

STScI | SPACE TELESCOPE
SCIENCE INSTITUTE

EXPANDING THE FRONTIERS OF SPACE ASTRONOMY

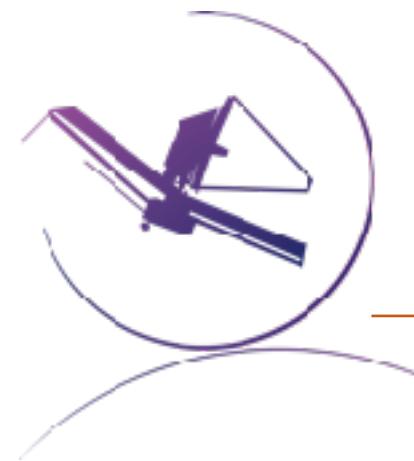
The JWQL Web Application

Lauren Chambers & Matthew Bourque

August 29th, 2018

The JWQL **web application** is built upon Python **Django**, 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 will run on a **server** that ITSD will set up.





Outline

- Web application basics
- Django: a Python web framework
- The JWQL web app
 - Front end specifics
 - Back end specifics
 - Directory structure
 - Mini-code review: thumbnails

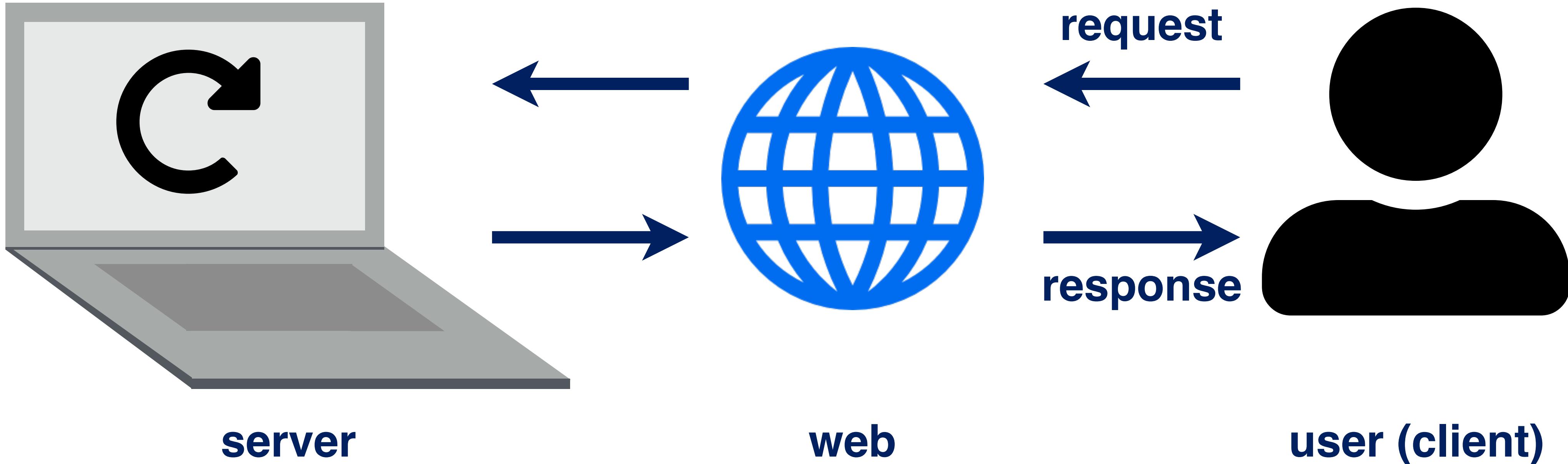
Web Application Basics

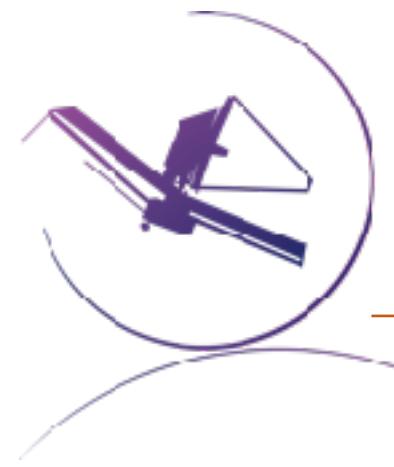


Web applications

“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 in a web browser.”

