Stredná priemyselná škola Myjava

Ul. SNP 413/8, 907 01, Myjava

**LinkUP**

Stredoškolská odborná činnosť

Č. odboru: 11 - Informatika

Riešiteľ: Lukáš Pollák

Myjava

2024/2025 Ročník štúdia: štvrtý

Školiteľ: Mgr. Slavomír Mosný

*Vyhlasujem, že prácu stredoškolskej odbornej činnosti na tému webová aplikácia som vypracoval samostatne, s použitím uvedených literárnych zdrojov. Práca nebola prihlásená a prezentovaná v žiadnej inej súťaži, ktorá je pod gestorskom Ministerstva školstva, vedy, výskumu a šport SR*

**V Myjave dňa 15.11.2024**

Obsah

[1 Úvod 4](#__RefHeading___Toc3007_3165936259)

[2 Problematika práce 5](#__RefHeading___Toc3009_3165936259)

[2.1 Komunikačné protokoly a reálny čas 5](#__RefHeading___Toc9957_3165936259)

[2.2 Bezpečnosť a šifrovanie 5](#__RefHeading___Toc9959_3165936259)

[2.3 Architektúra aplikácií 5](#__RefHeading___Toc9961_3165936259)

[2.4 Frontend a UX dizajn 5](#__RefHeading___Toc9963_3165936259)

[3 Ciele práce 6](#__RefHeading___Toc3011_3165936259)

[3.1 Návrh a implementácia backendu aplikácie 6](#__RefHeading___Toc9967_3165936259)

[3.2 Návrh a implementácia frontendu aplikácie 6](#__RefHeading___Toc9969_3165936259)

[4 Použité technológie 7](#__RefHeading___Toc3013_3165936259)

[4.1 Programovací jazyk: Golang (Go) 7](#__RefHeading___Toc11954_3165936259)

[4.2 Ukladanie dát: PostgreSQL 7](#__RefHeading___Toc11956_3165936259)

[4.3 Frontend: Next.js a TailwindCSS 7](#__RefHeading___Toc11958_3165936259)

[5 Metodika 8](#__RefHeading___Toc12633_3165936259)

[5.1 Návrh a implementácia backendu 8](#__RefHeading___Toc12635_3165936259)

[5.1.1 Navrhol som logickú schému backendu. 8](#__RefHeading___Toc12962_3165936259)

[5.1.2 Návrhol som schému databázy 8](#__RefHeading___Toc12964_3165936259)

[5.1.3 Vytvoril som dotazy 9](#__RefHeading___Toc12966_3165936259)

[5.1.4 Implementácia užívateľskej služby 9](#__RefHeading___Toc12968_3165936259)

[5.1.5 Užívateľský handler 10](#__RefHeading___Toc13621_3165936259)

[5.1.6 Koncové body UserHandlera 10](#__RefHeading___Toc14422_3165936259)

[5.2 Implementácia WebSocket komunikácie 11](#__RefHeading___Toc12637_3165936259)

[5.2.1 WebSocket klient 11](#__RefHeading___Toc14388_3165936259)

[5.2.2 WebSocket room 11](#__RefHeading___Toc14390_3165936259)

[5.2.3 WebSocket hub 11](#__RefHeading___Toc14392_3165936259)

[5.2.4 Handler pre WebSocket 12](#__RefHeading___Toc14424_3165936259)

[5.2.5 Koncové body WebSocket handlera 13](#__RefHeading___Toc14428_3165936259)

[5.3 Vývoj frontendovej aplikácie 14](#__RefHeading___Toc12639_3165936259)

[6 Výsledky práce 15](#__RefHeading___Toc3015_3165936259)

[7 Diskusia 16](#__RefHeading___Toc3017_3165936259)

[8 Zavery prace 17](#__RefHeading___Toc3019_3165936259)

[9 Zoznam použitej literatury 18](#__RefHeading___Toc3021_3165936259)

# Úvod

Rýchla komunikácia je základom dnešnej doby. Je nevyhnutnou súčasťou nášho života, a preto som sa rozhodol vytvoriť vlastnú chatovaciu aplikáciu, pomocou ktorej si prajem spríjemniť internetovú komunikáciu. Ako povedal Mark Zuckerberg: *„Existuje obrovská potreba a príležitosť prepojiť každého na svete, dať každému hlas a pomôcť transformovať spoločnosť do budúcnosti.“* Táto vízia je pre mňa inšpiráciou, aby som vytvoril nástroj, ktorý umožní ľuďom byť bližšie k sebe a komunikovať efektívnejšie.

