INFO 215: Web Science.
Lecture 02: Web Architecture, Server side development with Django

- 23 January, 2023

- *Fazle Rabbi, Ph.D*
- *Assoc. Prof in Information Science*
- *University of Bergen*
- *Fazle.Rabbi@uib.no*
- *Bergen, Norway*

Learning Objectives:

- Theoretical knowledge:
- What is Web Service? What are the benefits of using WebService ?
- What do we mean by service oriented architecture (SOA)?
- What is RestAPI ?
- How does MVC framework work ?
- How does Django MVT work ?

- Practical knowledge:
- How to setup a django server?
- How to create a web project in Django?
- How to create an application in Django?
- How to develop static page/dynamic page/service in Django?

WebService

- [Definition]

- A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.
- It has an interface described in a machine-processable format (using XML). Other systems interact with the Web service in a manner prescribed by its description, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

- Definition from W3C

WebService

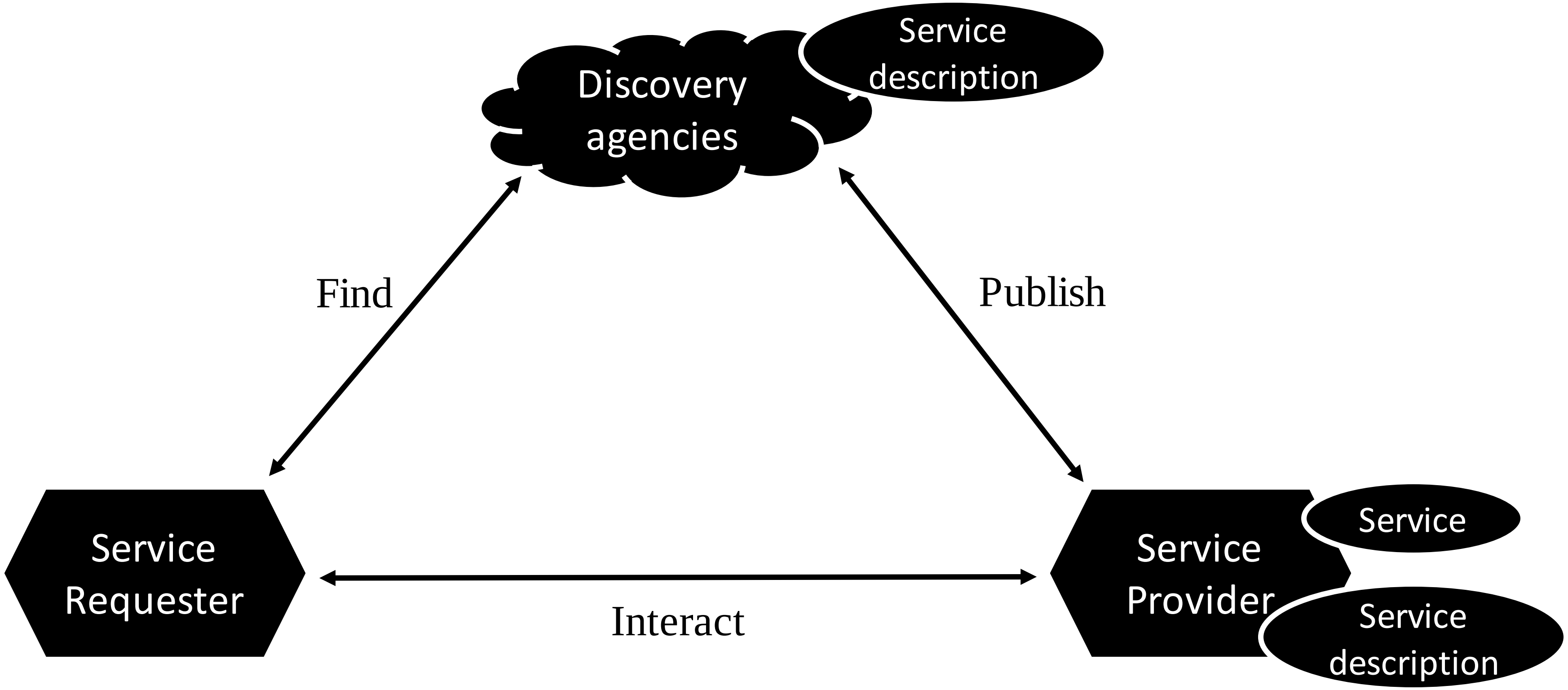
- [Purpose]

- Interoperability: It support interoperable machine-to-machine interaction over a network. Therefore, it is possible to develop cross platform applications. For example, A .NET application can talk to a Java application or vice versa.

- Availability: A web service is a unit of managed code that can be remotely invoked using HTTP requests. In this way we can expose functionality to other applications/systems.

Service Oriented Architecture

- There are 3 major roles within the web service architecture:
- Service Provider: The service provider implements the service and makes it available on the internet.
- Service Requester: A consumer of a web service utilizes an existing web service by opening a network connection and sending an XML request.
- Service Registry: This is a centralized directory for services. Developers can publish new services and find existing ones.



What is an API ?

"An API (Application Programming Interface) is a set of routines, protocols and tools for building software applications" - wikipedia

APIs are often considered as machine readable interfaces.

An WebAPI is an HTTP based API that allows a programmatic access to data and platform.

Useful link: https://www.programmableweb.com/

REpresentational State Transfer (REST) API

- REST is an architecture style which is based on Web-standard and HTTP protocol.
- The key abstraction of information in REST is a resource.

- A resource is a conceptual mapping to a set of entities
  - Any information that can be named can be a resource: a document or image, a temporal service (e.g. "today's weather in Los Angeles"), a collection of other resources, a non-virtual object (e.g. a person), and so on

- REST uses URI to identify resources

REpresentational State Transfer (REST) API

Relevant terminology from database concepts: CRUD

CREATE
READ
UPDATE
DELETE

- HTTP GET, POST, PUT, DELETE methods are used to represent the actions to be performed on resources.
- HTTP GET: GET request transfers the resource identified by the URI from the server to the client
- HTTP POST: Creates a resource identified by the URI
- HTTP PUT: Updates a resource identified by the URI
- HTTP DELETE: Removes the resource identified by the URI

REpresentational State Transfer (REST) API

- JSON and XML formats are used to return resource to the client.
- It is common to have multiple representations of the same resource.

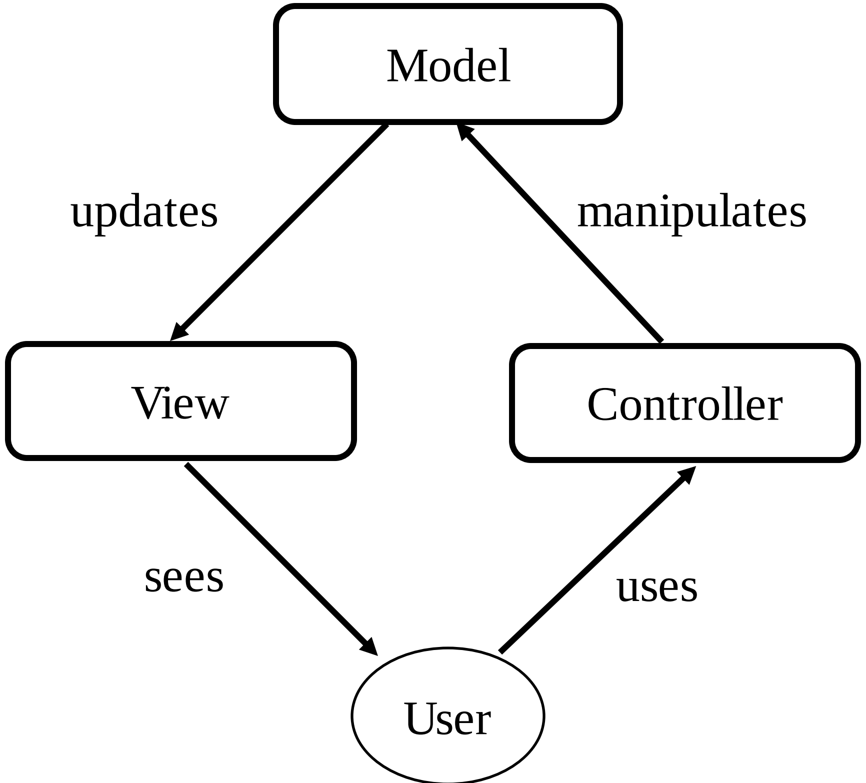
- Homework: Try getting message from the following news provider using REST API:

- https://newsapi.org/

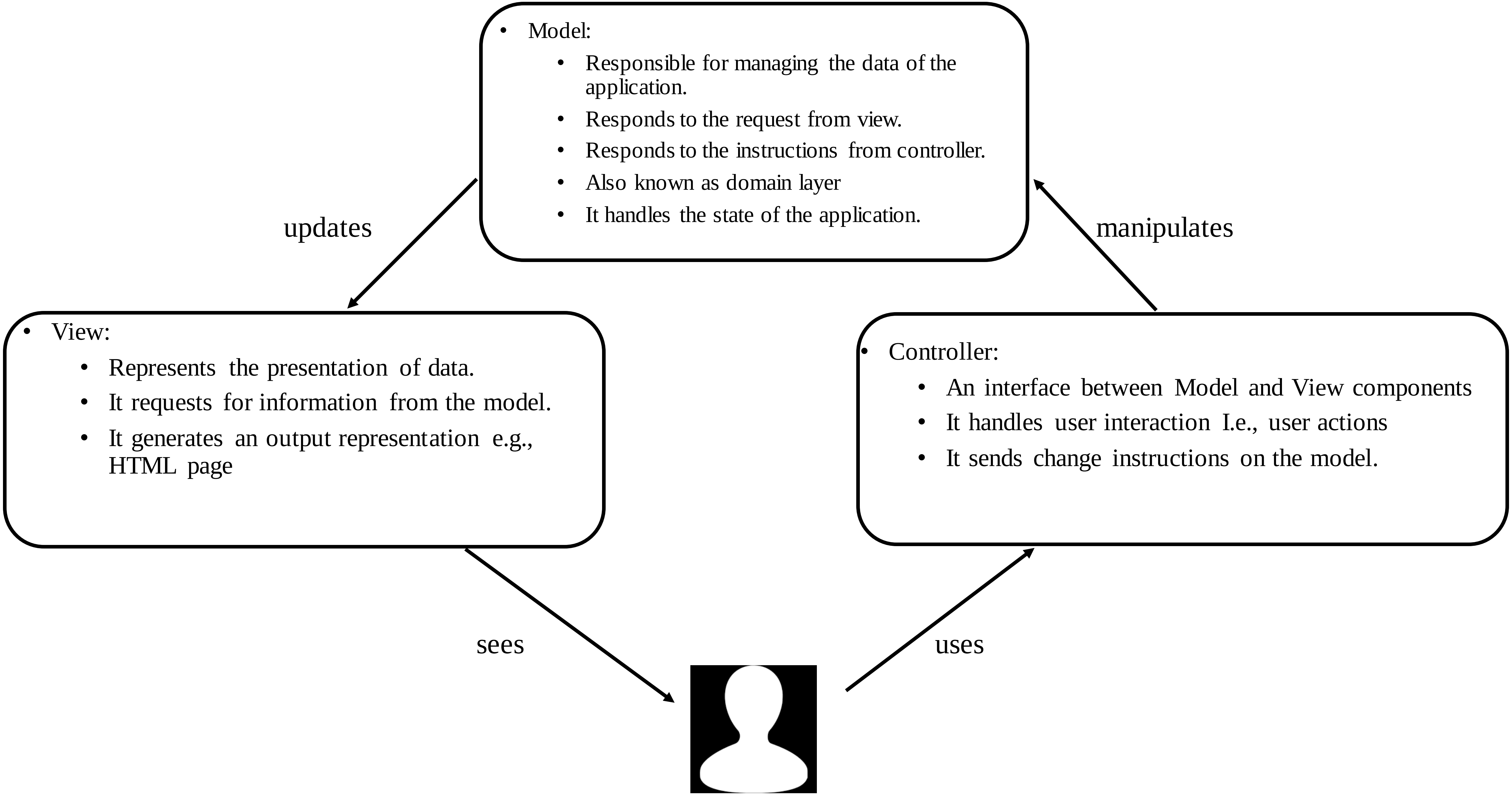
- Useful links: https://www.programmableweb.com/