- Wikipedia





Web applications

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

Some say it's just a question of semantics.

Some say:

- Websites 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
 - CSS
 - JS

Back End

- HTTP
 - Python
 - SQL





HTTP

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

- Jeff Knupp, Python guru

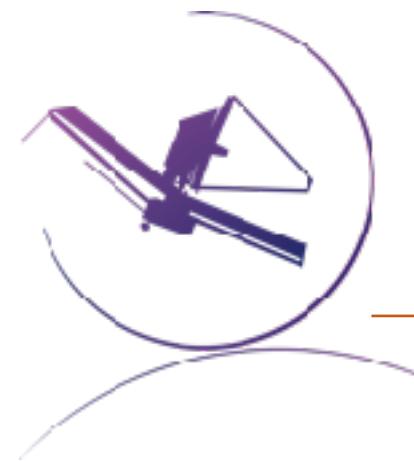
Hypertext Transfer Protocol, or HTTP, is a protocol for interacting with the web. It uses a request-response protocol.

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

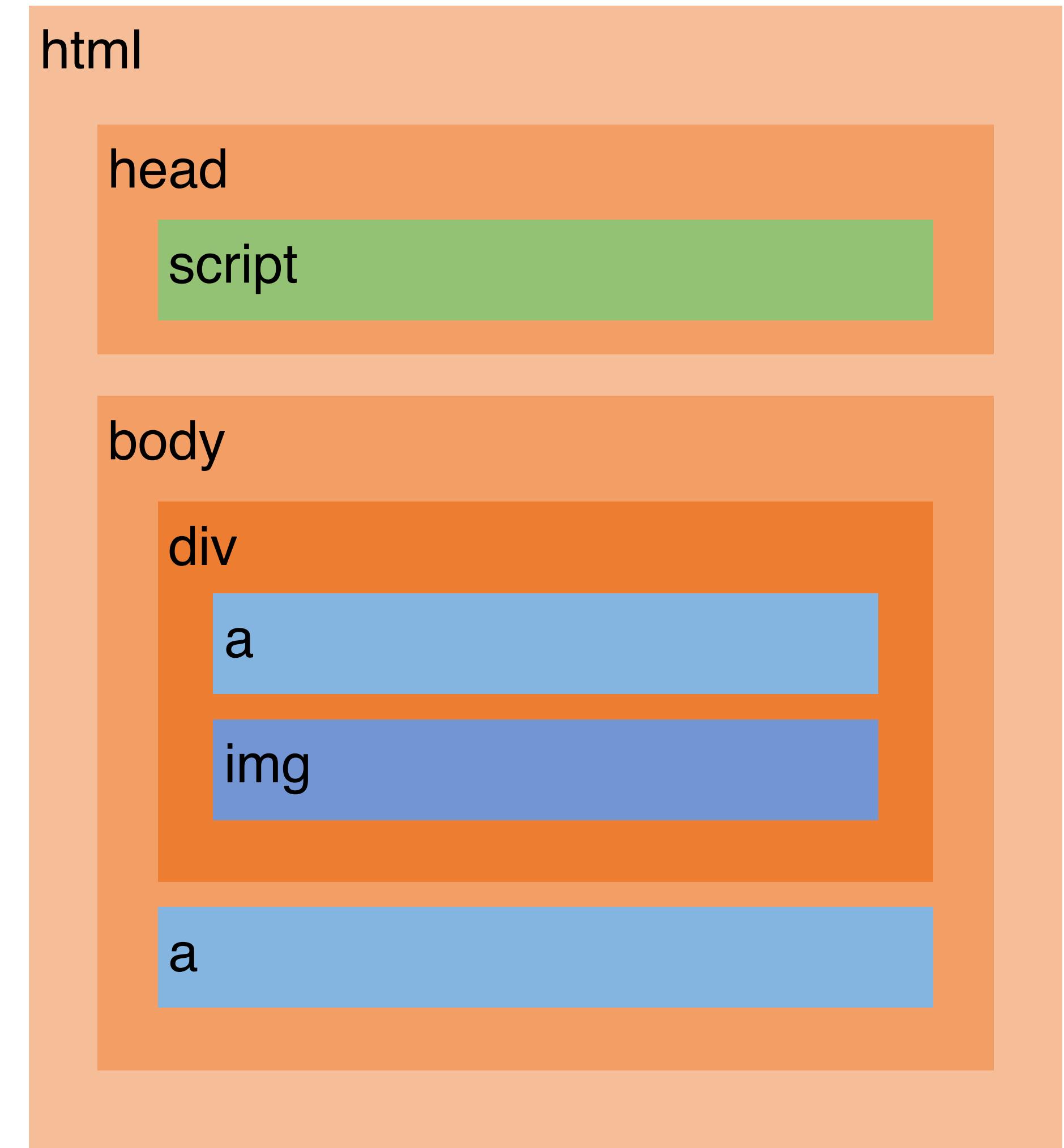
Response Codes

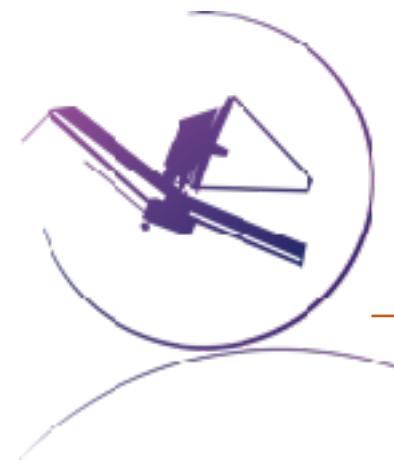
- 200 - everything went fine!
- 404 - everything did not go fine.



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.)
- Normally, HTML just 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 string {}’.format(string_number)





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
- 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 (i.e. desktop versus mobile phone)



CSS

HTML file:

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

CSS file:

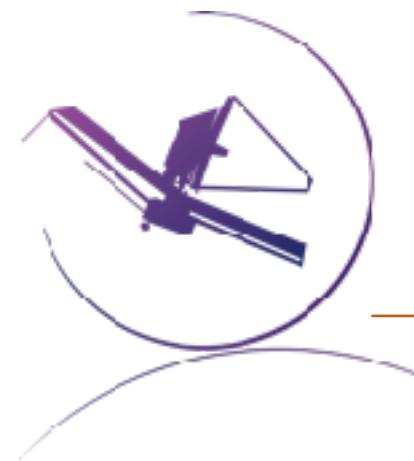
```
.example_name {  
    color: blue;  
    background-color: lightblue;  
    margin: 5rem;  
    padding: 5rem;  
    font-size: 24px;  
    font-weight: bold;  
    font-family: sans-serif;  
    border: 5px solid red;  
    text-transform: uppercase  
}
```

Before CSS:

Some text

After CSS:

SOME TEXT



JavaScript

- A programming, not style, language
- NOT the same as Java!
- “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
- Syntactically obnoxious



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
 - Enables responsive design

The screenshot shows a web browser displaying a Bootstrap template example. The page has a large header with the text "Hello, world!". Below the header is a callout box containing three items: "Action", "Another action", and "Something else here". The main content area features three columns, each with a heading and some placeholder text. The first column's heading is "Heading" and its text is a long Lorem ipsum paragraph. The second column's heading is "Heading" and its text is another long Lorem ipsum paragraph. The third column's heading is "Heading" and its text is a third long Lorem ipsum paragraph. Each column has a "View details »" button at the bottom.

Secure | https://getbootstrap.com/docs/4.1/examples/jumbotron/ Lauren

Navbar Home Link Disabled Dropdown Search

Action
Another action
Something else here

Hello, world!

This is a template for a simple marketing or informational website. It includes a large callout called a jumbotron and three supporting pieces of content. Use it as a starting point to create something more unique.

