DJANGO

PYTHON BACKEND FOR WEB APPLICATIONS

Chris Winsor
2/26/2020

Prepared for Metrowest Boston Developers Machine Learning Group
Available from https://github.com/cwinsor/django_102_pluralsight

References…

Django:

- "Django Fundamentals“ (Reindert-Jan Ekker) https://app.pluralsight.com This is the “tictactoe” application – excellent
- Django Tutorial https://docs.djangoproject.com/en/3.0/intro/tutorial01/ Intro from The Source

Node, Postgres, Express:

- “Build a CRUD single page application with Node, Express, Angular, Postgres” (Michael Herman) https://mherman.org/blog/postgresql-and-nodejs/ This is an example frontend/backend javascript web app with postgres db. It uses express web server/routing and (a little) angular on the front-end. You will use npm, express, node, browser trace/debug features. You will see javascript used on both client and server. This is very standard (server-side javascript) architecture.

Front-end:

- “Front-End Web Development Quick Start With HTML5, CSS, and JavaScript” (Shawn Wildermuth) https://app.pluralsight.com/course-player?clipId=e5482b13-c204-4d52-89ec-94a1099592b0 Beginner HTML5, CSS, JavaScript – excellent

Alternatives

- Server-side Javascript (node.js, express)
  - *Most common implementation*
  - *Many many libraries*
  - *Python hooks in on the back-end*

- Flask
  - *Server-side Python – lighter than Django*

Why Django?

- *Python is the language of Machine Learning*
- *ONE language on the server, not two*
- *Is robust and suitable for commercial sites (vs Flask)*
- *Well documented, well structured (DRY principle)*
- *Python is fun!*

Setup

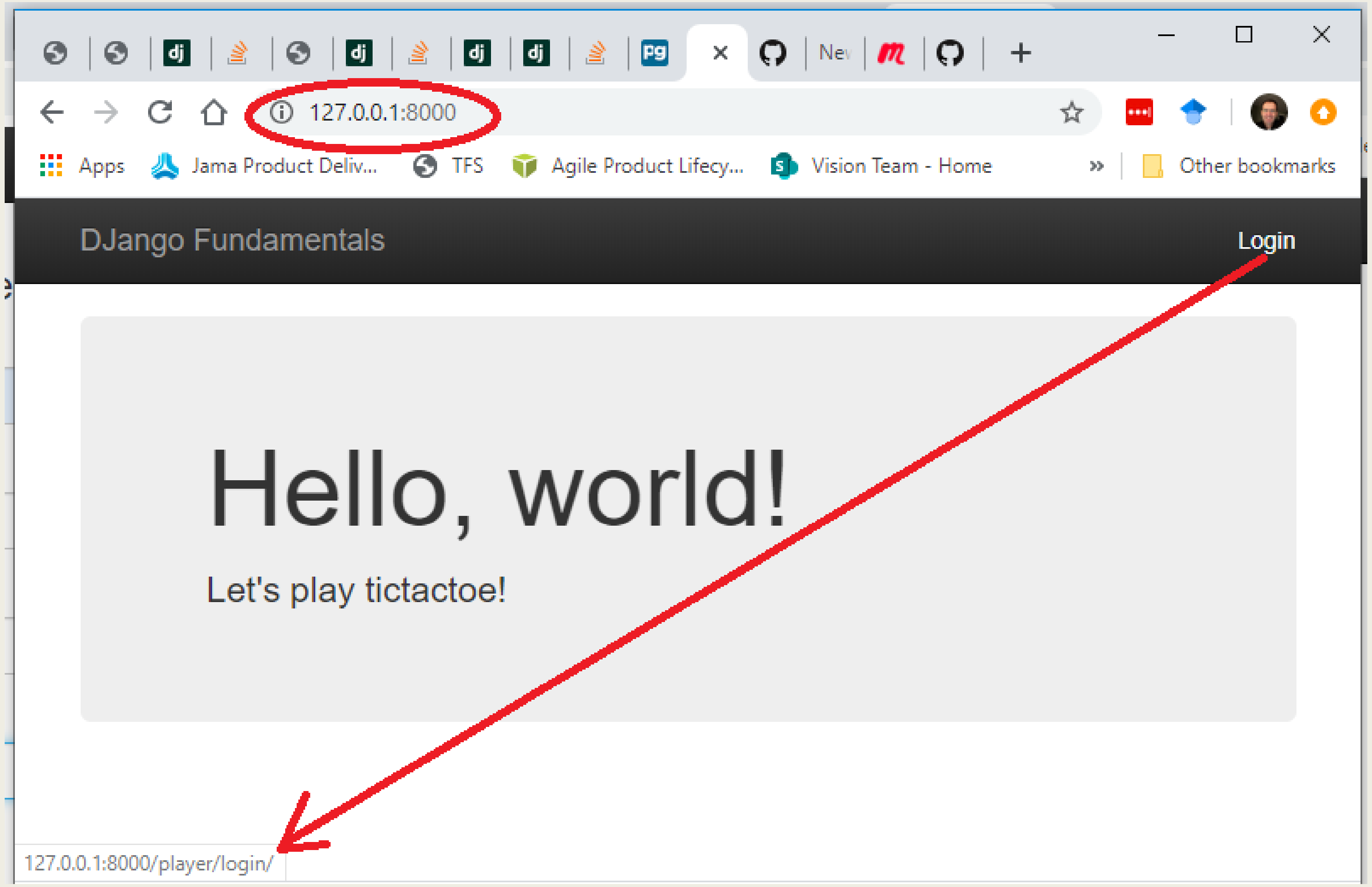
(you’ve seen this before)
Visual Studio Code, setup script, virtualenv+pip for modules

