Memopad

HyScript7
hyscript7@gmail.com

mobilex1122
contact@mobilex1122.eu

SomeKristi
kmaas2007@gmail.com

GYMNÁZIUM A STŘEDNÍ ODBORNÁ ŠKOLA
Rokycany, Mládežníků 1115

Obsah

1 Úvod

Každý student si během studia píše poznámky, ať už z hodin, nebo z cvičení. Problém však nastává ve chvíli, když je chce sdílet nebo doplnit od spolužáků. Každý totiž používá jiný nástroj, a většinou i formát. Někdo preferuje jednoduchý plaintext, jiný používá Markdown, další spoléhá na LaTeX a někteří si vše zaznamenávají v tradičních textových dokumentech jako Microsoft Word. Tato rozmanitost formátů však komplikuje vzájemnou spolupráci a často vede ke ztrátě formátování nebo složité konverzi mezi různými formáty.

Tento problém nás přivedl k myšlence vytvořit vlastní webovou aplikaci MemoPad, která by umožnila snadné psaní, ukládání, správu a sdílení poznámek v univerzálním prostředí. Hlavním cílem bylo vytvořit jednoduchý, přehledný a efektivní nástroj, který by eliminoval potíže s konverzí formátů a zajistil bezproblémovou synchronizaci mezi zařízeními. Zároveň jsme tuto práci vnímali jako výzvu, chtěli jsme zjistit, zda dokážeme navrhnout a implementovat aplikaci, která by se mohla stát praktickým řešením tohoto problému.

Při návrhu jsme se inspirovali moderními webovými aplikacemi, jako notesnook, nebo Overleaf, a rozhodli jsme se využít koncept Single Page Application (SPA) pro plynulé a rychlé uživatelské prostředí. Plánujeme také automatickou synchronizaci s cloudovým úložištěm a potenciální podporu exportu do různých formátů, aby si uživatelé mohli své poznámky snadno přizpůsobit vlastním potřebám.

Práce se skládá z několika hlavních částí. Nejprve provedeme analýzu existujících řešení a identifikujeme jejich silné a slabé stránky. Následně se zaměříme na návrh a implementaci aplikace, kde podrobně popíšeme použitou technologii jak na straně serveru (backend – část aplikace, která běží na serveru a stará se o zpracování dat), tak na straně uživatelského rozhraní (frontend – část aplikace, se kterou komunikuje uživatel). Poté se budeme věnovat testování funkčnosti aplikace a ověříme, že splňuje stanovené požadavky. Nakonec připravíme uživatelskou příručku, která popíše, jak aplikaci používat, a shrneme dosažené výsledky v závěru.

Tato práce, aneb dokumentace, poskytuje nejen ucelený pohled na vývoj webové aplikace pro poznámky, ale také rozvíjí možnosti jejího budoucího rozšíření o další funkce, jako je podpora formátování textu, spolupráce více uživatelů nebo exportování pro využití mimo prostředí aplikace.

2 Analýza

V současné době existuje mnoho různých aplikací pro psaní poznámek, přičemž každá je zaměřena na jiný způsob práce s textem. Mezi nejpopulárnější aplikace, se kterými jsme se setkali, patří následující programy a stránky:

- **Obsidian.md** – pokročilá aplikace pro psaní poznámek v Markdownu, zaměřená na propojení jednotlivých poznámek do sítě znalostí.
- **Overleaf** – online editor zaměřený na vytváření dokumentů v LaTeXu, často používaný studenty technických oborů.
- **Microsoft Word** – tradiční textový procesor s širokými možnostmi formátování a bohatou funkcionalitou.
- **Notesnook** – bezpečná a šifrovaná aplikace podporující Markdown pro psaní poznámek.

Hlavním problémem při práci s těmito nástroji je jejich vzájemná nekompatibilita. Každá aplikace využívá jiný formát pro ukládání poznámek, což vede k potížím při sdílení a spolupráci mezi uživateli. Pokud chce například uživatel převést markdown soubor z Obsidian.md do Microsoft Wordu, může dojít ke ztrátě formátování nebo nekompatibilitě s určitou funkcionalitou. Uživatelé pak často volí dvě nepohodlné alternativy: Buď se dál pokouší poznámky konvertovat, což vede k častým chybám a ztrátě formátu textu, nebo je musí ručně přepisovat, což je časově náročné a zbytečně odvádí pozornost od samotného učení.

