



Fundamental for Software Development



**[https://github.com/up1/
course-full-stack-development](https://github.com/up1/course-full-stack-development)**



Technical Excellence





SPECIFICATION BY EXAMPLE



CONTINUOUS
INTEGRATION



CONTINUOUS DELIVERY



TEST AUTOMATION



THINKING ABOUT TESTING



TECHNICAL
EXCELLENCE



TEST-DRIVEN DEVELOPMENT



ARCHITECTURE
& DESIGN



ACCEPTANCE
TESTING



CLEAN CODE



UNIT TESTING

<https://less.works/less/technical-excellence/index>



Delivery Process



Delivery process

UI Flow

API design

Database design

Pipeline



Version Control



Git



git --distributed-even-if-your-workflow-isnt

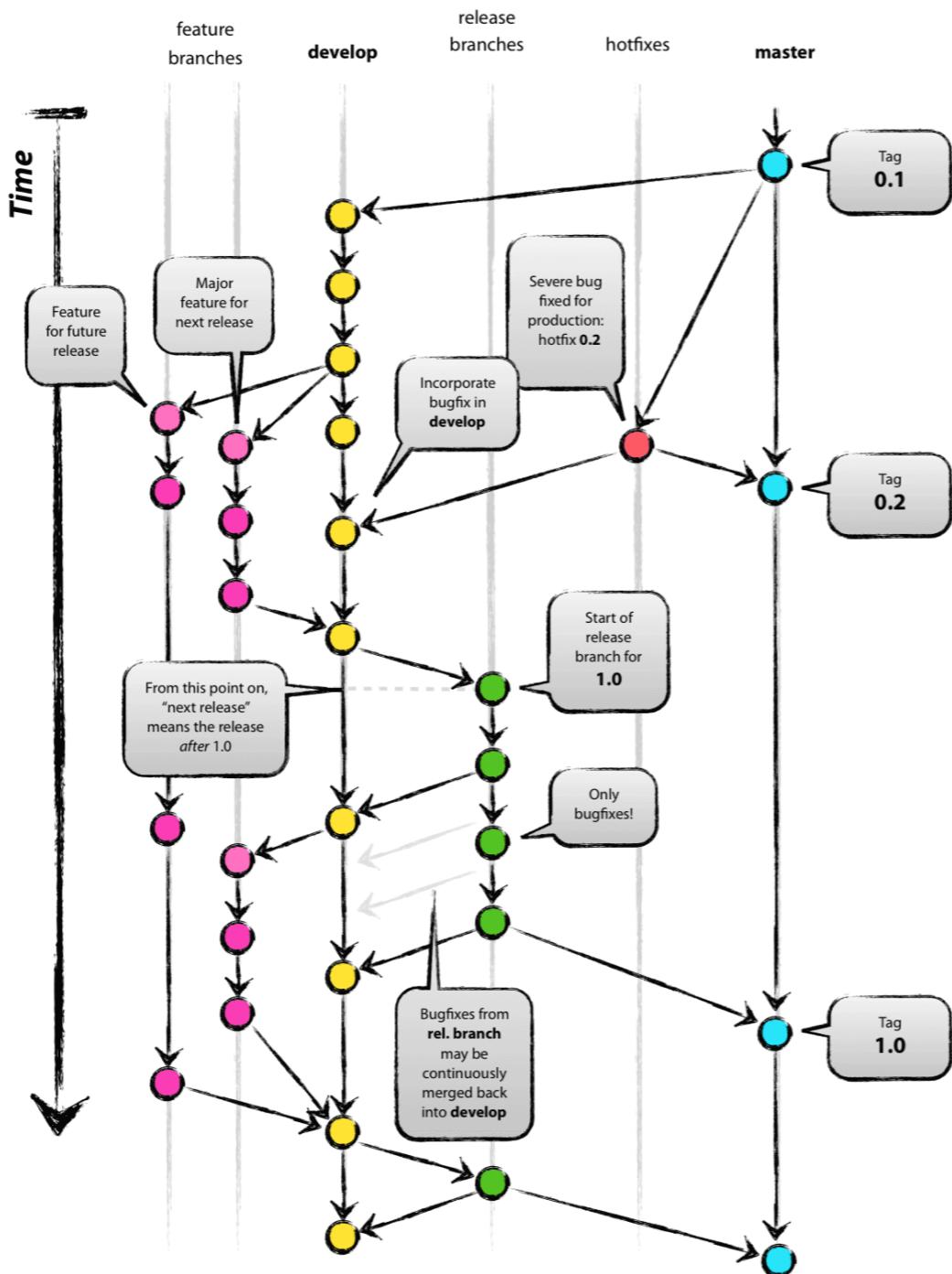
Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, **convenient staging areas**, and **multiple workflows**.

<https://git-scm.com/>



Branch Strategies



<https://nvie.com/posts/a-successful-git-branching-model/>



VS Code

The screenshot shows the official Visual Studio Code website. At the top, there's a dark navigation bar with the Visual Studio Code logo, a search bar labeled "Search Docs", and a "Download" button. Below the header, a blue banner announces "VS Code Day 2024" and encourages users to "Register now for a full day of community, learning, and all things Visual Studio Code". To the right of the banner is a date and time indicator: "April 24, 2024" and "12:00 AM - 07:00 AM UTC +7 hours". The main content area features a large white text "Code editing. Redefined." over a dark background. Below it, the text "Free. Built on open source. Runs everywhere." is displayed. A prominent blue button says "Download Mac Universal Stable Build". To the right of this button is a "▼" icon. Below the download button, a link says "Web, Insiders edition, or other platforms". A small note below states: "By using VS Code, you agree to its license and privacy statement." On the right side of the page, a screenshot of the VS Code interface is shown. It displays a code editor with several tabs: "blog-post.js", "index.js", and "utils.js". The "index.js" tab is active, showing code related to "gatsby-graphql-app". The code includes imports for React and Image, and a function that returns a JSX object. To the left of the code editor is the "Extensions: Marketplace" sidebar, which lists popular extensions like Python, GitLens, C/C++, ESLint, Debugger for Chrome, Language Support, vscode-icons, and Vetur. The bottom of the screenshot shows the VS Code status bar with information like "master", "Gatsby Develop (gatsby-graphql-app)", and terminal output.

<https://code.visualstudio.com/>



TypeScript



TypeScript



Download Docs Handbook Community Tools

Search Docs

TypeScript is JavaScript with syntax for types.

TypeScript is a strongly typed programming language that builds on JavaScript, giving you better tooling at any scale.

Try TypeScript Now

Online or via npm

...

[Editor Checks](#) [Auto-complete](#) [Interfaces](#) [JSX](#)

```
const user = {  
    firstName: "Angela",  
    lastName: "Davis",  
    role: "Professor",  
}  
  
console.log(user.name)
```

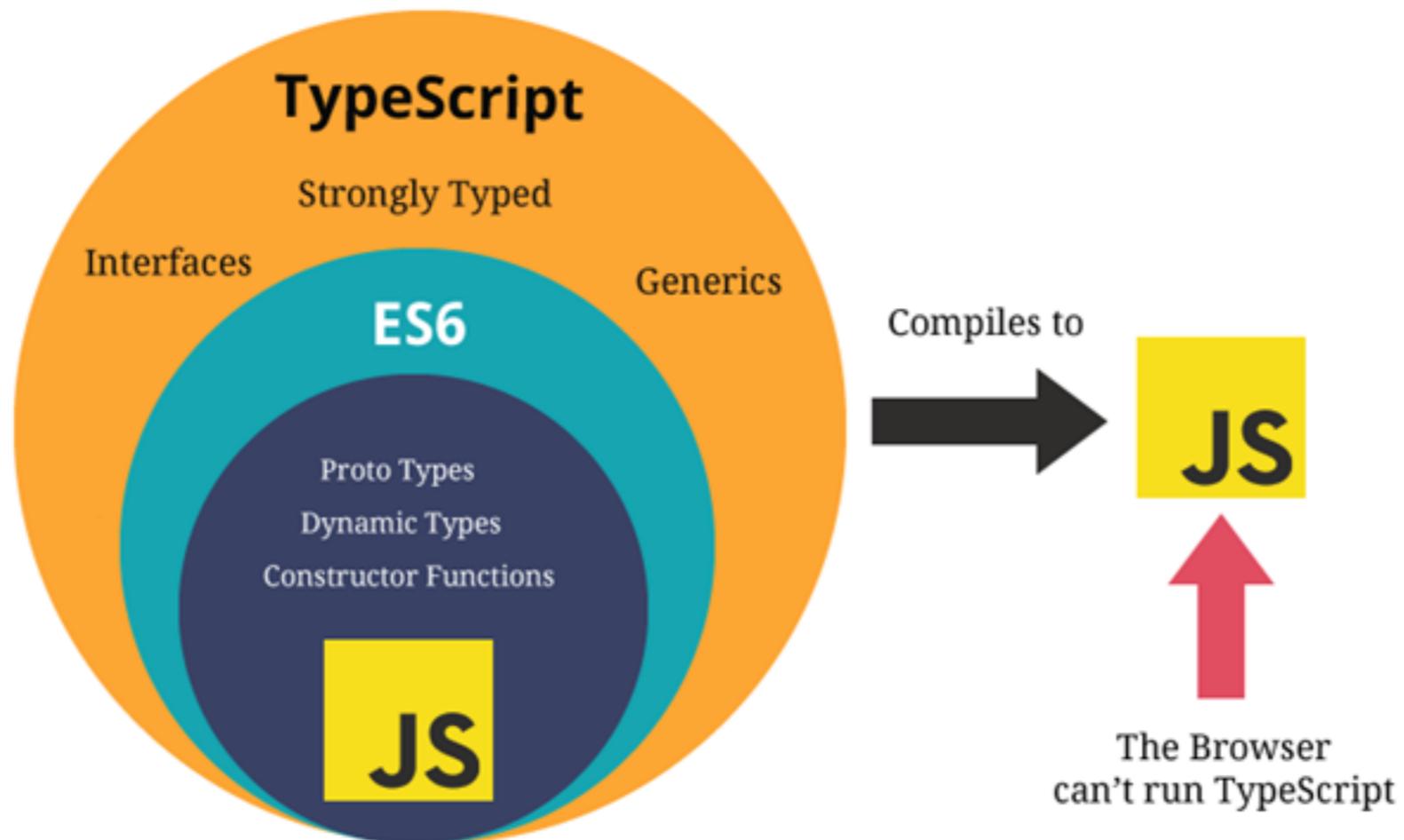
Property 'name' does not exist on type '{
 firstName: string; lastName: string; role:
 string; }'.