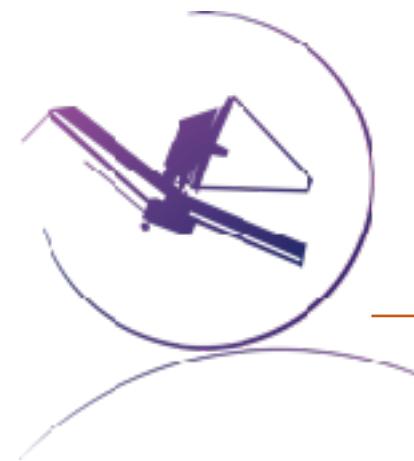
Learn more »

Heading
Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

Heading
Donec id elit non mi porta gravida at eget metus. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus. Etiam porta sem malesuada magna mollis euismod. Donec sed odio dui.

Heading
Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Vestibulum id ligula porta felis euismod semper. Fusce dapibus, tellus ac cursus commodo, tortor mauris condimentum nibh, ut fermentum massa justo sit amet risus.

example.com View details » View details » View details »



Web Frameworks

- Turns out, developing web applications is 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 (WF)... 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 Django

- 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 a database
- Dynamically serving content (querying a database to get latest values) using HTML templates
- Provides an “admin” interface for developers/maintainers to interact with “models” (data)

django

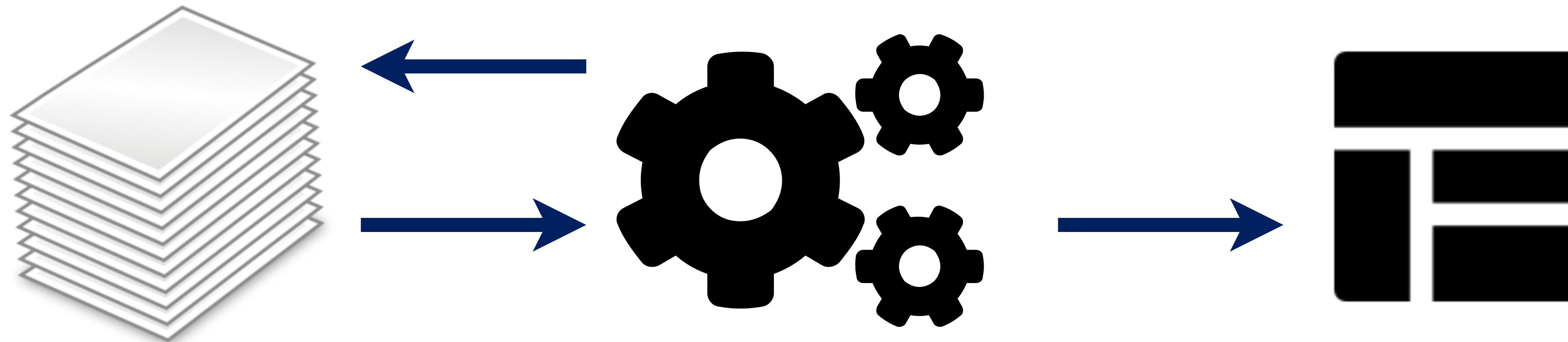


Django's Model-View-Controller Pattern

model:
database

controller:
Python function

view:
HTML template





Models with Django - Object-Relational Mapper

```
from django.db import models

class ImageData(BaseModel):
    """A model that collects image file paths, instrument labels, and
    publishing date/time. Just an example used for learning Django.
    """

    # Create columns for the instrument, publication date, and filepath
    inst = models.CharField('instrument', max_length=6,
                           choices=INSTRUMENT_LIST, default=None)
    pub_date = models.DateTimeField('date published')
    filepath = models.FilePathField(path='/user/lchambers/jwql/')

    def filename(self):
        return os.path.basename(self.filepath)
```

(Not currently being used in the web app, but it will be once we integrate Joe's anomaly database.)



