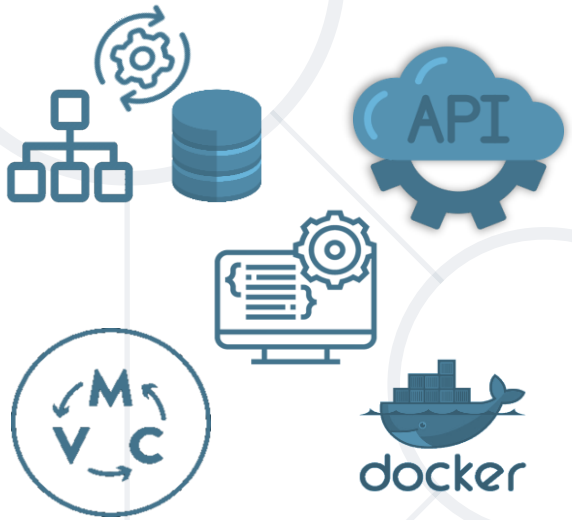
Back-End Technologies Overview

Databases, ORM Frameworks, MVC Frameworks, REST, Containers and Docker



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

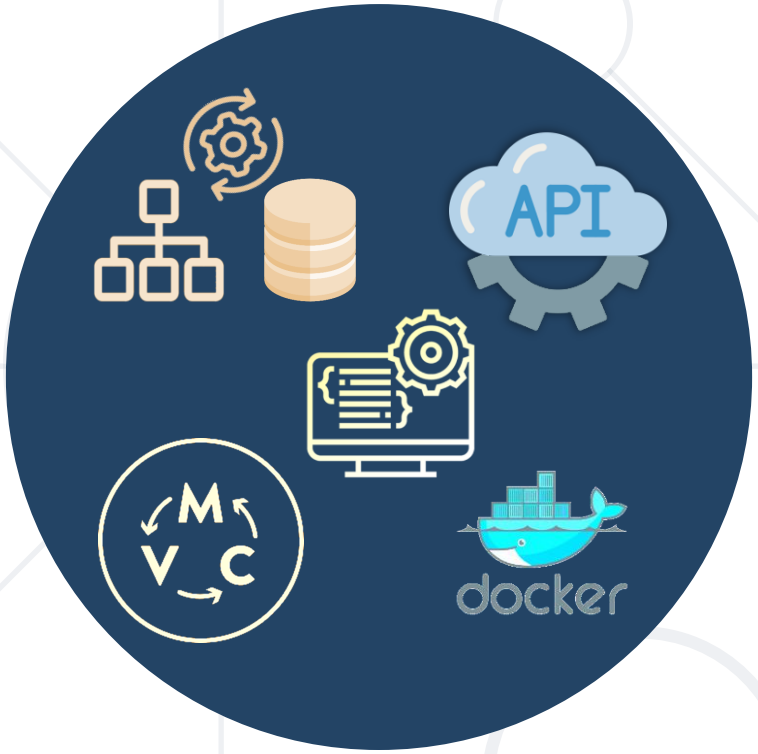
You Have Questions?

Software University

sli.do

#QA-Auto-BackEnd

Table of Contents

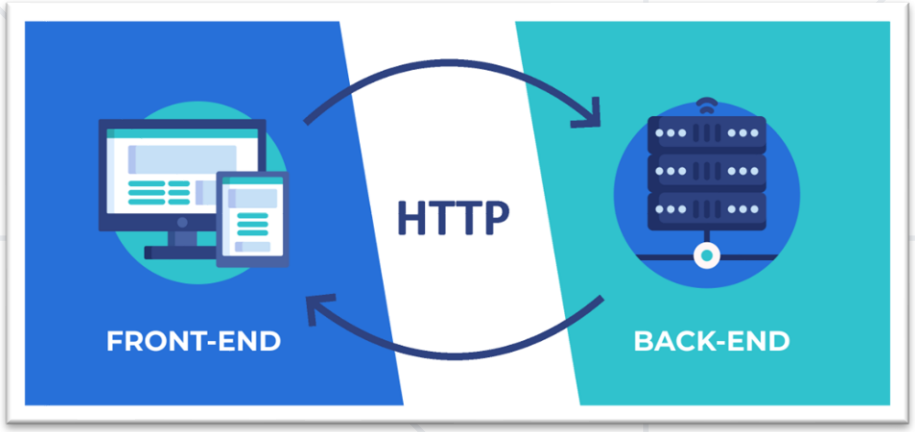


Back-End

Concepts and Technologies

Front-End and Back-End

- **Front-end** and **back-end** separate the modern apps into **client-side** (UI) and **server-side** (data) components
- **Front-end** == client-side components (presentation layer)
  - Implement the **user interface** (UI)
- **Back-end** == server-side components (data and business logic APIs)
  - Implements **data storage and processing**



- **APIs** connects front-end with back-end

Back-End Technologies

- **Back-end technologies** are about server-side programming
  - **Data management** technologies and **ORM frameworks**
  - Backend **Web frameworks** and **MVC** frameworks
  - **REST API** frameworks, **reactive** APIs, other services and APIs
  - **Microservices**, **containers** and **cloud**
- **Back-end testing** deals with the server-side, e.g. business logic, data processing, data storage, APIs

