UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Fakturační aplikace technická dokumentace backend

Semestrální práce

Obsah

1. Úvod

Backend aplikace Fakturka je robustní a škálovatelná část systému, navržená pro efektivní zpracování dat, komunikaci s databází a poskytování REST API pro frontendovou aplikaci. Cílem backendu je zajistit bezpečný, rychlý a flexibilní provoz aplikace pro správu faktur.

Hlavní vlastnosti

- **Architektura**
  - Backend využívá architekturu postavenou na Spring Boot, která umožňuje rychlou a efektivní implementaci API.
- **Databáze**
  - Data jsou ukládána v relační databázi PostgreSQL, která zajišťuje integritu dat a škálovatelnost.
- **Bezpečnost**
  - Backend implementuje autentizaci a autorizaci pomocí JWT tokenů, zabezpečující přístup k citlivým údajům.
- **Migrace databáze**
  - Správa migrací je řešena pomocí Liquibase, což zajišťuje konzistenci schématu napříč různými prostředími.
- **Testování**

- Kvalita kódu je zajištěna pomocí unit testů a integrálních testů.

Technické požadavky

Back-end

- Implementace back-end logiky s využitím jazyka Java.
- Vytvoření a správa REST API pro komunikaci mezi front-endem a back-endem.

Databáze

- Použití relační databáze PostgreSQL pro ukládání uživatelských účtů a faktur.
- Implementace základních operací CRUD (Create, Read, Update, Delete) pro práci s fakturami.

Autentizace a autorizace

- Využití bezpečného autentizačního systému (JWT nebo OAuth) pro správu uživatelských přihlášení.
- Ochrana přístupu k fakturám a dalším citlivým datům prostřednictvím zabezpečených endpointů API.

Primární moduly

1. **Autentizace a autorizace:** Zabezpečení aplikace pomocí JWT tokenů.
2. **Správa faktur:** CRUD operace pro faktury včetně validace vstupů.
3. **Uživatelská správa**: Registrace, přihlášení a správa uživatelských údajů.
4. **Šablony faktur:** Práce se šablonami pro snadnější vytváření faktur.

Cílová skupina

Tato dokumentace je určena pro vývojáře, kteří se budou podílet na údržbě nebo rozšiřování backendu aplikace Fakturka, a pro DevOps inženýry zajišťující nasazení a provoz aplikace.

2. Implementace back-end

Struktura balíčků je tradiční dle vrstvené architektury

Balíček common

obsahuje obecné funkcionality a sdílené komponenty, které lze využít napříč celou aplikací. Je rozdělen do několika tematických podbalíčků:

- Controller
    - Validation

        - OnCreate
          Rozhraní či validační skupina používaná pro validaci entit při vytváření nových záznamů.
        - OnUpdate
          Rozhraní či validační skupina určená pro validaci entit při aktualizaci existujících záznamů.
    - GlobalControllerAdvice
      Globální „advice“ pro kontrolery (třída s anotací @ControllerAdvice), jejím úkolem je zachytávat výjimky. Zpracovává je a vrací vhodné HTTP odpovědi.
        - Typicky zde může docházet k mapování aplikačních výjimek na odpovídající HTTP status kódy.
        - Třída tak zajišťuje centralizovanou error-handling logiku pro všechny REST kontrolery v aplikaci.
- email
    - IMailSender
      Rozhraní definující metody pro odesílání e-mailů.
    - IMailTemplateEngine
      Rozhraní pro práci s e-mailovými šablonami.
    - impl
        - Zde se nacházejí implementace rozhraní IMailSender a IMailTemplateEngine.

- exceptions
    - BadRequestException
      Výjimka vyjadřující chybný požadavek (HTTP 400).
    - ConflictException
      Výjimka pro konflikt (HTTP 409), typicky při vytváření záznamu, který už existuje.
    - InvalidTokenException
      Výjimka používaná při neplatném tokenu (např. JWT).
    - NotFoundException
      Výjimka pro neexistující zdroj (HTTP 404).
    - UnauthorizedException
      Výjimka pro nepovolený přístup (HTTP 401 nebo 403).

