สยามชำนาญกิจ
SIAM CHAMNAN KIT

Fundamental for Software Development

https://github.com/up1/course-full-stack-development

Technical Excellence

SPECIFICATION BY EXAMPLE

CONTINUOUS INTEGRATION

CONTINUOUS DELIVERY

TEST AUTOMATION

TECHNICAL EXCELLENCE

ARCHITECTURE & DESIGN

ACCEPTANCE TESTING

THINKING ABOUT TESTING

TEST-DRIVEN DEVELOPMENT

UNIT TESTING

CLEAN CODE

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

Delivery Process

Delivery process

UI Flow

API design

Database design

Pipeline

Version Control

Git

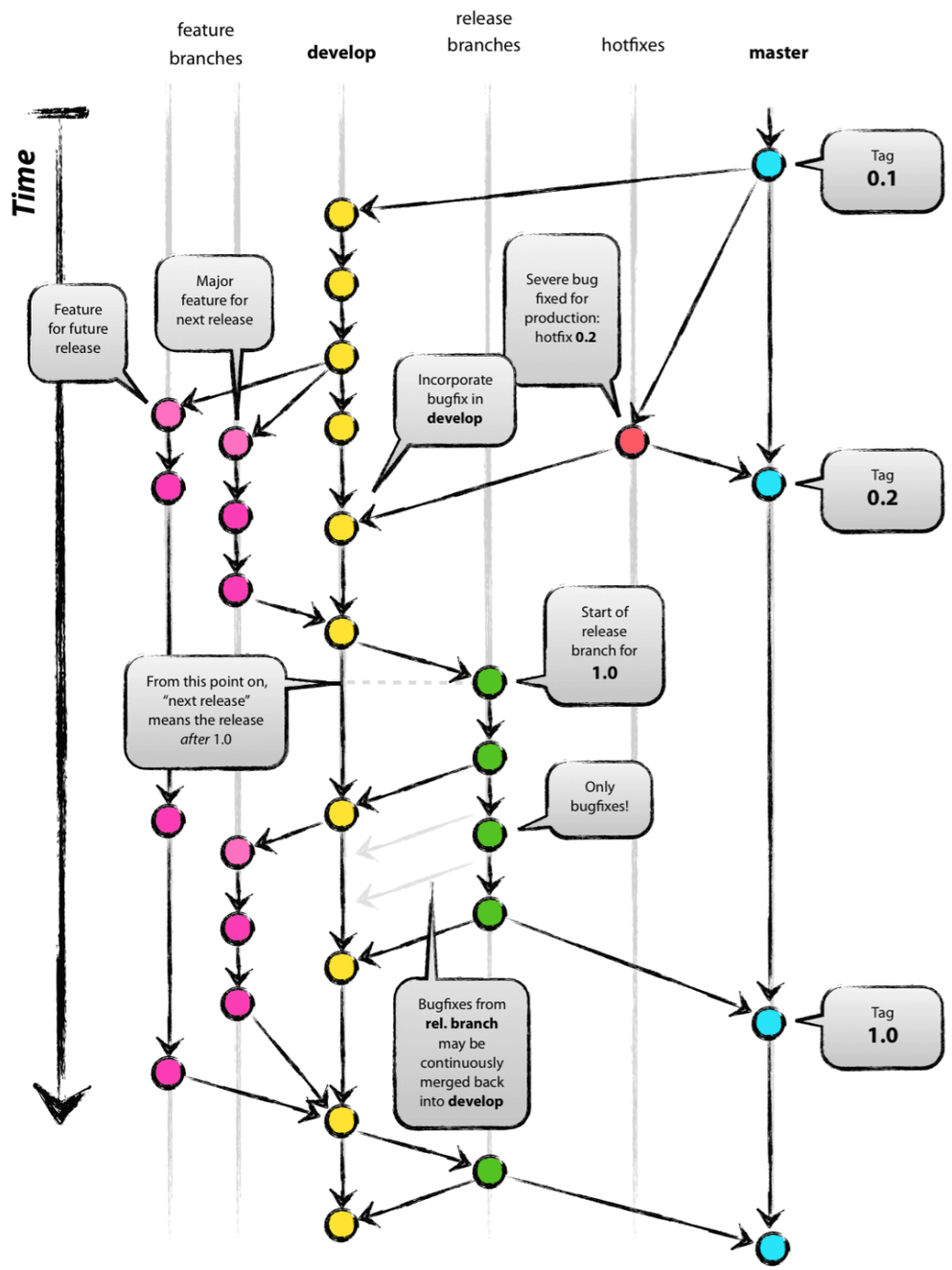
git --distributed-even-if-your-workflow-isnt

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.