Back-End Languages and Platforms

Software University

- Back-end **technologies**: server-side frameworks and libraries
  - **C# / .NET back-end**: ASP.NET MVC, Web API, Entity Framework, …
  - **Java back-end**: Java EE, Spring MVC, Spring Data, Hibernate, …
  - **JavaScript back-end**: Node.js, Express.js / Meteor, MongoDB, …
  - **Python back-end**: Django / Flask, Django ORM / SQLAlchemy, …
  - **PHP back-end**: Apache, Laravel / Symfony, …

Databases and Data Storage

Data Management

Databases and Data Management

- **Database:**
  - Systematic **collection of data** that supports electronic **storage** and **manipulation** of information
- **Data Management:**
  - **Organizes** data to support business needs
  - Keeps data **accurate** and **usable**
  - **Sets rules** for secure data use
  - **Protects** data from loss
  - Ensures timely **access** for users
- **Importance:**
  - Sustains **integrity**, guarantees **security**, facilitates **accessibility**

Databases and Back-End

- **Databases** hold and manage data in the back-end systems
- **Relational databases** (**RDBMS**)
  - Store data in **structured tables** + **defined relationships**
  - Utilize SQL for **querying** and **data modification**
  - Examples: MySQL, PostgreSQL, Web SQL in HTML5

- **NoSQL databases**
  - Store **collections of documents**, **key-value pairs** or **other data models**
  - Examples: MongoDB, IndexedDB in HTML5

Relational Database Management System (RDBMS)

- **Structure:**
  - Data is **organized** into **tables**, **columns**, and **rows**
  - Tables have **relationships** with each other (Primary & Foreign Keys)
- **Query Language: SQL**
- **Example**:
  - E-commerce site with tables for products, customers, and order histories, ensuring data integrity across purchases

NoSQL

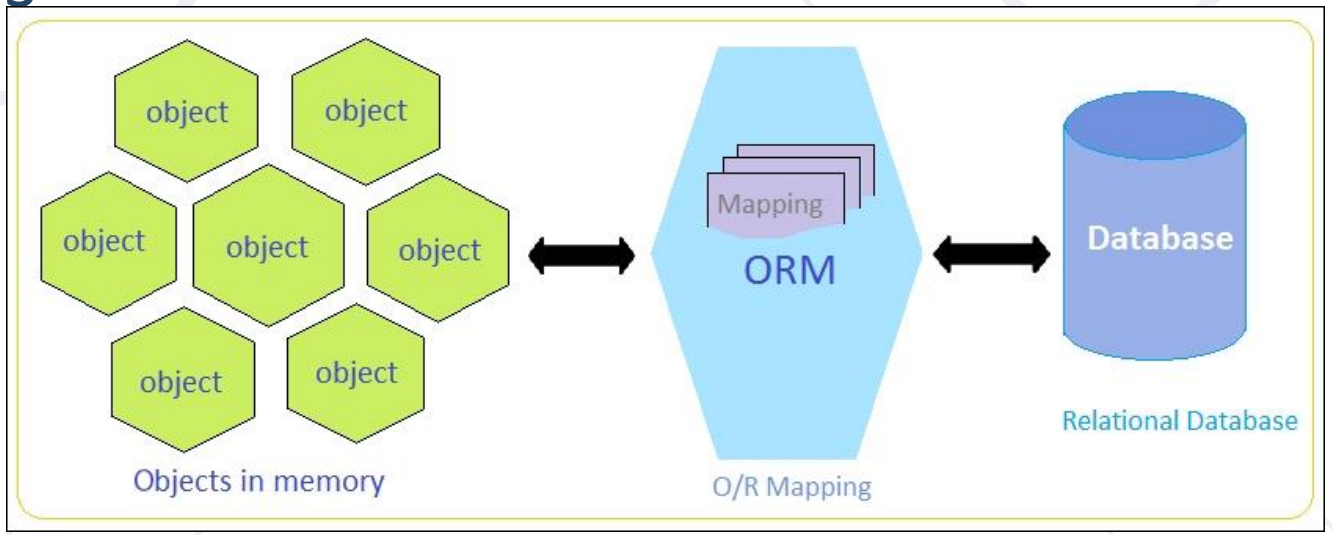
- **Structure:**
  - **No fixed** schema, can store JSON-like documents, **key-value** pairs, **graphs**, or **columnar** data
- **Query Language:**
  - **Varies** based on the NoSQL type (e.g., BSON for MongoDB)
- **Example:**
  - Social media platform storing varied user posts (text, images, videos) in MongoDB without a fixed schema

ORM

Object-Relational Mapping Frameworks

What is ORM?

- **Definition:**
  - **Object-Relational Mapping** (ORM) is a programming technique to interact with relational databases using object-oriented paradigms
- **Simple explanation:**
  - ORM is like a **translator** between the language of object-oriented programming and relational databases



ORM Purpose

- **Transforms data** between incompatible systems (objects in code to relational tables)
- Serve as an **abstraction layer** between the application and the database
- **Replaces complex SQL queries** with object methods for readability and maintainability
- **Automates database operations** to simplify development
- **Easier adaptation** to changes in database schema

Why Use ORM?

