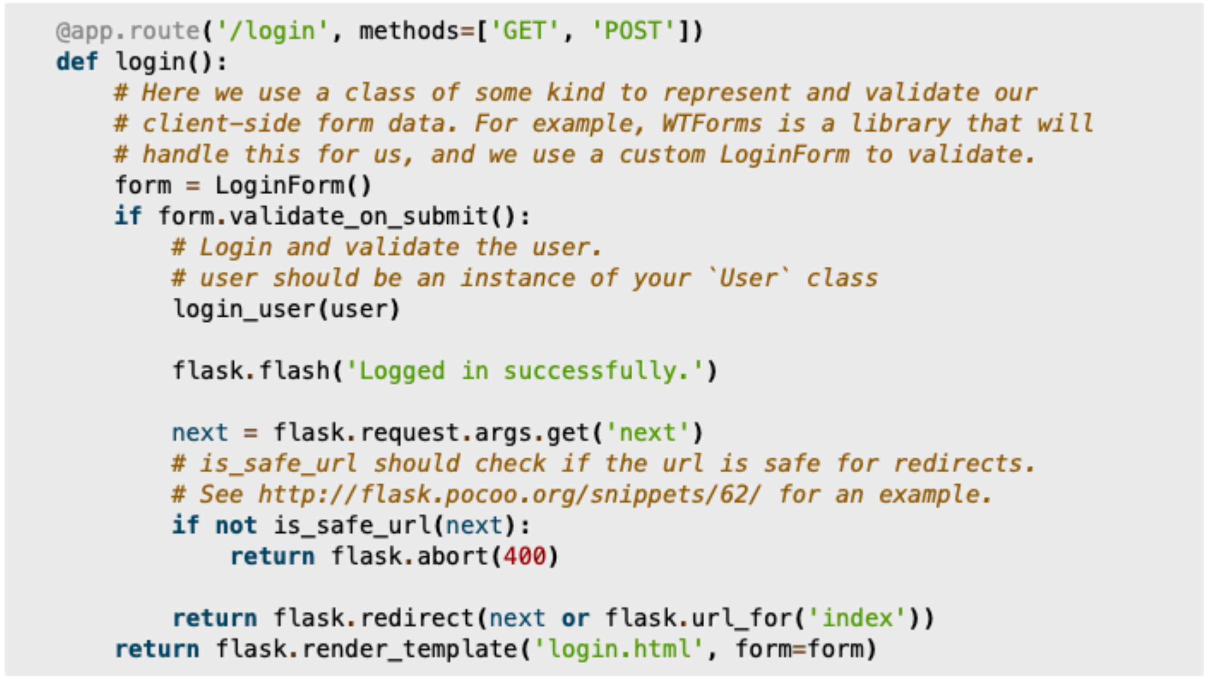
Auth0 Flask Portal

Sung Eun Bae

Objectives

- Authentication : Login-protected contents
- Authorization: Access-level control (Devel, EA, Stable, Admin)
- Login once and give access to all products that the user is allowed to access
- Should be easy to update the access-level of a product
- Should insulate the product development from tedious business layer: The dev team only focus on the functionality of the product
- Should protect from cyber-attacks or unusual activities
- Cybersecurity standards compliance

Simple + Free solution : flask-login

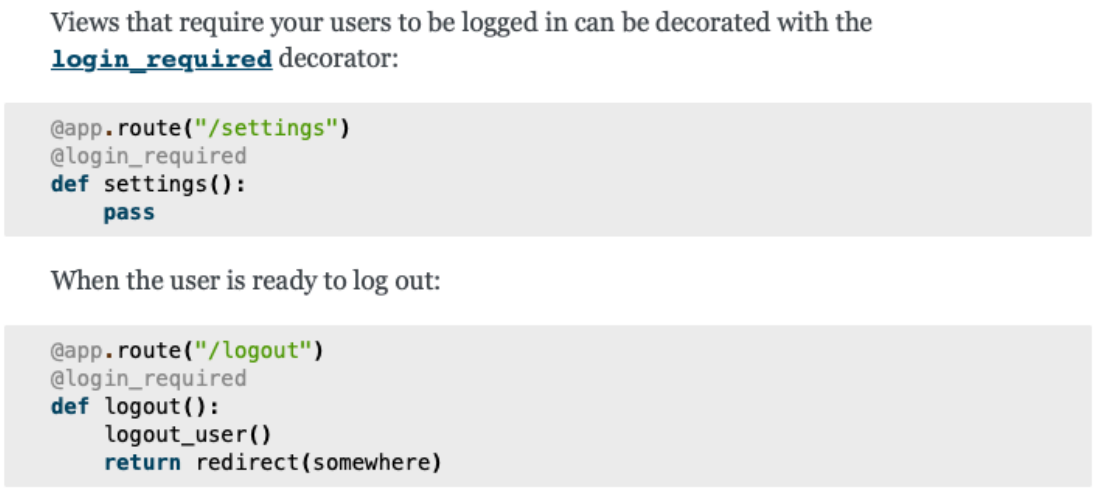


```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    # Here we use a class of some kind to represent and validate our
    # client-side form data. For example, WTForms is a library that will
    # handle this for us, and we use a custom LoginForm to validate.
    form = LoginForm()
    if form.validate_on_submit():
        # Login and validate the user.
        # user should be an instance of your `User` class
        login_user(user)

        flask.flash('Logged in successfully.')

        next = flask.request.args.get('next')
        # is_safe_url should check if the url is safe for redirects.
        # See http://flask.pocoo.org/snippets/62/ for an example.
        if not is_safe_url(next):
            return flask.abort(400)

        return flask.redirect(next or flask.url_for('index'))
    return flask.render_template('login.html', form=form)
```



Views that require your users to be logged in can be decorated with the `login_required` decorator:

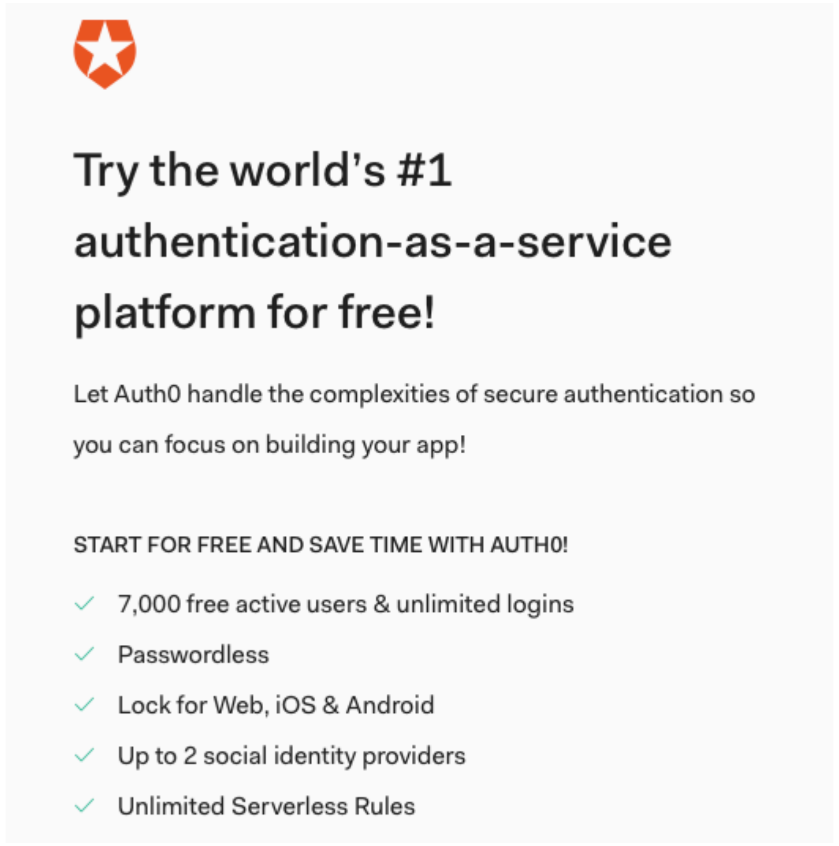
```python
@app.route("/settings")
@login_required
def settings():
    pass
```

