HTFome

A.M. Andersen, K. Antoniuk, N. Hardie and A. Sutradhar

About

- HTFome (https://htfome.com/) is a web application (webapp) designed to give users access to data regarding Human Transcription Factors (HTF's), drugs which target HTF's, and allow users to upload gene expression data for analysis in-browser.
- The webapp was developed as part of the Queen Mary University of London (QMUL) bioinformatics masters degree program. Specific application specifications can be found on the QMUL website: https://qmplus.qmul.ac.uk/course/view.php?id=16766.
- Developers of this app were A.M. Andersen, K. Antoniuk, N. Hardie and A. Sutradhar.
- The source code for the app can be found on Github: https://github.com/NHardie/HTFome.

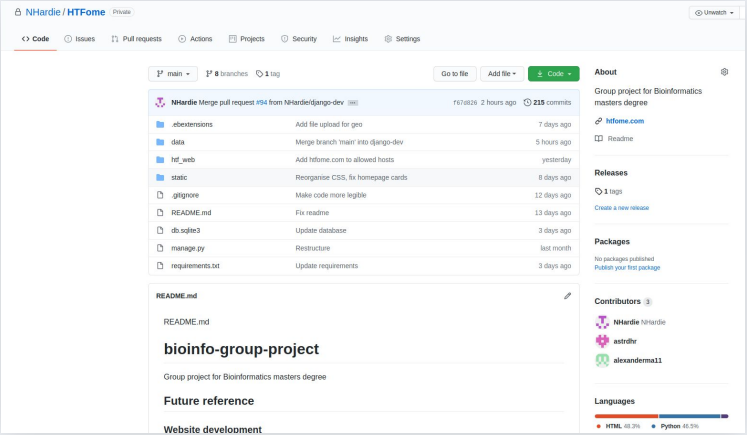
Abstract

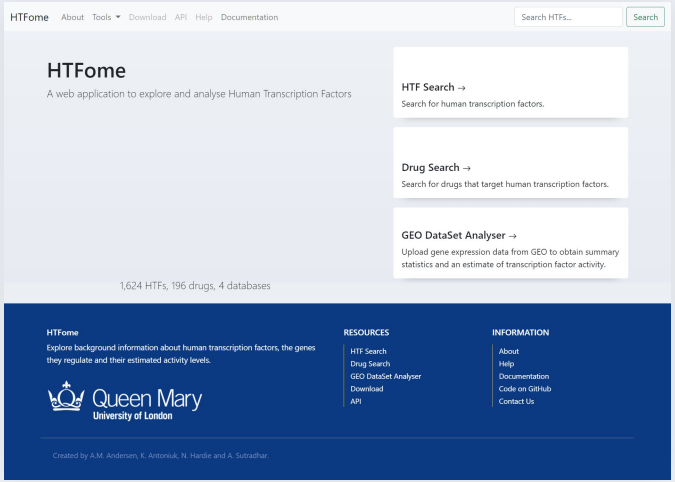
Human pluripotent stem cells (hPSCs) offer an unprecedented opportunity to model diverse cell types and tissues. To enable systematic exploration of the programming landscape mediated by transcription factors (TFs), we present the Human TFome, a

Ng, A.H.M., Khoshakhlagh, P., Rojo Arias, J.E. et al. A comprehensive library o human transcription factors for cell fate engineering. Nat Biotechnol (2020).

