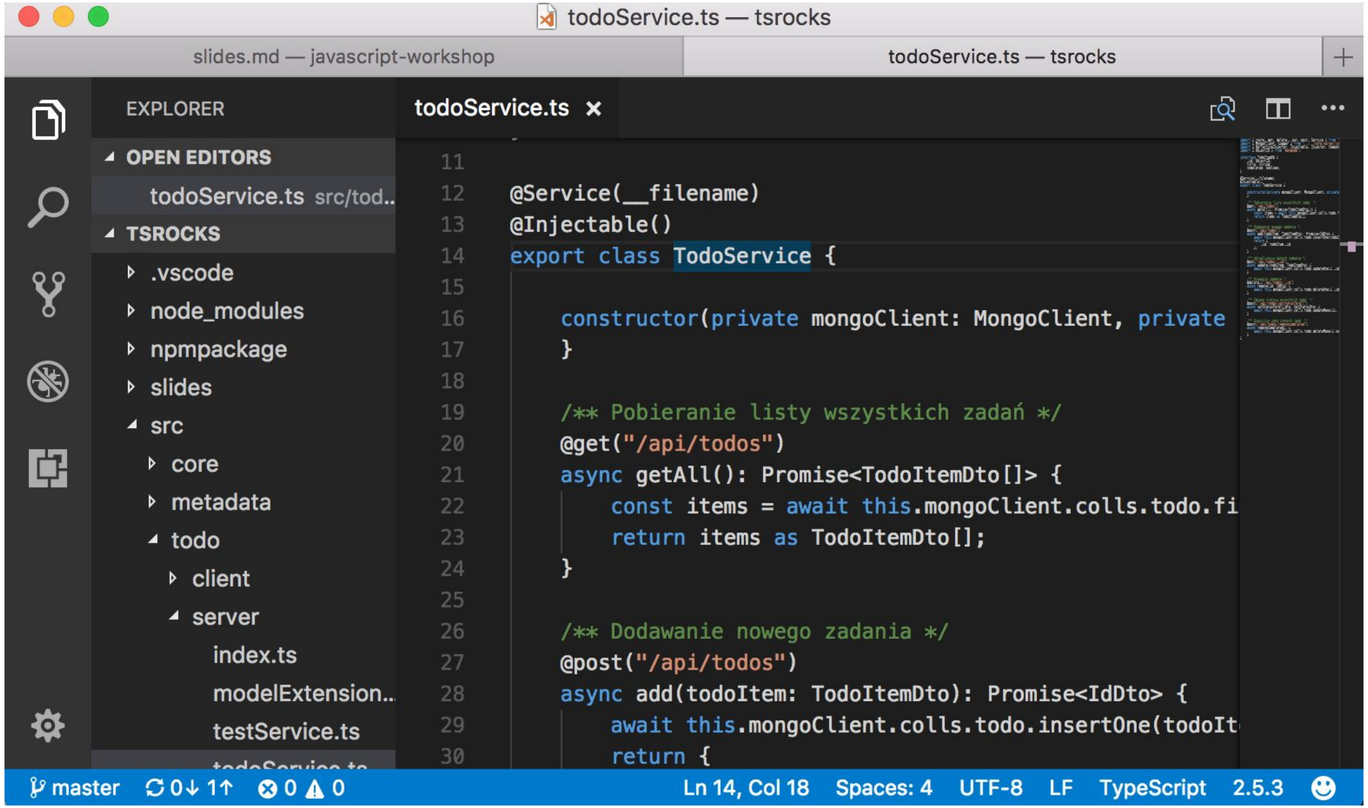
Full stack TypeScript dla programisty .net

Marcin Najder

Agenda

- TypeScript dla programisty .net
  - JS/TS === C#
  - JS/TS >? C#
- Full stack TypeScript
  - DTO
  - Serwisy
  - Proxy
  - Metadane
  - Angular

Visual Studio Code



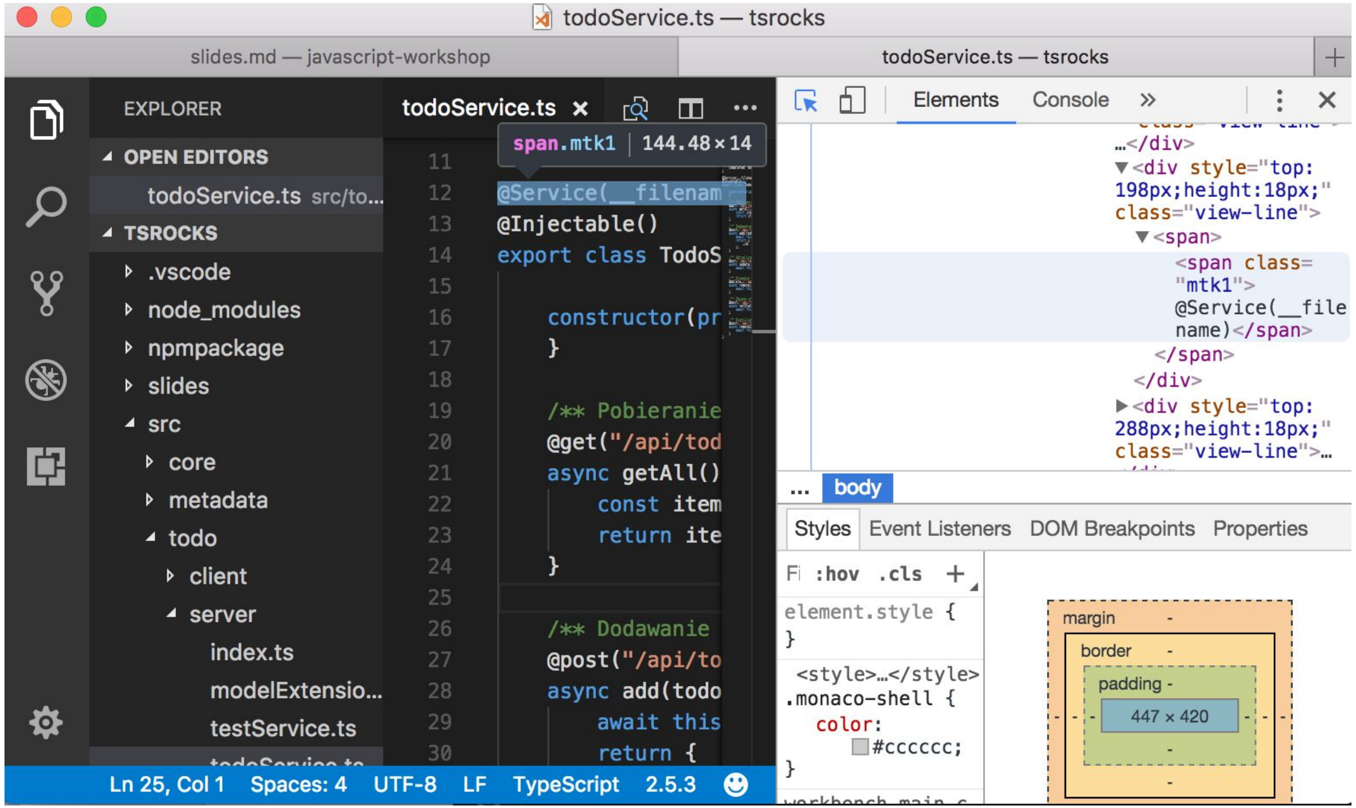
todoService.ts — tsrocks

slides.md — javascript-workshop | todoService.ts — tsrocks

```
@Service(__filename)
@Injectable()
export class TodoService {

    constructor(private mongoClient: MongoClient, private
    }

    /** Pobieranie listy wszystkich zadań */
    @get("/api/todos")
    async getAll(): Promise<TodoItemDto[]> {
        const items = await this.mongoClient.colls.todo.fi
        return items as TodoItemDto[];
    }

    /** Dodawanie nowego zadania */
    @post("/api/todos")
    async add(todoItem: TodoItemDto): Promise<IdDto> {
        await this.mongoClient.colls.todo.insertOne(todoIt
        return {
```

