Projektová dokumentace (DSP)

# Abstrakt

Samostatně vyvíjím webovou aplikaci banky v prostředí Django jako semestrální projekt pro předmět STIN. Cílem mé práce je vytvořit online platformu pro správu finančních prostředků a zákaznických účtů.

Můj projekt zahrnuje implementaci funkcionalit, jako jsou vytváření a správa účtů, ověřování identity, zpracování plateb a převodů mezi účty, historie transakcí a další.

V rámci dokumentace mé práce se zaměřuji na různé aspekty webové aplikace, jako jsou návrh databáze, architektura aplikace, použité technologie, zabezpečení, testování a další.

Věřím, že tato práce mi pomůže zlepšit své schopnosti v oblasti programování a návrhu softwarových aplikací a zároveň mi poskytne cenné zkušenosti v oblasti finančních systémů.

# Development stack

Všechny části této webové aplikace jsou implementovány v jazyce **Python**, který je využíván pro psaní backendového kódu. **Django** framework nám umožňuje jednoduše vytvářet efektivní a robustní webové aplikace a poskytuje nám mnoho užitečných funkcí, jako například ORM pro komunikaci s databází, automatické generování administrátorského rozhraní nebo zabudovanou podporu pro šablony a statické soubory.

Pro ukládání dat využívám integrovanou **SQLite databázi**, kterou můžu snadno komunikovat pomocí ORM. Pro background processing používám kombinaci technologií **Celery a RabbitMQ**, kde RabbitMQ slouží jako message queue pro tasky, které jsou následně zpracovávány pomocí Celery. Jedním z nejdůležitějších background procesů, které obstarává Celery, je fetchování směnných kurzů z České Národní Banky za pomoci jejich veřejného API. Tento workflow mi umožňuje efektivně zpracovávat úlohy na pozadí, což je důležité pro rychlost a škálovatelnost aplikace.

Na straně frontendu využívám CSS framework Bulma, který mi umožňuje snadno vytvářet responzivní design a rychle generovat uživatelské rozhraní. Jelikož tato webová aplikace využívá technologii SSR (Server Side Rendering), není třeba žádného frontendového frameworku jako React, VueJS, Angular apod. Stránka přijde ze serveru již vyrenderována, a jakékoliv dodatečné frontent UI/UX je obstaráno obyčejným klientským JavaScriptem.

Celkově tedy používám kombinaci moderních technologií a nástrojů, které mi umožňují efektivně vytvářet a provozovat webové aplikace. Použití Pythonu jako hlavního jazyka mi poskytuje silnou základnu pro psaní robustního backendového kódu, zatímco využití moderních technologií jako Celery a RabbitMQ mi umožňuje zpracovávat úlohy na pozadí a poskytovat uživatelům rychlé a plynulé uživatelské rozhraní.

# Autentizace

Django v sobě obsahuje vynikající funkce pro zajištění autentizace a autorizace v naší aplikaci. Když se uživatel pokouší přistoupit k určité stránce, Django kontroluje, zda je uživatel přihlášen a zda má oprávnění k přístupu k této stránce.

Pokud uživatel není přihlášen, Django automaticky přesměruje uživatele na stránku s přihlašovacím formulářem. Pokud uživatel zadá správné přihlašovací údaje, Django ho přihlásí a umožní mu přistoupit k požadovaným stránkám.

Pokud uživatel má již účet, ale nemá potřebná oprávnění k přístupu k určité stránce, Django ho přesměruje na stránku s chybovou zprávou.

Za tuto funkcionalitu v aplikaci jsou zodpovědné různé komponenty Django, jako například middleware. Middleware jsou komponenty, které se provádějí před nebo po zpracování požadavku v aplikaci. Můžu tedy vytvořit middleware, který se bude provádět před každým požadavkem a bude kontrolovat, zda je uživatel přihlášen a zda má oprávnění k přístupu k požadovanému view.

Také můžu využít Django view dispatch mechanismus, který umožňuje definovat metodu, která se bude provádět před zpracováním požadavku na daný view. Tato metoda může provést různé operace, například kontrolu oprávnění, přihlášení uživatele, nebo jiné operace spojené s autentizací a autorizací.

Jednou z podmínek pro zápočet byla také implementace dvoufázového ověření – nehledě na to, jaký typ si student zvolí. Jelikož se mi nechtělo patlat s SMTP a emaily, zvolil jsem populární 2FA metodu – TOTP.

