ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Semestrálna práca z predmetu *jazyk Rust*

FILAMENTY

vypracoval: Peter Cyprich
študijná skupina: 5ZYS33
cvičiaci: Ing. Peter Sedláček, PhD.
termín cvičenia: štvrtok 18:00

v Žiline dňa 11.1.2026

Obsah

1. Popis projektu 3
2. UML Diagramy 5
3. Zoznam použitých knižníc 10
4. Používateľská príručka 11
5. Programátorská príručka 18
6. Záver 22
7. Použité zdroje 23

Zoznam obrázkov

Obrázok 1 - Diagram - organizácia projektu 5
Obrázok 2 - Diagram - štruktúry 1 6
Obrázok 3 - Diagram - štruktúry 2 7
Obrázok 4 - Diagram - štruktúry 3 8
Obrázok 5 - Diagram enumov 8
Obrázok 6 - Diagram - databáza 9
Obrázok 7 - Aplikácia - úvod 12
Obrázok 8 - Aplikácia - materiály 12
Obrázok 9 - Aplikácia - vytvorené materiály 13
Obrázok 10 - Aplikácia - vytvorení výrobcovia 14
Obrázok 11 - Aplikácia - produkty 14
Obrázok 12 - Aplikácia - nový produkt 14
Obrázok 13 - Aplikácia - vytvorený produkt 15
Obrázok 14 - Aplikácia - nový filament 15
Obrázok 15 - Aplikácia - vytvorený filament 16
Obrázok 16 - Aplikácia - ukážka filamentov 16
Obrázok 17 - Aplikácia - ukážka štítkov 17

1. Popis projektu

Mojim cieľom bolo vytvorenie fullstack webovej aplikácie na správu filamentov do 3D tlačiarne. Aplikácia umožňuje ukladať údaje o filamentoch, výrobcoch, teplotách tlače a ďalších súvisiacich údajoch.

Aplikácia je navrhnutá ako fullstack riešenie pozostávajúce z troch hlavných častí

1. Backend
   - Implementovaný v jazyku Rust
   - Pomocou knižnice Actix-web poskytuje REST API
   - Využíva Tokio runtime na asynchrónne volania
   - Komunikuje s databázou pomocou knižnice SQLx
   - Rozdelený do dvoch bedničiek
     - *app* – spustiteľný balíček s REST API
     - *lib* – knižnica na prácu s databázou a pomocné funkcie
2. Frontend
   - Webové používateľské rozhranie vytvorené s pomocou knižnice React
   - Komunikuje s backend-om cez REST API – umožňuje zobrazovať, upravovať, pridávať a mazať dáta
   - Frontend nie je predmetom tejto semestrálnej práce, preto ho ďalej nebudem popisovať do detailov
3. Databáza
   - PostgreSQL databáza slúži ako perzistentné úložisko pre všetky údaje o filamentoch a súvisiacich údajoch
   - Databáza je normalizovaná do 3NF

Aplikácia pracuje so štyrmi základnými entitami

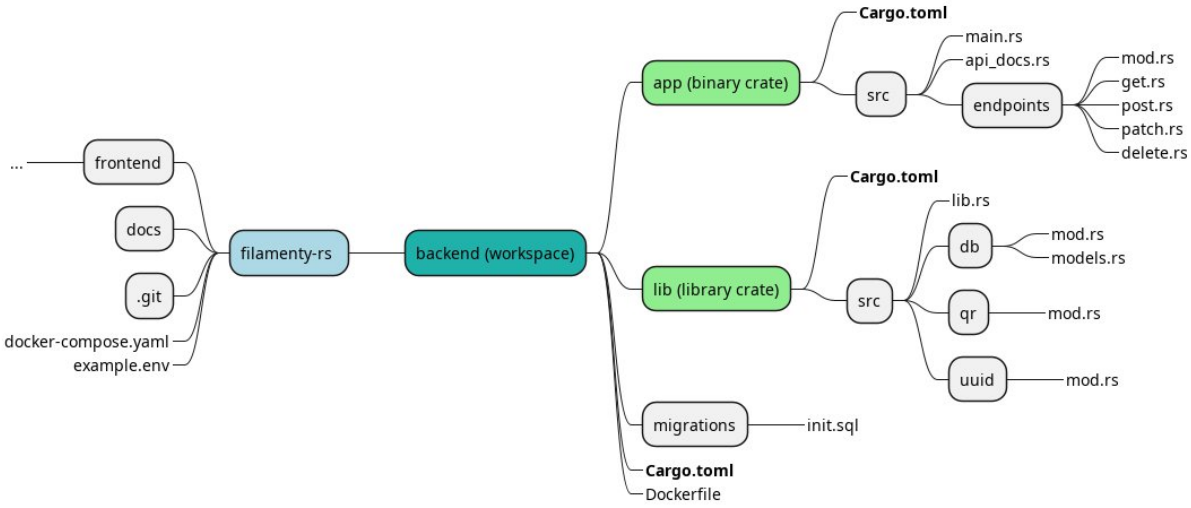
- Výrobca (Vendor) – reprezentuje výrobcu filamentov – napr. “Prusa Research“ alebo “Bambu Lab“
- Materiál (Material) – reprezentuje typ materiálu filamentov – napr. “PLA“, “ABS“, ...
- Produkt (Product) – reprezentuje typ, resp. produktovú radu filamentu. Zahŕňa informácie o výrobcovi, materiáli, teplotách tlače a o priemere vlákna. Príkladom by mohol byť “Prusament PLA“ alebo “Bambu Lab PLA Basic“. Analogicky by sa táto entita dala prirovnať k modelu mobilného telefónu – napr. iPhone 15