When the user is ready to log out:

```python
@app.route("/logout")
@login_required
def logout():
    logout_user()
    return redirect(somewhere)
```

- Very simple and FREE
- No direct support for Access-level, but should be easy enough
- But we are on our own re. all the security issues

Auth0 : Commercial Solution

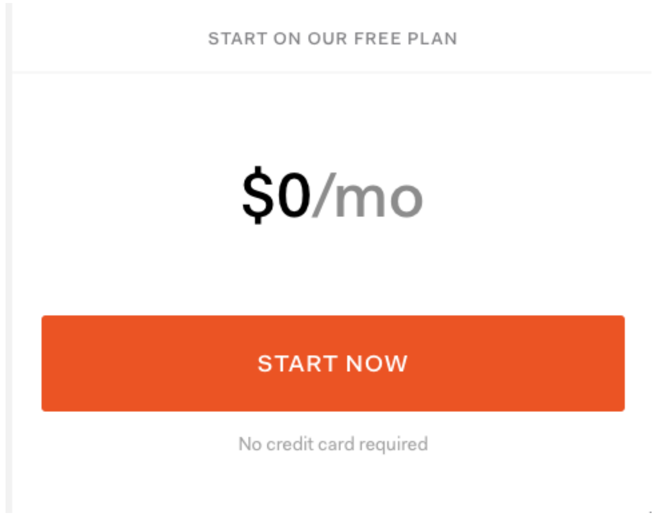


Try the world's #1 authentication-as-a-service platform for free!

Let Auth0 handle the complexities of secure authentication so you can focus on building your app!

START FOR FREE AND SAVE TIME WITH AUTH0!

- ✓ 7,000 free active users & unlimited logins
- ✓ Passwordless
- ✓ Lock for Web, iOS & Android
- ✓ Up to 2 social identity providers
- ✓ Unlimited Serverless Rules

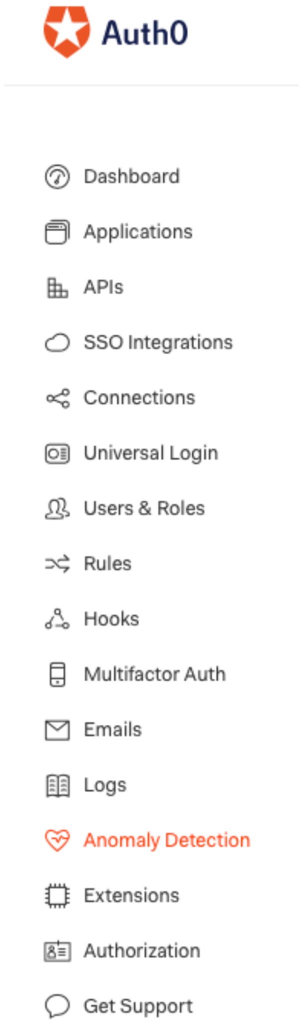


START ON OUR FREE PLAN

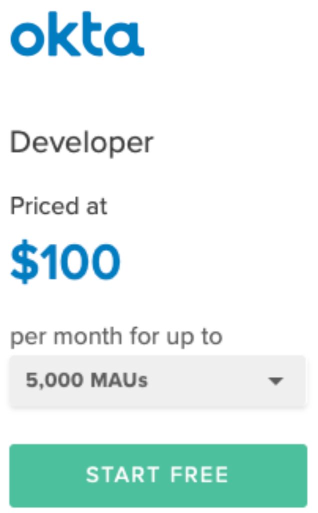
$0/mo

START NOW

No credit card required



Alternative commercial solutions?



okta

Developer

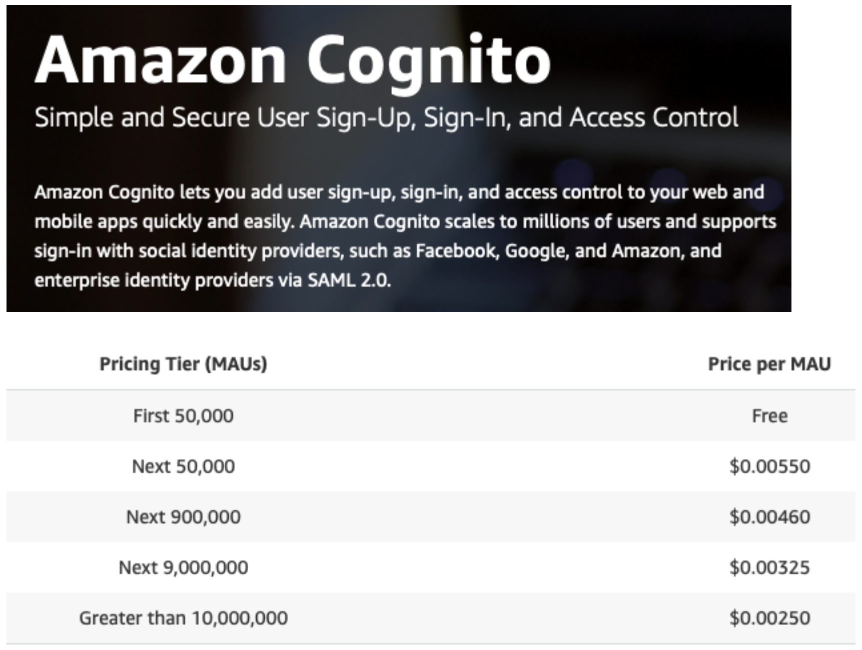
Priced at

$100

per month for up to

5,000 MAUs

Too expensive



Amazon Cognito

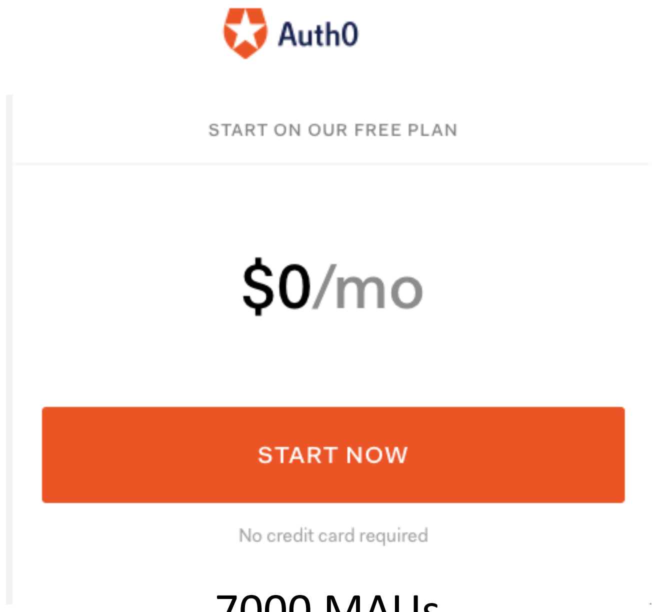
Simple and Secure User Sign-Up, Sign-In, and Access Control

