Designing simple web Application

Lukasz Pasierbiewicz, Łukasz Kutrzuba 2023

Before we start ...

Łukasz Pasierbiewicz

- Has been working in Ericsson for 5 years,
-Software Developer responsible for 4G/5G software testing and django web application in one of Ericsson's QAs
- Main technology used at work: Python.

Łukasz Kutrzuba

- Has been working at Ericsson for 10 years,
- Senior Software Developer responsible of automation in one of Ericsson's QAs,
- Main technology used at work: Java.

YOUR EXPERIENCE

- Have you ever created a web application?

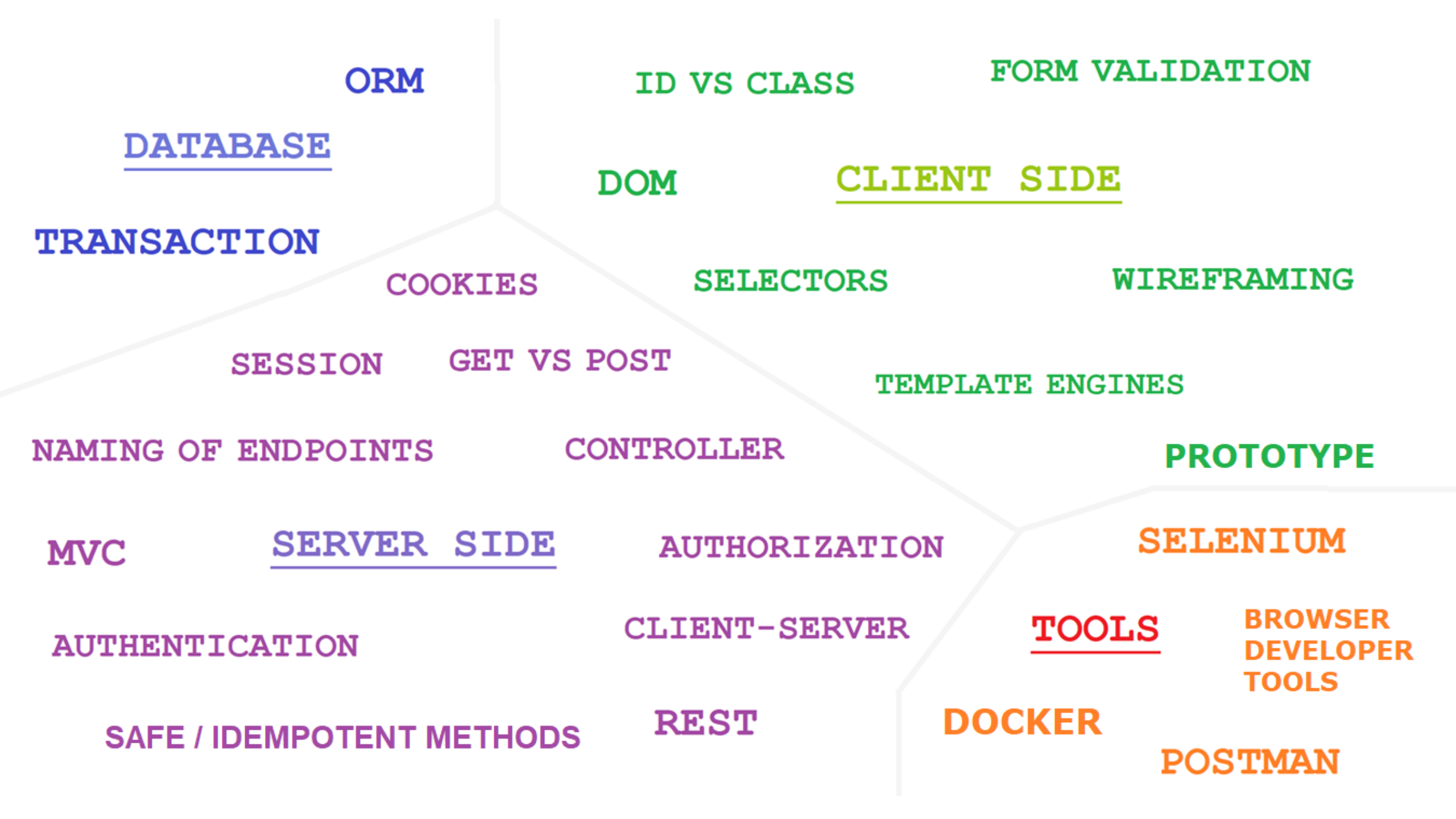
- Are you familiar with REST concept?

- Which technologies did you use?
  - **Servlets / JSP**
  - **ASP**
  - **Node JS**
  - **Django**
  - **Spring**
  - **JSF**

Content of presentation

1. **Theoretical Background**
   - REST, GET vs POST, Idempotent, Safe, Endpoints
2. **Model View Controller**
3. **Designing Application**
   - Roles, Scenarios, Artifacts/Items, Webpages&Forms, Flow, Wireframing,
4. **Useful Tools**
   - Developer Tools of Browser, Docker,
5. **Implementation Backend**
   - Django, NGinx, Gunicorn,
6. **Frontend**
   - Bootstrap, JQuery,

1 Theoretical Background


ORM
DATABASE
TRANSACTION
ID VS CLASS
FORM VALIDATION
DOM
CLIENT SIDE
SELECTORS
WIREFRAMING
COOKIES
SESSION
GET VS POST
TEMPLATE ENGINES
NAMING OF ENDPOINTS
CONTROLLER
PROTOTYPE
MVC
SERVER SIDE
AUTHORIZATION
SELENIUM
AUTHENTICATION
CLIENT-SERVER
TOOLS
BROWSER DEVELOPER TOOLS
SAFE / IDEMPOTENT METHODS
REST
DOCKER
POSTMAN

REST methods - overview

REST (**Re**presentational **S**tate **T**ransfer) - an architectural style defining constraints to be used for web services.

REST != HTTP

REST principles:

- Uniform interface,
- Client – server separation,
- Stateless,
- Cacheability,
- Layered system,
- Code on demand.

REST methods - overview

Example **REST** methods:

- **GET** – fetch data from database – e.g., show details for certain book or search book by title,
- **POST** – add new data to database – e.g., new book in library,
- **PUT** – update information in database – e.g., edit data for certain library user,
- **DELETE** – remove data from database – e.g., delete library user.

GET VS POST

- **GET** method is used for retrieving certain resources.

It is safe / read-only operation,

- **POST** method is used for creating new resources,

POST method should be used when sending confidential data (login/password) as it is more secure,

GET VS POST

TOPIC	GET	POST
Can be cached?	**YES**	**NO**
Can be bookmarked?	**YES**	**NO**
Limited size of data	**YES**	**NO**
Security	Less secure,	More Secure
Is method **safe ?**	**YES**	**NO**
Is method **idempotent ?**	**YES**	**NO**
Visibility	Data visible in URL	Data hidden,

WHAT IS IDEMPOTENT OPERATION?

- Operation is idempotent if it will give the same result if performed once or multiple times.

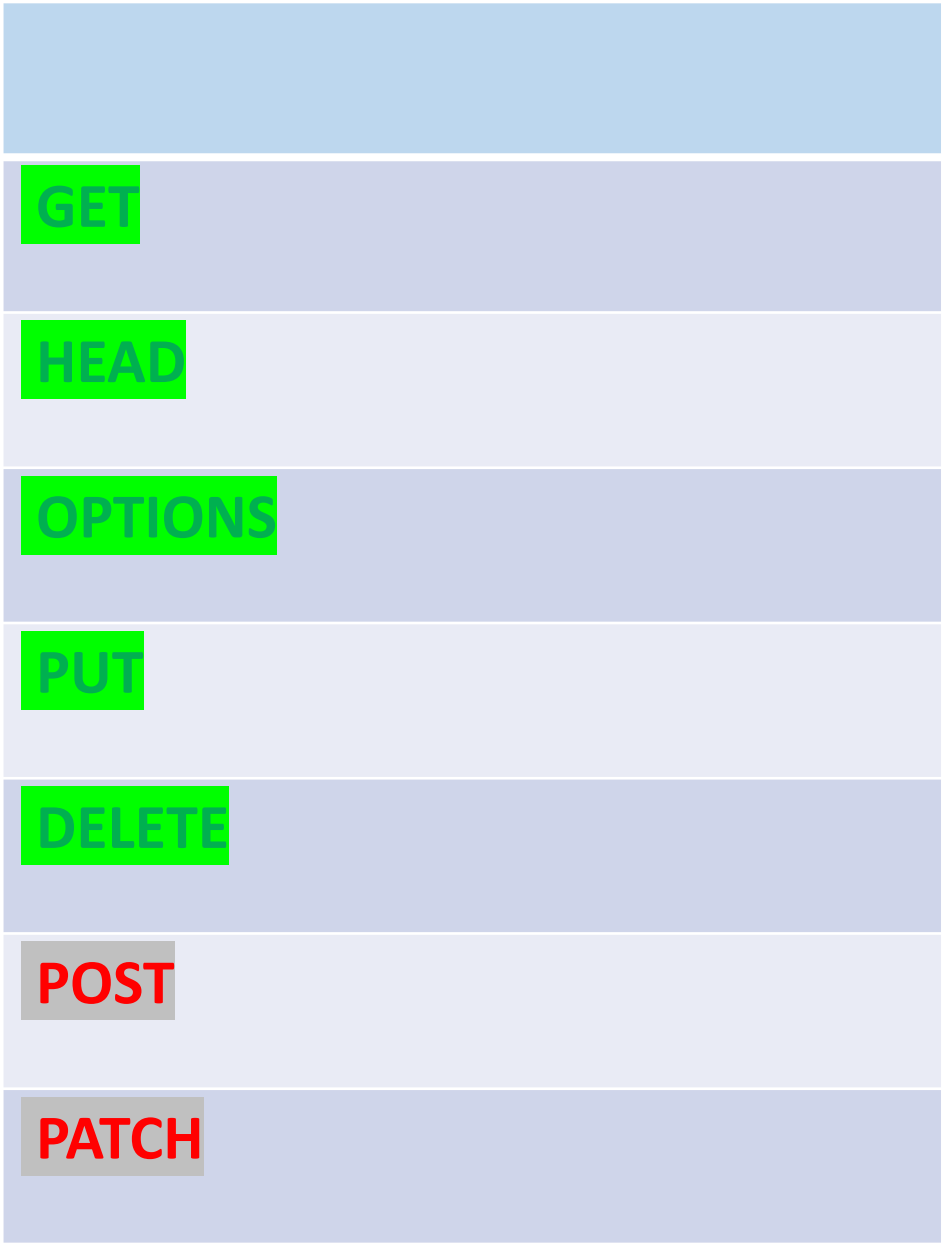
- For example – checking bank account balance is idempotent, we will get the same result.

- Withdrawing money is not idempotent – our balance changesevery time we take money,

IDEMPOTENCE - EXAMPLES

- Multiply number by 1 **(x*1)**
- Assigning variable **(variable = 100)**
- Add zero to certain number **(x + 0)**
- Absolute value **(|x|)**
- Eat one piece of a cake <not idempotent>
- Delete specific row from database,
- Check bank account balance,
- Withdraw 2000$ from bank account - <not idempotent>

REST: IDEMPOTENT METHODS



REST: SAFE METHODS

HTTP methods are considered safe if they do not change the server state.

In other words: safe methods do not modify resources (they are read-only).

For example using **GET** or **HEAD** on a resource url we should never change the resource data.

REST: SAFE METHODS

GET
HEAD
OPTIONS
PUT
DELETE
POST
PATCH

REST – NAMING OF ENDPOINTS

https://book-rental/books/authors

http://book-rental/book-management/

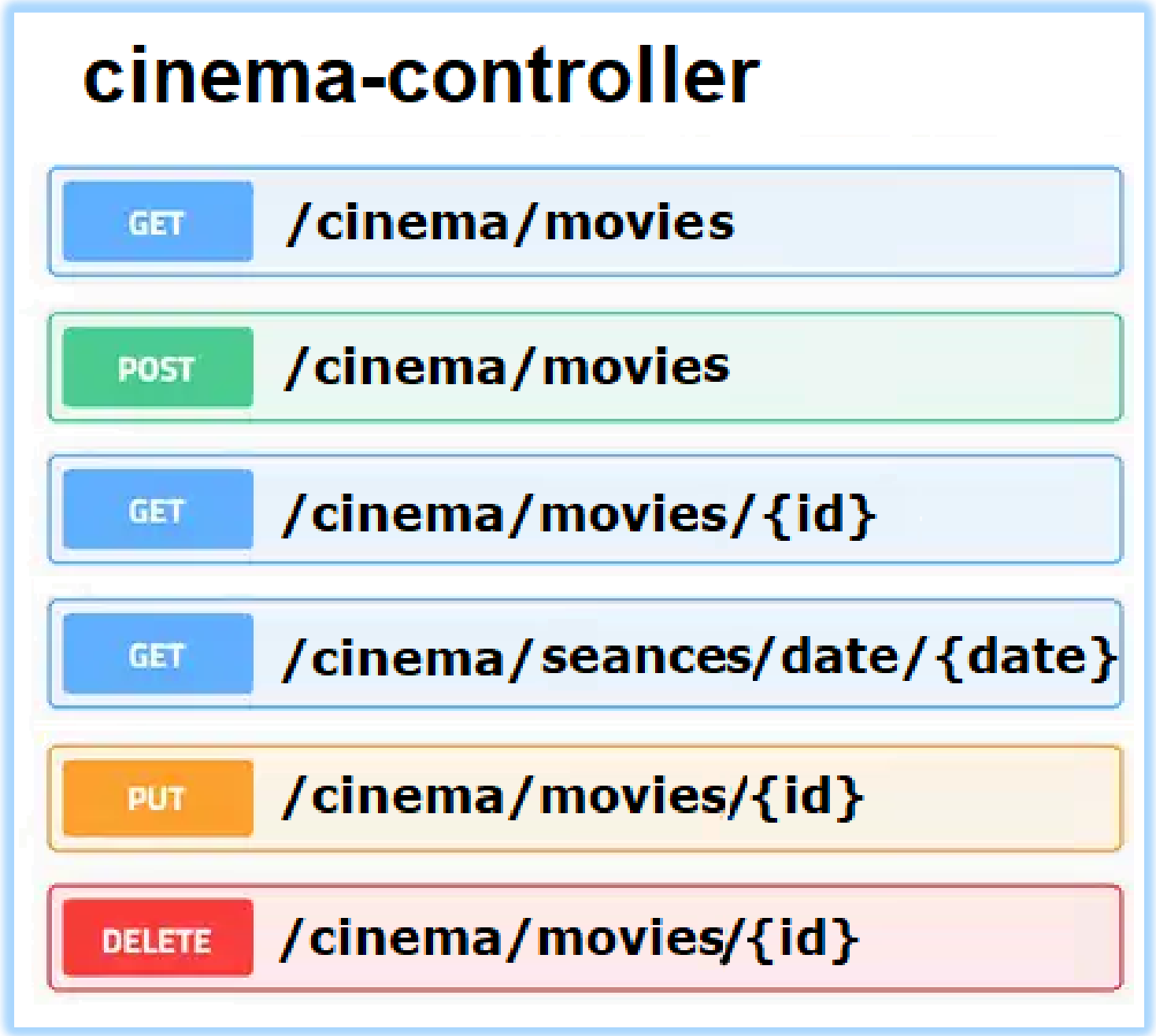
- Use nouns to name URIs,
- Separate words with hyphens,
- Use lowercase letters,
- Use clear, intuitive names,
- Use plural nouns (to avoid ambiguity),
- Avoid special characters,