master 0↓ 1↑ 0 0 Ln 14, Col 18 Spaces: 4 UTF-8 LF TypeScript 2.5.3

https://electron.atom.io/



```
@Service(__filenam
@Injectable()
export class TodoS

    constructor(pr
    }

    /** Pobieranie
    @get("/api/tod
    async getAll()
        const item
        return ite
    }

    /** Dodawanie
    @post("/api/to
    async add(todo
        await this
        return {
```

```
…</div>
▼<div style="top: 198px;height:18px;" class="view-line">
  ▼<span>
     <span class="mtk1">@Service(__filename)</span>
   </span>
 </div>
▶<div style="top: 288px;height:18px;" class="view-line">…
```

```
Fi  :hov  .cls  +
element.style {
}
<style>…</style>
.monaco-shell {
  color: ▨#cccccc;
}
```

Dlaczego JavaScript ?

- Aplikacje webowe
  - Angular, react.js, Aurelia, vue.js, ...
- Aplikacje serwerowe
  - Node.js
- Aplikacje mobilne
  - Natywne: React Native, NativeScript
  - WebView: Cordova/PhoneGap, Ionic
  - https://code.janeasystems.com/nodejs-mobile
- Aplikacje desktopowe
  - https://electron.atom.io/
  - https://electron.atom.io/apps/ vs code, atom, slack, github desktop, hyper, gitkraken, …
  - Universal Windows Platform
- Aplikacje IoT

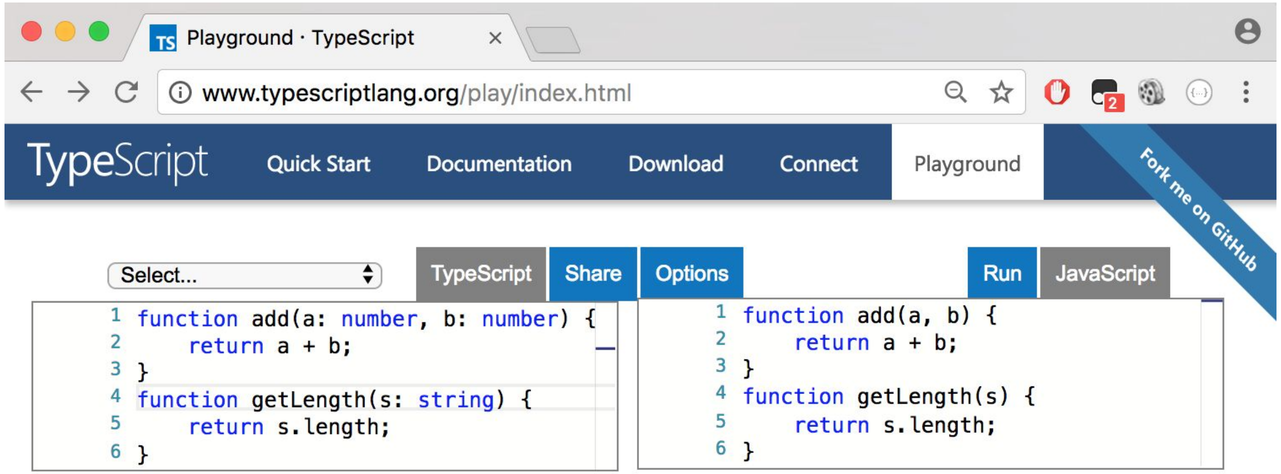
TypeScript dla programisty .net

Na starcie ...

- Większość pokazywanego dzisiaj kodu to czysty JS (nawet nie TS)
- TS nie wymusza na nas podejścia obiektowego

TypeScript

- Premiera 10.2012
- https://channel9.msdn.com/posts/Anders-Hejlsberg-Introducing-TypeScript
- Anders Hejlsberg - Turbo Pascal, Delphi, **C#, TypeScript**
- TypeScript = JS next + typowalność
- https://hackernoon.com/the-first-typescript-demo-905ea095a70f
- The first TypeScript demo



```
function add(a: number, b: number) {
    return a + b;
}
function getLength(s: string) {
    return s.length;
}
```

```
function add(a, b) {
    return a + b;
}
function getLength(s) {
    return s.length;
}
```

Klasy

```
class Point {
    x: number;
    y: number;
    constructor(x: number, y: number) {
        this.x = x;
        this.y = y;
    }
    // constructor(public x: number, y: public number) { }

    reset() {
        this.x = this.y = 0;
    }
    toString() {
        return `x=${this.x} y=${this.y}`;
    }

    static get zero() {
        return new Point(0, 0);
    }
}

var point = new Point(10, 20);
console.log(point.toString());
console.log(Point.zero.toString());
```

Dziedziczenie, typy wyliczeniowe

```
class Point3 extends Point {
    constructor(x: number, y: number, public z: number) {
        super(x, y);
    }

    reset() {
        super.reset();
        this.z = 0;
    }

    toString() {
        return `x=${this.x} y=${this.y} z=${this.z}`;
    }
}


enum Visibility1 { Hidden, Visible }
enum Visibility2 { Hidden = "Hidden", Visible = "Visible" }
```

Typy generyczne

```
function identity<T>(value: T) {
    return value;
}
var text = identity("hej");         // -> string
var one = identity(1);              // -> number

class Pair<T1, T2> {
    constructor(readonly item1: T1, readonly item2: T2) {
    }
    toTuple(): [T1, T2] {
        return [this.item1, this.item2];
    }
}

var pair1 = new Pair<number, string>(1, "one");
var pair2 = new Pair(2, "two");
```

Obiekt funkcji, lambdy

```
function filterItems<T>(items: T[], predicate: (item: T) => boolean) {
    var result: T[] = [];
    for (var item of items) {
        if (predicate(item)) {
            result.push(item);
        }
    }
    return result
}

var result1 = filterItems([1, 2, 3, 4, 5], function (item) {
    return item % 2 === 0;
});
var result2 = filterItems([1, 2, 3, 4, 5], item => item % 2 === 0);


// Delegaty?
type Func0<TResult> = () => TResult;
type Func1<T1, TResult> = (arg1: T1) => TResult;
type Func2<T1, T2, TResult> = (arg1: T1, arg2: T2) => TResult;
// ...
type Action0 = () => void;
type Action1<T1> = (arg1: T1) => void;
```

Iteratory, generatory

```
function* return123() {
    for (var i = 1; i <= 3; ++i) {
        yield i;
    }
}

var iterable = return123();
for (var item of iterable) {
    console.log(item);
}

function* return01231239() {
    yield 0;
    yield* return123();
    yield* return123();
    yield 9;
}
console.log(Array.from(return01231239()));
```