- **Productivity** - Makes building software faster by handling common database tasks automatically
- **Maintainability** - Makes it easier to update and improve the code as time goes on
- **Abstraction** - Lets programmers use more general concepts instead of detailed and complex code
- **Performance** - Speeds up getting data from the database but might make the system slower in some cases
- **Portability** - Allows the use of the same code with different types of databases

Popular ORM Frameworks

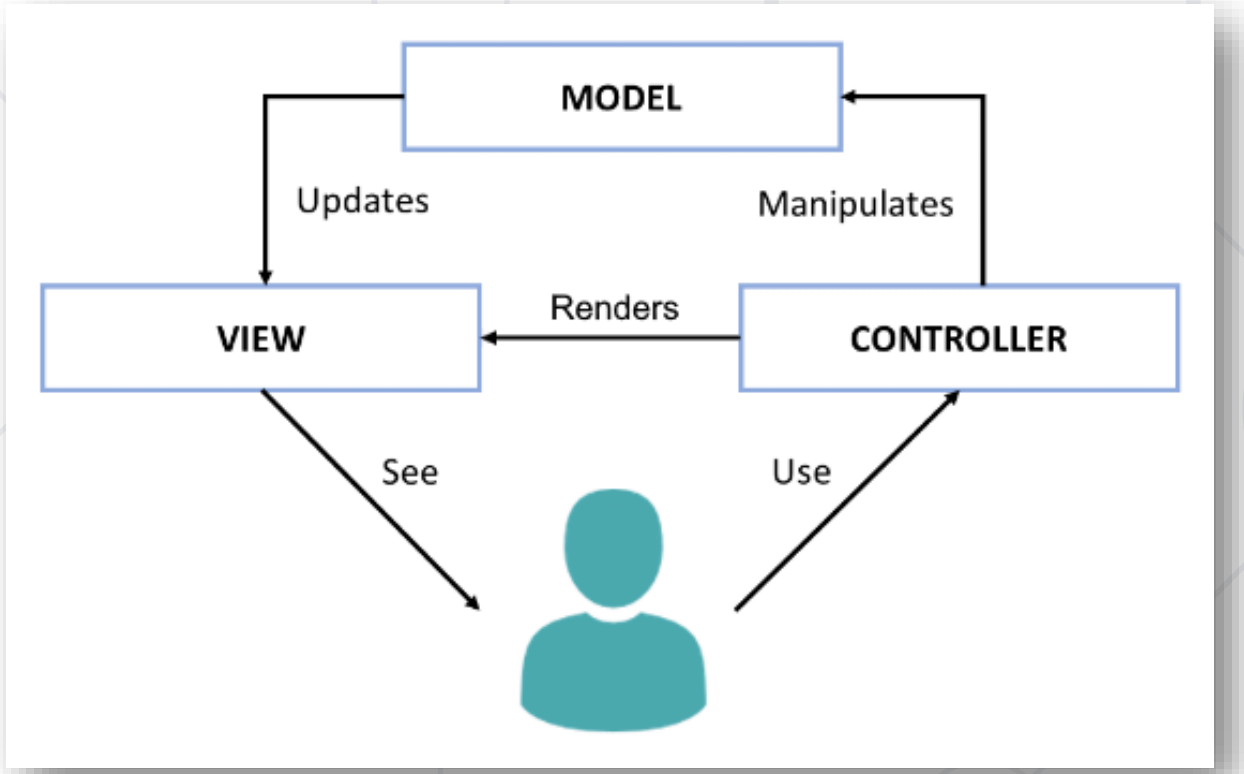
- **Entity Framework (C#)** - Microsoft's ORM for .NET applications, offering LINQ for querying
- **Hibernate (Java)** - Robust Java ORM that maps Java classes to database tables
- **Sequelize (JavaScript)** - A promise-based ORM for Node.js, supporting PostgreSQL, MySQL, SQLite, and more
- **SQLAlchemy (Python)** - Offers a full suite of ORM tools for Python, known for its flexibility

MVC

The Model-View-Controller Pattern

The Model-View-Controller (MVC) Pattern

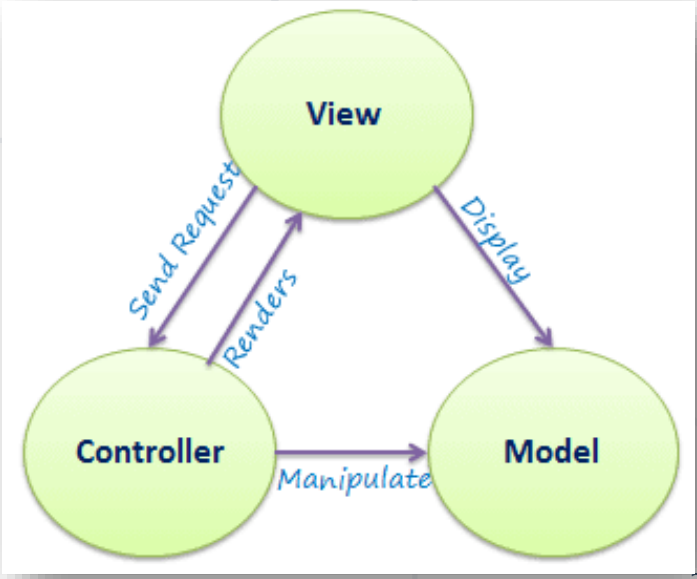
- The **Model-View-Controller** (**MVC**) pattern