Controllers (Views) 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

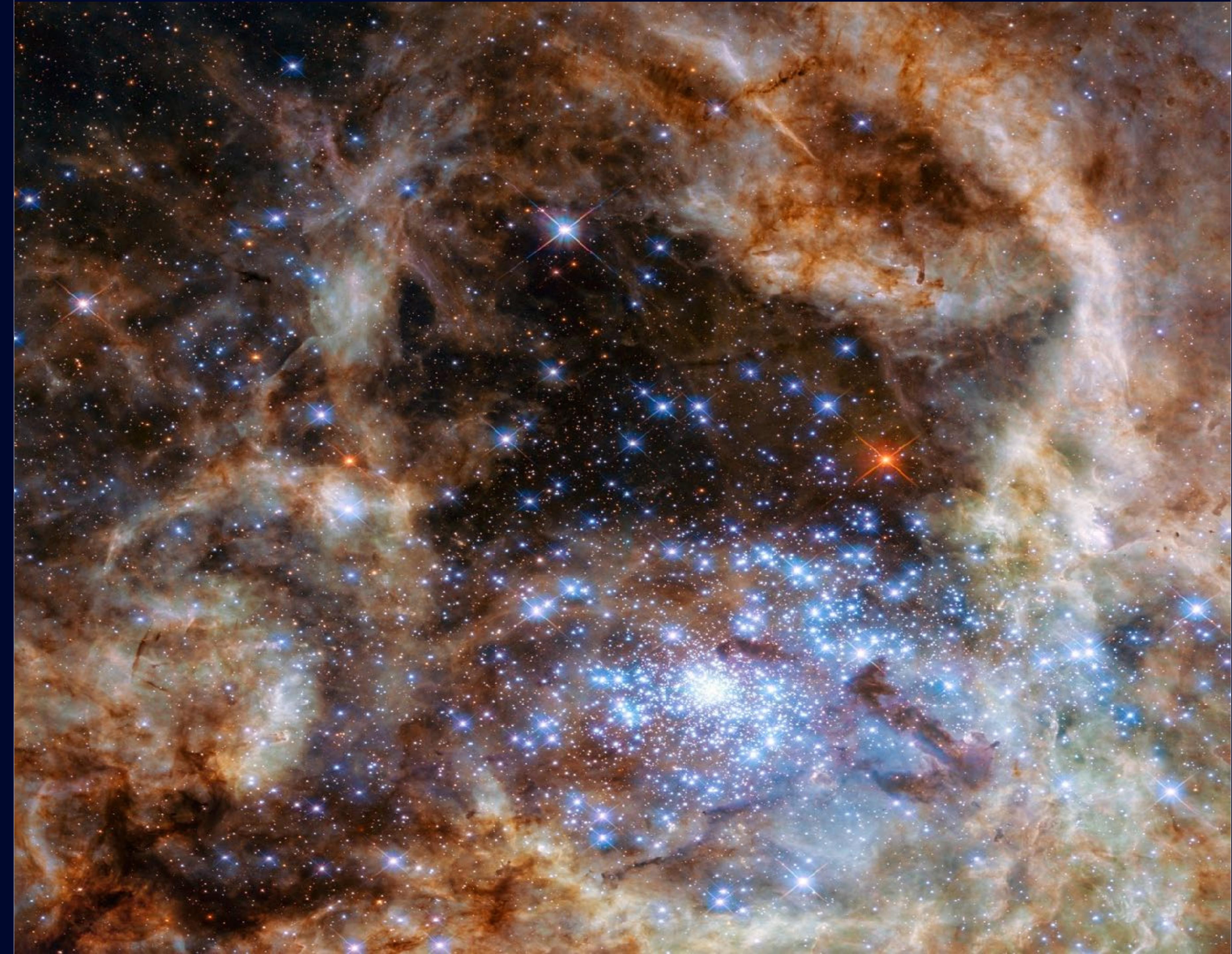
from . import views

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 **Django**, 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 will run on a **server** that ITSD will set up.





Front End: Current Home Page

The screenshot shows a web browser window titled "Home - JWST Quicklook". The address bar displays "127.0.0.1:8000/jwql/". The bookmarks bar includes links to various sites like Apps, GitHub, and STScI. The main header features the "JWST QUICKLOOK" logo and navigation links for HOME, ABOUT, DASHBOARD, FGS, MIRI, NIRCAM, NIRISS, and NIRSPEC. Below the header is a decorative banner with a space-themed image. The main content area has a dark background with the text "WELCOME TO THE JWST QUICKLOOK PAGE." in large, bold, white letters. It features five square icons representing different instrument components: a central square with two smaller squares containing plus signs, a folded multi-colored panel, a grid of four smaller squares, a circular aperture with a star, and a cross-shaped detector array. Below these icons are two orange buttons labeled "Dashboard" and "Query Database". A descriptive paragraph explains the purpose of the application, mentioning it is a database-driven web application and automation framework for monitoring JWST instruments. It also notes the 1.0 release is expected in 2019. The footer contains the STScI logo and the text "SPACE TELESCOPE SCIENCE INSTITUTE" along with the JWST logo.

