Česká zemědělská univerzita v Praze

Technická fakulta

Katedra elektrotechniky a automatizace



Diplomová práce

Návrh a realizace kontrolního systému na WiFi síti

Martin Novák

© 2024 ČZU v Praze!!!

Místo tohoto textu vložte PŘEDNÍ stranu zadání práce, které si můžete vyexportovat do PDF v IS.CZU.cz, pokud již máte schválené zadání i děkanem TF.

!!!

!!!

Místo tohoto textu vložte ZADNÍ stranu zadání práce, které si můžete vyexportovat do PDF v IS.CZU.cz, pokud již máte schválené zadání i děkanem TF.

V případě, že Vaše zadání je na více než 2 strany, vložte i další strany.

!!!

**Čestné prohlášení**

Prohlašuji, že svou diplomovou práci "Návrh a realizace kontrolního systému na WiFi síti" jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne datum odevzdání \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Poděkování**

Rád(a) bych touto cestou poděkoval(a) jméno vedoucího, případně dalších osob, a informace, za co děkujete.

Návrh a realizace kontrolního systému na WiFi síti

**Abstrakt**

Souhrn práce (cca 15 řádek textu).

**Klíčová slova:** klíčová slova (cca 10)

**Design and implementation of a control system on a WiFi network**

**Abstract**

Anglický překlad českého souhrnu

**Keywords**: klíčová slova anglicky

**Obsah**

[1 Úvod 1](#_Toc160228439)

[2 Cíl práce a metodika 2](#_Toc160228440)

[2.1 Cíl práce 2](#_Toc160228441)

[2.2 Metodika 2](#_Toc160228442)

[3 ARM 3](#_Toc160228443)

[3.1 ESP 8266 3](#_Toc160228444)

[4 Protokoly 3](#_Toc160228445)

[4.1 TCP/IP 3](#_Toc160228446)

[4.2 HTTP 3](#_Toc160228447)

[4.3 JSON 3](#_Toc160228448)

[5 Návrhové vzory 3](#_Toc160228449)

[5.1 Dependency injection 3](#_Toc160228450)

[5.2 DTO 3](#_Toc160228451)

[5.3 MVVM 3](#_Toc160228452)

[6 Hlavní uzel 3](#_Toc160228453)

[6.1 Komunikační vrstva 3](#_Toc160228454)

[6.2 Logická vrstva 4](#_Toc160228455)

[6.3 Uživatelské rozhraní 4](#_Toc160228456)

[7 Uzly 4](#_Toc160228457)

[7.1 Uzel 1 4](#_Toc160228458)

[7.2 Uzel 2 4](#_Toc160228459)

[7.3 Uzel 3 4](#_Toc160228460)

[8 Výsledky a diskuse 5](#_Toc160228461)

[8.1 Podkapitola úroveň 2 5](#_Toc160228462)

[8.1.1 Podkapitola úroveň 3 5](#_Toc160228463)

[8.1.2 Podkapitola úroveň 3 5](#_Toc160228464)

[8.2 Podkapitola úroveň 2 5](#_Toc160228465)

[9 Závěr 6](#_Toc160228466)

[10 Seznam použitých zdrojů i](#_Toc160228467)

[11 Přílohy ii](#_Toc160228468)

**Seznam obrázků**

Odkazovaný seznam obrázků

**Seznam tabulek**

Odkazovaný seznam tabulek

**Seznam použitých zkratek**

Soupis a definování zkratek (vyskytuje-li se jich v textu velké množství)

# Úvod

Text text text text text text text text text text text text text text text text text text text text text text text.