REST ENDPOINTS

employees-controller

GET	/employees	- list all employees
GET	/employees/{id}	- display employee with given id
POST	/employees	- create new employee
DELETE	/employees/{id}	– remove employee with given id
PUT	/employees/{id}	- update employee with given id

REST ENDPOINTS



REST ENDPOINTS - DESIGN

POST	/quotes	- creating new quote
GET	/quotes	- list all quotes
GET	/quotes/{id}	- display quote with given id
PUT	/quotes/{id}	- update quote with given id
DELETE	/quotes/{id}	- delete quote with given id

ENDPOINTS

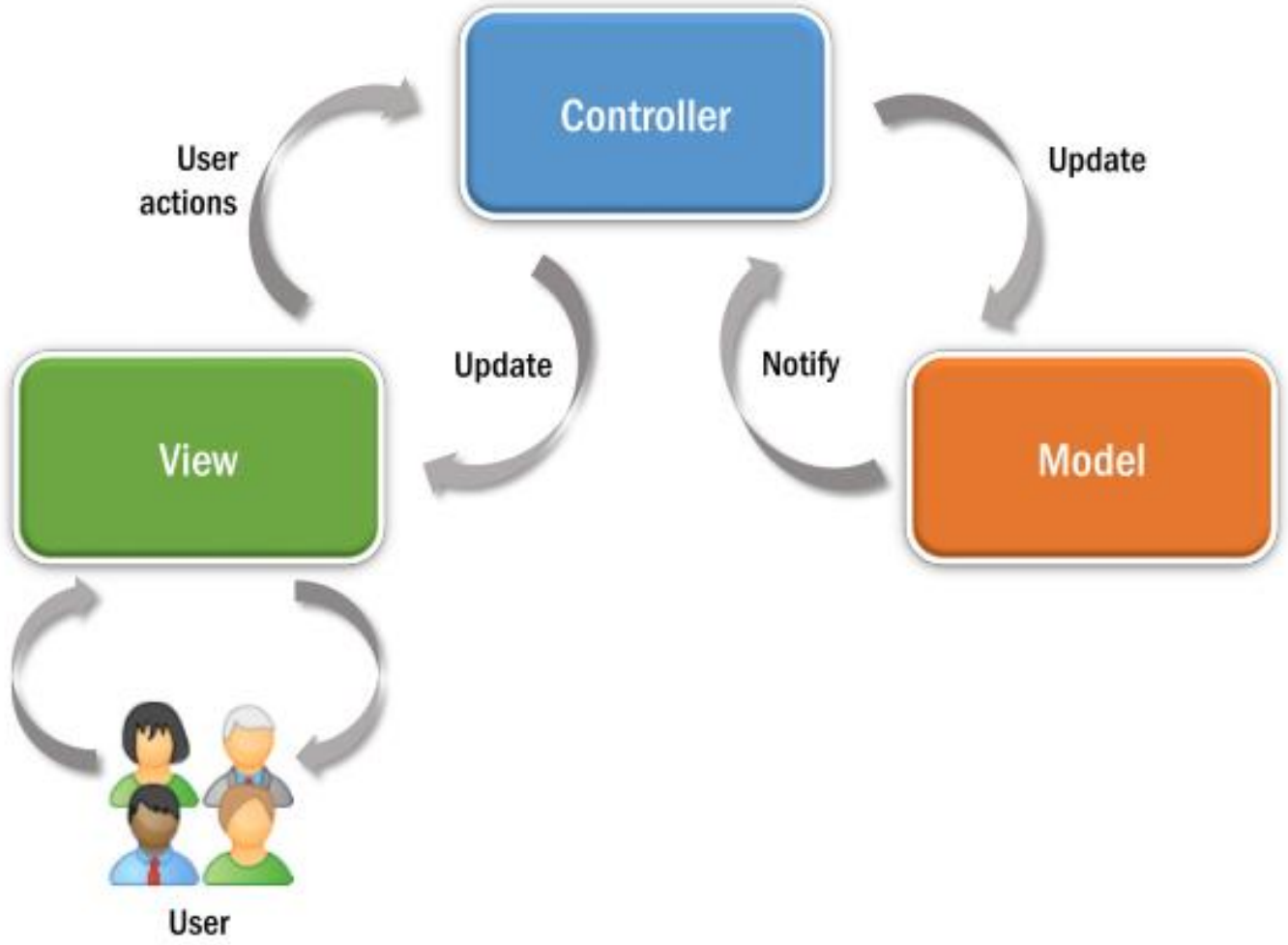
- Of course, our URLs can have multiple parameters:
- /users/**{userId}**/cars/**{carId}**
- /users/**12454**/cars - list with cars that belong to user with id '12454’
- /users/**12454**/cars/**1233** - display the information of car '1233' which owner is user with id '12454'
- /users/**12454**/groups - list with groups where certain user belongs to,

2 Model –View –Controller

Model-View-Controller

MVC, or Model-View-Controller, is a software architectural pattern commonly used in the development of web applications. It divides an application into three interconnected components.

This separation makes the code more modular, maintainable, and scalable because changes in one component do not impact the others. MVC is widely used in many software development frameworks (especially for web development frameworks)

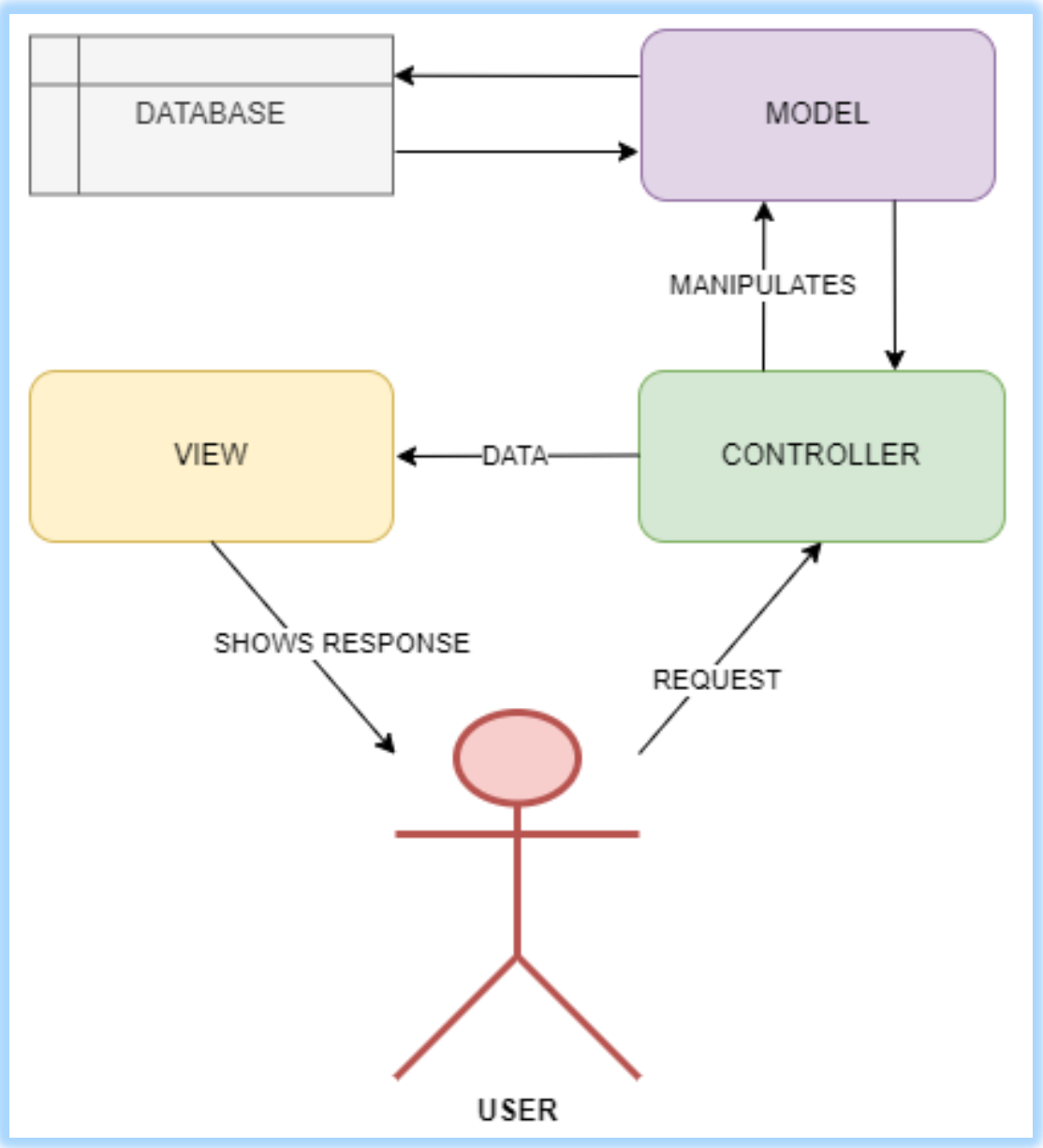


Source: https://upload.wikimedia.org/wikipedia/commons/thumb/7/7a/Model-View-Controller_architectural_pattern.svg/800px-Model-View-Controller_architectural_pattern.svg.png?20190801172842

MVC - Controller

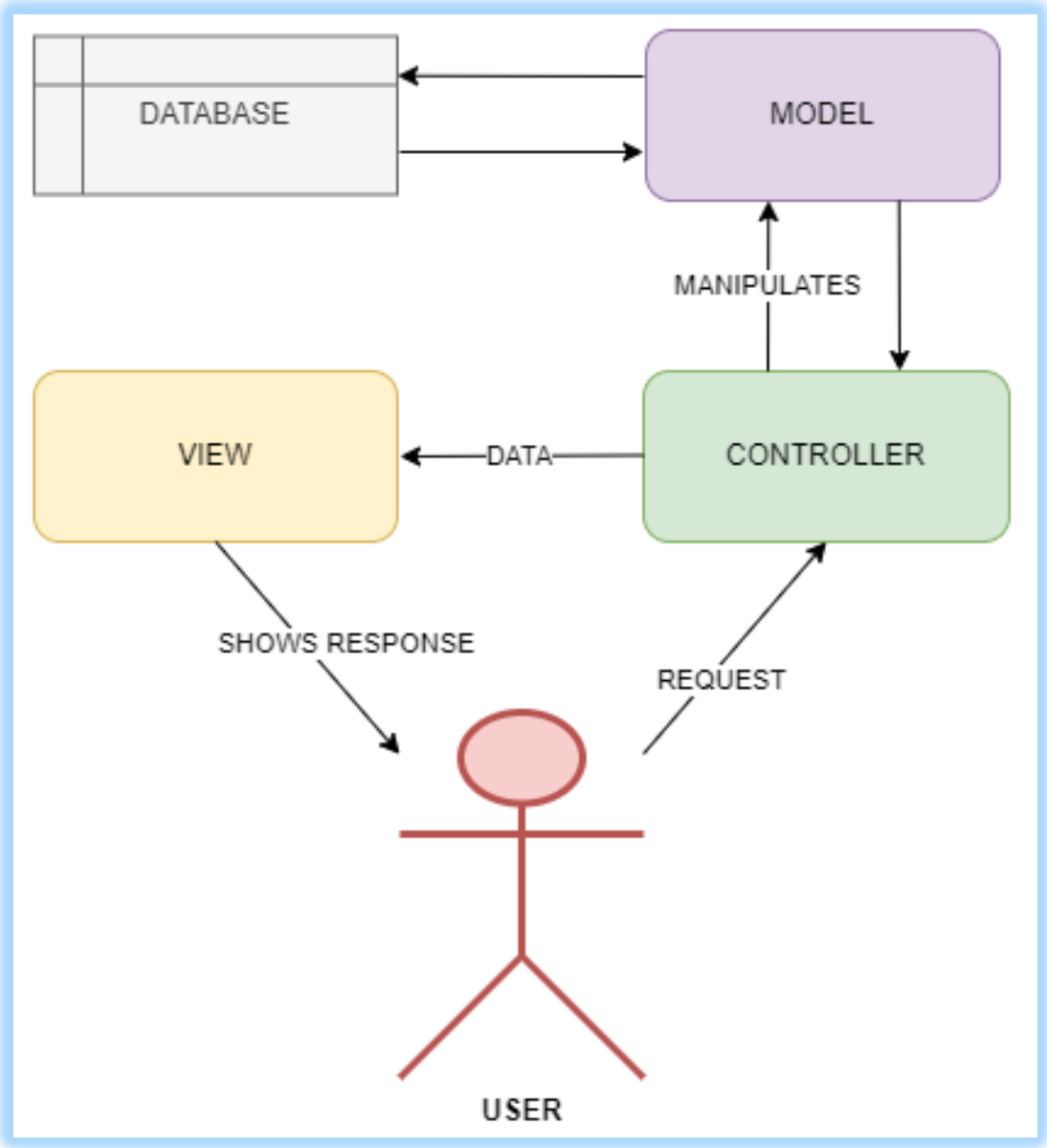
Controller - in web applications, the controller plays the role of an intermediary between the model and the view. It receives user input, processes it, updates the model if necessary, and then updates the view to reflect the changes.

The controller manages the flow of data between the model and the view, helping to maintain separation of concerns and making the application more modular and maintainable.



MVC - View

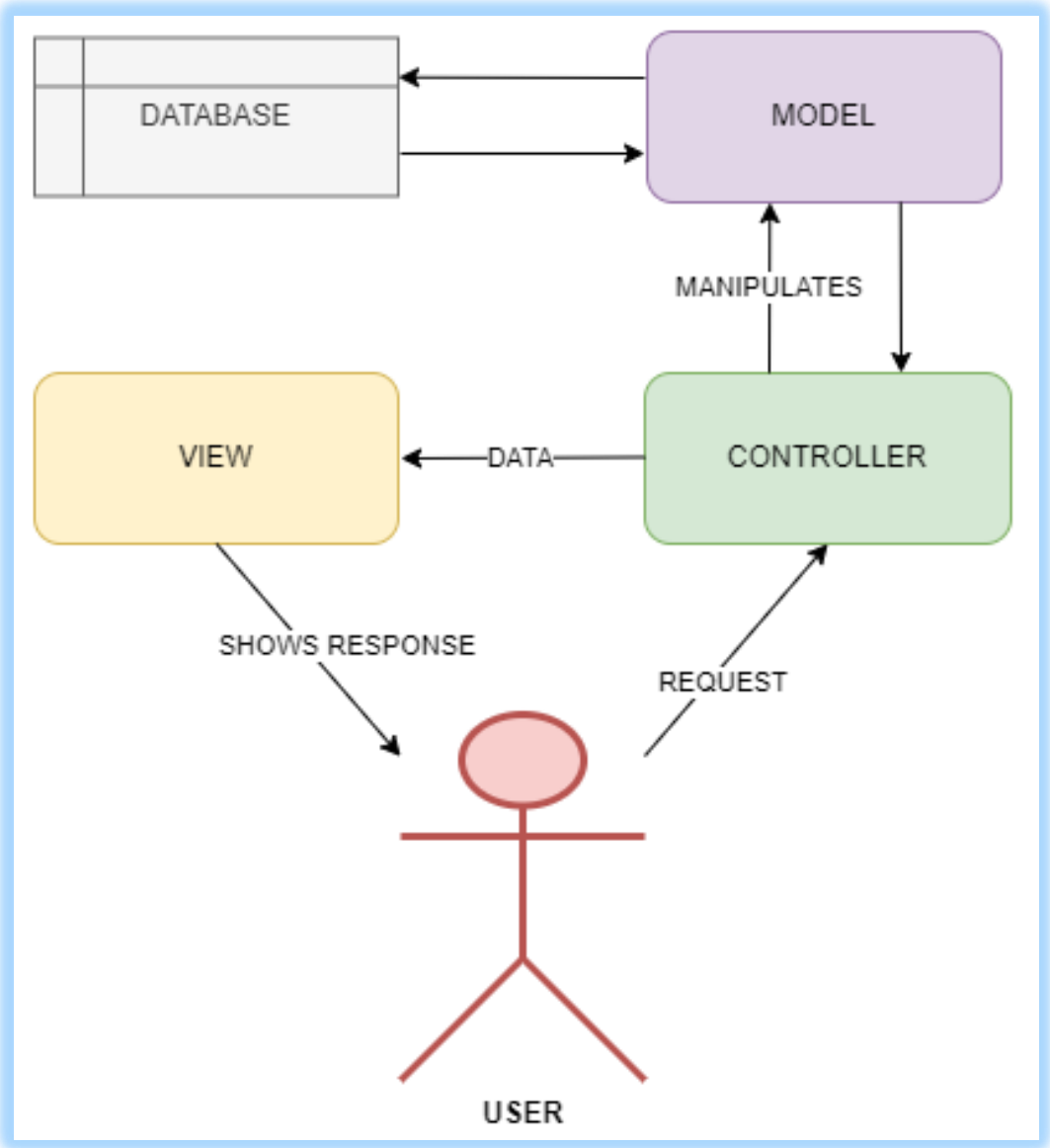
View - represents the user interface and presentation layer. It displays information to the user and sends user input to the controller for processing.