Model View Controller (MVC)

- Model View Controller (MVC) is a software design pattern for developing web applications.
- MVC pattern separates the representation of information from users interaction.
- It is made up of 3 parts:
  - Model
  - View
  - Controller



Model View Controller (MVC)

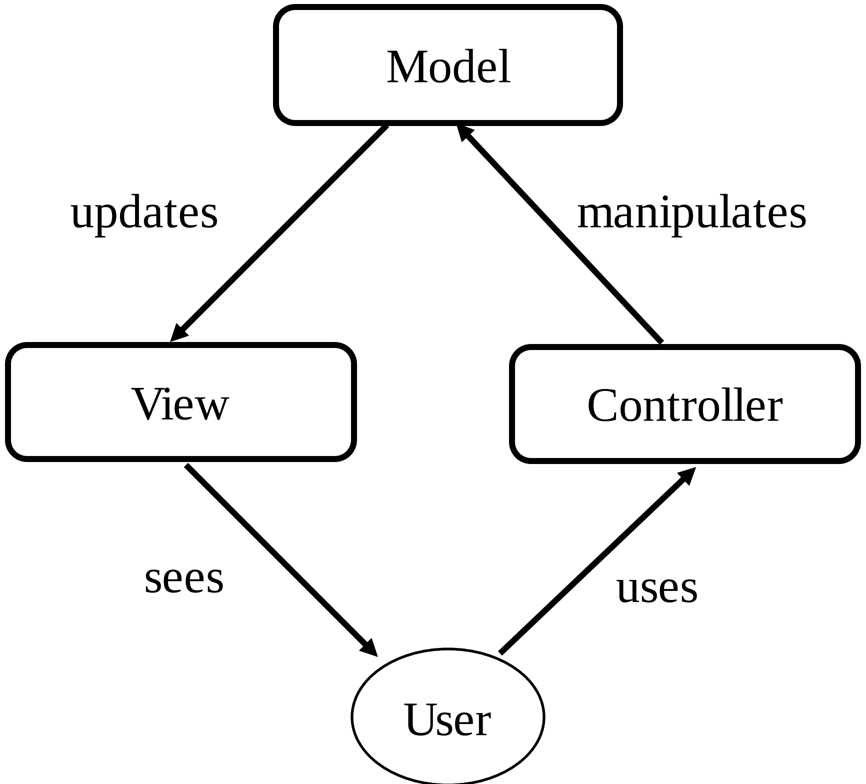


- Model:
  - Responsible for managing the data of the application.
  - Responds to the request from view.
  - Responds to the instructions from controller.
  - Also known as domain layer
  - It handles the state of the application.

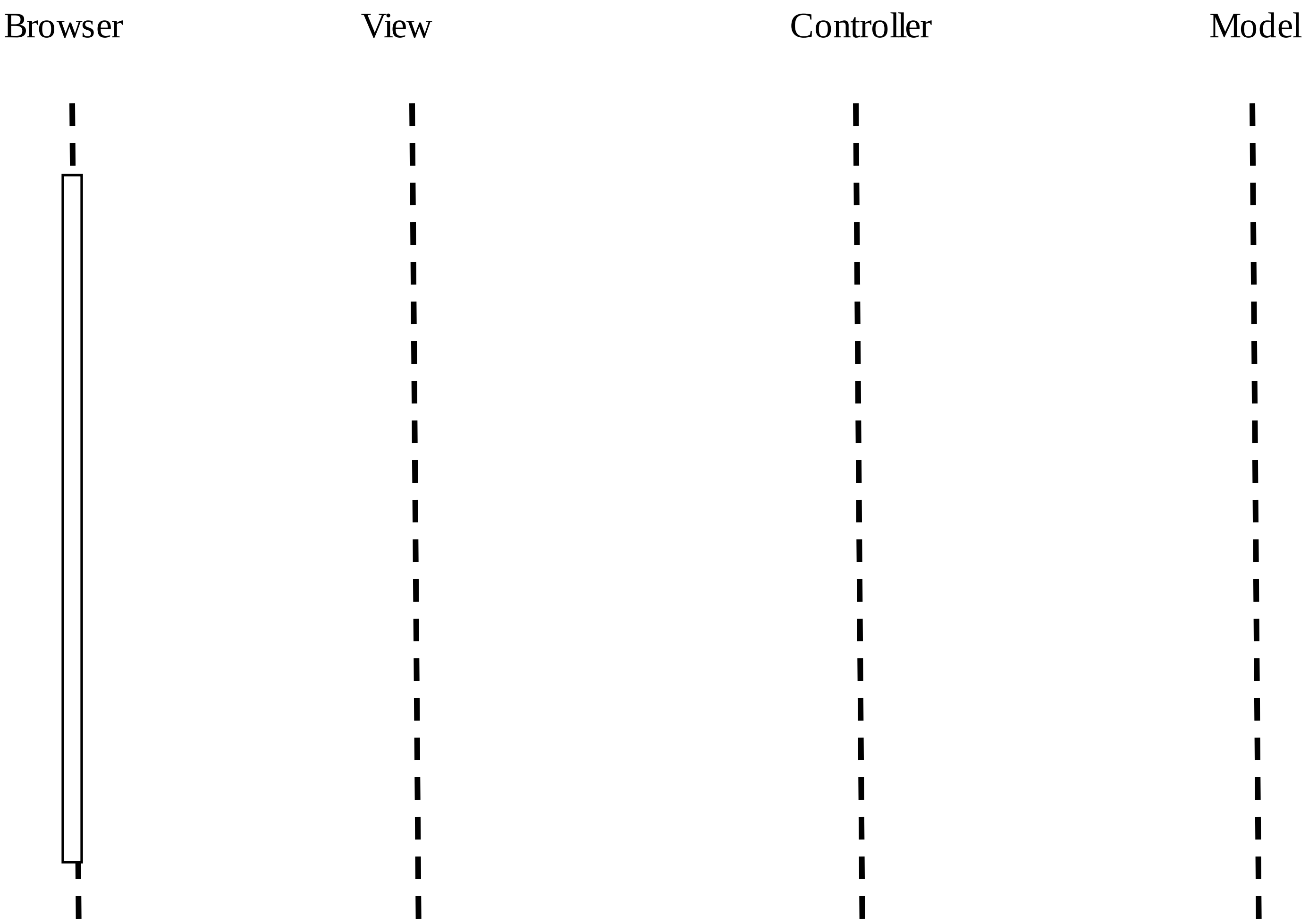
- View:
  - Represents the presentation of data.
  - It requests for information from the model.
  - It generates an output representation e.g., HTML page

- Controller:
  - An interface between Model and View components
  - It handles user interaction I.e., user actions
  - It sends change instructions on the model.

Model View Controller (MVC)

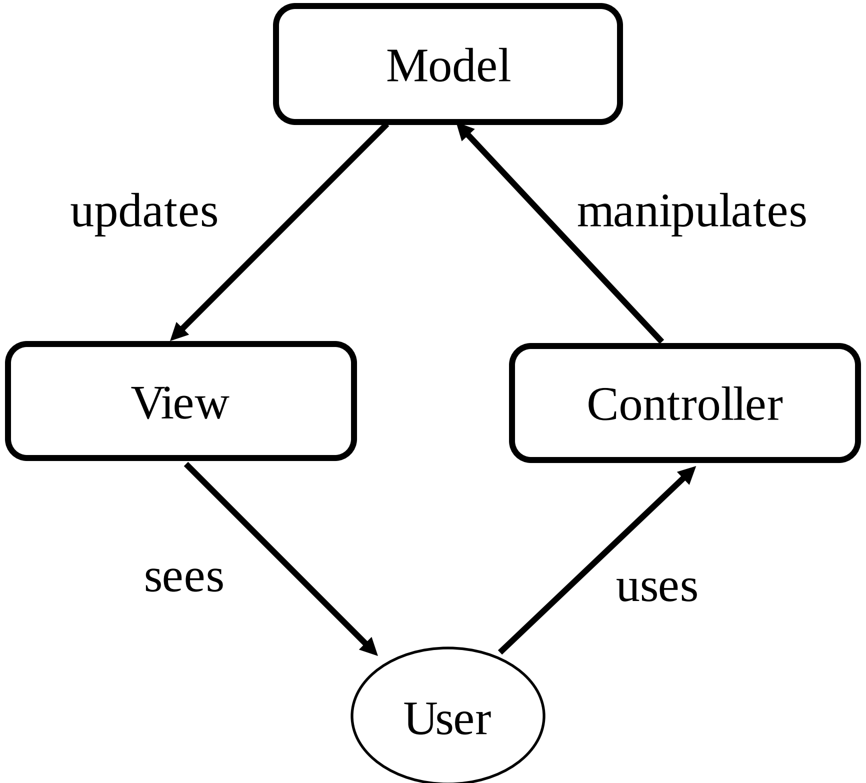


Example: a HTTP request. A sequence diagram of a single request/response pair.

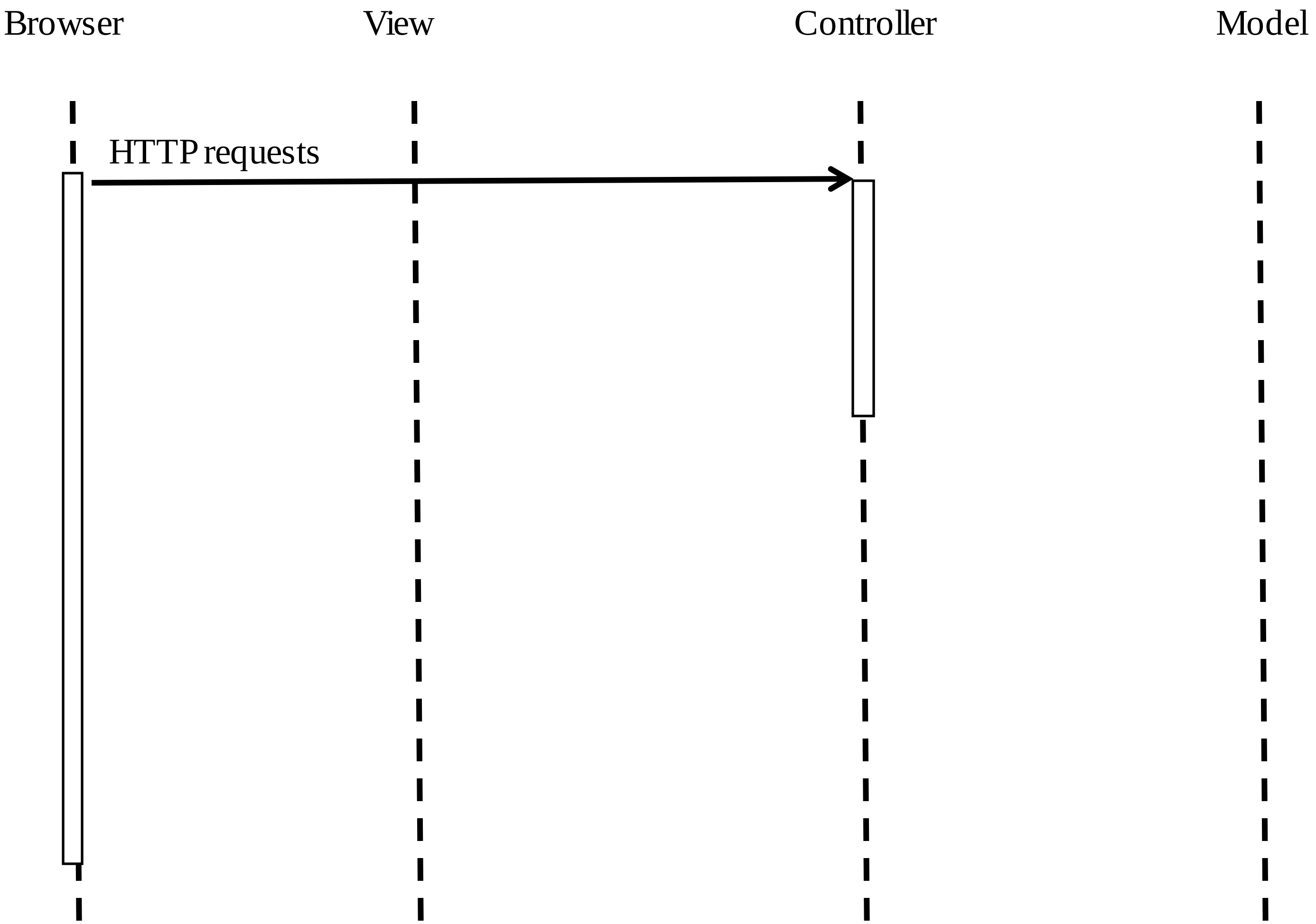


Notice that the Controller does not handle the communication between the View and the Model: the Views make direct requests to the Model.