# Cíl práce a metodika

## Cíl práce

Text…

## Metodika

Text…

# ARM

ARM (Advanced RISC Machine) je

## ESP 8266

Text…

# Protokoly

Text

## TCP/IP

text

## HTTP

Text

## JSON

\*tohle není protokol, takže budu muset vymyslet kam s ním

# Návrhové a architektonické vzory

Návrhové a architektonické vzory jsou léty ověřené techniky pro řešení opakujících se problémů v objektově orientovaném programování. Nejedná se o konkrétní kód, ale jen o koncept. Z tohoto důvodu nejsou svázány s konkrétní technologií a je tak možné je použít v téměř libovolném jazyce. Výhodou takto pojmenovaných a popsaných postupů je, že je zná většina vývojářů po celém světě a při komunikaci stačí říci jaký vzor použít, bez nutnosti vysvětlovat detaily. Tyto dvě skupiny se od sebe liší oblastí, kterou pokrývají. Návrhové vzory se zabývají chováním jedné třídy, nebo její komunikaci s ostatními. Oproti tomu Architektonické vzory určují sktrukturu celého projektu a mají přímý vliv na jeho modularitu a škálovatelnost. [1–3]

## Zapouzdření

Tímto pojmem je obvykle myšlen jeden za základních pilířů objektově orientovaného programování, kdy třída skryje své hodnoty a metody používané pro vnitřní fungování a ostatním přístupní jen ty potřebné ke komunikaci. Tento přístup také pomáhá zajistit konzistenci, protože stav objektu může být upraven pouze zamýšleným způsobem. Toto lze přenést i do většího měřítka, kdy je aplikace rozdělena na více zapouzdřených částí. Aby ostatní části mohli komunikovat nepotřebují znát vnitřní fungování, ale pouze rozhraní.[4]

## N-vrstvá architektura

Pro složitější aplikace, nebo tam, kde se očekává potřeba měnit některé celky, se často na základě pokrývané oblasti rozděluje aplikace na části označované jako vrstvy. Obvykle se každá vrstva nachází ve vlastním projektu. Hlavní výhodou je přehledná struktura, ve které se snáze hledá. V kombinace se zapouzdřením také zvyšuje modularitu a bezpečnost. Jelikož okolní vrstvy vidí pouze rozhraní, a nikoliv konkrétní implementaci je snadné vrstvu nahradit jinou bez ovlivnění ostatních. Komunikace je obvykle omezena na vrstvy o jednu pod a nad čili případný útočník nemůže z nejvyšší vrstvy přistupovat přímo k nejnižší. Rozdělení vrstev sebou však nese komplikaci v podobě komunikace mezi nimi.[4, 5]

Nejběžnější je třívrstvá architektura. Nejvyšší vrstva komunikuje s uživatelem a podle typu aplikace se jedná o uživatelské rozhraní, nebo v případě API o endpointy. Prostřední a nejdůležitější vrstvou je business logika, která zpracovává požadavky od uživatele. Poslední vrstva se stará o přístup k datům. Tím může být například zápis do databáze, nebo komunikace s jiným systémem.[4]

## Dependency injection

Dependency injection je technika, která snižuje závislost třídy na jiné. Toto umožňuje aplikaci být více modulární, lépe testovatelná a snáze upravitelná.[6]

Pokud má třída například zpracovat data a výsledek uložit do databáze, při klasickém přístupu je pevně svázána s konkrétním databázovým systémem. V horším případě obsahuje všechen kód, čímž porušuje Single responsibility principle (S ze SOLID)[7]. V lepším případě je práce s databází umístěna do vlastní třídy, ale její instance je součástí objektu s logikou, který je zodpovědný za jeho správu. Oba tyto případy komplikují přechod z jednoho typu databáze na jiný a testování je velice obtížné, protože kód očekává připojení k funkční databázi.[6]

Aby se těmto problémům předešlo, je instance této pomocné třídy, která je obvykle označována jako služba, předávána zvenčí. Nyní za správu služby není zodpovědný objekt s logikou, ale Injector. Dále třída většinou není závislá na konkrétní třídě, ale na rozhraní definující metody, které je možné zavolat. Díky této abstrakci je možné snadno změnit implementaci. Mimo jiné je takto umožněno místo skutečné implementace použít testovací třídu, která pouze simuluje volání databáze. Služba je nejčastěji vkládána pomocí konstruktoru, ale může být také použita metoda.[6]