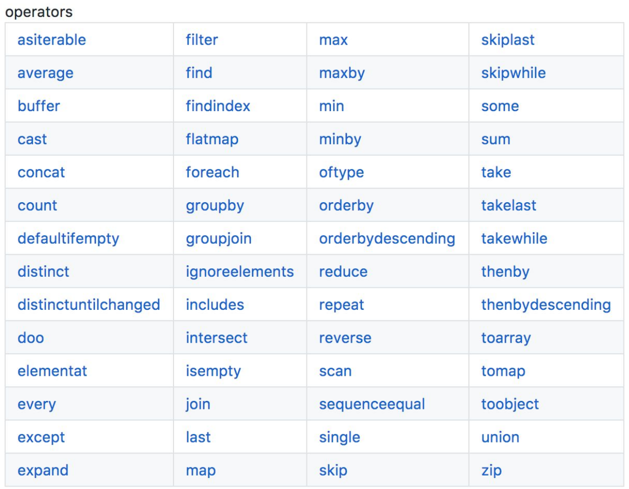
LINQ ???

```
function* range(start: number, count: number) {
    var end = start + count;
    for (var i = start; i < end; i++) {
        yield i;
    }
}
function* filter<T>(source: Iterable<T>, predicate: (item: T) => boolean) {
    for (var item of source) {
        if (predicate(item)) {
            yield item;
        }
    }
}
function* take<T>(source: Iterable<T>, count: number) {
    var counter = count;
    if (counter > 0) {
        for (var item of source) {
            yield item;
            if (--counter === 0) return; // return??
        }
    }
}
var a = range(0, Number.MAX_VALUE);
var b = filter(a, x => x % 2 === 0);
var c = take(b, 5);
[...c];
```

// https://github.com/marcinnajder/powerseq

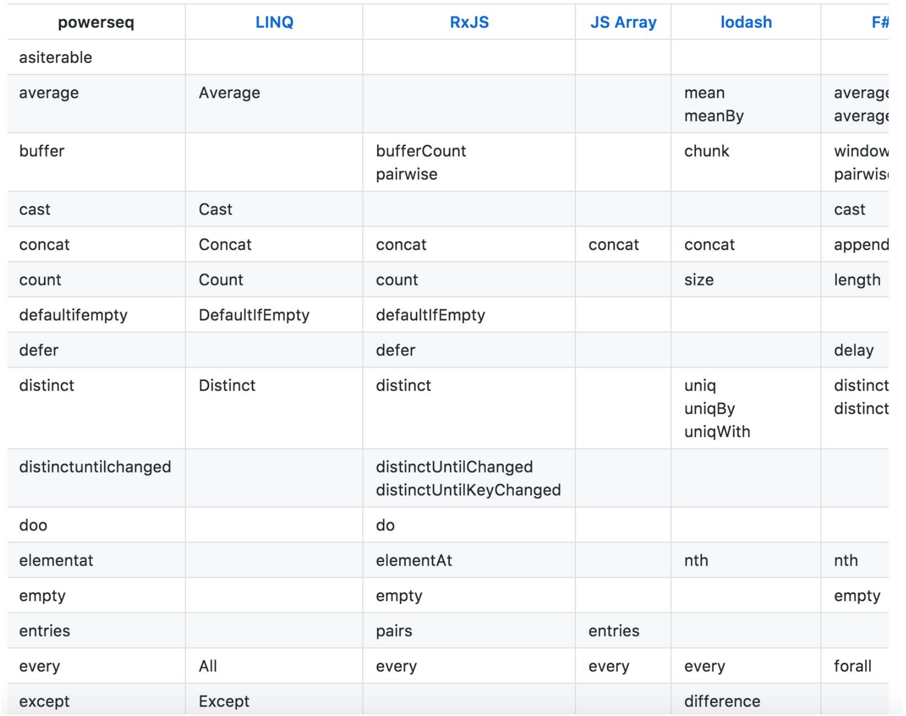
Biblioteka powerseq

```
var { Enumerable } = require("powerseq");

var q = Enumerable
    .range(1, Number.MAX_VALUE)
    .filter(x => x % 2 === 0)
    .take(5);

console.log(q.toarray());
```



operators

asiterable	filter	max	skiplast
average	find	maxby	skipwhile
buffer	findindex	min	some
cast	flatmap	minby	sum
concat	foreach	oftype	take
count	groupby	orderby	takelast
defaultifempty	groupjoin	orderbydescending	takewhile
distinct	ignoreelements	reduce	thenby
distinctuntilchanged	includes	repeat	thenbydescending
doo	intersect	reverse	toarray
elementat	isempty	scan	tomap
every	join	sequenceequal	toobject
except	last	single	union
expand	map	skip	zip



powerseq	LINQ	RxJS	JS Array	lodash	F#
asiterable					
average	Average			mean meanBy	average average
buffer		bufferCount pairwise		chunk	window pairwis
cast	Cast				cast
concat	Concat	concat	concat	concat	append
count	Count	count		size	length
defaultifempty	DefaultIfEmpty	defaultIfEmpty			
defer		defer			delay
distinct	Distinct	distinct		uniq uniqBy uniqWith	distinct distinct
distinctuntilchanged		distinctUntilChanged distinctUntilKeyChanged			
doo		do			
elementat		elementAt		nth	nth
empty		empty			empty
entries		pairs	entries		
every	All	every	every	every	forall
except	Except			difference	

Interfejsy

```
interface EntityBase {
    id: number;
}

interface Repository<T extends EntityBase> {
    getAll(): T[];
    getById(id: number): T;
}

class DatabaseRepository<T extends EntityBase> implements Repository<T>{
    getAll(): T[] {
        throw new Error("Method not implemented.");
    }
    getById(id: number): T {
        throw new Error("Method not implemented.");
    }
}

// tylko typowalnosc tutaj troche inaczej dziala ...
```

Promise

```
function delay(timeout: number) {
    return new Promise<number>(function (resolve, reject) {
        setTimeout(function () {
            resolve(0);
        }, timeout);
    });
}
function getValueAsync<T>(value: T) /*: Promise<T>*/ {
    return delay(1000).then(_ => value);
}

getValueAsync(10)
    .then(n => {
        console.log("n", n);
        return (n * 10).toString();
    })
    .then(s => {
        console.log("s", s);
        return getValueAsync(new Date());
    })
    .then(d => {
        console.log("d", d);
    });
```

async/await

```
async function asyncFunction() /*: Promise<Date>*/ {
    var n = await getValueAsync(10);
    console.log("n", n);

    var d = await getValueAsync(new Date());
    console.log("d", d);

    return d;
}

async function timer() {
    for (var i = 0; i < 3; i++) {
        var dd = await asyncFunction();
        console.log(i, "dd", dd);
    }
    return "TypeScript !!";
}

timer().then(console.log, console.error);
```

Dekoratory

```
class Login {
    @required
    @maxLength(30)
    userName: string;

    @required
    password: string;

    @memoize
    increment(value: number) {
        return value + 1;
    }
}

function memoize(target, key, descriptor) {
    var func = descriptor.value, cache = {};
    descriptor.value = arg => cache[arg] || (cache[arg] = func.call(this, arg));
    return descriptor;
}
```

Ale JavaScript jest …
dynamiczny, skryptowy, funkcyjny

Obiekty w JS

```
var o = {
    name: "marcin",
    age: 25,
    isOk: function () {
        return true;
    }
};

function printO(o) {
    console.log(JSON.stringify(o), o.isOk());
}

printO(o);

delete o.age;
o.isOk = () => false;

printO(o);
```

Obiekty w TS

```
function printO(o: { name: string; age: number; isOk(): boolean; }) {
    console.log(JSON.stringify(o), o.isOk());
}

interface O {
    name: string;
    age: number;
    isOk(): boolean;
}

function printO(o: O) {
    console.log(JSON.stringify(o), o.isOk());
}

type O = { name: string; age: number, isOk(): boolean };
```

Structural vs nominal typing

```
interface Ok {
    isOk: boolean;
}

class Language /* implements Ok*/ {
    constructor(public name: string, public isOk: boolean) {
    }
}

var ok1: Ok = new Language("TypeScript", true);

var js = { name: "JavaScript", isOk: true };
var ok2: Ok = js;


class GoogleLanguage {
    constructor(public name: string, public isOk: boolean, public releaseDate: Date) {
    }
}
var language: Language = new GoogleLanguage("Dart", true, new Date(2011, 9, 10));
```

