STScI | SPACE TELESCOPE SCIENCE INSTITUTE

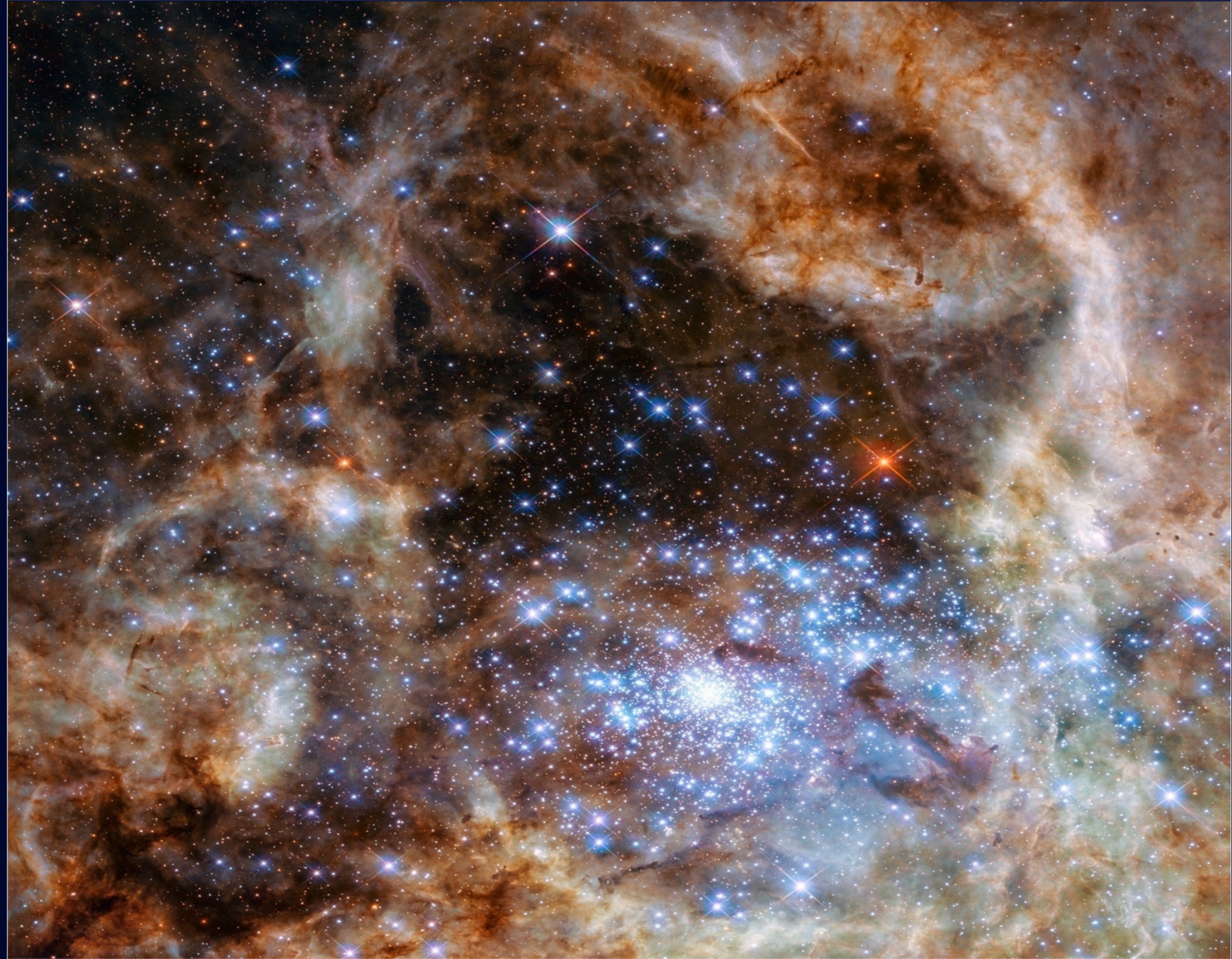
EXPANDING THE FRONTIERS OF SPACE ASTRONOMY

The JWQL Web Application

Lauren Chambers, Matthew Bourque

Last Updated: 05/2020

The JWQL **web application** is built upon Python's **Django** library**,** a **Model-View- Controller web framework**. The front- end is written in **HTML** with embedded **JavaScript** and styled using **Bootstrap** and **CSS**. The web app will interact with databases using an **Object- Relational Mapper**. It runs on a **server** that ITSD set up.