MVC - Model

Model represents the data and business logic of the application.

It is responsible for managing the data, processing user inputs, and responding to requests from the view.



3 Designing application

About our example application

Our example application **Book Rental** is designed for librarians, and it is supposed to be used in small libraries. It helps with managing the process of renting books to readers.

Source: https://www.rawpixel.com/image/7419720/vector-book-public-domain-illustrations

DESIGNING WEB APPLICATION

- User Roles,
- Use Cases & Scenarios,

- Items/Object in real world,

- Planning Endpoints (URLs)

- Views: Subpages / Forms in app,
- Flow between pages,

User Roles

Source https://openclipart.org/detail/314949/librarian-2

Librarian

Reader

Librarian

Librarian

- Registering the new users,
- Adding new books,
- Editing data about books / readers,
- Renting books to users,
- Checking how many books are rented by particular user,
- Handling returned books in system.

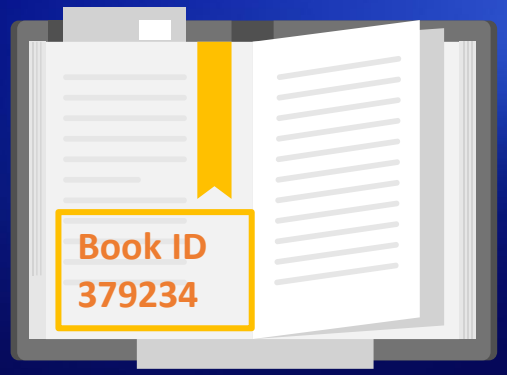
Reader

Reader

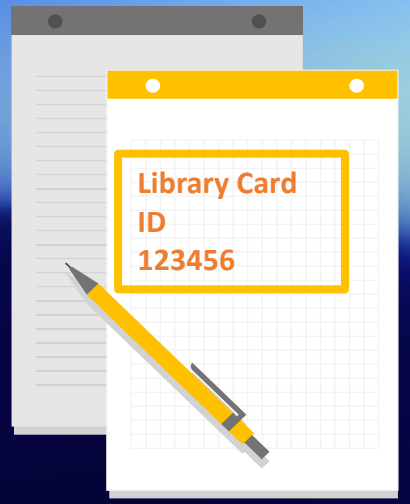
- Searches available books in Library,
- Checks which books are currently rented by him.

Items / Objects in system

Real–world objects



Book

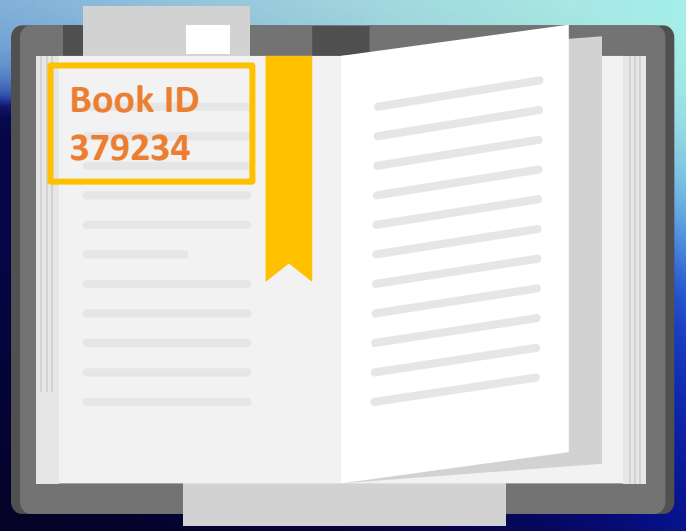


Library card ID

RENTED

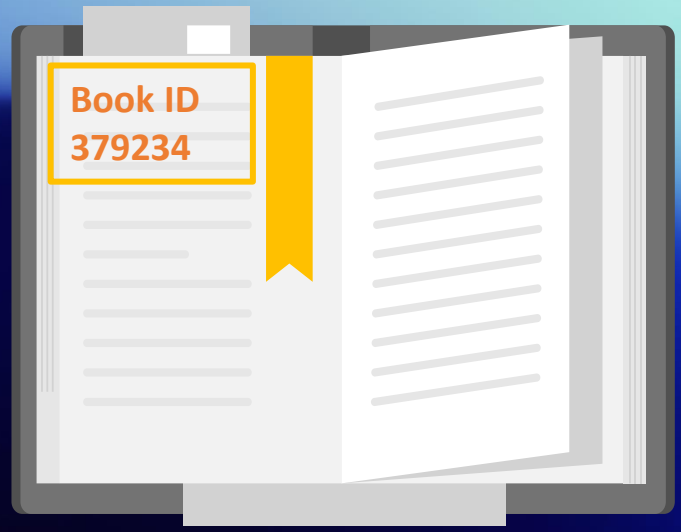
Book

- Usually inside any library we have many books on the shelves,
- Each book **has unique book ID**,
- Each book has **title** and **author.**



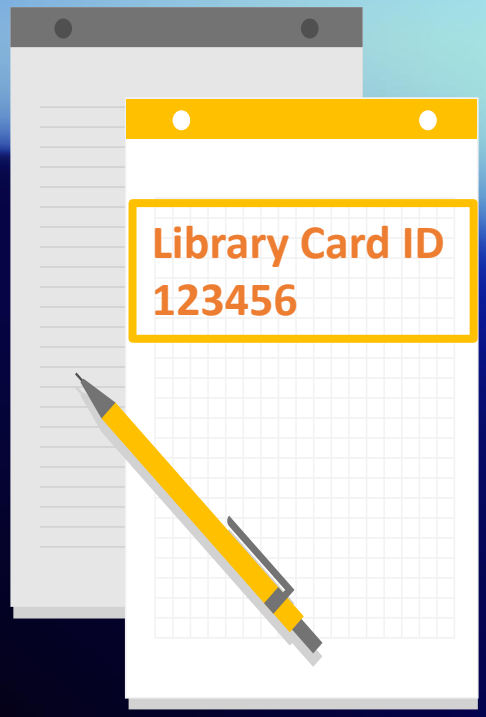
Book

- Other important properties are **ISBN** and **year of publication**,
- Book can have a few **categories** (for example: novel, romance, horror, fantasy, science-fiction...),
- Finding a particular book in big store might be difficult. To help Librarians with this task we added field **shelf**. This field represents physical location of book in library building.

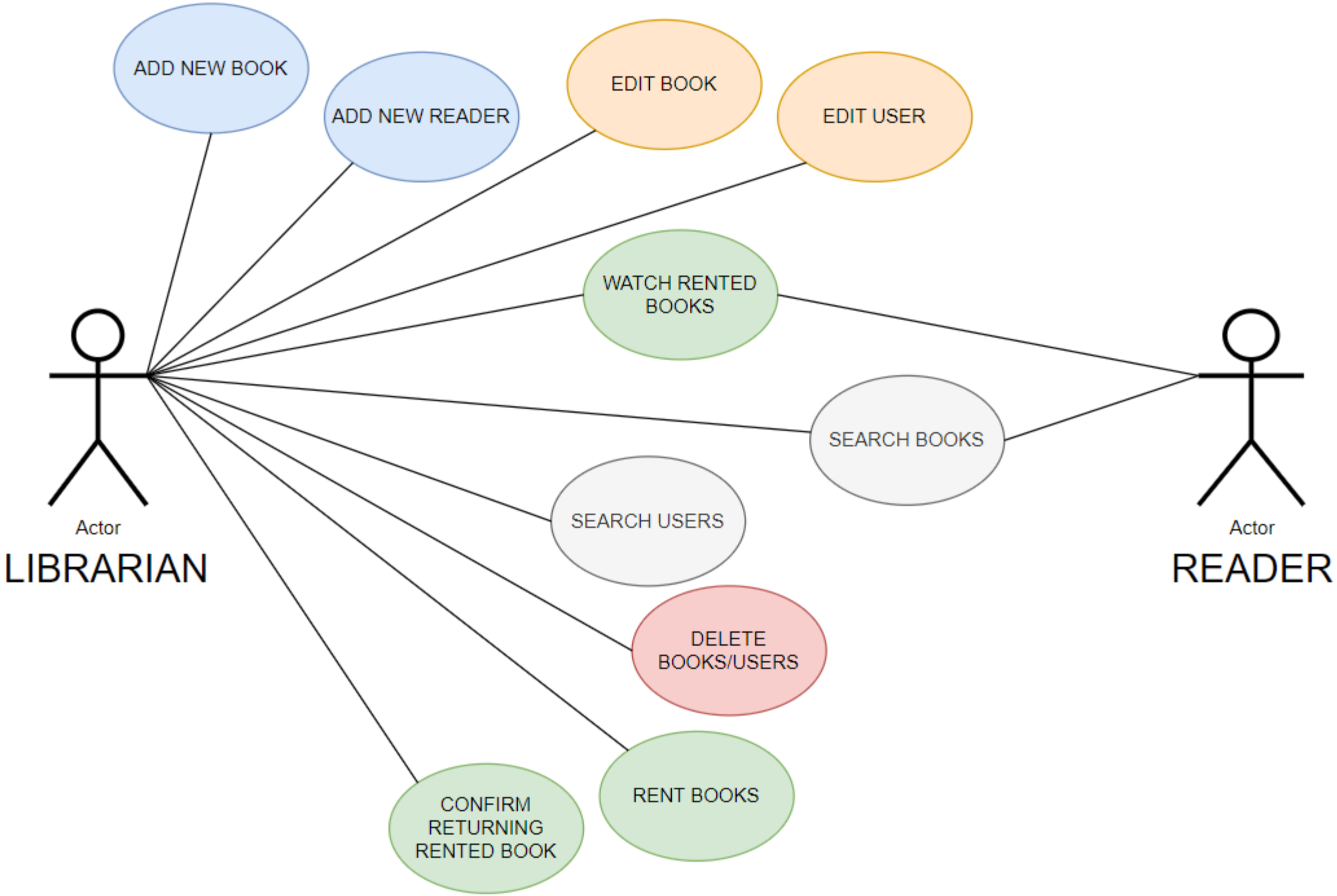


Library card

- Each user registered in library gets the printed document: “Library Card“,
- Each Library CARD has **unique ID**,
- The same ID is used in our system. Book Rental application identifies users by Library CARD ID.



Use cases


ADD NEW BOOK
ADD NEW READER
EDIT BOOK
EDIT USER
WATCH RENTED BOOKS
SEARCH BOOKS
SEARCH USERS
DELETE BOOKS/USERS
RENT BOOKS
CONFIRM RETURNING RENTED BOOK
Actor
LIBRARIAN
Actor
READER

Wireframing

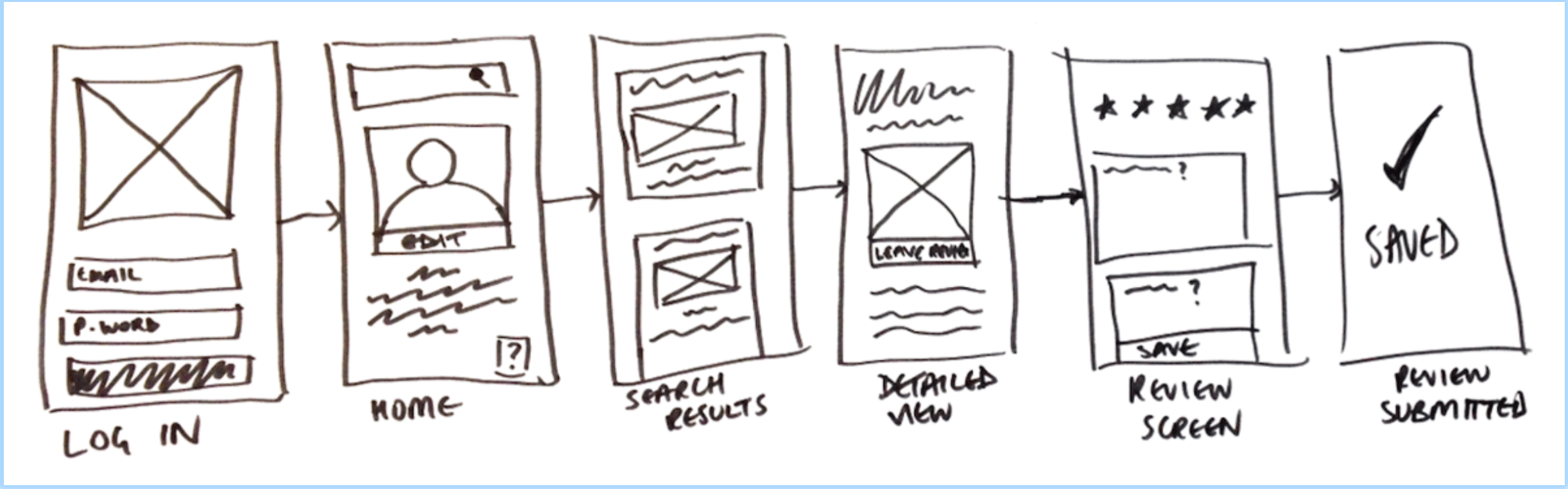
WIREFRAMING

A way of visualizing your future application.

Outline/draft/sketch of basic user interface for your website (views/subpages).