Model View Controller (MVC)

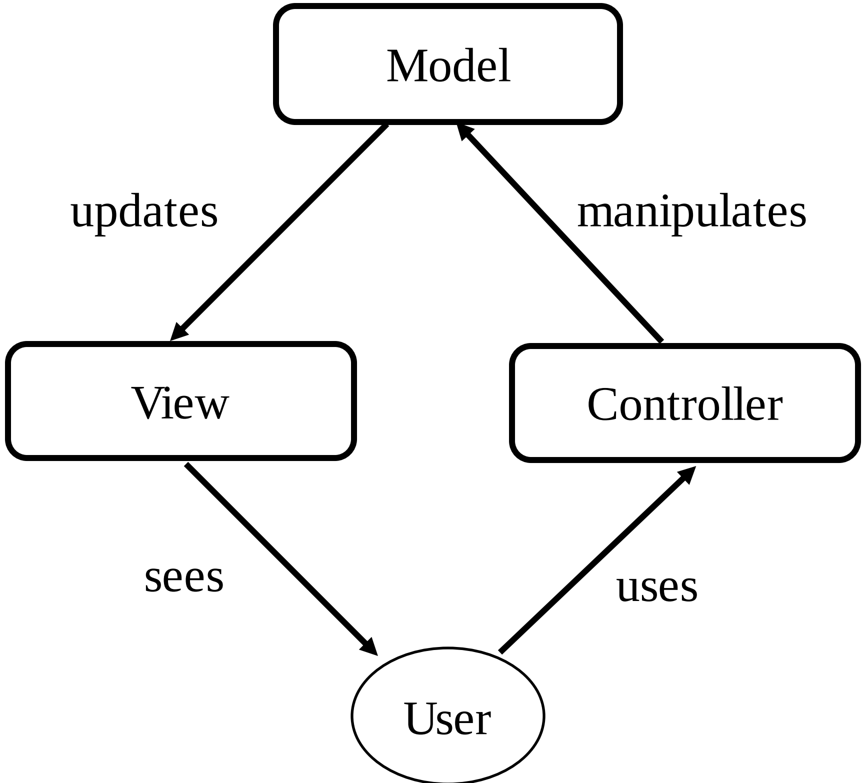


Example: a HTTP request. A sequence diagram of a single request/response pair.

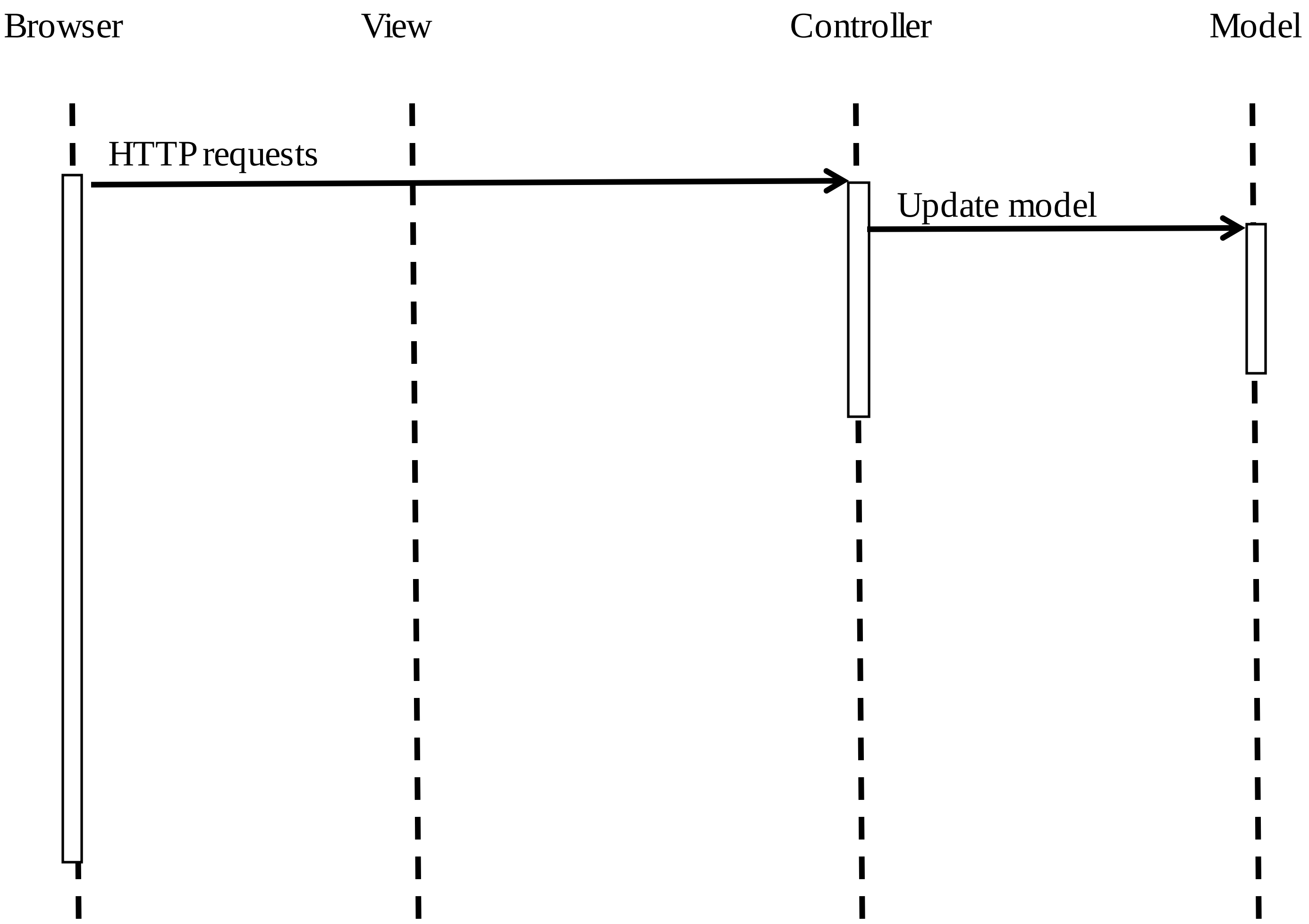


Notice that the Controller does not handle the communication between the View and the Model: the Views make direct requests to the Model.

Model View Controller (MVC)

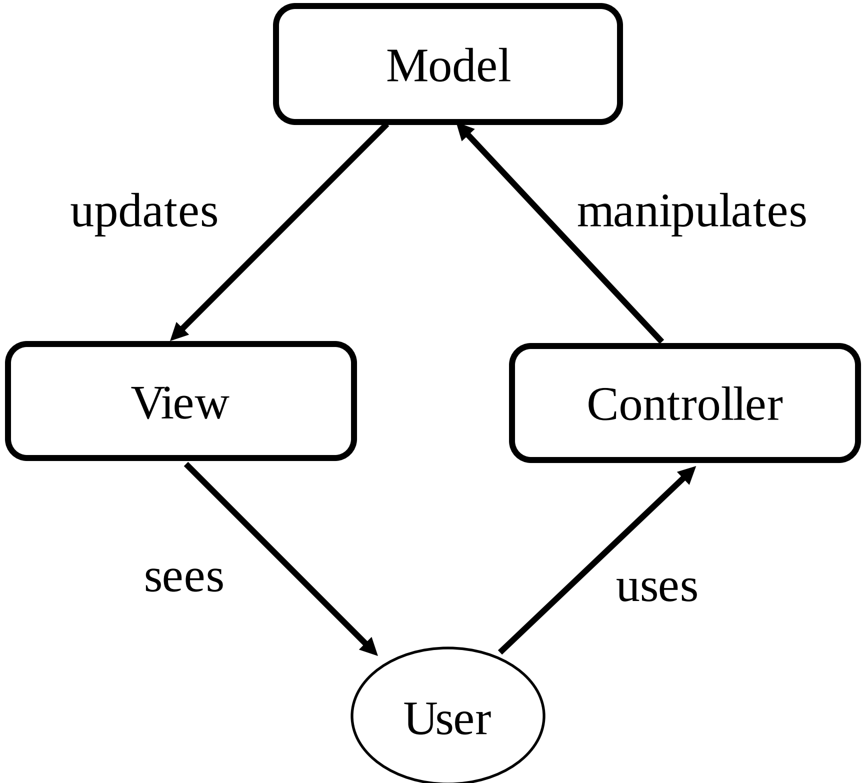


Example: a HTTP request. A sequence diagram of a single request/response pair.

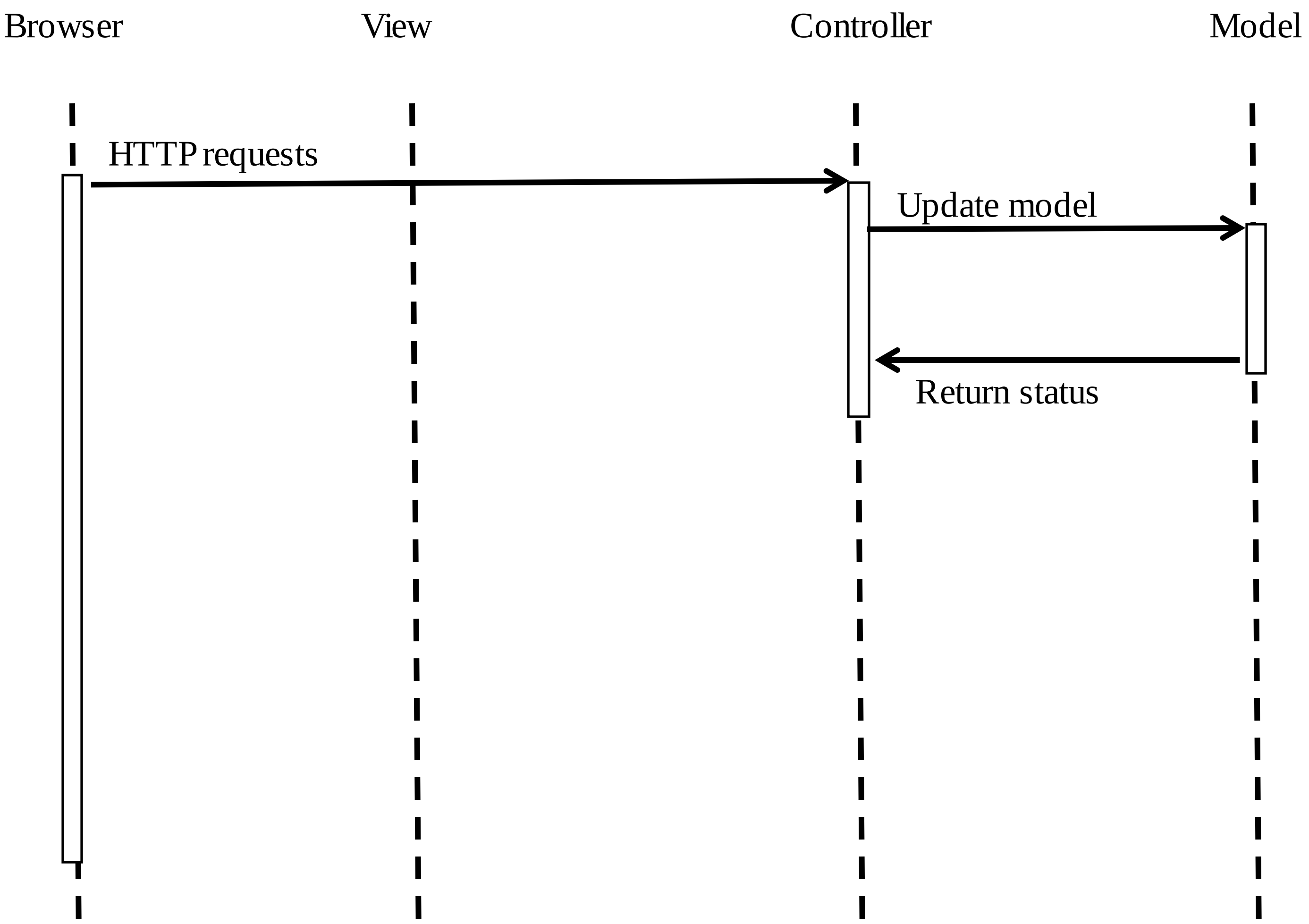


Notice that the Controller does not handle the communication between the View and the Model: the Views make direct requests to the Model.