Tyto výjimky jsou pak obvykle zachycovány v GlobalControllerAdvice, které na základě typu výjimky vrátí příslušnou HTTP chybu a případně další datae

- hashing
    - IHashProvider
      Rozhraní definující metody pro hashování dat (např. hesel).
    - impl
        - Sha256Provider
          Implementace IHashProvider využívající SHA-256 algoritmus.
        - Další možné implementace různých hashovacích algoritmů (např. Bcrypt, MD5, PBKDF2), pokud v projektu existují.

Tento balíček tak zajišťuje generické rozhraní pro hashování a konkrétní implementace zabezpečené funkce (SHA-256).

- mapping
    - IGenericMapper
      Rozhraní (nebo abstraktní třída) určující, jak převádět mezi doménovými objekty (např. entitami) a DTO.
    - impl
        - GenericModelMapper
          Konkrétní implementace IGenericMapper, využívá se k mapování polí mezi entitami a DTO na základě reflexe.

Tato část zjednodušuje převody datových tříd (model-DTO) v rámci aplikace a centralizuje mapovací logiku, aby se neopakovala na více místech.

Balíček configuration

obsahuje konfigurační třídy využívající Spring Boot (resp. Spring Framework) mechanismus pro Java-based konfiguraci. Typicky jsou označeny anotací @Configuration a poskytují konfigurační @Bean metody. Díky tomu se dají snadno udržovat a přizpůsobovat požadavkům projektu.

- AsyncConfiguration
  - Umožňuje asynchronní zpracování úloh.
  - Definuje bean notificationExecutor, což je ThreadPoolTaskExecutor (vláknový pool).
  - Nastavení parametru corePoolSize, maxPoolSize a queueCapacity určuje, kolik vláken je vyhrazeno pro asynchronní úlohy, kolik jich maximálně může vzniknout a kolik úloh se může zařadit do fronty.
  - @EnableAsync povoluje v celém projektu použití asynchronních metod označených @Async.

Tato konfigurace je užitečná tam, kde je potřeba spouštět úlohy na pozadí, aniž by se blokoval hlavní běh programu (např. rozesílání e-mailových notifikací).

- CacheConfiguration (Deprecated)
  - Konfigurace cache pro použití s Redisem.
  - Definuje bean RedisCacheManagerBuilderCustomizer, který konfiguruje cache s názvem "jwt-blacklist".
  - V RedisCacheConfiguration nastavuje entryTtl(...), tedy dobu platnosti položek v této cache (dle JwtTokenProvider.ACCESS_EXPIRATION_TIME).

Protože je třída označena jako @Deprecated, je možné, že se v budoucnu přestane používat nebo bude nahrazena jiným řešením. V této verzi projektu ale stále plní svou roli pro nastavení Redis cache (např. kvůli blokování už neplatných JWT tokenů).

- ModelMapperConfiguration
    - Centralizované nastavení knihovny ModelMapper, používané k mapování mezi entitami a DTO objekty.
    - Vytváří bean modelMapper, který je k dispozici zbytku aplikace.
    - Obsahuje metody MapInvoiceToInvoiceDto a MapInvoiceDtoToInvoice, kde se definují vlastní pravidla mapování mezi entitou Invoice a objektem InvoiceDto.
    - Nastavení TypeMap (pro mapování vybraných polí) a PostConverter, který po dokončení mapování nastavuje složitější struktury (např. PartySnapshotDto pro dodavatele a zákazníka).

Díky tomu je mapování přehledné a soustředěné na jednom místě, takže zbytek aplikace jen využívá připravený ModelMapper.

- OwnershipCheckerConfiguration
    - Registruje bean který slouží k validaci vlastnictví (zda má uživatel právo manipulovat s určitými entitami).
    - Metoda ownershipCheckerMap(List<IOwnershipChecker> checkers) projde všechny implementace rozhraní IOwnershipChecker v aplikaci.
    - Ty registruje do mapy podle toho, s jakým typem entity daný checker pracuje.
    - Pokud existují dvě stejné implementace pro tentýž typ entity, vyhodí výjimku.

Tato konfigurace zajišťuje centrální evidenci všech „ownership checkerů“, které kontrolují, zda má aktuální uživatel právo k dané entitě.