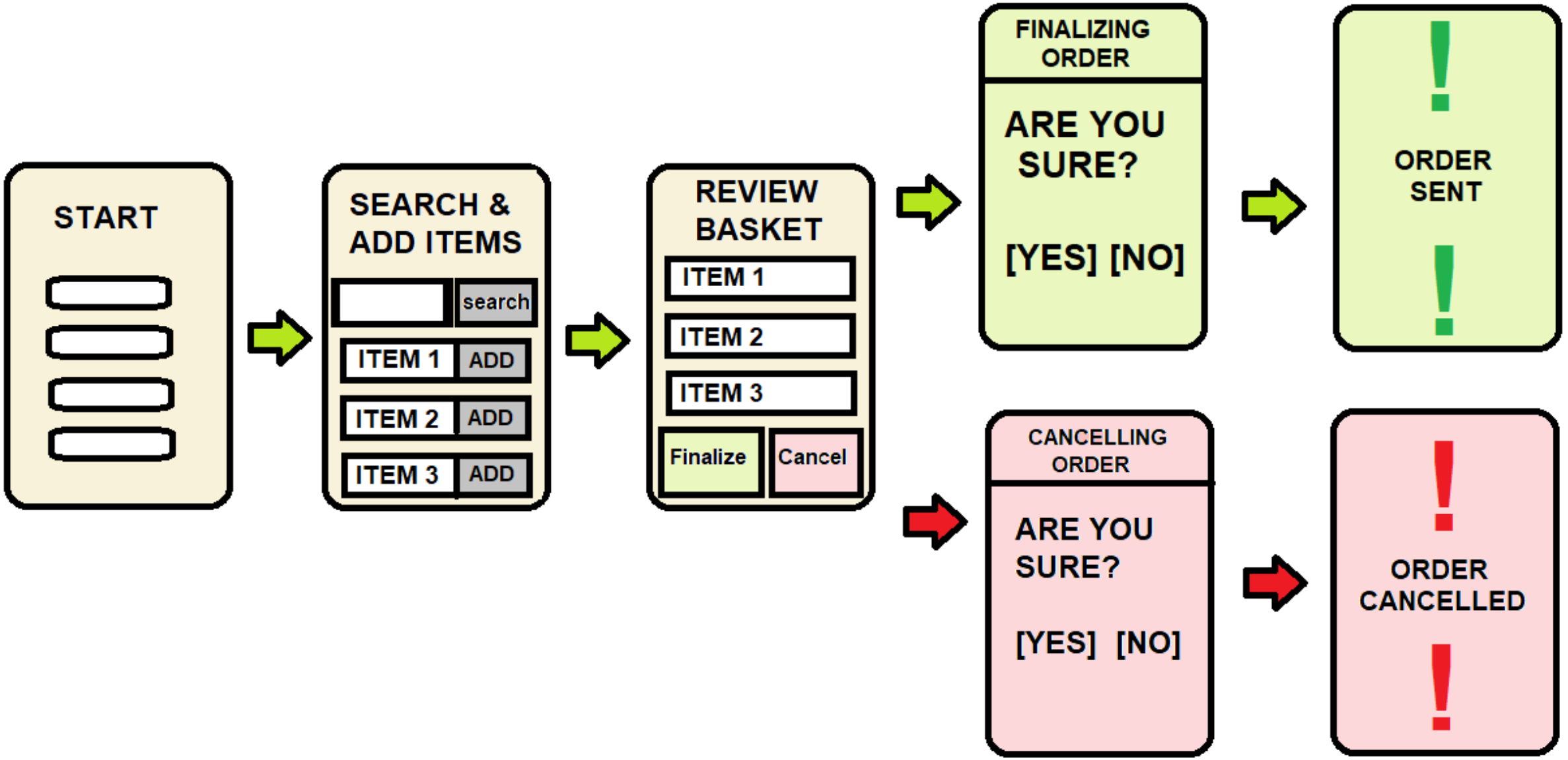
Wireframes can be drawn on paper or created digitally in some applications.

WIREFRAMING



- Source: **https://careerfoundry.com/en/blog/ux-design/what-is-a-wireframe-guide/**

WIREFRAMING



Scenarios

Yellow – activities in "real world", for example putting book on the shelf

White – virtual activities in Book Rental application

Scenario 1: Library bought a new book

Librarian

<<INSIDE LIBRARY>>

- Librarian starts the Book Rental application, logs in and clicks “Add New Book” button.
- Librarian fills in the form and confirms operation. Book is added to the system.
- Librarian puts the new book on the shelf.

Scenario 2: New reader came to the library

<<INSIDE LIBRARY>>

Reader

- New Reader arrived. The person wants to join library.
- Librarian starts the Book Rental application, logs in and fills in the form "Add New User". Unique ID is generated.
- Library CARD ID is printed by Librarian. Document is given to the user.

Librarian

Scenario 3: Reader wants to rent a book

<<INSIDE LIBRARY>>

Reader

- Reader comes in into the library building. Guest informs librarian about the book he wishes to rent.
- Librarian searches book in System. The book is available.
- Librarian takes the book from the shelf.
- Librarian registers renting book in the system using ID of Library CARD showed by user.

Librarian

Scenario 4: Reader comes to library and returns book

<<INSIDE LIBRARY>>

Reader

- Reader comes in into the library building and returns the book.
- Librarian searches book in System and registers that the book is back.
- Librarian puts the book on the shelf.
- From now on other readers might borrow this book.

Librarian

Scenario 5: Pile of returned books, readers already left

<<INSIDE LIBRARY>>

- In the morning many readers came to the library building. Librarian was very busy...
- In meanwhile a few readers put their books on the table and left the library.
- Now, we have a pile of returned books,
- One by one book, Librarian opens the books and finds the rubber stamp with book id,
- For each book librarian finds record in the system under "Rented Books" and clicks "Return Book" button.

Librarian

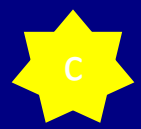
Scenario 6: Reader logs in to Rental App from home,

<<ONLINE>>

Reader

- Reader needs certain book for his university work.
- Reader logs in to the system,
- Reader goes to "Search Book" and types the query.
- System displays the information about the desired book.
- Book is available in the library and not rented, so user will visit the building in person,

Limitations



Limitations

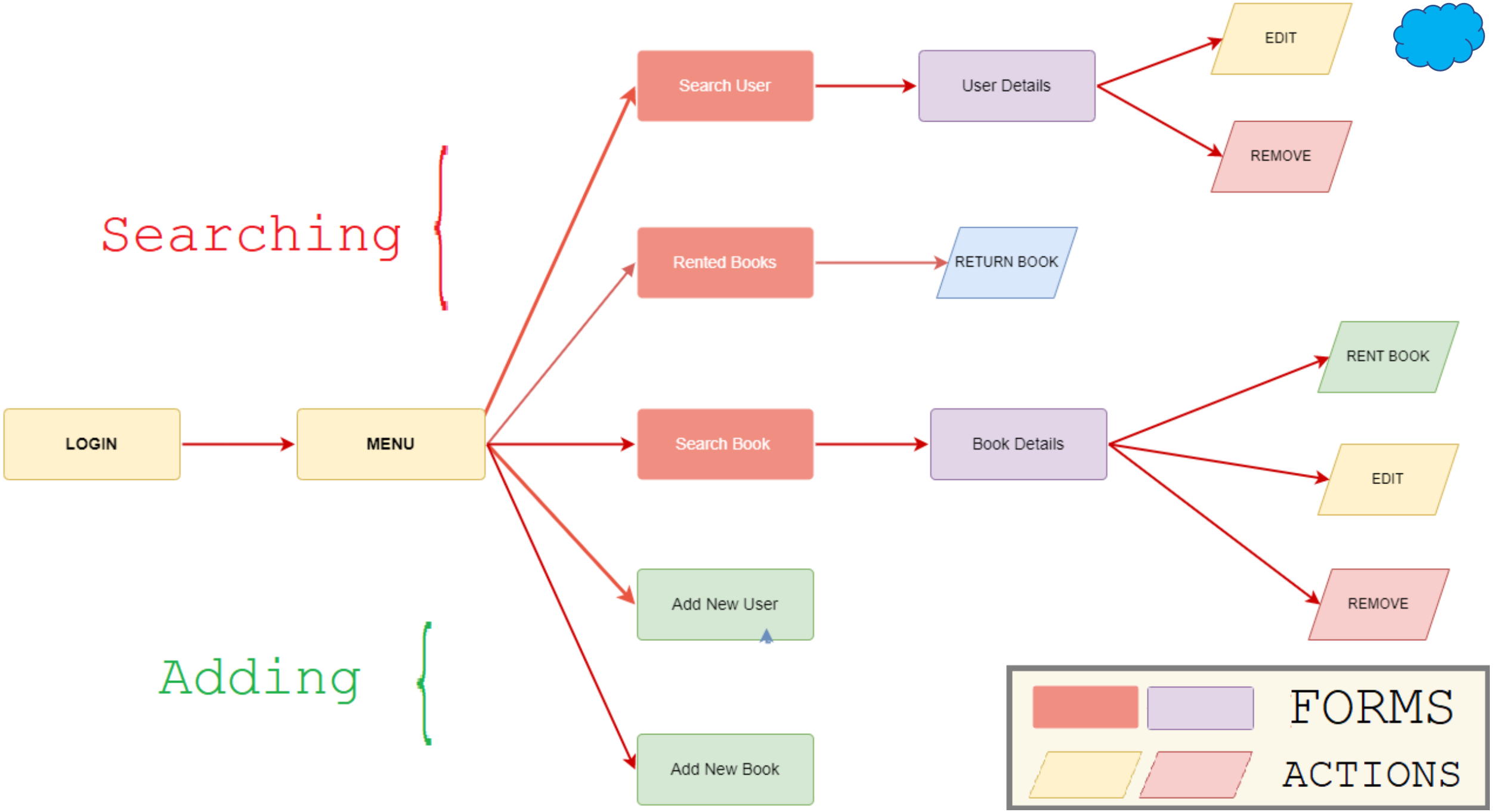
- Sometimes we cannot handle all possible situations,
- We need to think about limitations of our program – whicj scenarios will not be supported,

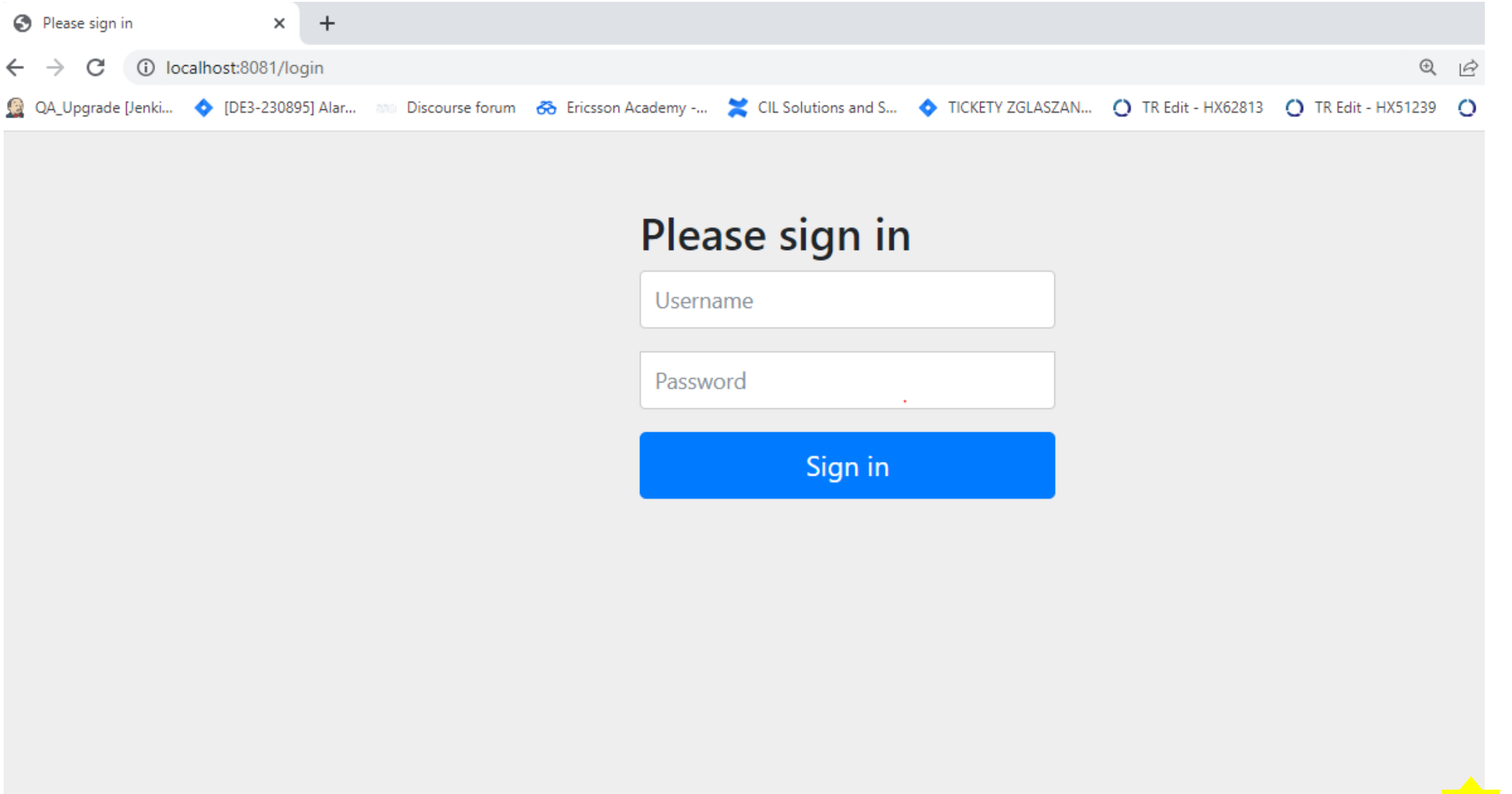
Endpoints naming

REST URLs/Endpoints naming – Nouns,

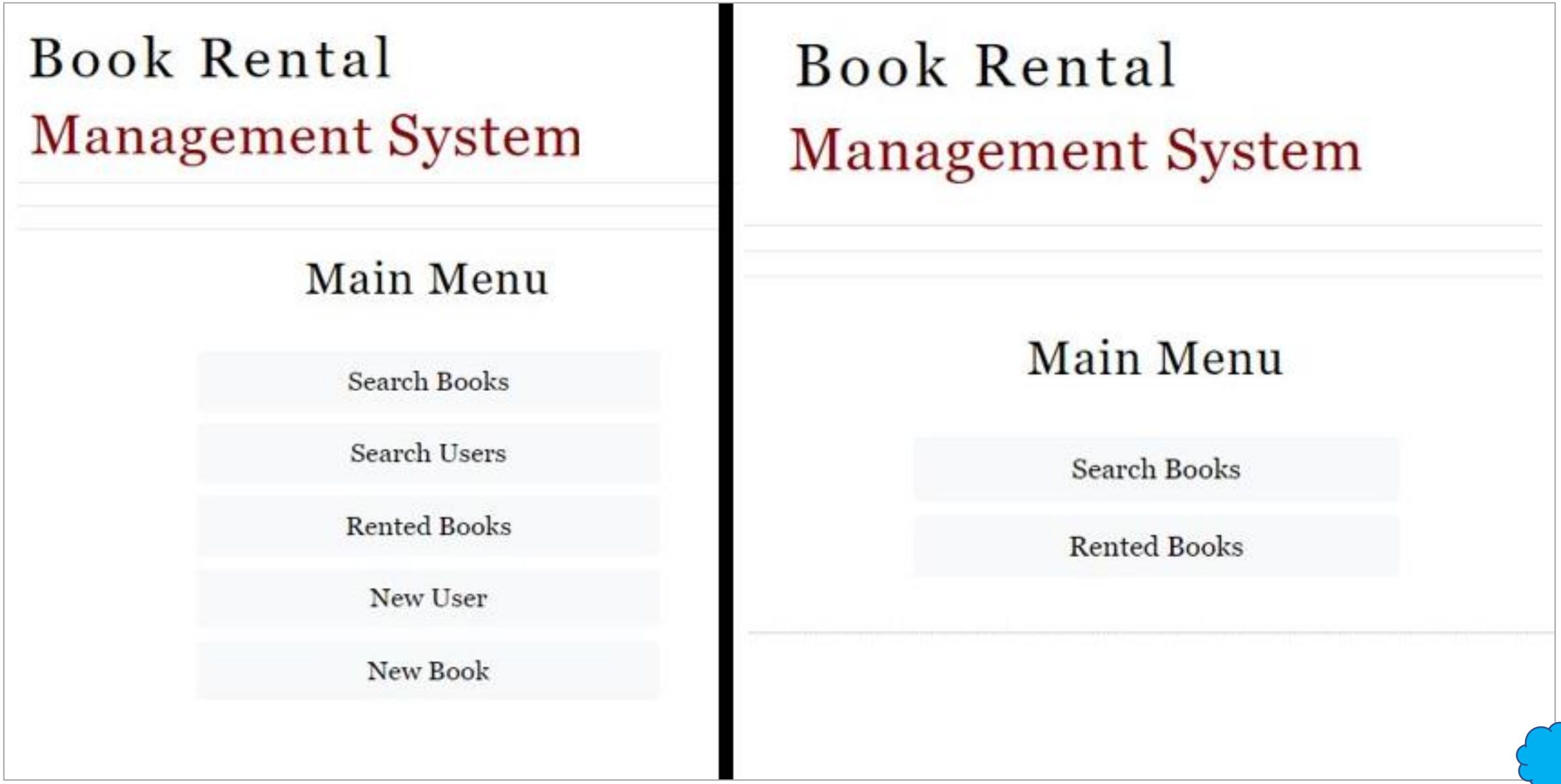
URL / ENDPOINT	HTTP METHOD	ACTION
/users	GET	List with **many** users,
/users	POST	Adding **new** user,
/users/{id}	GET	Details page for **one** user **having certain ID**,
/users/{id}	PUT	Updating one **user** having certain ID,
/books	GET	List with **many** books,
/books	POST	Adding **new** book,
/books/{id}	GET	Details page for **one** book **having certain ID**,
/books/{id}	PUT	Updating one **book** having certain ID,
/rented-books	GET	List with **many** books **that are rented,**
/books-rented-by-user/{userId}	GET	List with **many** books **rented by certain user** (user has certain userId)