Intersection Types

```
import * as express from "express";

type MyRequest = express.Request & { userName: string };

var app = express();

app.get('/api/user', (req: MyRequest, res: express.Response) => {
    res.json({ path: req.path, userName: req.userName });
});


type DNS = Date & Number & String; // zawiera wszystkie skladowe wymieniowych typow
```

Union Types

```
type SendRequestOptions = string | {
    method: "get" | "post";
    path: string;
    data?: any;
}

function sendRequest(options: SendRequestOptions) {
    var args: SendRequestOptions;

    if (typeof options === "string") {
        // tutaj options jest typu "string"
        args = { method: "get", path: options };
    } else {
        // tutaj options jest typu "{ method: ..., path: ... }"
        args = options;
    }
    // todo: ....
}

type DNS = Date | Number | String; // wspolne skladowe wymienionych typow
```

Nullable Types

```
interface Person {
    firstName: string;
    lastName: string;
    middleName: string | null;
}

function formatPersonInfo(p: Person) {
    var info = p.firstName.toUpperCase() + " ";

    if (typeof p.middleName === "string") {
        // tutaj p.middleName jest typu "string"
        info += p.middleName.toUpperCase() + " ";
    } else {
        // tutaj p.middleName jest typu "null"
        info += "";
    }
    info += p.lastName.toUpperCase();

    return info;
}
```

C#?

Podejście obiektowe

```
abstract class Shape {
    abstract getArea(): number;
}

class Square extends Shape {
    size: number;
    getArea() { return Math.pow(this.size, 2); }
}

class Rectangle extends Shape {
    width: number;
    height: number;
    getArea() { return this.width * this.height; }
}

class Circle extends Shape {
    radius: number;
    getArea() { return Math.PI * Math.pow(this.radius, 2); }
}

// https://en.wikipedia.org/wiki/Expression_problem
```

Podejście funkcyjne

```
// ./shapes.ts
export interface Square {
    kind: "square";
    size: number;
}
export interface Rectangle {
    kind: "rectangle";
    width: number;
    height: number;
}
export interface Circle {
    kind: "circle";
    radius: number;
}
export type Shape = Square | Rectangle | Circle;
// discriminated unions / tagged unions / algebraic data types

export function getArea(s: Shape) {
    switch (s.kind) {
        case "square": return s.size * s.size;
        case "rectangle": return s.height * s.width;
        case "circle": return Math.PI * s.radius ** 2;
    }
}
// ./main.ts
// import { Shape, getArea } from "./shapes";
```

C#?

Immutability

```
const array1 = [1, 2, 3, 4];
const array2 = [...array1, 10];


const p1: Person = { firstName: "marcin", lastName: "najder", middleName: null };
const p2: Person = { ...p1, middleName: "lukasz" };


const array3: ReadonlyArray<number> = array1;
const p3: Readonly<Person> = p1;
// p3.firstName = "";
// Cannot assign to 'firstName' because it is a constant or a read-only property


const array4: ReadonlyArray<number> = Object.freeze(array1);
const p4: Readonly<Person> = Object.freeze(p1);
```

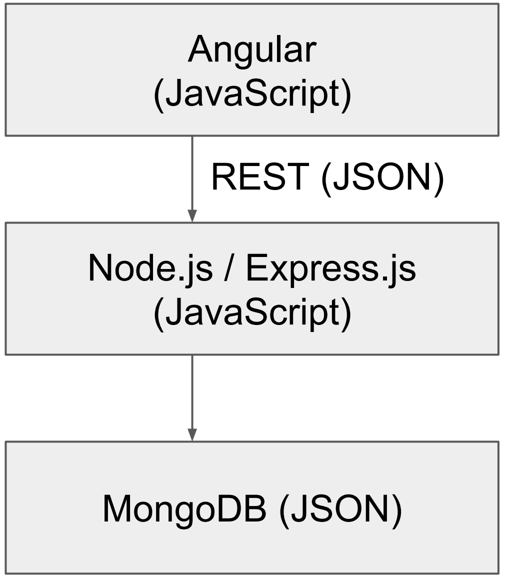
Asynchroniczne iteratory

```
async function* dates() {
    while (true) {
        await delay(1000);
        yield new Date();
    }
}

async function processDates() {
    var datesGenerator = dates();
    for await (var date of datesGenerator) {
        if (date.getSeconds() === 13) {
            return date;
        }
    }
}
```

C#?

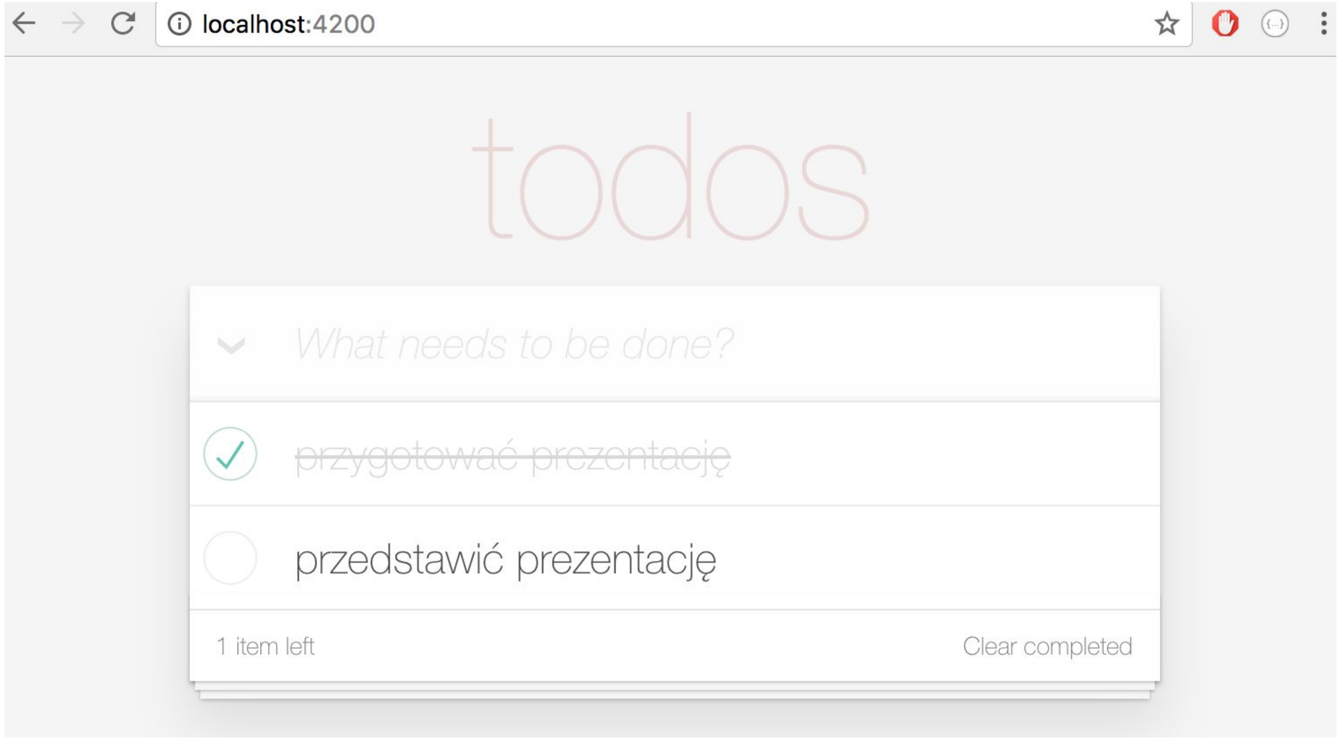
Full stack TypeScript

MEAN - **M**ongoDB, **E**xpress.js, **A**ngular(JS), **N**ode.js



Aplikacja demo

Angular (4) -> Node.js -> MongoDB


localhost:4200

todos

What needs to be done?