- SecurityConfiguration
    - Nastavuje Spring Security – autentizaci, autorizaci a související filtry.
    - Funkce:
        - passwordEncoder() vytváří bean BCryptPasswordEncoder pro ukládání hesel v zašifrované podobě.
        - securityFilterChain(HttpSecurity http) vypíná CSRF, nastavuje JWT filtr (jwtAuthenticationFilter) před UsernamePasswordAuthenticationFilter, a definuje URL cesty, které jsou přístupné bez autentizace (/swagger-ui/**, /v3/api-docs/**, /public/**).
        - Pro ostatní cesty je vyžadována autentizace.
        - @EnableMethodSecurity umožňuje v kódu používat anotace jako @PreAuthorize pro jemnější řízení přístupu.

Díky tomu zde máte kompletní nastavení zabezpečení, včetně JWT tokenů a nastavení přístupů pro veřejné a chráněné endpointy.

- SwaggerConfiguration
    - Integruje OpenAPI (dříve Swagger) do aplikace.
    - Funkce:
        - Metoda apiInfo() nastavuje základní informace o API (jako jméno projektu, verzi a popis) a přidává bezpečnostní schéma typu HTTP s Bearer tokenem (JWT).
        - Metoda apiGroupV1() definuje OpenAPI group pro cesty /v1/**. Nastavuje bezpečnostní požadavky (JWT) pro operace.
        - Metoda publicGroupV1() definuje skupinu pro "/public/v1/**", označenou jako public.

Zajišťuje tak, že se ve Swagger UI (či generované OpenAPI definici) zobrazí rozdělení endpointů podle verzí/skupin. Lze pak snáze testovat a dokumentovat REST rozhraní.

- WebConfiguration
    - Nastavuje CORS (Cross-Origin Resource Sharing) pravidla, popř. další webové konfigurace (např. formátování JSON, validace).
    - Metoda addCorsMappings(CorsRegistry registry) povoluje požadavky z definovaných domén (např. http://localhost:5000), určuje povolené metody (GET, POST, PUT, DELETE, OPTIONS), hlavičky (*) a nastavení allowCredentials(true).
    - maxAge(3600) určuje, jak dlouho se nastavení CORS může cachovat (v sekundách).

To umožňuje frontendu (na lokální adrese http://localhost:5000) bezproblémově komunikovat s API.

Balíček controller

Všechny třídy v tomto balíčku jsou označeny anotací @RestController, mají definované HTTP cesty pomocí @RequestMapping a obvykle využívají @RequiredArgsConstructor pro injektování závislostí (služeb) přes konstruktor. Navíc jsou zde použity Swagger/OpenAPI anotace (@Operation, @ApiResponse atd.) pro generování dokumentace API.

- AuthControllerV1
  - Závislosti:
    - IAuthService authService pro autentizační logiku (přihlášení, registrace),
    - IJwtTokenService refreshTokenService pro práci s JWT a refresh tokeny.
  - Hlavní endpointy:
    - POST /login: Přihlásí uživatele, vrátí access token a nastaví refresh token do HttpOnly cookie.
    - POST /register: Zaregistruje nového uživatele, vrátí access token a nastaví refresh token do HttpOnly cookie.
  - Swagger:
    - @Operation(summary="Login", description="...") + detailní definice request/response modelů.
    - @Operation(summary="Register", description="...") + validace vstupních dat.

Tento kontroler zajišťuje kompletní proces přihlášení a registrace nových uživatelů.

- PasswordResetControllerV1
  - Závislosti:
    - IUserCredentialsService userCredentialsService pro správu uživatelských přihlašovacích údajů (vytvoření a odeslání reset tokenu, nastavení nového hesla).
  - Hlavní endpointy:
    - POST /request: Vytvoří a odešle na e-mail token pro reset hesla.
    - POST /confirm: Provede reset hesla na základě tokenu.

Tento kontroler řeší veškerou funkcionalitu okolo obnovení hesla (token, e-mail, validace).