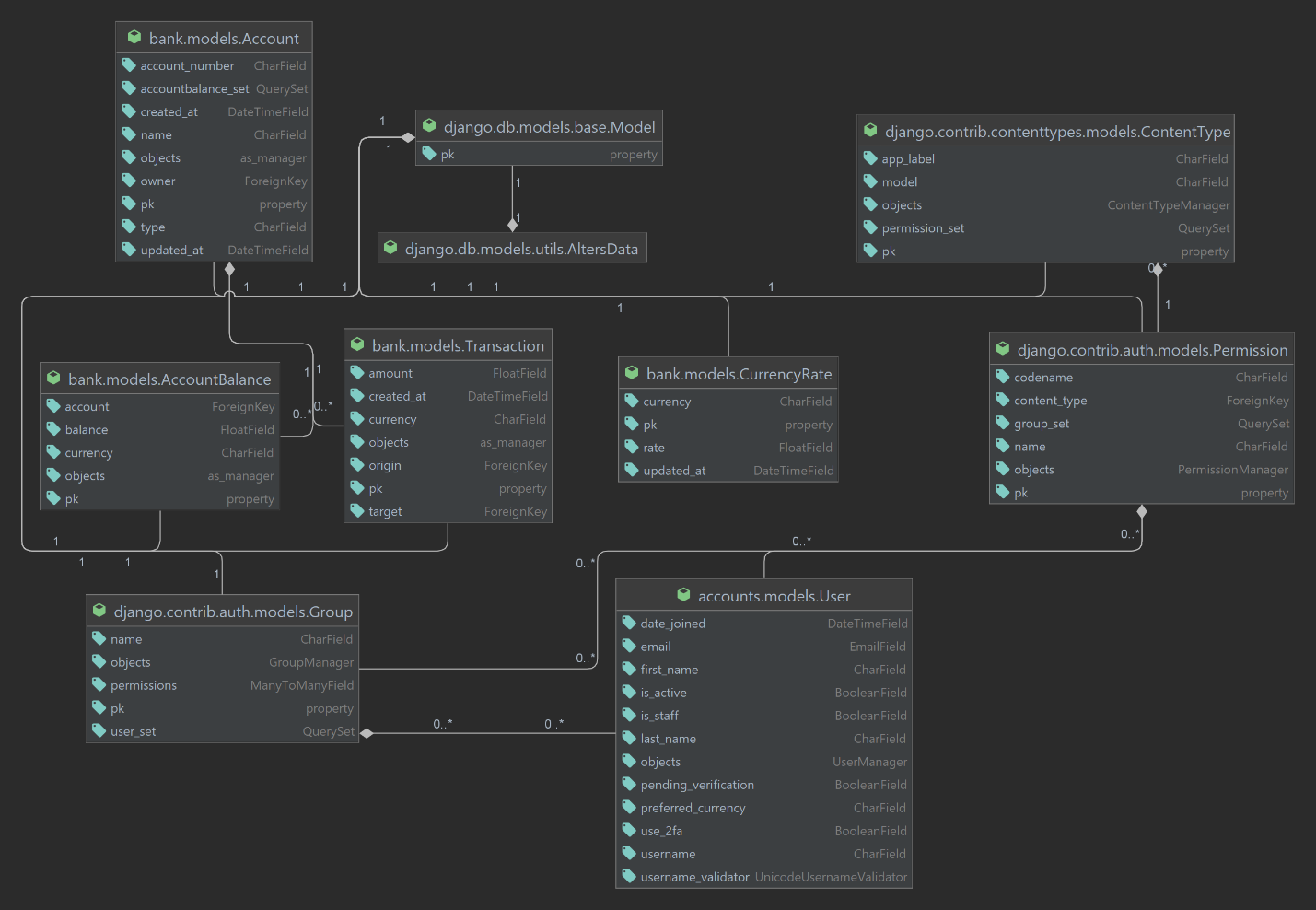
S využitím knihovny pyotp a qrcode můžeme snadno implementovat dvoufaktorovou autentizaci v naší aplikaci. PyOTP umožňuje generovat a ověřovat jednorázové kódy pro autentizaci a Qrcode knihovna zase umožňuje generování QR kódů pro zobrazení a snadné skenování v aplikaci Google Authenticator.

Implementace byla nadmíru jednoduchá – určitě jednodušší než posílat ověřovací kódy na email. Pokud bych to dělal znovu, myslím si, že bych to mohl mít do 20 minut hotové, tak jednoduché to je. Pro implementaci stačí pouze natahat již zmíněné knihovny – dodat secret key, který bude použit pro generování QR kódů, uživatelovo jméno a jméno naší aplikace (**issuer name**), a knihovna vyprdne nazpátek OTP URI. Tato OTP URI má určitý formát (více [zde](https://github.com/google/google-authenticator/wiki/Key-Uri-Format)), který lze snadno uložit do QR kódu, který je pro uživatele mnohem jednodušší na použití.

Poté bychom museli vytvořit funkci pro ověření jednorázového kódu. Tuto funkci můžeme také zavolat v našem view, který se stará o autentizaci uživatele. Funkce by pak ověřila, zda zadaný kód odpovídá jednorázovému kódu, který jsme vygenerovali při vytváření QR kódu. Pokud kódy odpovídají, uživatel by mohl pokračovat v přihlašování do aplikace.

Celkově tedy Django poskytuje robustní zabezpečení pro naši aplikaci a umožňuje nám snadno definovat oprávnění pro uživatele, kontrolu přístupu a mnoho dalších funkcí spojených s autentizací a autorizací.

# Databáze

Jak jsem již výše zmiňoval, pro ukládání dat používám vestavěnou SQLite databázi. V Djangu se s ní komunikuje za pomoci ORM. Vývojář definuje takzvané *Modely*, řekne Djangu, že se má struktura databáze změnit, a Django – za pomoci tzv. migrací, které se vytvářejí po spuštění příkazu *python manage.py makemigrations* – upraví strukturu databáze. Já se tedy nemusím vrtat přímo v SQL, stačí mi pouze nadefinovat třídy a atributy, které korespondují k entitám a jejich atributům. Níže je vidět celá struktura hlavních modelů, díky kterým je chod banky možný.

# Testování