<https://www.typescriptlang.org/>



TypeScript

A superset of JavaScript



<https://www.typescriptlang.org/>



TypeScript Essentials

Basic

Type in TypeScript

Advance

Design patterns



Key idea of TypeScript

It adds **static-typing** to JavaScript

More maintainable

Easy to learn and use



Setup project with TypeScript



Software Requirements

NodeJS 18+

The screenshot shows the official Node.js website. The main heading "Run JavaScript Everywhere" is prominently displayed. Below it, a paragraph describes Node.js as a free, open-source, cross-platform JavaScript runtime environment. A green button labeled "Download Node.js (LTS)" is visible. To the right, a code editor window displays a simple HTTP server example. The code uses ES6 imports and the `node:http` module to create a server that responds with "Hello World!". The code editor includes tabs for "Create an HTTP Server", "Write Tests", and "Read and Hash a File". A "JavaScript" label and a "Copy to clipboard" button are also present.

```
1 // server.mjs
2 import { createServer } from 'node:http';
3
4 const server = createServer((req, res) =>
5   res.writeHead(200, { 'Content-Type': 'text/plain' });
6   res.end('Hello World!\n');
7 );
8
9 // starts a simple http server locally on
10 server.listen(3000, '127.0.0.1', () => {
11   console.log('Listening on 127.0.0.1:3000');
12 });
13
14 // run with `node server.mjs`
```

<https://nodejs.org/en>



Software Requirements

Node Version Manager



<https://github.com/nvm-sh/nvm>

<https://github.com/coreybutler/nvm-windows>



Create project (1)

\$npm install -g typescript

\$tsc --init



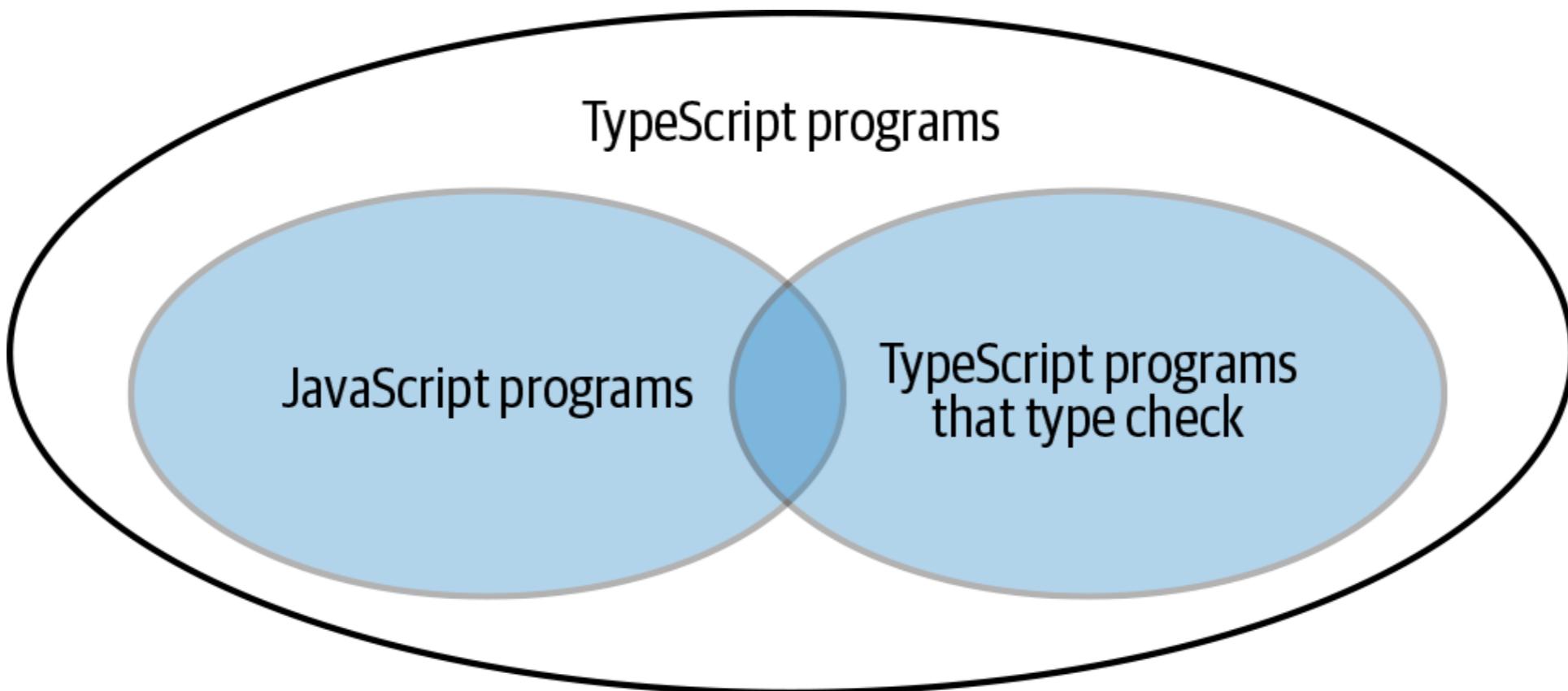
Create project (2)

\$npm init

\$npm install typescript -S



Type Checking

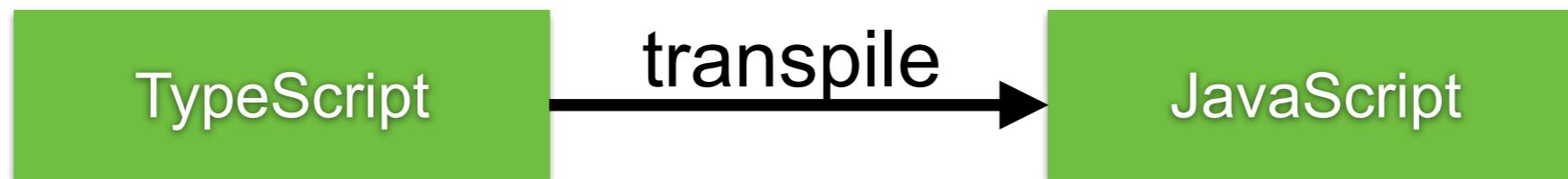


Hello TS

\$tcs hello.ts

\$node hello.js

```
let message: string = "Hello, World!";
console.log(message);
```



Variable Declarations

var , let, const

	var	let	const
Stored in Global Scope	✓	✗	✗
Function Scope	✓	✓	✓
Block Scope	✗	✓	✓
Can Be Reassigned?	✓	✓	✗
Can Be Redeclared?	✓	✗	✗
Can Be Hoisted?	✓	✗	✗



Function declaration

```
// Simple function
function add(a: number, b: number): number {
    return a + b;
}

// Arrow function
const add2 = (a: number, b: number): number => a + b;
```



Data Types

```
let isDone: boolean = false;
let decimal: number = 6;
let color: string = "blue";
let myTuple: [string, number] = ["hello", 10];
let myArray: number[] = [1, 2, 3];
let myObject: { name: string, age: number }
  = { name: "John", age: 30 };

let myAny: any = 4;
let nothing: undefined = undefined;
let myNull: null = null;
```



Custom Object Types

```
// Type declaration
type Person = {name: string, age?: number, email?: string};

// Variable declaration
let p1: Person = {name: 'demo', };
let p2: Person = {name: 'demo', age: 23};
let p3: Person = {name: 'demo', age: 23, email: ''};

p1.name = "new name";
p1.age = 30;
```



Object-Oriented Programming

Class Interface



Class

Blueprint for creating objects

```
class Person {  
    name: string;  
    age: number;  
  
    constructor(name: string, age: number) {  
        this.name = name;  
        this.age = age;  
    }  
  
    greet() {  
        console.log(`Hello, my name is ${this.name} and I'm ${this.age} years old.`);  
    }  
}  
  
let john = new Person("John", 30);  
john.greet();
```



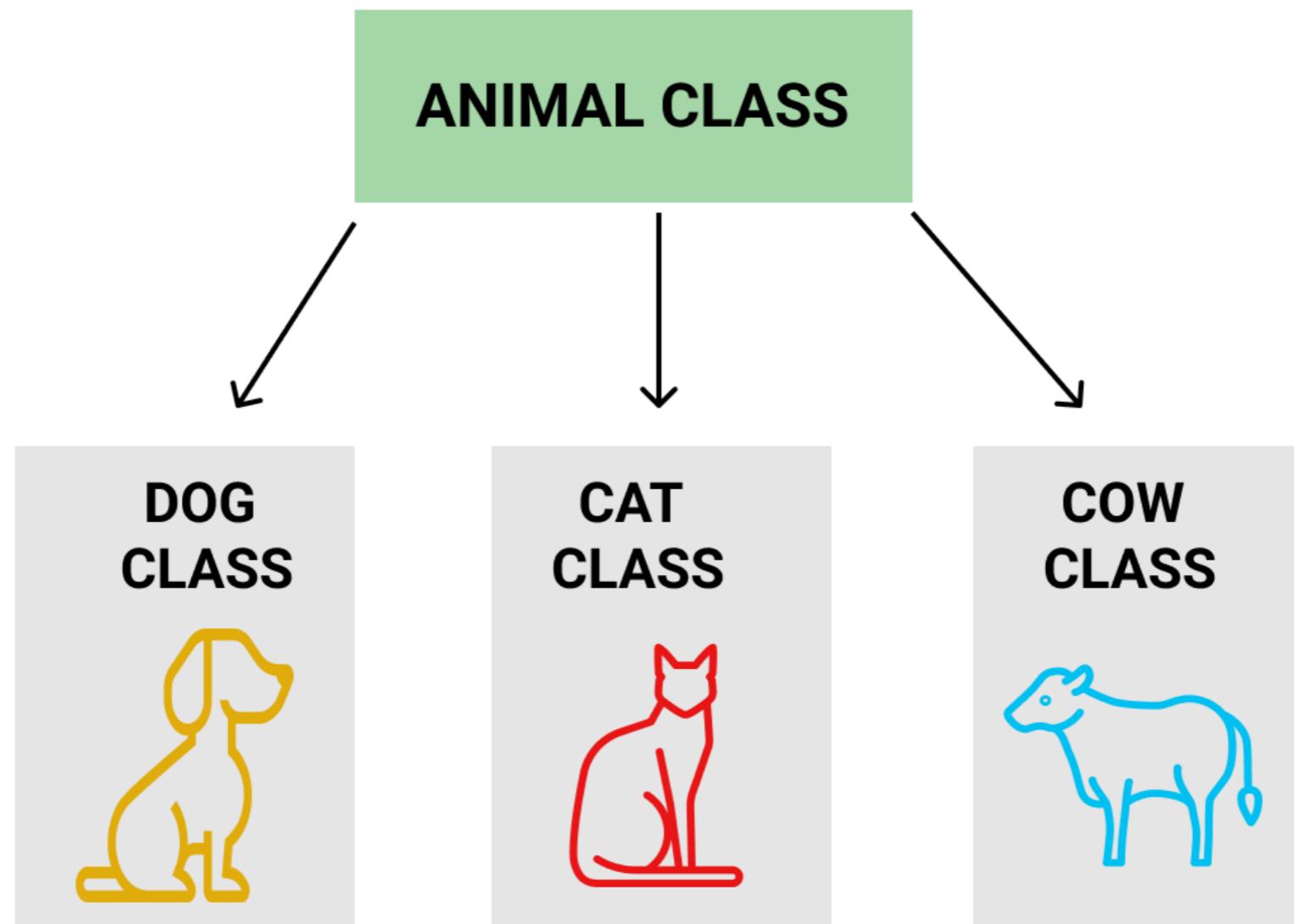
Class

Shorthand constructor

```
class Person {  
    constructor(public name: string, public age: number) {}  
    greet() {  
        console.log(`Hello, my name is ${this.name} and I'm ${this.age} years  
old.`);  
    }  
}  
  
let john = new Person("John", 30);  
john.greet();
```



