MINIMAL API – .net 6

OBSAH

[Rozprava 2](#_Toc129774197)

[API dnes 2](#_Toc129774198)

[Zvyky 2](#_Toc129774199)

[Jak by to melo byt 2](#_Toc129774200)

[Specification first 2](#_Toc129774201)

[Výhody 2](#_Toc129774202)

[Open API Standardizace (Swagger) 2](#_Toc129774203)

[Vytvoření specifikace 2](#_Toc129774204)

[Specifikace ve Wordu 2](#_Toc129774205)

[Stoplight studio 3](#_Toc129774206)

[DEMO - SPECIFIKACE 3](#_Toc129774207)

[Specifikace - Word 3](#_Toc129774208)

[GITHUB 3](#_Toc129774209)

[Stoplight studio (STLS) 3](#_Toc129774210)

[Tvorba API 3](#_Toc129774211)

[POSTMAN 3](#_Toc129774212)

[Import 4](#_Toc129774213)

[REDOC 4](#_Toc129774214)

[Mock server 4](#_Toc129774215)

[PRISM 4](#_Toc129774216)

[Stopligh studio 4](#_Toc129774217)

[DEMO – MiniAPI – POC 4](#_Toc129774218)

[MiniApi v .net 6 – struktura, mechanismus 4](#_Toc129774219)

[CancellatonToken 4](#_Toc129774220)

[Mediator 4](#_Toc129774221)

[Caching 5](#_Toc129774222)

[Validace 5](#_Toc129774223)

[FluentValidation 5](#_Toc129774224)

[Exceptions 5](#_Toc129774225)

[RequestId 5](#_Toc129774226)

[HealthCheck 5](#_Toc129774227)

# Rozprava

Popis tech dvou dilu, o cem to dnes je

* O cem byl prvni, specifikace + api, vsechny nastroje, c# 6,
* O cem byl druhy – api mini, grupovani, c# 7

# API dnes

Zvyky

* Postavit API
* Z API vygenerovat dokumentaci

Jak by to melo byt

* API si nejdříve specifikujeme (word)
* Využijeme standardizované specifikace k vygenerování MOCKů (stoplight studio)
* Specifikace k vygenerování dokumentace
* Validace samotného API
* Na základě specifikace se provádí implementace

# Specification first

Myšlenkový postup, shoda sw architektu, pristup, kdy je nahlizeno na API jako samostatný produkt, samostatne rozhrani pro vyvojare, rozhradni pro vyvojare navrzene, resime potreby vyvojari

## Výhody

* Jediný bod pravdy (pokryte standardem), veskera pravidla, endpointy
* Rychlejší něž vývoj
* Díky specifikaci lze paralelně vyvíjet BE i FE
* Každý specifikaci rozumí
* Lze ji použít pro automatizaci, mocků, dokumentaci

## Open API Standardizace (Swagger)

* Swagger do verze 2, od verze 3.0 OpenAPI (unor 2021)
* Podpora u většiny moderních nástrojů
* Popis API pomocí yaml nebo json, moderni nastroje preferuji yaml

## Vytvoření specifikace

### Specifikace ve Wordu

1. Ideální je začít nějakou word šablonou
2. naprototypovat si budoucí API specifikace
3. šablona by měla co nejvíce reflektovat OpenApi specifikace a co nejvíce se jí přiblížit
4. být stále co nejvíce čitelná pro uživatele
5. jakmile bude hotová, vygenerovat specifikaci k příslušném nástroji

#### Výhody Wordu

* Mnohem rychlejší, než rovnou psát v nějakém komplexním nástroji
* Pro porady nebo meetingy se zákazníky je to srozumitelnější

### Stoplight studio

Jedná se o nejkomplexnějsí a nejpraktičtějsí nástroj pro tvorbu OpenApi specifikace

* Ondemand aplikace
* online služna
* propojeni s githubem, kde mame ulozenou samotnou specifikaci

# DEMO - SPECIFIKACE

## Specifikace - Word

Krok č.1 – co nejvíc reflektovat OpenApi specifikaci

* https://swagger.io/specification/
* Začíná úvodními společnými vlastnostmi
* Podporovaný formát – př. appliacation.json
* Deklarování stavových kodu
* Formátování – pravidla, pokrýt vešker hodnoty standardy
* Path – endpointy – popis
* Odkazování – ctrl+k
* + povinne, o optional

## GITHUB

* Před prací se stoplightstudiem si založme GtHub repo
* STLS vytvoří adresář **reference**
* Sem se budou ukladat yaml api file
* StopLight studio se umí na github napojit

## Stoplight studio (STLS)

* Napojení na repositář
* Ukazka pripravene specifikace
* Prepnout na code – ukazaka yaml, specfikace API

### Tvorba API

* Modely
  + - komponent schema z wordu - popis
* Responses
  + - Na zaklade modelu lze definovat respo, + code, reference na model
* Paths
  + - url, metoda, parametry, komentare, nazev operace, navratovy typ – reference, 400,404,500 – nastaveni referenci
* Commit na git
  + - Msg, push, adresar reference, yaml, raw – surova data

## POSTMAN

* Díky yaml můžeme jednoduše vygenerovat kolekci v postmanu

### Import

* Import
* Link
* Název yaml kolekce – url na raw na githubu
* Check – Generate collection – import
* Automaticky se vytvori API

## REDOC

* Slouží pro vygenerování dokumentace, nejrozšířenější nástroj pro dokumentaci
* HTML soubor (default) – prida se JavaScript s linkem na REDOC + odkaz na GitHUB s yaml filem
* Spustit v browser: je vygenerovaná dokumentace na základě specifikace ze StopL st.

## Mock server

### PRISM

* Nástroj pro vytvoření lokálního api-serveru na základě opět yaml dokumentace
* Spustí na url server s naším API včetně dat
* Za pomocí Postmana lze tak ihned provolávat API a FE může vyvíjet nezávisle na BE
* Podpora jen pro linux, na win jsem nenasel

### Stopligh studio

* STLS nutné zapnout v konfiguraci podporu test a mock serveru
* Settings / Docs Settings / Enable Try It
* Settings / Docs Settings / Show Mock Servers

# DEMO – MiniAPI – POC

## MiniApi v .net 6 – struktura, mechanismus

* Minimal hosting = sloučení program.cs + statrup.cs
* Tvorba endpointu, app.MapPost atd.
* Nepouziva MVC mechanismu
  + Slozita pipeline
  + Je rozsahly, velke mnozstvi sluzeb
  + Pomalejsi
* Extension metody, samostatny soubor
* Popis parametru v metodach
* Extension metoda with – popis , tagy, nazvy, cors policy, resp.kody

## CancellatonToken

* Vysvetleni k cemu je
* Priklad s IF

## Mediator

* Prakticky ve vsech api, obliene a rozsiritelne reseni
* Pro kazdy http pozadavek vytvari novou tridu a pro kazdou tridu handler
* Na zaklade In a Out dokaze vyhledat Handlery
* Implementace
  + Csproj – pridat nuget balicek
  + Program.cs – AddMediatr + assembly, kde jsou vsechny handlery
* Vyhody:
  + Pro vyreseni http pozadavku jsou pouzity pouze zavislosti, ktere jsou potreba, nic vic, puvodni MVC – mnoho metod v conrolleru, mnoho zavislosti atd.

### Caching

* Vytvorit rozhrani Icachable (key, time)
* Vytvorit cacheBehavior
  + Definice podminky, pro jaky request bude spustena tato funkcionalita
  + Dle typi icachable a ireqiest

## Validace

* Minimal API neumi validace, nicmene MVC stejne umi malo, takze jsme odkazani na rucni validace

### FluentValidation

* Nuget balicek (csproj, program.cs)
* Program.cs – bud zadat rucne, kde jsou, nebo pres pretizene metody mu dat assembly a on to proscannuje a zaregistruje sam
* 2 typy validaci
  + Get – primitivni typy – moznost jako balicek a vystavit pro ostatni
  + Post – zapojeni DB check – privatni
* Popis jak fluentValidation funguje
  + ScopeSluzba – registrace v proram.cs
  + V prubehu pozadavku se plni chybami
  + Metoda zda pokracovat v kodu ci vyhodit chybu

## Exceptions

* Samostatny adresar s vyjimkami
* ApiExceptionMiddleware – jedno jedine misto, kde chytame vyjimky
* Dle chyby nastavime status code

## RequestId

* Pridat do hlavicky v response
* Z duvody debugingu, zrychluje hledani v lozich

## HealthCheck

* Csproj balicky
* Program.cs
  + Pridat service
  + MapHealthChecks – pridat url, na ktere healtCheck bude naslouchat