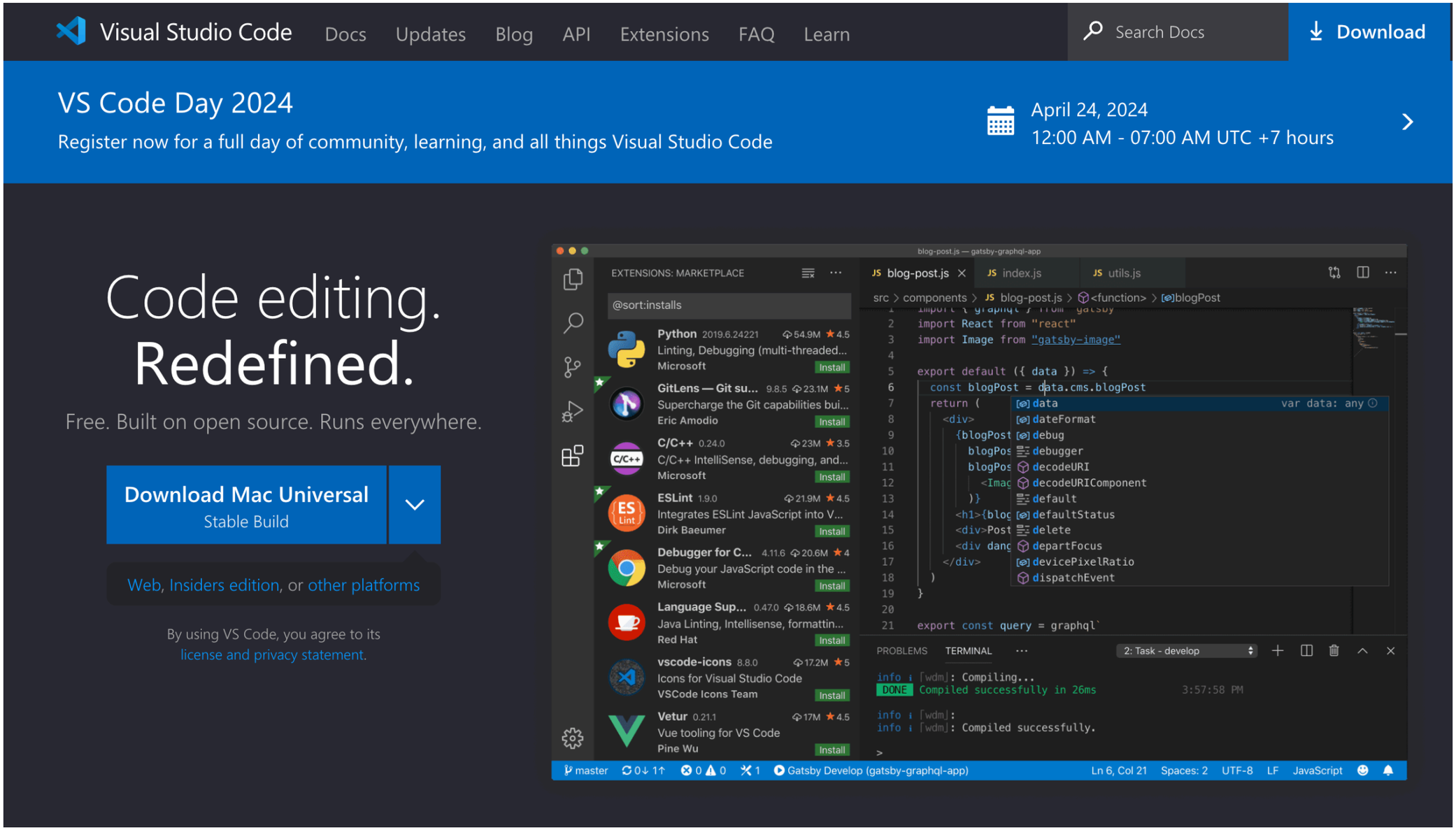
https://git-scm.com/

Branch Strategies



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

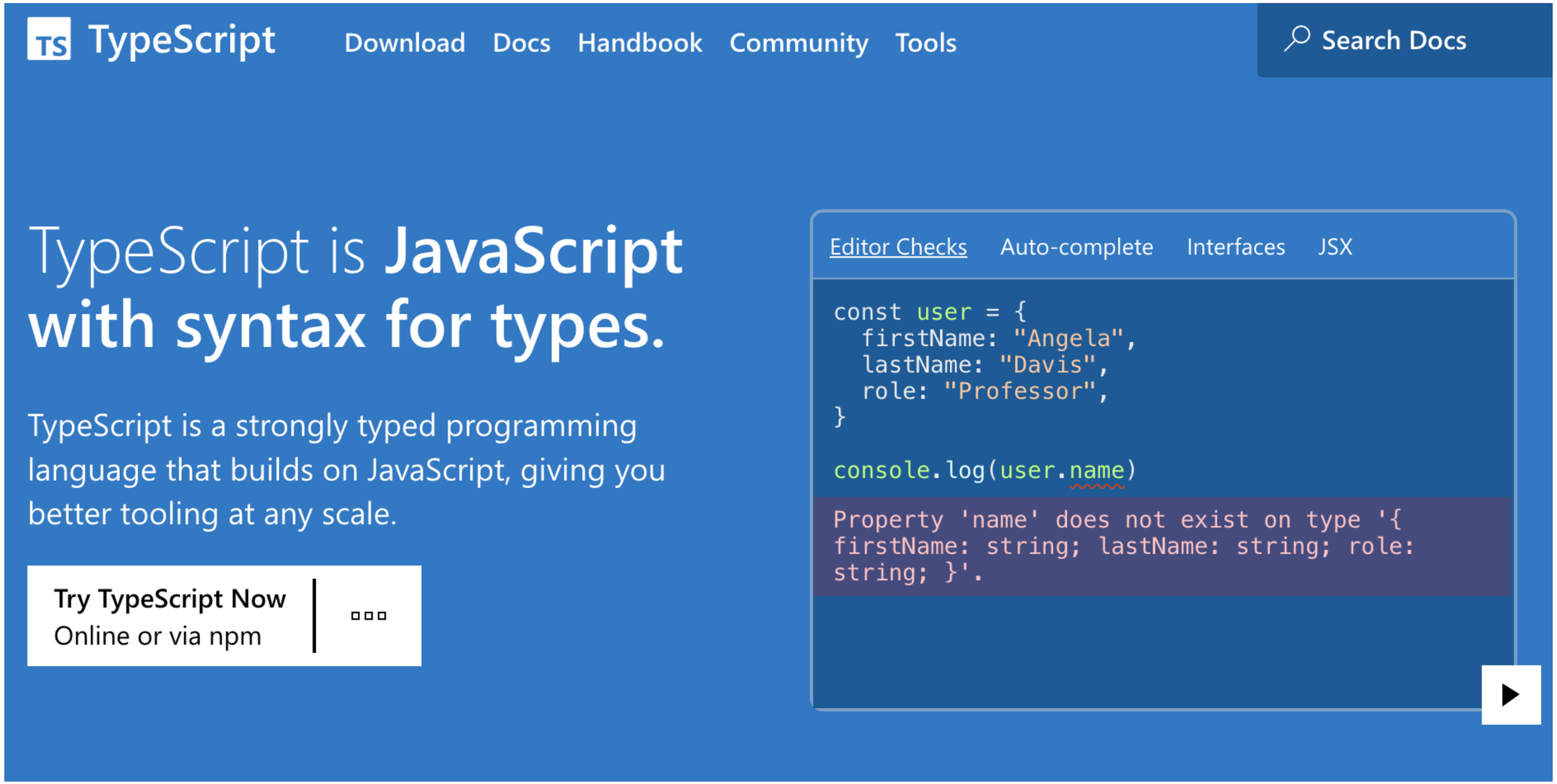
VS Code



https://code.visualstudio.com/

TypeScript

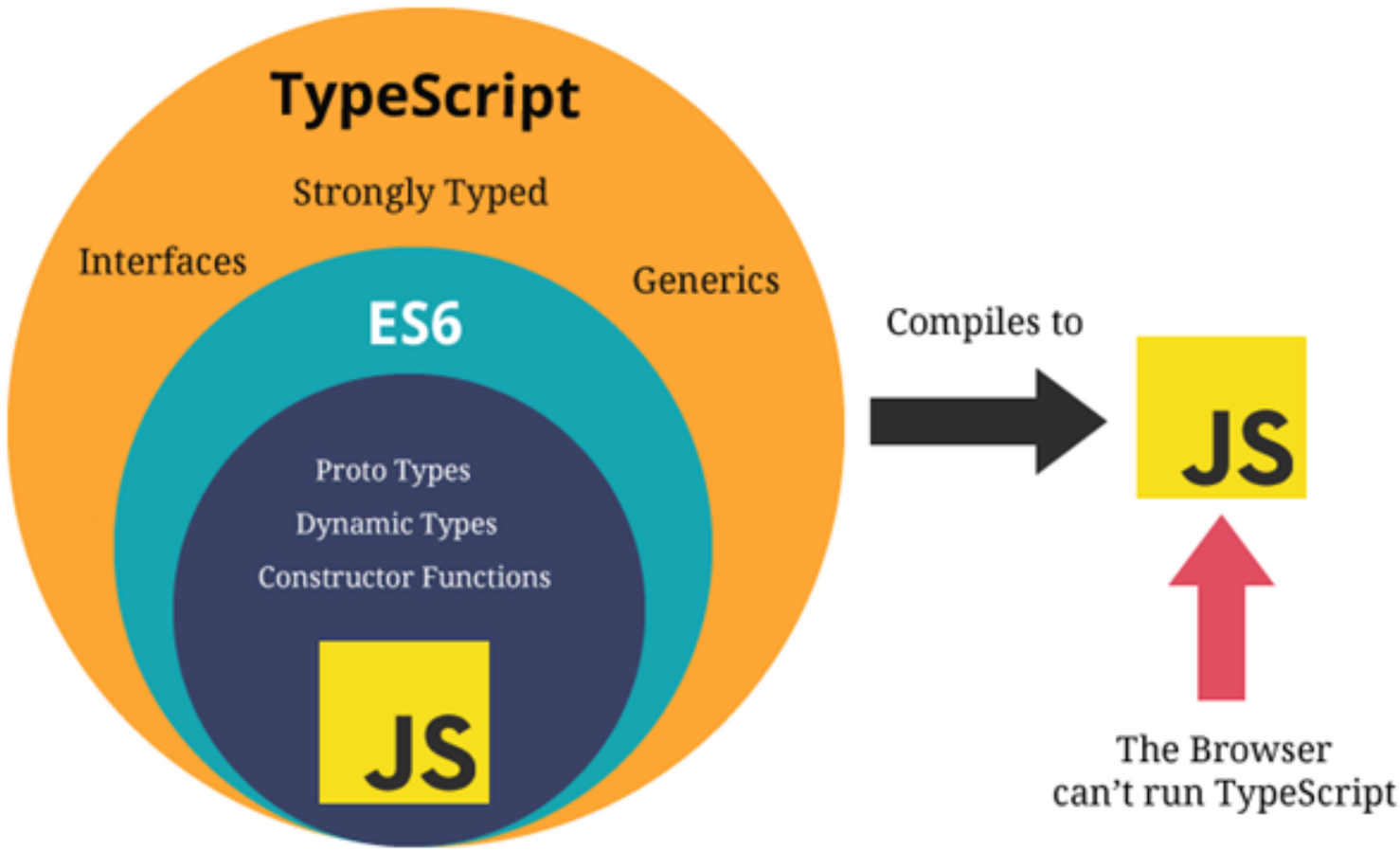
TypeScript



https://www.typescriptlang.org/

TypeScript

A superset of JavaScript



https://www.typescriptlang.org/

TypeScript Essentials

Basic

Type in TypeScript

Advance

Design patterns

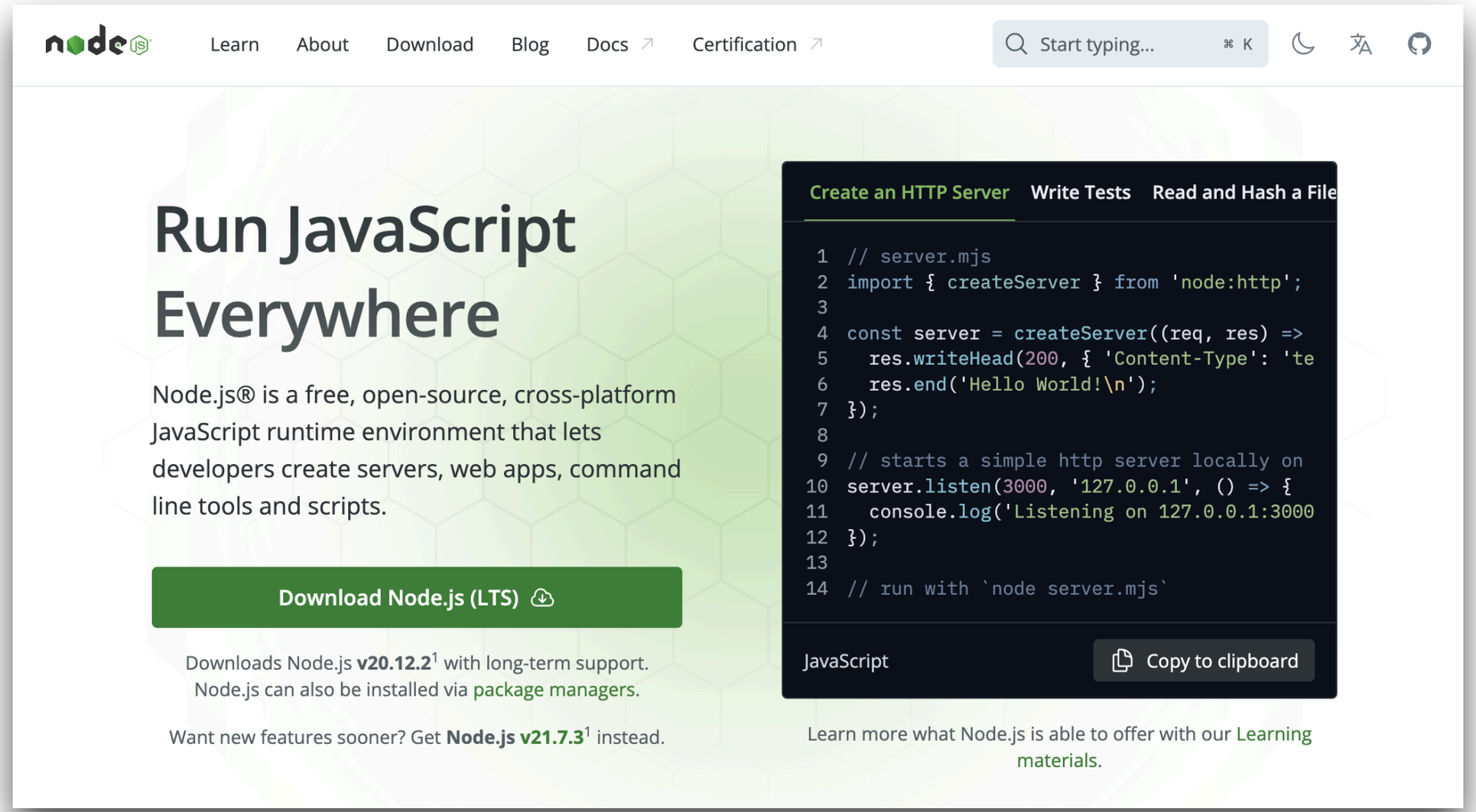
Key idea of TypeScript

It adds **static-typing** to JavaScript
More maintainable
Easy to learn and use

Setup project with TypeScript

Software Requirements

NodeJS 18+



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