Amazon Cognito lets you add user sign-up, sign-in, and access control to your web and mobile apps quickly and easily. Amazon Cognito scales to millions of users and supports sign-in with social identity providers, such as Facebook, Google, and Amazon, and enterprise identity providers via SAML 2.0.

Pricing Tier (MAUs)	Price per MAU
First 50,000	Free
Next 50,000	$0.00550
Next 900,000	$0.00460
Next 9,000,000	$0.00325
Greater than 10,000,000	$0.00250

Most generous free plan, but..

- Grouping feature is lacking
- Lack of info re. Python Flask integration
- Divorce from Amazon won't be easy (tied to Amazon IAM)



Auth0

START ON OUR FREE PLAN

$0/mo

No credit card required

7000 MAUs

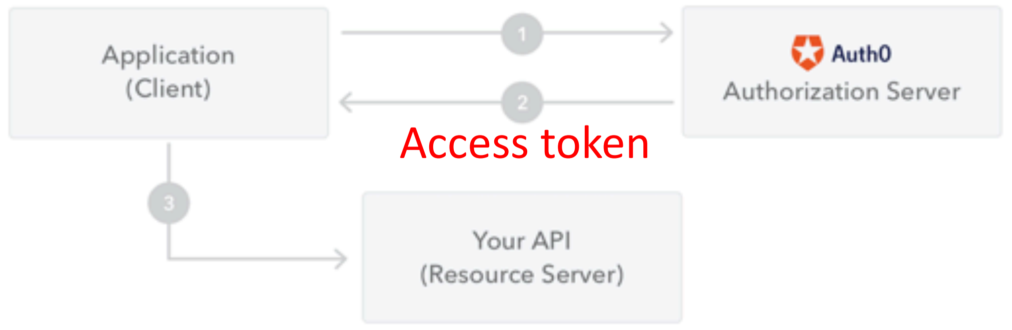
Appeared best compromise of free use cap + ease of use + devel. resource

Should we use a commercial service?

- Why not? It’s still free within limit (unlikely to exceed)
- Adopting industry’s best practice
- Cloud-based : No need to worry about security updates, anomaly detection, compliances etc.

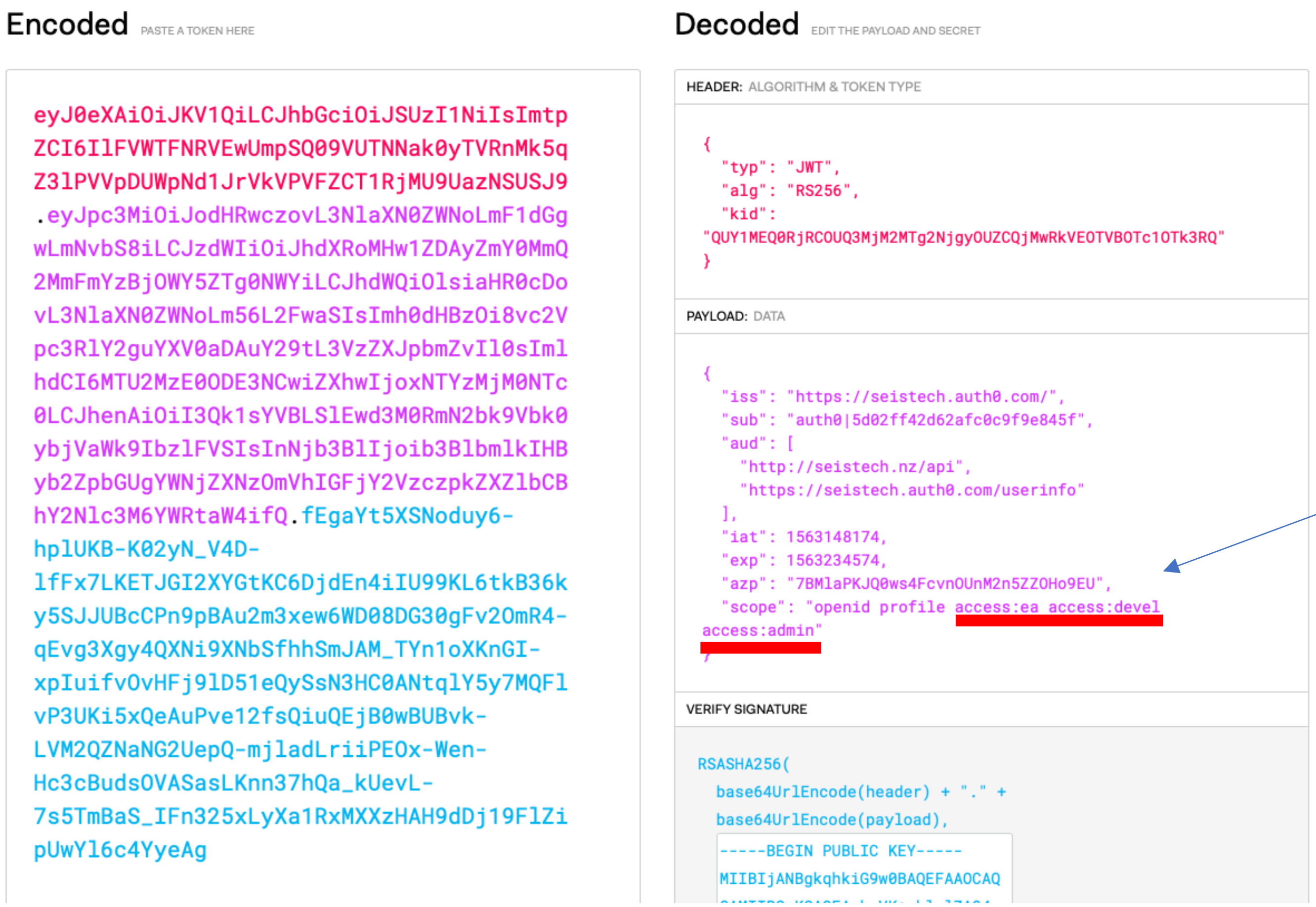
jwt (JSON Web Token)

- Open Standard defining a compact and self-contained way for securely transmitting information between parties as a JSON object.
- Digitally signed: Can be verified and trusted
- Signed with a secret or public/private key



1. The application or client requests authorization to the authorization server. This is performed through one of the different authorization flows. For example, a typical OpenID Connect compliant web application will go through the `/oauth/authorize` endpoint using the authorization code flow.
2. When the authorization is granted, the authorization server returns an access token to the application.
3. The application uses the access token to access a protected resource (like an API).