Inheritance



Visibility of class member

Public (default)

Protected

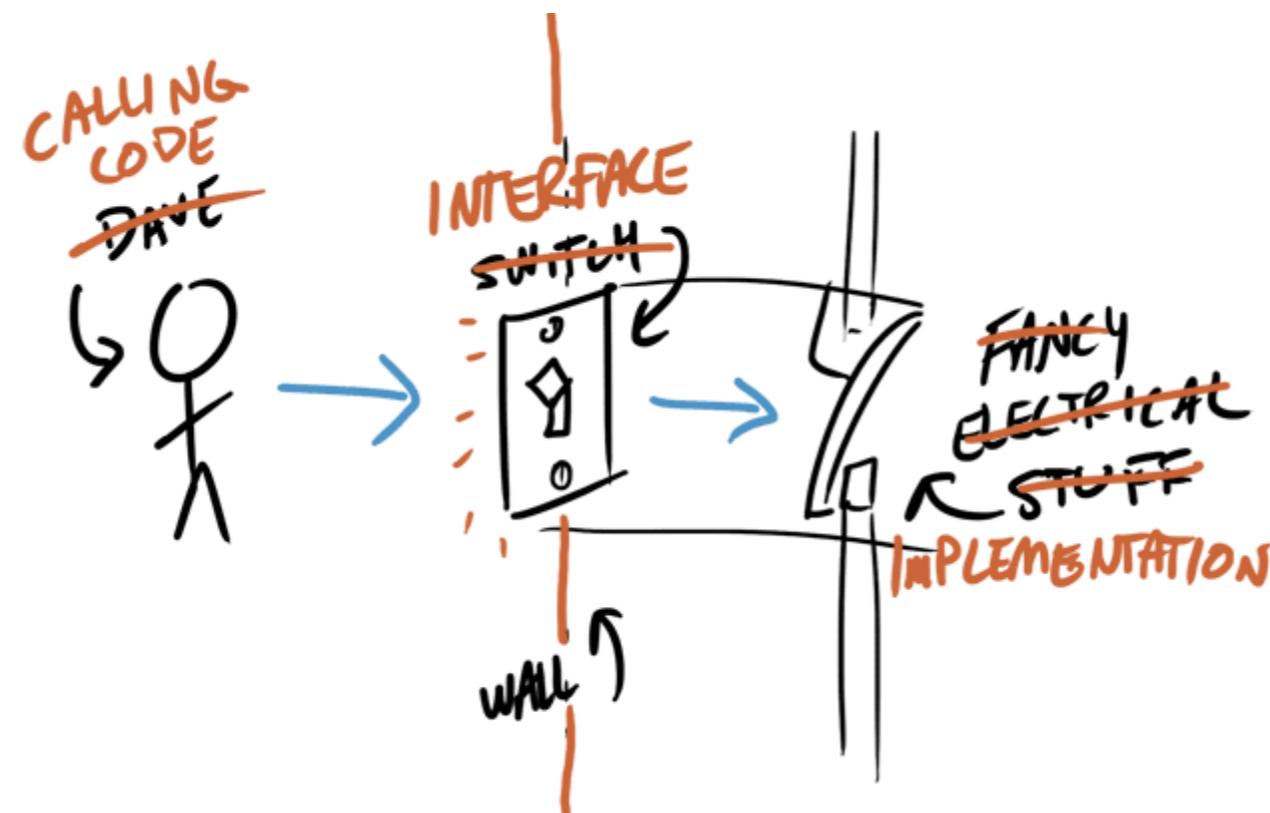
Private

Readonly



Interface (Encapsulation)

Contract for a class or function
Can't create an object/instance
Used for type checking



Interface

```
interface IPerson {  
    name: string;  
    age: number;  
    greet(): void;  
}
```

```
class Person implements IPerson {  
    name: string;  
    age: number;  
  
    constructor(name: string, age: number) {  
        this.name = name;  
        this.age = age;  
    }  
  
    greet() {  
        ...  
    }  
}
```



Generic

Use in function and class
Reusable and flexible !!

```
class QueueV1 {  
    private items: string[];  
  
    constructor() {  
        this.items = [];  
    }  
  
    push = (newItem: string) => this.items.push(newItem);  
    pop = () => this.items.shift();  
}
```



Generic

Improve code with Generic way

```
class QueueWithGeneric<T> {  
    private items: T[];  
  
    constructor() {  
        this.items = [];  
    }  
  
    push = (newItem: T) => this.items.push(newItem);  
    pop = () => this.items.shift();  
}
```



Mapped types

Pick

Partial

Readonly

<https://www.typescriptlang.org/docs/handbook/2/mapped-types.html>



Pick some properties

```
type User = {  
    id: string,  
    name: string,  
    age: number  
};  
  
type NewUserData = Pick<User, 'name'>;  
const user1: NewUserData = { name: 'John' };  
  
const user2: User = {  
    ...user1,  
    id: '1',  
    age: 20  
};
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax



Partial

```
type User = {  
    id: string,  
    name: string,  
    age: number  
};  
  
type PartialUser = Partial<User>;  
const user1: PartialUser = {  
    name: 'John',  
};
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax



ReadOnly

```
type ReadonlyUser = Readonly<User>;  
  
const user2: ReadonlyUser = {  
  id: '123',  
  name: 'John',  
  age: 30  
};  
  
user2.age = 31; // Error: Cannot assign to 'age'  
// because it is a read-only property.
```



More configuration ...



Configurations in project

tsconfig.json

Linting

Prettier

Husky



tsconfig.json file

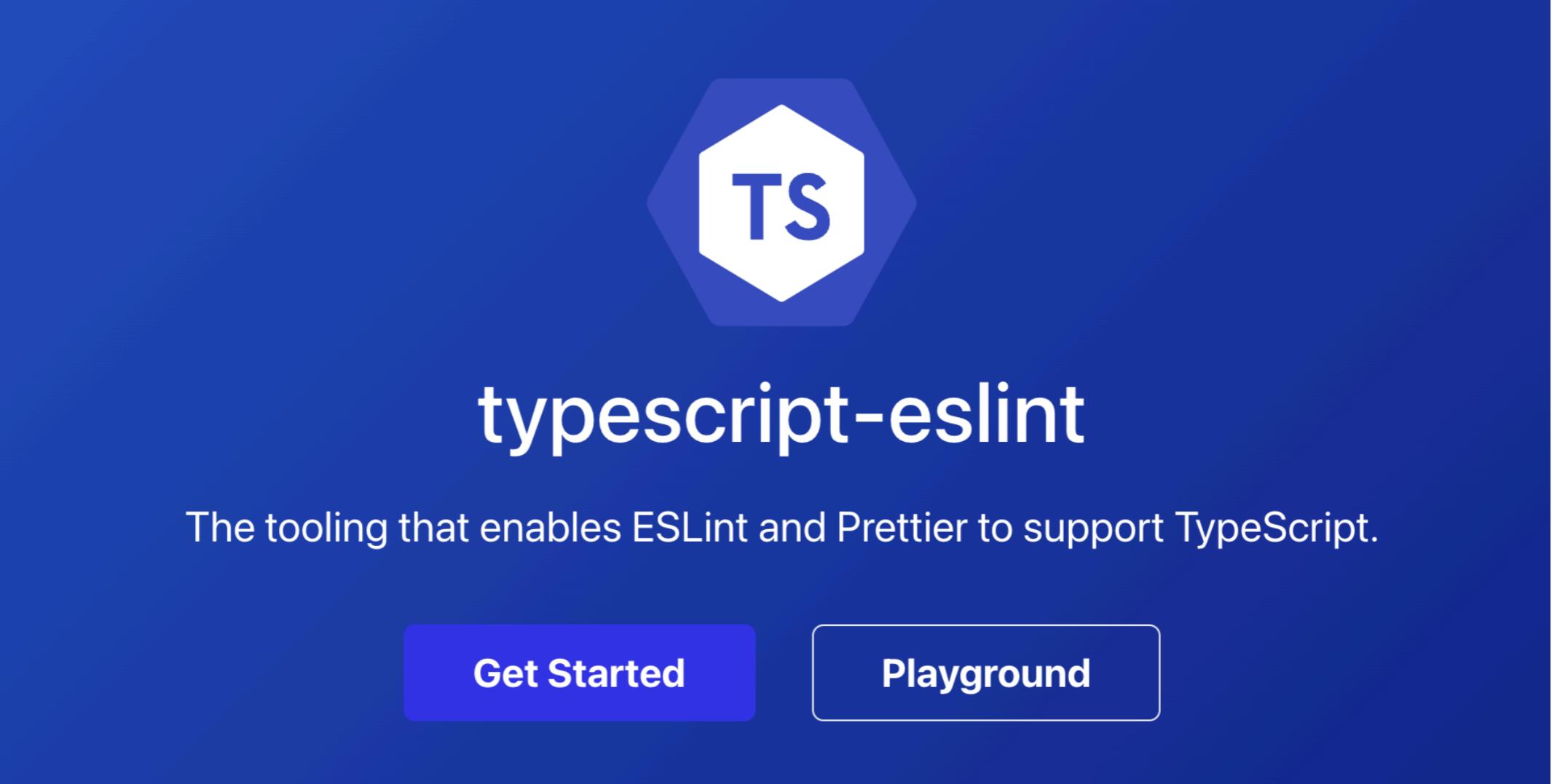
TypeScript configuration file
Compiler settings

```
{  
  "include": ["src", "tests"],  
  "exclude": ["node_modules"],  
  
  "compilerOptions": {  
    "outDir": "build",  
    "target": "es6"  
  }  
}
```



TSLint and ESLint

Enforce coding style

The image shows the homepage of the typescript-eslint project. It features a large blue hexagonal logo with the letters 'TS' in white. Below the logo, the word 'typescript-eslint' is written in a large, white, sans-serif font. Underneath that, a subtitle reads 'The tooling that enables ESLint and Prettier to support TypeScript.' At the bottom, there are two buttons: a solid blue button on the left labeled 'Get Started' and a white button with a black border on the right labeled 'Playground'.

The tooling that enables ESLint and Prettier to support TypeScript.