Outline

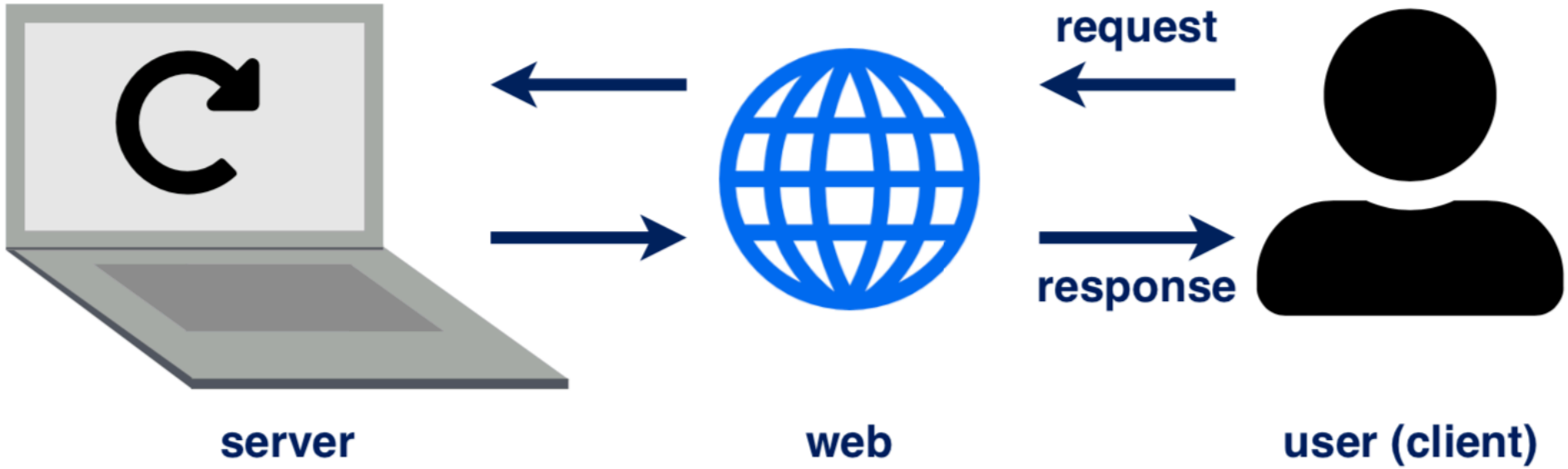
- Web application basics
- Django: A Python web framework
- The JWQL web app
  - Front-end specifics
  - Back-end specifics
  - Directory structure
- How to contribute


Web Application Basics

Web Applications

"In computing, a web application or web app is a client-server computer program which the client (including the user interface and client-side logic) runs a web browser."

- Wikipedia



Web Applications

What is the difference between a “website” and a “web application”?

Some say it’s just a question of semantics.

Some say:

- Website are built to serve content (e.g. news websites)
- Web apps are built to allow user interaction (e.g. gmail)

Web Applications

Front End

- HTML (HypterText Markup Language
- CSS (Cascading Style Sheets)
- JS (JavaScript)

Back End

- HTTP (HyperText Transfer Protocol
- Python
- SQL (Standard Query Language)

HTTP

"at the end of the day, all a web application really does is send HTML to browsers."

- Jeff Knupp, Python guru

HypterText Transfer Protocol, or HTTP, is a protocol for interacting with the web. It uses a request-response protocol.

Methods

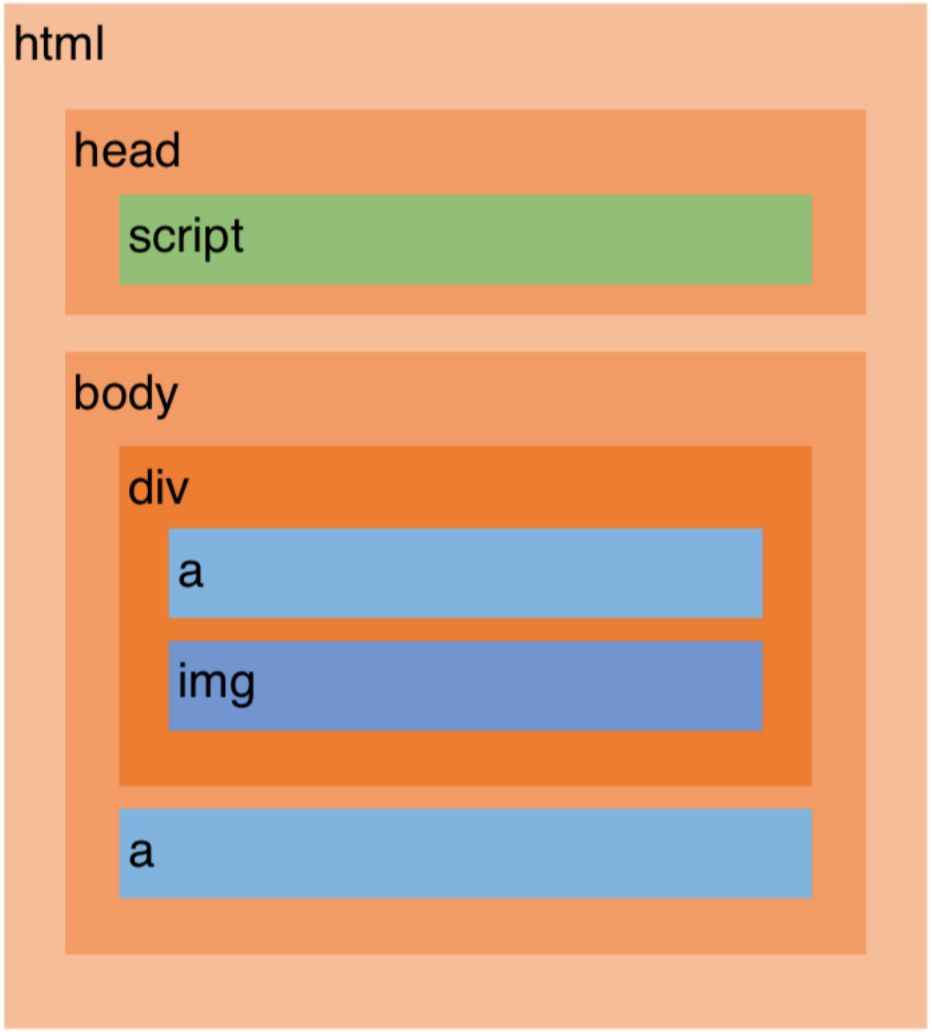
- GET – client requests a webpage's HTML from web server
- POST – client sends data to a webpage through a form, changes the application state