- **Controller**
  - Handles user actions
  - Updates the model
  - Renders the view (UI)
- **Model**
  - Holds app data
- **View**
  - Displays the UI, based on the model data

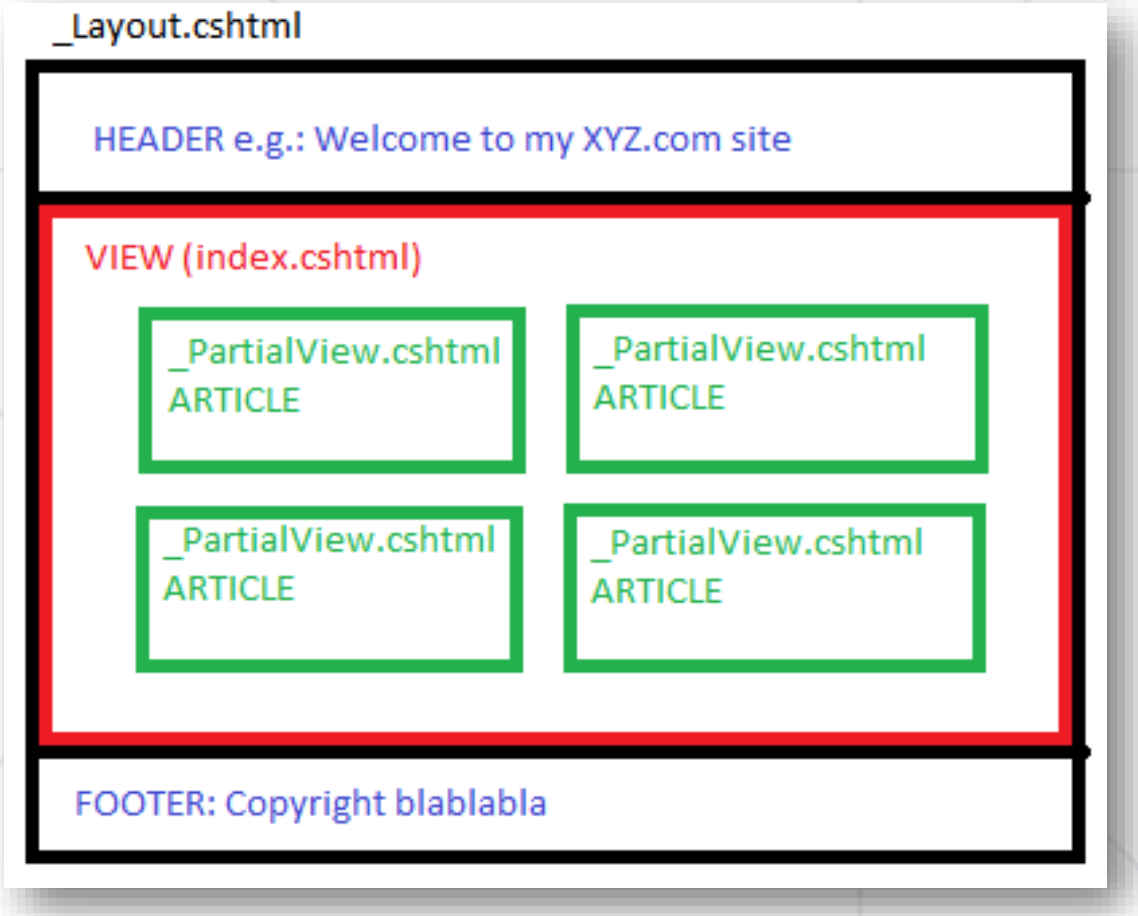
The MVC Pattern

- **Model–view–controller** (MVC) is a **software architectural pattern**
- Originally formulated in the late 1970s by Trygve Reenskaug as part of the **Smalltalk** (object-oriented programming language)
- **Code reusability** and **separation of concerns**
- Originally developed for **desktop**, then adapted for **internet applications**



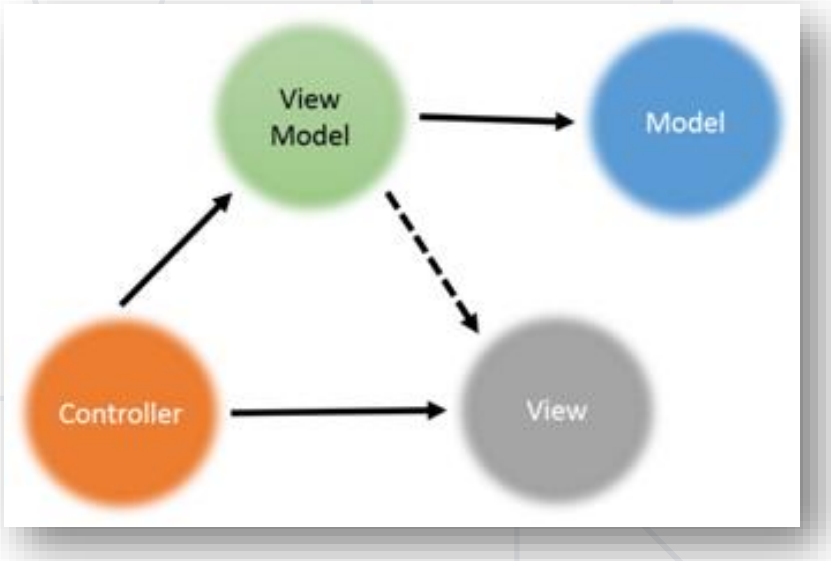
View

- The **View** in **MVC** represents:
  - Defines how the application's user interface (**UI**) will be displayed
  - May support **Master Views** (layouts) and **Sub-Views** (**partial views** or controls)
  - In Web apps: template to dynamically generate HTML