Get Started Playground

<https://typescript-eslint.io/>



eslint.config.js file

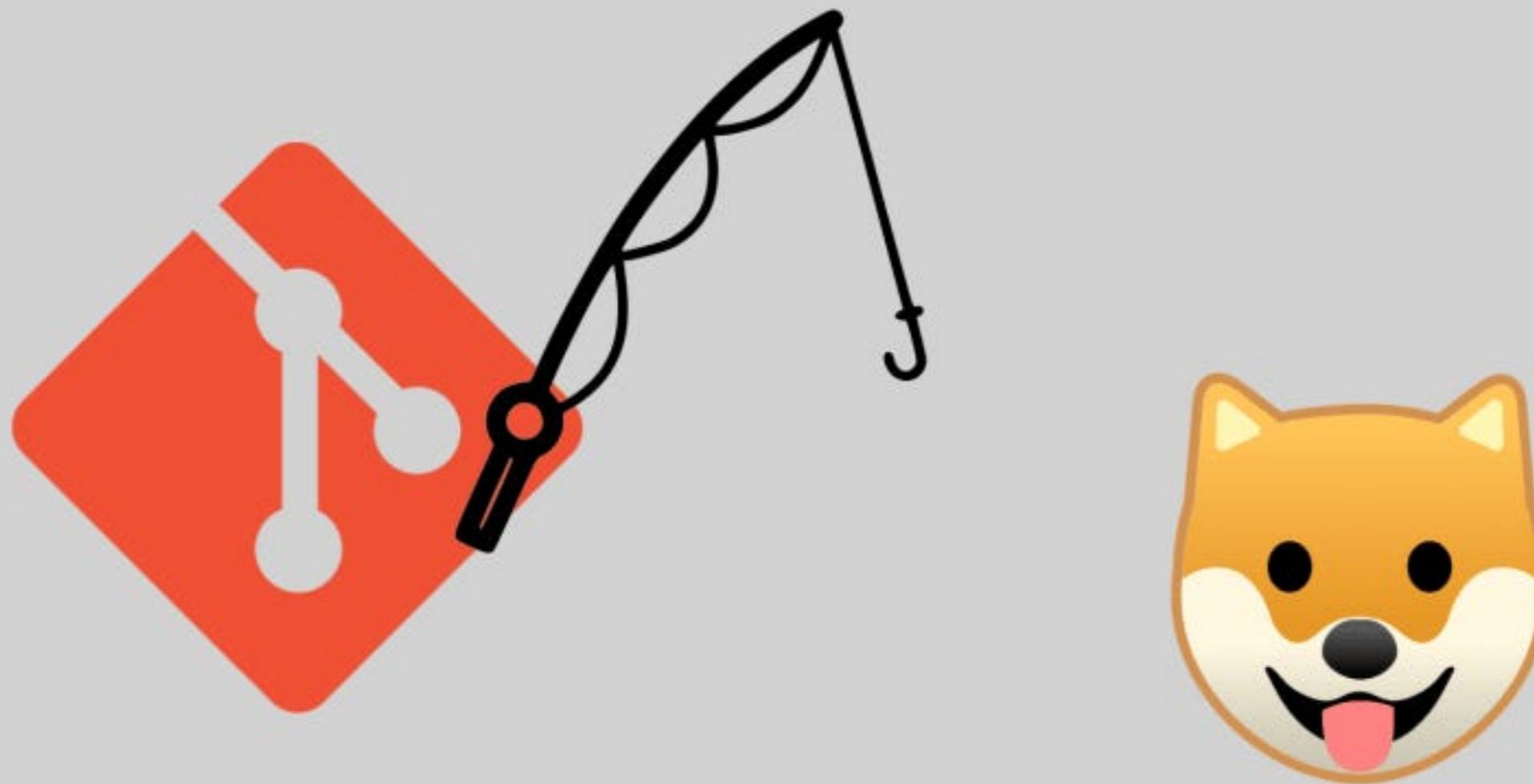
```
// @ts-check
import eslint from '@eslint/js';
import tseslint from 'typescript-eslint';

export default tseslint.config(
  eslint.configs.recommended,
  ...tseslint.configs.recommended,
);
```

\$npx eslint .



GitHook with Husky



<https://typicode.github.io/husky/>



Testing



Learning from Test



Jest



Jest



The graphic features a central card labeled "JEST" with a green crown icon. To its left are two smaller cards, both labeled "PASS" with green checkmark icons. Below the central card are four red buttons: "GET STARTED", "DOCS", "CONFIG", and "GET HELP". A large grey box contains the following text:

Jest is a delightful JavaScript Testing Framework with a focus on simplicity.

It works with projects using: [Babel](#), [TypeScript](#), [Node](#), [React](#), [Angular](#), [Vue](#) and more!

<https://jestjs.io/>



Install Jest in project

```
npm install -D jest @types/jest ts-jest ts-node  
npm test
```

```
"scripts": {  
  ...  
  "test": "jest"  
}
```

<https://jestjs.io/>



Jest.config.ts

jest --init

```
import type {Config} from '@jest/types';
// Sync object
const config: Config.InitialOptions = {
  moduleDirectories: ['node_modules', 'src'],
  verbose: true,
  transform: {
    '^.+\\.tsx?$': 'ts-jest',
  }
};
export default config;
```

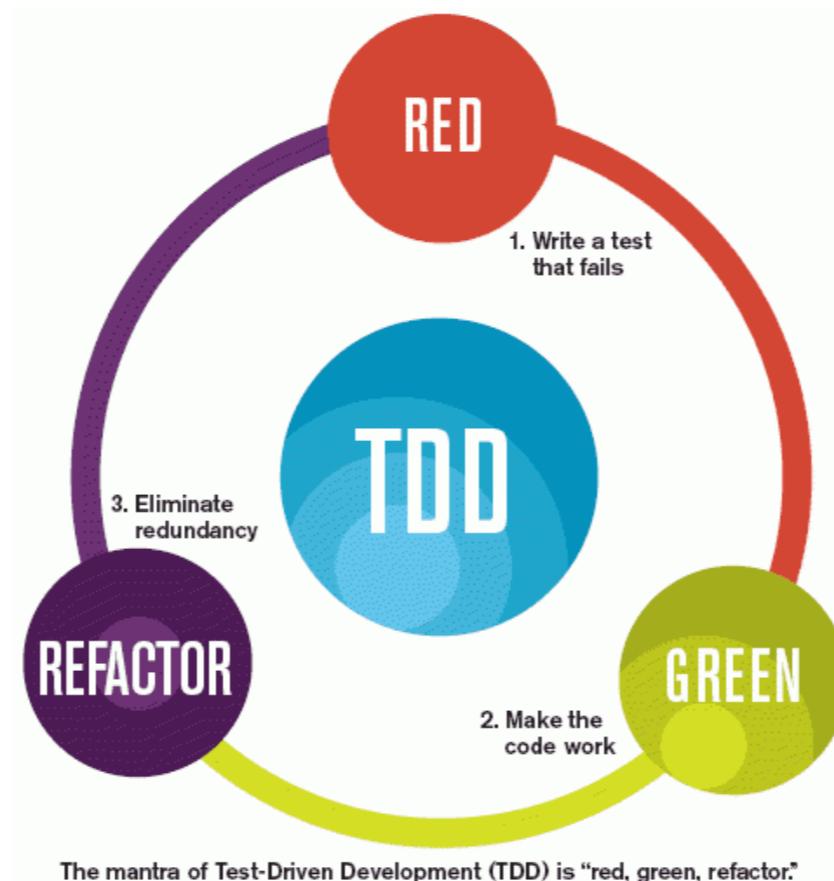
<https://jestjs.io/docs/configuration>



Learn from Testing

Test
code

Production
code



Hello with Test

hello.test.ts

```
import { sayHi } from "../../src/hello";

test('Hello world with sayHi', () => {
  expect(sayHi()).toBe("Hello world!");
});
```

hello.ts

```
export const sayHi = () => {
  return "Hello world!";
}
```



Remove ../../ from code !!

hello.test.ts

```
import { sayHi } from "../../src/hello";

test('Hello world with sayHi', () => {
  expect(sayHi()).toBe("Hello world!");
});
```

jestconfig.json

```
{
  "compilerOptions": {
    "baseUrl": "./src",
    "paths": {
      "*": ["*"]
    }
  },
}
```

```
import { sayHi } from "hello";
test('Hello world with sayHi', () => {
  expect(sayHi()).toBe("Hello world!");
});
```



Test Coverage



Jest.config.ts

Config test coverage report

```
const config: Config.InitialOptions = {  
  moduleDirectories: ['node_modules', 'src'],  
  verbose: true,  
  transform: {  
    '^.+\\.tsx?$': 'ts-jest',  
  },  
  collectCoverage: true,  
  coverageReporters: ['json', 'lcov', 'text', 'clover'],  
};
```

<https://jestjs.io/docs/configuration>



Run test with coverage

npm test

```
PASS  tests/unit/hello.test.ts
  ✓ Hello world with sayHi (1 ms)

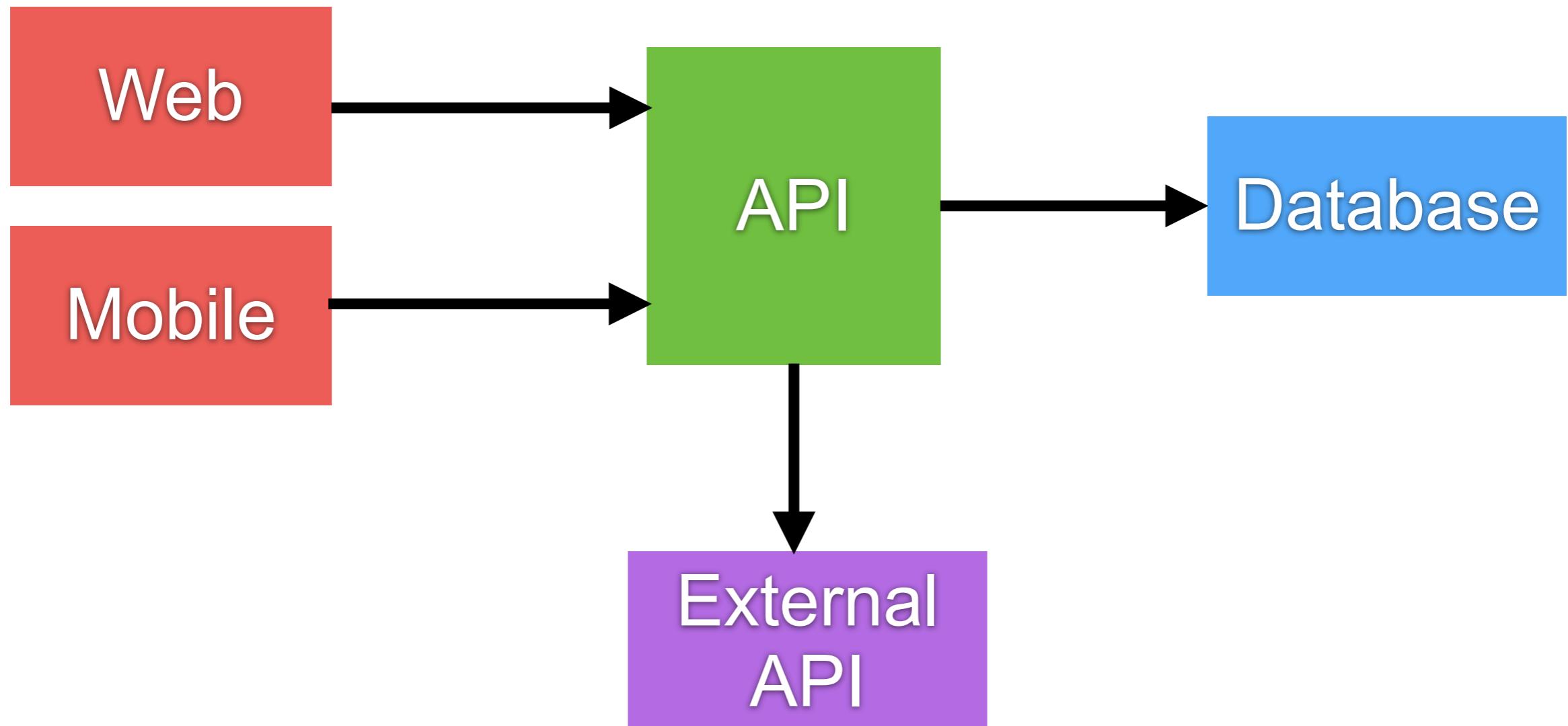
-----|-----|-----|-----|-----|-----|
File    | %Stmts | %Branch | %Funcs | %Lines | Uncovered Line #s
-----|-----|-----|-----|-----|-----|
All files | 100   | 100   | 100   | 100   |
hello.ts  | 100   | 100   | 100   | 100   |
-----|-----|-----|-----|-----|-----|
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.368 s, estimated 1 s
Ran all test suites.
```



