

rororo.logger

- default_logging_dict
- update_sentry_logging

Supports DEBUG Mode from Settings

```
class Application(web.Application):  
  
    def __init__(self, **kwargs):  
        self['settings'] = kwargs.pop('settings')  
        super().__init__(**kwargs)  
  
    def make_handler(self, **kwargs):  
        kwargs['debug'] = self['settings']['DEBUG']  
        kwargs['loop'] = self.loop  
        return self._handler_factory(self, self.router, **kwargs)
```

Schemas

- You need to validate request & response data
- Use [JSON Schema](#) for validation
- Describe Schemas with [jst](#)
- Welcome, [rororo.schemas](#)

In **api/views.py**,

```
from rororo.schemas import Schema

from . import schemas

@asyncio.coroutine
def add_project(request):
    schema = Schema(schemas.add_project, response_factory=json_response)
    data = schema.validate_request((yield from request.post()))
    ...
    return schema.make_response(project_dict)
```

Schemas

Describing Schemas

In `api/schemas/add_project.py`,

```
from jsl import Document, IntegerField, StringField

class Project(Document):
    name = StringField(min_length=1, max_length=32, required=True)
    slug = StringField(min_length=1, max_length=32, required=True)
    description = StringField(max_length=255)

class Response(Project):
    id = IntegerField(minimum=0, required=True)

request = Project.get_schema()
response = Response.get_schema()
```

Or use plain Python dicts instead

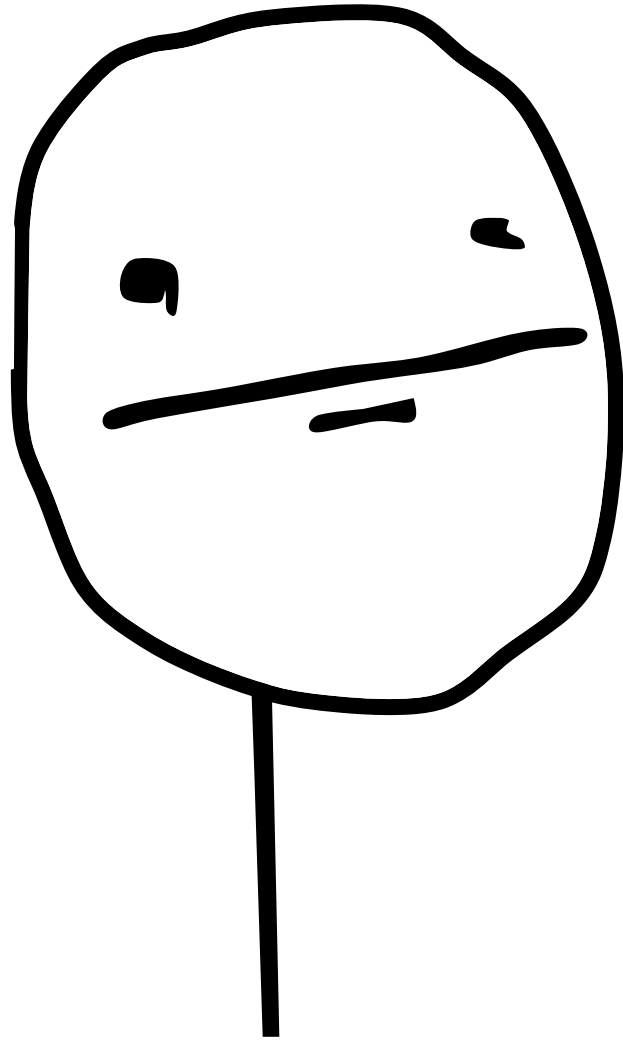
Schemas

Describing Schemas

In `api/schemas/__init__.py`,

```
from . import add_project # noqa
```

Why you might need Asyncio Stack?