Example : Access token



Encoded PASTE A TOKEN HERE

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtp
ZCI6IlFVWTFNRVEwUmpSQ09VUTNNak0yTVRnMk5q
Z3lPVVpDUWpNd1JrVkVPVFZCT1RjMU9UazNSUSJ9
.eyJpc3MiOiJodHRwczovL3NlaXN0ZWNoLmF1dGg
wLmNvbS8iLCJzdWIiOiJhdXRoMHw1ZDAyZmY0MmQ
2MmFmYzBjOWY5ZTg0NWYiLCJhdWQiOlsiaHR0cDo
vL3NlaXN0ZWNoLm56L2FwaSIsImh0dHBzOi8vc2V
pc3RlY2guYXV0aDAuY29tL3VzZXJpbmZvIl0sIml
hdCI6MTU2MzE0ODE3NCwiZXhwIjoxNTYzMjM0NTc
0LCJhenAiOiI3Qk1sYVBLSlEwd3M0RmN2bk9Vbk0
ybjVaWk9IbzlFVSIsInNjb3BlIjoib3BlbmlkIHB
yb2ZpbGUgYWNjZXNzOmVhIGFjY2VzczpkZXZlbCB
hY2Nlc3M6YWRtaW4ifQ.fEgaYt5XSNoduy6-
hplUKB-K02yN_V4D-
lfFx7LKETJGI2XYGtKC6DjdEn4iIU99KL6tkB36k
y5SJJUBcCPn9pBAu2m3xew6WD08DG30gFv2OmR4-
qEvg3Xgy4QXNi9XNbSfhhSmJAM_TYn1oXKnGI-
xpIuifvOvHFj9lD51eQySsN3HC0ANtqlY5y7MQFl
vP3UKi5xQeAuPve12fsQiuQEjB0wBUBvk-
LVM2QZNaNG2UepQ-mjladLriiPEOx-Wen-
Hc3cBudsOVASasLKnn37hQa_kUevL-
7s5TmBaS_IFn325xLyXa1RxMXXzHAH9dDj19FlZi
pUwYl6c4YyeAg
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid":
"QUY1MEQ0RjRCOUQ3MjM2MTg2NjgyOUZCQjMwRkVEOTVBOTc1OTk3RQ"
}
```

PAYLOAD: DATA

```
{
  "iss": "https://seistech.auth0.com/",
  "sub": "auth0|5d02ff42d62afc0c9f9e845f",
  "aud": [
    "http://seistech.nz/api",
    "https://seistech.auth0.com/userinfo"
  ],
  "iat": 1563148174,
  "exp": 1563234574,
  "azp": "7BMlaPKJQ0ws4FcvnOUnM2n5ZZOHo9EU",
  "scope": "openid profile access:ea access:devel
access:admin"
}
```

VERIFY SIGNATURE

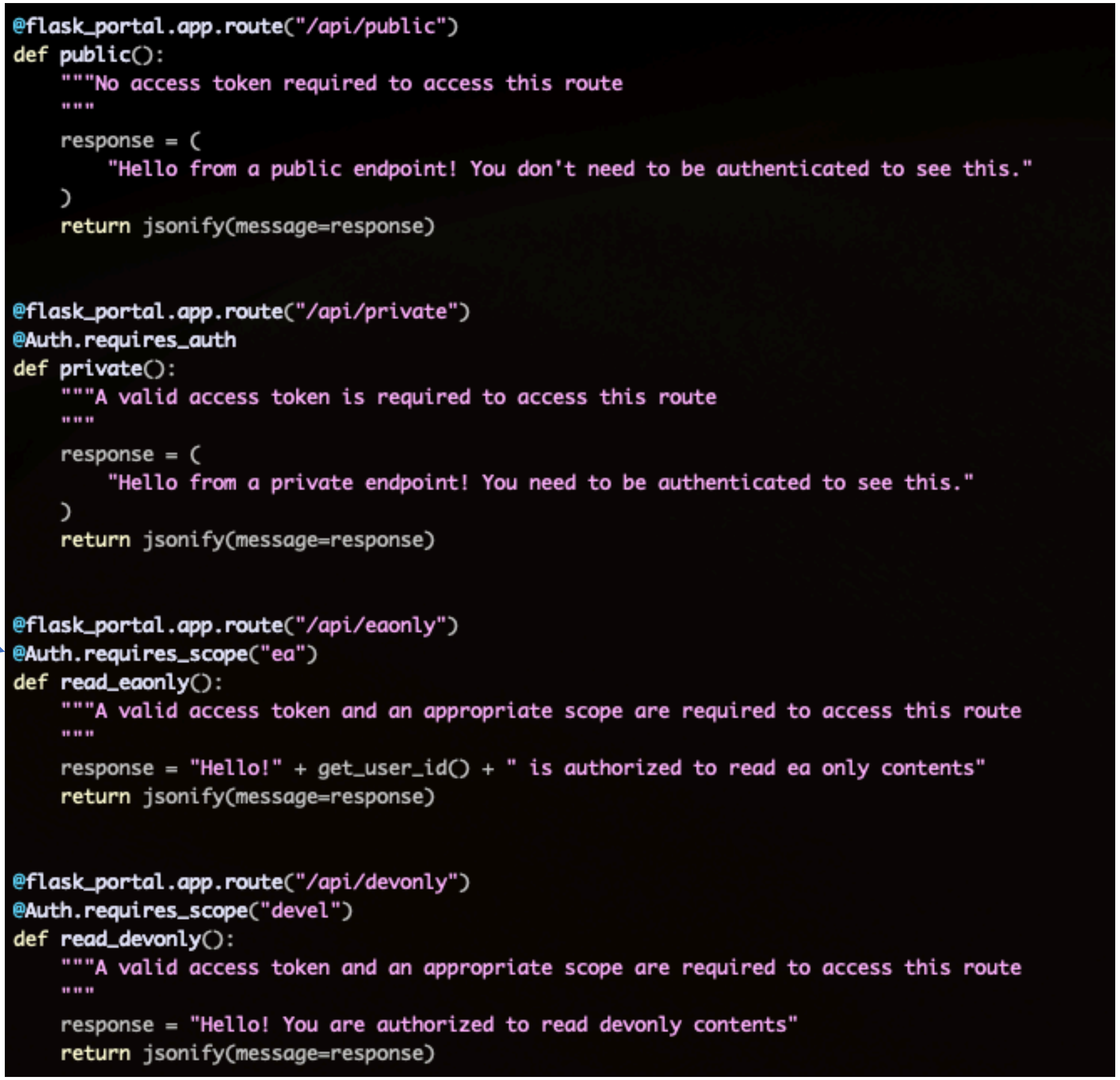
```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  -----BEGIN PUBLIC KEY-----
  MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ
```

Logged in with a ”super” user (admin, devel, ea, stable access)

(Note: ”stable” scope is not explicitly stated. Everyone with login already has this access)

JWT in Flask

- Using Auth0's Authorization Extension, access-level groups were created : admin > devel > ea
- Lower-level access scopes are automatically added to jwt
- Just add "@requires_scope("xxx") after @route(/endpoint)
- @requires_auth is implicitly checked by @requires_scope