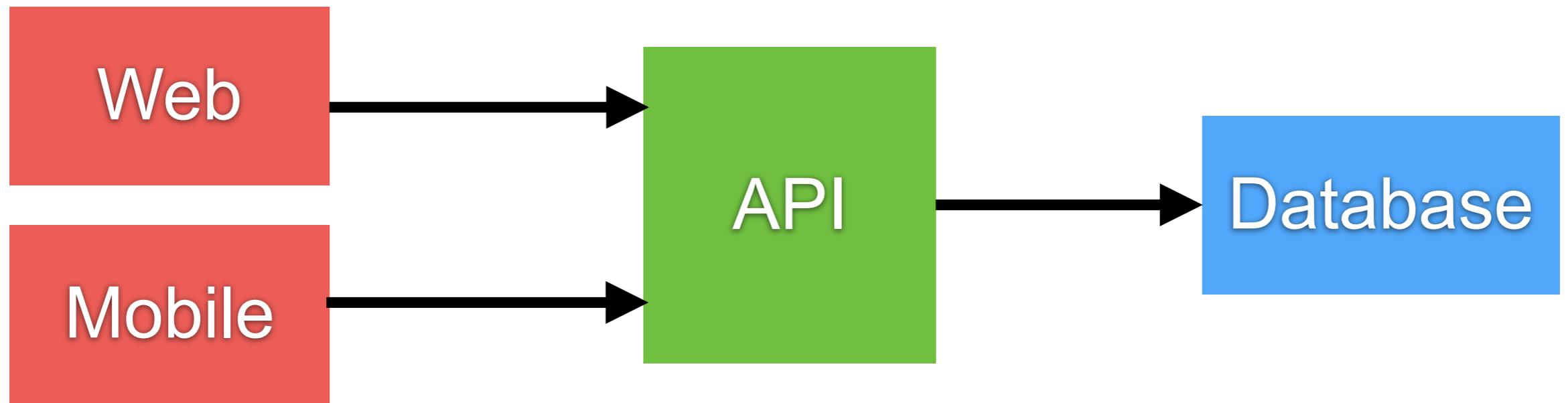
Test Strategies



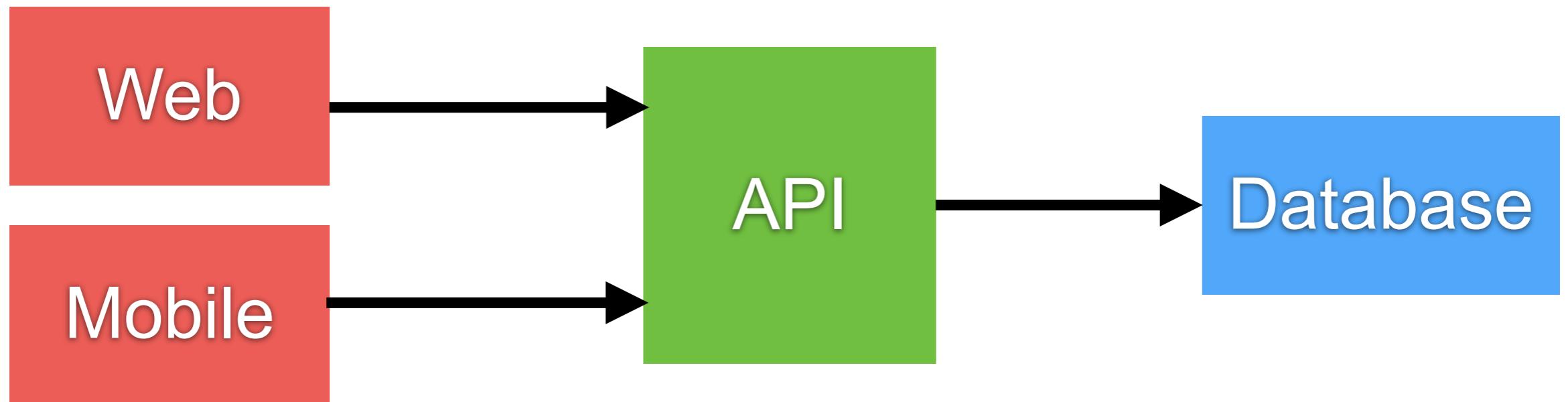
Architecture



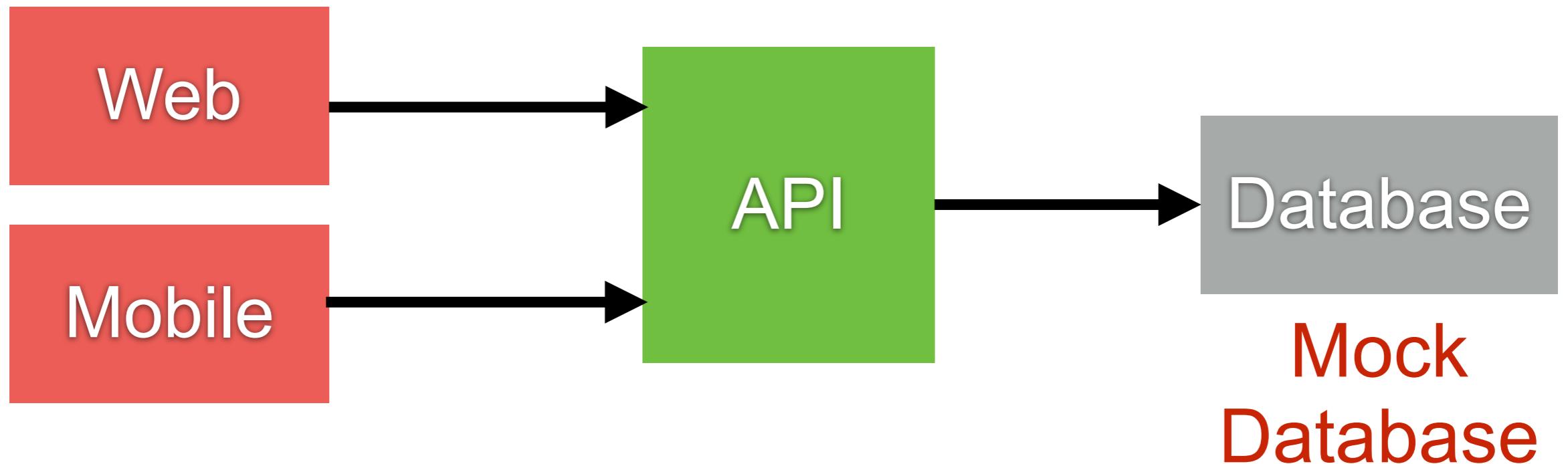
Test ?



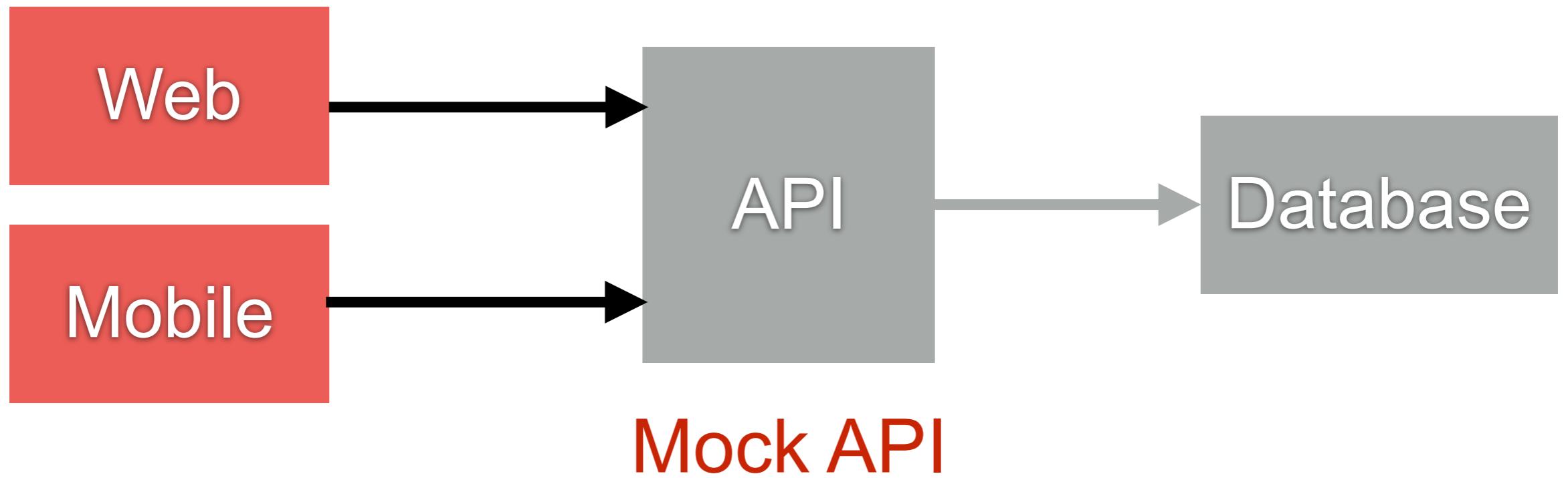
UI Test ?



