Modern Software Engineering with DevOps:

Week 1 Workshop Presentation

Workshop Agenda

Activity	Estimated Duration
Set Up & Introductions	10 mins
Week 1 Review	60 mins
Assignment	40 mins
Break	15 mins
Assignment	100 mins
Check-Out (Feedback & Wrap-Up)	15 mins

Set up

Along with Zoom, please also be logged into Slack and the learning portal!

Week 1 Review

Overview

Software Development Methodologies	Docker Docker CLI
Waterfall	Docker Compose
Agile	Application Design
Scrum & Kanban	MVC vs MVT
SDLC with Agile	Django Architecture
DevOps	Django REST Framework
Git & GitHub	Django Project & Apps

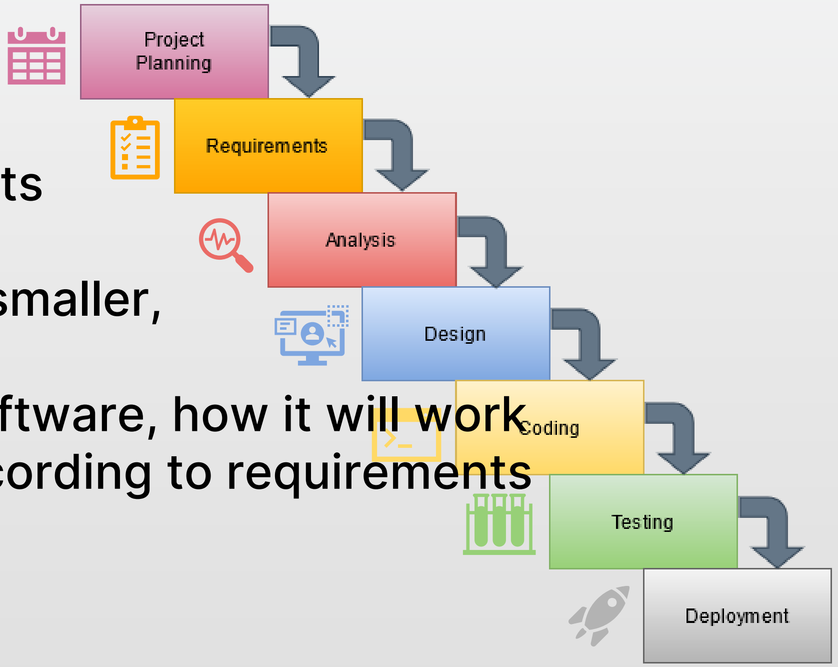
Review: Software Development Methodologies

- Organize and divide work into smaller steps or phases
- Common today are:
  - Waterfall
  - Agile
  - Scrum
  - Kanban
  - (and more)

Review: Waterfall

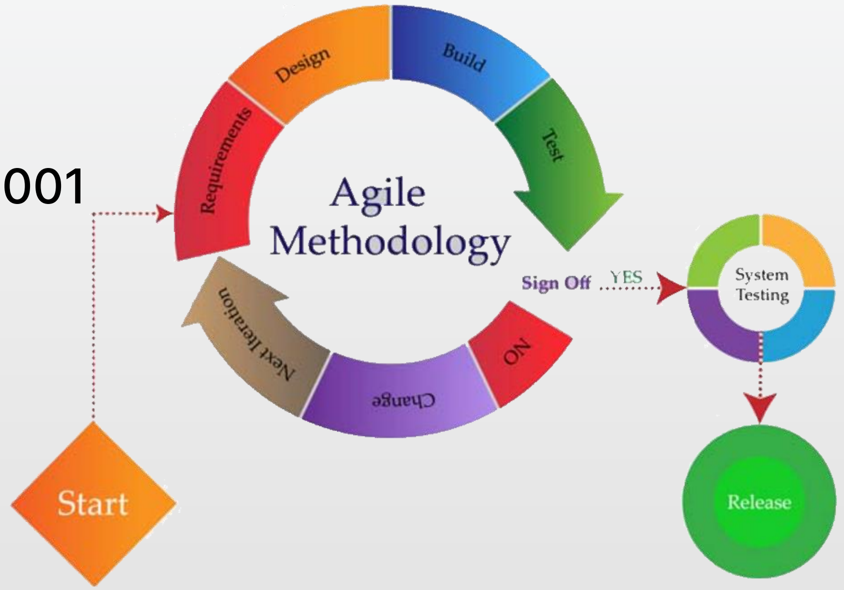
- **Project planning**: Software conceptualized & planned
- **Requirements:** Complete list of requirements gathered for project
- **Analysis:** Requirements broken down into smaller, easier to understand parts
- **Design**: Make decisions on how to build software, how it will work
- **Coding**: Design implemented into code according to requirements
- **Testing**: Code tested for bugs and defects
- **Deployment:** Project released to end users