```python
@flask_portal.app.route("/api/public")
def public():
    """No access token required to access this route
    """
    response = (
        "Hello from a public endpoint! You don't need to be authenticated to see this."
    )
    return jsonify(message=response)


@flask_portal.app.route("/api/private")
@Auth.requires_auth
def private():
    """A valid access token is required to access this route
    """
    response = (
        "Hello from a private endpoint! You need to be authenticated to see this."
    )
    return jsonify(message=response)


@flask_portal.app.route("/api/eaonly")
@Auth.requires_scope("ea")
def read_eaonly():
    """A valid access token and an appropriate scope are required to access this route
    """
    response = "Hello!" + get_user_id() + " is authorized to read ea only contents"
    return jsonify(message=response)


@flask_portal.app.route("/api/devonly")
@Auth.requires_scope("devel")
def read_devonly():
    """A valid access token and an appropriate scope are required to access this route
    """
    response = "Hello! You are authorized to read devonly contents"
    return jsonify(message=response)
```

@requires_scope(xxx) is simple, but…

- Placing this in front of every single endpoint in a product can be tedious.
- What if the product advances to next maturity level? Should we update

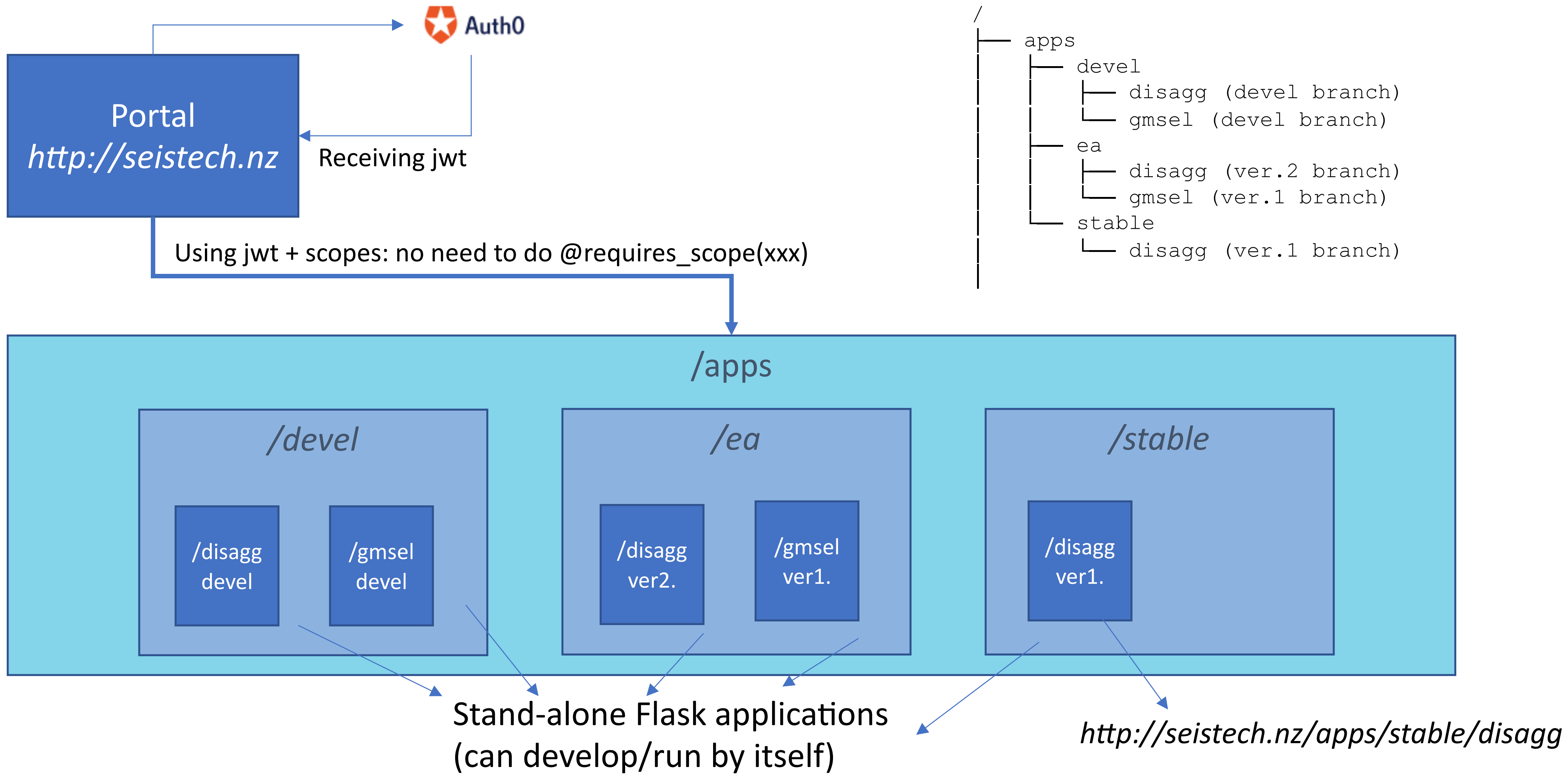
```
@requires_scope(“devel”) → @requires_scope(“ea”)
```

for every endpoint? (of course, we don’t need to hard-code it!)

- What if different versions of one product with different maturity level need to be accessible? eg. Disagg ver.1 is in “stable”, but Disagg ver.2 is in “ea”.

Can we just NOT worry about authentication/access-level control at the product level?

Auth0 Flask Portal


Auth0
Portal
http://seistech.nz
Receiving jwt
Using jwt + scopes: no need to do @requires_scope(xxx)
/apps
/devel
/disagg devel
/gmsel devel
/ea
/disagg ver2.
/gmsel ver1.
/stable
/disagg ver1.
Stand-alone Flask applications (can develop/run by itself)
http://seistech.nz/apps/stable/disagg

```
/
├── apps
│   ├── devel
│   │   ├── disagg (devel branch)
│   │   └── gmsel (devel branch)
│   ├── ea
│   │   ├── disagg (ver.2 branch)
│   │   └── gmsel (ver.1 branch)
│   └── stable
│       └── disagg (ver.1 branch)
│
```

No need to do @requires_scope(xxx) ?? HOW ??

- Suppose we have a product called “test” in “devel” stage

```
from flask import Flask

app = Flask(__name__)


@app.route("/")
@requires_scope(“devel”)
def hello_world():
    return "Hello World from {} :
You have {} permission to view
this page.".format(
        app.import_name,
app.permission
    )