- TokenControllerV1
    - Závislosti:
        - IJwtTokenService refreshTokenService pro práci s JWT tokeny.
    - Hlavní endpointy:
        - POST /refresh: Umožňuje obnovit access token pomocí refresh tokenu (uloženého v HttpOnly cookie).

Kontroler slouží k opětovnému vystavení platného access tokenu, když ten stávající vyprší.

- AccountControllerV1
    - Závislosti:
        - IUserCredentialsService credentialsService pro správu uživatelských účtů (hesla, e-maily, uživatelská jména),
        - IJwtTokenService jwtTokenService pro práci s tokeny při odhlášení.
    - Hlavní endpointy:
        - POST / (logout): Odhlásí uživatele (zneplatní access i refresh token).
        - POST /password: Změní heslo.
        - POST /username: Změní uživatelské jméno (navíc znovu vygeneruje tokeny).
        - POST /email: Změní e-mail uživatele.

Poskytuje možnost správy vlastního účtu – odhlášení, změna hesla, uživatelského jména a e-mailu.

- InvoiceControllerV1
    - Závislosti:
        - IInvoiceService invoiceService pro byznys logiku faktur.
    - Hlavní endpointy:
        - GET /search: Pokročilé vyhledávání faktur (filtry, stránkování, řazení).
        - GET /: Seznam faktur (stránkovaný i nestránkovaný).
        - GET /{id}: Získání detailu faktury.
        - POST /: Vytvoření nové faktury (201 Created).
        - PUT /{id}: Úprava existující faktury.
        - DELETE /{id}: Smazání faktury.

Tento kontroler pokrývá CRUD operace pro faktury (invoices) a nabízí možnost vyhledávání a stránkování.

- PartyControllerV1
    - Závislosti:
        - IPartyService partyService pro správu „parties“ (např. zákazníků či jiných subjektů).
    - Hlavní endpointy:
        - GET /: Seznam „parties“, volitelně stránkovaný.
        - GET /{id}: Získání detailu jedné party. S anotací @IsOwnedByUser(entityClass=Party.class), což indikuje, že uživatel musí být vlastníkem.
        - POST /: Vytvoření nové party (201 Created).
        - PUT /{id}: Aktualizace existující party.
        - DELETE /{id}: Smazání party.

Slouží k správě „parties“ – typicky jde o kontakty nebo subjekty, které se mohou pojit k fakturám a dalším entitám v systému.

Balíček dto

Tento balíček disponuje předpisy objektů určených k přenosu dat mezi jednotlivými vrstvami či komponentami bez nutnosti odkrytí přímé implentace

- ChangeEmailRequestDto
- AuthenticationResponseDto
- ChangePasswordRequestDto
- ChangeUsernameRequestDto
- ChangeUsernameResponseDto
- CreatePasswordResetTokenDto
- InvoiceDto
- InvoiceItemDto
- InvoiceSearchDto
- LoginRequestDto
- PartyDto
- PartySnapshotDto
- RegistrationRequestDto
- ResetPasswordRequestDto
- TokenDto
- UserDto

Balíček model

zde se nacházejí třídy reprezentující databázové entity v rámci JPA/Hibernate. Každá třída je označena anotací @Entity, má definovaný primární klíč (typicky @Id) a potřebná mapování na tabulky/sloupce.

User

- Primární klíč: id typu Long (generován GenerationType.IDENTITY).
- Unikátní pole: username, email.
- Vztahy:
    - @OneToMany(mappedBy = "user") List<Party> parties – uživatel může mít více záznamů typu Party.
    - @OneToMany(mappedBy = "user") List<Invoice> invoices – uživatel může mít více faktur.
    - @OneToMany(mappedBy = "user") List<RefreshToken> refreshTokens – každý uživatel může mít přiřazeno více refresh tokenů.
    - @OneToOne(mappedBy = "user") PasswordResetToken passwordResetToken – uživatel může mít jeden token pro reset hesla.
- Implementace UserDetails: umožňuje použít Spring Security přímo nad entitou. Obsahuje pole jako accountNonExpired, accountNonLocked, credentialsNonExpired, enabled.
- Popis: Reprezentuje koncového uživatele aplikace. Uchovává přihlašovací údaje (username, email, password) a umožňuje vazby na související entity (faktury, parties, tokeny).