Pro drobné projekty může jako injector sloužit prosté zavolání konstruktoru z kódu[6]. Ve většině případů je použit framework, který automaticky řeší vytváření a předávání potřebných instancí. Může se jednat o knihovnu třetí strany, nebo v některých případech přímo o systémovou knihovnu. Od verzí *.NET Core 1.0* a *.NET Framework 4.5* mezi tyto jazyky patří také C#[8]. V závislosti na typu projektu je knihovna již importována, nebo je třeba dodat příslušný NuGet. Při přidávání služby do seznamu je možné definovat životnost instance. První možností je *Transient*, který je při každém zavolání vytvořen nový. Druhou možností je *Singleton*, jehož instance je vytvořena jen jednou. Poslední je *Scoped* využívaný v ASP.NET pro situace, kde je potřeba aby každé zavolání API mělo vlastní instanci. Od .NET 8.0 je přidán atribut *FromKeyedServices* umožňující zaregistrovat více implementace jednoho rozhraní odlišených klíčem a zvolit implementaci podle aktuální potřeby.[9]

## Data Transfer Object (DTO)

Data transfer Object je instance třídy sloužící k přenosu dat mezi systémy. Použití speciálních objektů umožňuje skrýt hodnoty používané k vnitřní funkci jedné strany, ale pro druhou stranu zbytečných nebo jejichž přenos by mohl být bezpečnostní hrozbou. Současně je takto snížen objem dat, který je nutné přenášet. Další výhodou je možnost naráz přenést údaje nacházející se na více místech a uspořádat je do vhodné struktury. Tyto objekty slouží k serializaci a deserializaci a neměli by obsahovat žádnou logiku.[10, 11]

## MVVM

Pro jednodušší vývoj a testování uživatelských rozhraní se využívají návrhové vzory MVC, MVP a MVVM. Všechny tři od sebe oddělují data, vzhled a logiku, čímž usnadňují udržení struktury a umožňují modulárnost aplikace. Liší se v datových tocích a závislostech jedné části na ostatních.[12]

Nejstarším z těchto návrhových vzorů je MVC (Model-View-Controller). Model obsahuje aplikační data a je zodpovědný za komunikaci s databází, serverem, či jinou externí částí aplikace. View má na starosti zobrazování dat z modelu uživateli. Controller reaguje na uživatelské akce a dává modelu a view pokyny k aktualizaci. Jak je vidět na Obr. 1 jednotlivé části jsou úzce provázány, což komplikuje testovatelnost a úpravy.[12, 13]



Obr. datový tok MVC [12]

Většinu problémů MVC řeší MVP (Model-View-Presenter), kde view a model nekomunikují napřímo, ale přes presenter jako prostředníka (viz Obr. 2). Oproti MVC zde na uživatelské akce reaguje view, které informaci předává presenteru. Ten při vracení aktualizovaných dat z modelu může provést další zpracování. Díky většímu oddělení jednotlivých částí usnadňuje testování a úpravy.[12, 13]



Obr. datový tok MVP [12]

MVVM (Model-View-ViewModel) je podobný MVP, ale view neobsahuje žádnou logiku a pouze vykresluje data, která dostane z viewModelu. Svůj obsah aktualizuje na základě eventu OnPropertyChanged (viz Obr. 3). Většina logiky se nachází ve viewModelu, který má také na starosti stav aplikace. Tento přístup umožňuje, aby více view bylo navázáno na jeden viewModel. Oproti svým předchůdcům je MVVM modulárnější, testovatelnější a snáze škálovatelný. Avšak za cenu vyšší komplexity tříd.[12, 13]



Obr. datový tok MVVM [12]

# Hlavní uzel

Hlavní uzel je realizován jako počítačový program. Řešení je rozděleno na tři části, které řeší komunikační, logickou a uživatelskou vrstvu. Každá vrstva má referenci jen na vrstvu pod ní. Toto řešení umožňuje snadnou změnu jednotlivých částí, bez výrazných zásahů do kódu.

## Komunikační vrstva

Text…

## Logická vrstva

Text…

## Uživatelské rozhraní

Text

# Uzly

Jednotlivé uzly jsou tvořeny jednočipovými počítači ESP8266.

## Uzel 1

text

## Uzel 2

text

## Uzel 3

text

# Výsledky a diskuse

## Podkapitola úroveň 2

Text…

### Podkapitola úroveň 3

Text…

### Podkapitola úroveň 3

Text…

## Podkapitola úroveň 2

Text…

# Závěr

Text…

# Seznam použitých zdrojů

[1] What’s a design pattern? *Refactoring Guru* [online]. [vid. 2025-01-25]. Dostupné z: https://refactoring.guru/design-patterns/what-is-pattern

[2] Why should I learn patterns? *Refactoring Guru* [online]. [vid. 2025-01-25]. Dostupné z: https://refactoring.guru/design-patterns/why-learn-patterns

[3] Difference Between Architectural Style, Architectural Patterns and Design Patterns - GeeksforGeeks. *GeeksForGeeks* [online]. [vid. 2025-01-26]. Dostupné z: https://www.geeksforgeeks.org/difference-between-architectural-style-architectural-patterns-and-design-patterns/

[4] STEVE “ARDALIS” SMITH. *Architecting-Modern-Web-Applications-with-ASP.NET-Core-and-Azure* [online]. 2023 [vid. 2025-01-21]. Dostupné z: https://dotnet.microsoft.com/en-us/download/e-book/aspnet/pdf

[5] RITVIK GUPTA. Software Architecture Patterns: What Are the Types and Which Is the Best One for Your Project | Turing. *Turing* [online]. [vid. 2025-01-26]. Dostupné z: https://www.turing.com/blog/software-architecture-patterns-types

[6] Dependency Injection(DI) Design Pattern - GeeksforGeeks. *GeeksForGeeks* [online]. [vid. 2025-01-19]. Dostupné z: https://www.geeksforgeeks.org/dependency-injectiondi-design-pattern/

[7] Single Responsibility in SOLID Design Principle - GeeksforGeeks. *GeeksForGeeks* [online]. [vid. 2025-01-19]. Dostupné z: https://www.geeksforgeeks.org/single-responsibility-in-solid-design-principle/

[8] NuGet Gallery | Microsoft.Extensions.DependencyInjection 1.0.0. *NuGet* [online]. [vid. 2025-01-23]. Dostupné z: https://www.nuget.org/packages/Microsoft.Extensions.DependencyInjection/1.0.0#supportedframeworks-body-tab

[9] Dependency injection - .NET | Microsoft Learn. *Microsoft Learn* [online]. [vid. 2025-01-23]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection

[10] Create Data Transfer Objects (DTOs) | Microsoft Learn. *Microsoft Learn* [online]. [vid. 2025-01-24]. Dostupné z: https://learn.microsoft.com/en-us/aspnet/web-api/overview/data/using-web-api-with-entity-framework/part-5

[11] BAELDUNG. The DTO Pattern (Data Transfer Object) | Baeldung. *Baeldung* [online]. [vid. 2025-01-24]. Dostupné z: https://www.baeldung.com/java-dto-pattern

[12] Difference Between MVC, MVP and MVVM Architecture Pattern in Android - GeeksforGeeks. *GeeksForGeeks* [online]. [vid. 2024-11-26]. Dostupné z: https://www.geeksforgeeks.org/difference-between-mvc-mvp-and-mvvm-architecture-pattern-in-android/

[13] NIMROD KRAMER. Android Architecture Patterns: MVC vs MVVM vs MVP. *daily.dev* [online]. [vid. 2025-01-03]. Dostupné z: https://daily.dev/blog/android-architecture-patterns-mvc-vs-mvvm-vs-mvp

# Přílohy

Odkazovaný seznam příloh