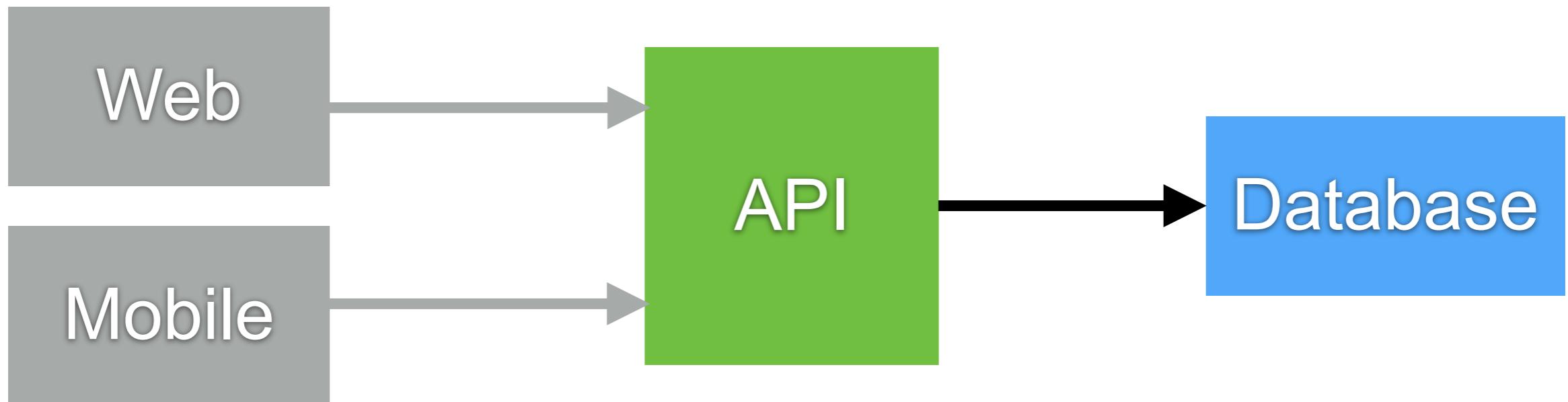
UI Test ?



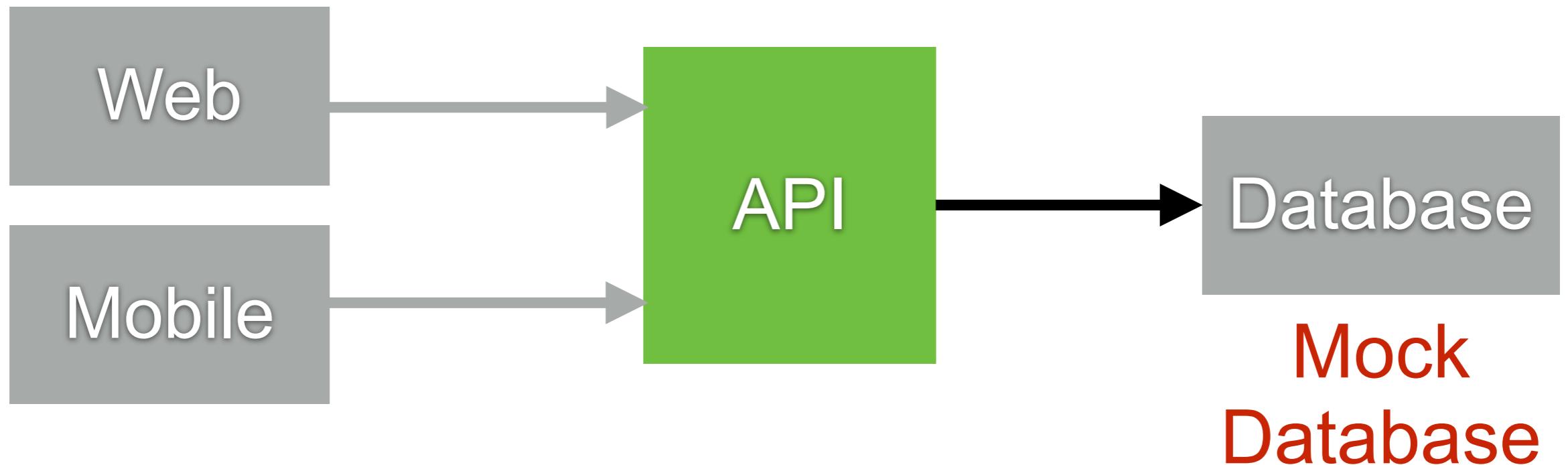
UI Test ?



API Testing ?



API Testing ?



Backend

Develop REST API



Develop REST API

Design

Develop

Test

Deploy

Mock
server

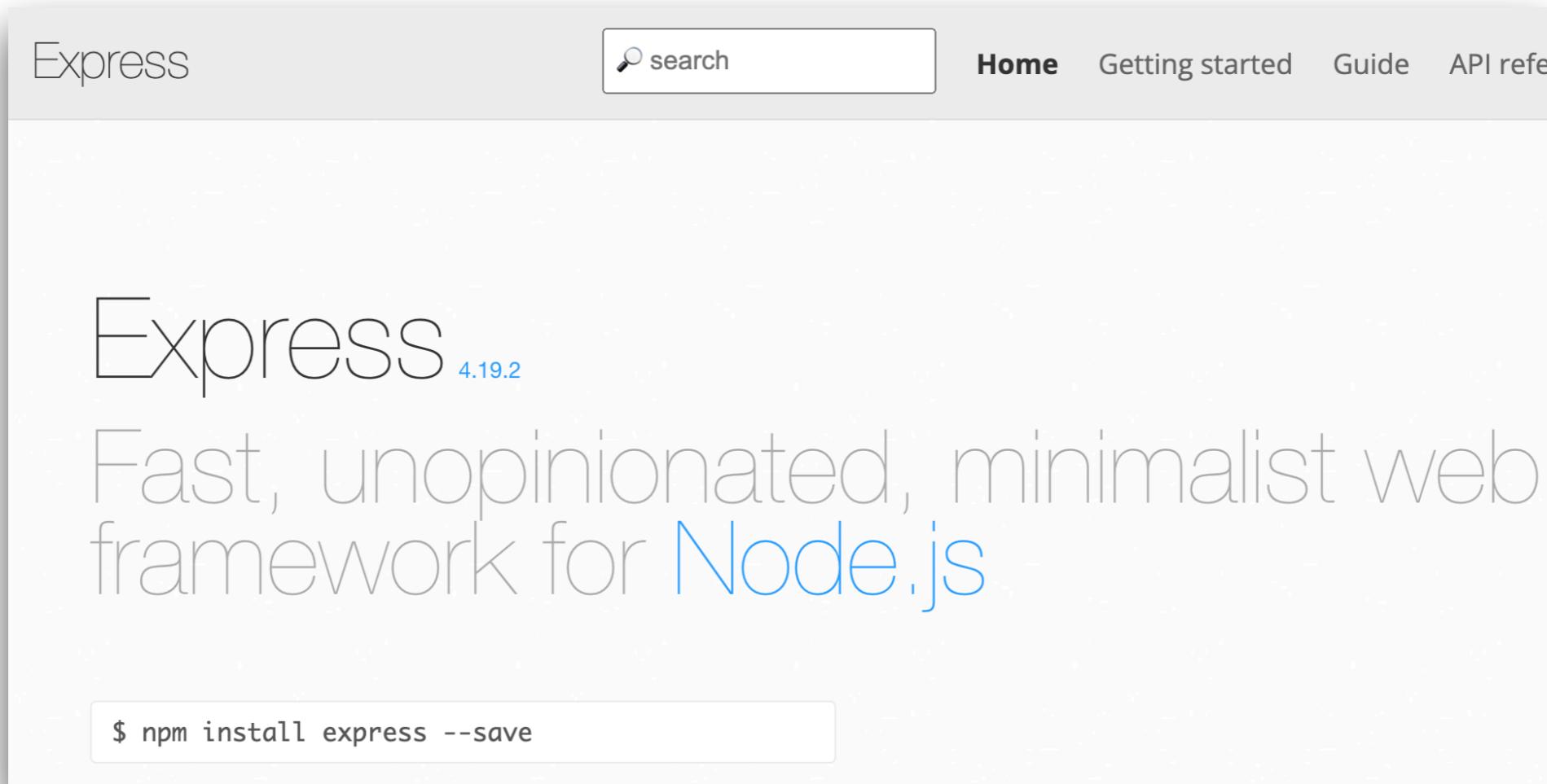


How to design REST API ?



Develop REST APIs

Use expressjs library



<https://expressjs.com/>



Create project

```
npm init -y  
npx tsc --init  
npm install -S express  
npm install -D @types/express ts-node nodemon
```

tsconfig.json

```
{  
  "include": ["src"],  
  "compilerOptions": {  
    "outDir": "./build",
```

<https://expressjs.com/>



Hello API

<http://localhost:3000>

server.ts

```
import express from "express";

const app = express();
const port = 3000;

app.get("/", (req, res) => {
  res.json({
    message: "Hello World from TypeScript!",
  });
});

app.listen(port, () => {
  console.log(`Server started at http://localhost:${port}`);
});
```

<https://expressjs.com/>



How to run project

`npx ts-node src/server.ts`

<https://github.com/TypeStrong/ts-node>



How to run project

Edit file **package.json**
Working with nodemon

```
{  
  
  "scripts": {  
    "dev": "npx tsc && node build/server.js",  
    "serve": "nodemon --watch 'src/**/*.{ts,js}' --exec ts-node src/server.ts",  
  },  
}
```

<https://www.npmjs.com/package/nodemon>



Dev Mode

npm run dev

```
> ts03-rest@1.0.0 dev
> tsc && nodemon build/server.js

[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node build/server.js`
Server started at http://localhost:3000
```



Restart when changed

npm run serve

```
> ts03-rest@1.0.0 serve
> nodemon --watch 'src/server.ts' --exec ts-node src/server.ts

[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): src/server.ts
[nodemon] watching extensions: ts,json
[nodemon] starting `ts-node src/server.ts`
Server started at http://localhost:3000
```



How to test your API ?



How to test your API ?

External Testing

Internal Testing



How to test your API ?