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

```
from authflask import AuthFlask

app = AuthFlask(__name__)


@app.route("/")
def hello_world():
    return "Hello World from {} :
You have {} permission to view
this page.".format(
        app.import_name,
app.permission
    )


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

overriding

```
class AuthFlask(Flask):
   def __init__(self, *args, **kwargs):
...

   def route(self, rule, **options):
      def decorator(f):
         Auth.requires_scope(level)(f)
...
```

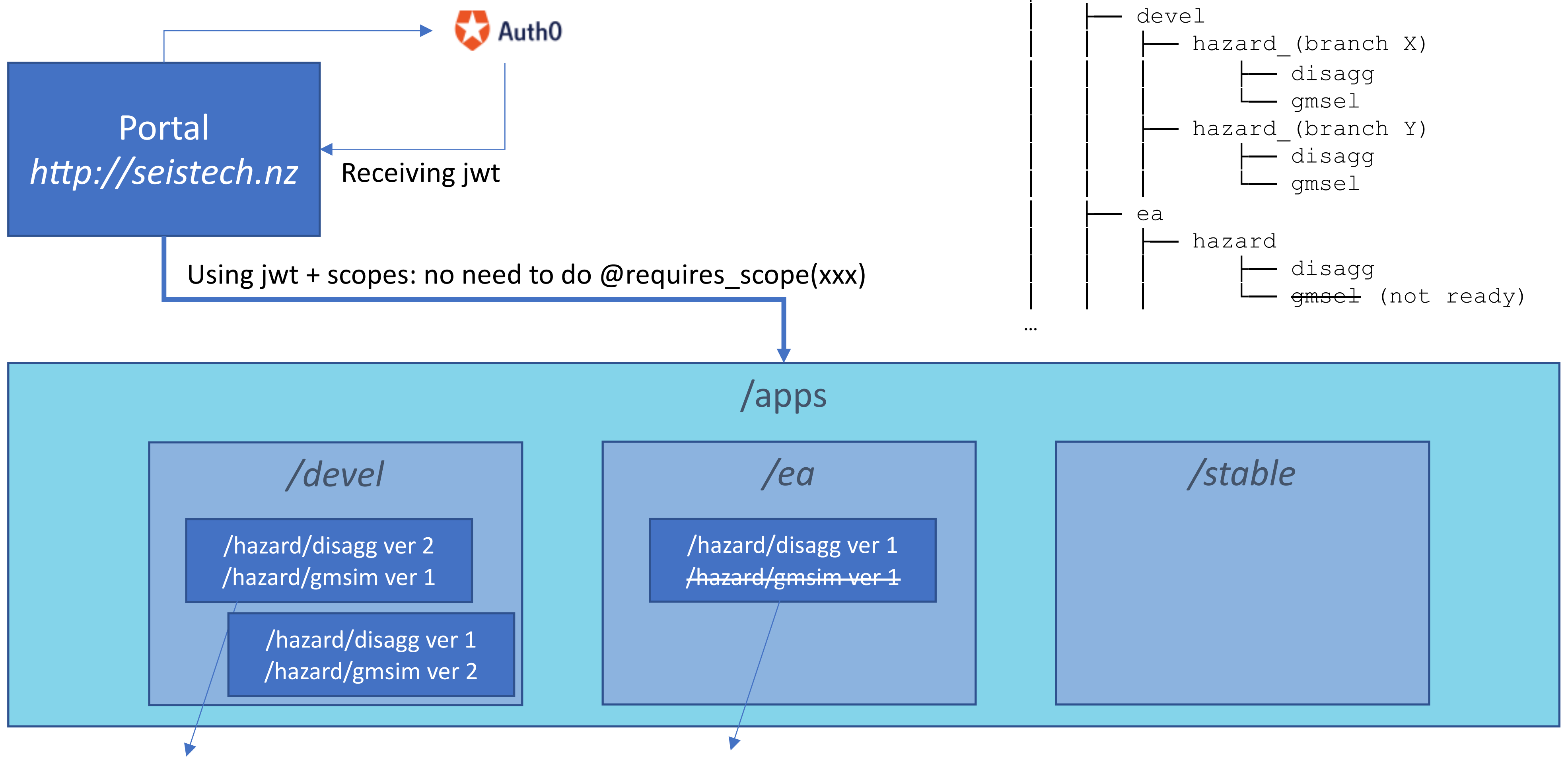
All routes defined in “test” are automatically protected with “devel” access-level just by

1. Placing the code in /apps/devel subdirectory
2. Replacing `Flask()` with `AuthFlask()`

My contribution so far

- Overall architecture : AuthFlask subclass and overriding route(), DispatcherMiddleware
- Injecting the group info (ie. access level) into JWT scope : Despite horrendous documentation with no example code
- Extending Auth0 User DB by connecting to external User table in MariaDB (hosted on EC2) that stays in sync with Auth0 User DB (saves $$$ on Auth0 and Amazon)
- Wrote a proxy layer that can interact with Auth0 management API (will make business logic related to user management very easy to implement)