Model View Controller (MVC)

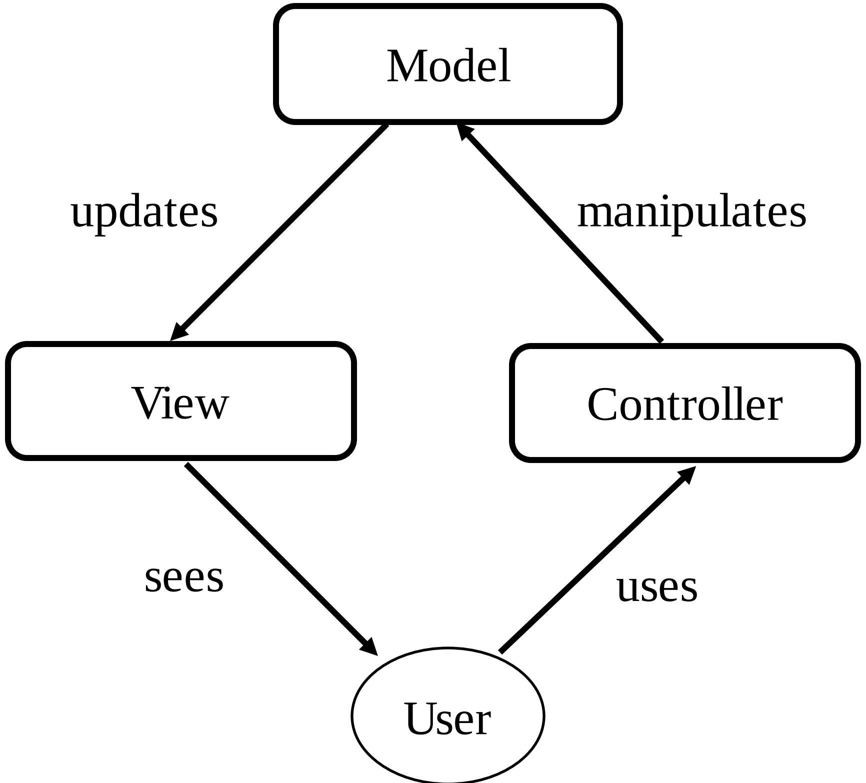


Example: a HTTP request. A sequence diagram of a single request/response pair.

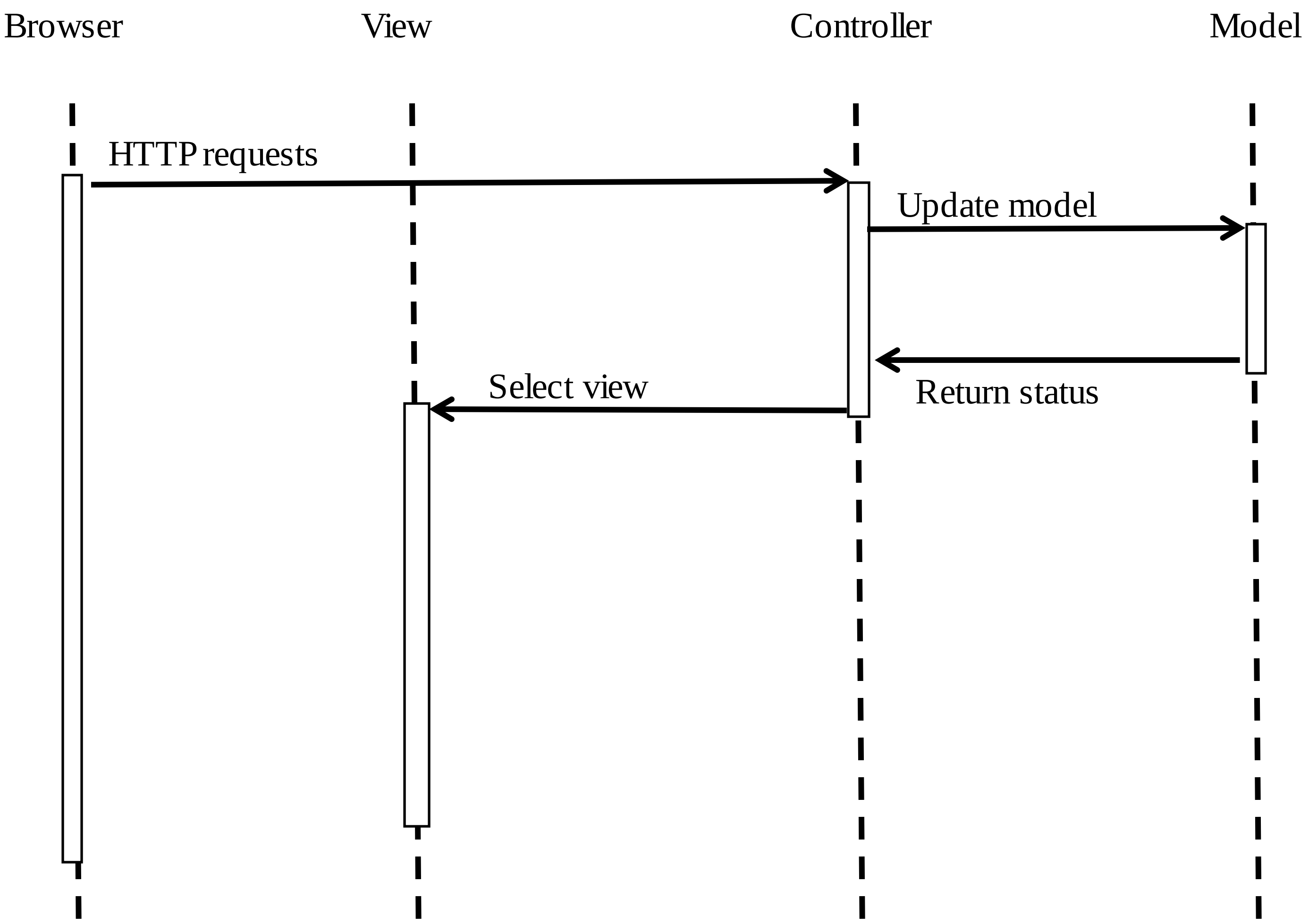


Notice that the Controller does not handle the communication between the View and the Model: the Views make direct requests to the Model.

Model View Controller (MVC)

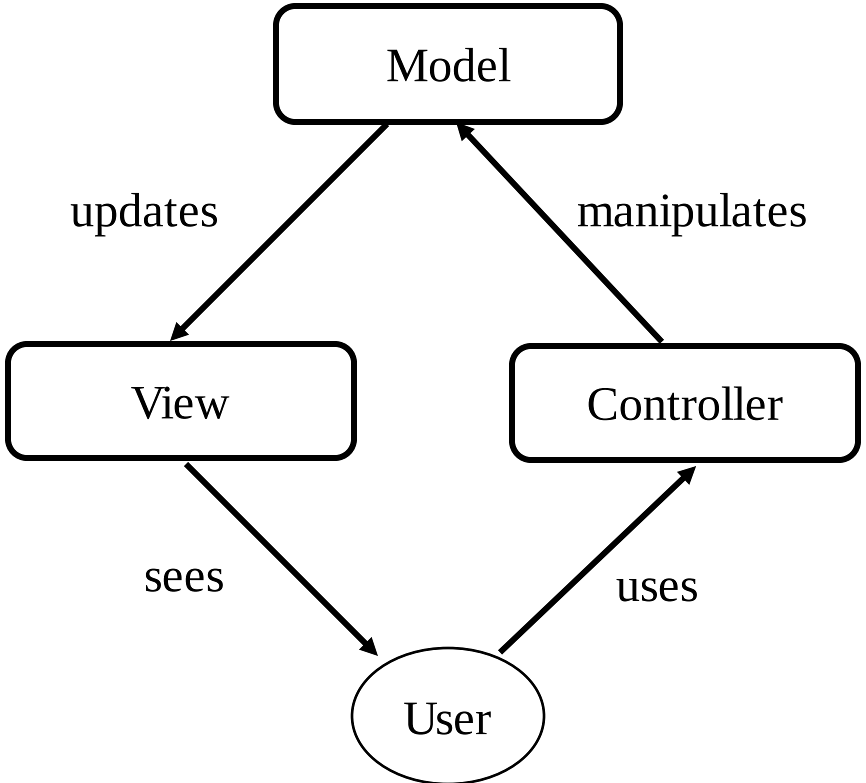


Example: a HTTP request. A sequence diagram of a single request/response pair.

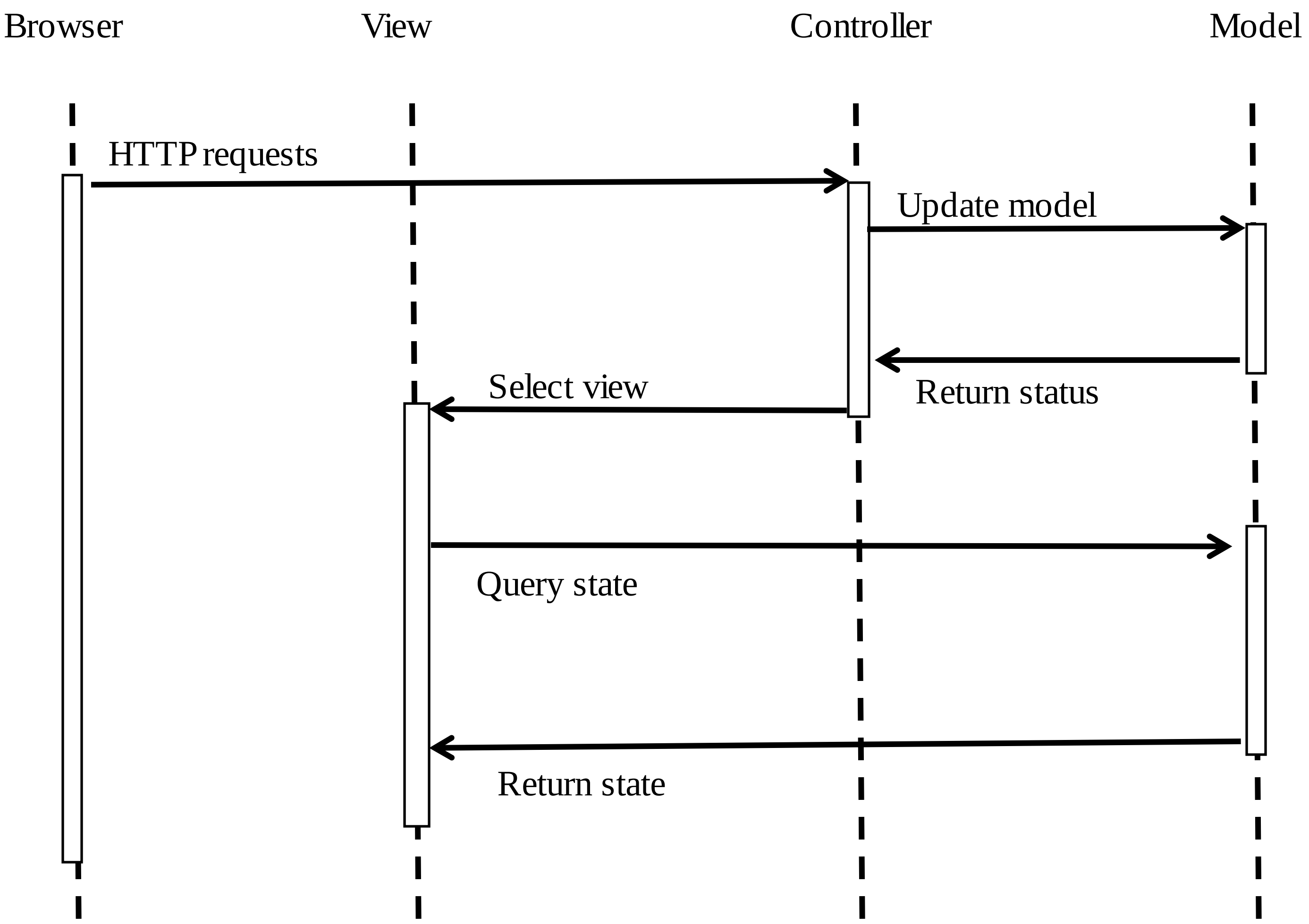


Notice that the Controller does not handle the communication between the View and the Model: the Views make direct requests to the Model.