- mkdir myfolder; cd myfolder
- git clone https://github.com/cwinsor/django_102_pluralsight
- cd django_102_pluralsight\project
- ./setup.ps1

- to start visual studio code:
- cd .\tictactoe; code -n .

- to run the application:
- cd tictactoe; python manage.py runserver
- URLs are:
- http://127.0.0.1:8000 (user login)
- http://127.0.0.1:8000/admin/ (admin login)

Explore the app…


127.0.0.1:8000

DJango Fundamentals

Login

Hello, world!

Let's play tictactoe!

127.0.0.1:8000/player/login/

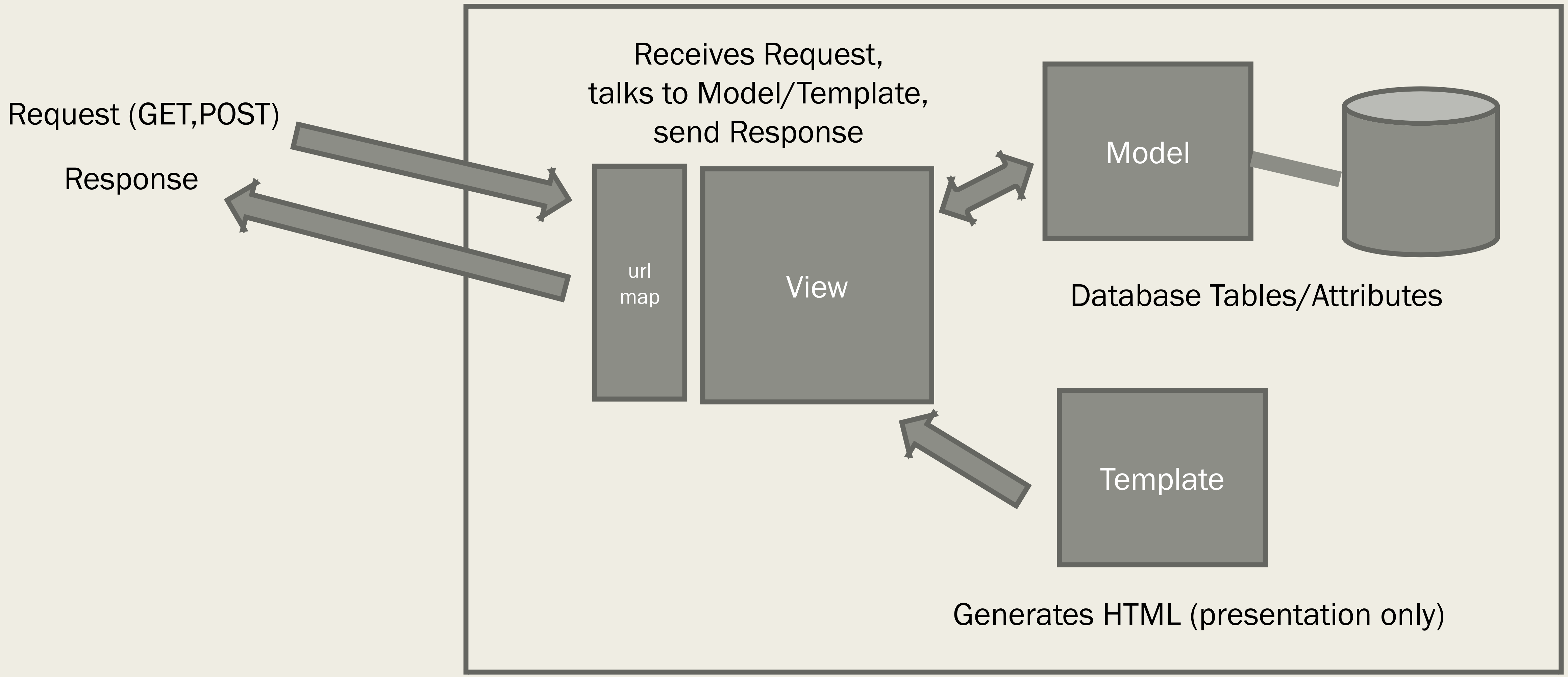
Explore the app…

- python manage.py runserver 0.0.0.0:9595

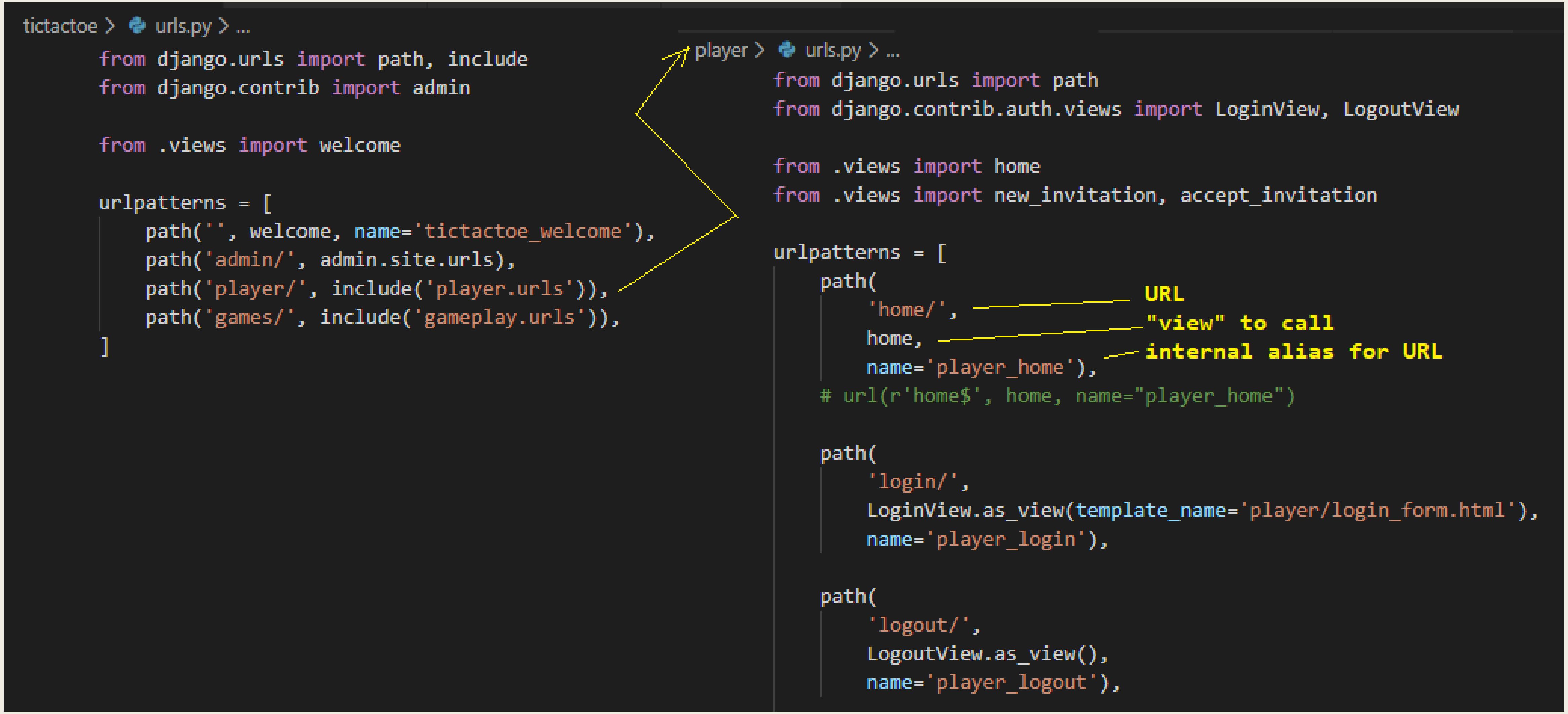
- http://192.168.1.220:9595
- alice   aabbddcc
- bob   aabbddcc

Model Template View

Similar to MVC



Mapping URL to View



```
from django.urls import path, include
from django.contrib import admin

from .views import welcome

urlpatterns = [
    path('', welcome, name='tictactoe_welcome'),
    path('admin/', admin.site.urls),
    path('player/', include('player.urls')),
    path('games/', include('gameplay.urls')),
]
```

```
from django.urls import path
from django.contrib.auth.views import LoginView, LogoutView

from .views import home
from .views import new_invitation, accept_invitation

urlpatterns = [
    path(
        'home/',
        home,
        name='player_home'),
    # url(r'home$', home, name="player_home")

    path(
        'login/',
        LoginView.as_view(template_name='player/login_form.html'),
        name='player_login'),

    path(
        'logout/',
        LogoutView.as_view(),
        name='player_logout'),
```

View

Receive request

If GET:

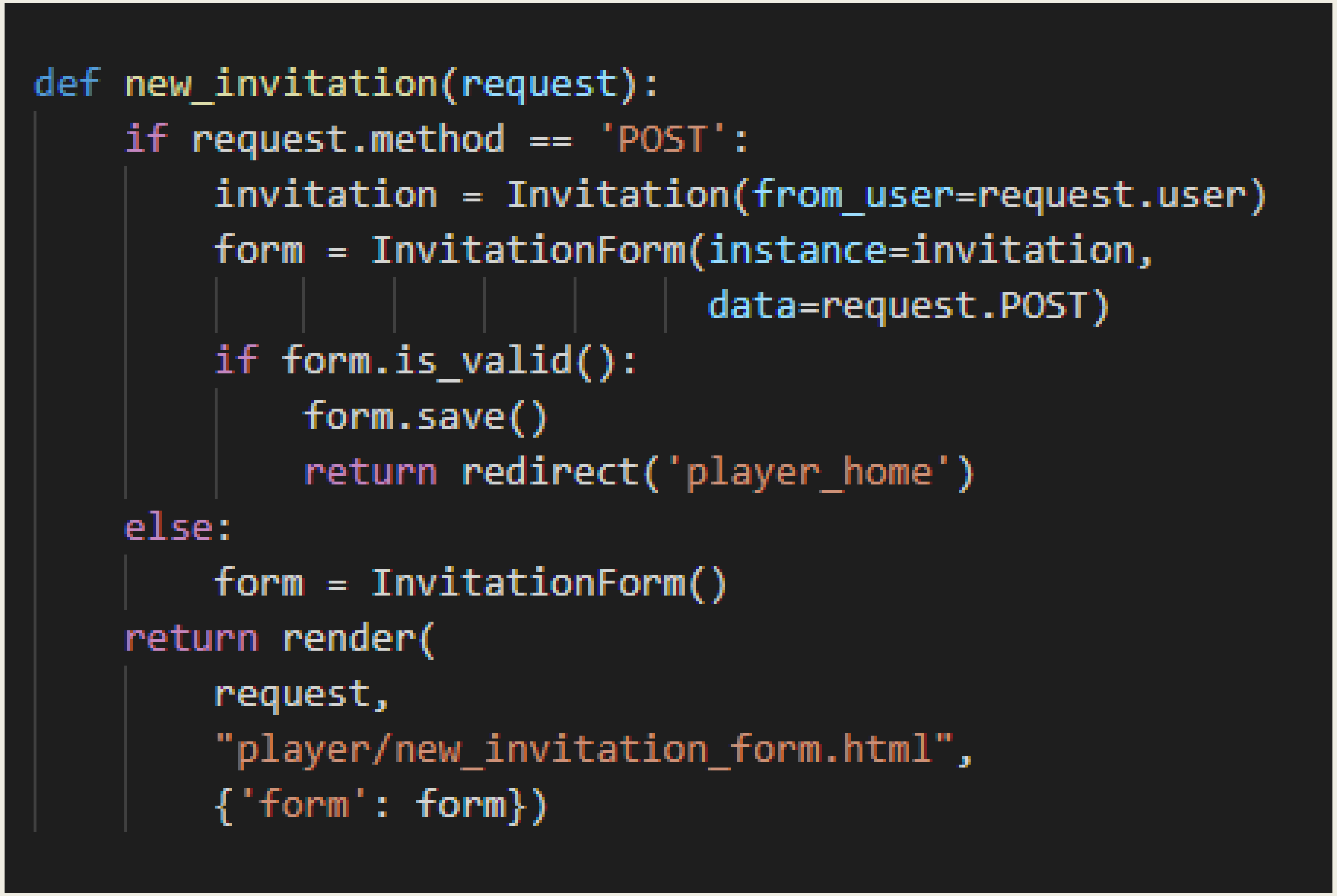
    Create a blank form and return it

If POST:

    Create a form using data from the POST (i.e. validate the data)

    If valid – save to DB and redirect to player_home page

    If not valid – send form back to user (with errors)