Forms / Views


Searching
Adding
LOGIN
MENU
Search User
User Details
EDIT
REMOVE
Rented Books
RETURN BOOK
Search Book
Book Details
RENT BOOK
EDIT
REMOVE
Add New User
Add New Book
FORMS
ACTIONS


Please sign in
localhost:8081/login
QA_Upgrade [Jenki...
[DE3-230895] Alar...
Discourse forum
Ericsson Academy -...
CIL Solutions and S...
TICKETY ZGLASZAN...
TR Edit - HX62813
TR Edit - HX51239
Please sign in
Username
Password
Sign in

Menu for Librarian vs Menu for Reader



Main Menu | Search Books | Search Users | Rented Books | New Book | New User | **Logout**

Adding New Book

Book title: Title

Author: Author

ISBN: ISBN

Description:

Description

Shelf (in library): Shelf

Year of publication: 0

Categories:

- SCHOOL_BOOK
- FANTASY
- POETRY
- DRAMA
- ROMANCE
- MANAGEMENT_AND_ECONOMY
- NOVEL
- FAIRYTALE
- SHORT_STORY
- THRILLER
- BIOGRAPHY
- HISTORY
- CRIME_STORY
- ADVENTURE

Create Book

Show Book Info

Book Id:	**300005**
Title:	W Pustyni i W Puszczy
Author:	Henryk Sienkiewicz
Shelf in Library Building:	ADVENTURE
ISBN:	978-83-240-2959-4
Categories:	HISTORY,ADVENTURE
Description:	Przygody Stasia i Nel na pustyni oraz w dżungli podczas Powstania Mahdiego,
Year of publication:	2014

Edit Book Remove

Book Rental Management System

Main Menu | Search Books | Search Users | Rented Books | New Book | New User | **Logout**

Adding New User

Name and Surname:

UserName

Login:

Login

User Role: READER

Create User

Book Rental Management System

Main Menu | Search Books | Search Users | Rented Books | New Book | New User | **Logout**

Search Users

Username (name/surname):

Search

Username:	Library CARD ID:	User Role:	Operations:		
JanKowalski	100000	READER	Rented Books	View User Info	Edit User Info
Anna Kowalska	100001	READER	Rented Books	View User Info	Edit User Info
Maria Komornicka	100003	READER	Rented Books	View User Info	Edit User Info

Add Library User

Book Rental Management System

Main Menu | Search Books | Search Users | Rented Books | New Book | New User | **Logout**

Search Books

Book title: Find book by title

Category: Find book by category

Author: Find book by author

Search

	Title:	Author:	Book Id:	Shelf:	
[B]	W Pustyni i W Puszczy	Henryk Sienkiewicz	300005	ADVENTURE	Edit View Info
(F)	W Puszczy i Nie W Puszczy	Tomasz Kaniol	300006	HORROR	LIB CARD ID: Rent Book | Edit View Info
[B]	Zaraza	Jan Malkowski	300000	HORROR	Edit View Info

Add New Book

Book Rental Management System

Main Menu | Search Books | Search Users | Rented Books | New Book | New User | **Logout**

Search Rented Books

Book ID: [ ] Search

Title:	Author:	Book Id:	Operations:
Zaraza	Jan Malkowski	300000	Return Book
W Pustyni i W Puszczy	Henryk Sienkiewicz	300005	Return Book

Add New Book

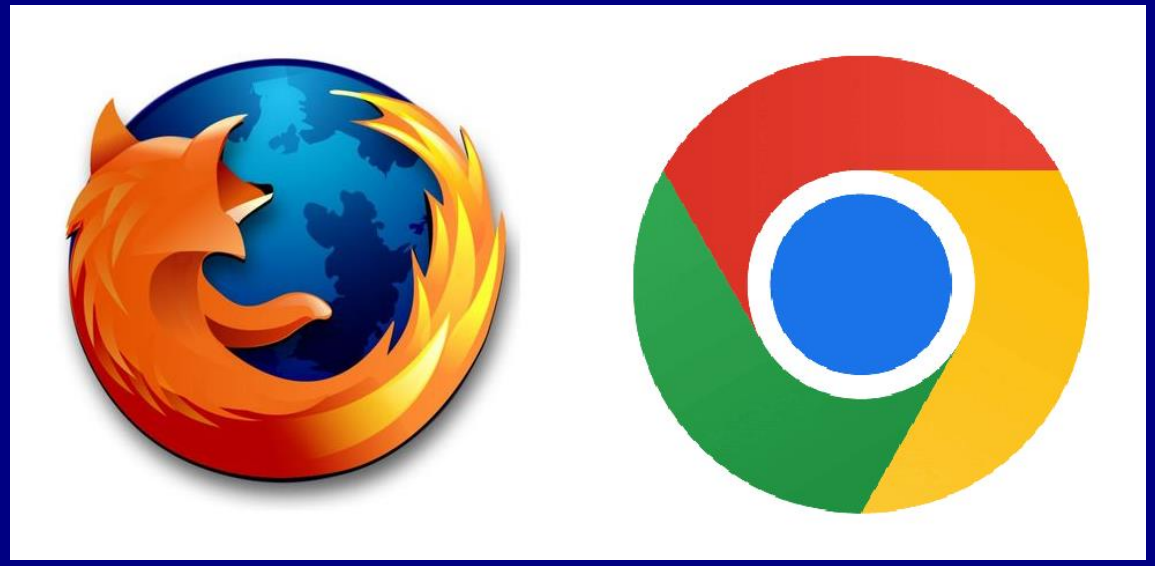
Prototype

Prototype

- Usually it is better to show something than just talk,
- Visualization helps our clients with understanding our concepts,
- It is good to start developing your application as **prototype** with limited functionality. Then, your stakeholders can comment your vision on early stages.
- We can design dummy application using frontend technologies (HTML, Bootstrap, Javascript)

4 Useful Tools

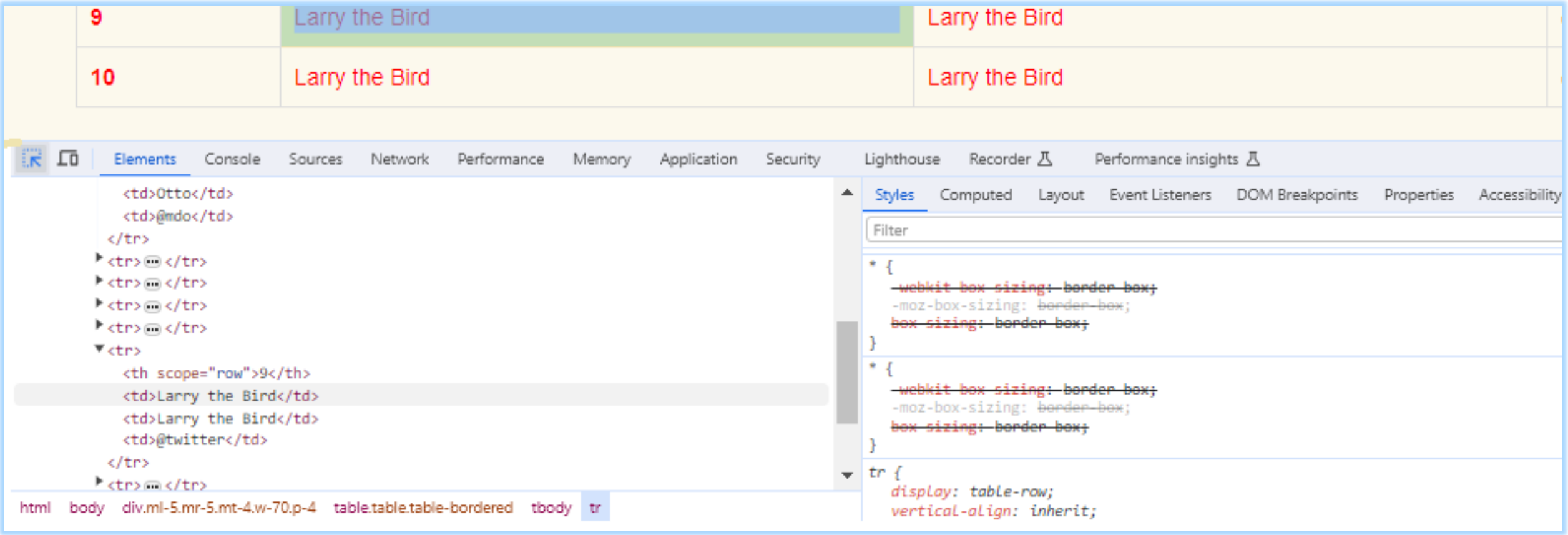
Developer Tools – Web Browsers



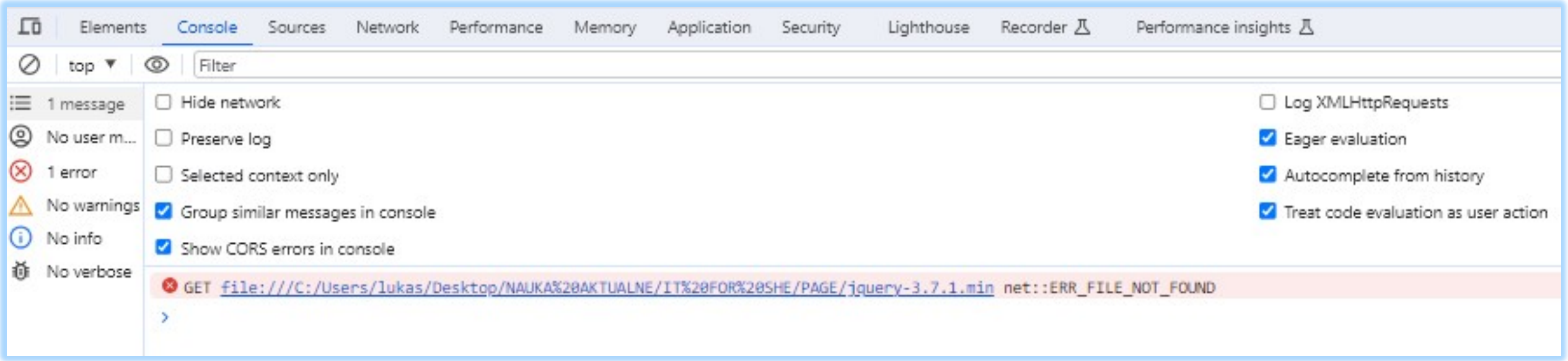
WEB BROWSER DEVELOPER TOOLS

- What can we do using Developer Tools?
- 1) Modifying the look of elements (css style),
- 2) Finding elements on page,
- 3) Debugging javascript code,
- 4) Calling javascript code,
- 5) Network: checking what data was sent/received (HTTP requests for example JSON messages),

DEVELOPER TOOLS - INSPECT



DEVELOPER TOOLS - CONSOLE



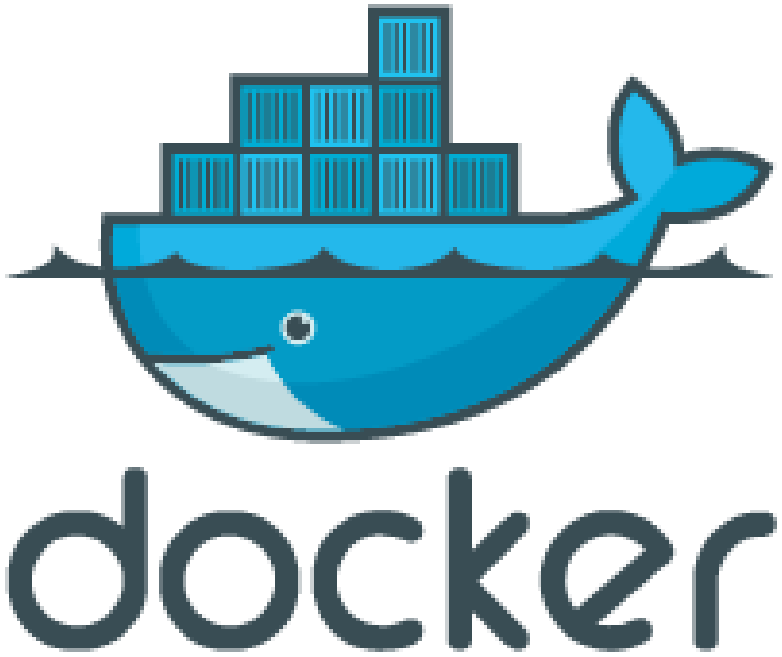
DEVELOPER TOOLS - NETWORK

- We can check what kind of requests/responses were sent:

- GET/POST calls, endpoints,

- JSON data,

DOCKER



provides resources to run your application

What is a docker ?