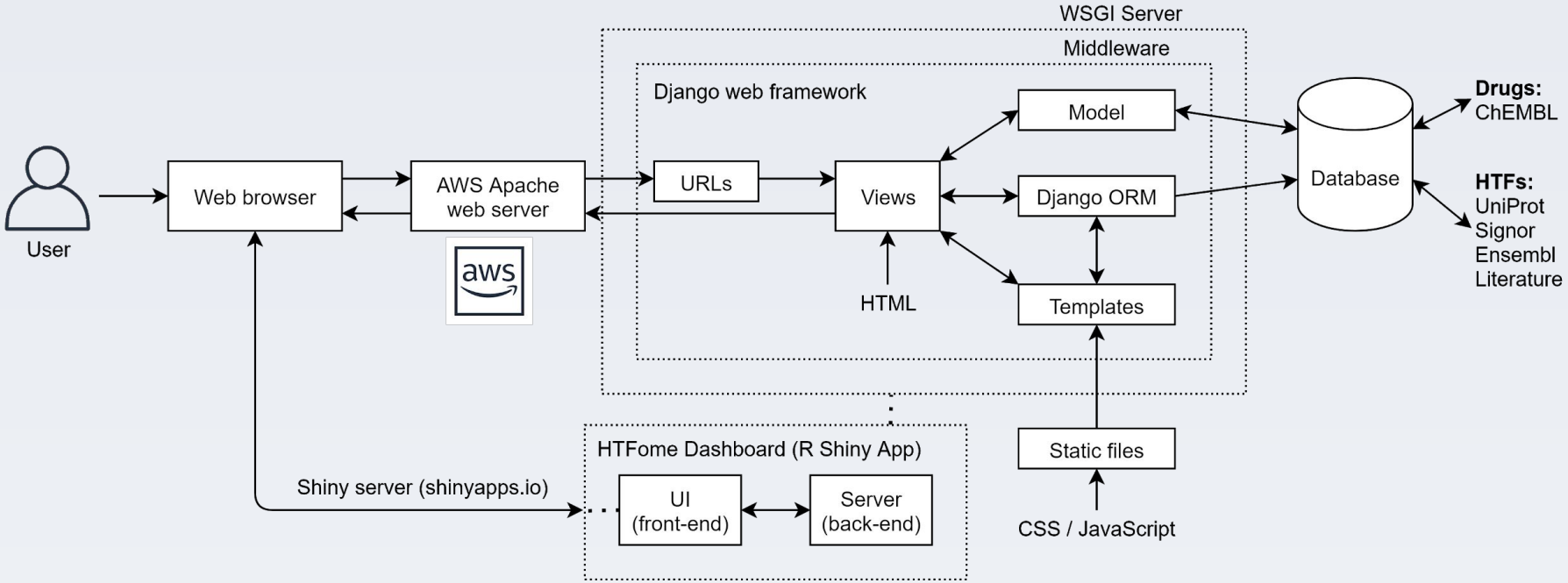
Overall design philosophy

- The HTFome app was designed to provide biologists with a simple method of accessing data related to HTF's and the drugs that target them.
- The aim was to make it easy for a biologist to enter some parameters, such as a gene symbol, and quickly be provided with useful information about the relationships that gene has to other HTF genes, and to drug compounds.
- App cannot rival Ensemble, UniProt or ChEMBL, we can provide a small selection of data, and hyperlink to the original sources.
- The app is designed to be modular and scalable, to aid any future development efforts. Github was used throughout development to aid with version control and app releases.