- Websocket support with Flask DispatcherMiddleware : Open problem in StackOverflow

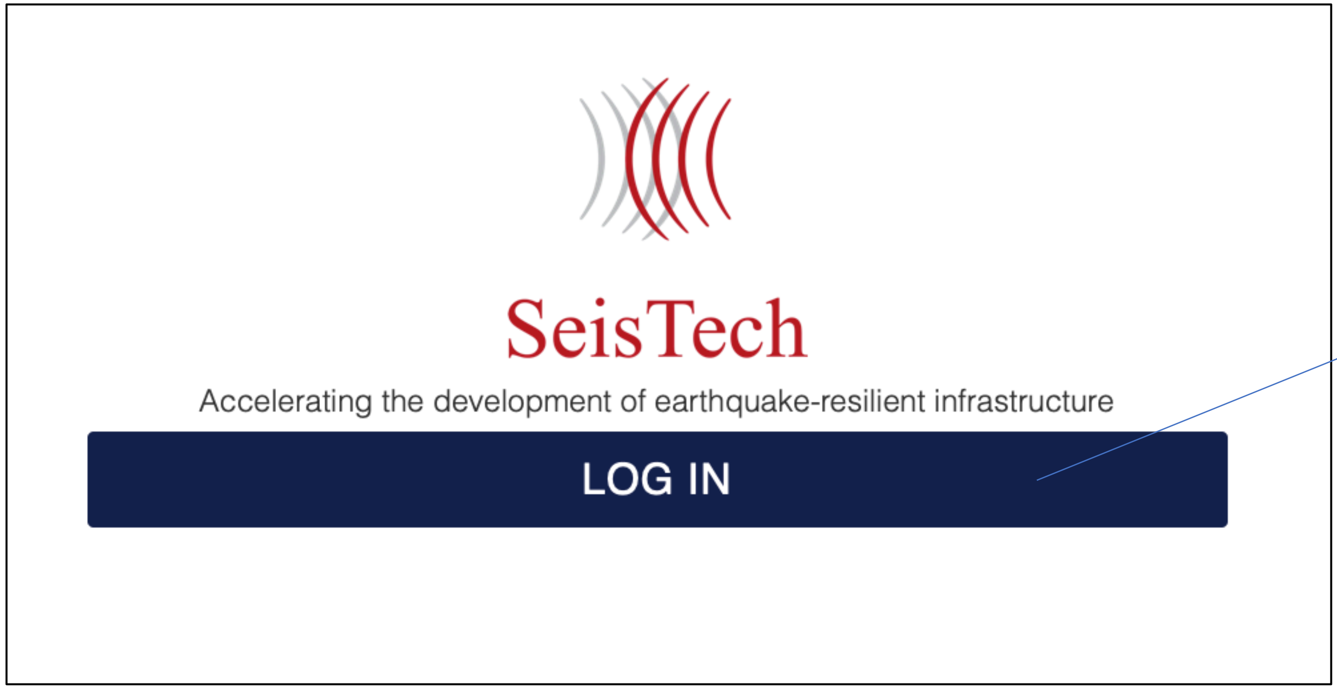
Flexible deployment



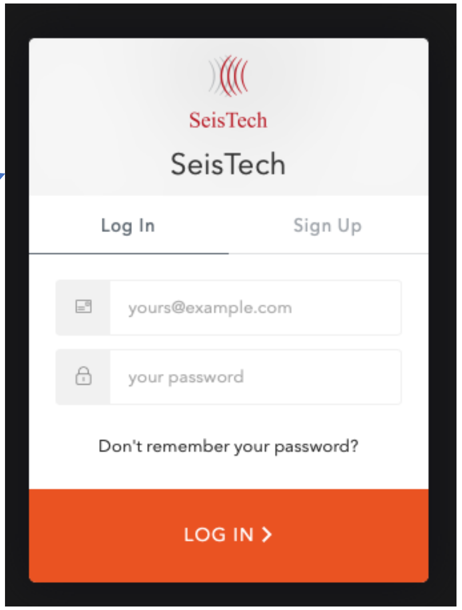
```
/
├── apps
│   ├── devel
│   │   ├── hazard_(branch X)
│   │   │       ├── disagg
│   │   │       └── gmsel
│   │   ├── hazard_(branch Y)
│   │   │       ├── disagg
│   │   │       └── gmsel
│   ├── ea
│   │   ├── hazard
│   │   │       ├── disagg
│   │   │       └── ~~gmsel~~ (not ready)
…
```

Even if products follow monolithic design, can still deploy them separately (can block endpoints if needed)

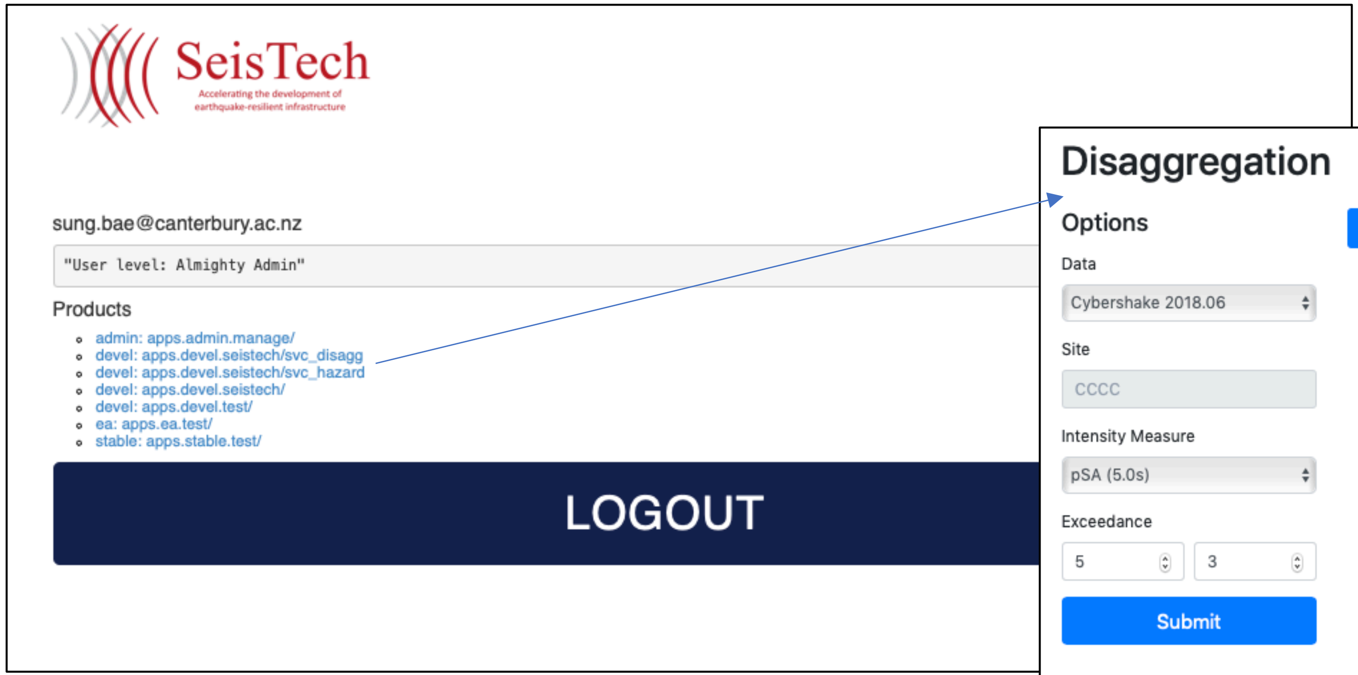
In action


SeisTech

Accelerating the development of earthquake-resilient infrastructure



In action



SeisTech
Accelerating the development of earthquake-resilient infrastructure

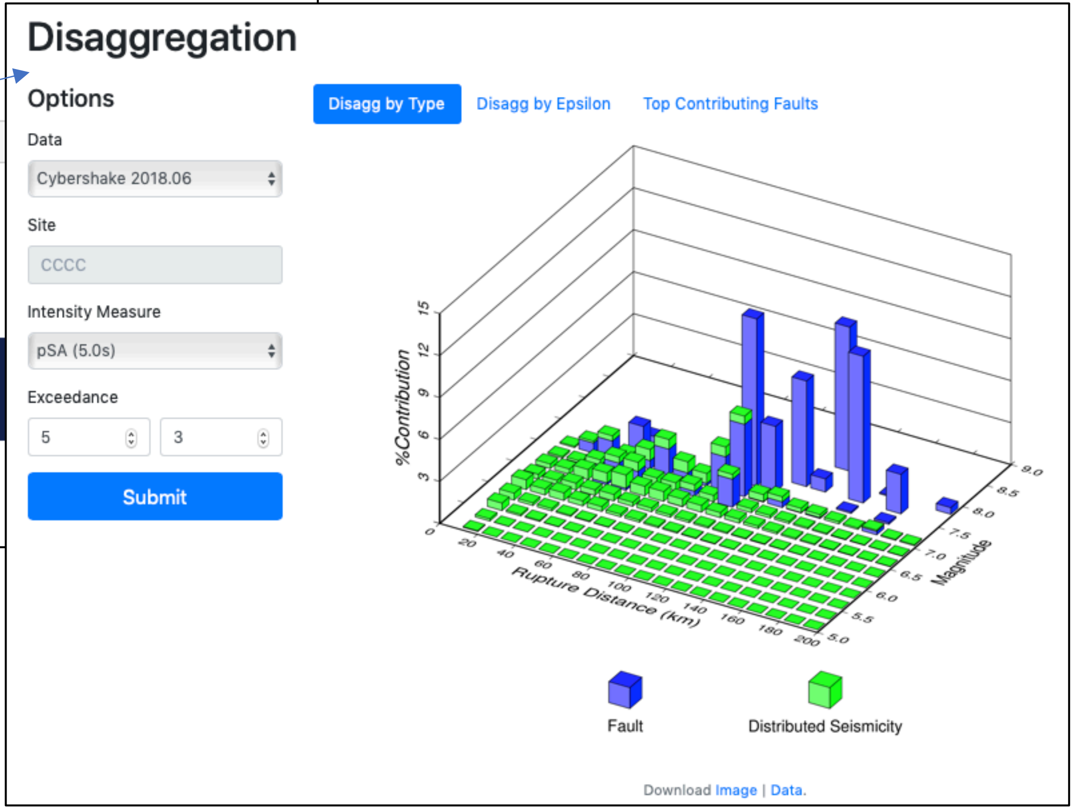
sung.bae@canterbury.ac.nz

"User level: Almighty Admin"

Products

- admin: apps.admin.manage/
- devel: apps.devel.seistech/svc_disagg
- devel: apps.devel.seistech/svc_hazard
- devel: apps.devel.seistech/
- devel: apps.devel.test/
- ea: apps.ea.test/
- stable: apps.stable.test/

LOGOUT



Disaggregation

Options

Data: Cybershake 2018.06

Site: CCCC

Intensity Measure: pSA (5.0s)