# Problematika práce

Chatovacie aplikácie predstavujú dynamicky sa rozvíjajúcu oblasť informačných technológií, ktorá kombinuje prvky komunikácie v reálnom čase, správy užívateľských údajov a zabezpečenia. V tejto kapitole sa zameriame na teoretické východiská, ktoré súvisia s vývojom chatovacích aplikácií, a prehľad existujúcej literatúry v tejto oblasti.

## Komunikačné protokoly a reálny čas

Jednou z hlavných technologických výziev pri vývoji chatovacích aplikácií je implementácia protokolov na prenos správ v reálnom čase. WebSocket je štandardom, ktorý poskytuje obojsmerné a perzistentné spojenie medzi klientom a serverom, čo je nevyhnutné pre rýchlu a efektívnu komunikáciu. Alternatívne protokoly, ako napríklad Server-Sent Events (SSE), ponúkajú jednoduchšie riešenia s nevýhodou jednosmernú komunikáciu.

## Bezpečnosť a šifrovanie

Bezpečnosť je kľúčovým aspektom chatovacích aplikácií. Symetrické a asymetrické šifrovanie, ako je napríklad AES alebo RSA, sú často používané na zabezpečenie dôverného prenosu správ. End-to-end šifrovanie je zárukou, že správy sú dešifrované len na zariadeniach odosielateľa a prijímateľa, je považované za štandard pre moderné aplikácie.

## Architektúra aplikácií

Moderné chatovacie aplikácie často využívajú architektúry založené na mikroservisoch, ktoré umožňujú lepšiu škálovateľnosť a úržovú úpravu jednotlivých komponentov. V prostredí Go Echo frameworku, ktorý je špecifický svojou jednoduchosťou a vysokým výkonom, je možné efektívne spravovať HTTP požiadavky a zabezpečiť kompatibilitu s databázovými riešeniami, ako je PostgreSQL.

## Frontend a UX dizajn

Používanie Next.js spolu s Tailwind CSS predstavuje moderný prístup k tvorbe užívateľských rozhraní. Tailwind CSS umožňuje rýchlu tvorbu štýlových komponentov vďaka svojmu utilitárnemu prístupu, čo potvrdzujú aj štúdie autorov ako Johnson (2023).

# Ciele práce

Hlavným cieľom tejto práce je vytvoriť modernú, bezpečnú a škálovateľnú chatovaciu aplikáciu s názvom LinkUP, ktorá bude slúžiť na efektívnu a intuitívnu komunikáciu medzi užívateľmi. Aplikácia bude navrhnutá tak, aby bola použiteľná na rôznych zariadeniach a poskytovala kvalitný užívateľský zážitok.

**Z hlavého cieľa vyplývajú nasledovné čiastkové ciele:**

## Návrh a implementácia backendu aplikácie

* + Navrhnutie robustného a škálovateľného servera.
  + Vytvorenie a optimalizácia databázovej štruktúry pomocou PostgreSQL na zabezpečenie rýchlej a spoľahlivej manipulácie s dátami.
  + Implementácia autentifikácie a autorizácie užívateľov s dôrazom na bezpečnosť.

## Návrh a implementácia frontendu aplikácie

* + Vytvorenie responzívneho užívateľského rozhrania.
  + Zabezpećiť konzistentný dizajnu.
  + Integrácia s backendom prostredníctvom REST API a WebSocketov.

# Použité technológie

Pri vývoji mojej aplikácie sa sústredím na efektívnosť, škálovateľnosť a moderné technologické riešenia. Používam nasledovné technológie:

## Programovací jazyk: Golang (Go)

Go je moderný programovací jazyk navrhnutý špeciálne na podporu paralelizmu a konkurenčného spracovania úloh. Vďaka zabudovaným mechanizmom, ako sú gorutiny a kanály, je ideálny pre aplikácie, ktoré vyžadujú vysoký výkon a prácu s internetovými protokolmi. Táto vlastnosť je zásadná pre moju aplikáciu, keďže umožňuje rýchlu a spoľahlivú komunikáciu medzi používateľmi.

## Ukladanie dát: PostgreSQL

Pre ukladanie a správu užívateľských dát využívam PostgreSQL – robustný a vysoko spoľahlivý relačný databázový systém. PostgreSQL poskytuje pokročilé funkcie, ako je podpora JSON, indexácia na mieru a transakcie, čo zabezpečuje bezpečné a efektívne spracovanie dát.