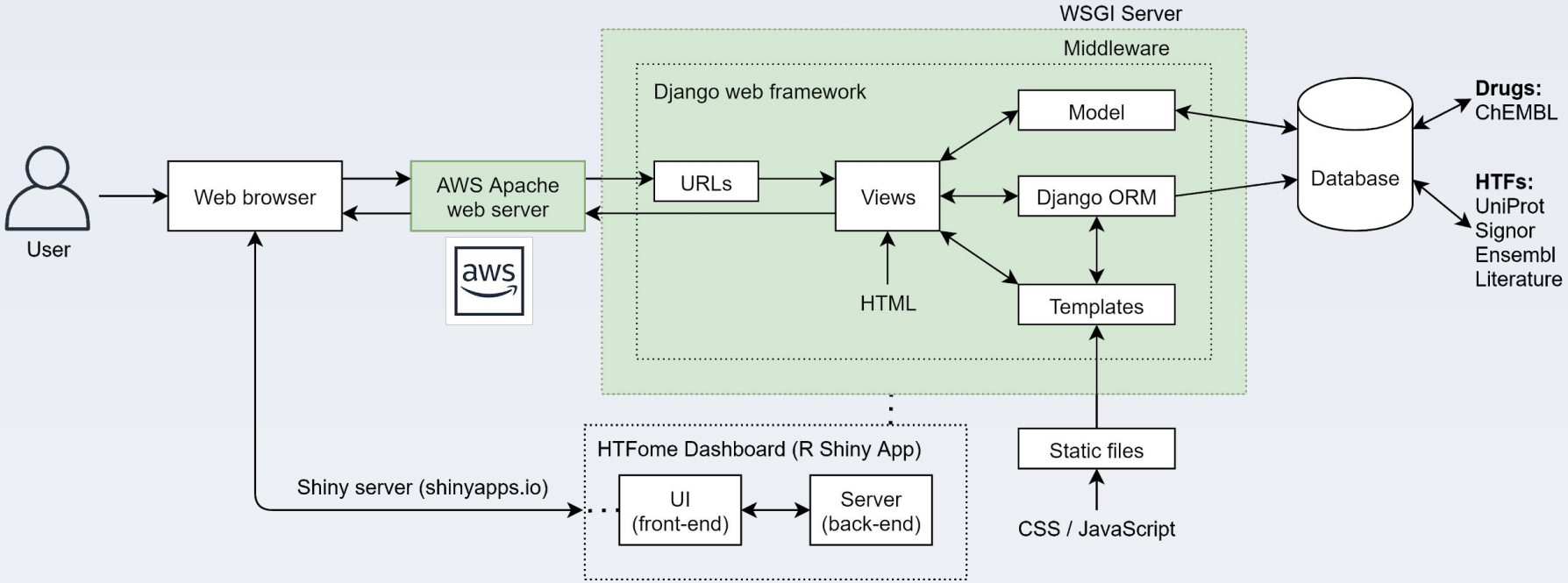


Software architecture



Deep-dive

Django web-framework



Django web-framework

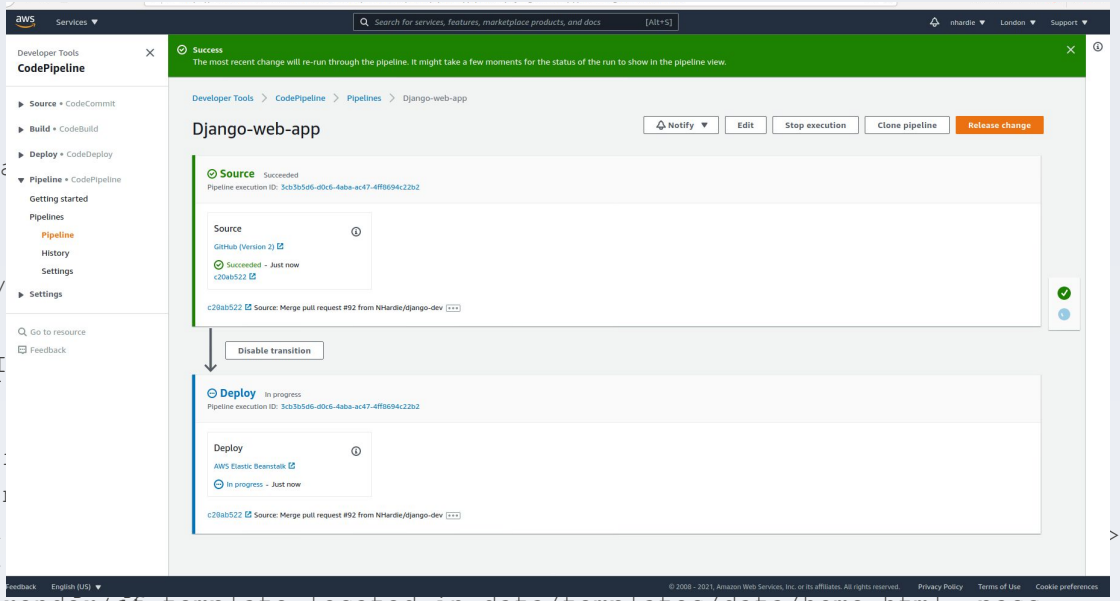
- The project itself is developed predominantly in Django, a python-based framework for creating web app's.
- Initially we opted to use the Flask framework, Flask is recognised as a micro-framework, perfect for developers who are new to web apps. (i.e: Us)
- There is some compromise between built-in features and overall design flexibility.
- Django includes several useful features missing from Flask, which make development easier, such as a built-in administration system and a database ORM (Object-Relational-Mapper).
- Django is seen as more scalable, Django releases have forwards-compatibility in mind.
- In a Django project there can exist multiple apps, which allows for more modular development, as well as the ability to copy and repurpose apps later in production.
- Django documentation is somewhat more complex and confusing than Flask's.

Django workflow

General Django development follows these steps:

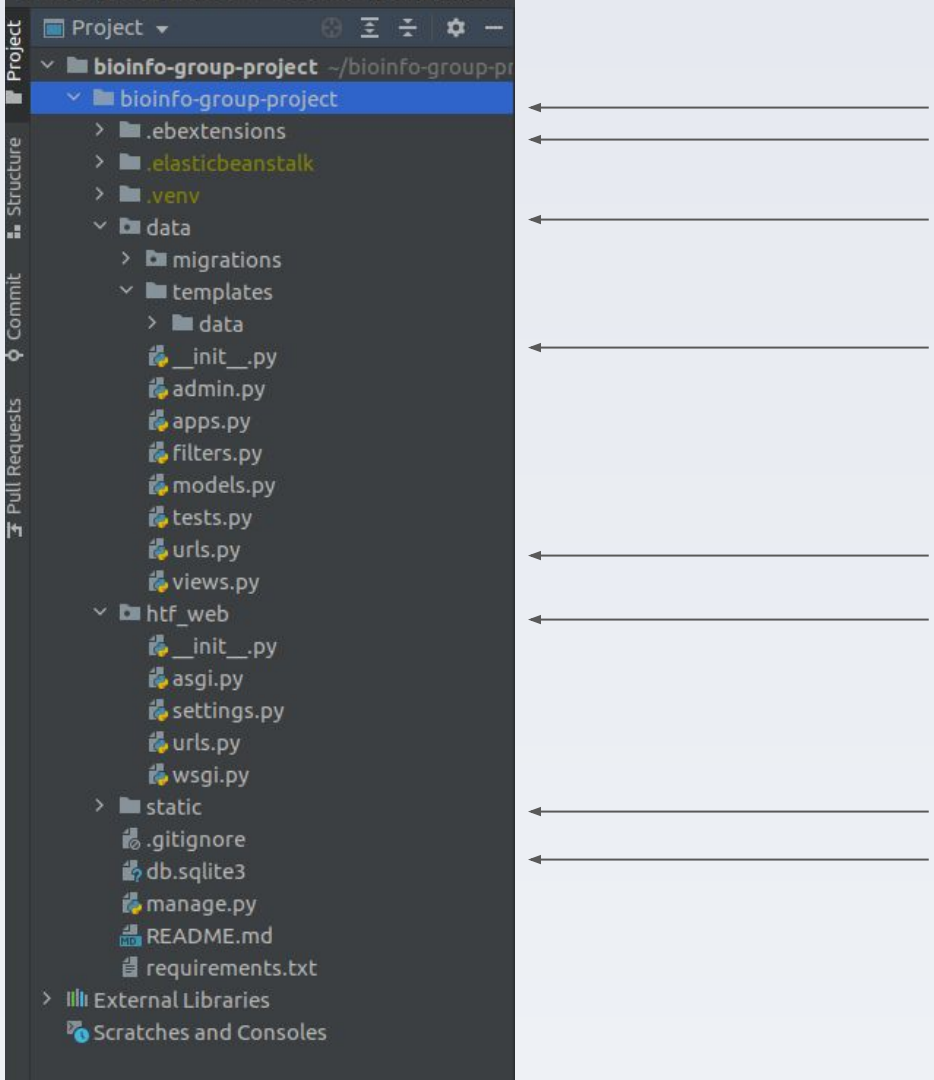
- Create the project
- Create an app
- Link app to project URL's and in settings
- Write app URLS
- Write app views
- Write HTML templates
- Test locally
- Deploy changes
- (Models/ Database, etc)

```
/da

{%

/
{%

I
/views.py

def home(
    # Defi
    return
    Factor
# return render(request, template located in data/templates/data/home.html, pass
# title variable to html page
```