Invoice

- Primární klíč: id typu UUID (generován GenerationType.UUID).
- Vztahy:
    - @OneToMany(mappedBy = "invoice") Set<InvoiceItem> items – faktura může mít více položek (InvoiceItem).
    - @ManyToOne(fetch = FetchType.LAZY) Party customer – faktura má odkaz na „zákazníka“.
    - @ManyToOne(fetch = FetchType.LAZY) Party supplier – faktura má odkaz na „dodavatele“.
    - @ManyToOne User user – faktura je vytvořena konkrétním uživatelem.
- Denormalizovaná data: supplierName, supplierAddress, customerName, … a další pole pro snapshot původních údajů o dodavateli i zákaznícovi. Díky tomu se při změně informací o Party do budoucna nemění starší faktury.
- NamedEntityGraph: WITH_ITEMS_AND_PARTIES_GRAPH, usnadňuje fetch (načítání) položek items.

- Pole:
    - createdAt, expiresAt (časové údaje, TIMESTAMP WITH TIME ZONE),
    - paymentMethod, variableSymbol (byznysová logika platby),
    - contact (např. kontakt na faktuře),
    - supplierXxx a customerXxx (denormalizované sloupce).
- Popis: Faktura (Invoice) obsahuje sadu položek, odkaz na zákazníka a dodavatele (jak reálné entity Party, tak i snapshot pro historii). Propojena s uživatelem (vlastníkem).

InvoiceItem

- Primární klíč: id typu UUID (generován GenerationType.AUTO).
- Vztahy:
    - @ManyToOne Invoice invoice – každá položka faktury patří k jedné faktuře.
- Pole:
    - name: Název položky (např. „Produkt X“).
    - quantity: Množství (kusy).
    - unitPrice, taxPrice, totalPrice: Ceny (základní, daňová, celková).
- Popis: Reprezentuje jednu řádku faktury. Faktura se tak může skládat z více položek, které se sčítají do finální ceny.

Party

- Primární klíč: id typu UUID (generován GenerationType.UUID).
- Unique constraints: ic_tax + user_id, dic_tax + user_id (tzn. u jednoho uživatele nelze mít dvě parties se stejným IČ / DIČ).
- Vztahy:
    - @ManyToOne User user – každá party patří konkrétnímu uživateli (autorovi).
- Pole:
    - name, address, country, telephone, email (základní kontaktní údaje),
    - icTax, dicTax (IČ, DIČ).
- Popis: Entity typu Party představují subjekt (zákazník/dodavatel) – jméno, adresa a další identifikační údaje. Faktury i jiné oblasti mohou na tyto subjekty odkazovat (pokud se rozhodneme ukládat reference místo snapshotů).

PasswordResetToken

- Primární klíč: id typu Long (generován GenerationType.IDENTITY).
- Pole:
    - tokenHash: Hash tokenu pro ověření.
    - expiryDate: Datum/čas expirace (typ Instant).
- Vztahy:
    - @OneToOne(fetch = FetchType.LAZY) User user – každý token náleží konkrétnímu uživateli.
- Metody:
    - isExpired(Instant now): Zjišťuje, zda je token již propadlý oproti zadanému času.
- Popis: Slouží k obnovení hesla uživatele. Po vypršení expiryDate je token neplatný. Hash tokenu se ukládá do DB, aby se token nedal jednoduše odhadnout.

RefreshToken

- Reprezentuje refresh token pro uživatele (obvykle JWT token sloužící k vydávání nového access tokenu). Pokud je expirován, aplikace ho nemůže dál použít.
    - isExpired(): Kontroluje, zda nastala doba expirace.

Balíček security

zde se nacházejí komponenty a aspekty, které se starají o ověření vlastníka zdrojů (ownership), JWT autentizaci a další bezpečnostní logiku aplikace.

- Ownership Checking (kontrola vlastnictví zdroje)
  - IOwnershipChecker<T>
    - Rozhraní definující metody pro kontrolu, zda zdroj (entita) s daným ID existuje a zda je vlastněn konkrétním uživatelem.
    - Class<?> getEntityClass() vrací typ entity, které se kontrola týká (např. Invoice.class).
    - boolean resourceExists(T resourceId) určuje, zda entita s daným ID vůbec existuje.
    - boolean isOwnedBy(T resourceId, User user) kontroluje, zda entita s daným ID patří danému uživateli.