## Frontend: Next.js a TailwindCSS

* 1. **Next.js**: Next.js je moderný framework pre vývoj webových aplikácií postavený na Reacte. Poskytuje server-side rendering (SSR) a statickú generáciu stránok, čo zlepšuje výkon a SEO optimalizáciu. Vďaka svojim flexibilným možnostiam je ideálny pre vývoj dynamických a škálovateľných frontendových aplikácií.
  2. **TailwindCSS:** Tento nástroj pre tvorbu CSS štýlov umožňuje rýchle prototypovanie a precízne ovládanie vizuálneho vzhľadu stránky. Vďaka preddefinovaným utilitám dokáže TailwindCSS urýchliť vývoj moderného a responzívneho dizajnu.

# Metodika

Pri realizácii projektu LinkUP som postupoval systematicky, pričom práca bola rozdelená do troch hlavných fáz:

## Návrh a implementácia backendu

V prvej fáze som sa zameral na vytvorenie robustného a efektívneho backendu, ktorý zabezpečuje správu dát a komunikáciu s frontendovou časťou aplikácie. Postupoval som nasledovne:

### Navrhol som logickú schému backendu.

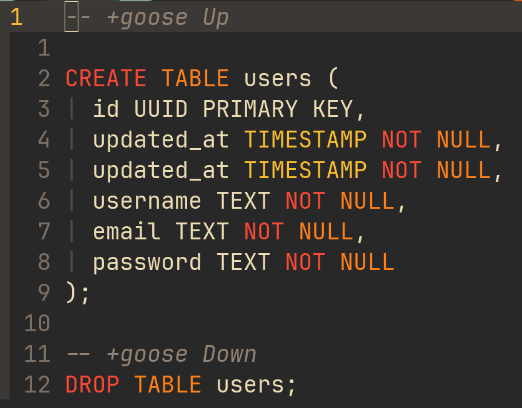
### 

**Obrázok 1. Logická schéma backendu**

### ****Návrhol som schému databázy****

Použitím nástroja Go Goose som navrhol a implementoval schému databázy users v PostgreSQL, pričom som sa sústredil na optimalizáciu pre ukladanie a manipuláciu s užívateľskými dátami.

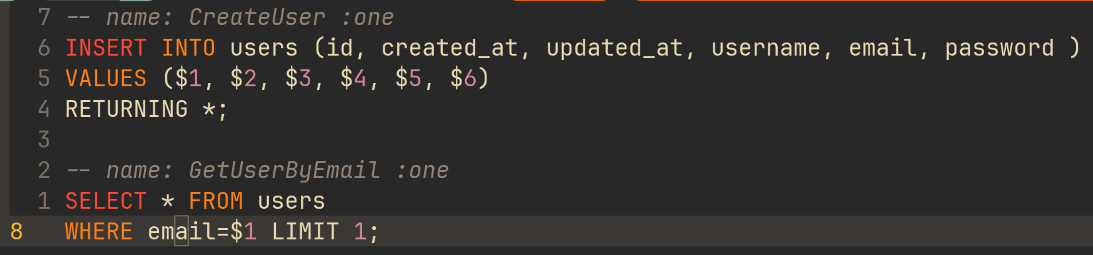
Schméma obsahuje polia: Id, Username, Email, Password



Obrázok 2. Schéma databázy users

### ****Vytvoril som dotazy****

S pomocou Go SQLC som automaticky generoval SQL dotazy pre vytvorenie nového použivateľa, a pre vyhladanie uživateľa podla emailu.



**Obrázok 2. SQLC dotazy**

### ****Implementácia užívateľskej služby****

Vytvoril som službu pre správu užívateľov, ktorá zahŕňa registráciu, prihlásenie s overením a uloženie udajov do databázy.

1. **Funkcia CreateUser** – Očakáva parametre meno, email a heslo. Po zavolaní zašifruje heslo a odošle data nového uživatela do databázy. Po úspšnom vykonaní vráti naspäť vygenerovane id, meno a email.
2. **Funkcia Login** – Očakáva parametre email a heslo. Po prijatí vyhlada zaśifrované heslo uživateľa v databáze a skontroluje ho s heslom z parametrov. Ak sa hesla zhodujú tak funkcia vygeneruje pristupovy token a spoločne s meno a idčkom ho navráti.