External Testing

Internal Testing



POSTMAN

Coding with
Jest and SuperTest



Working with Jest and SuperTest

```
npm install -D jest @types/jest ts-jest ts-node
```

```
npm install -D supertest @types/supertest
```

<https://www.npmjs.com/package/supertest>



Testable application

server.ts

```
import app from "./app";
const port = 3000;
app.listen(port, () => {
  console.log(`Server started at http://localhost:${port}`);
});
```

app.ts

```
import express from "express";
const app = express();

app.get("/", (req, res) => {
  res.json({
    message: "Hello World from TypeScript",
  });
});

export default app;
```



Hello API Test

hello.api.ts

```
// Test with supertest
import request from 'supertest';
import app from '../src/app';

test('GET /', async () => {
  const response = await request(app).get('/');
  expect(response.status).toBe(200);
  expect(response.body).toEqual({
    message: 'Hello World from TypeScript'
  });
});
```

<https://www.npmjs.com/package/supertest>



Config jest.config.ts

npm test

```
import type {Config} from '@jest/types';
// Sync object
const config: Config.InitialOptions = {
  verbose: true,
  roots: ['<rootDir>/tests'],
  transform: {
    '^.+\\.tsx?$': 'ts-jest',
  },
  collectCoverage: true
};
export default config;
```

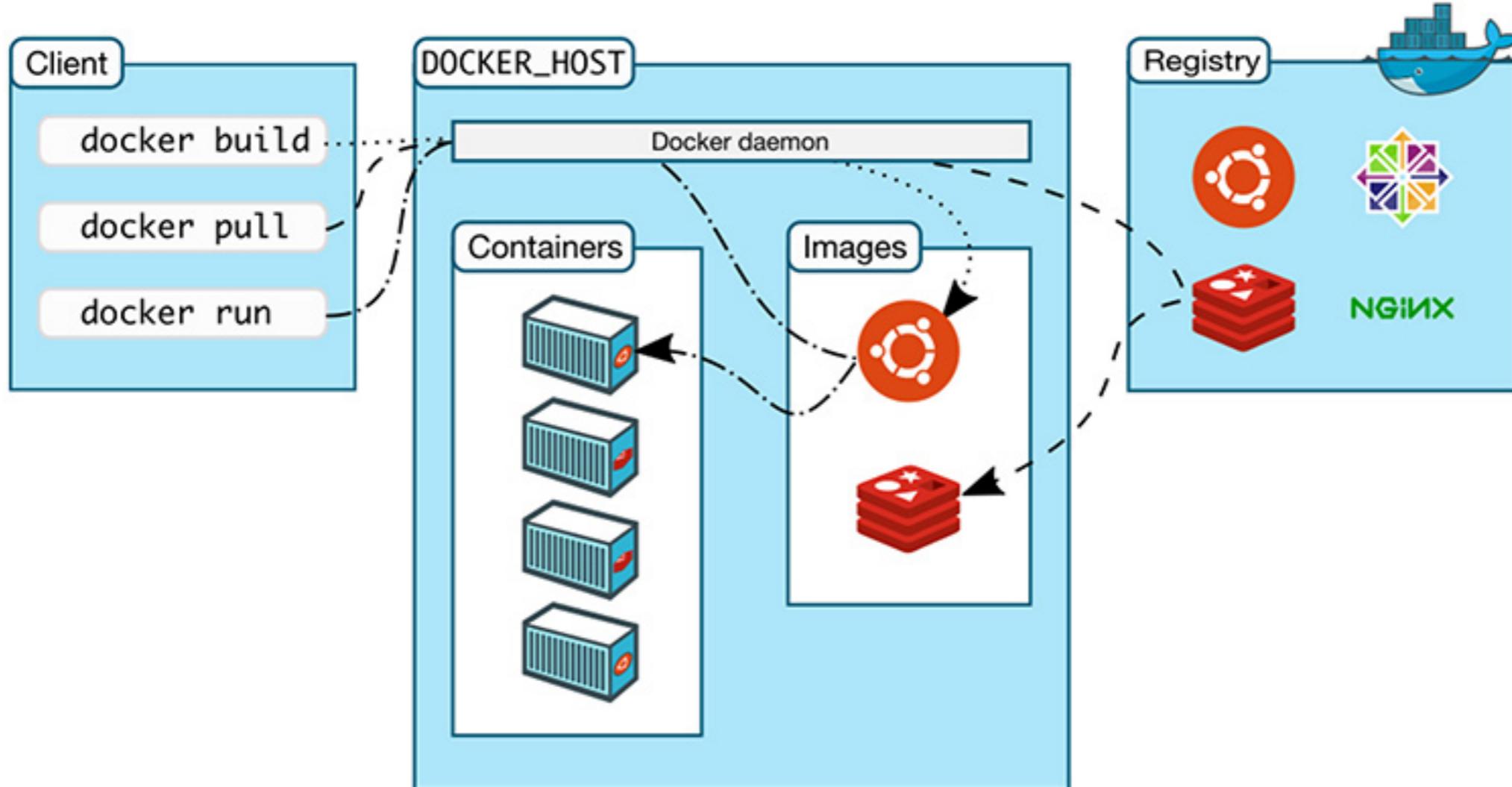
<https://www.npmjs.com/package/supertest>



Working with Docker



Docker



<https://www.docker.com/>



Build Image for Project

Dockerfile

```
FROM node:18-alpine3.18 as build
WORKDIR /app
COPY package*.json .
RUN npm install
COPY . .
RUN npx tsc

FROM node:18-alpine3.18
WORKDIR /app
COPY --from=build /app/build ./build
COPY package*.json ./
RUN npm install
EXPOSE 3000
CMD ["node", "build/server.js"]
```

https://hub.docker.com/_/node/



Working with Docker compose

docker-compose.yml

```
services:  
  app:  
    build: .  
    ports:  
      - "3000:3000"
```



Build and run

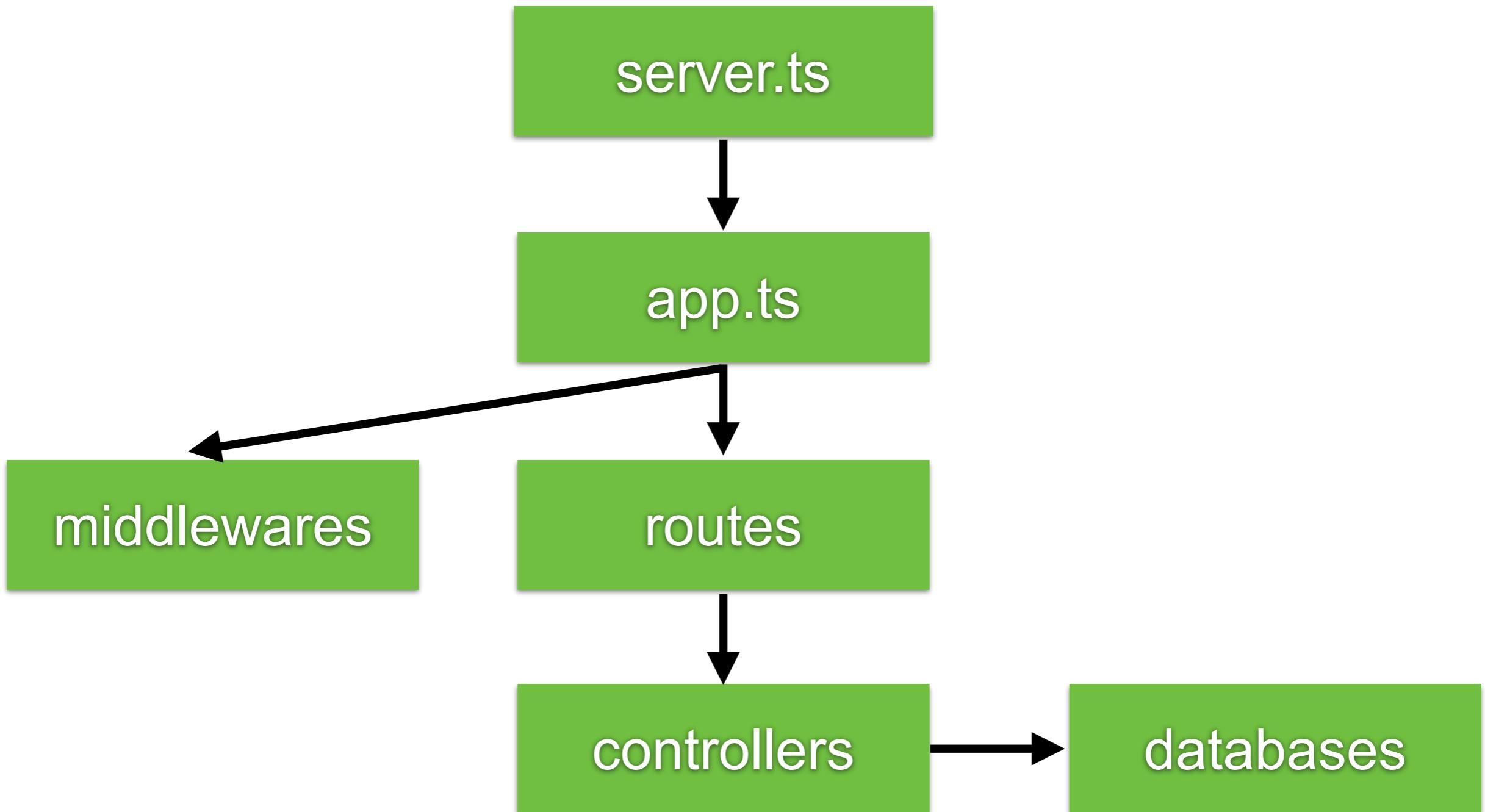
```
docker compose build  
docker compose up -d
```



Structure of project ?



Structure of project ?



Routes and Controller

Authentication and Authorization
Error handling
Validation



Controller

Receive request and return response to client

```
import { NextFunction, Request, Response } from "express";

export default async (req: Request, res: Response, next:
NextFunction) => {
  try {
    res.status(200).json({ message: "Hello World!" });
  } catch (error) {
    next(error);
  }
};
```



Routes

Manage router for domain (group of function)

```
import { Router } from 'express';
import getHello from '../controllers/helloController/
getHello';

const router = Router();
router.get('/', getHello);

export default router;
```



Middlewares

Authentication and Authorization
Error handling
Validation



Middlewares

Error handling

```
import { NextFunction, Request, Response } from "express";

export const errorHandler = (
  err: Error,
  req: Request,
  res: Response,
  next: NextFunction
) => {
  console.error(err);
  res.status(500).send({ errors: [{ message: "Something
went wrong" }] });
};
```