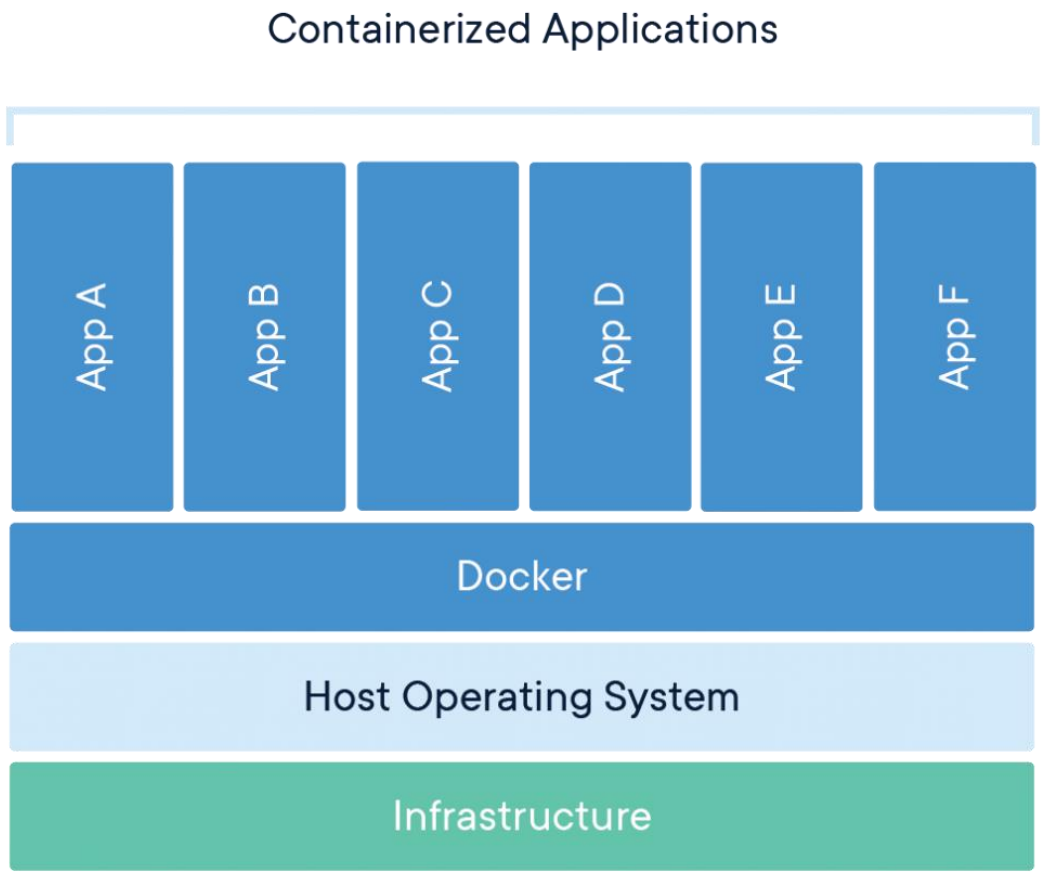
Docker is one of the most popular platform for developing, shipping, and running applications.
Docker enables you to separate your applications from your infrastructure so you can deliver software quickly.

With Docker, you can manage your infrastructure in the same ways you manage your applications.
By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

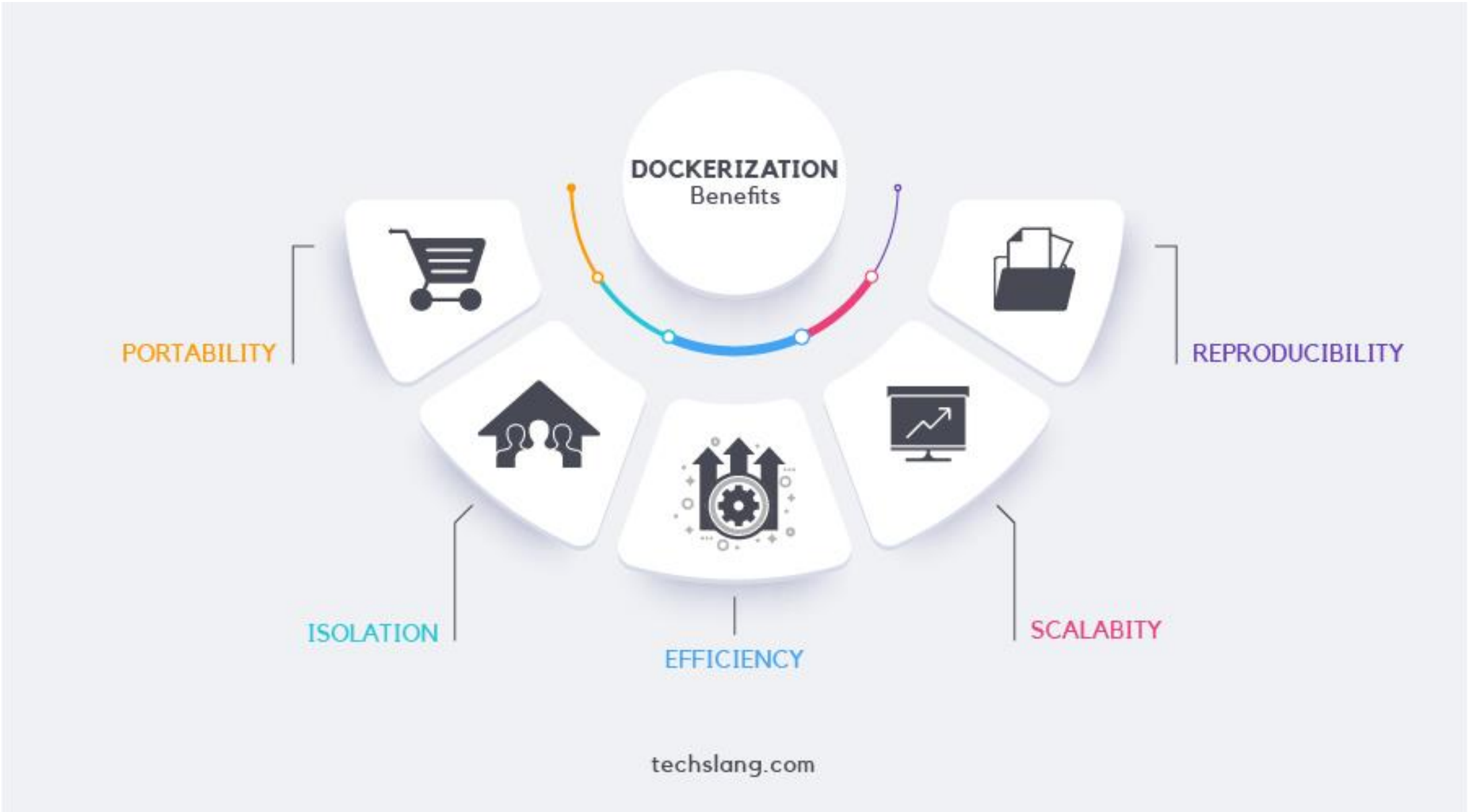
What is dockerization ?

Dockerization is the practice of bundling an application and its dependencies into a standardized container.

Containers wrap both the application and its dependencies, guaranteeing uniform and repeatable deployment in various environments.



source : https://commons.wikimedia.org/wiki/File:Docker-containerized-and-vm-transparent-bg.png



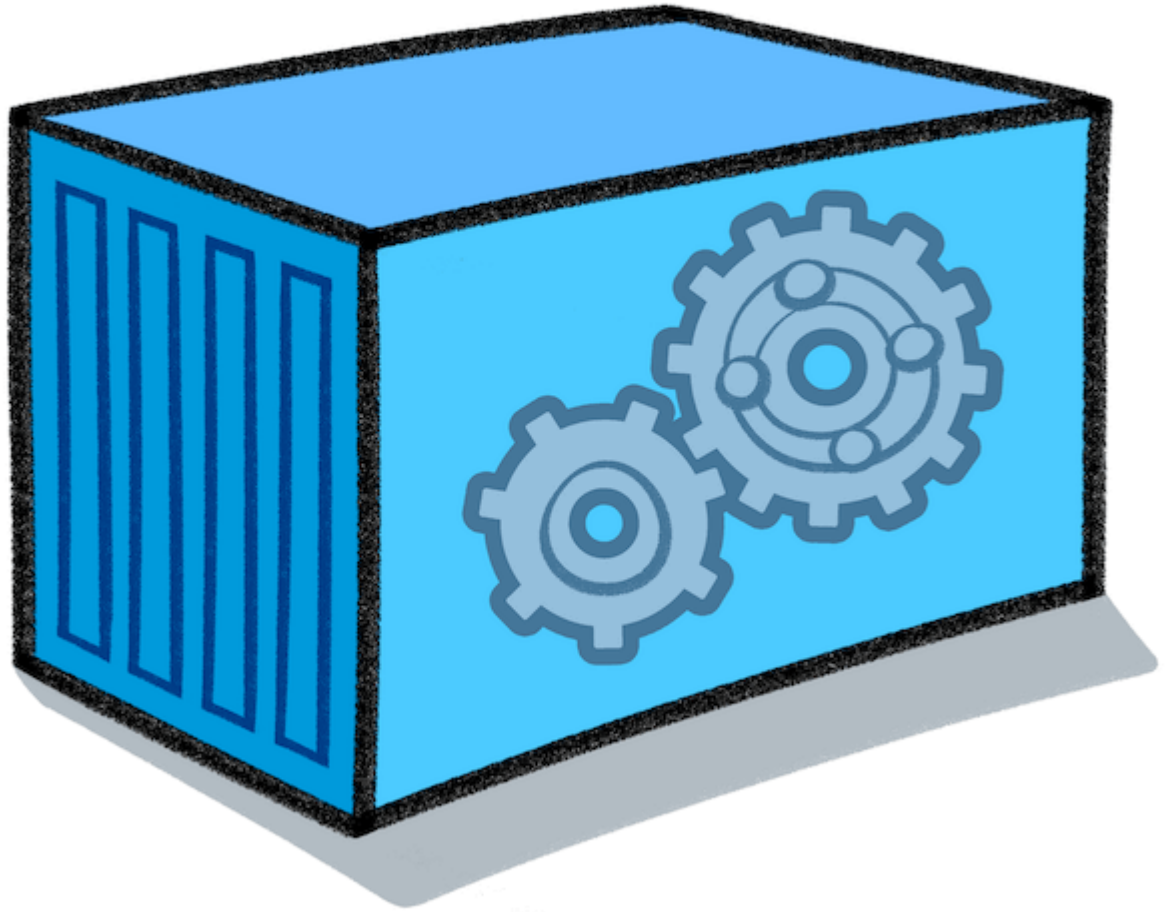
source : https://www.techslang.com/definition/what-is-dockerization/

Docker - containers

In short, a container is an independent entity placed on a small part of our hard drive.

This part of the disk contains files required for the proper launch and operation of the application/program/process.

It depends on what's in the container.

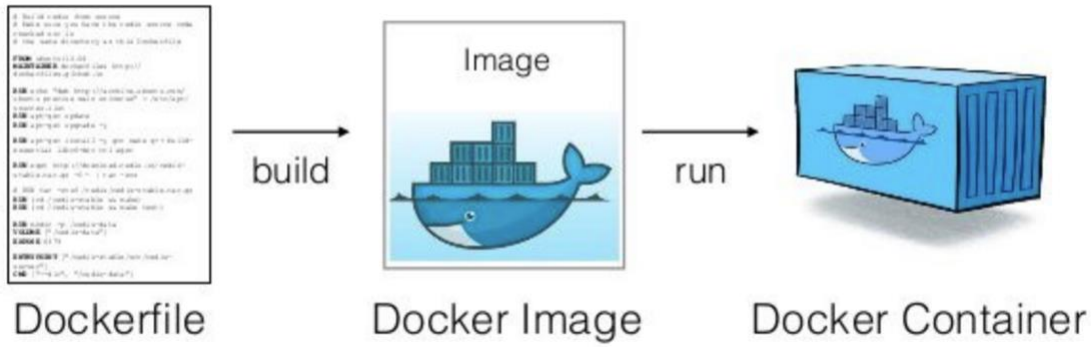


Each container gets some hardware from our machine just for its own use (disk, RAM, network, etc.). Thanks to this, the containers are completely independent of each other and the host computer.

Docker - images

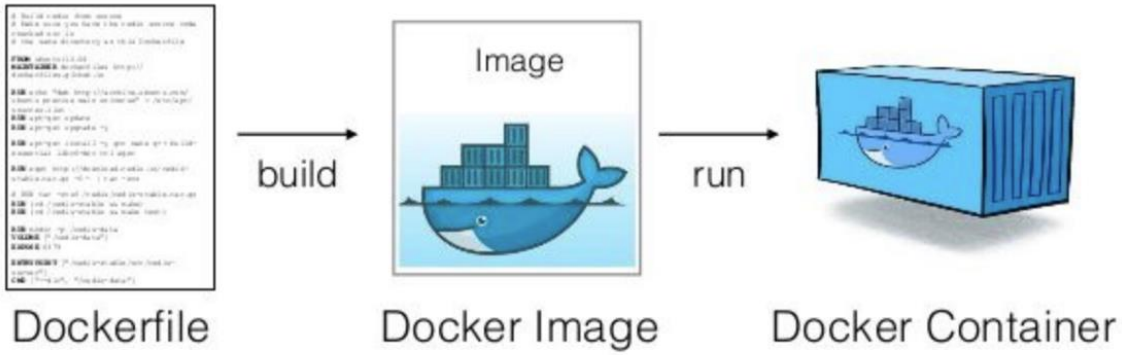
An image is a collection of software that runs as a container, containing a set of instructions for creating a container that can run on Docker.

Images are like compact packages that contain everything an application needs to run: code, libraries, and dependencies. They serve as a portable, consistent unit, ensuring the application runs reliably across different environments. Docker images streamline the deployment process and enhance scalability by isolating applications and their dependencies in a standardized way.



Docker – creating images

Dockerfile is a simple script that tells Docker how to build a Docker image. It specifies the steps to create the environment needed for your application to run. In essence, it's a set of instructions for Docker to follow when assembling.




```
FROM python:3.6.9
ENV PYTHONUNBUFFERED 1
ENV PIRSTPMNGR_USER user_name
ENV PIRSTPMNGR_PASSWORD password
ENV DATABASE_NAME db_name
ENV DATABASE_USER db_user
ENV DATABASE_PASSWORD ABCdef123


RUN mkdir /app
RUN mkdir /app/web
RUN mkdir /app/web/staticfiles
WORKDIR /app/web/staticfiles


RUN mkdir /dir_app
WORKDIR /dir_app
COPY requirements.txt /dir_app
COPY users_credentials /dir_app


RUN pip install -r requirements.txt
COPY . /dir_app
```

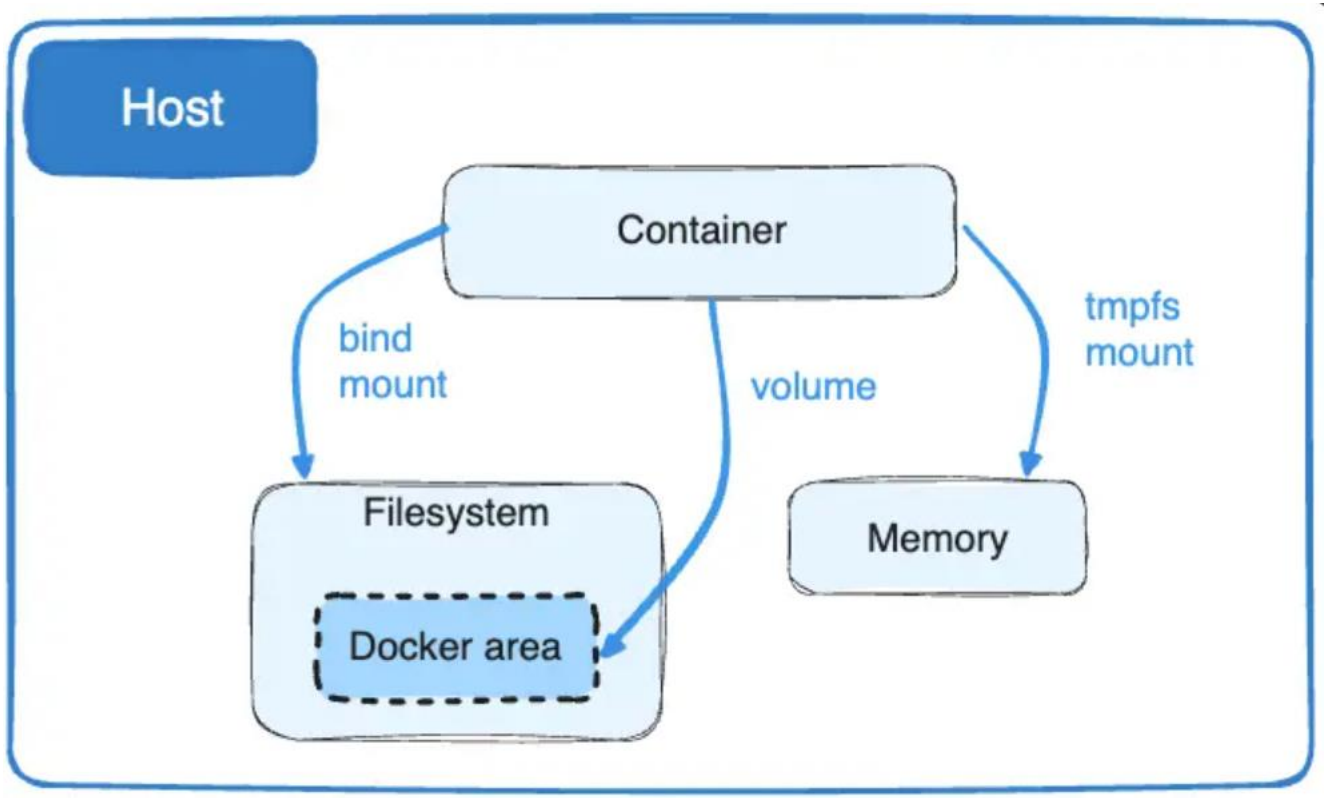
Dockerfile example

Docker - volumes

Volumes - allows us to map directories between the container and the machine on which Docker is running.