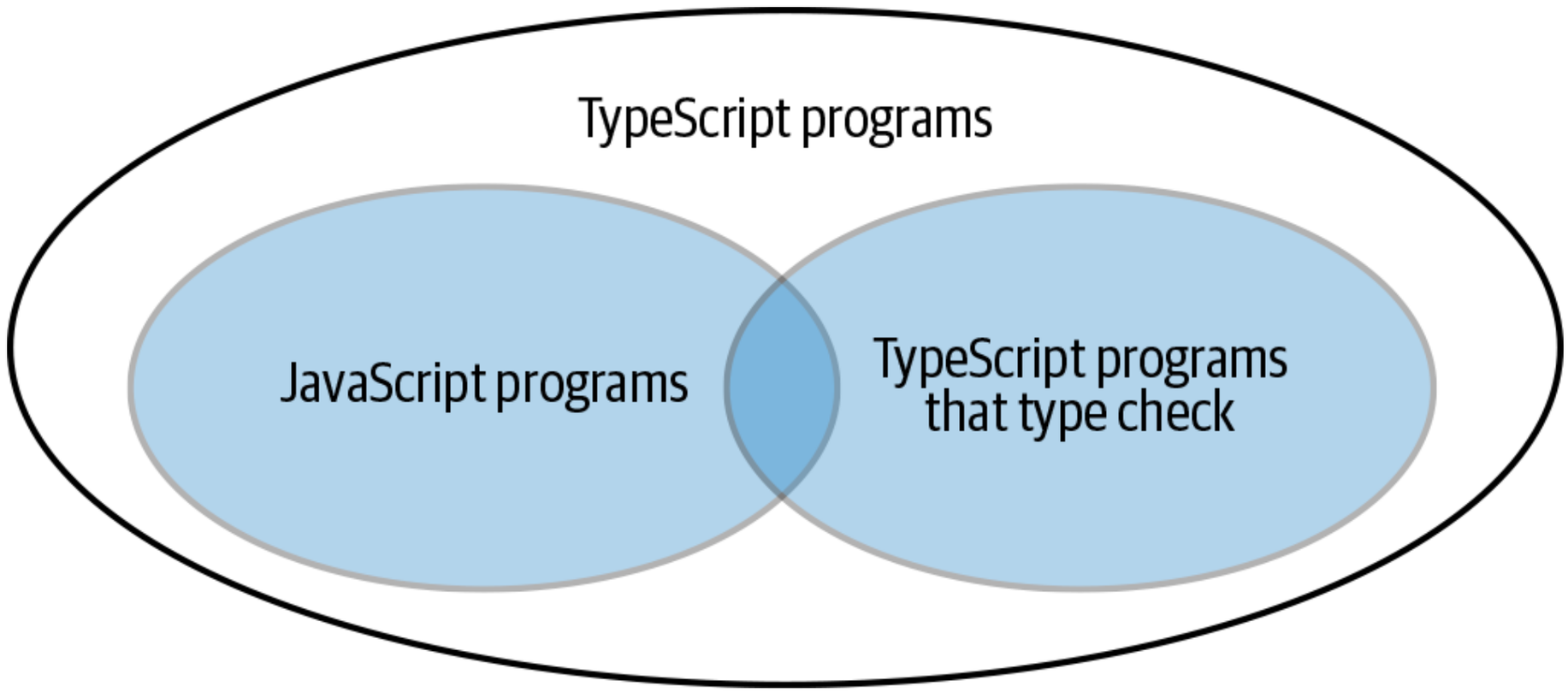
$tsc --init

Create project (2)

$npm init

$npm install typescript -S

Type Checking

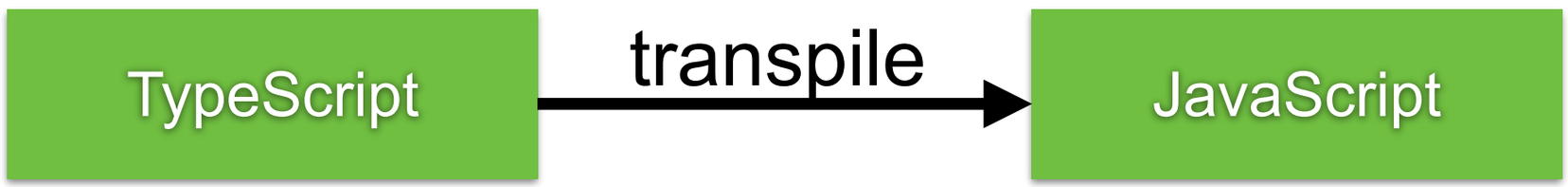


Hello TS

$tcs hello.ts

$node hello.js

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



Variable Declarations

var , let, const

Function declaration

```ts
// 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

```ts
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

```ts
// 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

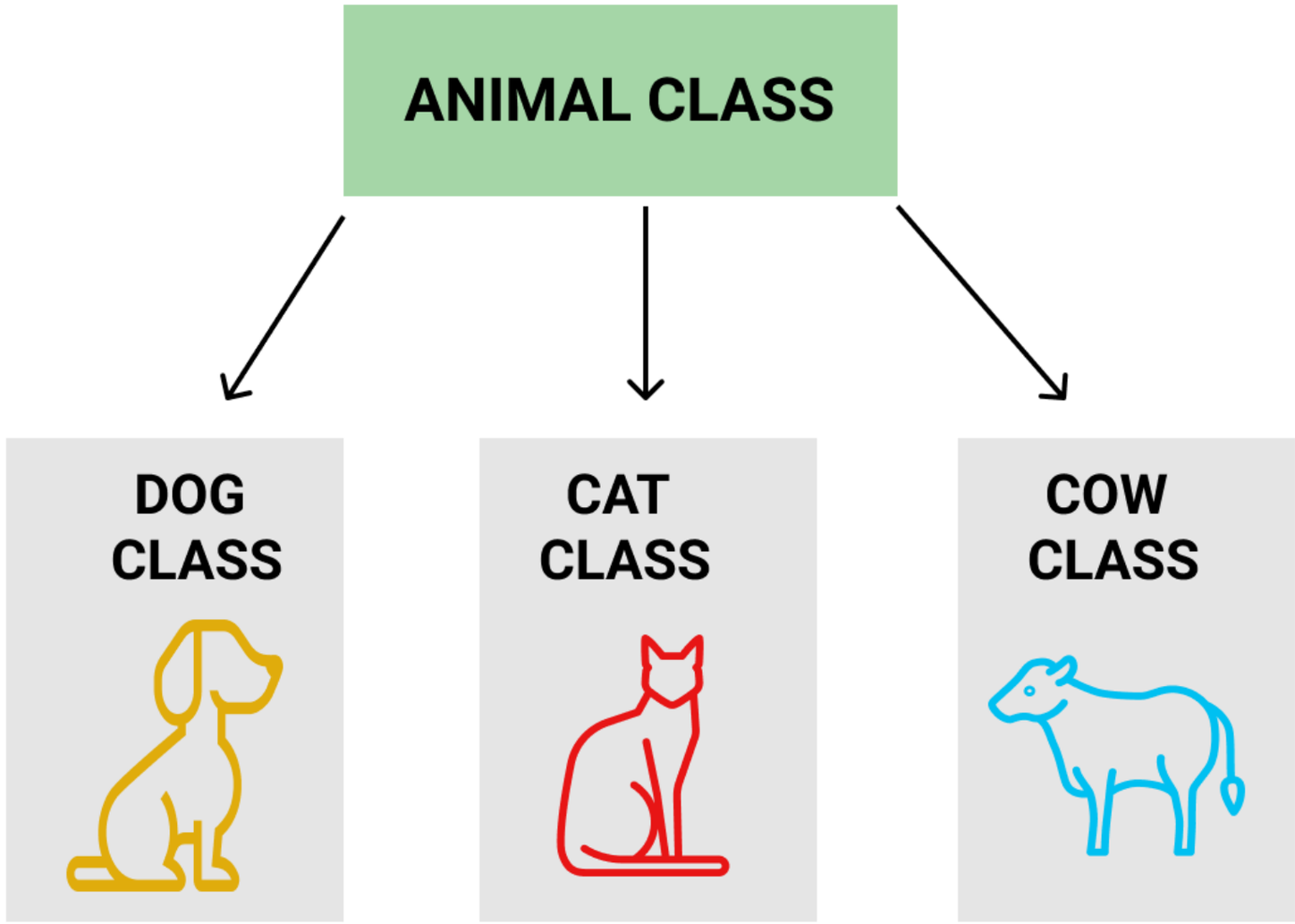
```
class Person {
    name: string;
    age: number;

    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)

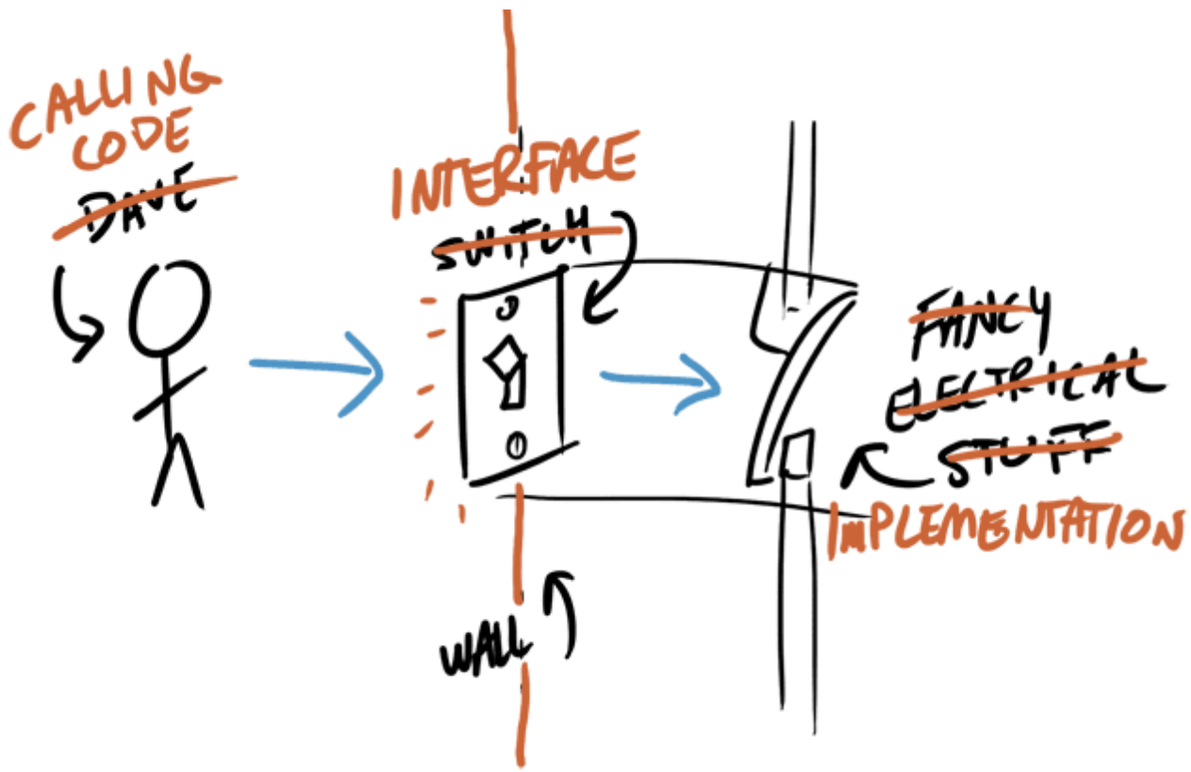
Protected

Private

Readonly

Interface (Encapsulation)