Model View Controller (MVC)

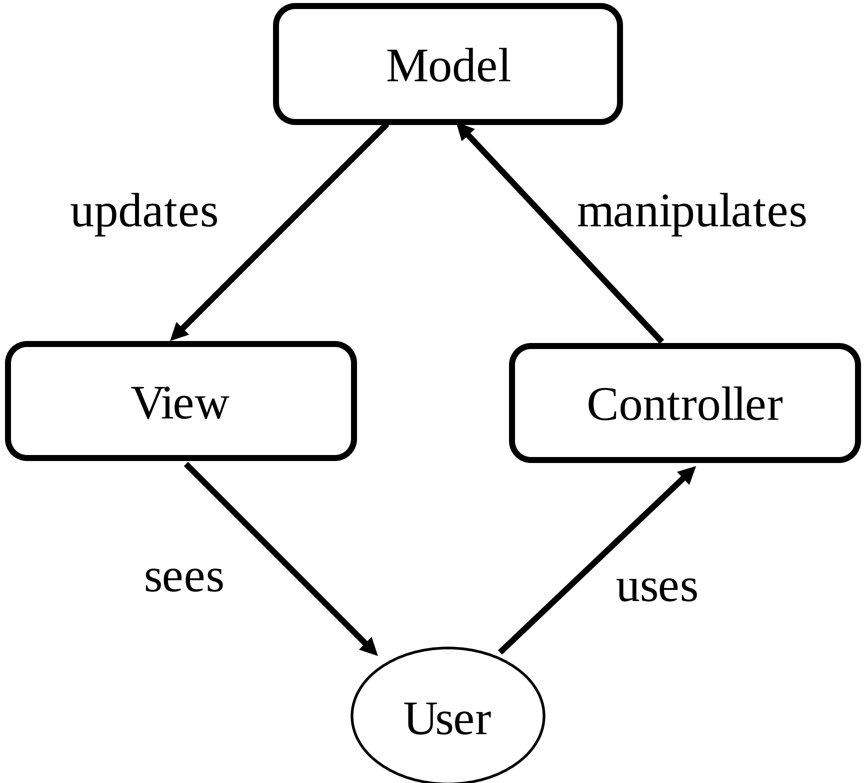


Example: a HTTP request. A sequence diagram of a single request/response pair.

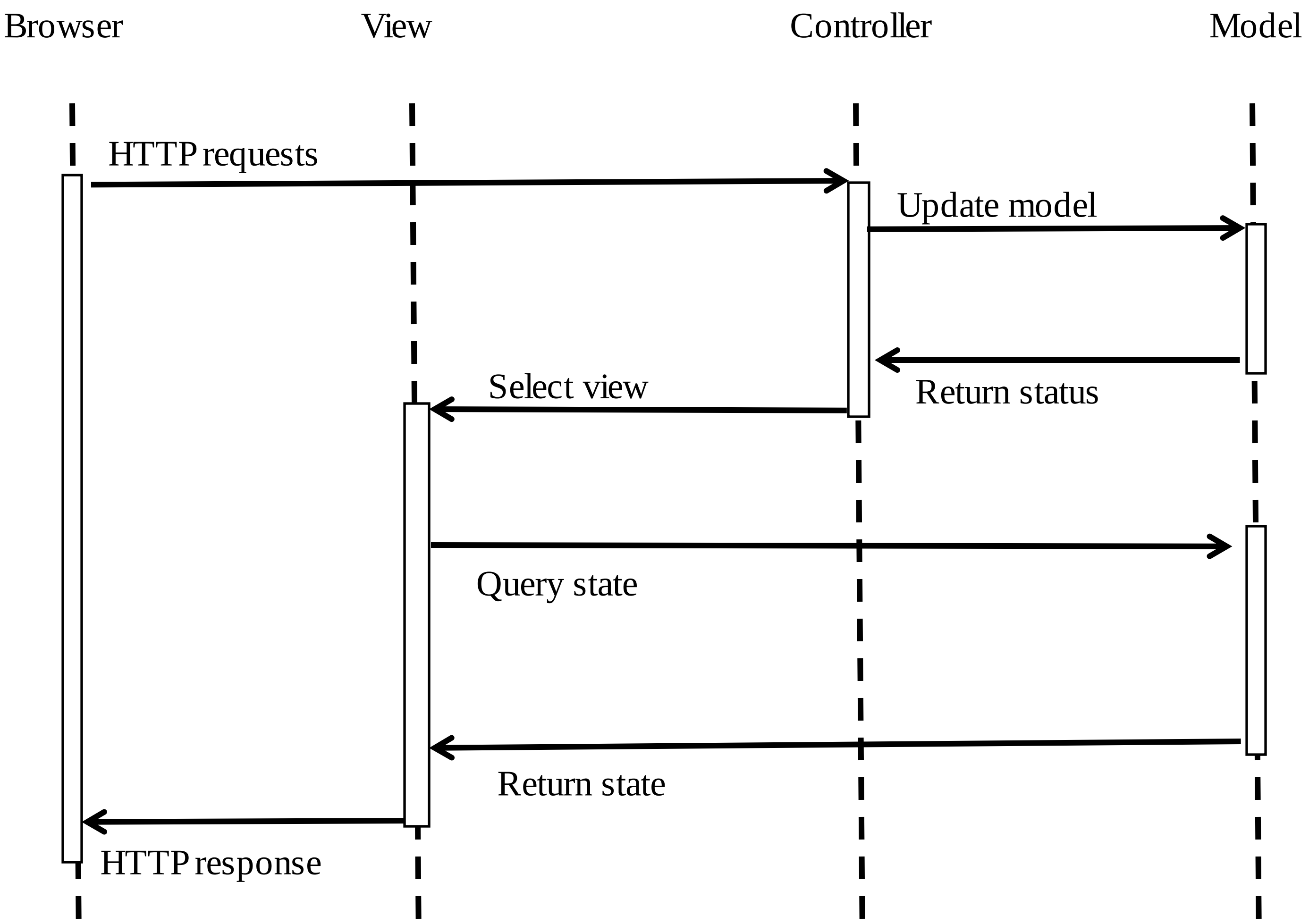


Notice that the Controller does not handle the communication between the View and the Model: the Views make direct requests to the Model.

Model View Controller (MVC)



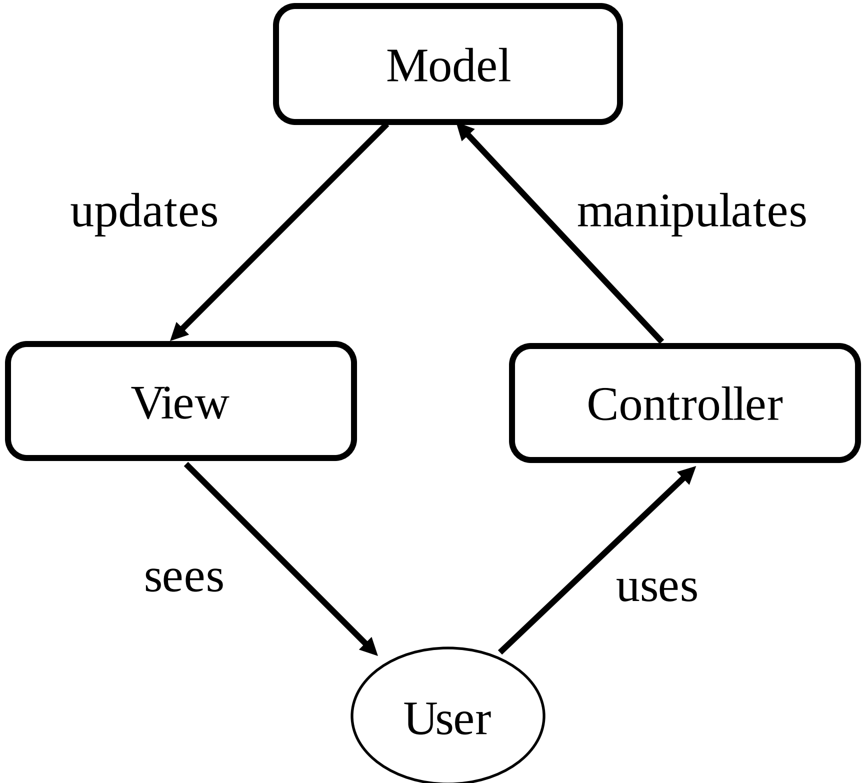
Example: a HTTP request. A sequence diagram of a single request/response pair.



Notice that the Controller does not handle the communication between the View and the Model: the Views make direct requests to the Model.

Model View Controller (MVC)

- Advantages:
  - Clear separation between presentation logic and business logic.
  - Parallel development of application components.
  - Easy to maintain and future enhancement.
  - Other advantages? [Homework!]



Server-side website programming using Django

- Django is an open-source web framework built with the Python language that aims for flexibility, simplicity, scalability, and reliability.
- Django is a widely chosen framework - and its popularity is still growing, mainly due to the increasing use of Python.

INSTALLING PYTHON AND DJANGO

- THREE STEP PROCESS:
  - INSTALL PYTHON
  - INSTALL VIRTUAL ENVIRONMENT
  - INSTALL DJANGO

NOTE THAT DJANGO IS NOT COMPATIBLE WITH PYTHON 2, THEREFORE YOU HAVE TO USE PYTHON 3.

DOWNLOAD LINK FOR PYTHON 3: https://www.python.org/downloads/

Need for creating a Python virtual environment

- When you are working with multiple projects, it is common that you need to use different versions of software as the projects may have a variety of dependencies.
- This can cause many problems, so we need a way to avoid these issues.
- Python virtual environment provides a way to solve these problems as it allows us to wrap all the dependencies and environment variables that your projects need into a filesystem.
- The virtual environment in python is called venv.