They are fully created and managed by Docker. Each new volume is stored on the host disk. This means that if the container is killed, our data will not be affected. It is worth noting that the volume data is completely isolated from the host system because it is managed by Docker.

Once a volume is created, it can be used by many containers at the same time.



source: https://docs.docker.com/storage/volumes/

Docker-compose

Docker Compose is a tool that lets you define and manage multi-container Docker applications. It uses a simple YAML file to configure the services, networks, and volumes for your application.

Essentially, Docker Compose simplifies the process of running and connecting multiple Docker containers, making it easier to manage complex applications with multiple components.

```yaml
version: "3.8"

services:
  db:
    image: postgres
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=postgres
  web:
    build: .
    command: gunicorn app.wsgi:application --bind 0.0.0.0:10020
    volumes:
      - /staticfiles:/home/app/web/staticfiles/
    env_file:
      - ./.env.prod
    ports:
      - "10020:10020"
    depends_on:
      - db

  nginx:
    build: ./nginx
    ports:
      - "10030:80"
    depends_on:
      - web
    restart: always
    volumes:
      - /staticfiles:/home/app/web/staticfiles/
```

Docker-compose example

5 Implementation - Backend

SERVER TECHNOLOGIES

- **Python**

- **Django,**

- **Javascript:**

- NodeJS,

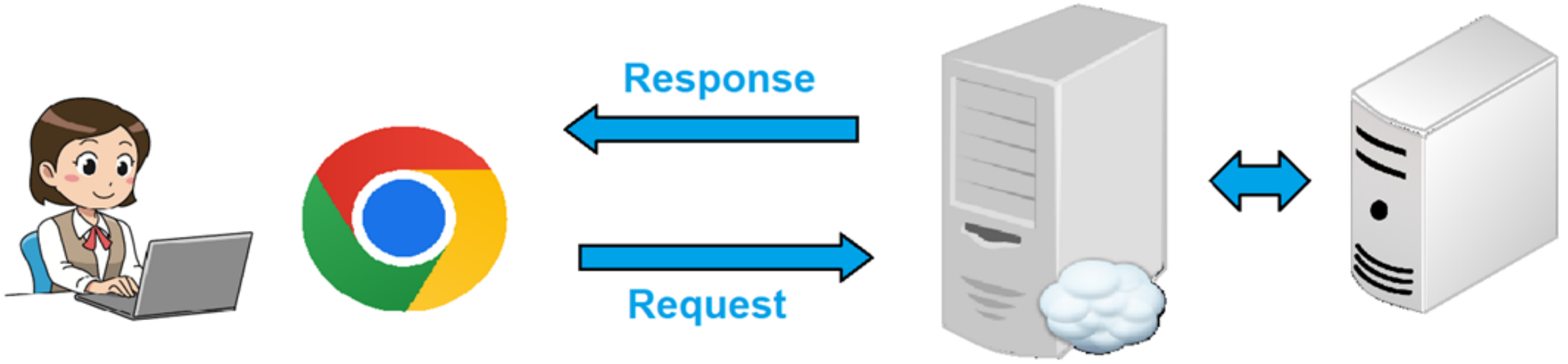
- **JAVA:**

- **Spring,**
- **Spring Boot,**
- **JSF**

- **PHP:**

- Symfony,

Client - Server



Browser

(Javascript, CSS, HTML)

Server

Server Side Code
(e.g Django/Python)

Database

(e.g SQL)

Created Image uses following images:
https://openclipart.org/detail/295783/business-man-working
https://commons.wikimedia.org/wiki/File:Tower_torre_pc_clon_server.svg
https://commons.wikimedia.org/wiki/File:Cloud_server.svg

DJANGO FRAMEWORK

framework that simplifies building application

Django Framework

Django is a high-level web framework for Python that follows the Model-View-Template (MVT) architectural pattern.

It simplifies web development by providing tools and conventions for common tasks, promoting code reusability, and emphasizing rapid development.

MVC vs MVT

Differences		
	MVC	MVT
Responsibilities	Controller handles user input and model updates	Template manages presentation and some control
View Role	Deals with presentation, separate from Model	Akin to MVC's Controller, managing request logic.
Template Concept	Controller and View can be more intertwined	Template is a distinct component using Django's language
Example Frameworks	General concept in frameworks like Spring (Java) or Ruby on Rails (Ruby)	Commonly used in Django(Python)

In essence, MVT and MVC differ in how they handle presentation, control, and component organization

Django – creating a web application

1. Install Django using the command:

in the terminal, run : pip install django.

2. Create a Project:

In the terminal, run: *django-admin startproject project_name*.

3. Create an App:

In the terminal, create a new app: *python manage.py startapp app_name*.

4. Define Data Models:

In the app's models.py file, define data structures.

5. Generate migrations for the models:

In the terminal, run: *python manage.py makemigrations*.

Django – creating a web application

6. Apply migrations to create the database schema:

python manage.py migrate.

7. Create Views:

In the app's views.py file, define views for business logic.

8. Create HTML Templates:

Create a templates folder inside the app and place HTML files.

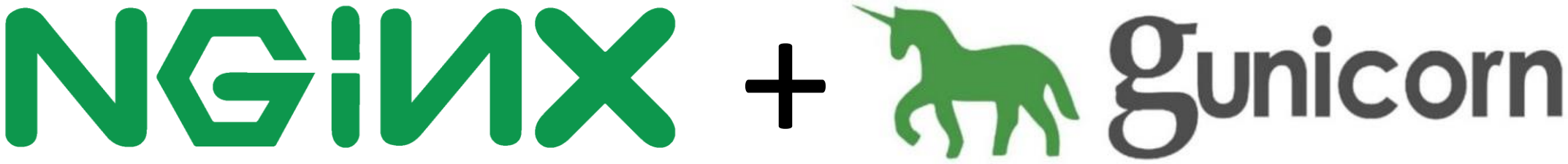
9. Configure URLs:

In the app's urls.py file, map URLs to views.

10. Run the Development Server:

Start the Django development server: *python manage.py runserver*.

Supporting Technologies



service and management of the web application

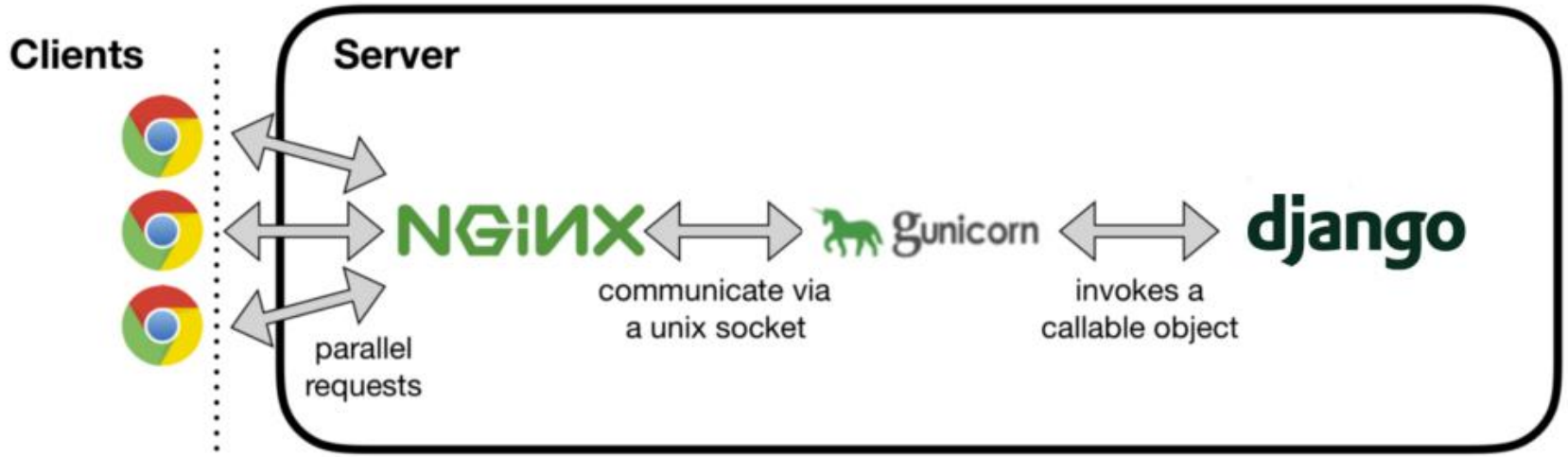
NGINX

Nginx is a high-performance web server and reverse proxy commonly used in web applications. It serves static content, acts as a reverse proxy to distribute traffic among multiple servers, and handles SSL/TLS termination.

Nginx improves application performance, scalability, and security by offering features such as caching, access control, and WebSocket support.

Gunicorn

Gunicorn is a WSGI (Web Server Gateway Interface) HTTP server for running Python web applications. It serves as a bridge between web frameworks (e.g., Flask, Django) and the web server, handling incoming requests and managing communication between the application and the server. Gunicorn is widely used in web applications to provide a reliable and efficient deployment solution for Python-based web services.



source : https://medium.com/analytics-vidhya/dajngo-with-nginx-gunicorn-aaf8431dc9e0

6 Frontend

Technologies used for frontend

TEMPLATES

Bootstrap

is **open-source front-end development framework** for the creation of websites and web apps.

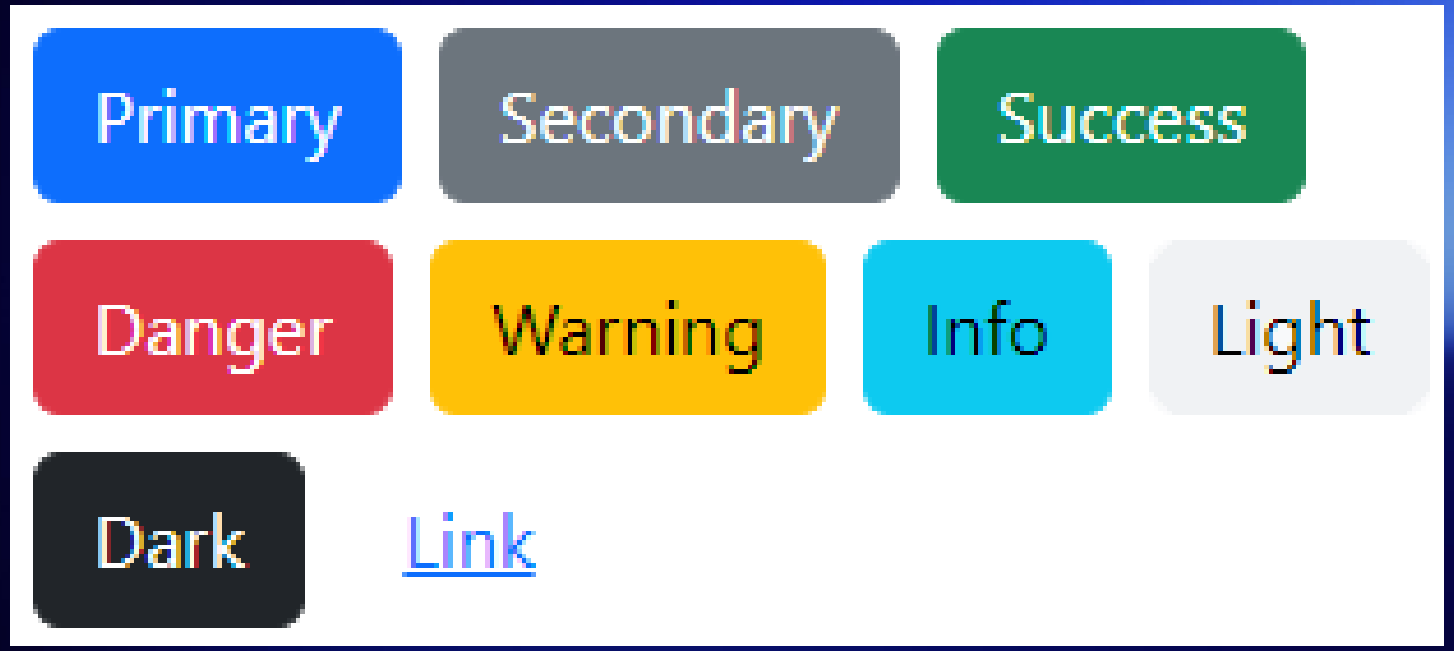
Bootstrap simplifies creating responsive, mobile-first front-end programs.

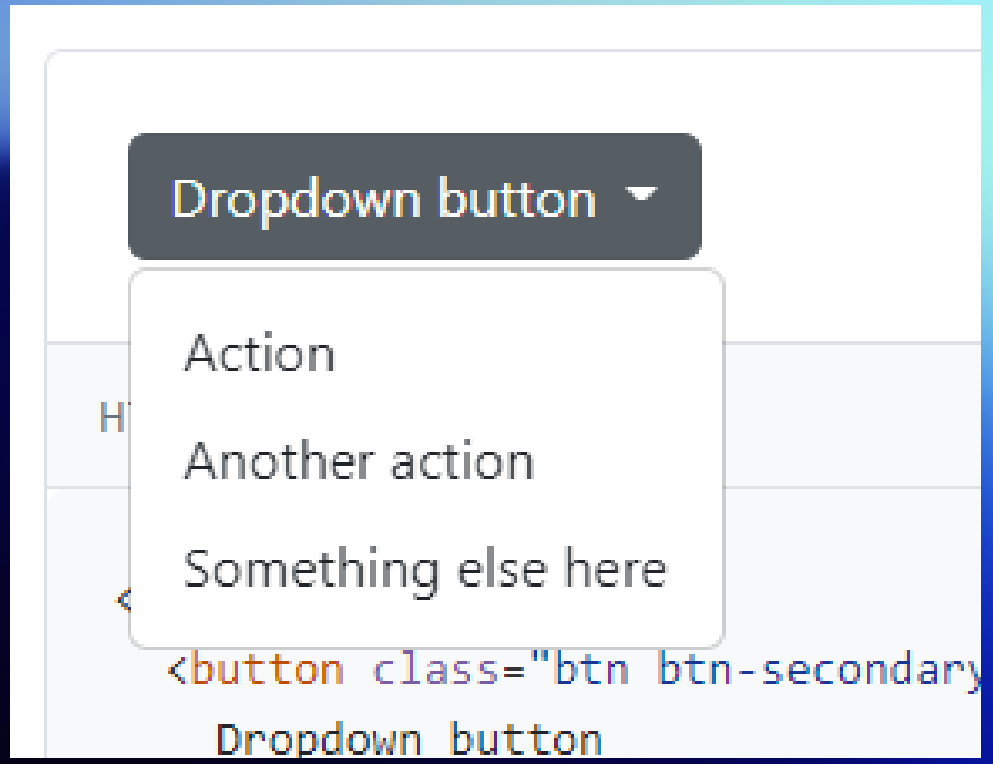
Bootstrap help us with appearance of our applications.