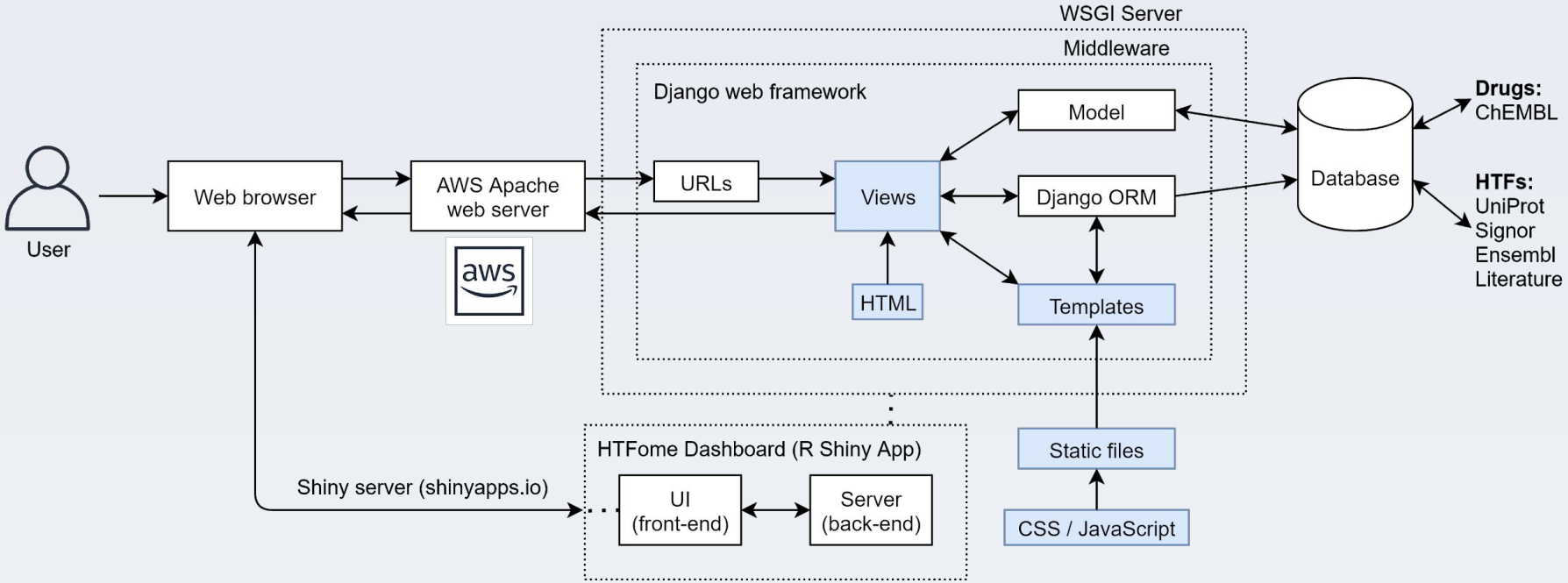



```
            'htf_web',
            'data.apps.DataConfig',
            'django_filters',
        ]
```

Django File Structure



Front-end development

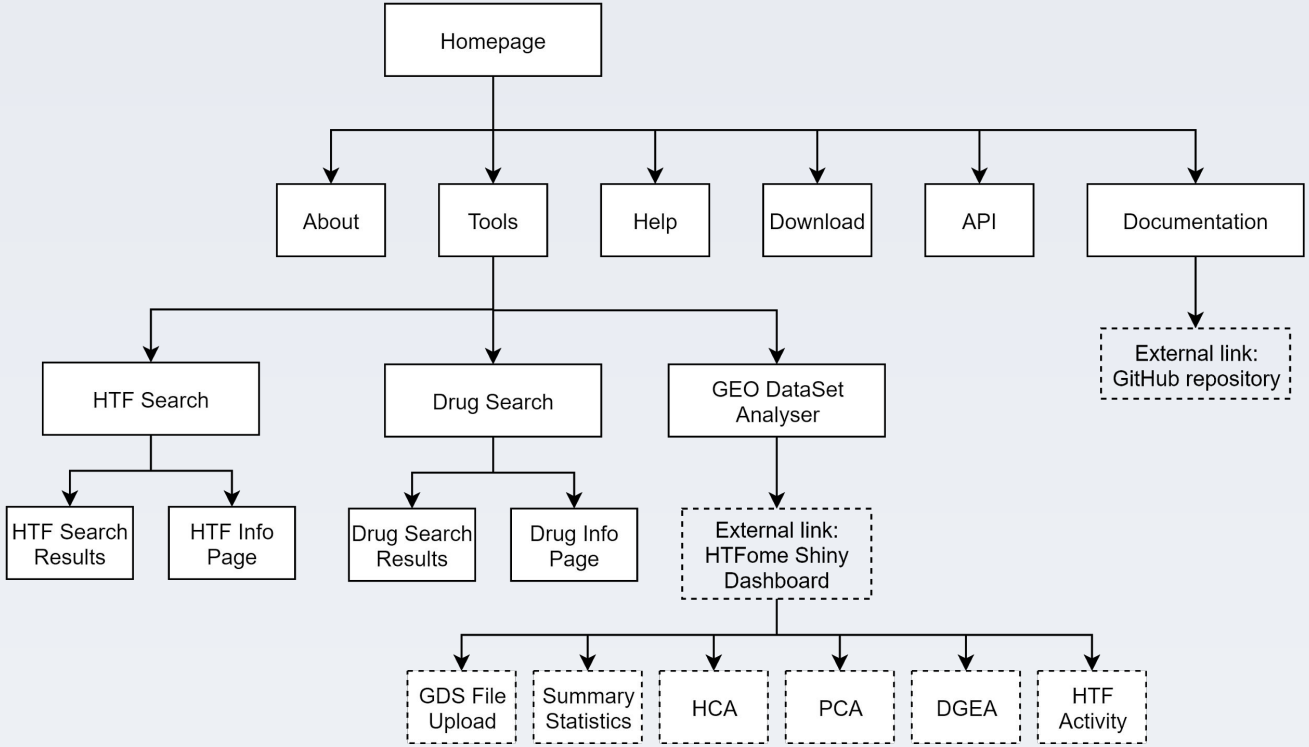


Front-end development

Main components:

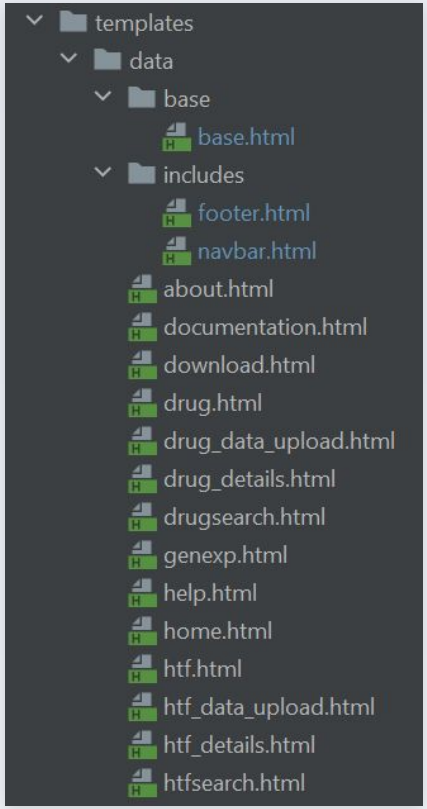
- HTML
- CSS (with Bootstrap v4.0)
- JavaScript

```
/*-------------------------------------

Master stylesheet for HTFome website:

1. Body and general page layout
2. Navigation bar
3. Footer
4. Homepage
5. About
6. HTF/Drug search
7. HTF/Drug info pages
8. GEO DataSet analyser

-------------------------------------
```



Schematic showing HTFome site map.

Front-end: Django integration



base.html loads static and basic front-end for all pages.

footer.html and navbar.html get included in base.html.

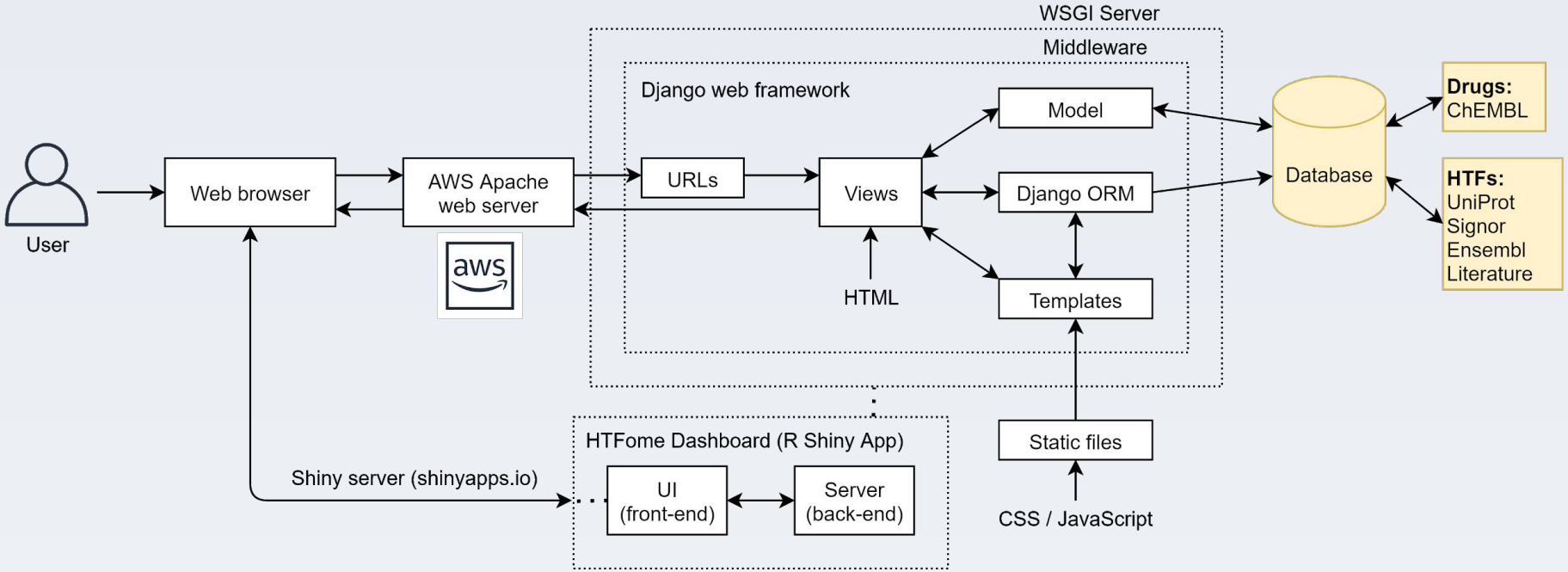
All .html pages extend (i.e. inherit from) base.html.

```
28  <body>
29  <!-- Navigation bar -->
30  {% include 'data/navbar.html' %}
31
32  <!-- Page content -->
33  {% block content %}{% endblock %}
34
35  <!-- Footer -->
36  {% include 'data/footer.html' %}
```

```
1  {% extends "data/base.html" %}
2
3  {% block content %}
4
5      <!-- HTML content goes here -->
6
7  {% endblock content %}
```