- InvoiceOwnershipChecker
  - Ověřuje, zda faktura (Invoice) patří přihlášenému uživateli.
  - resourceExists(UUID resourceId) → volá invoiceRepository.existsById(resourceId)
  - isOwnedBy(UUID resourceId, User user) → volá invoiceRepository.existsByIdAndUser(resourceId, user)

- PartyOwnershipChecker
  - Ověřuje, zda konkrétní Party patří přihlášenému uživateli
  - resourceExists(UUID resourceId) → volá partyRepository.existsById(resourceId)
  - isOwnedBy(UUID resourceId, User user) → volá partyRepository.existsByIdAndUser(resourceId, user)

- IsOwnedByUser
  - Slouží ke značkování endpointů, aby se na ně aplikovala logika kontroly vlastnictví (viz OwnershipAspect).

- OwnershipAspect
    - @Pointcut("@annotation(isOwnedByUser)") – definice bodu, kde se aspect spustí (metody s @IsOwnedByUser).
    - checkOwnership(JoinPoint jp, IsOwnedByUser isOwnedByUser) – před samotným voláním metody:
        - Získá přihlášeného uživatele z SecurityContextHolder.
        - Najde IOwnershipChecker pro danou entitu.
        - Z metody vyhledá ID parametru (resourceIdParam).
        - Ověří existenci entity (checker.resourceExists(...)). Pokud neexistuje, vyhodí NotFoundException.
        - Ověří vlastnictví (checker.isOwnedBy(...)). Pokud selže, vyhodí UnauthorizedException.

JWT autentizace

- JwtAuthenticationFilter
    - Zachytává příchozí HTTP požadavky, vyžadující autentizaci.
        - isExcluded(uri) – zkontroluje, zda URL patří mezi cesty, které nevyžadují autentizaci (např. /public/**).
        - Pokud není vyřazena, extrahuje token z hlavičky Authorization: Bearer
        - Ověří platnost tokenu (fetchDetailsIfAccessTokenValid). Pokud je neplatný, nastaví 401 a vrátí chybovou odpověď.
        - Vytvoří Spring Security kontext s uživatelem (UsernamePasswordAuthenticationToken).
        - Pokračuje v běhu filtru (zavolá filterChain.doFilter).
    - Chybové stavy:
        - Chybí token → 401 Unauthorized.
        - Expirace, neplatné credentialy → 403 Forbidden (pro některé stavy).

- JwtKeyProvider
    - Slouží k správě klíčů pro podepisování a ověřování tokenů.
        - getKey(JwtTokenType type) – vrací příslušný klíč podle typu tokenu (ACCESS nebo REFRESH).

- JwtTokenProvider

    - Slouží jako provider JWT tokenů pro generování a verifikaci
        - generateAccessToken
            - Vytvoří JWT s předem danou expirací (15 min) a claimem authorities.
            - Vrací TokenDto (obsahuje token a metadata).
        - generateRefreshToken
            - Vytvoří refresh token s delší platností (7 dnů). Claim rtid slouží jako identifikátor refresh tokenu v DB.
            - Přiřazuje se obvykle do cookie (HttpOnly) metodou attachRefreshTokenToCookie().
        - isTokenValid
            - Zkontroluje, zda token není expirovaný a zda username v tokenu odpovídá reálné existenci uživatele.

        - extractUsername, extractRefreshTokenId
            - Z tokenu vyčítají subject (= username) a rtid (ID tokenu) pomocí knihovny io.jsonwebtoken.Jwts.
        - attachRefreshTokenToCookie & invalidateRefreshTokenCookie
            - Metody pro nastavení / zneplatnění refresh tokenu v prohlížeči.

- JwtTokenType

    - Enum s hodnotami ACCESS a REFRESH, pomocí kterého se rozlišuje, jaký klíč a jaká expirace se má použít.