Response Codes

- 200 – everything went fine!
- 404 – everything did not go fine.

HTML

- ”Hypertext Markup Language (HTML) is the standard markup language for creating web pages and web applications.” – Wikipedia
- HTML files are made up of elements (“head”, “main”, ”div”, “h1”, etc.)
- HTML allows embedding of CSS and JavaScript
- Django and Jinja2 expand this to enable HTML “template” syntax
    - Basically include variables in HTML, like in Python: **‘my name is {}’.format(name)**



CSS

- ”Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language like HTML” – Wikipedia
- CSS can control layout, colors, fonts, sizing, spacing, and more
- Think HTML = content, CSS = presentation
- Using CSS files separates the content of a webpage from the definitions of how to present that content
- CSS can also enable “responsive” design, allowing for different presentations in different situations (e.g. desktop versus mobile phone)

CSS

HTML file:

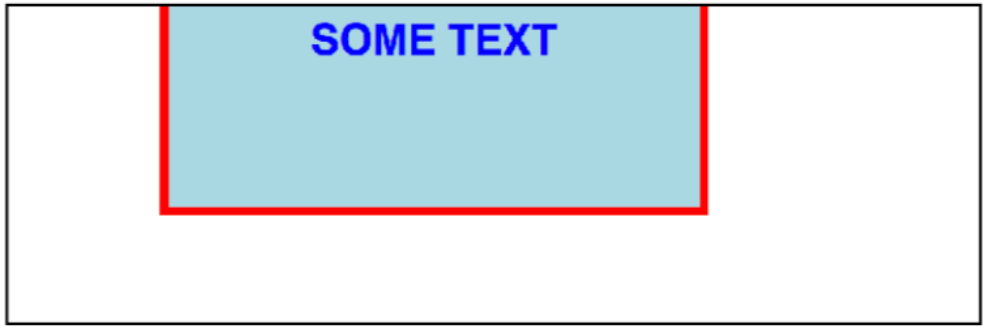
```
<a class='example_name'> Some text</a>
```

CSS file:

```
.example_name {
    color: blue;
    background-color: lightblue;
    margin: 5rem;
    padding: 5rem;
    font-size: 24px;
    font-weight: bold;
    font-family: sans-serif;
    border: 5px solid red;
    text-transform: uppercase
}
```

Before CSS:

After CSS:

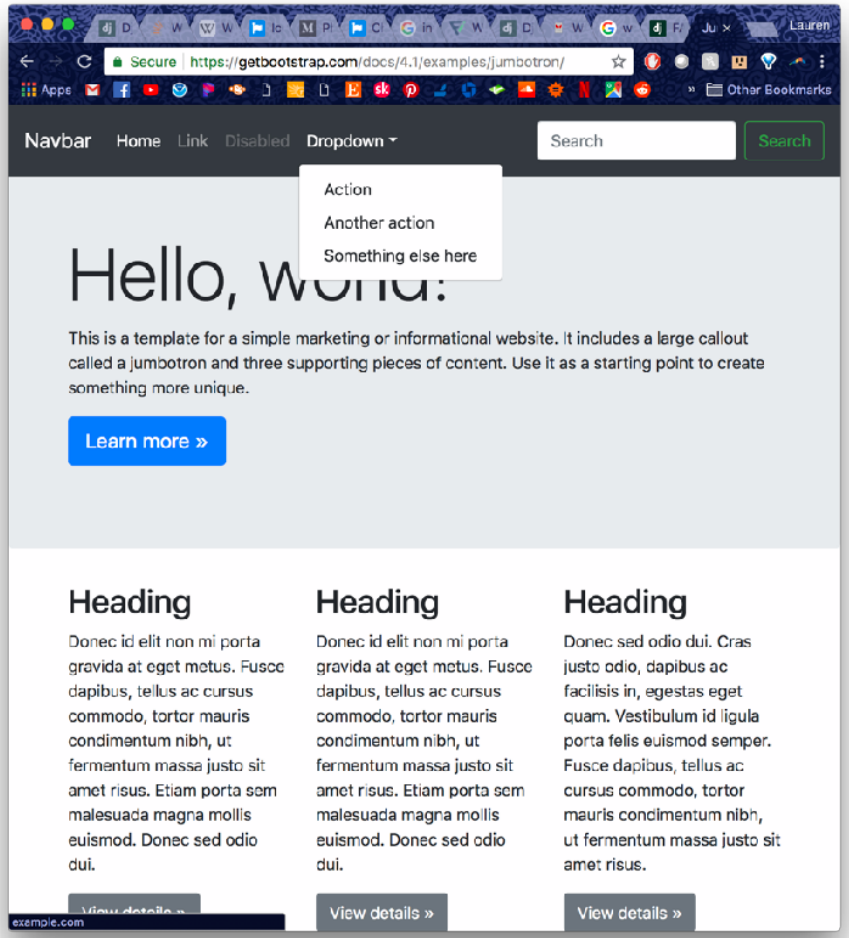


JavaScript

- A programming, not style, language
- NOT the same as Java! Java is to JavaScript as a car is to a carpet
- “JavaScript supports event-driven, functional, and imperative (including object-oriented and prototype-based) programming styles” - Wikipedia
- Allows webpages to be interactive
  - e.g. if you click a button, sort the images by date
  - e.g. As the user types, display only the images whose titles match the search criteria

Bootstrap

- “Bootstrap is a free and open-source front-end framework (library) for designing websites and web applications.” – Wikipedia
- Includes design templates that use HTML, CSS, and JS
- Default styles for things like:
  - Navigation bars
  - Buttons & dropdown menus
  - Grid layouts
  - Headings



Web Frameworks

- Turns out, developing web applications are hard.
  - Handling POST requests securely
  - Sessions? Cookies?
  - Handling many concurrent connections
  - Accessing databases
  - Dynamically generating new HTML
  - Routing (mapping URL to content/pages)
- ”A web framework … is a software framework that is designed to support the development of web applications including web services, web resources, and web APIs.” – Wikipedia
- A web framework is a scaffold upon which to build a web application.

Django

Python's Django Library

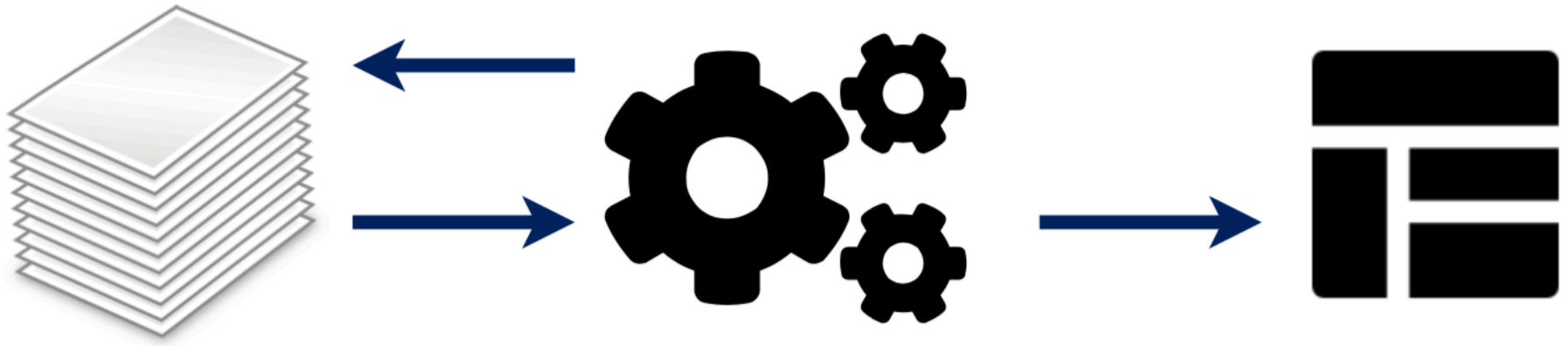
- Model-View-Controller web framework
- Python-based
- Uses regular expressions to map URLs to particular functions/pages (routing)
- Uses a Object Relational Mapper (ORM) to map Python classes to database tables
- Dynamically serves content (e.g. query a databases to get latest values)

Django's Model-View-Controller Pattern

model:
database

controller:
Python function

view:
HTML template



Controllers with Django

```
from django.shortcuts import render

def view_header(request, inst, file):
    template = 'view_header.html'
    header = get_header_info(file)
    file_root = '_'.join(file.split('_')[:-1])

    return render(request, template,
                  {'inst': inst,
                   'file': file,
                   'header': header,
                   'file_root': file_root})
```