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



Interface

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

```ts
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

```ts
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

```ts
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

```ts
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

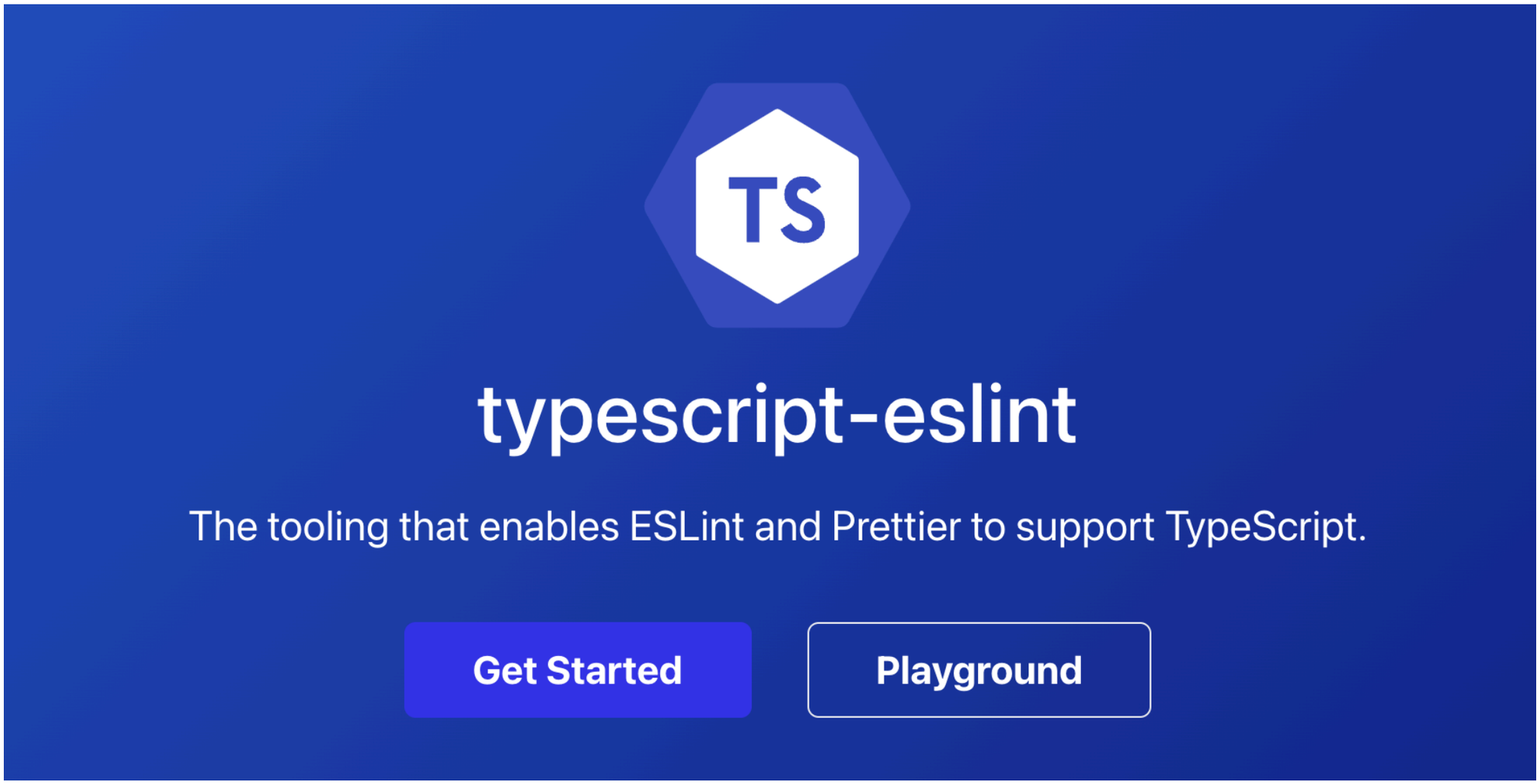
tsconfig.json file

TypeScript configuration file
Compiler settings

```
{
    "include": ["src", "tests"],
    "exclude": ["node_modules"],

    "compilerOptions": {
        "outDir": "build",
        "target": "es6"
    }
}
```

TSLint and ESLint

Enforce coding style



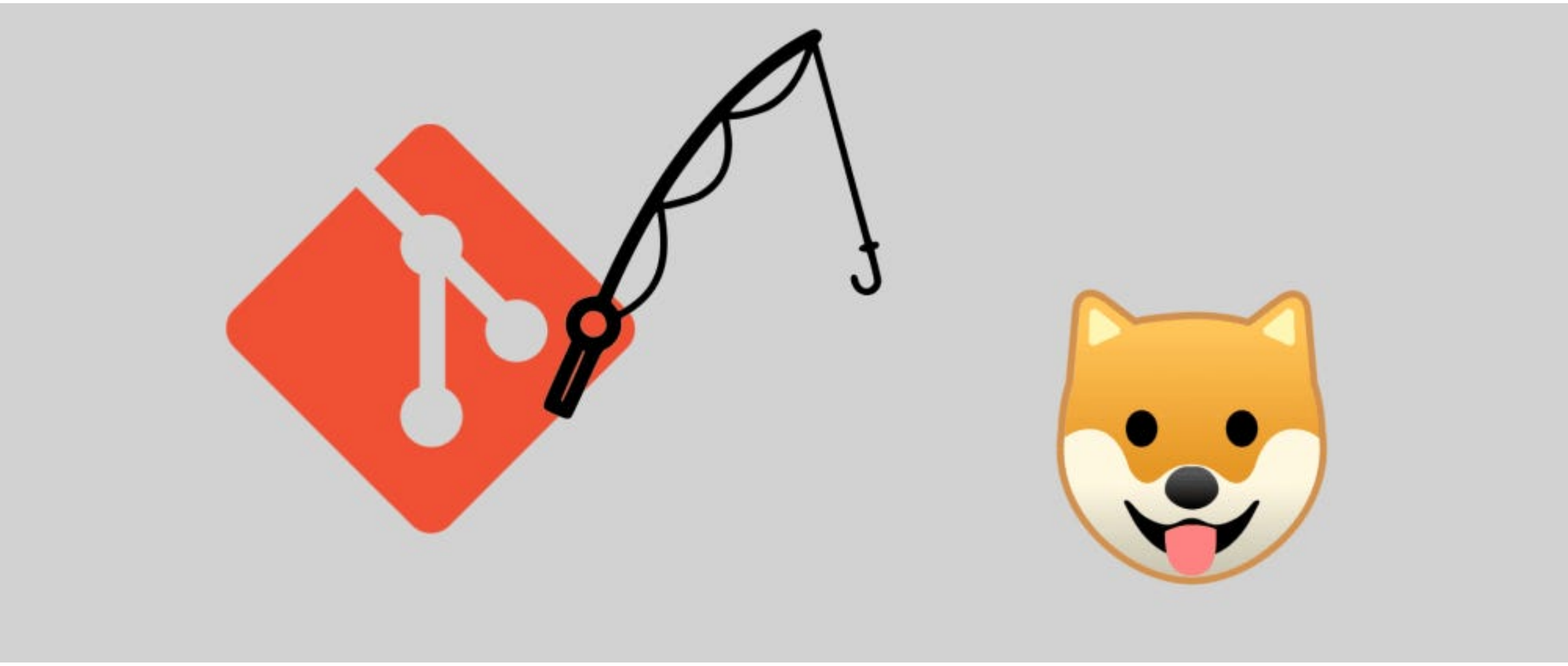
https://typescript-eslint.io/

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


```
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

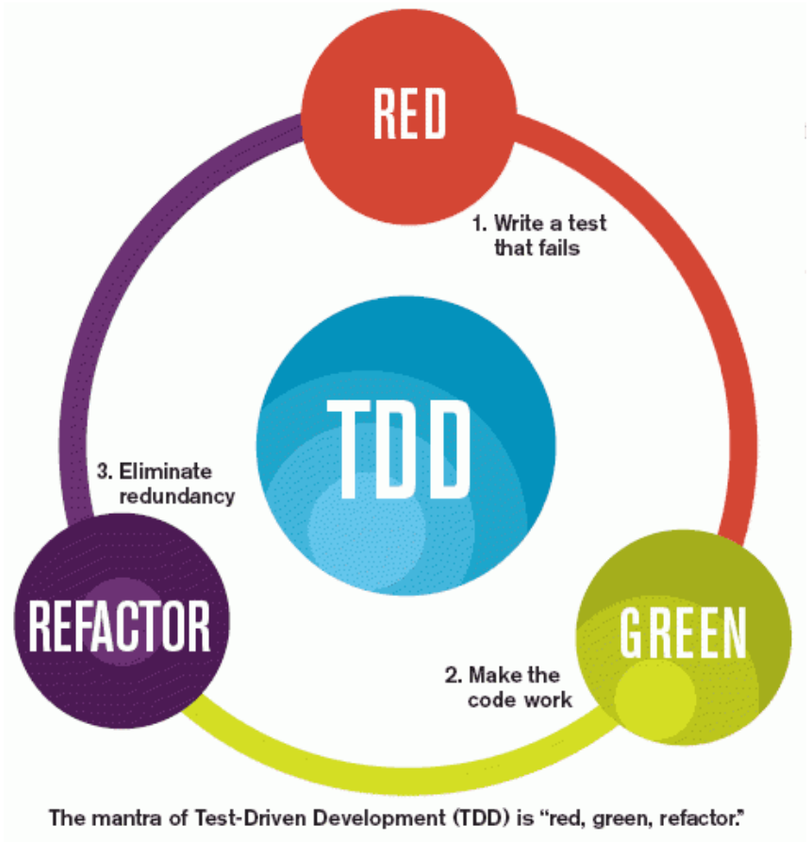
```ts
import type {Config} from '@jest/types';
// Sync object
const config: Config.InitialOptions = {
  moduleDirectories: ['node_modules', 'src'],
  verbose: true,
  transform: {
  '^.+\\.tsx?$': 'ts-jest',
  }
};
export default config;
```

https://jestjs.io/docs/configuration

Learn from Testing

Test code

Production code



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

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