### ****Užívateľský handler****

UserHandler je handler určený na spracovanie požiadaviek súvisiacich s registráciou, prihlásením a odhlásením užívateľov. Tento handler komunikuje so službou user.Service, ktorá vykonáva logiku správy užívateľských účtov.

**Metódy handlera**

1. CreateUser - Táto metóda umožňuje vytvoriť nového užívateľa.
   * **Vstup:** JSON objekt obsahujúci údaje o novom užívateľovi, ako je meno, email, heslo atď. (v štruktúre CreateUserReq).
   * **Výstup:** Informácie o vytvorenom užívateľovi alebo chybová správa.
   * **Postup:**
     + Požiadavka sa zbinduje do štruktúry CreateUserReq.
     + Služba CreateUser vykoná operáciu vytvorenia nového užívateľa.
     + Ak je operácia úspešná, vráti sa odpoveď s informáciami o užívateľovi. V prípade chyby sa vráti chybová správa.
2. Login - Táto metóda umožňuje užívateľovi prihlásiť sa do aplikácie.
   * **Vstup:** JSON objekt obsahujúci prihlasovacie údaje užívateľa, ako je email a heslo (v štruktúre LoginUserReq).
   * **Výstup:** Informácie o prihlásenom užívateľovi, vrátane prístupového tokenu (JWT).
   * **Postup:**
     + Požiadavka sa zbinduje do štruktúry LoginUserReq.
     + Služba Login vykoná operáciu prihlásenia.
     + Ak je prihlásenie úspešné, vráti sa prístupový token (JWT), ktorý sa uloží do cookies pre ďalšie autentifikačné požiadavky.
     + V prípade chyby sa vráti chybová správa.
3. LogOut - Táto metóda umožňuje užívateľovi odhlásiť sa z aplikácie.
   * **Vstup:** Žiadne.
   * **Výstup:** Správa o úspešnom odhlásení.
   * **Postup:**
     + Odstráni sa prístupový token z cookies, čím sa ukončí relácia užívateľa.
     + Vráti sa správa o úspešnom odhlásení.

### Koncové body UserHandlera

UserHandler je pripojený na nasledujúce koncové body:

1. POST /register
2. POST /login
3. POST /logout

## Implementácia WebSocket komunikácie

Druhá fáza bola zameraná na vytvorenie služby pre komunikáciu v reálnom čases medzi užívateľmi:

### ****WebSocket klient****

Každého klienta na serveri reprezentuje štruktúra Client, ktorá obsahuje nasledovné polia:

* Conn – Aktívne WebSocket pripojenie umožňujúce komunikáciu medzi serverom a frontendom.
* Message – Kanál na odosielanie správ od klienta. Tento kanál umožňuje efektívnu výmenu dát medzi klientom a serverom.
* ID – Jedinečný identifikátor klienta vo forme UUID, ktorý zabezpečuje jeho jednoznačnú identifikáciu v systéme.
* RoomID – Identifikátor miestnosti, ku ktorej klient patrí, čo umožňuje organizáciu komunikácie medzi rôznymi skupinami užívateľov.
* Username – Užívateľské meno klienta, ktoré slúži na jeho identifikáciu z pohľadu ostatných užívateľov.

### WebSocket room

**Miestnosti, ktoré umožňujú skupinovú komunikáciu, sú reprezentované štruktúrou** Room**. Tá obsahuje:**

* ID – Jedinečný identifikátor miestnosti.
* Name – Názov miestnosti, ktorý ju identifikuje pre užívateľov.
* Clients – Zoznam pripojených klientov, ktorí sú členmi danej miestnosti.

### WebSocket hub

**Hub predstavuje centrálnu štruktúru na správu miestností a komunikácie medzi klientmi. Štruktúra** Hub **obsahuje:**

* Rooms – Zoznam aktívnych miestností spravovaných hubom.
* Register – Kanál, ktorý slúži na pripojenie klienta do príslušnej miestnosti. Tento mechanizmus zabezpečuje správne smerovanie užívateľov.
* Unregister – Kanál, ktorý umožňuje odpojenie klienta z miestnosti, čím sa udržiava aktuálny stav miestností.
* Broadcast – Kanál, ktorý slúži na distribúciu správ od klientov všetkým ostatným užívateľom v danej miestnosti.