Exceedance: 5 | 3

Submit

Download Image | Data.

Objectives

- Authentication : Login-protected contents ✓
- Authorization: Access-level control (Devel, EA, Stable, Admin) ✓
- Login once and give access to all products that the user has clearance ✓
- Should be easy to update the access-level of a product ✓
- Should insulate the product development from tedious business layer: To focus on the functionality of the product ✓
- Should protect from cyber-attacks or unusual activities ✓
- Cybersecurity standards compliance ✓

@requires_scope(xxx) is simple, but…

- Placing this in front of every single endpoint in a product can be tedious.
- What if the product advances to next maturity level? Should we update

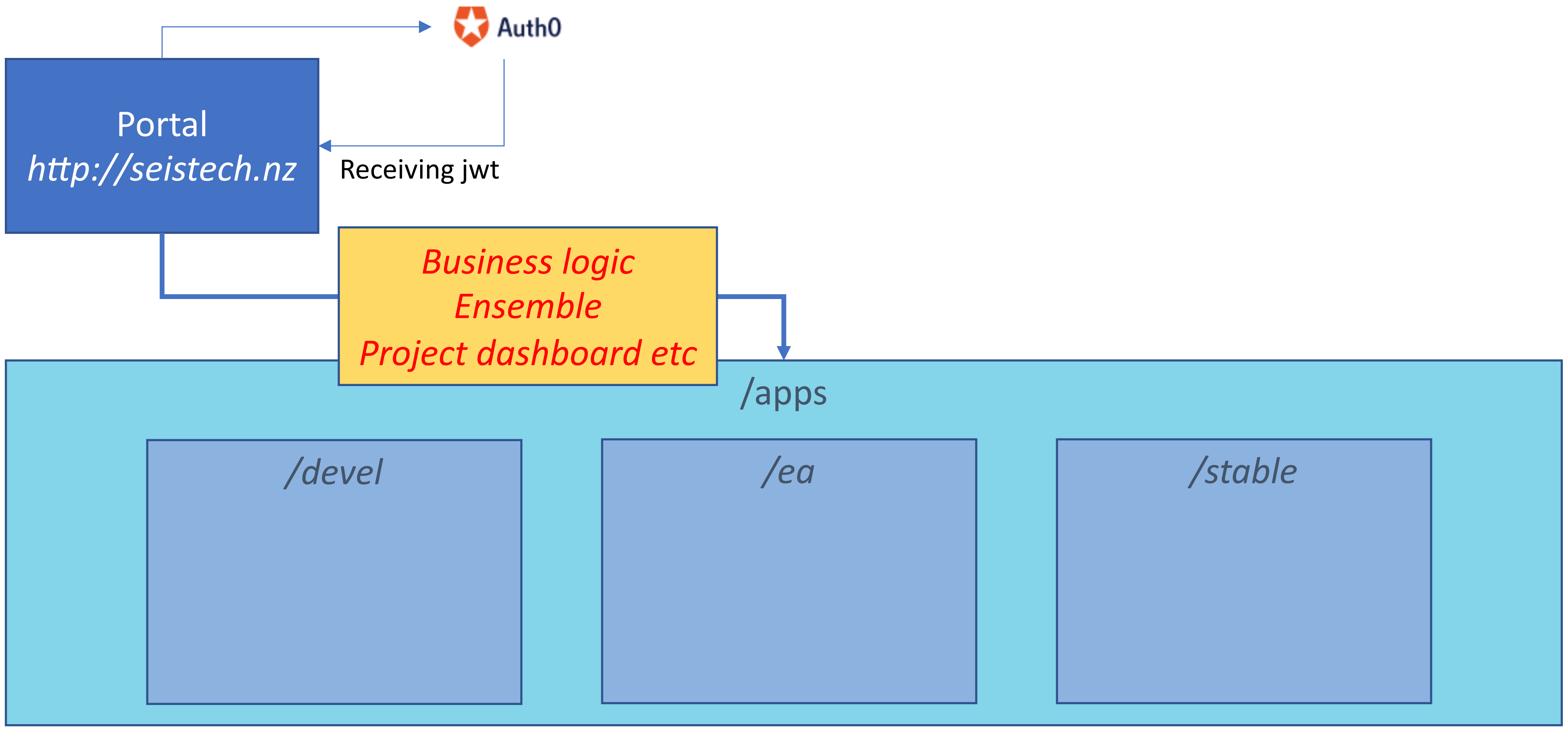
```
@requires_scope(“devel”) → @requires_scope(“ea”)
```

for every endpoint?   (of course, we don’t need to hard-code it!)

- What if different versions of one product with different maturity level need to be accessible? eg. Disagg ver.1 is in “stable”, but Disagg ver.2 is in “ea”.

Can we just NOT worry about authentication/access-level control at the product level? ✓ ✓ ✓

Next step


Auth0
Portal
http://seistech.nz
Receiving jwt
Business logic
Ensemble
Project dashboard etc
/apps
/devel
/ea
/stable