Zvažovali jsme několik možných řešení tohoto problému:

- **Vytvoření konvertoru formátů** – Mohli jsme vytvořit nástroj, který by dokázal konvertovat různé formáty poznámek mezi sebou. Nicméně to by nebylo tak zajímavé jako vývoj celé fullstack aplikace pro poznámky. Navíc, většina konvertorů je podle nás uživatelsky nepřívětivá a často nefunguje dokonale.
- **Desktopová aplikace s podporou různých formátů** – Další možností bylo vytvořit nativní aplikaci, která by umožňovala práci s různými formáty a jejich snadnou integraci. To znělo už přívětivěji, ale mělo to jednu zásadní nevýhodu – většina uživatelů dnes pracuje na mobilních zařízeních, a vytvoření multiplatformní desktopové i mobilní aplikace by vyžadovalo značné úsilí.
- **Webová aplikace s cloudovým úložištěm** – Nakonec jsme se rozhodli vytvořit webovou aplikaci, která poznámky ukládá přímo v cloudu a umožňuje přihlášení odkudkoliv s okamžitou synchronizací změn. Díky webovým technologiím bude možné v budoucnu snadno přidávat nové funkce, například podporu transpilace formátů, a to buď rozšířením REST API, nebo vytvořením specializované mikroslužby, na kterou by frontend mohl posílat žádosti. Aby aplikace byla plynulá a interaktivní, rozhodli jsme se využít Single Page Application (SPA), což umožňuje okamžitou odezvu na změny bez nutnosti neustálého načítání stránky.

2.1 Rozdělení problému na podúlohy

Naše zvolené řešení jsme rozdělili na několik nezávislých funkcí:

Backend:

- Správa složek a podsložek
- Správa poznámek, včetně obsahu a umístění ve složkách
- Zabezpečení (uživatelské účty apod.)
- Exportování jako PDF, HTML nebo zdrojový kód
- Sdílení poznámek pomocí odkazu nebo přístupu

Frontend:

- Formy pro přihlášení a registraci (správa tokenu)
- Panel pro správu a přehled složek (integrace s API složek)
- Editor poznámek

Ke dni 17.03.2025 jsou hotové funkce 1-3 na backendu a 1-2 na frontendu.

3 Implementace

3.1 Architektura aplikace

Naše aplikace je rozdělena na backend a frontend projekty.

Backend je implementován jako Gradle Spring Boot aplikace v Javě 21. Tento framework jsme zvolili, protože jsme potřebovali rychle implementovat přihlašování a registraci, abychom se mohli soustředit na složitější funkce.

Frontend je implementován jako pnpm Angular aplikace v TypeScriptu. Angular jsme zvolili, protože to je frontend framework, s kterým jsme za posledních pár projektů nasbírali nejvíce zkušeností, a jeho OOP architektura nám umožnila reagovat na náhle změny implementace na backendu.

Tyto dvě části spolu komunikují tak, že frontend posílá žádosti na adresu, ze které byl načtený, a to na routy začínající /api/. Backend zpracovává tyto žádosti a vrací odpovědi. Pokud backend obdrží žádost, která nezačíná na /api nebo /static, vrátí vždy index.html, čímž umožní správnou funkčnost SPA.

Pro databázi využíváme PostgreSQL. Kromě toho jsme přidali Elasticsearch, který umožňuje pokročilé vyhledávání nejen podle obsahu, ale také fuzzy search a potenciálně i vyhledávání podle významu textu. Ačkoliv jsme prakticky Elasticsearch nestihli implementovat, nadstavbu pro něj máme připravenou.

V následujících pod-kapitolách detailně rozebereme implementaci backendu, frontendu, teoretickou implementaci fuzzy search a konkrétní implementace specifických nebo žláštně implementovaných funkcí.

3.2 Implementace backendu

Backend využívá Spring Boot a následující projekty:

- Spring Data JPA pro práci s databází.
- Spring Web pro implementaci REST API.
- Spring Security pro autentizaci a autorizaci.
- Spring Actuator pro monitoring a správu aplikace.
- JJWT pro práci s JWT tokeny.
- Flyway pro migrace databáze.

Struktura backendového projektu se dělí podle stereotypu komponent:

- eu.projnull.memopad.config – Spring Beany a jejich konfigurace.
- eu.projnull.memopad.controllers – REST controllery.
- eu.projnull.memopad.controllers.dto – DTO objekty pro JSON odpovědi API.
- eu.projnull.memopad.models – Jakarta entity.
- eu.projnull.memopad.repositories – Spring Data JPA repozitáře.
- eu.projnull.memopad.security – Výjimka oproti ostatním balíčkům - zde se nachází obecné komponenty pro vlastní chování Spring Security.
- eu.projnull.memopad.services – aplikační logika.