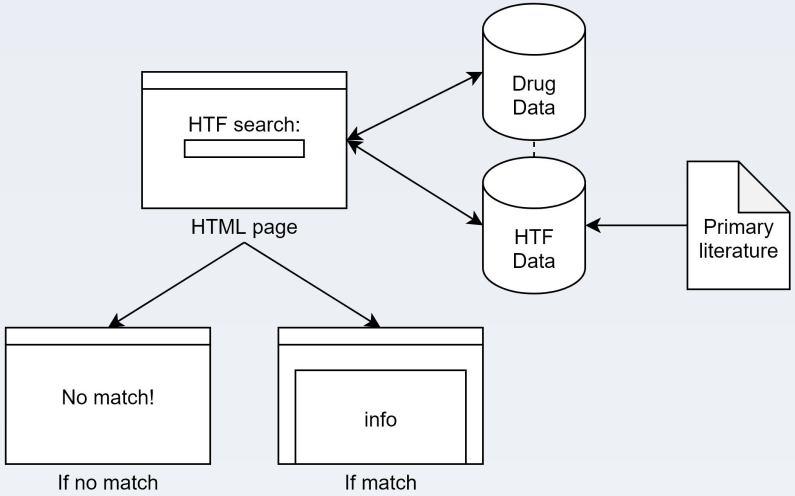
Example use case of “extends” and “include” code blocks.

Databases


WSGI Server
Middleware
Django web framework
Model
Database
Drugs:
ChEMBL
User
Web browser
AWS Apache
web server
URLs
Views
Django ORM
HTFs:
UniProt
Signor
Ensembl
Literature
aws
HTML
Templates
HTFome Dashboard (R Shiny App)
Static files
Shiny server (shinyapps.io)
UI
(front-end)
Server
(back-end)
CSS / JavaScript

HTF database

- HTFome requires HTF and drug data to provide to the user, and for simplicity this data has to be freely and easily accessible by the public.
- We utilised the Application-Programming Interfaces (APIs) of major biological data repositories to write requests specific to the HTF and drug background data we required.
- Once this data was gathered, we used Python to clean/manipulate the data before it was added to the webapp database.
- The HTF data were gathered from several sources
    - Lambert, Samuel A. et. al 2018 - List of HTFs
    - UniProt - Biological information
    - Ensemble - Gene position
    - Signor - Gene regulation



Drug database

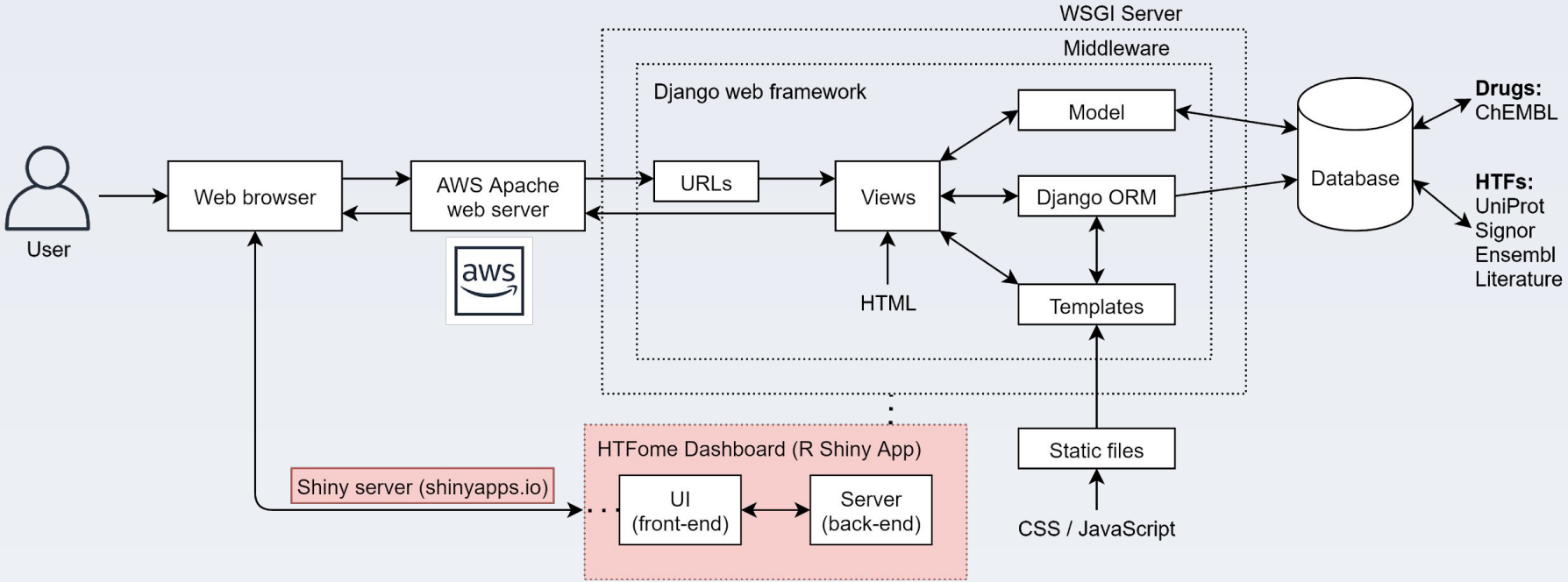
- The drug data was gathered from ChEMBL, a database organised and maintained by the European Molecular Biology Laboratory (EMBL).
- We gathered data on small molecules with drug-like effects, including drugs used in medicine, drugs used in research, including in trials, and predicted drug compounds.
- We eventually chose to display only phase 4 drugs.
- ChEMBL includes a wealth of data for each compound, as well as drug targets, making it perfect for our requirements.

```python
#PARSE CSV WITH TFs CHEMBL IDS
with open('chemblids.csv', 'r') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    for row in reader:
        compounds2targets[row[0]] = set()

chunk_size = 50
keys = list(compounds2targets.keys())

#MAP TARGET TFs TO COMPOUNDS (DRUGS)
for i in range(0, len(keys), chunk_size):
    activities = new_client.mechanism.filter(target_chembl_id__in=keys[i:i + chunk_size]).only(['target_chembl_id', 'molecule_chembl_id'])
    for act in activities:
        compounds2targets[act['target_chembl_id']].add(act['molecule_chembl_id'])



#FILTER ONLY PHASE 4 DRUGS
drug_chembl_ids=[]
for key, val in compounds2targets.items():
    lval = list(val)
    genes = []
    temp=[]
    for i in range(0, len(val), chunk_size):
        compounds = new_client.molecule.filter(molecule_chembl_id__in=lval[i:i + chunk_size]).only(
            ['molecule_chembl_id','max_phase','pref_name'])
        for compound in compounds:
            if compound['max_phase'] == 4:
                    genes.append(compound['pref_name'])
                    temp.append(compound['molecule_chembl_id'])
    drug_chembl_ids.append(temp)
    compounds2targets[key] = genes
print(drug_chembl_ids)

new_dict=compounds2targets.copy()
genesymbols=[]

#ADD GENE SYMBOLS FOR TFS
for key in keys:
    symbol = []
    targets = new_client.target.filter(target_chembl_id=key).only(
```