- Best with clear, detailed requirements up front, minimum client involvement, well-defined timeline and budget



Review: Agile

- Newer than Waterfall, formally defined in 2001
- Key concepts:
  - Frequent iterations
  - Incremental builds
- Iterates through development phases
- Each iteration is called a "sprint"
- Lasts 1-4 weeks
- Collaborative and adaptive
- Best with clients who want high involvement and less rigid requirements up-front

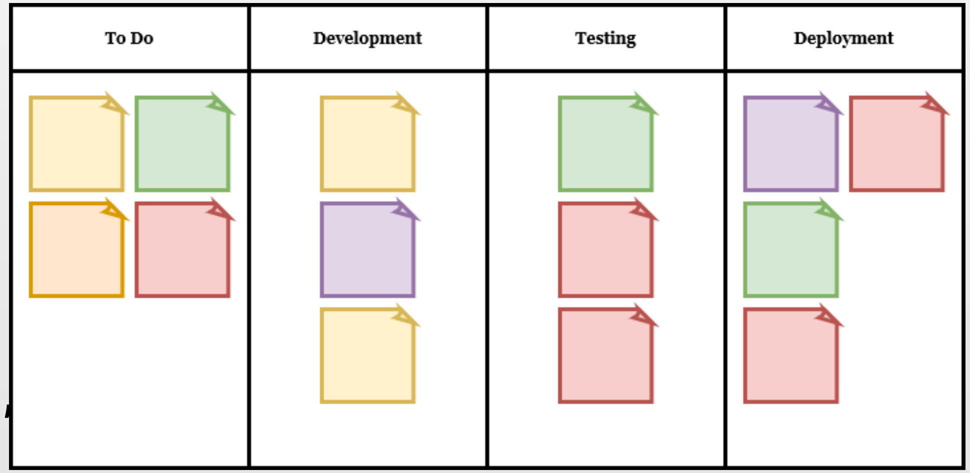


Review: Scrum & Kanban

- Two of many offshoot methodologies from Agile
- Scrum adds "**Scrum Rituals**" – regular meetings that include:
    - **Daily Scrums** – quick daily team-wide meetings about progress
    - **Sprint planning meetings** before start of each new sprint
    - **Demo/Review** – Clients can evaluate product at end of each iteration
    - **Retrospectives** – Team meetings after demo to evaluate previous sprint
- Scrum works well for quick MVP (minimum viable product) development, regular improvements

Review: Scrum & Kanban

- Kanban adds "Kanban boards"
  - Software like Trello or actual physical board with post-its defining tasks/"user stories"
  - "User story" describes feature from user's point of view
  - Tasks move between columns: To Do, Deployment
  - Like Scrum, also has daily meetings, demos for clients, retrospectives
  - No sprints! Continuous development
- Kanban works well for support & maintenance phases of a project, or when requirements are undefined and often changing