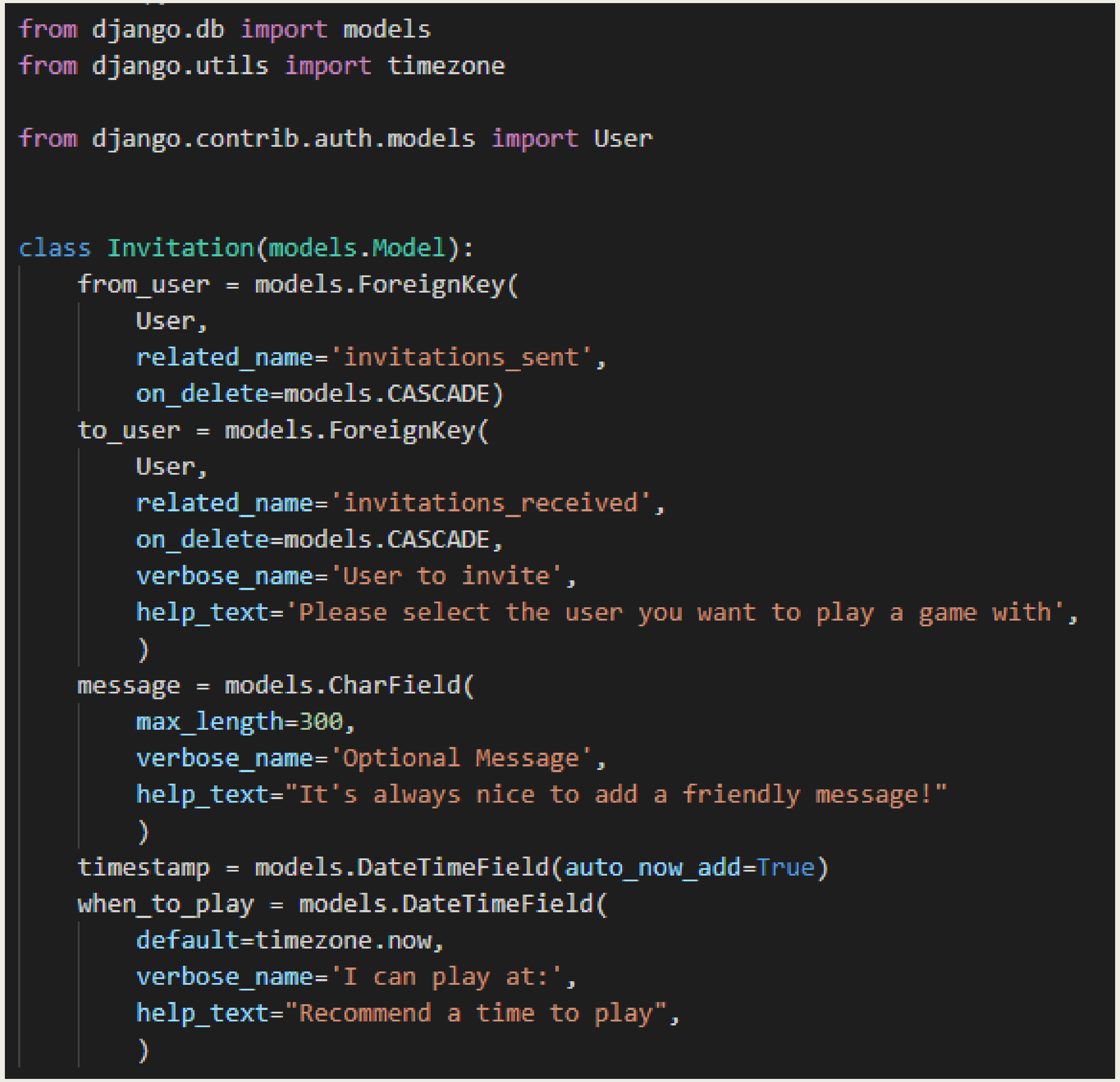
```python
def new_invitation(request):
    if request.method == 'POST':
        invitation = Invitation(from_user=request.user)
        form = InvitationForm(instance=invitation,
                              data=request.POST)
        if form.is_valid():
            form.save()
            return redirect('player_home')
    else:
        form = InvitationForm()
    return render(
        request,
        "player/new_invitation_form.html",
        {'form': form})
```

Model

- Game(models.Model)
  - firstPlayer = models.ForeignKey(User)
  - startTime = models.DateTimeField()
  - status = models.CharField()

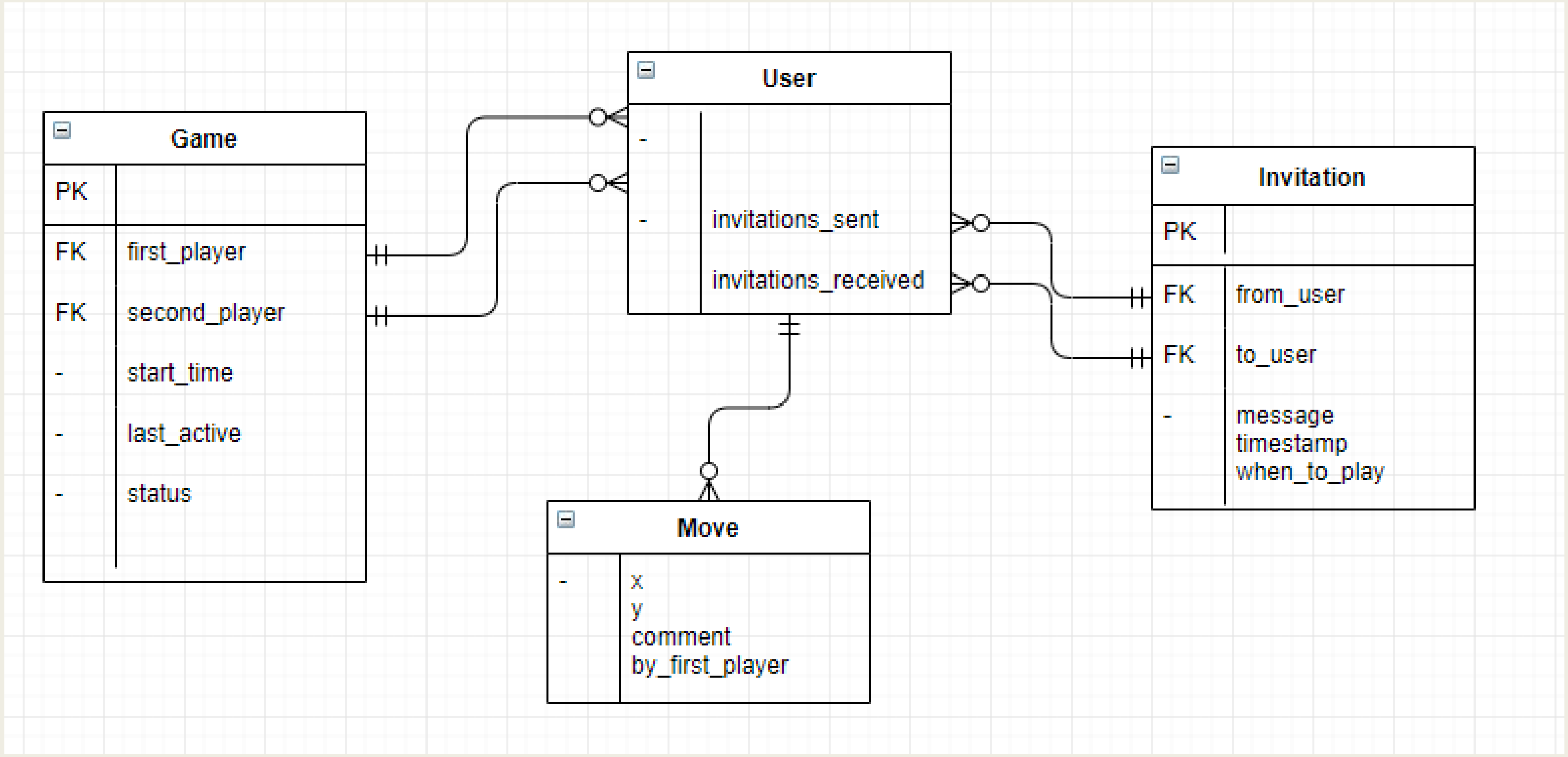
- Move(models.Model)
  - x, y = models.IntegerField()
  - game = models.ForeignKey(Game)

- Invitation(models.Model)
  - from_user = models.ForeignKey(User)
  - to_user = models.ForeignKey(User)
  - message = models.CharField()
  - time_to_play = models.DateTime()



```
from django.db import models
from django.utils import timezone

from django.contrib.auth.models import User


class Invitation(models.Model):
    from_user = models.ForeignKey(
        User,
        related_name='invitations_sent',
        on_delete=models.CASCADE)
    to_user = models.ForeignKey(
        User,
        related_name='invitations_received',
        on_delete=models.CASCADE,
        verbose_name='User to invite',
        help_text='Please select the user you want to play a game with',
        )
    message = models.CharField(
        max_length=300,
        verbose_name='Optional Message',
        help_text="It's always nice to add a friendly message!"
        )
    timestamp = models.DateTimeField(auto_now_add=True)
    when_to_play = models.DateTimeField(
        default=timezone.now,
        verbose_name='I can play at:',
        help_text="Recommend a time to play",
        )
```