Backendové API zahrnuje následující nechráněné endpointy:

- POST/api/auth/login - Přijme uživatelské jméno a heslo a vrátí token
- POST/api/auth/register - Přijme uživatelské jméno, email a heslo a vrátí token

Dále všechny následující endpointy vyžadují autorizaci:

- GET /api/auth/info - Vrátí informace o uživateli
- GET /api/folders/ - Vrátí hlavní složku uživatele
- GET /api/folders/{id} - Vrátí určenou složku
- POST /api/folders/{id}/create - Vytvoří v dané složce podsložku
- POST /api/folders/{id}/rename - Přejmenuje danou složku

- POST /api/folders/{id}/move/{idParent} - Přesune danou složku id do jiné složky idParent
- DELETE /api/folders/{id}/delete - Smaže danou složku
- GET /api/folders/{id}/files - Vrátí všechny poznámky ve složce
- GET /api/folders/{id}/folders - Vrátí všechny podsložky ve složce
- POST /api/notes/create - Vytvoří novou poznámku v určené složce (request body)
- GET /api/notes/{id} - Vrátí obsah a informace o dané poznámce
- POST /api/notes/{id}/move - Přesune poznámku do jiné dané složky (request body)
- POST /api/notes/{id}/rename - Přejmenuje danou poznámku
- DELETE /api/notes/{id} - Smaže danou poznámku
- POST /api/notes/{id}/content - Aktualizuje obsah poznámky

Autorizace probíhá pomocí JWT tokenů předávaných v hlavičce Authorization. Bezpečnost zajišťuje Spring Security s filtrováním požadavků a hashováním hesel pomocí BCrypt. Databázová vrstva využívá Spring Data JPA pro práci s PostgreSQL.

3.3 Implementace frontendu

Frontend aplikace je postaven na Angularu, přičemž pro správu balíčků využíváme pnpm. Během vývoje používáme speciální konfiguraci ng-cli (konkrétně Vite) pro přesměrování požadavků začínajících /api na localhost:8080, kde běží backend.

V produkčním prostředí backend zachytává všechny požadavky na /api a obsluhuje je přímo, zatímco ostatní cesty zůstávají zodpovědností frontendu.

Statická SPA aplikace je po sestavení přesunuta do složky resources/static backendového projektu.

Proces sestavení probíhá následovně:

1. **pnpm build** – vytvoří optimalizovanou verzi frontendové aplikace.
2. **gradle bootjar** – sestaví kompletní backendový JAR soubor včetně statického frontendu.

Tím je zajištěno, že aplikace lze jednoduše spustit jako jeden spustitelný JAR soubor s integrovaným frontendem.

Celé uživatelské rozhraní je samozřejmě responzivní a funguje tudíž jak na desktopu, tak i na mobilních zařízeních.

3.4 Fuzzy search pomocí Elasticsearch

Ačkoliv se v rámci 4-týdenního sprintu na implementaci vyhledávání nedostalo, nadstavba a teorie jejího provedení je připravená.

Nezávyslá služba Search Index by průběžně synchronizovala záznamy z Postgresu do Elasticsearch, kde by také přidala vektory významu každého paragrafu nebo věty. (Konkrétně se jedná o implementaci pomocí knihovny „Natural Language Processing Toolkit“ pro python).

Dále bychom hledání ze strany další mikroslužby řešili tím, že se

1. Najdou nejlepší shody pro název souboru
2. Najdou nejlepší shody pro význam názvu souboru
3. Najdou nejlepší shody pro význam obsahu

A výsledky se seřadí dle % shody nezávysle na tom, v které kategorii se schoda nachází.

3.5 Detaily a zláštní implementace

Jedinou zvláštní implementací, která by stále za zmínění je jak řešíme identifikaci uživatele z tokenu.

Tradičním způsobem, a způsobem, který očekává Spring Security, je že se v tokenu uloží uživatelské jméno uživatele, nic méně, náš backend se orientoval kolem ID uživatele, tudíž, dle následující segmentu je vidět, že namísto UserDetailsService všude využíváme konkrétní třídy eu.projnull.memopad.services.UserService.