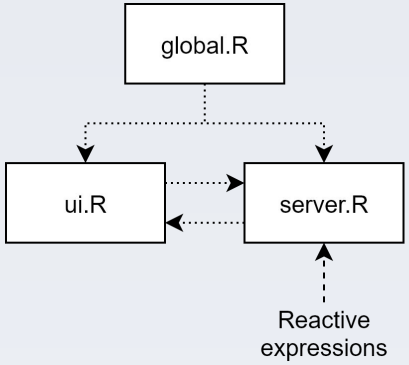
33	CHEMBL4523713	[]	['ENSG00000091831', 'ENSG00000113851', 'ENSG00000	['ESR1', 'CRBN', 'CUL4A']	[]
34	CHEMBL239	['CHEMBL457', 'CHEMBL565', 'CHEMBL672', 'CHEMBL9	['ENSG00000186951']	['PPARA']	['GEMFIBROZIL', 'CLOFIBRATE', 'FENOFIBRATE', 'FENOFIBRIC ACID', 'CIPROFIBRATE', 'CHOLINE FENOFIBRATE']
35	CHEMBL2111371	[]	['ENSG00000132170', 'ENSG00000112033']	['PPARG', 'PPARD']	[]
36	CHEMBL5716	[]	['ENSG00000177463']	['NR2C2']	[]
37	CHEMBL4523625	[]	['ENSG00000126561', 'ENSG00000173757']	['STAT5A', 'STAT5B']	[]
38	CHEMBL3509582	[]	['ENSG00000028277']	['POU2F2']	[]

DRUG PREF NAME	DRUG TRADE NAMES	DRUG CHEMBL ID	DRUG MOLECULE TYPE
['CERITINIB']	['Zykadia']	CHEMBL2403108	['Small molecule']
['GILTERITINIB']	[]	CHEMBL3301622	['Small molecule']
['BENZBROMARONE']	"['Desuric', 'Narcaricin mite']"	CHEMBL388590	['Small molecule']

GEO DataSet Analyser



GEO DataSet Analyser: R Shiny Dashboard



Define UI in ui.R:

```
# Main panel for displaying outputs
mainPanel(

    tabsetPanel(type = "tabs",
                tabPanel( title: "Data Summary", withSpinner(htmlOutput( outputId: "data_summary"))),
                tabPanel( title: "GDS Preview", withSpinner(dataTableOutput("gds_preview"))),
                tabPanel( title: "Phenotype Data Preview", withSpinner(dataTableOutput("pDat_preview")))
                )
```

Define server logic in server.R:

```
# Extract phenotype data from eSet object
pDat <- reactive({
    pData(eset())
})
```

Create reactive expression.

```
# display phenotype data in "pDat_preview" tab
output$pDat_preview <- renderDataTable({
    validate_upload()
    pDat()
})
```

Create reactive output.