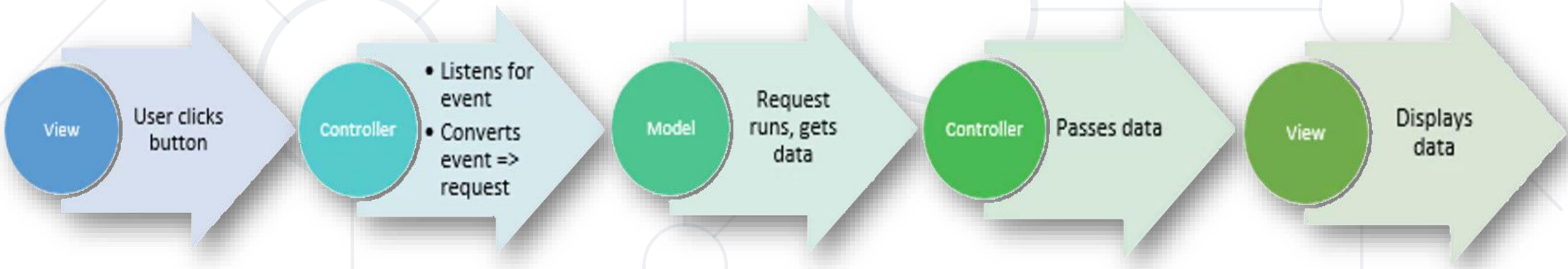
Model

- The **Model** in **MVC** represents:
  - A **set of classes** that describes the **data** we display in the UI
  - May contain **data validation rules**
- Two types of models
  - **View model / binding model**
    - Maps the UI of the Web page to C# class
    - Part of the **MVC** architecture
  - **Database model / domain model**
    - Maps database table to C# class (using ORM)



MVC Steps

- Incoming **Request** routed to **Controller**
- **Controller** processes **Request** and creates a **Model** (view model)
  - Controller also selects **appropriate result** (for example: **View**)
- **Model** is passed to the **View**
- **The View** transforms **Model** into appropriate output format (HTML)
- **Response** is rendered (**HTTP Response**)



Web Services and APIs

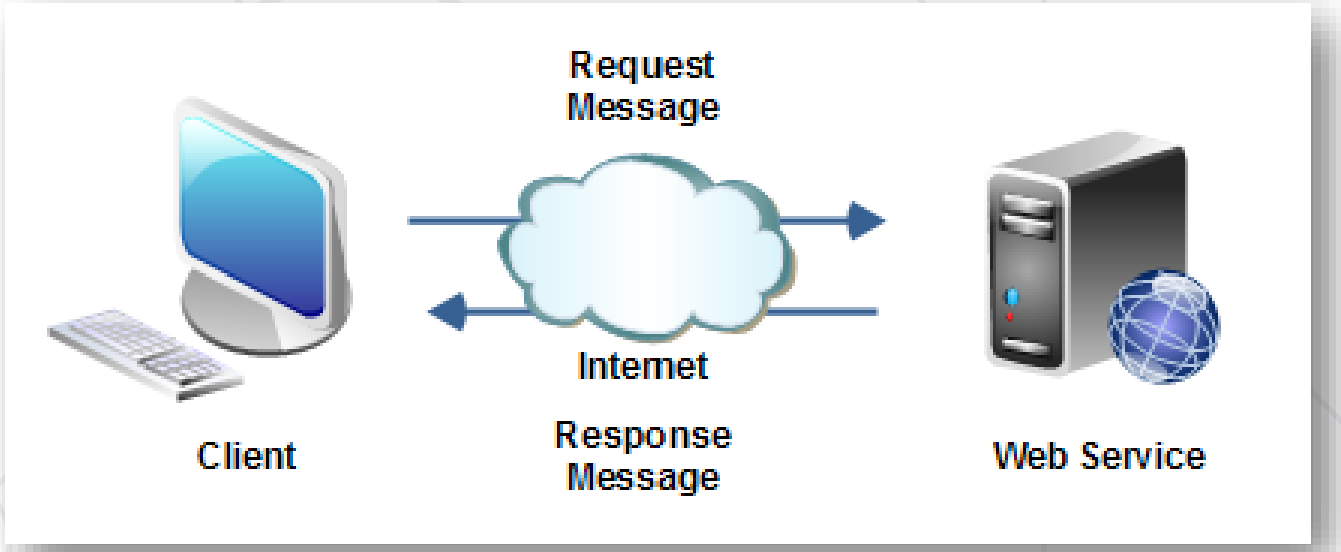
Communication between Systems and Components

What is API?