Bootstrap uses HTML, CSS, and JavaScript.

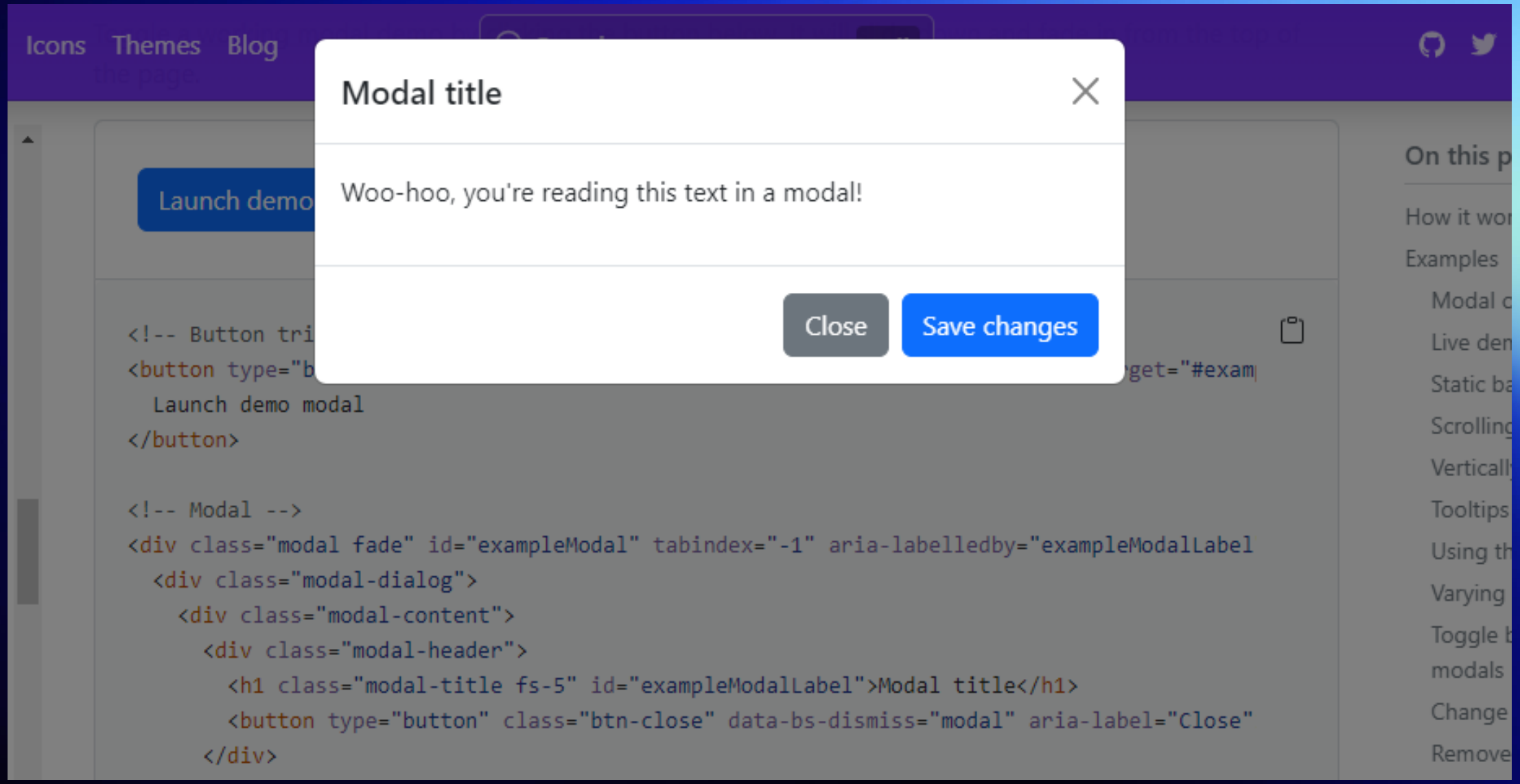
Bootstrap has ready solutions for many elements: buttons, alerts, modals, tabs, navigation bars, toolkits, dropdowns and many more...

Bootstrap


Primary
Secondary
Success
Danger
Warning
Info
Light
Dark
Link


Dropdown button
Action
Another action
Something else here
<button class="btn btn-secondary
Dropdown button

Bootstrap


Icons
Themes
Blog
Launch demo
Modal title
Woo-hoo, you're reading this text in a modal!
Close
Save changes
<!-- Button tri
<button type="b
get="#exam
Launch demo modal
</button>
<!-- Modal -->
<div class="modal fade" id="exampleModal" tabindex="-1" aria-labelledby="exampleModalLabel

<h1 class="modal-title fs-5" id="exampleModalLabel">Modal title</h1>
<button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"

On this p
How it wor
Examples
Modal c
Live den
Static ba
Scrolling
Verticall
Tooltips
Using th
Varying
Toggle b
modals
Change
Remove

JQuery

is a **feature-rich JavaScript library**.

It simplifies operations like: manipulation of HTML document (DOM operations), animations, event handling, AJAX.

Usually, no changes in HTML tags are required.

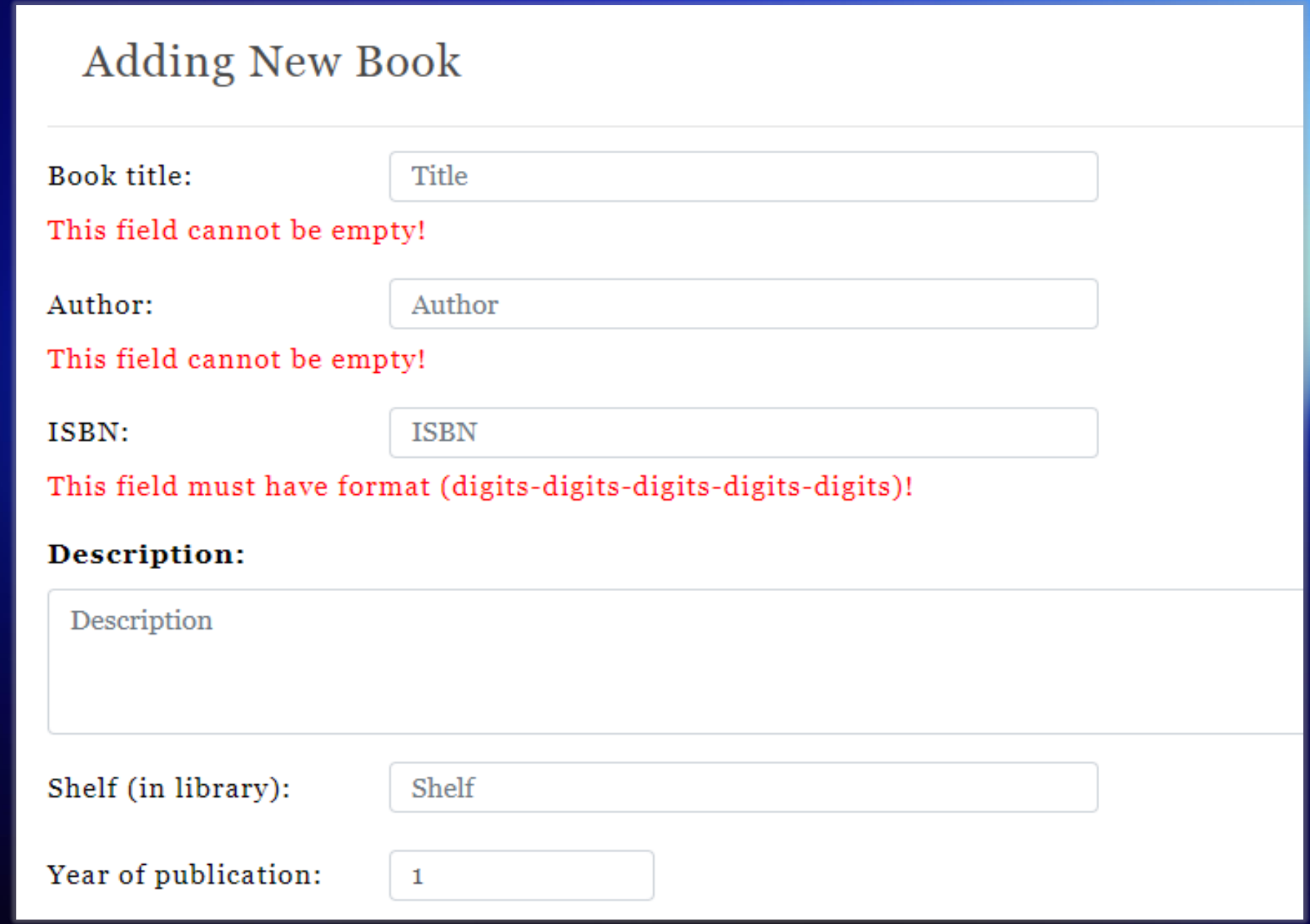
In our **Book Rental** application **JQuery helps us with form validation on client side.**

JQuery example

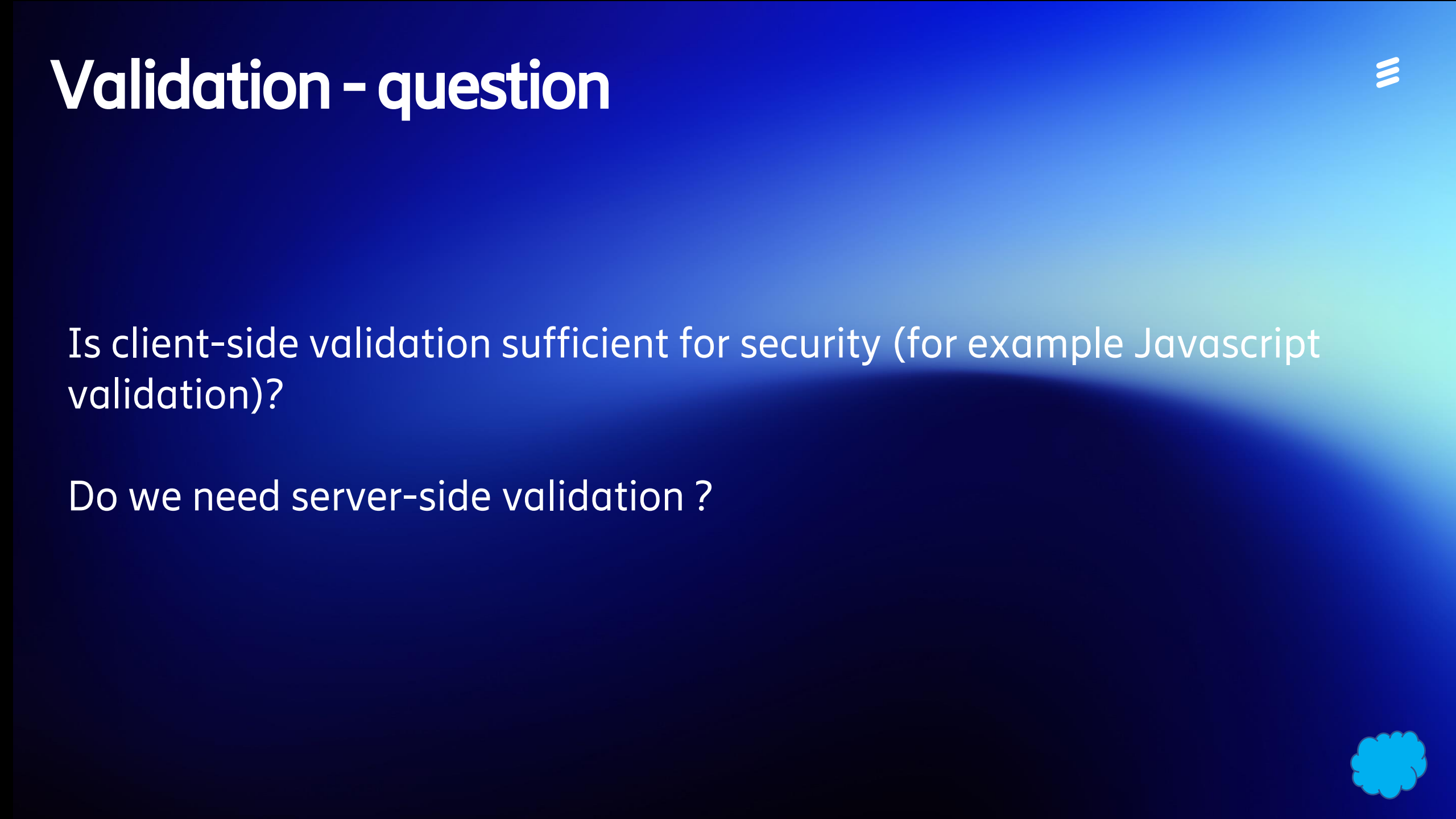
```
$(function() {
    $('#add-user-form').on('submit', function() {
        clearPreviousValidationErrors();
        var isValidationOk = true;
        isValidationOk = validateFieldNonEmpty("userName","add-user-form");
        if(!isValidationOk){
          $("html, body").animate({ scrollTop: 0 }, "slow");
        }
        return isValidationOk;
    });

    $('.delete-item-button').click(function() {
        var result = confirm("Are you sure?");
        return result;
    });
```

JQuery validate form



Validation - question



Is client-side validation sufficient for security (for example Javascript validation)?

Do we need server-side validation ?

Validation - question

The answer is “NO” !

Client side validation can be easily bypassed!

Malicious input can be submitted!

Questions

Bibliography

Images:

- https://publicdomainvectors.org/pl/wektorow-swobodnych/Grafika-wektorowa-budynku-szko%C5%82y/11750.html
- https://careerfoundry.com/en/blog/ux-design/what-is-a-wireframe-guide/
- https://openclipart.org/detail/314949/librarian-2
- https://freesvg.org/books
- https://images.rawpixel.com/image_800/czNmcy1wcml2YXRlL3Jhd3BpeGVsX2ltYWdlcy93ZWJzaXRlX2NvbnRlbnQvbHIvam9iNjczLTE1OS14LmpwZw.jpg?s=l3MJkiQKcO5Q1lqiFPEpZn-HyL-CZmKAjsEnk9XR1bA
- https://openclipart.org/detail/298193/librarian
- https://freesvg.org/female-computer-user-vector-icon
- https://www.rawpixel.com/image/7419720/vector-book-public-domain-illustrations

Bibliography (2)

- https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/
- https://www.techslang.com/definition/what-is-dockerization/
- https://medium.com/analytics-vidhya/dajngo-with-nginx-gunicorn-aaf8431dc9e0
- https://docs.docker.com/storage/volumes/
- https://www.google.com/url?sa=i&url=https%3A%2F%2Fopenclipart.org%2Fdetail%2F295783%2Fbusiness-man-working&psig=AOvVaw0GJqqPQuFHDBkfmOWh8_Up&ust=1701768583938000&source=images&cd=vfe&opi=89978449&ved=0CBMQjhxqFwoTCMjH-pC89YIDFQAAAAAdAAAAABAE
- https://commons.wikimedia.org/wiki/File:Tower_torre_pc_clon_server.svg
- https://commons.wikimedia.org/wiki/File:Cloud_server.svg
- https://hub.docker.com/_/docker