Creating a Python Virtual Environment

- The following commands are used to create a virtual environment:

```
> python3 –m virtualenv django-test
> cd django-test
> source bin/activate
```

Tips:
if 'python3 –m virtualenv django-test' command does not work for you try the following:
'python3 -m venv django-test

How to deactivate a virtual environment?
The following command should do it.

```
> deactivate
```

Installing Django

- To install Django use the following command: (Inside your virtual environment)

```
> pip3 install django
```

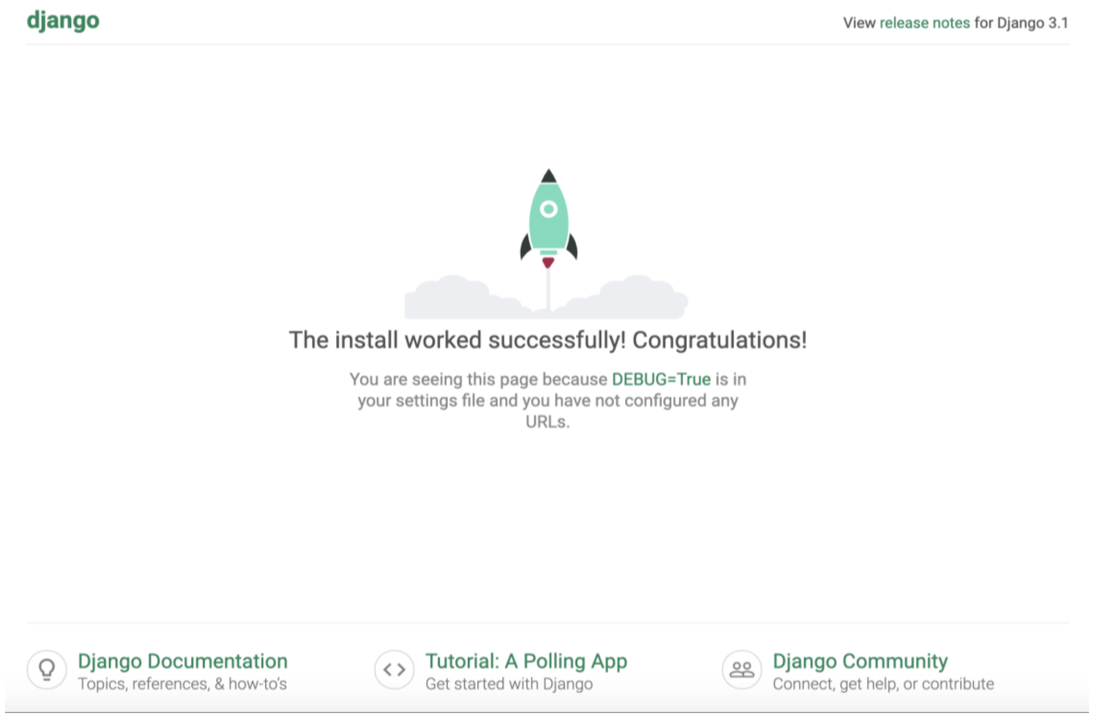
To check which django version has been installed, use the following command:

```
> python3 -m django --version
```

Starting a project

- A Django project is a collection of settings and files for a single Django website.
- To create a new Django project, we’ll be using a special command to auto-generate the folders, files and code that make up a Django project.
- This includes a collection of settings for an instance of Django, database configuration, Django-specific options and application-specific settings.

Putting all commands together:: Commands to create a django project and runserver



django
View release notes for Django 3.1

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Django Documentation
Topics, references, & how-to's
Tutorial: A Polling App
Get started with Django
Django Community
Connect, get help, or contribute

- python3 –m virtualenv django-test
- cd django-test
- source bin/activate
- pip3 install django
- django-admin startproject mysite
- cd mysite
- python3 manage.py runserver

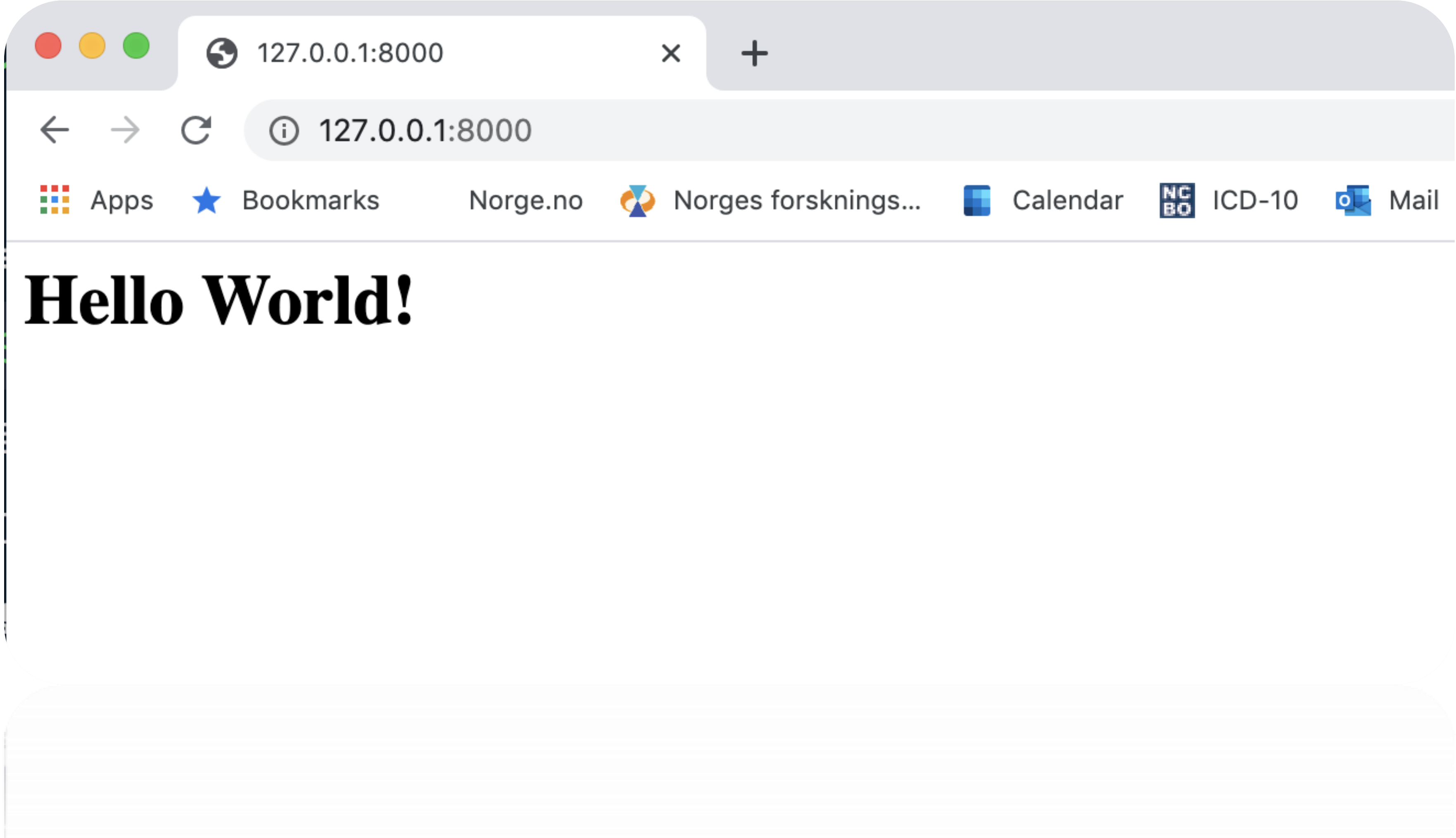
This will start the Django development server.
Django development server is a lightweight Web server written in Python.
If the server runs successfully, you should see the above page at: http://127.0.0.1:8000/
The 8000 on the end is telling you that Django is listening at port 8000 on your local host.

Django directory structure:

The command 'django-admin startproject mysite' created the following files and directories:

```
mysite/                 <----- The outer mysite/ root directory is a container for your project.
   manage.py            <----- A command-line utility that lets you interact with this Django project.
    mysite/
        __init__.py
        settings.py.    <----- Settings/configuration for this Django project.
        urls.py            <----- The URL declarations for this Django project
        asgi.py
        wsgi.py
```

Static page: This is what we want to build!



Hello World!

Static page: step by step instruction!

Step1:

Setup a new application inside your django project.

The command to create a new django application are shown on the right side:

A Django project may contain 1 or more applications. To create an application inside 'mysite' project, use the following commands:

```
> cd mysite
> python3 manage.py startapp myapp
```

Django directory structure for applications:

```
> cd mysite
> python3 manage.py startapp myapp
```

This command creates these files in mysite/myapp

```
__init__.py
admin.py
apps.py
migrations
models.py
tests.py
views.py
```

admin.py is where you register your app’s models with the Django admin application.

apps.py is a configuration file common to all Django apps.

The migrations folder is where Django stores migrations, or changes to your database.

models.py is the module containing the models for your app.

tests.py contains test procedures that run when testing your app.

views.py is the module containing the views for your app.

Static page: step by step instruction!

Step2:

Tell your project that about the app.

Edit the settings.py file as shown on the right:

```
# mysite/settings.py
INSTALLED_APPS = 
[
    ……..
    'myapp',
]
```

Static page: step by step instruction!

Step3:

Tell your app what should be included in the view by editing views.py file as shown on the right:

```
# myapp/views.py


from django.shortcuts import render
from django.http import HttpResponse


# Create your views here.
def index(request):
    return HttpResponse('<h1> Hello World </h1>')
```

Static page: step by step instruction!

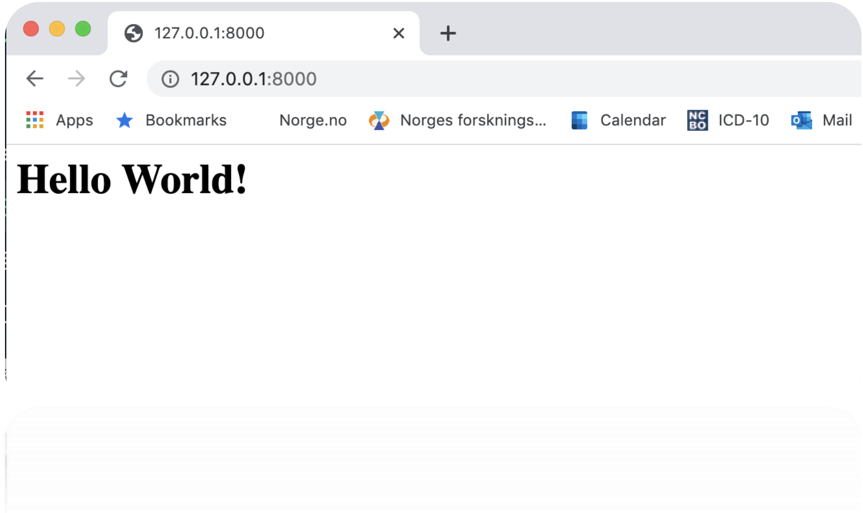
Step4:

Tell your project how the direct the incoming http request to your view in the app
by editing urls.py file as shown on the right:

mysite/urls.py

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



urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index, name='index'),
]
```

Static page: Putting them all together!



127.0.0.1:8000

Hello World!

Step1:

Setup a new application inside your django project.

Step2:

Tell your project that about the app by editing the settings.py file

Step3:

Tell your app what should be included in the view by editing views.py file

Step4:

Tell your project how to direct the incoming http request to your view in the app by editing urls.py file

Static page:

How can we use a html file in Django project?

We must use a template!

Static page with template: step by step instruction!

Step1:

You will need some html files. We stored them in a directory called 'templates'

mysite/templates/index.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
    <meta charset="utf-8">
    <title>Intro to Templates</title>
</head>
<body>
    <h1>Hello World (index.html)</h1>
</body>
</html>
```

Static page with template: step by step instruction!

Step2:

Tell your app what should be included in the view by editing views.py file as shown on the right:

Here we specified the name of the html file which will be used for rendering.

```
# myapp/views.py


from django.shortcuts import render

from django.http import HttpResponse

# Create your views here.

def index(request):

   return render(request, 'index.html', {})
```

Static page with template: step by step instruction!