- **API** == **A**pplication **P**rogramming **I**nterface
  - Designed for communication between system components
  - Set of **functions** and **specifications** that software programs and components follow to talk to each other
  - Examples
    - JDBC – Java API for apps to talk with database servers
    - Windows API – Windows apps talk with Windows OS
    - Web Audio API – play audio in the Web browser with JS

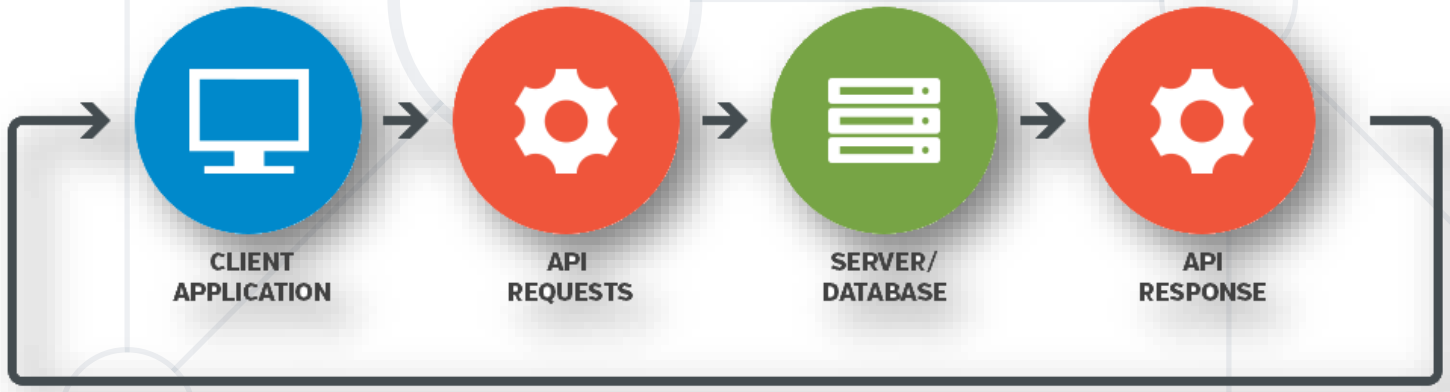
What is Web Service?

- **Web services** implement **communication** between software **systems** or **components** over the **network**
  - Using standard **protocols**, such as HTTP, JSON and XML
  - Exchanging **messages**, holding data and operations
- All **web services are APIs**, but not all APIs are web services



Web Services and APIs

- **Web services** expose **back-end APIs** over the **network**
  - May use different **protocols** and **data formats**: HTTP, REST, GraphQL, gRPC, SOAP, JSON-RPC, JSON, BSON, XML, YML, …
- **Web services** are hosted on a Web server (HTTP server)
  - Provide a set of functions, invokable from the Web (Web API)
- **RESTful APIs** is the most popular Web service standard