Toto se konkrétně děje ve třídách:

1. SecurityConfig

```java
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

    private final JwtService jwtService;

    private final UserRepository userRepository;

    SecurityConfig(UserRepository userRepository, JwtService jwtService) {
        this.userRepository = userRepository;
        this.jwtService = jwtService;
    }

    @Bean
    public UserService userService() {
        return new UserService(userRepository, jwtService, passwordEncoder());
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public UserDetailsService userDetailsService() {
        return userService();
    }

    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();
        authenticationProvider.setUserDetailsService(userDetailsService());
        authenticationProvider.setPasswordEncoder(passwordEncoder());
        return authenticationProvider;
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws
    Exception {
        return config.getAuthenticationManager();
    }

    @Bean
    public JwtAuthFilter jwtAuthFilter(UserService userService) {
        return new JwtAuthFilter(userService);
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http, JwtAuthFilter jwtAuthFilter)
    throws Exception {
        http
                .csrf(csrf -> csrf.disable())
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers("/api/auth/register", "/api/auth/login", "/static/**", "/{name
    :^(?!api).+}/**", "/").permitAll()
                        .anyRequest().authenticated())
                .sessionManagement(sess -> sess.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authenticationProvider(authenticationProvider())
                .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }
}
```

2. JwtFilter

```java
/*
 * Credits to https://www.geeksforgeeks.org/spring-boot-3-0-jwt-authentication-with-spring-security-using-mysql-database/
 * This is a modified version of the original code, which I wouldn't have had figured out by my self in a reasonable amount of time.
 * Might revisit this in the future, but for the time being, this will do.
 */
@Component
@Slf4j
public class JwtAuthFilter extends OncePerRequestFilter {

    // The proper way to do this is to inject the JWTService and UserDetailsService.
    // Since our UserService and JWTService use a Long userId, instead of a
    // username, we can't use a UserDetailsService here, so this is a workaround
    // until I refactor at some point in the future.
    private final UserService userService;

    public JwtAuthFilter(UserService userService) {
        this.userService = userService;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
            throws ServletException, IOException {
        String authHeader = request.getHeader("Authorization");
        String token = null;
        User user = null;

        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            token = authHeader.substring(7);
            user = userService.fromJwtToken(token).orElse(null);
        }

        // check if the token is valid and set the user in the security context
        if (user != null && SecurityContextHolder.getContext().getAuthentication() == null) {
            UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(
                    user, null, user.getAuthorities());
            authenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(authenticationToken);
        }
        filterChain.doFilter(request, response);
    }
}
```

V budoucnu kdy se tento přehlednutý detail opravý, toto jsou jediné dvě třídy, jejichž implementace se bude muset částečně změnit.

4 Testování

Testovali jsme jednotlivé funkce během implementace. Po dokončení každé části backendu jsme pomocí REST klienta ověřili všechny endpointy. Po implementaci odpovídajících funkcí na frontendu jsme ladili kontrakty mezi frontendem a backendem a poté znovu ověřili celkovou funkčnost.

Některé prvky na frontendu mají vlastní unit testy napsané v Angularu. Na backendu má jednotkové testy například JwtService, které jsou implementovány pomocí JUnit. Tento přístup nám umožnil identifikovat a opravit chyby v raných fázích vývoje a zajistit stabilitu aplikace před jejím nasazením.

5 Uživateslká Příručka

Tato sekce vás provede skrze různé funkcionality aplikace. Přeskočte na podsekci, která vás zajímá.

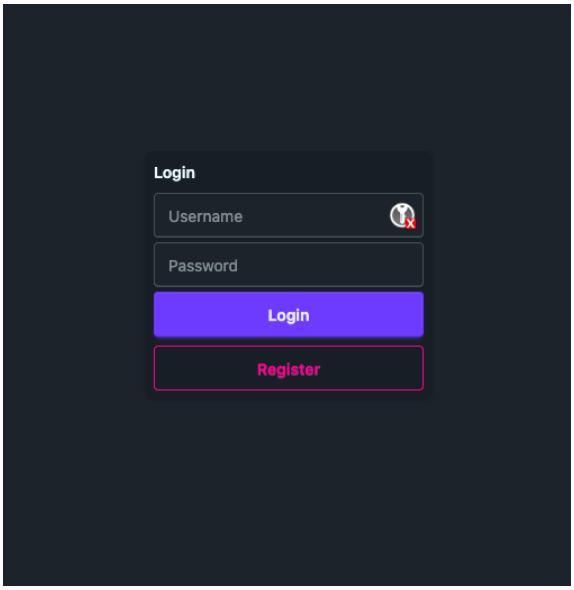
5.1 Přihlášení a registrace

Po navigaci na adresu Memopadu jsou před Vámi tlačítka Přihlásit a Registrovat:

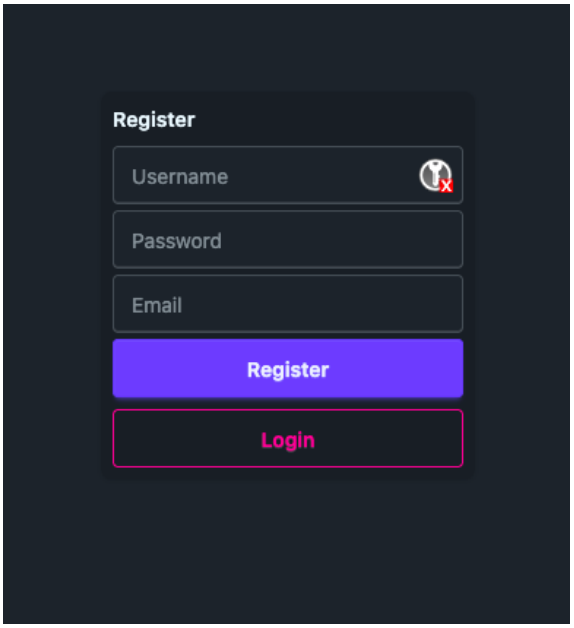
Pokud již účet máte, můžete vyplnit svoje údaje do polí username a password a poté kliknout na tlačítko „Login“.

Pokud účet ještě nemáte, klikněte na růžové tlačítko „Register“ a vyplňte všechny pole. Po kliknutí na modré tlačítko „Register“ budete automaticky přihlášeni.

Pokud jste nějakou náhodou skončili na obrazovce registra, ale účet už máte, můžete kliknout na růžové tlačítko „Login“, aby jste se vrátili na obrazovku přihlášení.



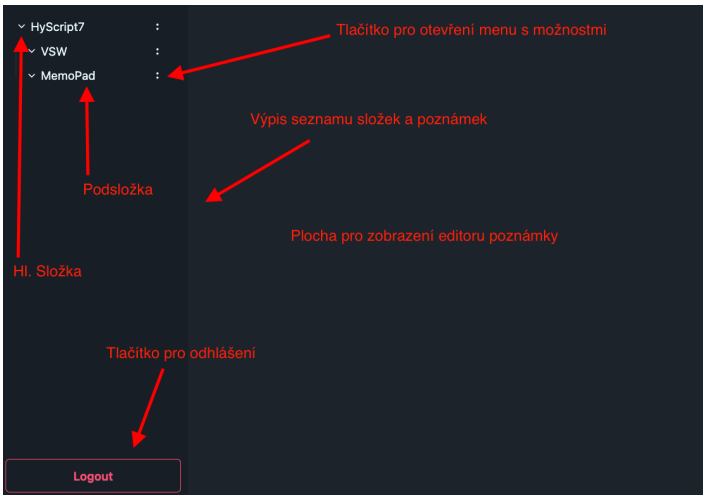
Obrázek 1: Obrazovka přihlášení



Obrázek 2: Obrazovka registrace

5.2 Uživatelské Rozhraní

Po přihlášení můžeme vidět uživatelské rozhraní MemoPadu. Následující grafika znázorňuje různé ovládací prvky rozhraní.



Obrázek 3: Uživatelské rozhraní MemoPadu

6 Závěr

Cílem práce bylo navrhnout implementaci webové aplikace, která by řešila problém se správou zápisků mezi různými formáty a jejím sdílením. Díky moderním technologiím, jako jsou Angular na frontendu a Spring Boot na backendu, bylo dosaženo poměrně rozsáhlého řešení, ačkoliv nebylo úplně dotaženo do konce.

Možnosti dalšího zlepšení zahrnují rozšíření podpory pro více formátů poznámek (například LaTeX, HTML či PDF) – Aktuálně je backendu jedno, co je v obsahu poznámky, prostě to uloží – , implementaci reálné spolupráce více uživatelů v reálném čase, vylepšenou správu oprávnění pro sdílené poznámky a dokončení implementace pokročilého vyhledávání. Dále by bylo vhodné optimalizovat výkon aplikace, zejména při práci s větším množstvím dat.

Celkově lze říci, že hotová aplikace splnila své základní cíle a poskytuje solidní základ pro další rozvoj. Její návrh umožňuje jednoduché rozšíření o další funkce, čímž by mohla nabídnout plnohodnotnou alternativu k existujícím řešením pro správu poznámek.