### ****Handler pre WebSocket****

WebSocket handler slúži na spracovanie požiadaviek súvisiacich s WebSocket komunikáciou. Jeho hlavnou úlohou je sprostredkovať interakciu medzi klientmi a WebSocket hubom (Hub), ktorý spravuje miestnosti a klientov. Handler obsahuje nasledujúce metódy:

**Metódy handlera**

1. CreateRoom - Táto metóda umožňuje vytvoriť novú miestnosť.
   * **Vstup:** JSON objekt s identifikátorom miestnosti (id) a názvom miestnosti (name).
   * **Výstup:** Informácie o vytvorenej miestnosti vo formáte JSON.
   * **Postup:**
     + Požiadavka sa načíta do štruktúry CreateRoomReq.
     + Nová miestnosť sa pridá do zoznamu miestností v hube.
     + Návratová hodnota obsahuje informácie o miestnosti.
2. JoinRoom - Táto metóda umožňuje klientovi pripojiť sa do konkrétnej miestnosti.
   * **Vstup:**
     + Parametre URL: room-id (ID miestnosti).
     + Query parametre: userId (ID užívateľa), username (užívateľské meno).
   * **Výstup:** HTTP status kód 204 No Content.
   * **Postup:**
     + WebSocket pripojenie sa inicializuje pomocou upgradera.
     + Klient sa zaregistruje v príslušnej miestnosti prostredníctvom hube.
     + Odošle sa správa o pripojení klienta všetkým užívateľom v miestnosti.
3. GetRooms - Táto metóda vráti zoznam všetkých aktívnych miestností.
   * **Vstup:** Žiadne.
   * **Výstup:** Zoznam miestností vo formáte JSON, pričom každá miestnosť obsahuje id a name.
   * **Postup:**
     + Prechádza sa zoznam miestností v hube.
     + Informácie o miestnostiach sa pridajú do návratovej hodnoty.
4. GetClients - Táto metóda vráti zoznam všetkých klientov pripojených v danej miestnosti.
   * **Vstup:**
     + Parametre URL: room-id (ID miestnosti).
   * **Výstup:** Zoznam klientov vo formáte JSON, pričom každý klient obsahuje id a username.
   * **Postup:**
     + Overí sa, či miestnosť existuje.
     + Ak miestnosť existuje, prechádza sa zoznam klientov a pridajú sa ich údaje do návratovej hodnoty.

### Koncové body WebSocket handlera

WebSocket handler je pripojený na nasledujúce koncové body:

1. POST /ws/create-room
   * Slúži na vytvorenie novej miestnosti.
   * Spracováva sa pomocou metódy CreateRoom.
2. GET /ws/join-room/:room-id
   * Umožňuje klientovi pripojiť sa do miestnosti.
   * Spracováva sa pomocou metódy JoinRoom.
3. GET /ws/get-rooms
   * Poskytuje zoznam všetkých aktívnych miestností.
   * Spracováva sa pomocou metódy GetRooms.
4. GET /ws/get-clients/:room-id
   * Poskytuje zoznam klientov pripojených v konkrétnej miestnosti.
   * Spracováva sa pomocou metódy GetClients.

## Vývoj frontendovej aplikácie

Tretia fáza sa sústredila na vývoj užívateľského rozhrania:

* **Použitie Next.js:** Vytvoril som novú aplikáciu s využitím frameworku Next.js, ktorý umožňuje rýchly a efektívny vývoj moderných webových aplikácií.
* **Dizajn stránky:** Pomocou Tailwind CSS som navrhol responzívny a intuitívny dizajn, ktorý zabezpečuje kvalitný užívateľský zážitok.
* **Integrácia frontend a backend:** Frontend som napojil na backendové API a WebSocket službu, čím som zabezpečil plnú funkcionalitu aplikácie.

# Výsledky práce

# Diskusia

# Zavery prace

# Zoznam použitej literatury

* Dokumentácia k jazyku Go: <https://go.dev/doc/>
* Dokumentácia k kniznici Go echo: <https://echo.labstack.com/>
* Ako pracovat s postgres v Go: <https://www.geeksforgeeks.org/connection-with-postgres-in-golang/>
* Dokumentácia ku Next.js: <https://nextjs.org/docs>
* Ako sa pracuje s WebSocketmi v JS: <https://www.geeksforgeeks.org/how-to-create-a-websocket-connection-in-javascript/>
* Tailwindcss pre začiatočníkov: <https://www.youtube.com/watch?v=DenUCuq4G04>