Home - JWST Quicklook

127.0.0.1:8000/jwql/

Apps GitHub STScI

JWST QUICKLOOK HOME ABOUT DASHBOARD FGS MIRI NIRCAM NIRISS NIRSPEC

WELCOME TO THE JWST QUICKLOOK PAGE.

Dashboard Query Database

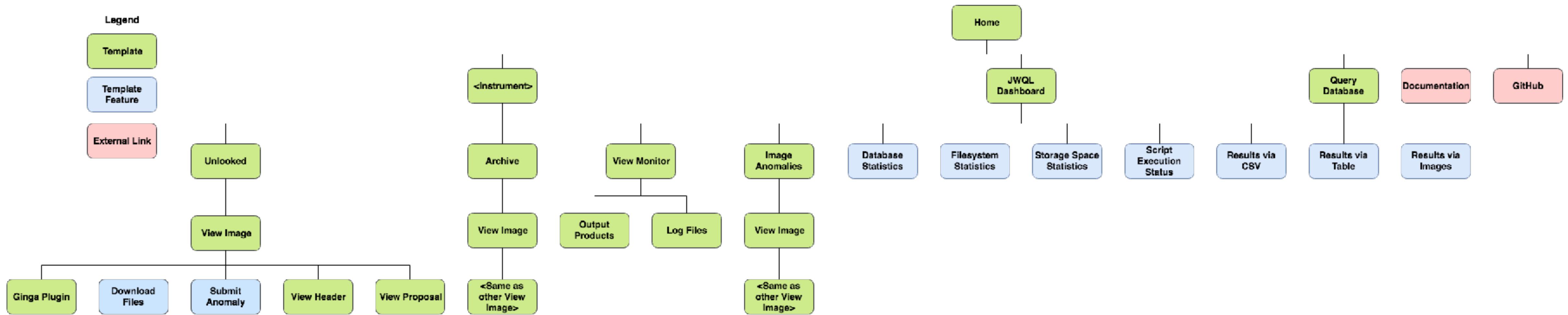
The James Webb Quicklook Application (JWQL) is a database-driven web application and automation framework for use by the JWST instrument teams to monitor the health and stability of the JWST instruments. Visit our [about page](#) to learn more about our project, goals, and developers.

The JWQL application is currently under heavy development. The 1.0 release is expected in 2019.