Views (Templates) with Django

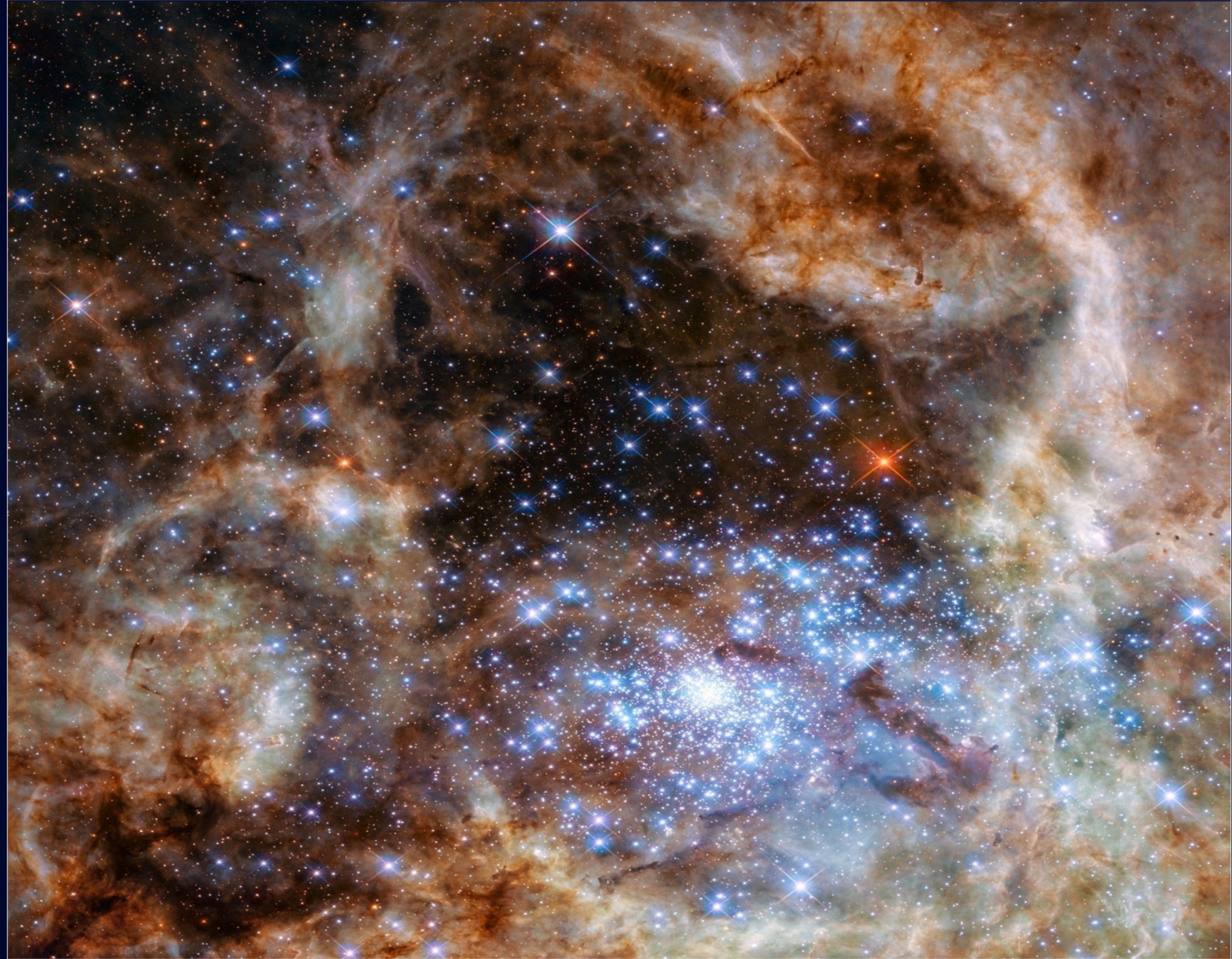
```
{% extends "base.html" %}

{% block content %}

   <h5>{{ file }}</h5>

    <p><a class="btn btn-primary" href={{ url('jwql:view_image', args=[inst, file_root]) }}>View
        Image</a></p>

    {% autoescape false %}
         <font face="Courier">{{ header | replace("\n", "<br/>") }}</font>
    {% endautoescape %}

    <p>
         <a class="btn btn-primary" href='{{ static("") }}{{ file[:7] }}/{{ file }}'>Download
            FITS</a>
         <a class="btn btn-primary" href='{{ static("") }}{{ file[:7] }}/{{ jpg }}'>Download
            JPEG</a>
    </p>

{% endblock %}
```