```ts
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

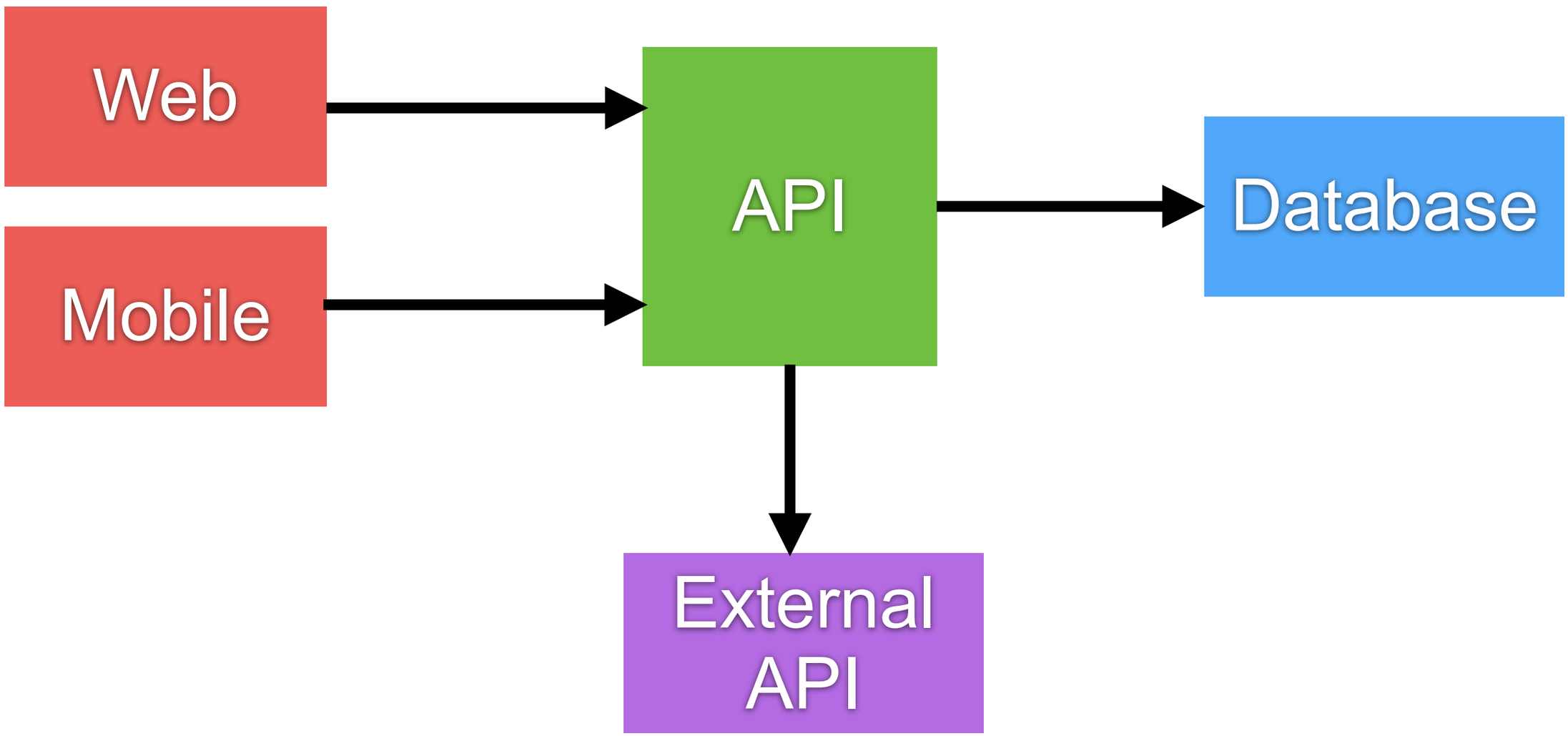
Run test with coverage

npm test

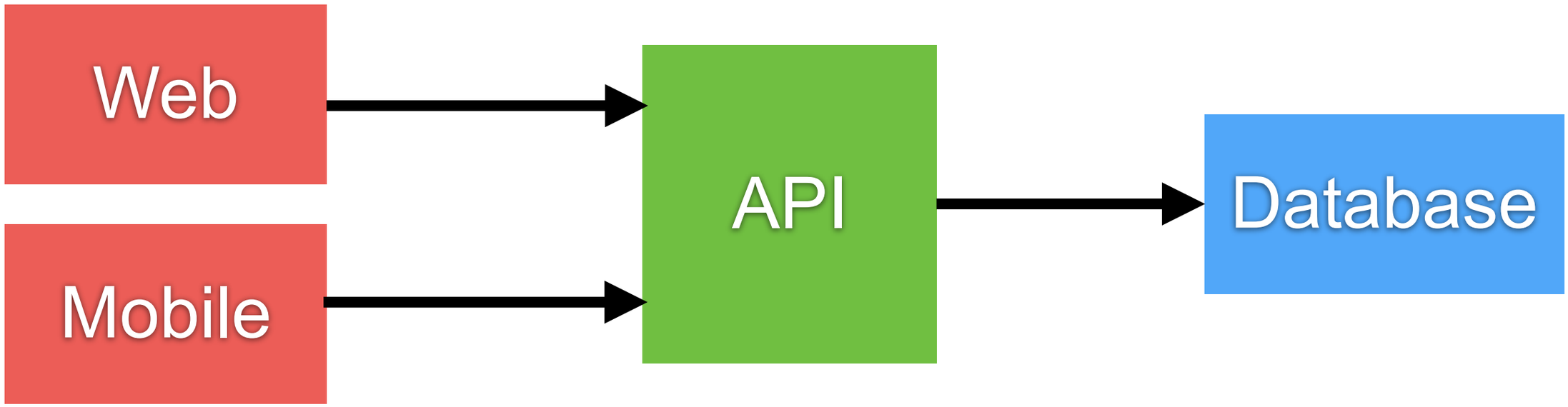
```
PASS tests/unit/hello.test.ts
  ✓ Hello world with sayHi (1 ms)

----------|---------|----------|---------|---------|-------------------
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

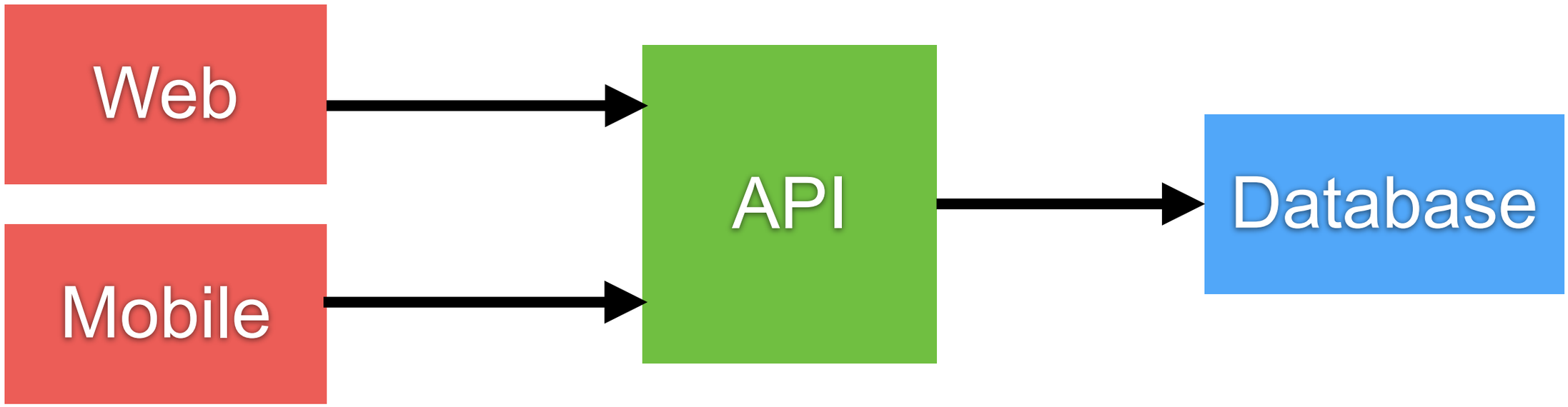
Architecture



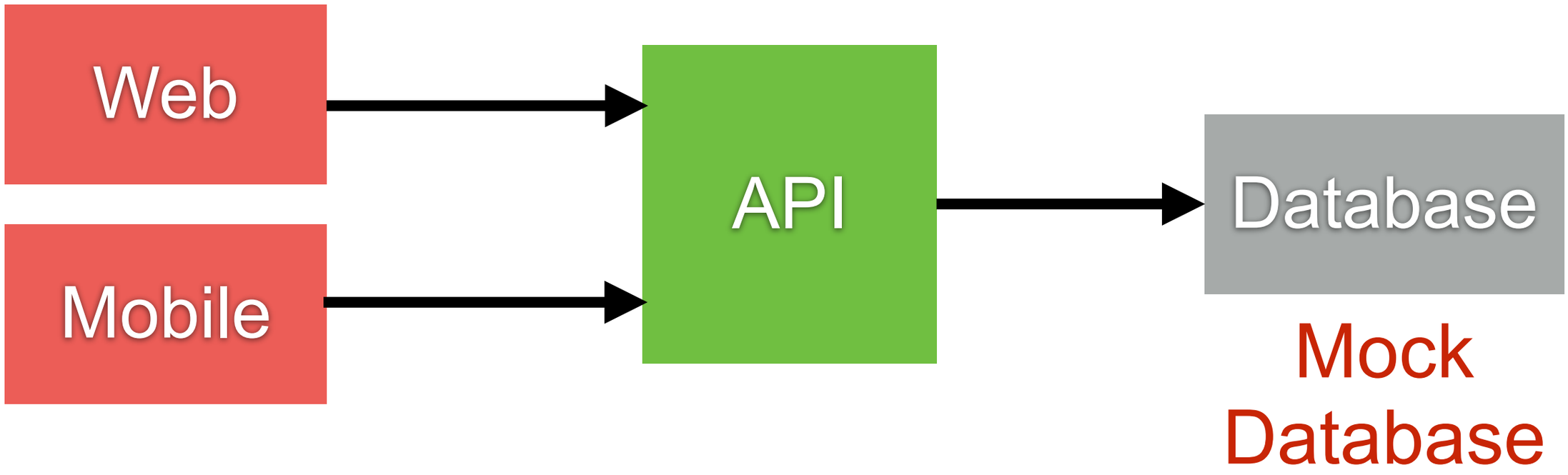
Test ?



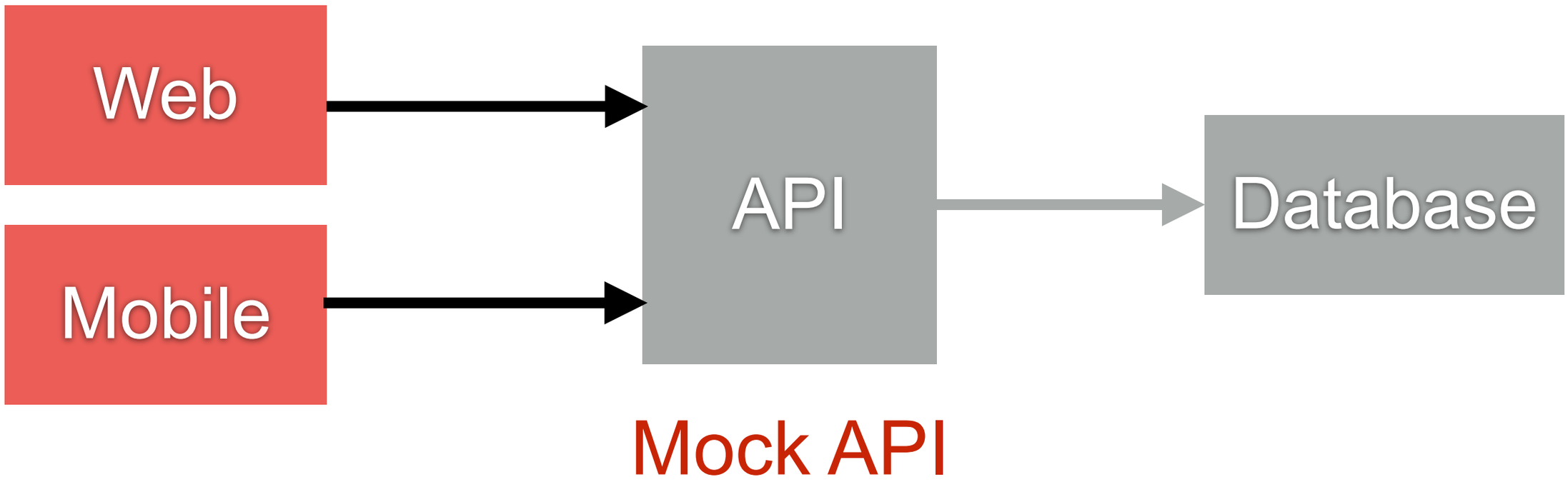
UI Test ?