STScI SPACE TELESCOPE SCIENCE INSTITUTE



Front End: Site Map



GitHub issue: <https://github.com/spacetelescope/jwql/issues/35>



Front End: JWQL 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

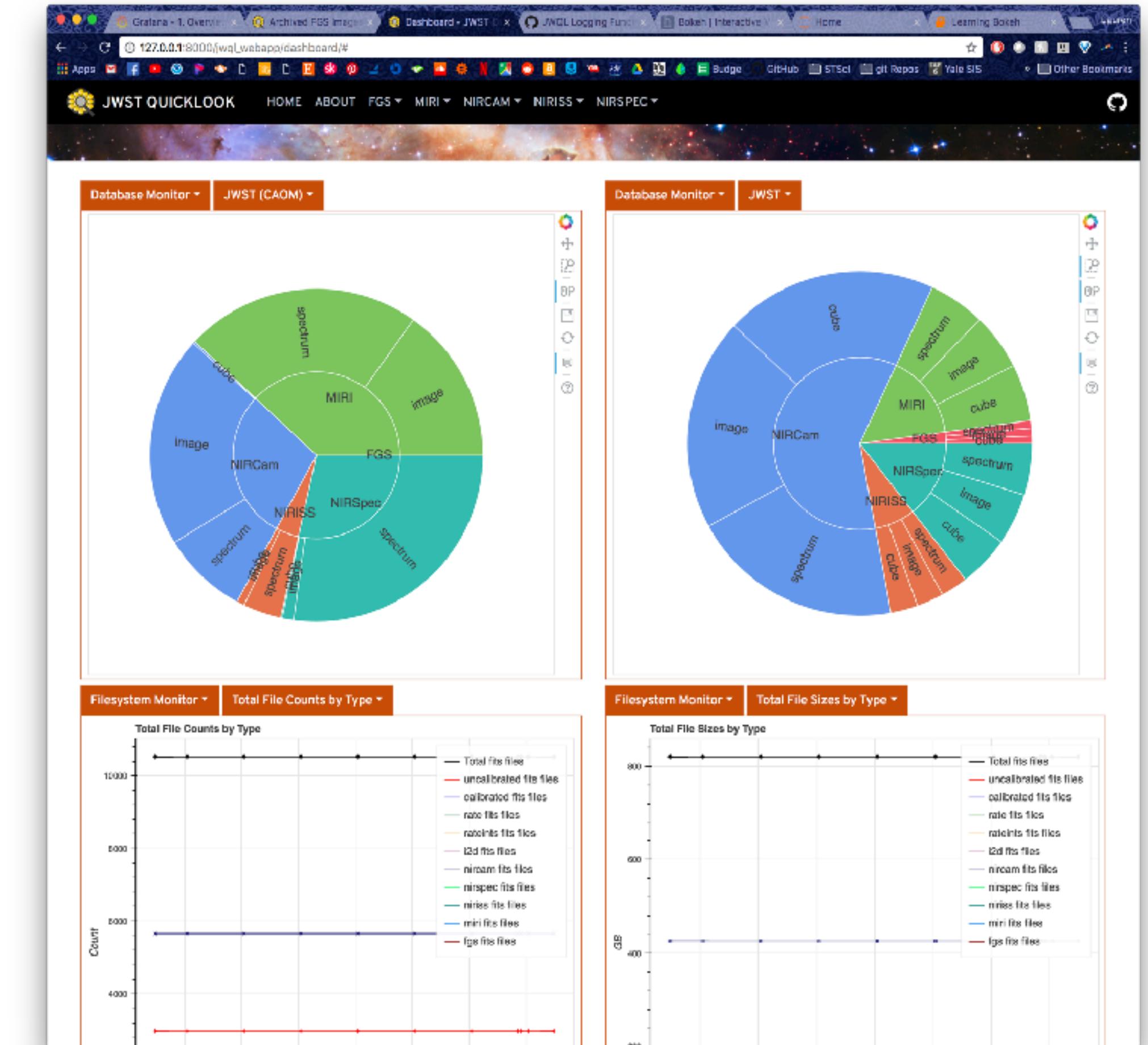
- Jinja2 is another template syntax, for the most part with similar functionality as Django templates
 - Django: `{% url 'jwql_webapp:home' %}`
 - Jinja: `{{ url('jwql_webapp:home') }}`
- BUT it might be easier to embed Bokeh plots in templates:

```
<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 FITS viewers in JWQL





JWQL Web App Directory Structure

```
website/
    __init__.py
    manage.py
    jwql_proj/
        jinja2.py
        settings.py
        urls.py
        wsgi.py
apps/
    __init__.py
    jwql/
        __init__.py
        apps.py
        data_containers.py
        db.py
        models.py
        tests.py
        urls.py
        views.py
    static/
        css/
        js/
        img/
    templates/
```



JWQL Web App Directory Structure

- `website/` - Main website directory
 - `__init__.py` - subpackage marker
 - `manage.py` - Django-generated file that performs operations such as running a local server, starting an interactive iPython shell, or running tests on the project
 - `jwql_proj/` - directory containing the project-specific files
 - `apps/` - directory containing different “applications”, kind of like reusable Python packages

```
website/
    __init__.py
    manage.py
    jwql_proj/
        jinja2.py
        settings.py
        urls.py
        wsgi.py
    apps/
        __init__.py
        jwql/
            __init__.py
            apps.py
            data_containers.py
            db.py
            models.py
            tests.py
            urls.py
            views.py
        static/
            css/
            js/
            img/
        templates/
```



JWQL Web App Directory Structure

- `jwql_proj/` - Django project directory
 - `jinja2.py` - Configures how the app will handle Jinja2 templates
 - `settings.py` - Defines project settings
 - `urls.py` - Defines URL-to-view mappings for the project
 - `wsgi.py` - Configures how to deploy the server with WSGI (Web Server Gateway Interface)...?