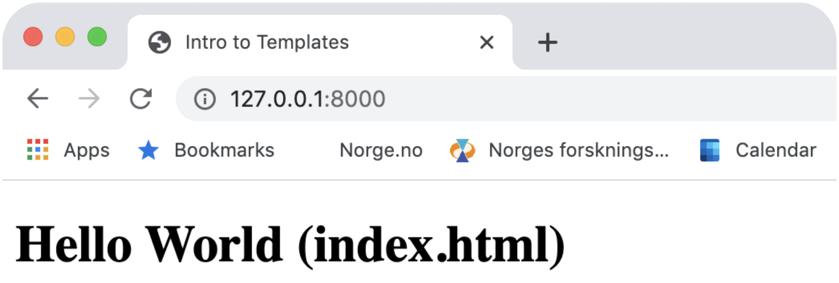
Step3:

Tell your project that about templates.

Specify the directory of template where html files are stored by editing the settings.py file as shown on the right:

```
# mysite/settings.py
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        .....
```

Static page with template: Putting them all together!



Hello World (index.html)

<u>Step1:</u>

You will need some html files.

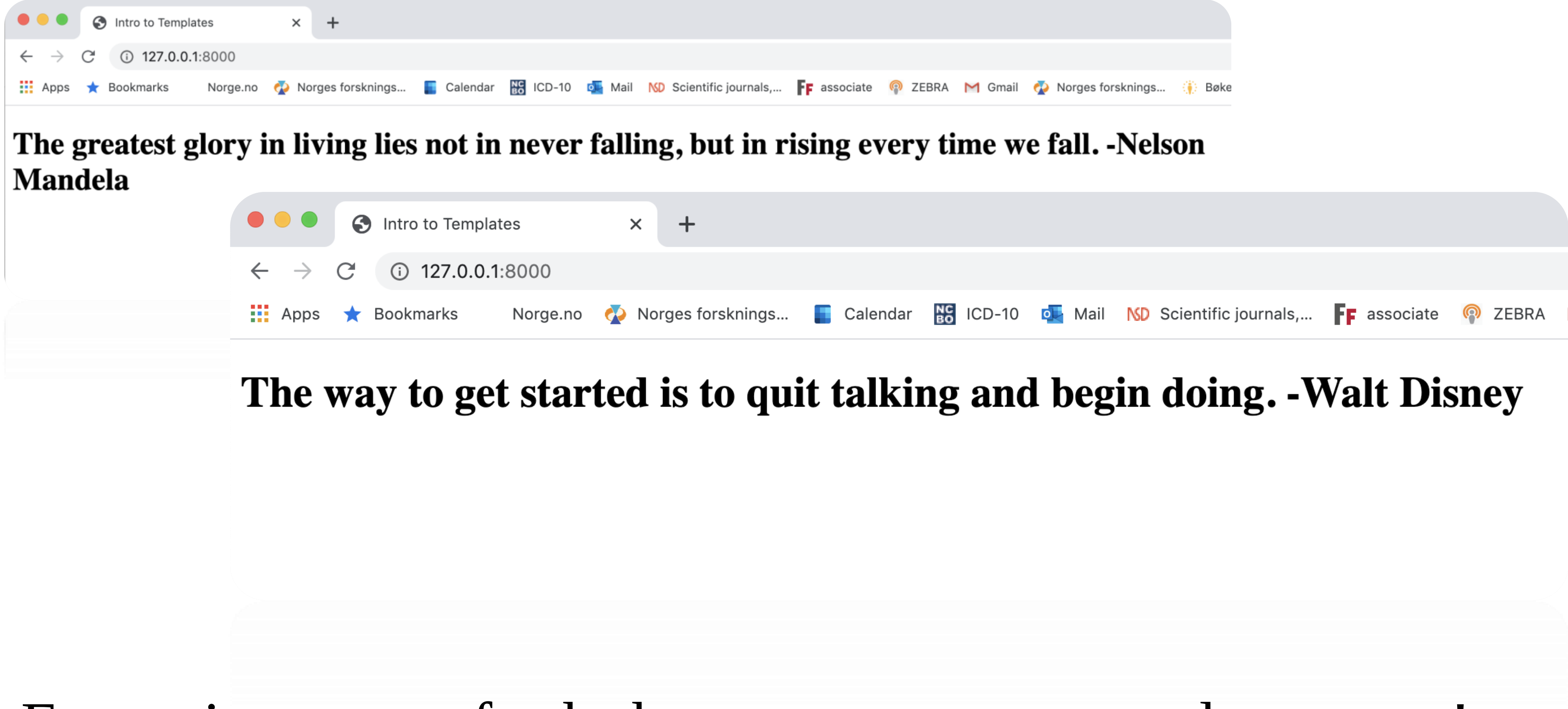
<u>Step2:</u>

Tell your app what should be included in the view by editing views.py file i.e., specify the name of html files.

<u>Step3:</u>

Tell your project about templates by specifying the directory of template where html files are stored by editing the settings.py file

Dynamic page with Django!
This is what we want to develop.



The greatest glory in living lies not in never falling, but in rising every time we fall. -Nelson Mandela

The way to get started is to quit talking and begin doing. -Walt Disney

Every time you refresh the page, you get a random quote!

Dynamic page with template: step by step instruction!

Step1:

Present dynamic content in a html file by using the following syntax: {{ }}.

The {{ }} expression inside this quote will be evaluated when the page is rendered.

mysite/templates/index.html

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
   <meta charset="utf-8">
   <title>Intro to Templates</title>
</head>
<body>
   <h1>{{ quote }} </h1>
</body>
</html>
```

Dynamic page with template: step by step instruction!

Step2:

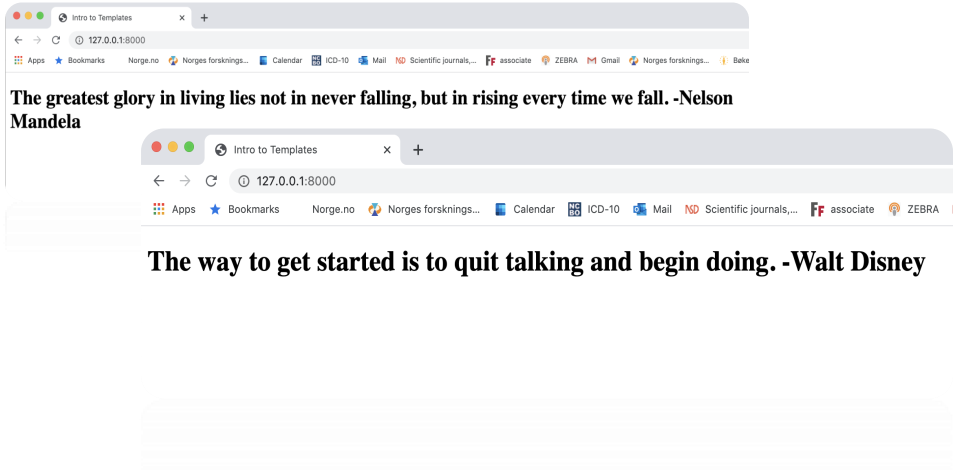
Tell your app what should be included in the view by editing views.py file as shown on the right:

Here we specified the name of the html file which will be used for rendering.

Also the dynamic content is sent as a parameter.

```
# myapp/views.py

from django.shortcuts import render

from django.http import HttpResponse

import random


def index(request):

    quotes = ['The greatest glory in living  lies not in never falling,  but in rising every time we fall. -Nelson Mandela',

        'The way to get started is to quit talking and begin doing. -Walt Disney',

        'If life were predictable it would cease to be life, and be without flavor. -Eleanor Roosevelt',

        'If you set your goals ridiculously high and it\'s a failure, you will fail above everyone else\'s success. -James Cameron']

    quote = random.choice(quotes)

    context = {

        'quote': quote

    }

    return render(request, 'index.html', context)
```

Dynamic page with template: Putting them all together!



The greatest glory in living lies not in never falling, but in rising every time we fall. -Nelson Mandela

The way to get started is to quit talking and begin doing. -Walt Disney

Step1:

Present dynamic content in a html file by using the following syntax: {{ }}.

The {{ }} expression inside this quote will be evaluated when the page is rendered.

Step2:

Tell your app what should be included in the view by editing views.py file.

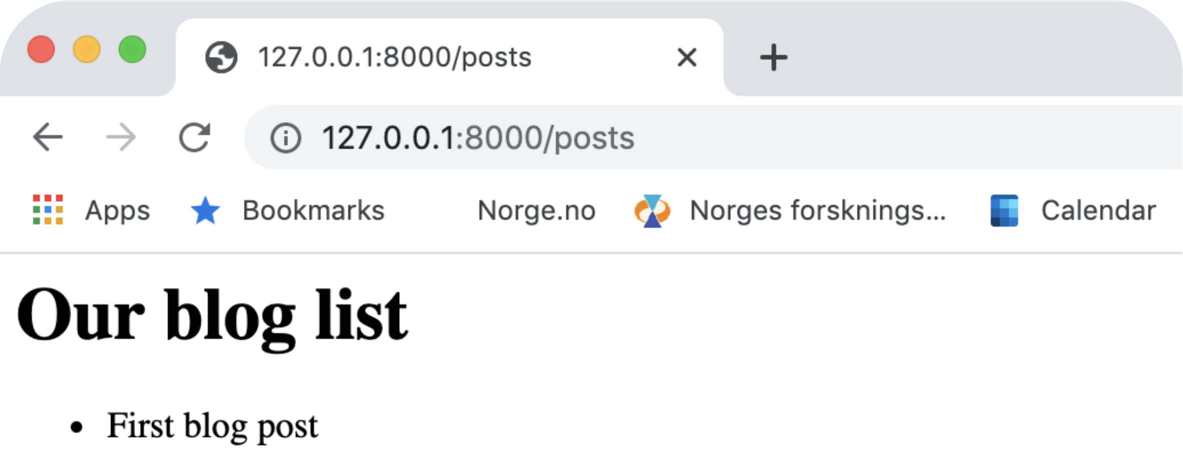
We need to specify the name of the html file which will be used for rendering.

The dynamic content is sent as a parameter.

Django Model View Template

- To-do list to create a Web app with Django Model View Template
  - Setup a new django-app
  - Create a model in the database that the Django ORM will manage
  - Setup the view
  - Setup the url

Dynamic page with Django MVT!
This is what we want to develop.



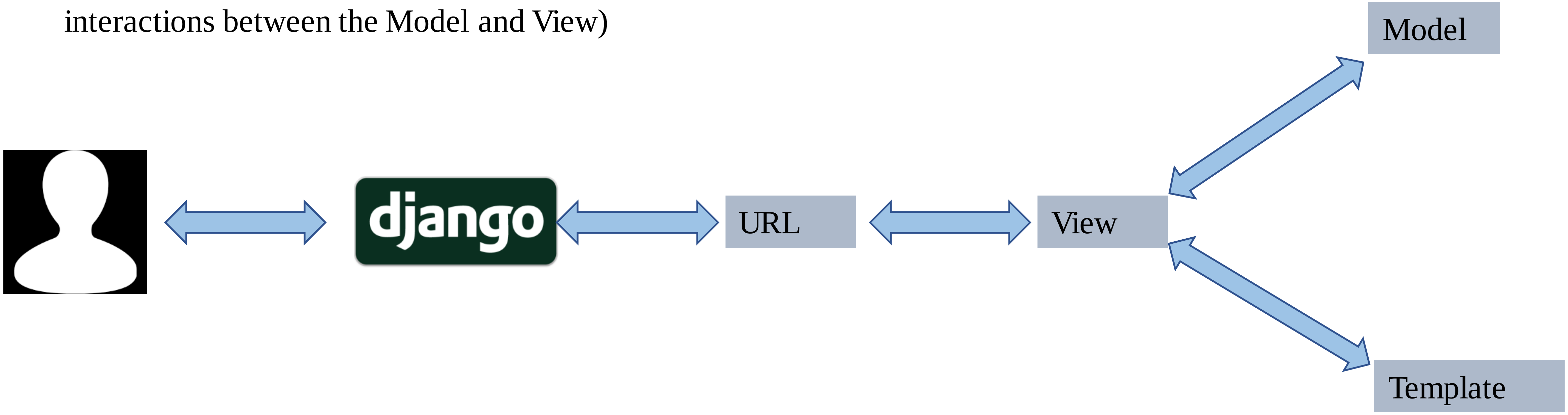
Our blog list

- First blog post

We create a webpage that shows blogpost entries from a database.

Django MVT!

MVT pattern is slightly different than MVC.
Django itself takes care of the Controller part (Software Code that controls the interactions between the Model and View)



This figure illustrates how each of the components of Django **MVT** framework interacts with each other to serve a user request.

Web app with Django MVT: step by step instruction!

Step1:

Setup a new Django app!

Web app with Django MVT: step by step instruction!

Step 1.1:

The command to create a new django application are shown on the right side:

To create an application inside 'mysite' project, use the following commands:

```
> cd mysite
> python3 manage.py startapp blog
```

Web app with Django MVT: step by step instruction!

Step 1.2:

Tell your project that about the app.

Edit the settings.py file as shown on the right:

```
# mysite/settings.py

INSTALLED_APPS = [

    …....