URLs with Django – Regular Expressions

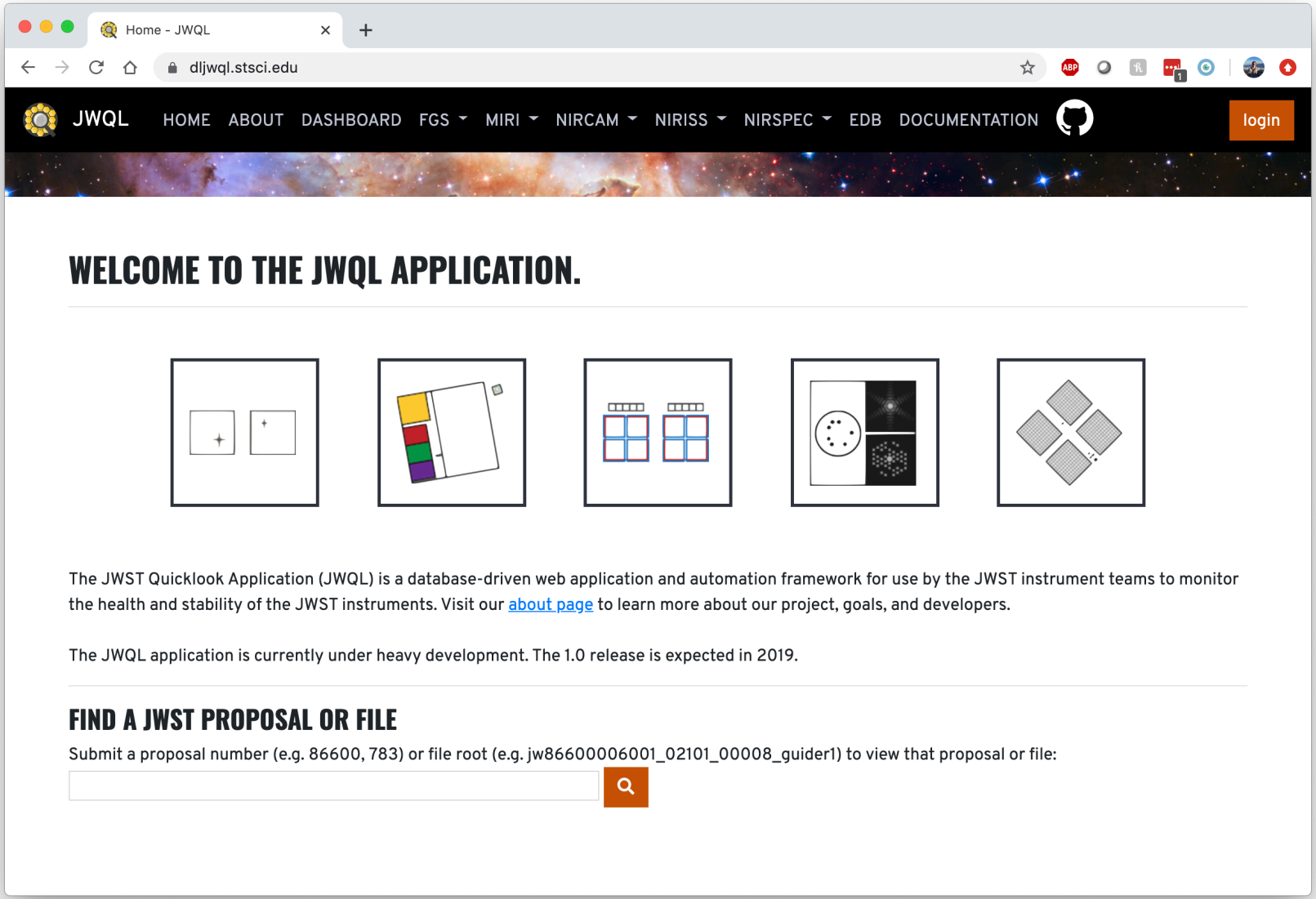
```
from django.urls import path

from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('about/', views.about, name='about'),
    path('dashboard/', views.dashboard, name='dashboard'),
    path('<str:inst>/', views.instrument, name='instrument'),
    path('<str:inst>/archive/', views.archived_proposals, name='archive'),
    path('<str:inst>/unlooked/', views.unlooked_images, name='unlooked'),
    path('<str:inst>/<str:file_root>/', views.view_image, name='view_image'),
    path('<str:inst>/<str:file>/hdr/', views.view_header,
name='view_header'),
    path('<str:inst>/archive/<str:proposal>', views.archive_thumbnails,
name='archive_thumb'),
]
```


The JWQL web app

The JWQL **web application** is built upon Python's **Django** library**,** a **Model-View- Controller web framework**. The front- end is written in **HTML** with embedded **JavaScript** and styled using **Bootstrap** and **CSS**. The web app will interact with databases using an **Object- Relational Mapper**. It runs on a **server** that ITSD set up.