REST and RESTful Services

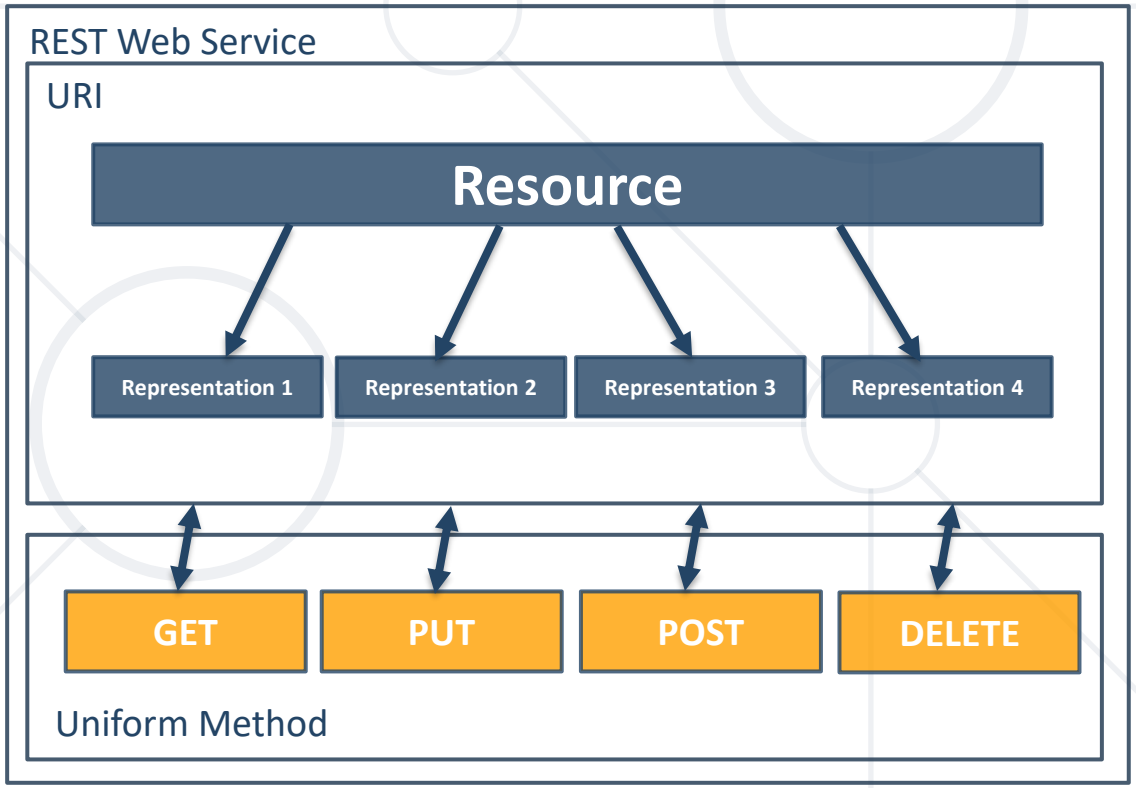
Web APIs based on HTTP, REST and JSON

REST

- **REST** (**Re**presentational **S**tate **T**ransfer) is an architectural style for building web services, that are lightweight, fast, and scalable
- RESTful web services **use** the **HTTP protocol** for communication
- Two key **principles**
  - **Stateless** - the server does not maintain any information about the client between requests
  - **Use of resource-based URLs** - each resource is identified by a unique URL, and the server responds to requests for that resource by returning the appropriate data.

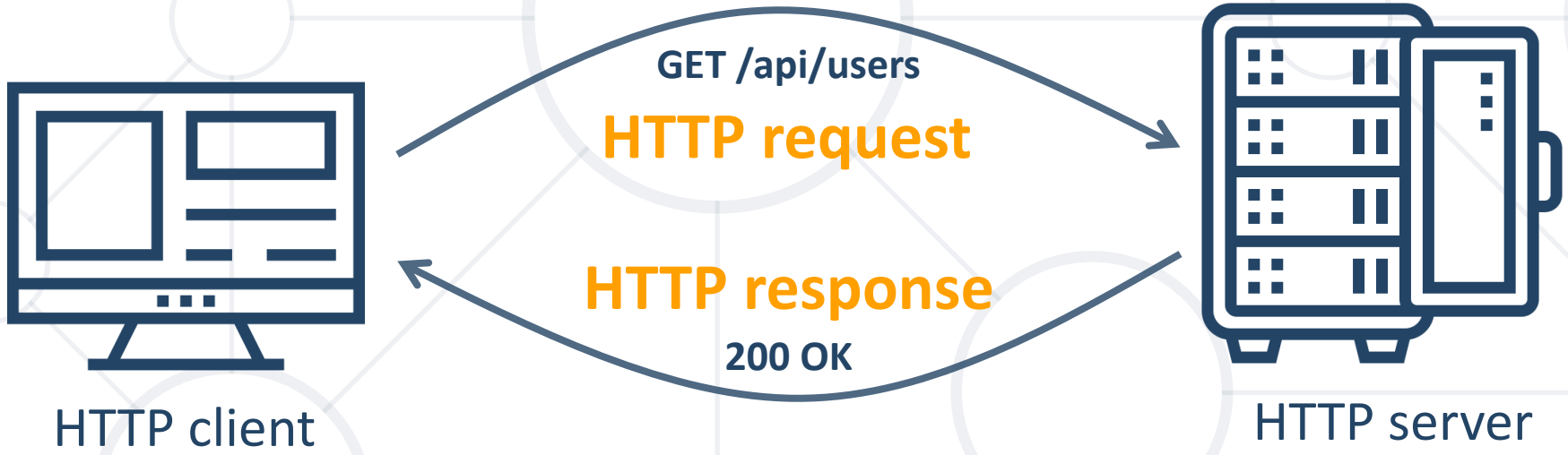
REST and RESTful Services

- Representational State Transfer (REST)
  - Architecture for client-server communication over HTTP
  - Resources have URI (address)
  - Can be created / retrieved / modified / deleted / etc.
- RESTful API / RESTful Service
  - Provides access to server-side resources via HTTP and REST



RESTful APIs

- **HTTP** is text-based client-server protocol for the Internet



- **RESTful APIs** are HTTP-based Web services (backend apps)

REST and RESTful Services – Example

Software University

- **Get** all posts / specific post

GET	http://some-service.org/api/posts
GET	http://some-service.org/api/posts/17

- **Create** a new post

POST	http://some-service.org/api/posts

- **Delete** existing post

DELETE	http://some-service.org/api/posts/17

- **Replace** / **modify** existing post

PUT/PATCH	http://some-service.org/api/posts/17

Virtualization, Containers, Docker

Virtualization, Containers, Running Docker Images

Virtualization

- **Virtualization** - The process of creating a virtual version of something that traditionally exists physically, e.g. computing environments
- It encompasses virtual computer hardware platforms, storage devices, network resources, and more
- **Benefits:**
  - Resource Utilization
  - Isolation
  - Environment Replication

Virtualization and Cloud

- **Virtualization** == running a **virtual machine** (VM) / virtual environment inside a physical hardware system
  - e.g., run Android VM or Linux inside a Windows host
  - Storage, memory, networking, desktops can also be virtual
- **Cloud** == computing resources, virtual machines, storage, platforms and software instances, available on demand
  - **IaaS** (infrastructure as a service) – virtual machines on demand
  - **PaaS** (platform as a service) – app deployment environments
  - **SaaS** (software as a service) – software instances, e.g. Office 365

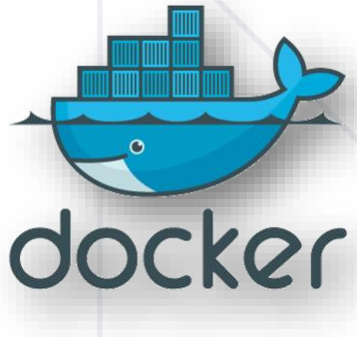
VMs vs Containers

- VMs virtualize the hardware
- Complete isolation
- Complete OS installation. Requires more resources
- Runs almost any OS