Review: SDLC with Agile

- Phase 1: Requirements
  - Analyze and specify business requirements
- Phase 2: Design
  - Software architecture and detailed design plan
- Phase 3: Development and Coding
  - Implement requirements/design into code
- Phase 4: Integration and Testing
  - Test code against requirements/design
- Phase 5: Deployment
  - Deploy to production environment, whether real or demo
- Phase 6: Post-Implementation Review
  - Development team & stakeholders meet to review progress
  - If all requirements completed, product is released
  - If work remains, new sprint begins

Review: DevOps

- Stands for **DEV**elopment + **OP**erations
- Combines Agile software **dev**elopment with IT **op**erations
- Cross-functional teams with overlapping responsibilities
- CI/CD (continuous integration / continuous delivery/deployment)
- 5 DevOps Principles:
  - Frequent releases
  - Team awareness and shared goals
  - Automation
  - End-to-end responsibilities
  - Continuous improvement

Review: Git

- Open source version control system
- Used for software version management, team collaboration
- Used by open source community as well as companies/corporations

- <u>Some Git CLI Commands</u>:
  - **git init** – initializes local repository
  - **git status** – view current repository status
  - **git add** *<filename>* - Adds specified file to staging
  - **git add .** – Adds all new and modified files to staging
  - **git commit** – Adds staged files to local repository
    - Use with **–m** flag for commit message, e.g. git commit "msg"
  - **git remote add origin** *<repository URL>* - link local repository to remote
  - **git push** – Push files to remote repository
  - **git clone** *<repository URL>* - download repository from online source
  - **git branch** – create new Git branch
  - **git pull** – pull changes from remote to local repository

Review: GitHub

- Repositories on networks let people share code and collaborate
- These can be private networks or online (on the internet)
- Online repository hosting services include GitHub, BitBucket, GitLab, SourceForge, others

- GitHub is the popular of these
- Web-based interface, or download desktop app
- Can have public or private repositories
- Also offers other services such as GitHub Pages for hosting websites, GitHub Actions for CI/CD workflows, issues tracking, etc

Review: Docker

Containers "package software into standardized units for development, shipment, and deployment." *- docker.com*

Docker is the most popular containerization software

Can be deployed anywhere, lightweight and portable

Three main parts:

- Docker Engine –used for building Docker images and creating Docker containers
- Docker Hub – a registry used to host Docker images
- Docker Compose – a utility for defining and managing multi-container applications

Review: Docker

Containers "package software into standardized units
for development, shipment, and deployment." *- docker.com*

Docker is the most popular containerization software

Can be deployed anywhere, lightweight and portable

Three main parts:

- Docker Engine –used for building Docker images and creating Docker containers
- Docker Hub – a registry used to host Docker images
- Docker Compose – a utility for defining and managing multi-container applications

Review: Docker

Basic steps for containerizing an app with Docker:

- Create Dockerfile
  - VS Code Docker extension can generate basic Dockerfile
- Build Docker image using Dockerfile
  - VS Code Docker can help with this also, or use **docker build** command
- Run Docker container instance from image
  - Use VS Code Docker, or use **docker run** command

Other helpful Docker commands:

docker ps	- list running containers
docker ps –a	- list running and stopped containers
docker images	- list Docker images in image repository
docker rm	- remove stopped Docker container

Review: Docker CLI

Q: What do these commands do?

docker ps
docker ps –a
docker images
docker stop
docker start
docker restart
docker rm
docker rmi
docker kill

Do you remember any other Docker CLI commands not listed here?
(Excluding the **docker compose** commands)

Review: Docker CLI

Q: What do these commands do?

docker ps	- list running containers
docker ps –a	- list running & stopped containers
docker images	- list available images
docker stop	- stop container, provide Container ID
docker start	- start stopped container, provide Container ID
docker restart	- stop then restart container, provide Container ID
docker rm	- remove stopped container, provide Container ID
docker rmi	- remove image provide Image ID
docker kill	- forcibly stop (kill) container, provide Container ID