~~przygotować prezentację~~

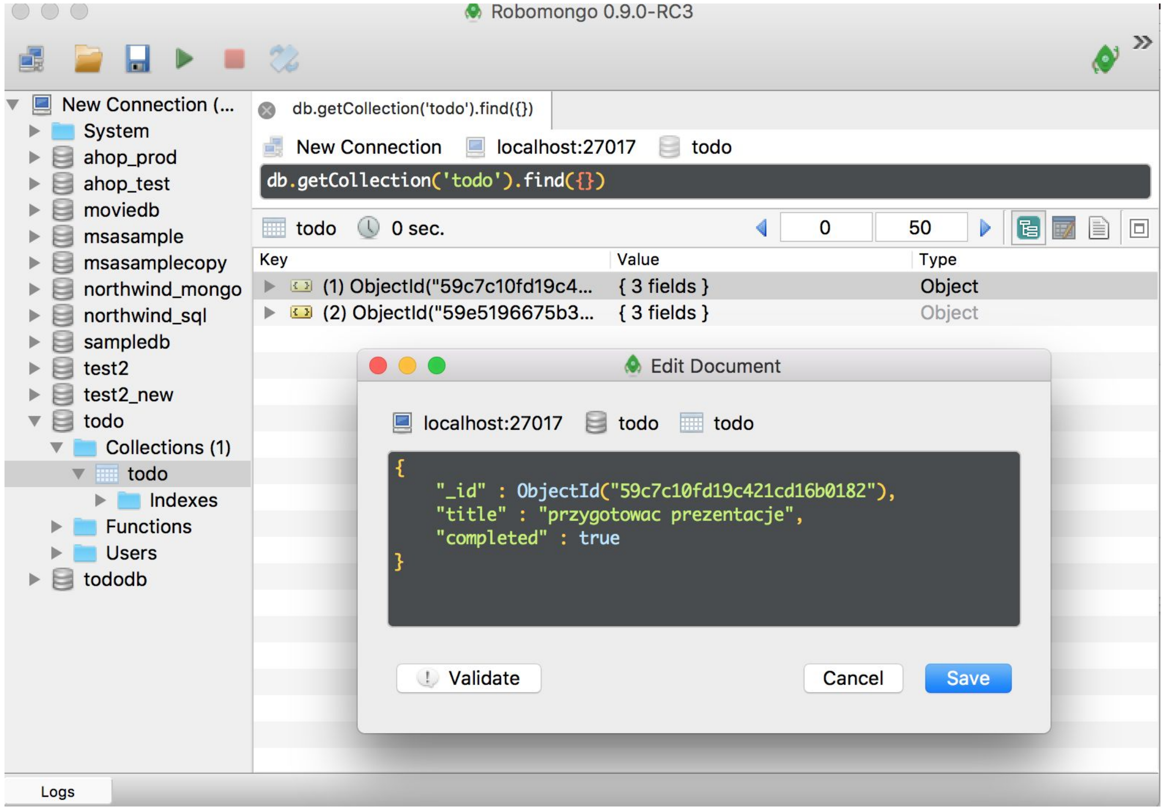
przedstawić prezentację

1 item left

Clear completed

http://todomvc.com/

MongoDB

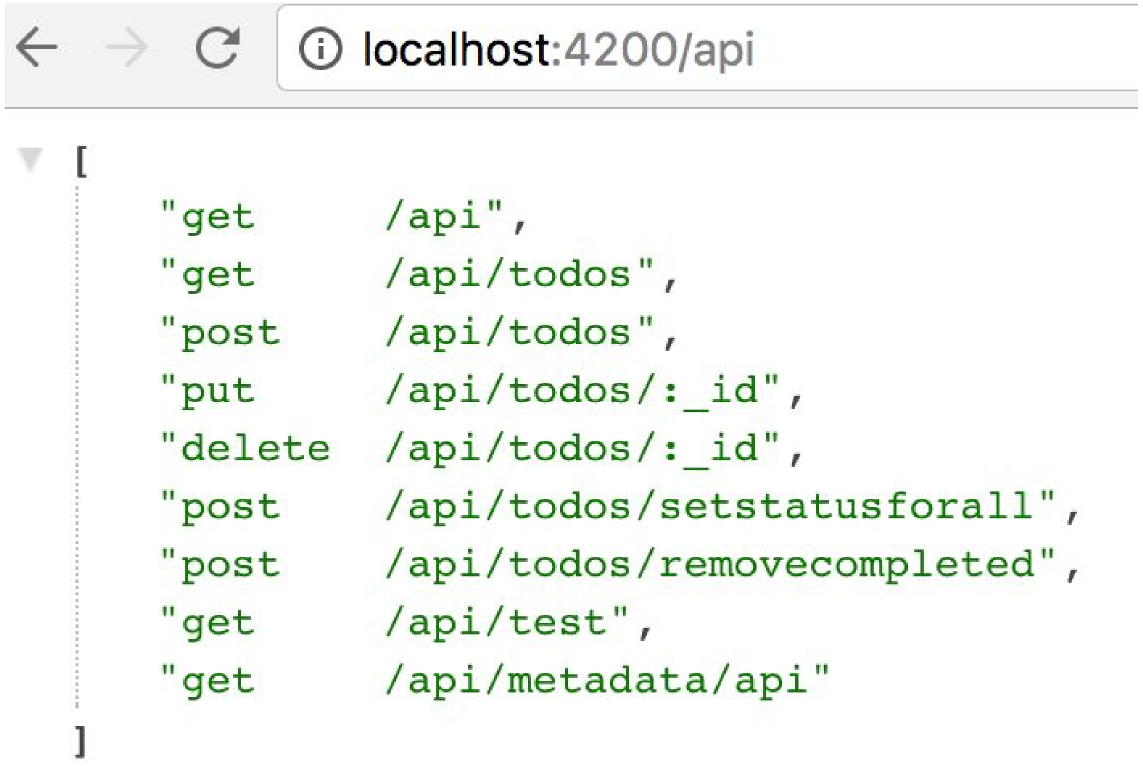


```
db.getCollection('todo').find({})
```

Key	Value	Type
(1) ObjectId("59c7c10fd19c4...	{ 3 fields }	Object
(2) ObjectId("59e5196675b3...	{ 3 fields }	Object

```
{
    "_id" : ObjectId("59c7c10fd19c421cd16b0182"),
    "title" : "przygotowac prezentacje",
    "completed" : true
}
```

REST



```
[
    "get      /api",
    "get      /api/todos",
    "post     /api/todos",
    "put      /api/todos/:_id",
    "delete   /api/todos/:_id",
    "post     /api/todos/setstatusforall",
    "post     /api/todos/removecompleted",
    "get      /api/test",
    "get      /api/metadata/api"
]
```

express.js

```
import * as express from "express";
import { MongoClient } from "mongodb";
import * as bodyParser from "body-parser";

const app = express();
app.use(bodyParser());

app.get('/api/todos', function (req, res) {

    const queryObject = req.query.text ? { title: { $regex: req.query.text } } : {};

    MongoClient.connect("mongodb://localhost:27017/todo", function (err, db) {
        db.collection("todo", function (err, collection) {
            collection.find(queryObject).toArray(function (err, data) {
                res.json(data);
            });
        });
    });

});

app.listen("5634", err => err ? console.error(err) : console.log("Listening on 5634 ... "));
```

express.js

```
app.get('/api/todos', function (req, res) {
    const queryObject = req.query.text ? { title: { $regex: req.query.text } } : {};

    MongoClient.connect("mongodb://localhost:27017/todo", function (err, db) {
        db.collection("todo", function (err, collection) {
            collection.find(queryObject).toArray(function (err, data) {
                res.json(data);
            });
        });
    });
});

// i pojawiaja sie pytania:
// - jaki jest model danych obiektow "request" i "response" ?
// - kto zwaliduje ich schemat ?
// - a moze ktos zrobi konwersje prostych typow string->Date|boolean ?
// - asynchronicznosc ... tak ma wygladac moj kod ? (tutaj nawet nie ma obslugi bledow)
// - jak wywolac te metode "z kodu" ? (np z unit testu)
// - ...
```