Front End: Current Home Page



Front End: JWQL Web Style Guide

TYPOGRAPHY

H1 - PAGE HEADING

Oswald Bold; 2.5rem size; 0.05em spacing; all caps

H2 - SECTION HEADING

Oswald Bold; 2rem size; all caps

H3 - SUBHEADING

Oswald Bold; 1.75rem size; all caps

H4 - SUBHEADING

Oswald Bold; 1.5rem size; all caps

H5 - Subheading

Oswald Bold; 1.25rem size

H6 - Subheading

Oswald Bold; 1rem size

TEXT HEADINGS

Overpass; all caps

Body Text

Overpass

BUTTONS

Primary | Disabled | Outline

COLOR PALETTE

#F2CE3A	#000000	#BEC4D4
#2D353C	#C85108	#4C8BD8

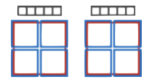
ICONOGRAPHY

Favicon/Brand

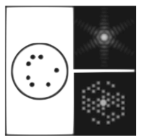
STScI Logo

FGS

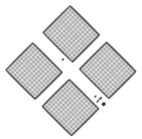
MIRI



NIRCam



NIRISS



NIRSpec

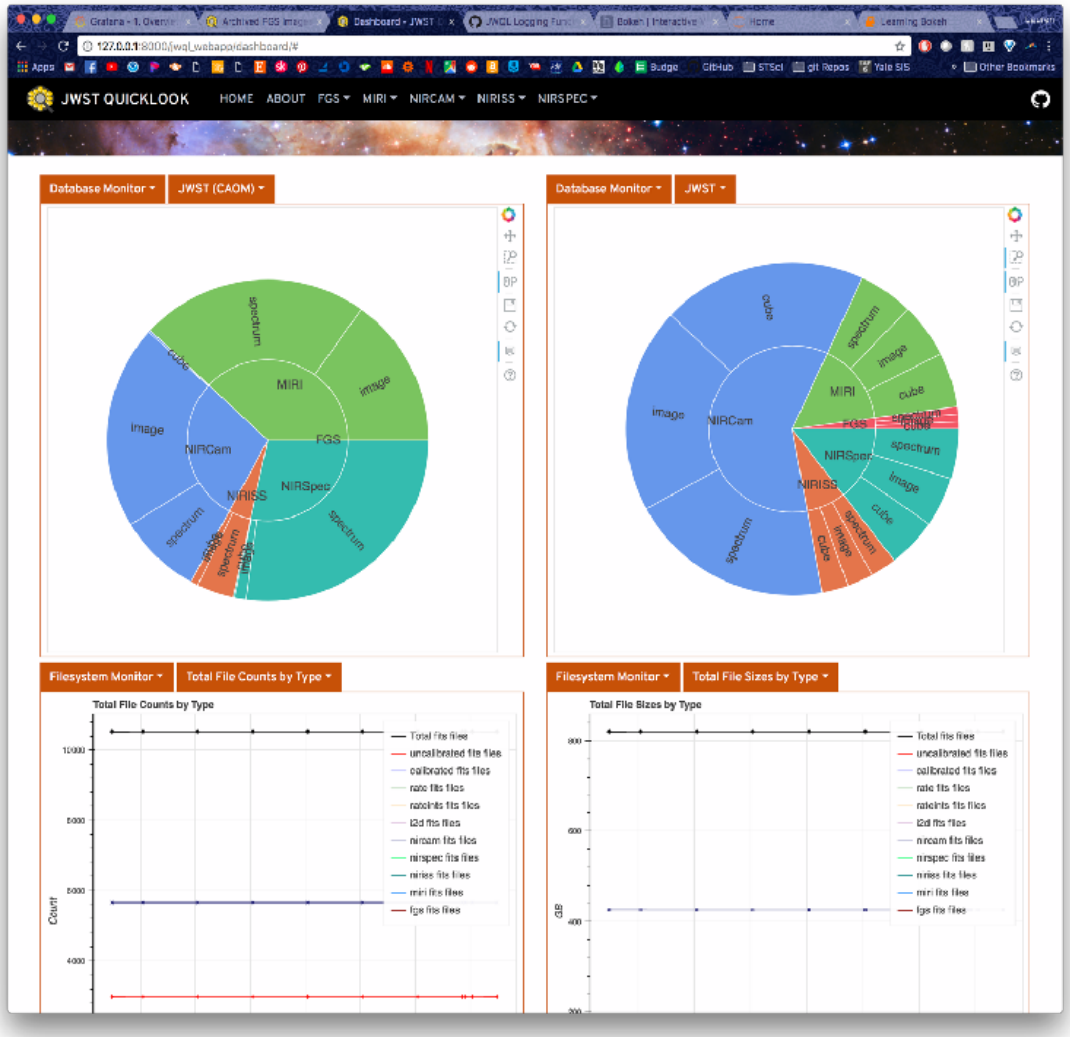
Back End: Views (Templates) with Jinja2

- Jinja2 is another template syntax, for the most part with similar functionality as Django templates
  - Django: {% url 'jwql_webapp:home' %}
  - Jinja2: {{ url('jwql_webapp:home') }}
- BUT it might be easier to embed Bokeh plots in templates

```
<div class="col-md-4 col-sm-4 col-xs-12">
  <div class="x_panel">
    <div class="x_title">
      <h2>By Region</h2>
      <div class="clearfix"></div>
    </div>
    {{ embed(roots.region) }}
  </div>
</div>
```

Back End: Embedding plots with Bokeh

- “Bokeh is an interactive visualization library that targets modern web browsers for presentation” – Bokeh
- It allows for users to display and interact with data in a web page (including built-in features like pan, zoom, export to file, etc.)
- Will be used for dashboard and instrument monitoring plots in JWQL