App.ts

```
import express from "express";
import helloRouter from "./routes/hello";
import { errorHandler } from "./middlewares/errors";
const app = express();

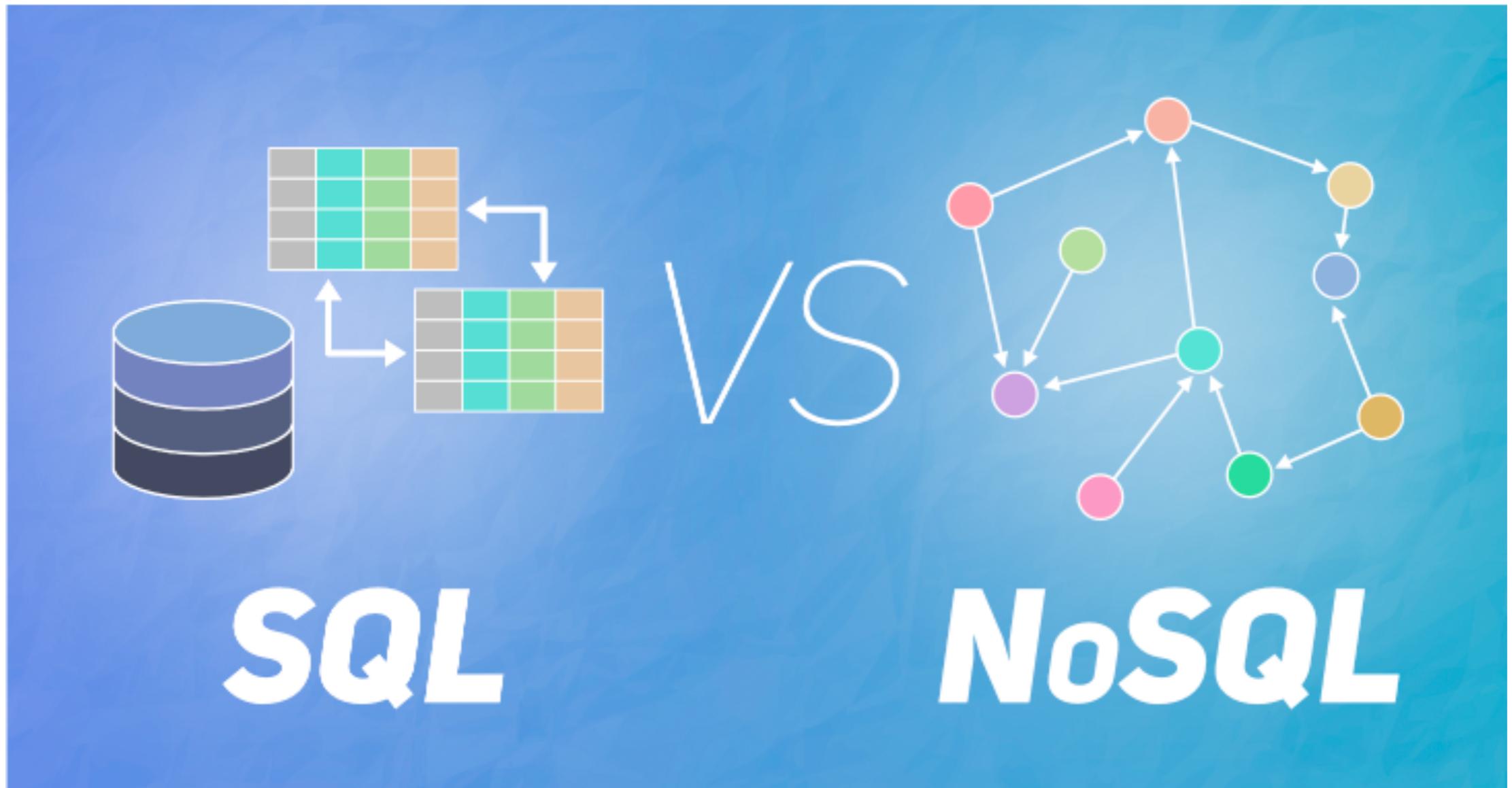
app.use(express.json());
app.use(helloRouter);
app.use(errorHandler);

export default app;
```



Working with Database





NoSQL

Key-Value



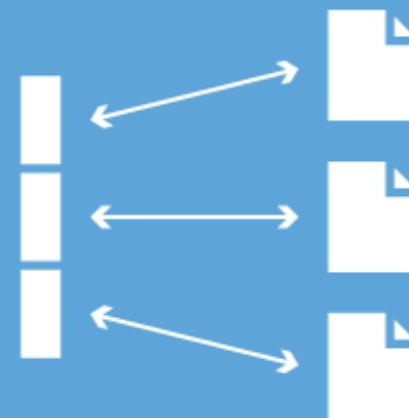
Graph DB



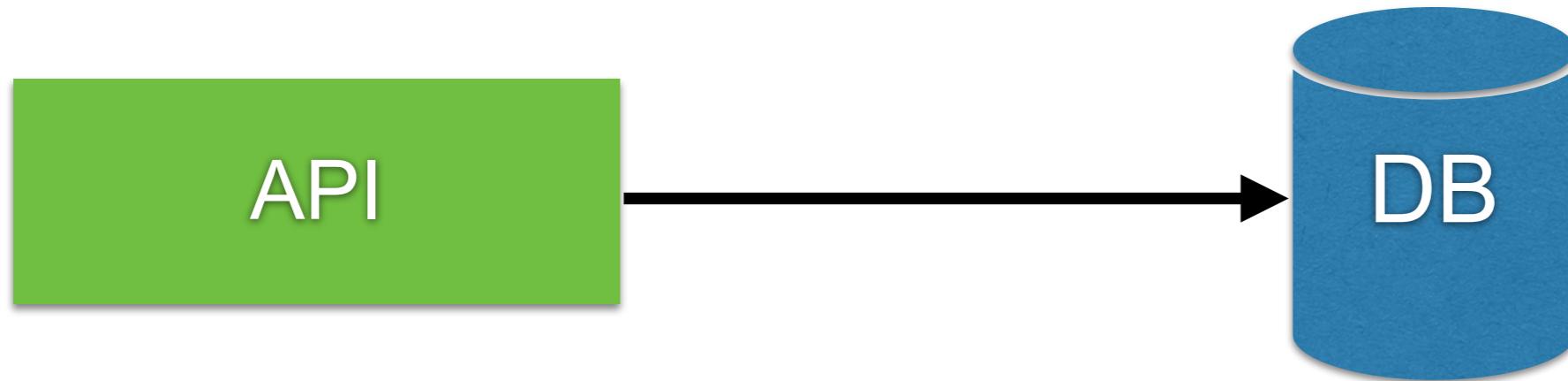
Column Family

1				1
	1	1		1
				1
1			1	1
1	1		1	1
		1	1	1
			1	1

Document



Working with Database



DBML (Database Markup Language)

DBML - Database Markup Language

Open source



Intro

DBML (Database Markup Language) is an open-source DSL language designed to define and document database schemas and structures. It is designed to be simple, consistent and highly-readable.

It also comes with command-line tool and open-source module to help you convert between DBML and SQL.

<https://dbml.dbdiagram.io/>



DBML (Database Markup Language)

```
Table Post {
    id Int [pk, increment]
    title String [not null]
    content String [not null]
    likesCount Int [not null, default: 0]
    createdAt DateTime [default: `now()`, n
    updatedAt DateTime [not null]
    comments Comment [not null]
}

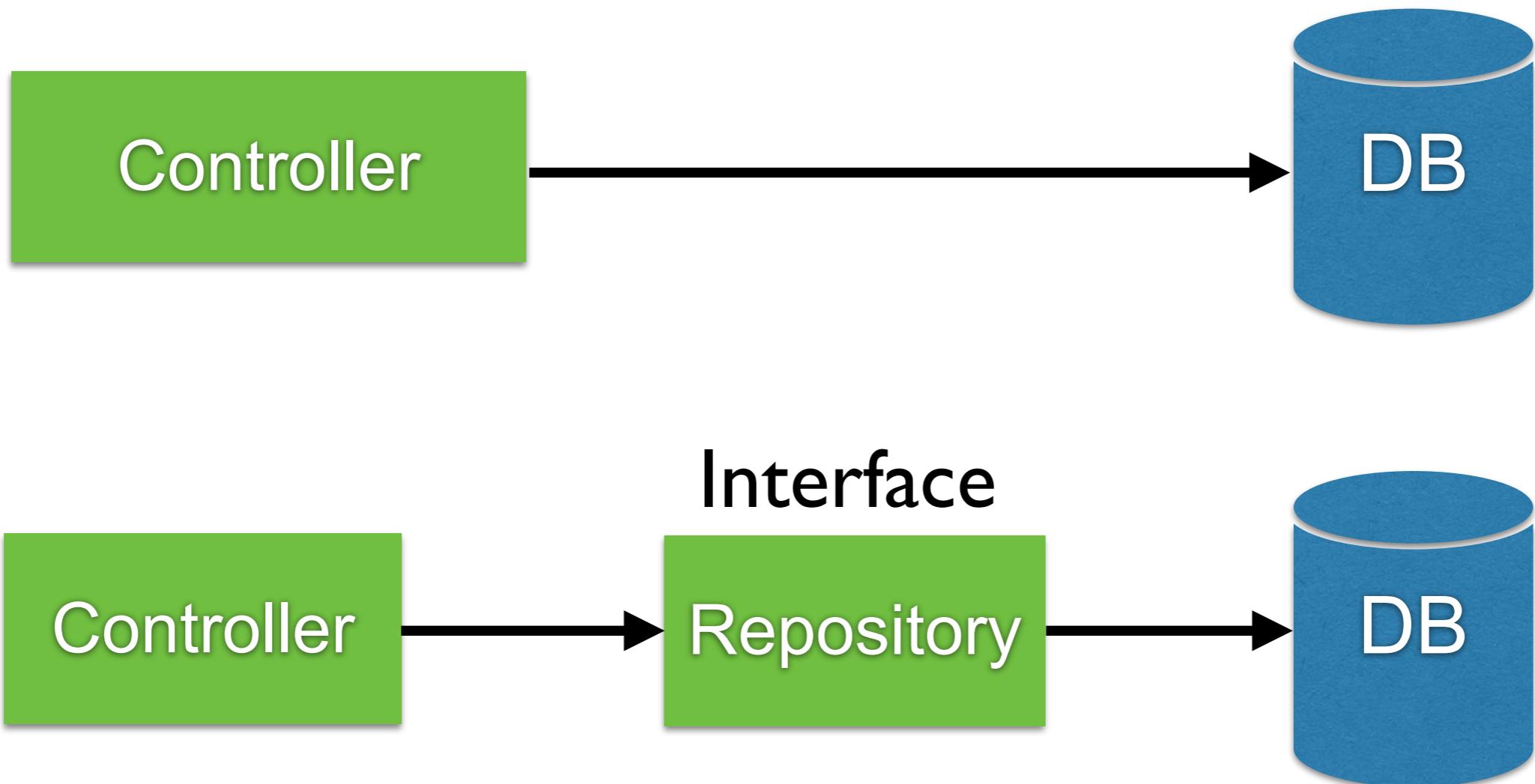
Table Comment {
    id Int [pk, increment]
    content String [not null]
    createdAt DateTime [default: `now()`, n
    updatedAt DateTime [not null]
    post Post [not null]
    postId Int [not null]
}
```

The diagram illustrates a database schema with two tables: **Post** and **Comment**.
Post Table:
- Column **id**: Int, Primary Key, Increment.
- Column **title**: String, Not Null.
- Column **content**: String, Not Null.
- Column **likesCount**: Int, Not Null, Default: 0.
- Column **createdAt**: DateTime, Default: `now()`, Not Null.
- Column **updatedAt**: DateTime, Not Null.
- Column **comments**: Comment, Not Null.
Comment Table:
- Column **id**: Int, Primary Key, Increment.
- Column **content**: String, Not Null.
- Column **createdAt**: DateTime, Not Null.
- Column **updatedAt**: DateTime, Not Null.
- Column **post**: Post, Not Null.
- Column **postId**: Int, Not Null.
A relationship line connects the **Post** table to the **Comment** table, indicating that multiple comments can be associated with a single post.

<https://dbml.dbdiagram.io/>



Working with Database



Continuos Integration