DTO - **D**ata **T**ransfer **O**bject

```
/** Obiekt zapytania */
declare interface QueryDto {
    /** Szukana fraza */
    text?: string;
}

/** Zadanie do wykonania */
declare interface TodoItemDto {
    /** Unikalne ID zadania */
    _id?: ObjectID;
    /** Tytuł zadania */
    title: string;
    /** Oznaczenie zakończenia zadania (true - zakończone, false - do zrobienia)*/
    completed: boolean;
}
```

DTO a DRY **d**on't **r**epeat **y**ourself

- Dedykowane miejsce w kodzie JS/JS opisujące model danych DTO
- TypeScript czyli: intellisense, nawigacja po kodzie, błędy kompilacji po zmianie definicji, ...
- Użycie po stronie klienta i serwera
- Definicja schematu obiektu
  - Walidacja obiektu: schemat, typy danych, wymagalność pól
  - Automatyczna konwersja typów danych np.: string -> Date (nie ma daty w JSON) , string->boolean/number/… (np api/todos?completed=true), string -> ObjectID
  - Mapowanie/”przycinanie” obiektów względem schematu
- Generowanie dokumentacji dla kodu, także dla RESTa (np swagger)

Serwisy - implementacja RESTowych adresów

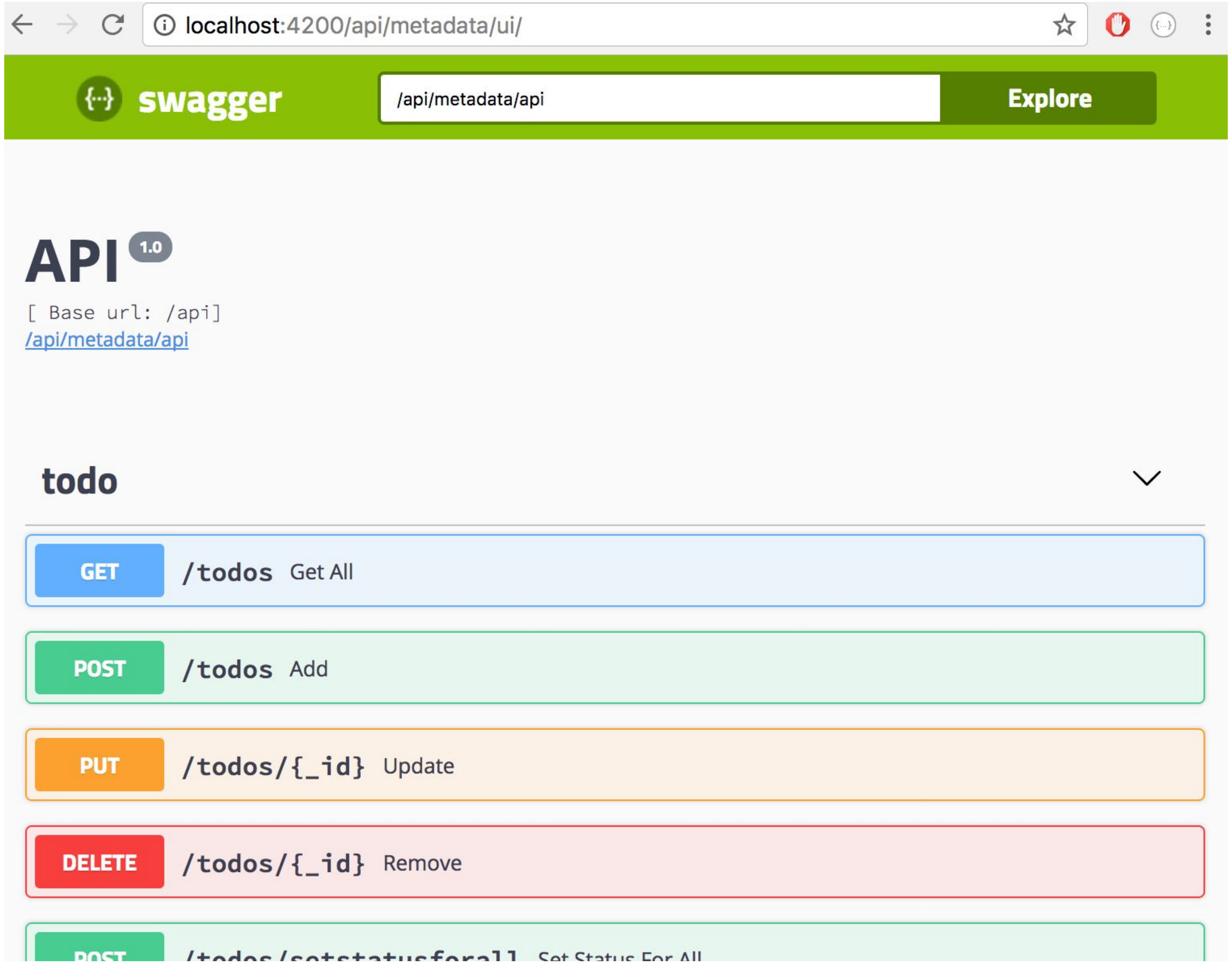
```
@Service(__filename)
@Injectable()
export class TodoService {
    constructor(private mongoClient: MongoClient, private logger: Logger) { }

    /** Pobieranie listy wszystkich zadań */
    @get("/api/todos")
    async getAll(dto: QueryDto): Promise<TodoItemDto[]> {
        const queryObject = typeof dto.text === "undefined" ? {} : { title: { $regex: dto.text } };
        const items = await this.mongoClient.colls.todo.find(queryObject).toArray<TodoItemDb>();
        return items as TodoItemDto[];
    }

    /** Dodawanie nowego zadania */
    @post("/api/todos")
    async add(todoItem: TodoItemDto): Promise<IdDto> {
        const { insertedId } = await this.mongoClient.colls.todo.insertOne(todoItem);
        return {
            _id: insertedId
        };
    }
    // ...
}
```

Serwisy

- Czyste klasy TypeScript, bez zależności do frameworka (np. express.js)
  - Wygoda tworzenia testów jednostkowych
  - Możliwość wywołania logiki serwisów bezpośrednio w kodzie serwera
- “Model binding” - wyliczanie obiektów DTO z żądania HTTP
- DI **d**ependency **i**njection
  - Wykorzystanie kontenera IoC z Angular :)
  - Ten sam mechanizm po stronie klienta i serwera, co szczególnie ważne przy renderowaniu serwerowym (server-side rendering, universal/Isomorphic JavaScript)
- Mechanizm adnotacji
- Asynchroniczność z wykorzystaniem async/await

Automatycznie generowane proxy

```
//src/todo/shared/proxy.generated.ts
declare module "@tsrocks/core/client/proxy" {
    interface Proxy {
        todo: TodoProxy;
    }
}
export class TodoProxy implements IProxy {
    se: ServiceExecutor;
    /** get /api/todos */
    getAll(dto: QueryDto) {
        const p = this.se<TodoItemDto[]>('get', '/api/todos', dto);
        return p;
    }
    /** post /api/todos */
    add(dto: TodoItemDto) {
        const p = this.se<IdDto>('post', '/api/todos', dto);
        // tutaj potencjalnie dodatkowy kod np konwertowanie string->Date
        return p;
    }
    // ...
```


localhost:4200/api/metadata/ui/

swagger

/api/metadata/api

Explore

API 1.0

[Base url: /api]