JWQL Web App Directory Structure

```
website/
    manage.py
    jwql_proj/
        settings.py
    apps/
        jwql/
            oauth.py
            urls.py
            context_processors.py
            views.py
            api_views.py
            monitor_views.py
            data_containers.py
            bokeh_containers.py
            templates/
                base.html
                home.html
                about.html
                view_image.html
                ...
            static/
                css/
                    bootstrap.min.css
                    jwql.css
                js/
                    jwql.js
                img/
                    jwql_logo.png
                    nircam_logo.png
                    ...
```

JWQL Web App Directory Structure

- **`website/`** - Main website directory
- **`manage.py`** – Django-generated file that performs operations such as running a local server, starting an interactive ipython shell, or running tests
- **`jwql_proj/`** - directory containing project-specific files
- **`settings.py`** – Defines project settings, such as debug mode, allowed hosts, which HTML templating language is being used, etc.
- **`apps/`** - directory containing different ”applications”, kind of like reusable Python packages.
- **`jwql/`** - directory for the jwql web app

```
website/
    manage.py
    jwql_proj/
        settings.py
    apps/
        jwql/
            oauth.py
            urls.py
            context_processors.py
            views.py
            api_views.py
            monitor_views.py
            data_containers.py
            bokeh_containers.py
            templates/
                base.html
                home.html
                about.html
                view_image.html
                ...
            static/
                css/
                    bootstrap.min.css
                    jwql.css
                js/
                    jwql.js
                img/
                    jwql_logo.png
                    nircam_logo.png
                    ...
```

JWQL Web App Directory Structure

- **`oauth.py`** – Provides functions for handling user authentication with **`auth.mast`**
- **`urls.py`** – Defines URL-to-view mappings
- **`context_processors.py`** – Provides functions for defining data inherent to all views
- **`views.py`** – Defines the views (webpages) of the web app
- **`api_views.py`** – Defines the views (webpages) of the web app, but for the jwql API specifically
- **`monitor_views.py`** – Defines the views (webpages) of the web app, but for instrument monitors specifically
- **`data_containers.py`** – Returns data needed for views
- **`bokeh_containers.py`** – Returns data need for monitors

```
website/
    manage.py
    jwql_proj/
        settings.py
    apps/
        jwql/
            oauth.py
            urls.py
            context_processors.py
            views.py
            api_views.py
            monitor_views.py
            data_containers.py
            bokeh_containers.py
            templates/
                base.html
                home.html
                about.html
                view_image.html
                ...
            static/
                css/
                    bootstrap.min.css
                    jwql.css
                js/
                    jwql.js
                img/
                    jwql_logo.png
                    nircam_logo.png
                    ...
```

JWQL Web App Directory Structure

- `templates/` – Stores the HTML templates for the various views of the web app.

```
website/
    manage.py
    jwql_proj/
        settings.py
    apps/
        jwql/
            oauth.py
            urls.py
            context_processors.py
            views.py
            api_views.py
            monitor_views.py
            data_containers.py
            bokeh_containers.py
            templates/
                base.html
                home.html
                about.html
                view_image.html
                ...
            static/
                css/
                    bootstrap.min.css
                    jwql.css
                js/
                    jwql.js
                img/
                    jwql_logo.png
                    nircam_logo.png
                    ...
```

JWQL Web App Directory Structure

- **`static/`** – Stores various static data needed to render webpages
- **`css/`** – Stores CSS files
- **`bootstrap.min.css`** – Contains CSS elements provided by **`bootstrap`**
- **`jwql.css -`** Contains custom-built CSS elements specific to the project
- **`js/`** – Stores files containing needed Javascript functions
- **`jwql.js`** – Contains custom-built JS functions specific to the project
- **`img`** – Contains static images, gifs, logos, etc.

```
website/
    manage.py
    jwql_proj/
        settings.py
    apps/
        jwql/
            oauth.py
            urls.py
            context_processors.py
            views.py
            api_views.py
            monitor_views.py
            data_containers.py
            bokeh_containers.py
            templates/
                base.html
                home.html
                about.html
                view_image.html
                ...
            static/
                css/
                    bootstrap.min.css
                    jwql.css
                js/
                    jwql.js
                img/
                    jwql_logo.png
                    nircam_logo.png
                    ...
```


How do *you* contribute?

How to run the web app on a local server

- `cd` into the `website/` directory
- Launch the server: `python manage.py runserver`
- Open the web app in a browser: http://127.0.0.1:8000/

How to add a webpage to the web application

- Check out https://github.com/spacetelescope/jwql/blob/webpage-template/WEBPAGE_TEMPLATE.md

Helpful Resources

- Blog on web frameworks: https://jeffknupp.com/blog/2014/03/03/what-is-a-web-framework/
- Documentation and examples for HTML, CSS, JS: https://www.w3schools.com/
- Sandbox for writing HTML alongside CSS and JS: https://jsfiddle.net/
- Bootstrap documentation: https://getbootstrap.com/docs/4.1/getting-started/introduction/
- Django 7-Part Tutorial: https://docs.djangoproject.com/en/2.1/intro/tutorial01/