HTF activity

- R-package:
  - Virtual Inference of Protein-activity by Enriched Regulon analysis (VIPER)
    - Regulon - A collection of a TF and its transcriptional targets
      - DoRothEA
    - ExpressionSet
      - GEOquery

```r
#convert DoRothEA network to regulon
data(dorothea_hs, package = "dorothea")
viper_regulons <- df2regulon(dorothea_hs)
```

Untitled1 | dorothea_hs | viper_regulons

Filter

	tf	confidence	target	mor
1	ADNP	D	ATF7IP	1
2	ADNP	D	DYRK1A	1
3	ADNP	D	TLK1	1
4	ADNP	D	ZMYM4	1
5	ADNP	D	ABCC1	1
6	ADNP	D	ABCC6	1
7	ADNP	D	ABHD5	1
8	ADNP	D	ABT1	1
9	ADNP	D	AC002066.1	1
10	ADNP	D	AC002127.4	1

Untitled1 | dorothea_hs | viper_regulons

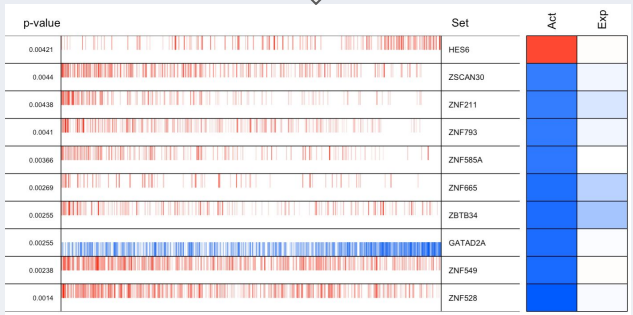
Show Attributes

Name	Type	Value
viper_regulons	list [1333]	List of length 1333
ADNP	list [2]	List of length 2
tfmode	double [436]	1 1 1 1 1 1 ...
likelihood	double [436]	1 1 1 1 1 1 ...
ADNP2	list [2]	List of length 2
AEBP2	list [2]	List of length 2
AHR	list [2]	List of length 2
AHRR	list [2]	List of length 2
AIRE	list [2]	List of length 2
ALX1	list [2]	List of length 2
ALX3	list [2]	List of length 2

HTF activity

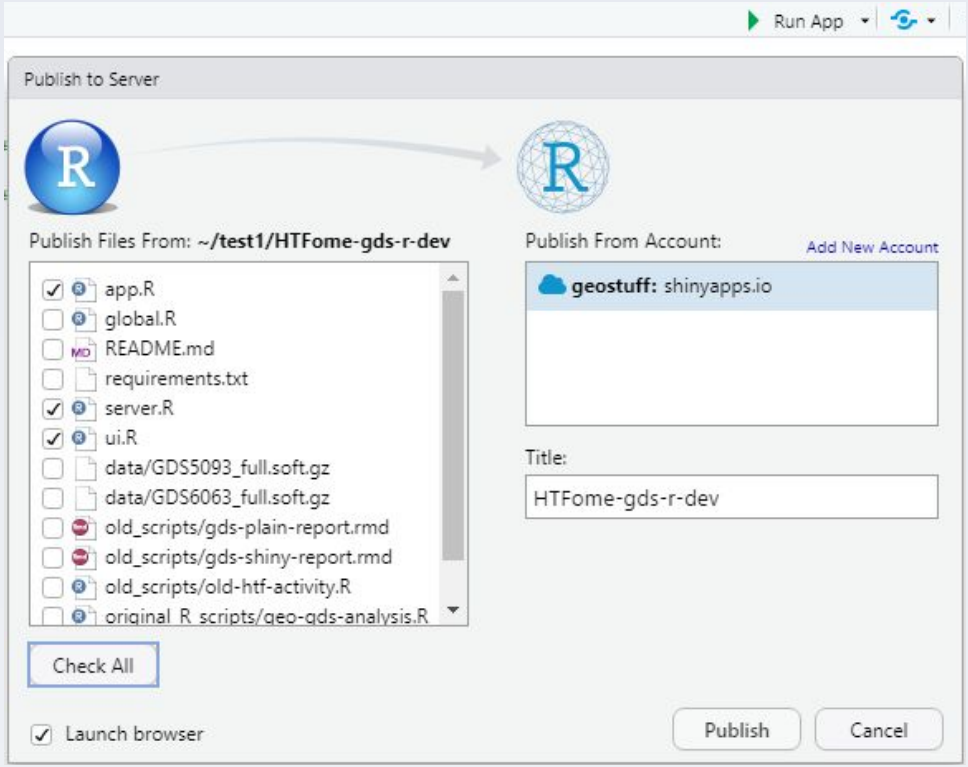
- VIPER analysis
  - ExpressionSet
    - Student’s t-test
    - Nullmodel

```r
#msVIPER analysis
mra <- msviper(signature, viper_regulons, nullmodel, verbose = FALSE)

plot(mra, cex=.7) #plot the analysis
```



Shiny app deployment

- Shiny app can be deployed on shinyapps.io or using AWS Linux server
- Shinyapps.io is easy to set up and does not require experience with Linux virtual machines.
- However, not all dependencies needed for our app to work are currently installed on shinyapps.io.
- It is possible to request adding those dependencies.
- Custom Linux server is hard to set up and needs to be maintained, but gives more control and allows to install custom dependencies.



HTFome Summary & Evaluation

Limitations

- The Database.
- Main point of SQL databases is handling relational data.
- We chose to relate the data in the webpage views/ HTML, rather in the Django models (database schema).
- The Search
- Compromise between broad and narrow search scope, we include two methods of searching to deal with this issue, but would be preferable to filter results by relevancy.
- GEO Data Analysis
- Currently cannot host GEO analysis on shinyapps server due to incompatibility of packages

Opportunities for future development

- Future development should focus on the database, changing from uploading the database directly, to hosting an SQL database server via AWS S3 and automatic database updates should be configured.
- The data input and models.py should be changed to allow the data to relate to each other via foreign keys/ ManytoMany fields.
- Making HTFome's GEO DataSet Analyser available in-browser.
- Search functionality could be improved.
- Allow users to access/ download data directly.

Questions?