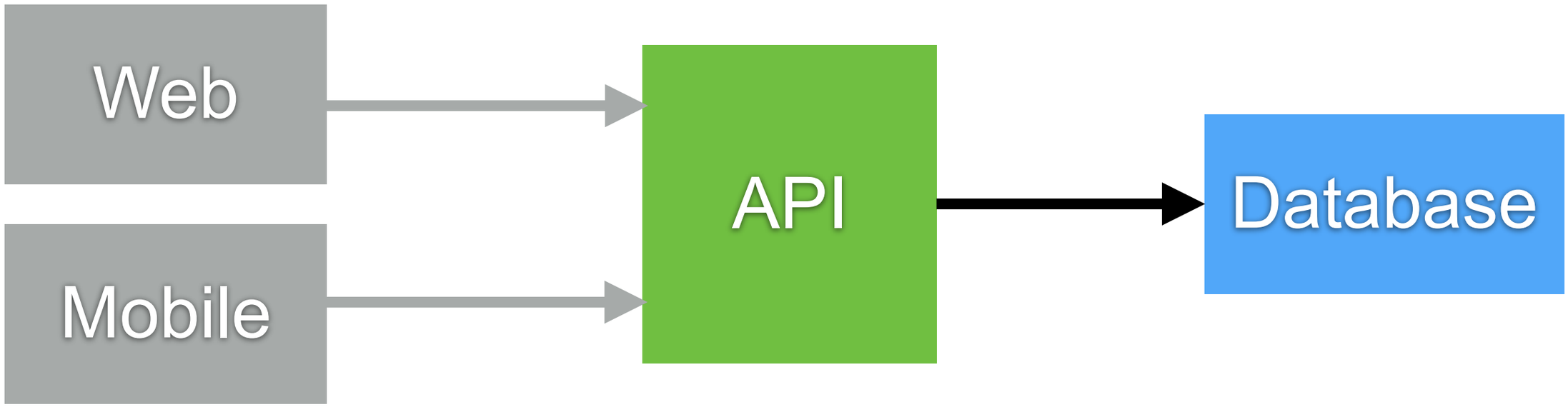
UI Test ?



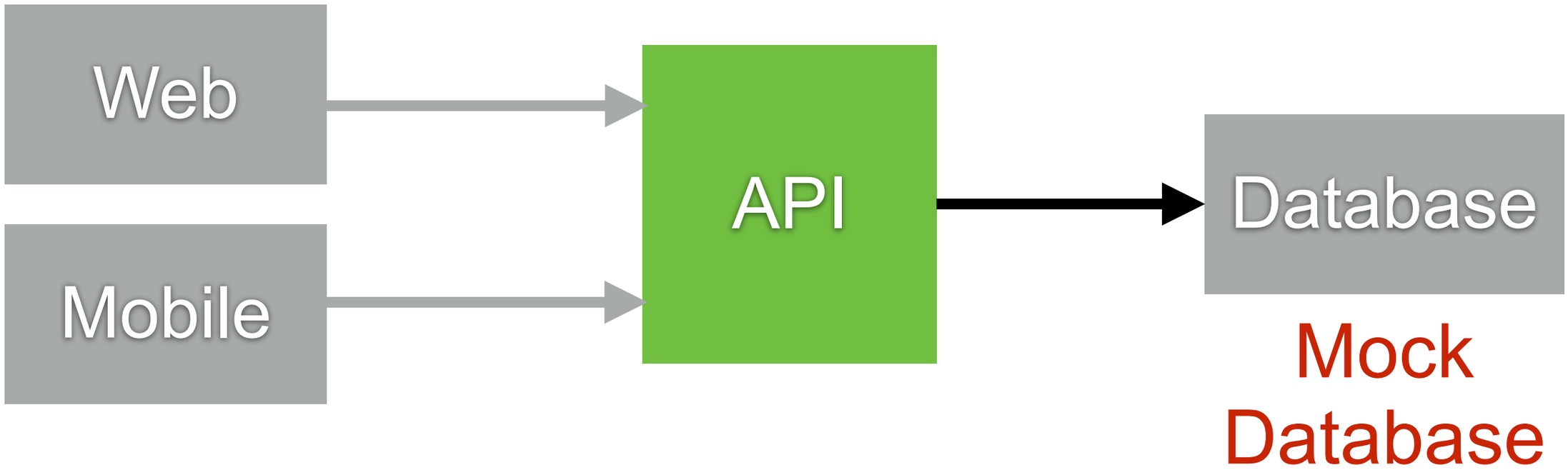
UI Test ?



API Testing ?



API Testing ?



Backend

Develop REST API

Develop REST API

Design

Develop

Test

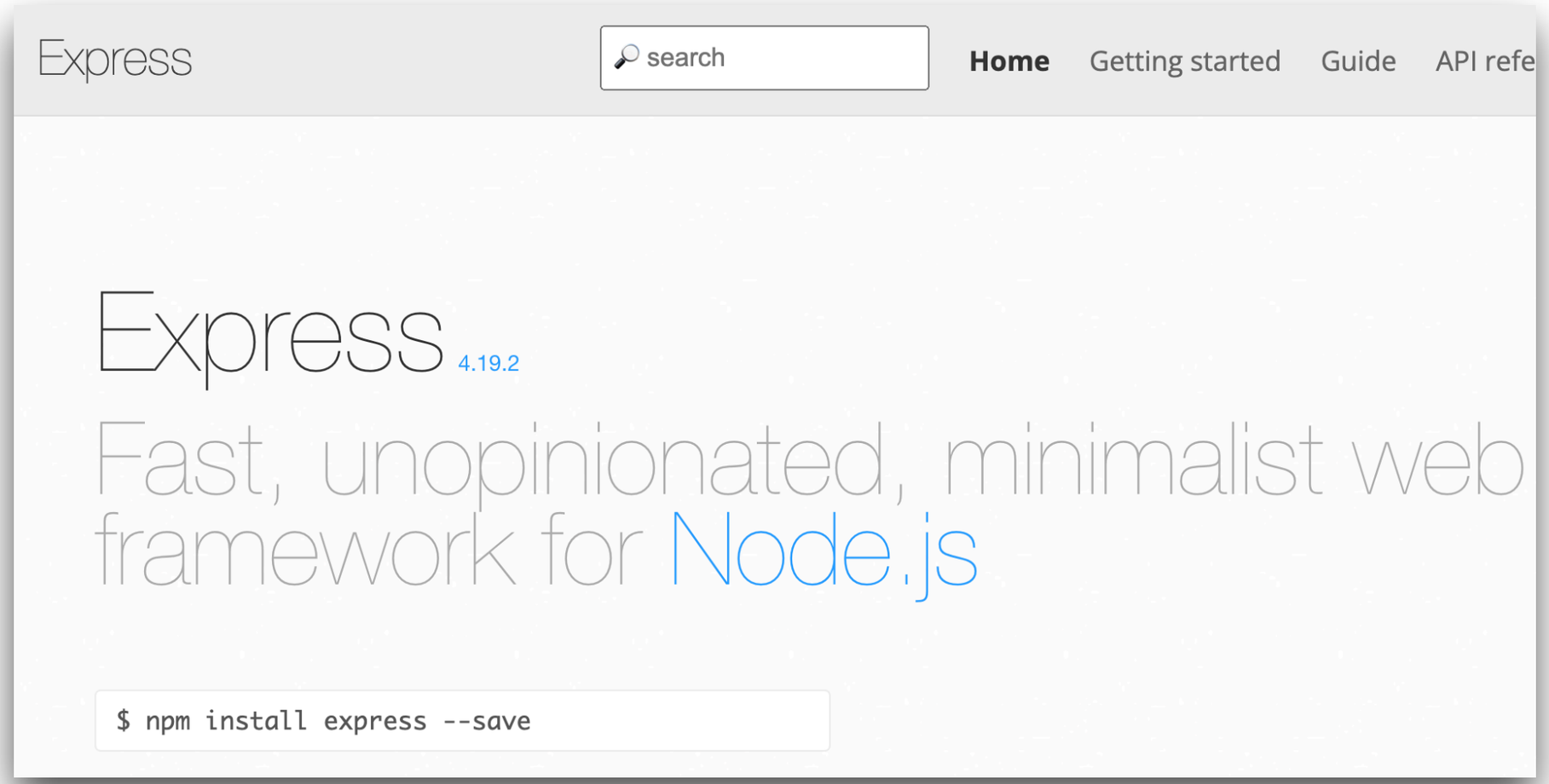
Deploy

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

```ts
import express from "express";

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

app.get("/", (req, res) => {
  res.json({
    messgae: "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' --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