- Containers virtualize the OS
- Lightweight isolation
- Shared kernel. Requires fewer resources
- Runs on the same OS

Containers and Docker

- **Container image** == software, packaged with its dependencies, designed to run in a virtual environment (like Docker)
  - e.g., WordPress instance (Linux + PHP + Apache + WordPress)
  - Simplified installation, configuration and deployment
- **Docker** is the most popular containerization platform
  - Runs **containers** from local **image** or downloaded from the **Docker Hub** online repository
  - Open-source, runs on Linux, Windows, Mac



Docker – Example

- Install **Docker** on your local computer
  - Or use the Docker online playground: https://labs.play-with-docker.com (with a free Docker Hub registration)
- Download and **run a Docker image** in a new container:

```
docker run -d -p:8080:80 dockersamples/static-site
```

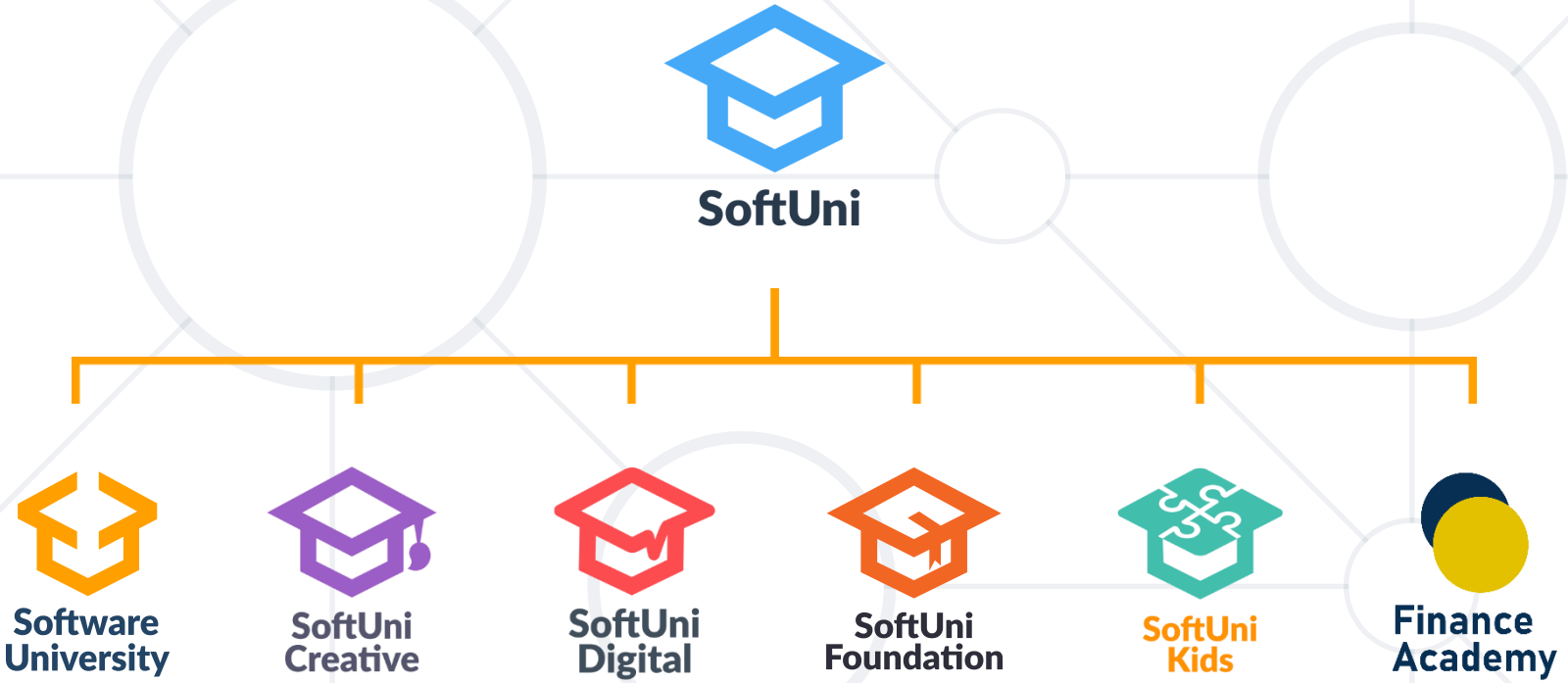
- Open the exposed URL: http://localhost:8080
- View currently running Docker containers

```
docker ps
```

Summary

- What is **Back-End**?
- Databases: **SQL** vs. **NoSQL**
- **ORM** - **Translator** between OOP language and relational databases
- **MVC** - Send request, Manipulate, Render, Display
- **APIs** - Communication between system components
- **Rest** and RESTful Services - HTTP, REST and JSON
- Virtualization, Containers, Docker
  - VMs vs Containers
  - **Docker** - the most popular containerization platform

Questions?



© SoftUni – https://about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

Flutter™ International

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

BOSCH

Postbank
Решения за твоето утре

PHAR VISION

SmartIT

DXC TECHNOLOGY

createX

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg