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 vzory

## Dependency injection

Dependency injection je technika, která snižuje závislost třídy na jiné. Toto umožňuje aplikaci být více modulární, lépe testovatelná a snáze upravitelná.[1]

Pokud má třída například zpracovat data a výsledek uložit do databáze, při klasickém přístupu je pevně svázána s konkrétním systémem. V horším případě obsahuje všechen kód, čímž porušuje Single responsibility principle (S ze SOLID). V lepším případě je práce s databází umístěna do vlastní třídy, ale její instance je součástí objektu s logikou. Oba tyto případy komplikují přechod z jednoho typu databáze na jiný a testování je velice obtížné, protože kód očekává funkční databázi.[1]

Aby se těmto problémům předešlo, je závislost předávána zvenčí, takže třída již není zodpovědná za správu. 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. Závislost je nejčastěji vkládána pomocí konstruktoru, ale může být také použita metoda.

## DTO

text

## 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.[2]

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.[2, 3]



Obr. datový tok MVC [2]

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.[2, 3]



Obr. datový tok MVP [2]

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.[2, 3]



Obr. datový tok MVVM [2]

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

[2] 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/

[3] 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