/api/metadata/api

todo

GET /todos Get All

POST /todos Add

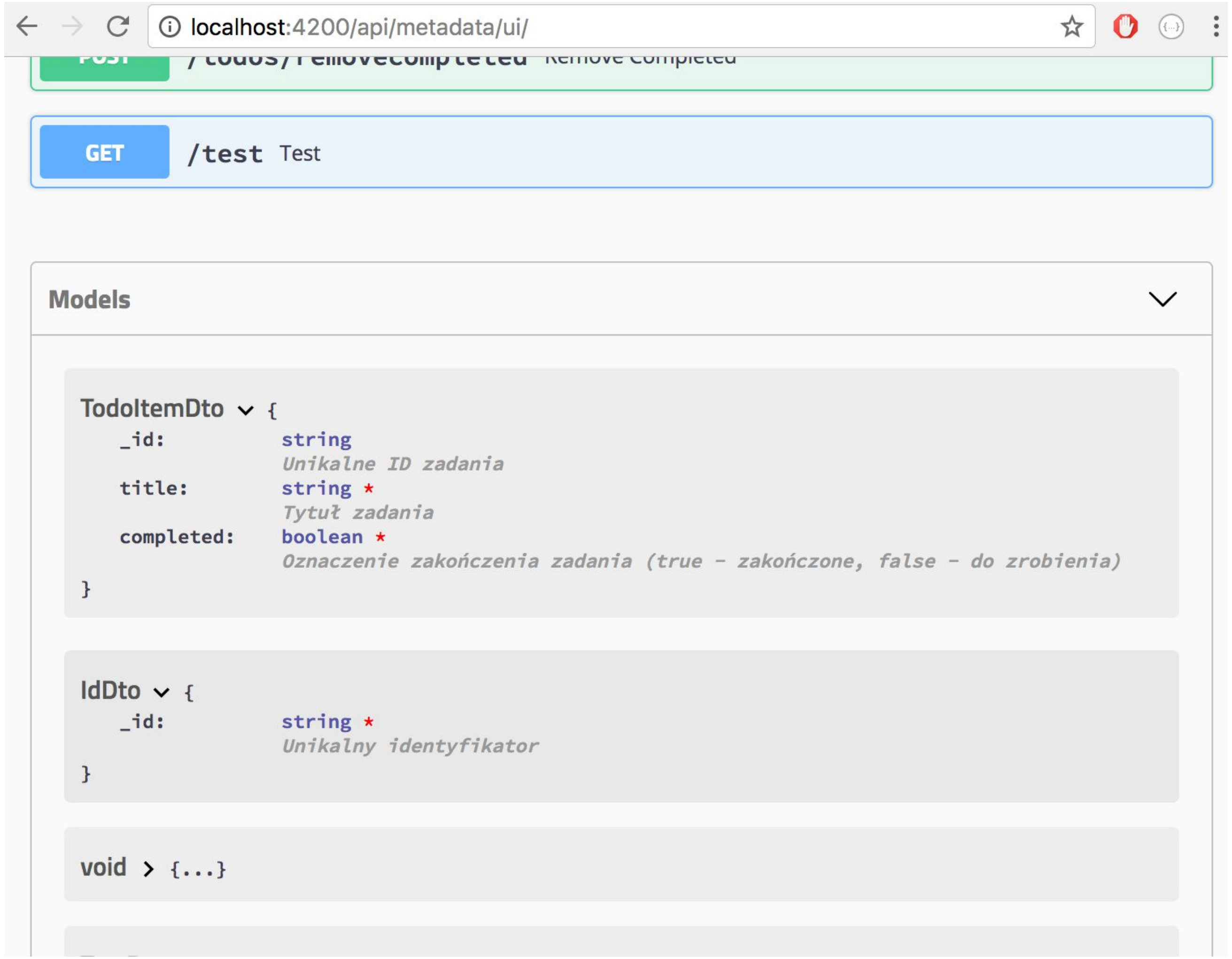
PUT /todos/{_id} Update

DELETE /todos/{_id} Remove

POST /todos/setstatusforall Set Status For All

- http://swagger.io/
- “Standard” opisu RESTowego API
- Narzędzia
    - UI do projektowania API
    - generatory kodu klienta/serwera
    - generatory dokumentacji

Opis modelu danych

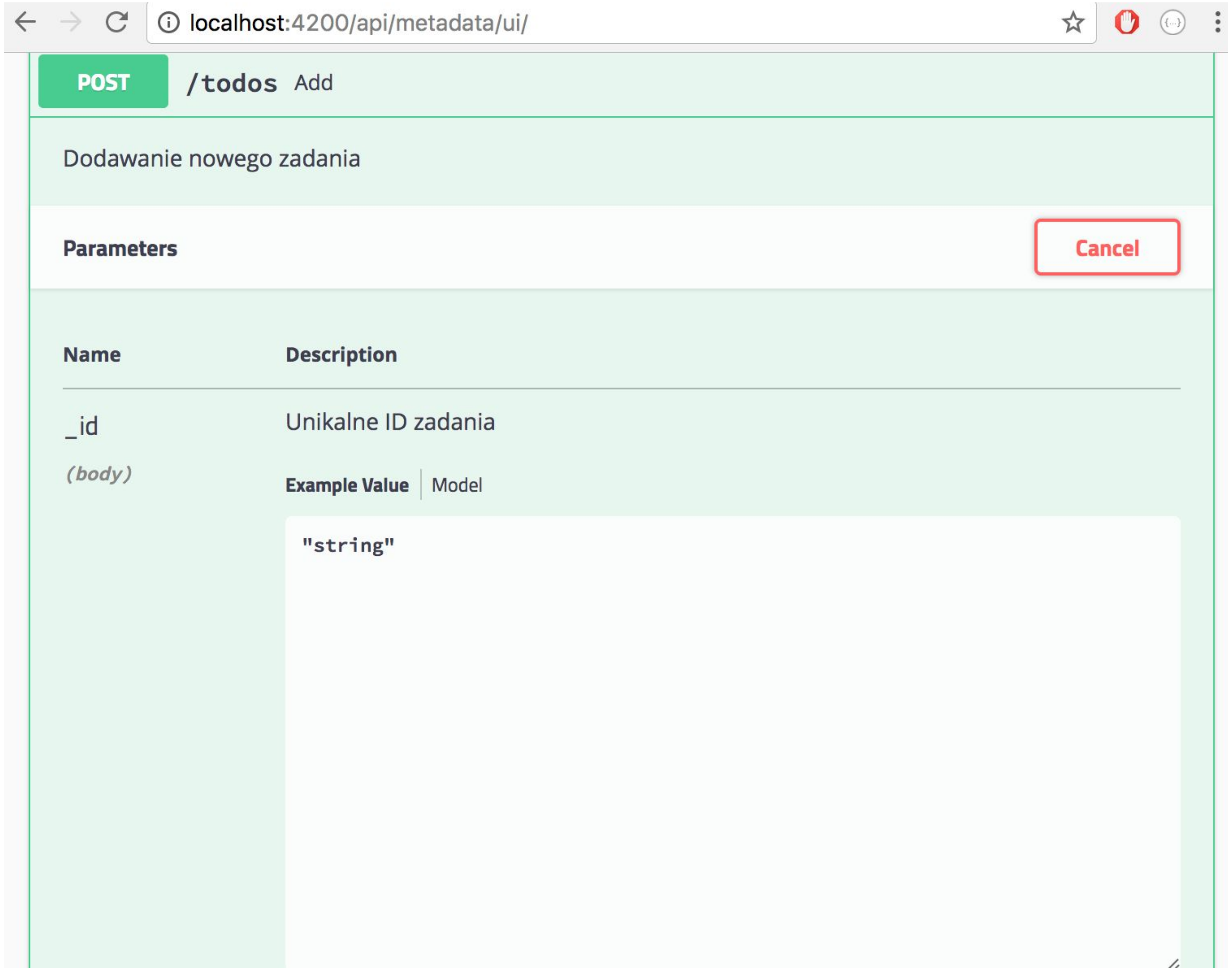

localhost:4200/api/metadata/ui/

GET /test Test

Models

```
TodoItemDto ⌄ {
    _id:          string
                  Unikalne ID zadania
    title:        string *
                  Tytuł zadania
    completed:    boolean *
                  Oznaczenie zakończenia zadania (true - zakończone, false - do zrobienia)
}

IdDto ⌄ {
    _id:          string *
                  Unikalny identyfikator
}

void > {...}
```