    'blog',

]
```

Web app with Django MVT: step by step instruction!

Step2:

Create a model in the database that the Django ORM will manage

Web app with Django MVT: step by step instruction!

Step 2.1:

Create a model by editing model.py file as shown on the right:

Django uses 'SQLite' as the default database. SQLite uses Object Relational Mapper (ORM). We can build model using Python code and the models are persisted in SQLite database by ORM.

```
# blog/models.py
from django.db import models


class Post(models.Model):
    title = models.CharField(max_length = 50)
    content = models.TextField()


    def __str__(self):
        return self.title
# title and content are attributes where we can store strings. The __str__ method
just tells Django what to print when it needs to print out an instance of the
Post model.
```

Web app with Django MVT: step by step instruction!

Step 2.2:

Making migrations I.e., synchronize models with database.

Following two commands create necessary database files by reading models.py file.

```
> python3 manage.py makemigrations
> python3 manage.py migrate
```

Web app with Django MVT: step by step instruction!

Step 2.3:

Create superuser for viewing administration panel.

```
> python3 manage.py createsuperuser
```

Web app with Django MVT: step by step instruction!

Step 2.4:

Register models for admin access by editing blog/admin.py file

```
# blog/admin.py
from django.contrib import admin
from .models import Post

# Register your models here.
admin.site.register(Post)
```

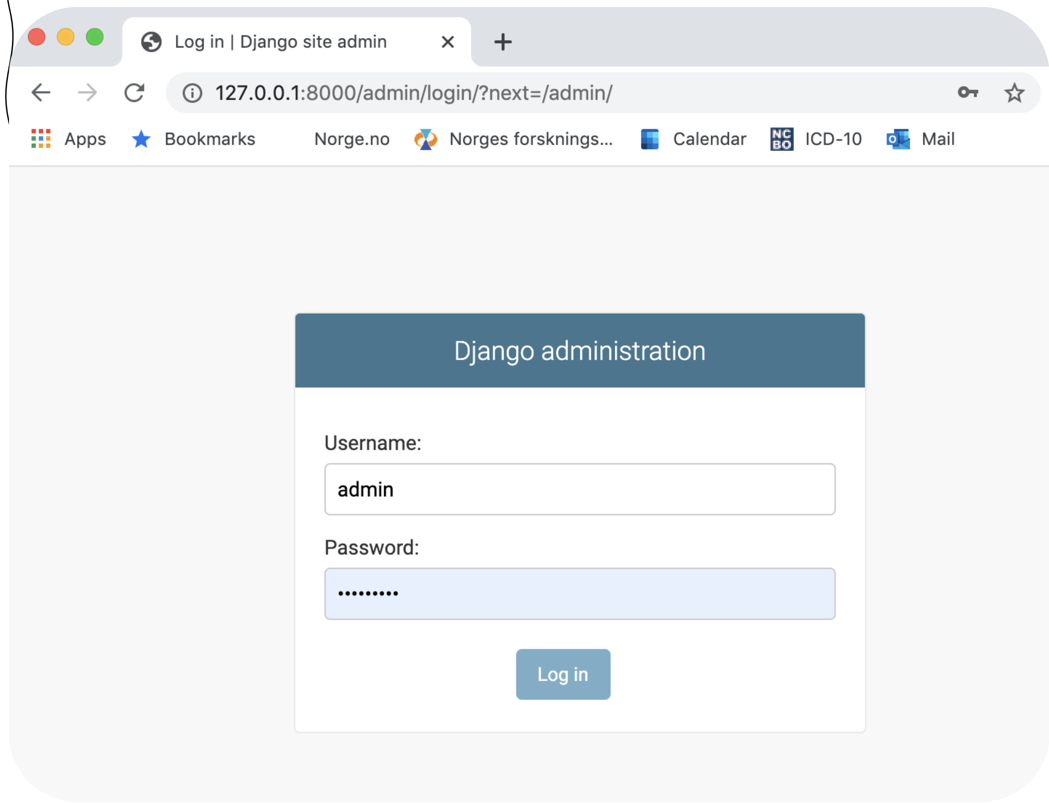
Web app with Django MVT: step by step instruction!

Step 2.5:

Run the server and open the admin panel.

The admin panel allows to edit data.

```
> python3 manage.py runserver
```



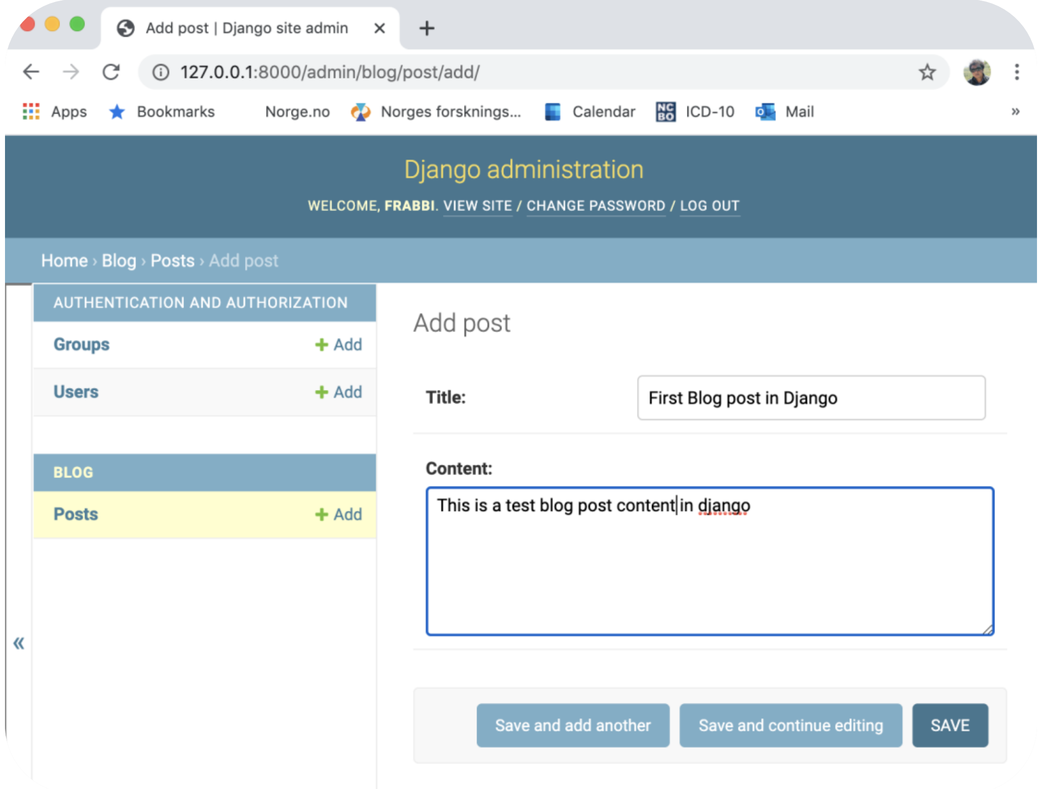
Log in | Django site admin

127.0.0.1:8000/admin/login/?next=/admin/

Django administration

Username: admin

Password: •••••••••



Add post | Django site admin

127.0.0.1:8000/admin/blog/post/add/

Django administration

WELCOME, FRABBI. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Blog › Posts › Add post

Add post

Title: First Blog post in Django

Content: This is a test blog post content in django

Web app with Django MVT: step by step instruction!

Step3:

Setup the view

Web app with Django MVT: step by step instruction!

Step 3.1:

Tell your app what should be included in the view by editing views.py file as shown on the right:

```
# blog/views.py


from django.shortcuts  import render
from .models import Post


# Create your views here.
def blog_post(request):
    post = Post.objects.all()
    context = {
        'blog_list' :post
    }
    return render(request,  "bloglist.html", context)
```

Web app with Django MVT: step by step instruction!

Step 3.2:

We present the model in html file.

We stored the html files in a directory called 'template'

```
#template/bloglist.html
<h1>Our blog list</h1>
<ul>
    {% for list in blog_list %}
        <li>{{list}}</li>
    {% endfor %}
</ul>
```

Web app with Django MVT: step by step instruction!

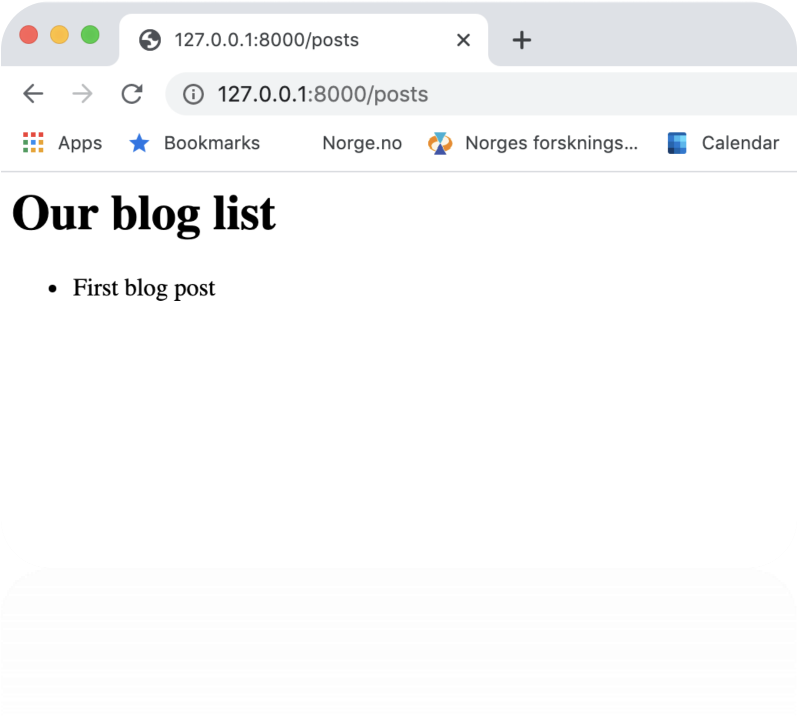
Step 4:

Tell your project how the direct the incoming http request to your view in the app by editing urls.py file as shown on the right:

mysite/urls.py

```
from django.contrib import admin
from django.urls import path, include
from blog.views import blog_post



urlpatterns = [
    path('admin/', admin.site.urls),
    path('posts', blog_post ),
]
```

Web app with Django MVT: step by step instruction!



Our blog list

- First blog post

Step1:

Setup a new application inside your django project and tell your project that about the app by editing the settings.py file

Step2:

Create a model in the database that the Django ORM will manage

Step3:

Tell your app what should be included in the view by editing views.py file

Step4:

Tell your project how the direct the incoming http request to your view in the app by editing urls.py file

Developing Web Application with Django

- In this assignment you must develop a Web application for a bookstore using Django framework.
  You must do the following:

- Set up Django
- Create a model for the bookstore in the database that the Django ORM will manage
- Setup the view for at least 1 template showing all book items e.g., book-title, book-author.
- Setup the url

- To deliver the assignment, include screenshot of the following items:
- - model.py
- - view.py
- - Screenshot showing the admin panel for book registration.
- - Screenshot showing the webpage in a browser

Obligatory Exercise 01 (Due date 10.02.2021)

- https://www.datacamp.com/community/tutorials/web-development-django
- https://docs.djangoproject.com/en/2.2/ref/templates/language/