```
website/
    __init__.py
    manage.py
    jwql_proj/
        jinja2.py
        settings.py
        urls.py
        wsgi.py
    apps/
        __init__.py
        jwql/
            __init__.py
            apps.py
            data_containers.py
            db.py
            models.py
            tests.py
            urls.py
            views.py
        static/
            css/
            js/
            img/
        templates/
```



JWQL Web App Directory Structure

- `apps/` - directory containing all apps in project
 - `__init__.py` - subpackage marker
 - `jwql/` - directory of the JWQL web app app 😊

```
website/
    __init__.py
    manage.py
    jwql_proj/
        jinja2.py
        settings.py
        urls.py
        wsgi.py
apps/
    __init__.py
    jwql/
        __init__.py
        apps.py
        data_containers.py
        db.py
        models.py
        tests.py
        urls.py
        views.py
        static/
            css/
            js/
            img/
        templates/
```



JWQL Web App Directory Structure

- `jwql/` - directory containing the JWQL app app
 - `__init__.py` - subpackage marker
 - `apps.py` - Optionally defines an AppConfig class that can be called in settings.py
 - `data_containers.py` - Python functions written by Matt and I that are used in views.py (located here for cleanliness' sake)
 - `db.py` - Defines Python classes as interfaces between the web app and any databases (e.g. MAST)
 - `models.py` - Defines Python classes that represent the structure of a Django database
 - `tests.py` - Define unit tests for testing the web app
 - ...

```
website/
    __init__.py
    manage.py
    jwql_proj/
        jinja2.py
        settings.py
        urls.py
        wsgi.py
    apps/
        __init__.py
        jwql/
            __init__.py
            apps.py
            data_containers.py
            db.py
            models.py
            tests.py
            urls.py
            views.py
            static/
                css/
                js/
                img/
            templates/
```



JWQL Web App Directory Structure

- `jwql/` - directory containing the JWQL app
 - `...`
 - `urls.py` - Defines URL-to-view mappings for the app
 - `views.py` - Defines views: Python functions that process HTTP URL requests, do something based on the request, and return an HTTP response and template
 - `static/` - Directory containing static files: CSS style files, JavaScript files, and images. Note that `static/css/jwql.css` and `static/js/jwql.js` include custom styles and functions.
 - `templates/` - Directory containing HTML/Jinja2 templates

```
website/
__init__.py
manage.py
jwql_proj/
    jinja2.py
    settings.py
    urls.py
    wsgi.py
apps/
__init__.py
jwql/
    __init__.py
    apps.py
    data_containers.py
    db.py
    models.py
    tests.py
    urls.py
    views.py
    static/
        css/
        js/
        img/
    templates/
```



How do *you* contribute?



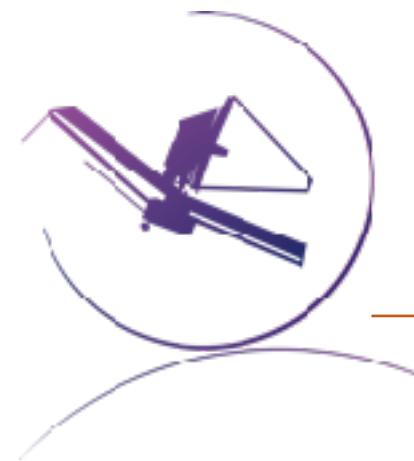
How to run the web app on a local server

- 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/jwql/>



How to add a webpage to the web application

- Create a new template (`my_template.html`) in `website/apps/jwql/templates/`
 - If needed, add new CSS to `website/apps/jwql/static/css/jwql.css` or new JavaScript functions to `website/apps/jwql/static/js/jwql.js`
- Create a new view (Python function) in `website/apps/jwql/views.py`
- Add your desired URL, linking to the view, to `website/apps/jwql/urls.py`



Helpful References

- 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/>

Code Review