- CustomUserDetailsService
    - Poskytuje uživatelské údaje (UserDetails) pro Spring Security na základě username.
        - loadUserByUsername(String username)
            - Volá userRepository.findByUsername(username).
            - Pokud uživatel s daným username neexistuje, vyhodí UsernameNotFoundException.
            - V opačném případě vrátí entitu User (implementuje UserDetails).

Balíček service

V tomto balíčku jsou umístěny byznysové služby aplikace. Každá služba obsahuje logiku, která se odvolává na úložiště (repository), mapovací vrstvy (mapper) či bezpečnostní komponenty (např. tvorba tokenů). Některé služby implementují rozhraní typu IAuthService, IInvoiceService atp., což usnadňuje dohledatelnost a testovatelnost.

AuthService

- Zajišťuje přihlášení a registraci uživatelů, včetně tvorby a správy access/refresh tokenů
    - Login (LoginRequestDto login, HttpServletResponse response):
        - Najde uživatele dle username, ověří heslo.
        - Při úspěchu vytvoří refresh token (uloží ho do cookie) a vygeneruje access token (vrací v JSON).
        - Pokud je cokoliv neplatné, vyhodí UnauthorizedException.
    - Register (RegistrationRequestDto registration, HttpServletResponse response):
        - Ověří, zda uživatelem zadaný username nebo email již neexistuje. Pokud ano, ConflictException.
        - Vytvoří nového uživatele (se zahashovaným heslem), uloží do DB.
        - Stejně jako login nastaví refresh token do cookie a vrátí access token.

InvoiceItemService

- Poskytuje služby nad položkami faktury, typicky pro listing, detail a validaci přislušnosti k faktuře
    - findForInvoice(Invoice invoice, Pageable pageable) a findForInvoice(Invoice invoice):
        - Najde všechny položky faktury (InvoiceItem) pro danou fakturu, případně stránkovaně (Page).
        - Mapuje je na InvoiceItemDto.
    - findInvoiceItemByIdForInvoice(UUID id, Invoice invoice):
        - Vyhledá konkrétní položku faktury podle ID a ověří, že patří příslušné faktuře.
        - Pokud neexistuje, vyhodí NotFoundException.

InvoiceService

- Hlavní byznysová logika faktur(search, CRUD, validace vlastnictví mapování)
    - Search (User user, InvoiceSearchDto criteria, Pageable pageable, Sort sort)
        - Vytváří Specification<Invoice> na základě kritérií (InvoiceSearchDto) a uživatele.
        - Vrací stránkovaný seznam nalezených faktur (převedených na DTO).
    - FindForUser (User user, Pageable pageable) či findForUser(User user)
        - Vrátí všechny faktury (event. stránkované) pro zadaného uživatele.
    - CRUD:
    - FindInvoiceByIdForUser (UUID id, User user) – detail faktury, ověřuje vlastnictví (@IsOwnedByUser(entityClass=Invoice.class)).
    - CreateInvoice (InvoiceDto invoiceDto, User user) – vytvoření nové faktury.
        - Denormalizuje údaje o Party do snapshotu (viz. snapshotSupplier(), snapshotCustomer()).
        - Uloží sadu položek InvoiceItem.
    - UpdateInvoice (UUID id, InvoiceDto invoiceUpdated, User user) – úprava existující faktury.
        - Ověří vlastnictví, pak aplikuje změny v polích.
        - deleteInvoice(UUID id) – smaže fakturu (opět s ověřením vlastnictví přes @IsOwnedByUser).
    - HandlePartyUpdate(...): Pomocná metoda, která se stará o vyhledání či vytvoření Party a snapshotování údajů do faktury.

JwtTokenService

- Správa refresh tokenů (ověřování, generování, mazání z DB a navazující akce)
    - refresh(HttpServletRequest request) – na základě refresh tokenu uloženého v cookie vrátí nový access token.
    - refreshAndInvalidate(String username, HttpServletRequest request, HttpServletResponse response) – pro uživatele, který mění username: starý refresh token se invaliduje a uloží se cookie s novým.
    - logout(User user, HttpServletRequest request, HttpServletResponse response) – zneplatní refresh token (odstraní z DB) a vymaže cookie (invalidateRefreshTokenCookie).
    - generateNewRefreshToken(User user) – vytvoří nový refresh token (v DB i JWT).