Review: Docker Compose

- Define and manage multi-container applications
- Use configuration file named **docker-compose.yml** to define services
    - Services contain data for running containers, such as image location or where to find files to build an image, network ports, etc
- Use **docker compose up –d** to run containers from defined services
- Use **docker compose stop** to stop all containers
- Docker Compose commands must be executed from the same folder as the docker-compose.yml file

Review: Application Design

High-level/architectural design

- Specifies major system components and how they interact
- Addresses issues such as scalability and maintainability
- Software architecture patterns used (e.g. MVC)

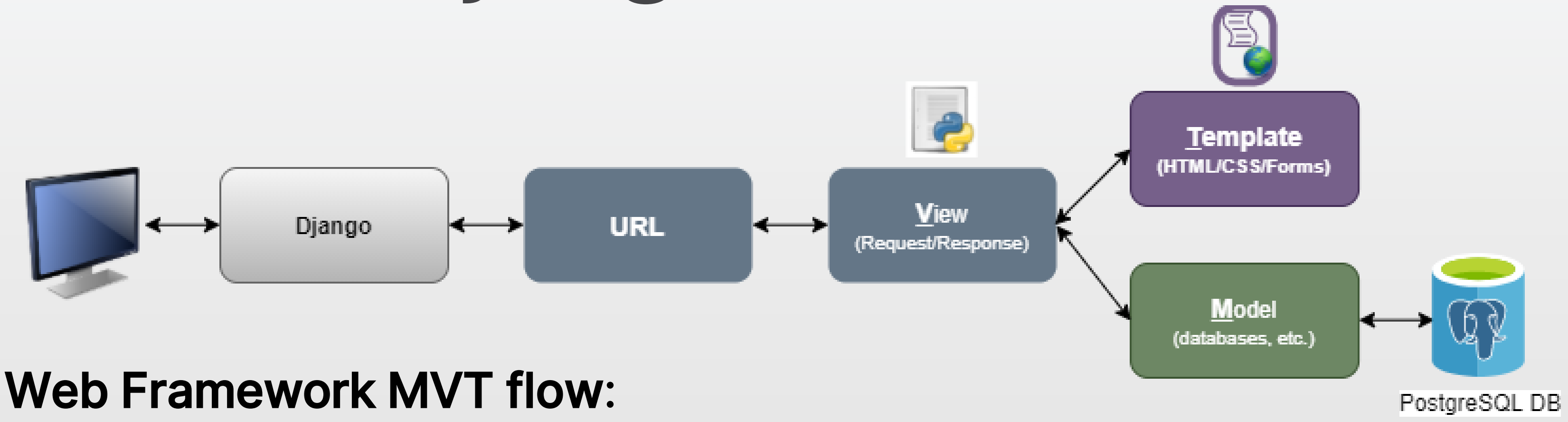
Low-level/detailed design

- Breaks down major components into smaller units
- Decides each unit's purpose and how they interact
- May include what algorithms and data structures are used
- Software design patterns used (e.g. Observer Pattern)

Review: MVC vs MVT

- Model-View-Controller (MVC) is a popular and common software architecture pattern with many variations
- **MVC**:
  - **Model** – set of classes that represent data and define business logic
  - **View** – User interface
  - **Controller** – Mediates between Model and View, controls data flow to Model
- **Model-View-Template (MVT)** is Django's variation of MVC:
  - **Model** – same as MVC
  - **View** - like Controller in MVC: mediates between Model and Template, controls data flow from HTTP requests to Model
  - **Template** – like View in MVC: User interface

Review: Django Architecture



Django Web Framework MVT flow:

User makes request to Django app

Django app uses defined URL patterns to determine which View to use

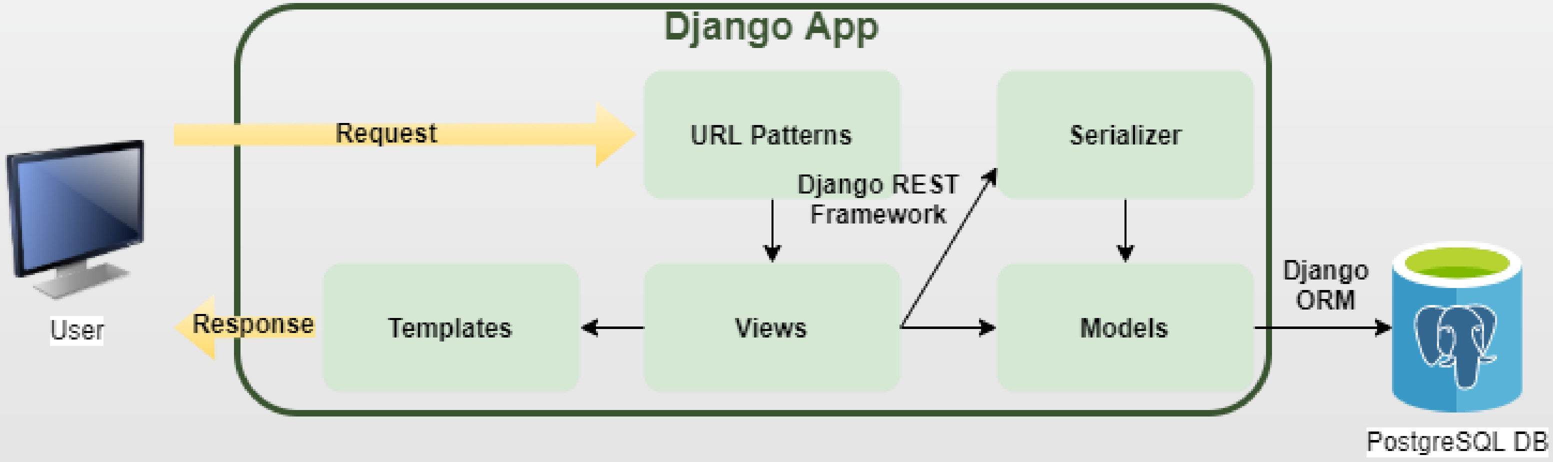
View determines which Models are needed, passes request to Models

Models interact with database through built-in Django ORM, return data to View

View sends data to Template

Template generates HTML pages that are passed back to user

Review: Django REST Framework



Helps produce endpoints that allow access/updates via RESTful API

Also serializes data returned in HTTP response

(i.e. transforms to JSON or XML)

Review: Django Project & Apps

- Use **django-admin startproject <projectname>** to scaffold out starter files for new Django project
- Django starter files inside project root folder include:
  - **manage.py** – main CLI tool for project, use *python manage.py runserver <portnum>* to start app; default port number is *8000*
  - subfolder for main application, has same name as the projectname given in *django-admin startproject <projectname>* command
  - **settings.py** – main settings file for project
  - **urls.py** – contains URL patterns used by app to direct HTTP requests
- One project can have many sub-applications
- Use **python manage.py startapp <appname>** to generate sub-app in project root folder
- Sub-app starter files (generated inside subfolder) include:
  - **views.py** – use to hold Python functions that handle HTTP requests
  - **models.py** – use to hold Python classes that model data for use by ORM
  - **urls.py** – URL patterns for sub-app

Workshop 1 Assignment

In this workshop, you will:

- Generate and set up a Django application as a Docker image
- Use Docker Compose to run it as a part of a multi-container application, along with a Postgres server and pgAdmin
- Implement password protection
- Use Django's built-in migrations and ORM
- Set up Django sub-applications
- Use Insomnia to POST new records to the database

You will be split up into groups to work on the assignment together.
Talk through each step out loud with each other, code collaboratively.
If your team spends more than 10 minutes trying to solve one problem,
ask your instructor for help!