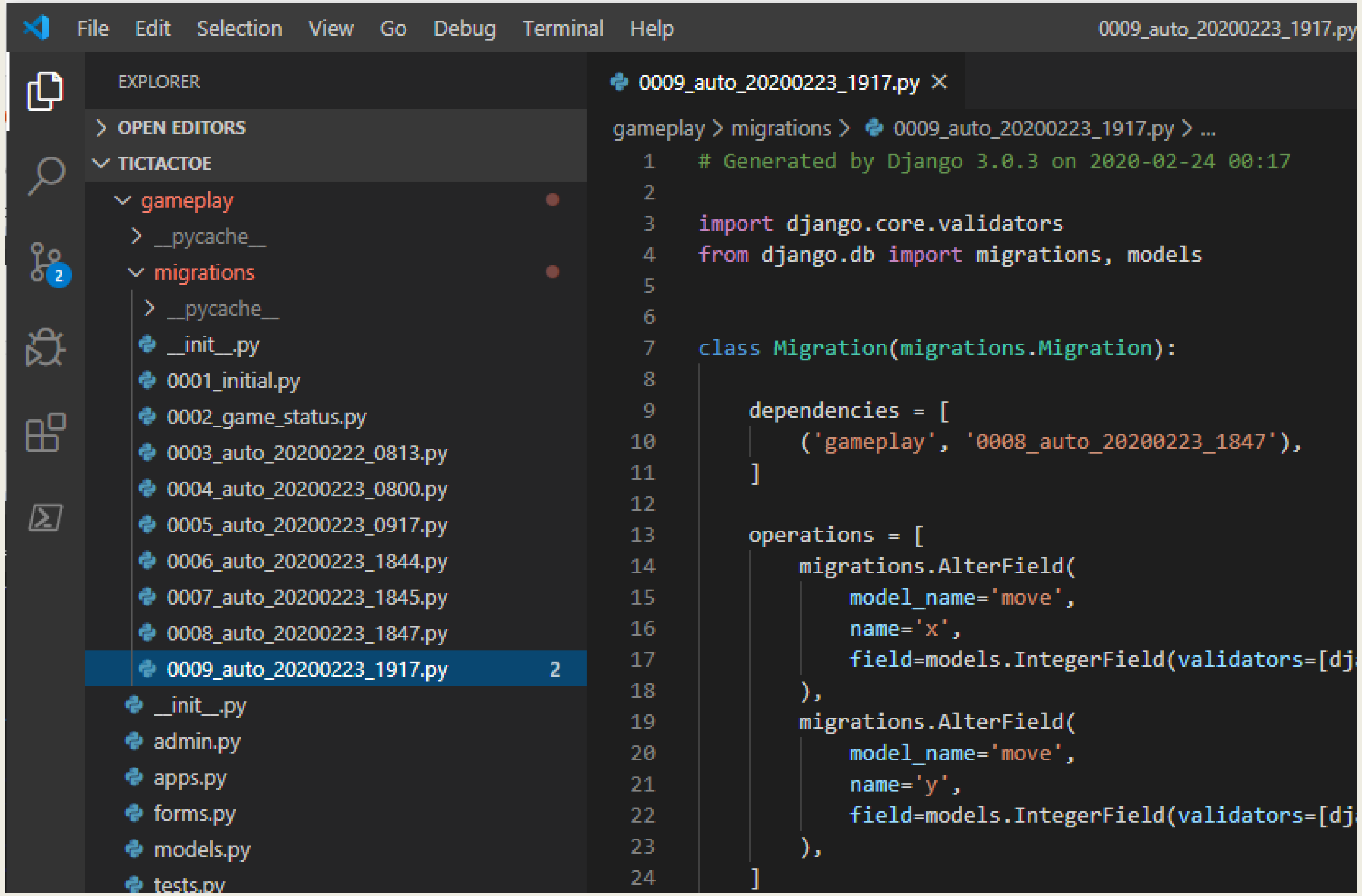
Desired Schema



Models and Migrations

(create and update schema) (provids API for Views) (and abstracts db-specifics)

- The Model is everything needed to create database tables
- Django creates “migrations” which implement the tables.
- Migrations also UPDATE existing schema
- Model provides an API for the View
- Model generates the SQL and hides vendor-specific details