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

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

app.ts

```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

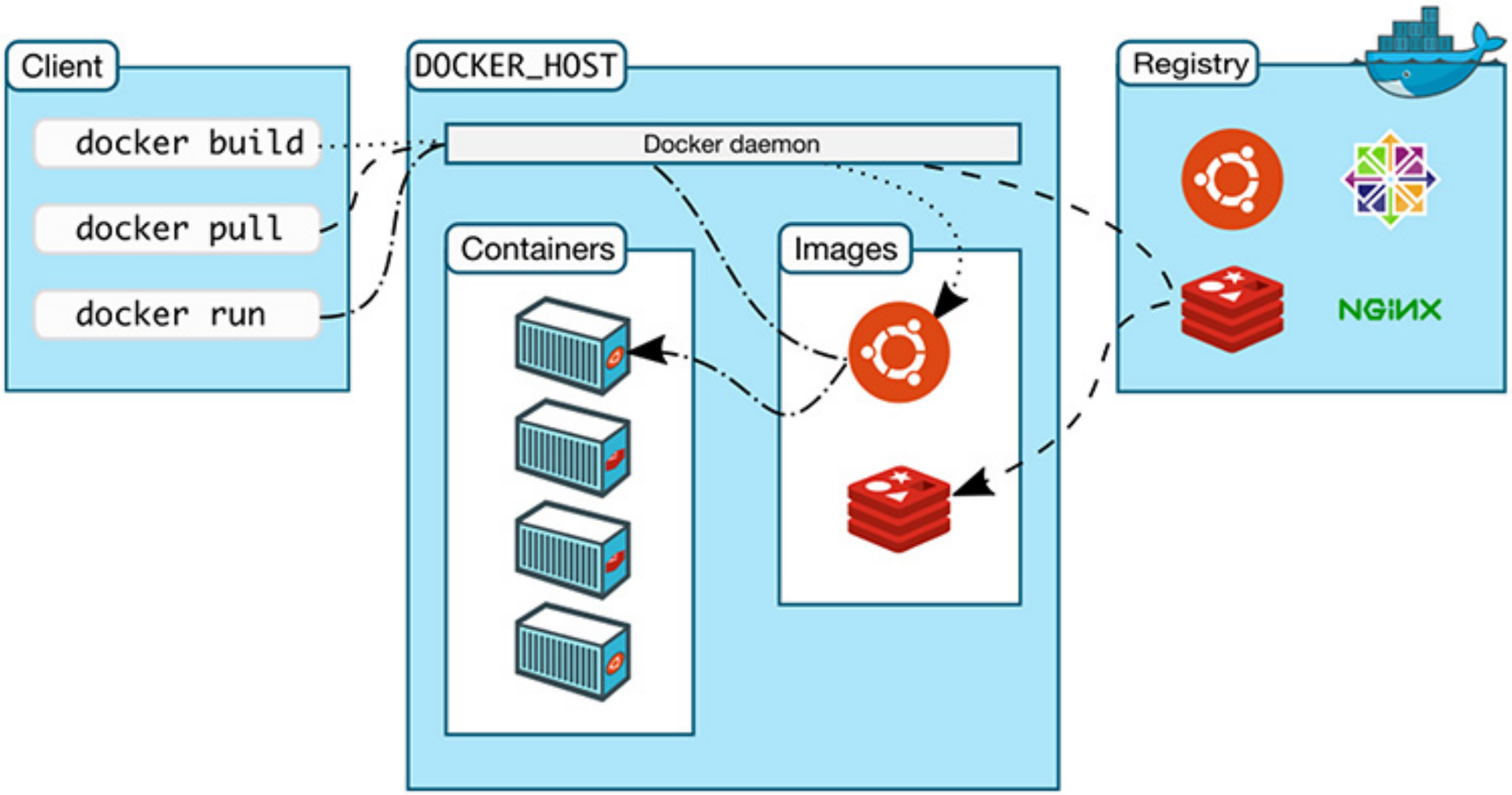
npm test

```ts
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

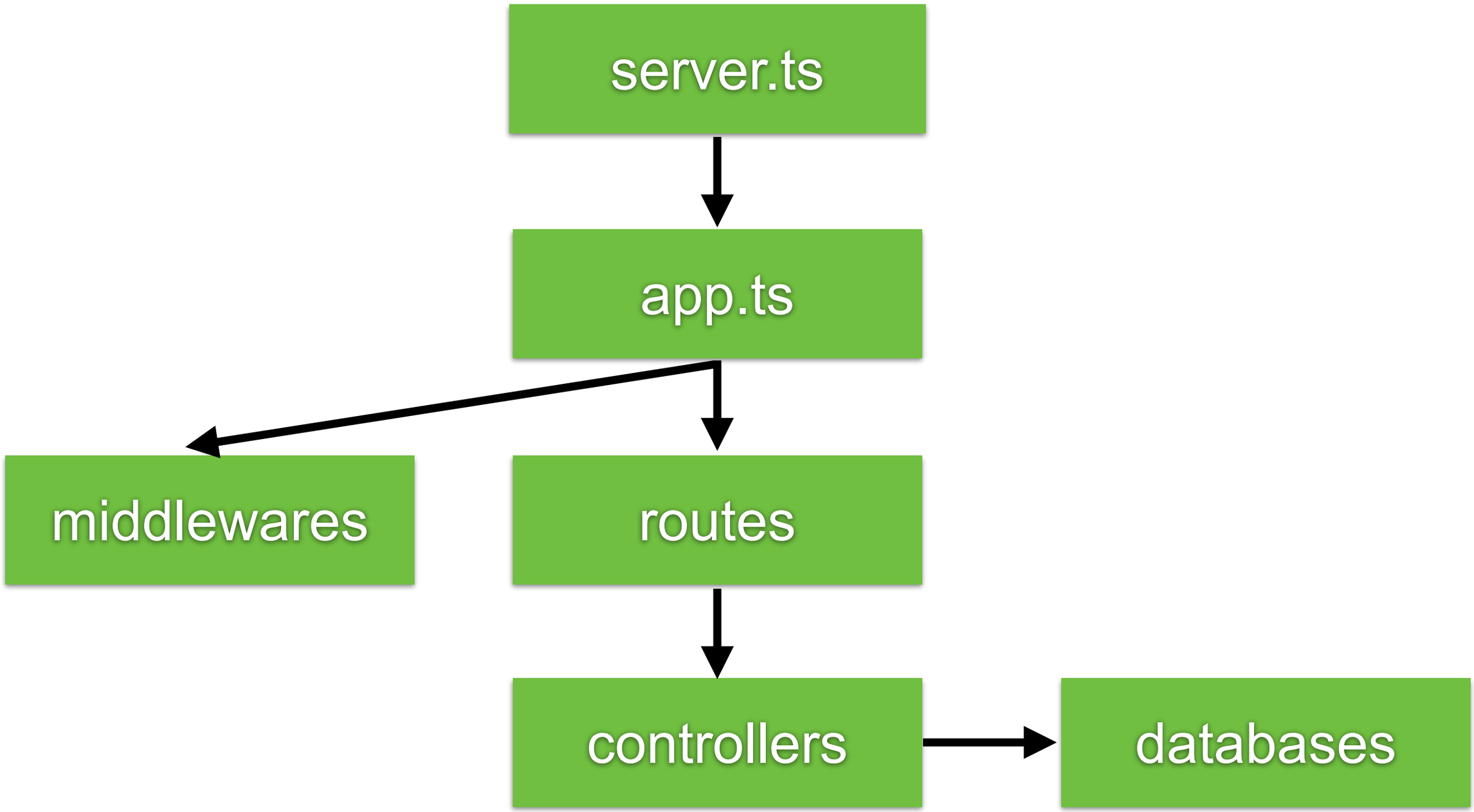
```yaml
services:
  app:
    build: .
    ports:
      - "3000:3000"
```

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

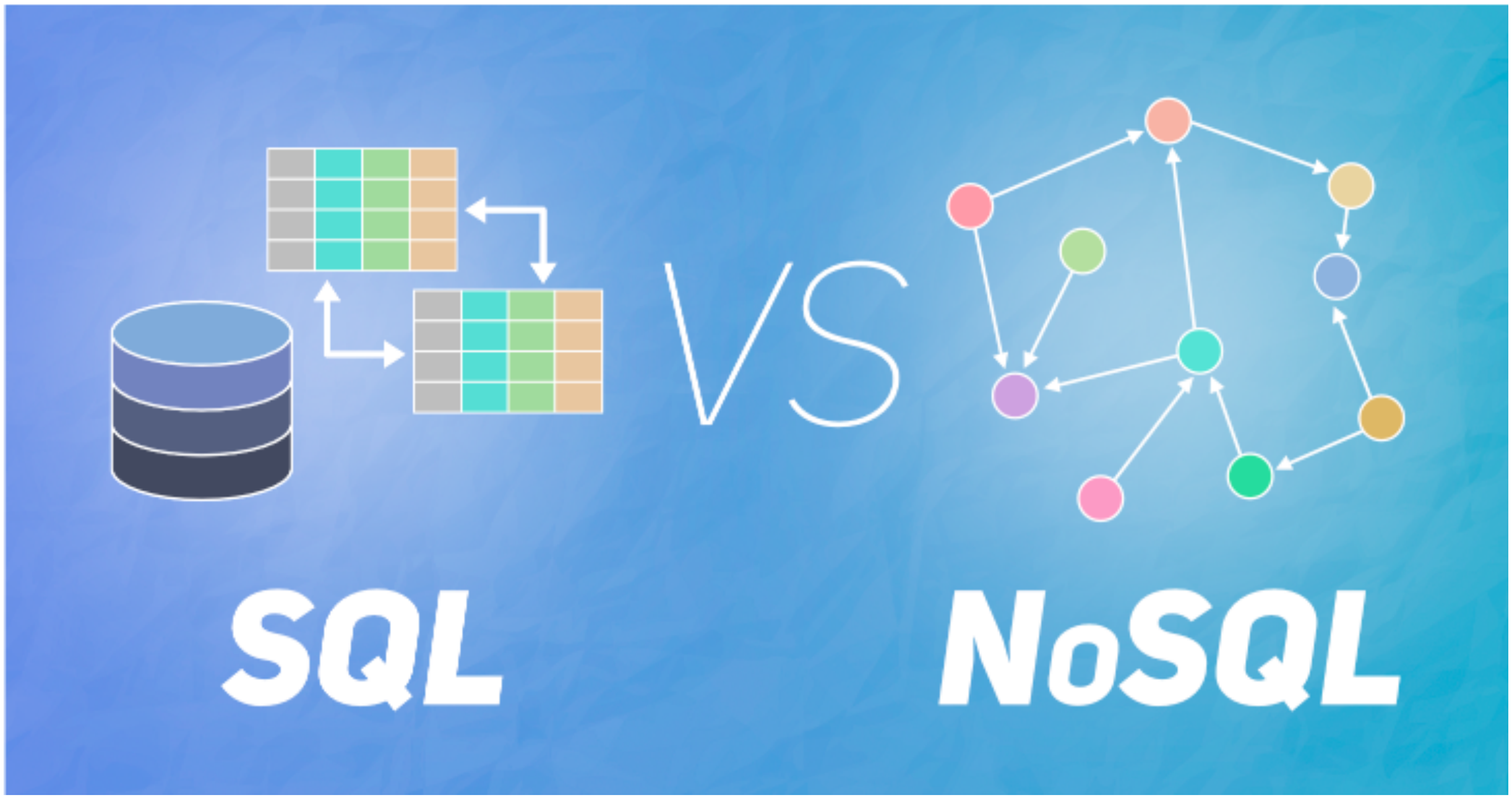
Error handling

```
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" }] });
};
```