PartyService

- CRUD operace nad entitou Party (subjekt – zákazník, dodavatel), včetně validace příslušnosti k uživateli (většinou prováděno přes rest kontrolery s anotací @IsOwnedByUser).
    - createParty(PartyDto partyDto, User user) – vytvoří novou party, naváže na uživatele a uloží do DB.
    - updateParty(PartyDto partyUpdated, User user) – úprava existující party (dle ID a vlastníka).
    - deleteParty(UUID id) – odstraní existující party.
    - findById(UUID id) – najde party podle ID (bez ohledu na vlastníka).
    - findForUser(User user, Pageable pageable) – vyhledá všechny party daného uživatele (s podporou stránkování).

UserCredentialsService

- Správa uživatelských údajů (heslo, username, e-mail)
- Password reset a veškerá logika okolo bezpečného generování a mazání reset tokenů
    - createAndSendPasswordResetToken(String username):
        - Vygeneruje novou tokenovou dvojici (token + hash) a uloží do PasswordResetToken.
        - Odešle e-mail uživateli s tímto tokenem.
    - resetPassword(String token, String newPassword):
        - Ověří existenci tokenu, jeho neexpiraci a vlastnictví.
        - Nastaví nové heslo uživateli.
        - Odstraní reset token z DB (one-to-one vazba).
    - changePassword(String username, String oldPassword, String newPassword):
        - Načte uživatele a zkontroluje staré heslo.
        - Nastaví nové heslo.
    - changeUsername(String username, String newUsername) & changeEmail(String username, String newEmail):
        - Ověří existenci starého uživatele, dostupnost nového username/e-mailu.
        - Uloží změny do DB.

UserService

- Jednodušší CRUD a dotazy nad entitou User, oddělené od složitější logiky v AuthService nebo UserCredentialsService
    - findUserDtoById(Long id) a findUserById(Long id):
        - Vyhledá uživatele podle ID, pokud neexistuje, vyhodí NotFoundException.
        - Vrací buďto UserDto (DTO) nebo entitu User.

- InvoiceSpecification
    - Zapouzdření podmínky pro vyhledávání faktur (Invoice). Lze je kombinovat do komplexních dotazů pomocí metod Specification.and(...) a Specification.or(...).

    - hasId(UUID id)
        - Vytvoří podmínku: invoice.id == id.
        - Využitelné při filtrování na konkrétní ID faktury.

    - hasVariableSymbol(String variableSymbol)
        - Filtrovaná faktura má variableSymbol rovný zadané hodnotě.

    - hasSupplierNameLike(String supplierName)
        - Vyhledává faktury, jejichž supplierName (dodavatel) obsahuje daný řetězec (case-insensitive).

    - hasCustomerNameLike(String customerName)
        - Vyhledává faktury, jejichž customerName (zákazník) obsahuje daný řetězec (case-insensitive).

    - hasPaymentMethod(PaymentMethod paymentMethod)
        - Filtrovaná faktura má danou hodnotu PaymentMethod.

    - createdBetween(Instant startDate, Instant endDate)
        - Filtrovaná faktura musí mít createdAt v zadaném rozmezí (včetně hranic).
        - Podporuje dílčí podmínky: pokud startDate je null, dolní hranice se ignoruje; pokud endDate je null, horní hranice se ignoruje.

    - expiresBetween(Instant startDate, Instant endDate)
        - Obdobné jako createdBetween, ale aplikuje se na pole expiresAt.
        - Lze tak filtrovat faktury dle data expirace.

    - hasContactLike(String contact)
        - Filtrovaná faktura má contact (např. telefon nebo jiný kontaktní údaj) obsahující daný řetězec (case-insensitive).

    - anyItemNamesLike(List<String> itemNames)
        - Rozšiřuje dotaz o join na items (kolekce InvoiceItem) a vyhledává, zda alespoň jedna položka obsahuje řetězec z předaného seznamu (používá se OR mezi zadanými jmény).
        - Metoda zároveň nastaví query.distinct(true), aby se vrátily pouze unikátní faktury