~~**Cause it cool and trendy!**~~

Database-less API

Requests to the external API

Async code execution

Predictable async IO

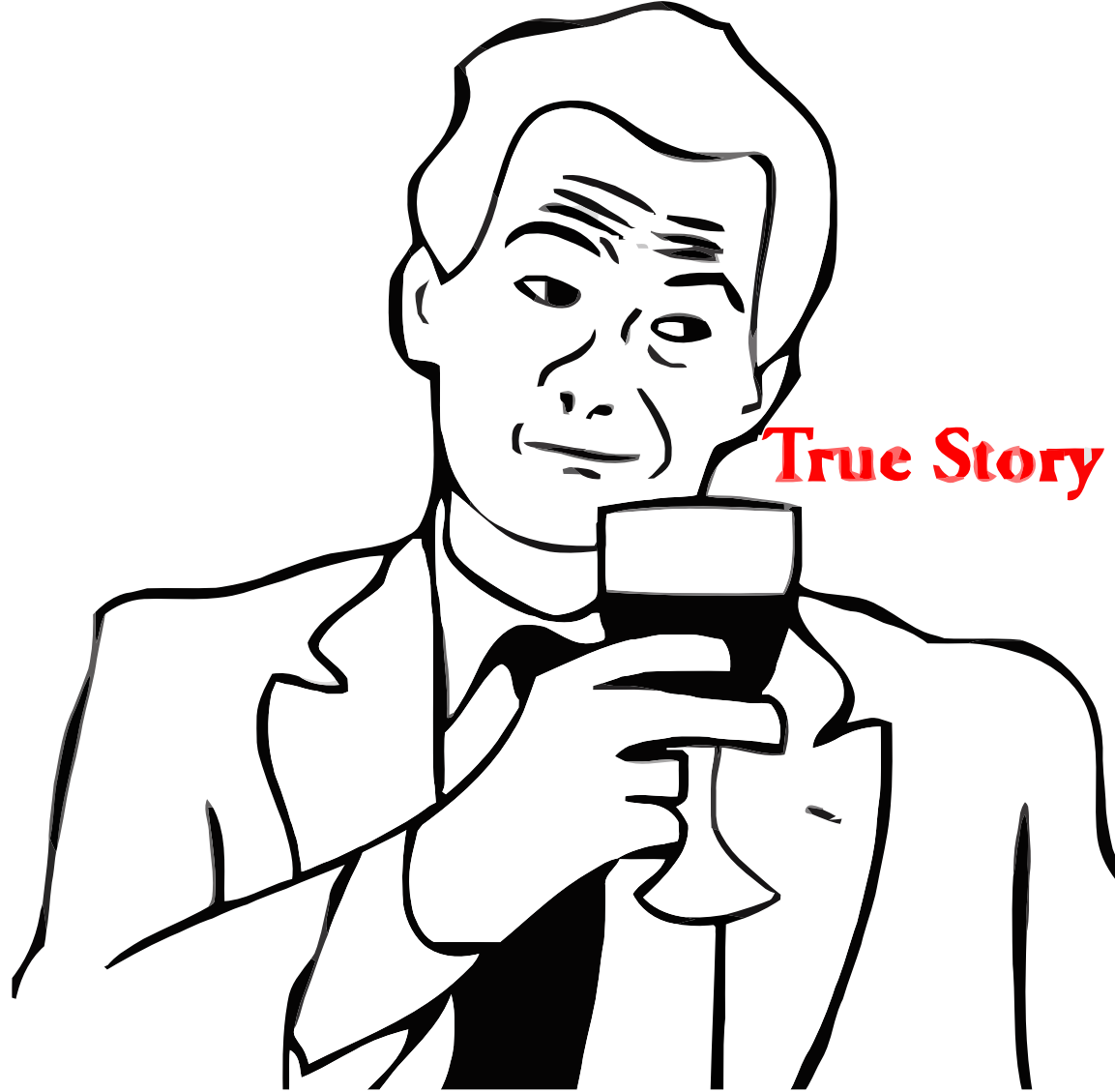
Summary

JavaScript is quite good right now

- **ES2015 is a great step in right direction**
- Many tools around JS is maturing too
- Bundling is easy, meet **webpack**
- Linting is easy, meet **eslint**
- Making UI is easy, meet **react**
- *And I still not talking about DX tools :)*

Asyncio Stack is ready for usage

- **The Future is Here!**
- Use Python 3.4 for all good things
- aiohttp.web, easy to start and easy to use
- aiopg.sa allows you to forget about ORM
- rororo contains useful helpers for aiohttp.web apps
- If you miss something, port it to Asyncio stack by **yourself**
- Don't forget to payback to Open Source Software



True Story

Questions?

Bonus. Python 3.5

```
async def logout(request):
    session = await get_session(request)
    session.invalidate()
    return Response(status=204)

async def projects(request):
    projects = []
    async with create_engine(DSN) as conn:
        query = ...
        async for project in conn.execute(query):
            projects.append(project)
    return json_response(projects)
```