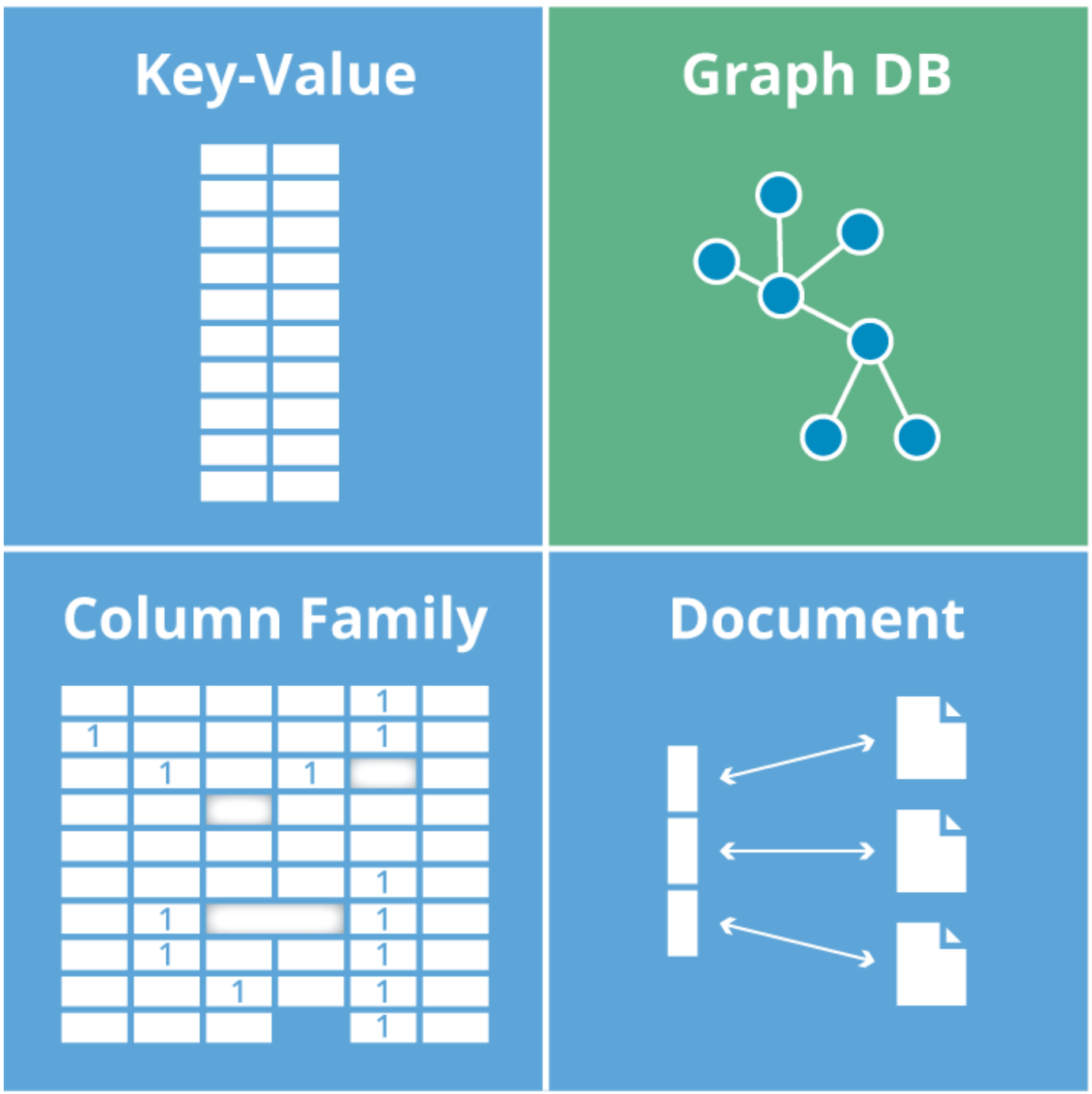
App.ts

```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;
```

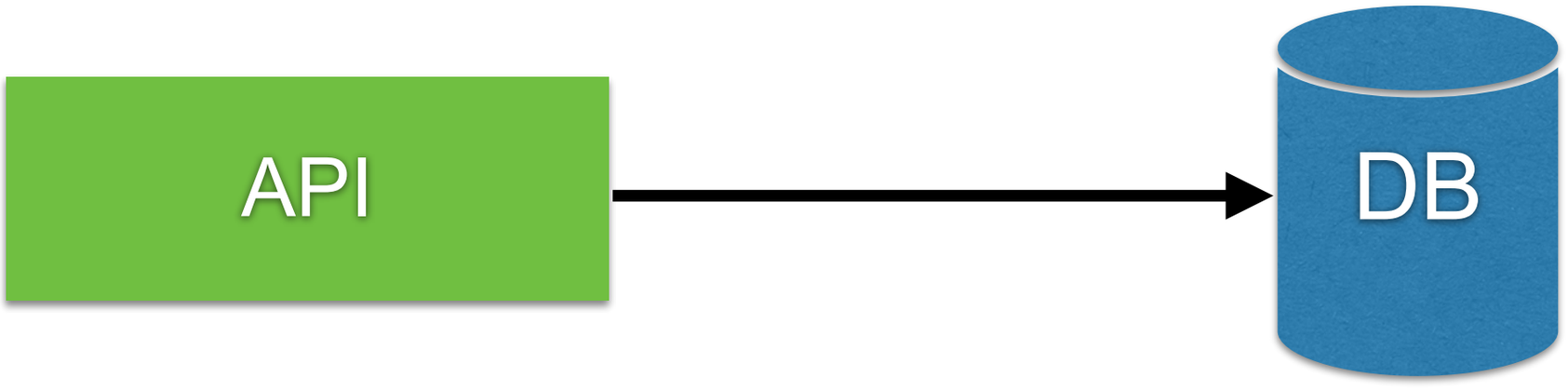
Working with Database



NoSQL



Working with Database



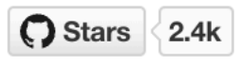
SQL Native

Prisma

TYPEORM

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.

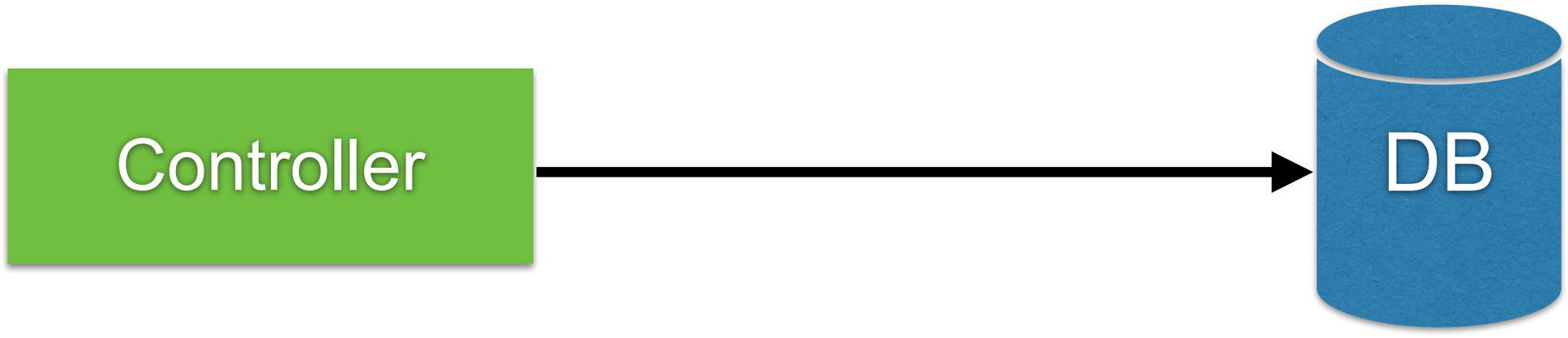
https://dbml.dbdiagram.io/

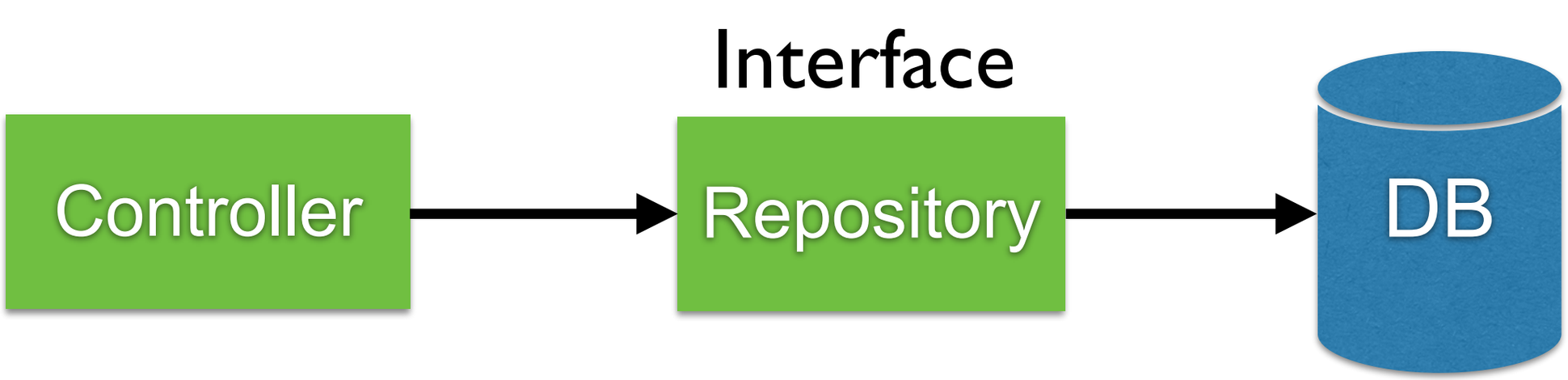
DBML (Database Markup Language)

```
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]
}
```

https://dbml.dbdiagram.io/

Working with Database





Continuos Integration