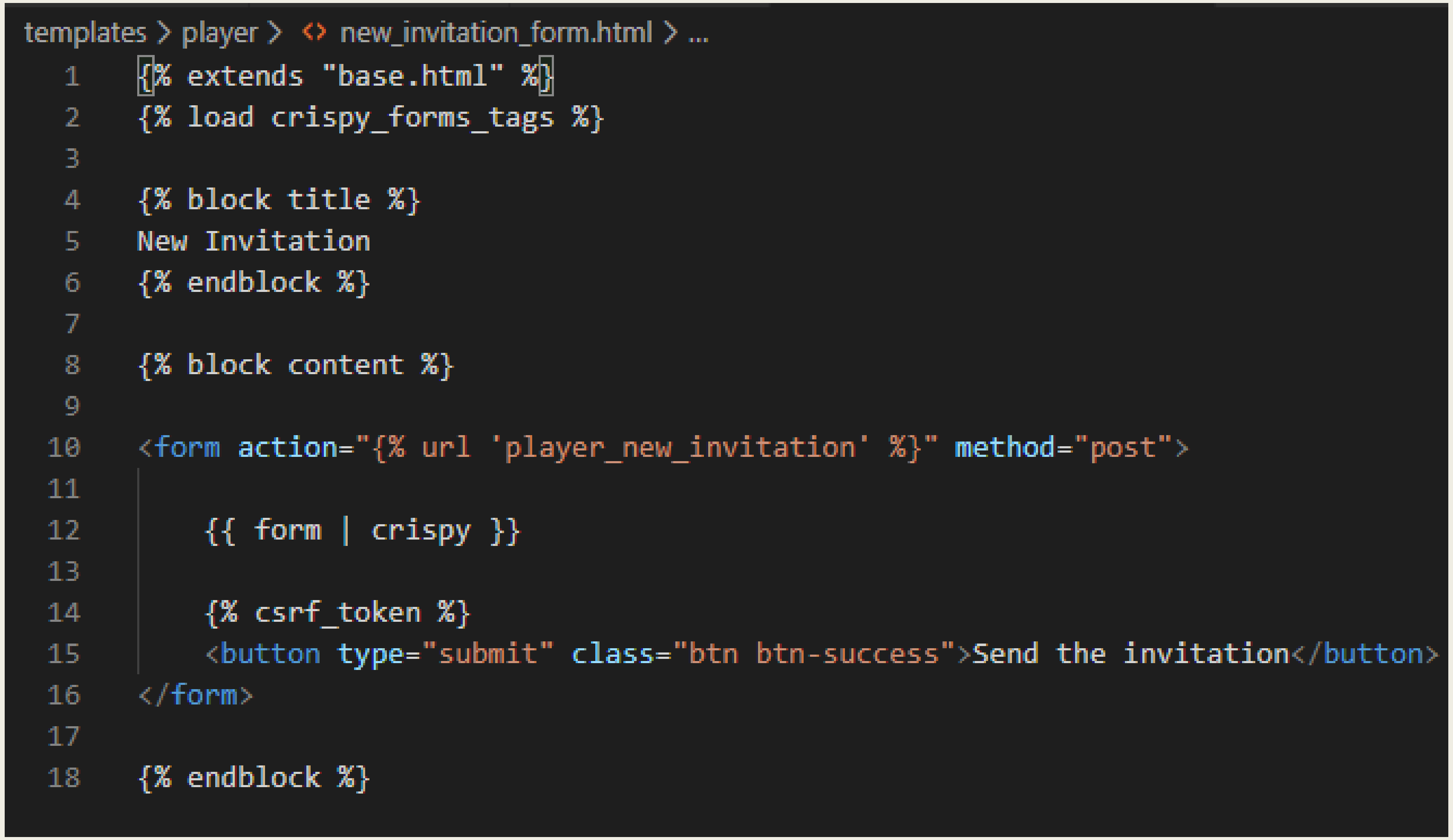


File
Edit
Selection
View
Go
Debug
Terminal
Help
0009_auto_20200223_1917.py
EXPLORER
OPEN EDITORS
TICTACTOE
gameplay
__pycache__
migrations
__pycache__
__init__.py
0001_initial.py
0002_game_status.py
0003_auto_20200222_0813.py
0004_auto_20200223_0800.py
0005_auto_20200223_0917.py
0006_auto_20200223_1844.py
0007_auto_20200223_1845.py
0008_auto_20200223_1847.py
0009_auto_20200223_1917.py
__init__.py
admin.py
apps.py
forms.py
models.py
tests.py
0009_auto_20200223_1917.py
gameplay > migrations > 0009_auto_20200223_1917.py > ...

```python
# Generated by Django 3.0.3 on 2020-02-24 00:17

import django.core.validators
from django.db import migrations, models


class Migration(migrations.Migration):

    dependencies = [
        ('gameplay', '0008_auto_20200223_1847'),
    ]

    operations = [
        migrations.AlterField(
            model_name='move',
            name='x',
            field=models.IntegerField(validators=[dj
        ),
        migrations.AlterField(
            model_name='move',
            name='y',
            field=models.IntegerField(validators=[dj
        ),
    ]
```

Templates

- Template is the structure to render the page.
- HTML + Bootstrap markup
- Extend “base.html” (the author got this from Initializr.com)
- “crispy” is CSS
- Form comes from View and knows how to render its elements.



templates > player > new_invitation_form.html > ...

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

{% block title %}
New Invitation
{% endblock %}

{% block content %}

<form action="{% url 'player_new_invitation' %}" method="post">

    {{ form | crispy }}

    {% csrf_token %}
    <button type="submit" class="btn btn-success">Send the invitation</button>
</form>

{% endblock %}
```

In Summary

- Django is Python + DB backend (can you say ML language and models…)
- User Authentication w/ forms
- Admin pages to view or edit project and DB tables
- Migrations
- Apache/PostgreSQL for production (starter DB/WS provided)
- Crispy forms

In summary – Django is production ready Python backend, perfect for Machine Learning and data science web applications.

Next steps …

PLaSTiCC database…

- 2 tables
- Maybe a dozen or so attributes
- Make a List View Page, Detail View Page
- How hard can it be!

THEN:

- Plug in (Python) predictive model from Kaggle from B. Trotta or Kyle Boone
- Make a game of it …

SO:

- Pick a star from the (List View) of stars
- Display detail view (macro data + timeseries data) in tabular form (bonus points for a chart)
- Have user make a prediction
- Push button to reveal predictive model, and actual result.
- Keep score

Thank You