Wykonania żądania HTTP



localhost:4200/api/metadata/ui/

POST /todos Add

Dodawanie nowego zadania

Parameters Cancel

Name	Description
_id *(body)*	Unikalne ID zadania

Example Value | Model

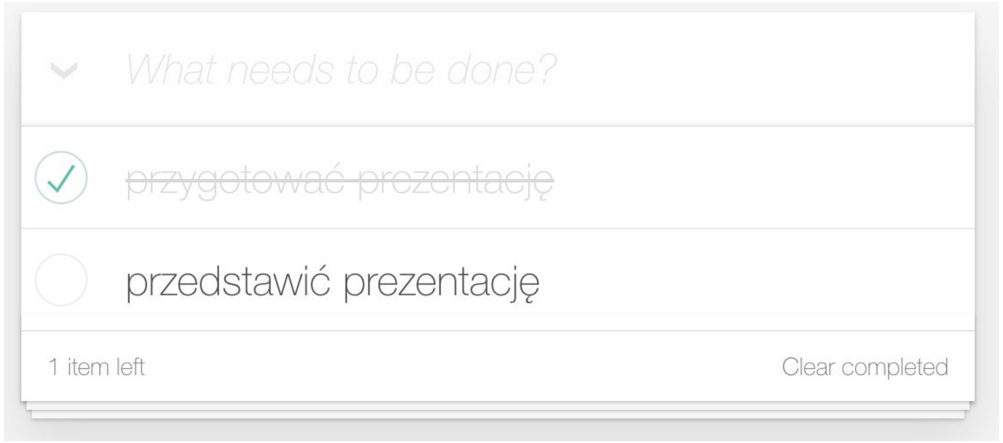
```
"string"
```

Angular

Czym jest Angular?

- Framework dla aplikacji SPA (Single Page Application)
- Wydany 09.2016 (następca AngularJS z 06.2012)
- Tworzony przez Google, open source, napisany w TypeScript
- Platforma do tworzenia aplikacji (framework vs biblioteka)
    - https://www.npmjs.com/~angular **gotowe moduły zrobione przez zespół Angulara**: żądania HTTP, ruter, animacje, formularze, tłumaczenie aplikacji, migracja z AngularJS, flex-layout, webowe aplikacje mobilne, PWA progressive web apps, service worker, web worker …
    - Angular universal - renderowanie HTML po stronie serwera
    - https://material.angular.io/ zestaw gotowych komponentów UI, ale także biblioteka do budowania własnych
    - **Angular CLI** - development, ale także budowanie zoptymalizowanej paczki wdrożeniowej
    - Angular language service - rozszerzenia dla narzędzi/edytorów
- Aplikacje mobilne: Ionic, NativeScript

Komponenty - szablon


What needs to be done?

~~przygotować prezentację~~

przedstawić prezentację

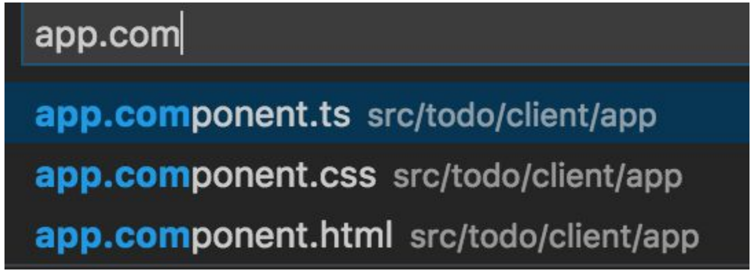
1 item left

Clear completed

```
<!-- app.component.html -->
<section class="todoapp">
  <todo-header (newTodoAdded)="newTodo($event)"></todo-header>
  <section class="main" *ngIf="todos.length > 0">
    <input type="checkbox" [checked]="allCompleted" (click)="setAllTo(toggleall.checked)"
*ngIf="todos.length" … >
    <ul>
      <li *ngFor="let todo of todos" [class.completed]="todo.completed" [class.editing]="todo.editing">
        <div>
         …
        </div>
      </li>
    </ul>
  </section>
  <todo-footer [todos]="todos" (removeCompleted)="removeCompleted()"></todo-footer>
</section>
```

Komponenty

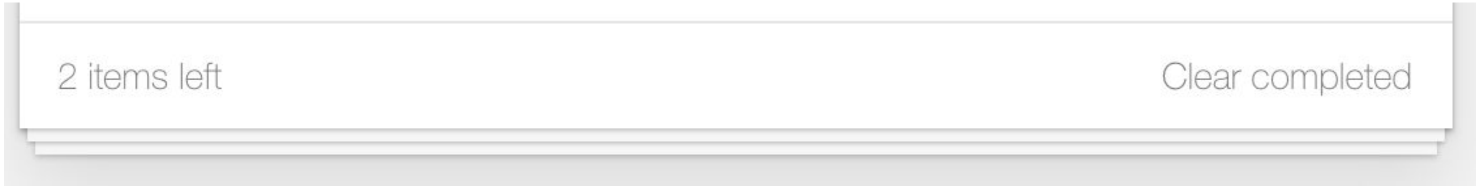
```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  todos: TodoItem[] = [];

  constructor(private proxy: Proxy)  { }

  async ngOnInit() {
    await this.refresh();
  }

  private async refresh() {
    this.todos = await this.proxy.todo.getAll();
  }

  async newTodo(text: string) {
    const todo: TodoItemDto = {
      title: text,
      completed: false
    };
    const idDto = await this.proxy.todo.add(todo);
    todo._id = idDto._id;
    this.todos.push(todo);
  }
}
```



```
app.com
app.component.ts src/todo/client/app
app.component.css src/todo/client/app
app.component.html src/todo/client/app
```

todo-footer - szablon

```
<!-- footer.component.html -->
<footer  *ngIf="allItems > 0">
  <span><strong>{{remainingItems}}</strong> {{remainingItems == 1 ? 'item' : 'items'}} left</span>
  <button *ngIf="completedItems  > 0" (click)="removeCompletedItems()">Clear completed</button>
</footer>
```



todo-footer

```
// footer.component.ts
@Component({
  selector: 'todo-footer',
  templateUrl: './footer.component.html',
  styleUrls: ['./footer.component.css']
})
export class FooterComponent {
  @Input() todos: TodoItemDto[] = [];
  @Output() removeCompleted = new EventEmitter<undefined>();

  get allItems() { return this.todos.length; }
  get remainingItems() { return this.todos.filter(t => !t.completed).length; }
  get completedItems() { return this.todos.filter(t => t.completed).length; }

  removeCompletedItems() {
    this.removeCompleted.emit();
  }
}
```

Koniec

Dziękuję za uwagę !