- Filament – reprezentuje konkrétnu cievku filamentu, jeden konkrétny výrobok z produktovej rady. Zahŕňa informácie o produkte, cene, farbe a hmotnosti. Príklad - “Bambu Lab PLA Basic Yellow 1KG“. Analogicky môžeme prirovnať ku konkrétnemu mobilnému telefónu – iPhone 15 128GB black

Zdrojový kód je dostupný na adrese https://github.com/cyprich/filamenty-rs

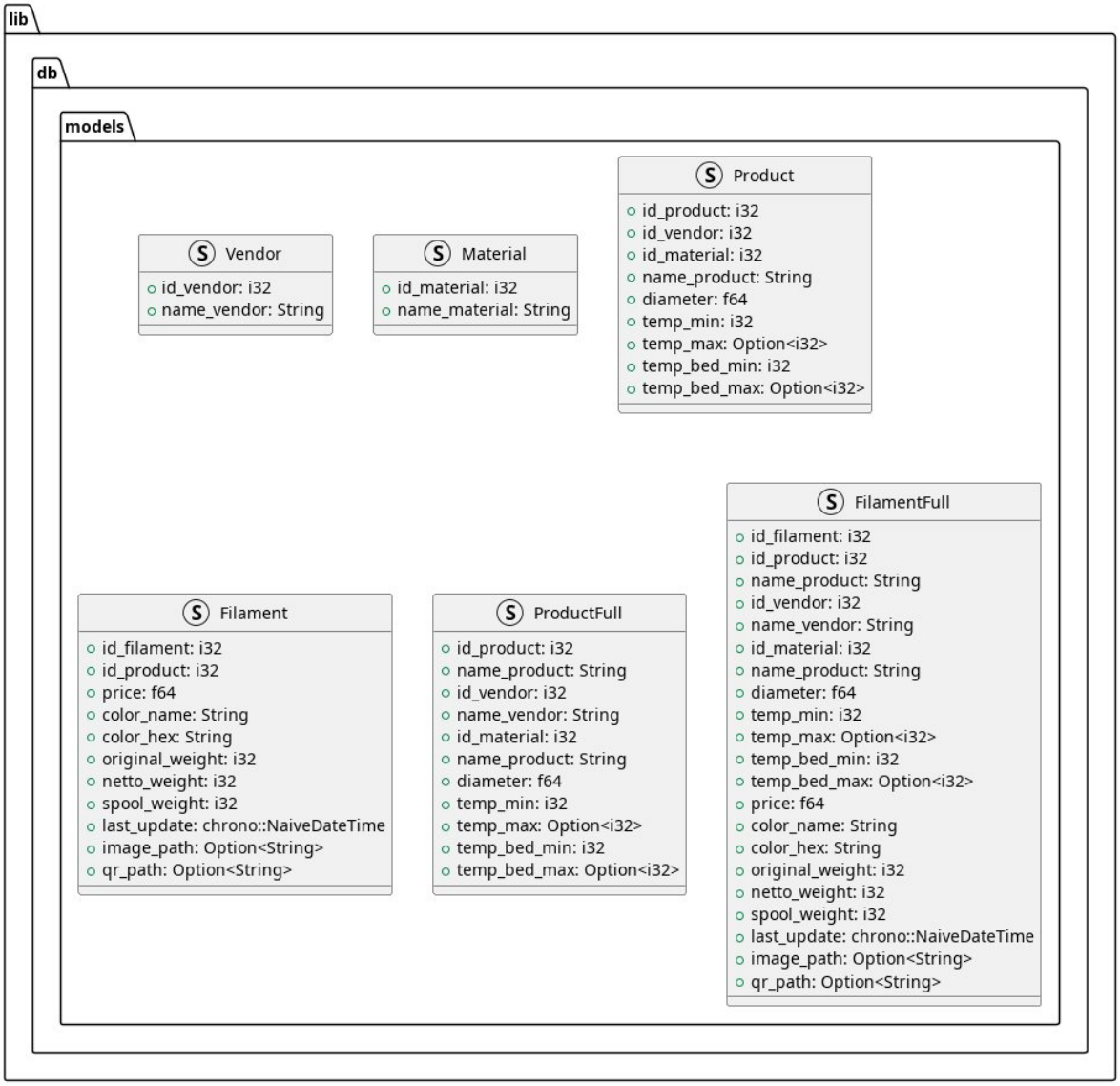
2. UML Diagramy

Ako už bolo spomenuté, projekt je rozdelený na frontend a backend, pričom backend je organizovaný ako Rust workspace s dvoma bedničkami. Celá štruktúra projektu je znázornená na nasledovnom diagrame. Zvýraznené sú dôležité časti – modrou je koreň projektu, tyrkysovou je Rust workspace a zelenou sú bedničky.



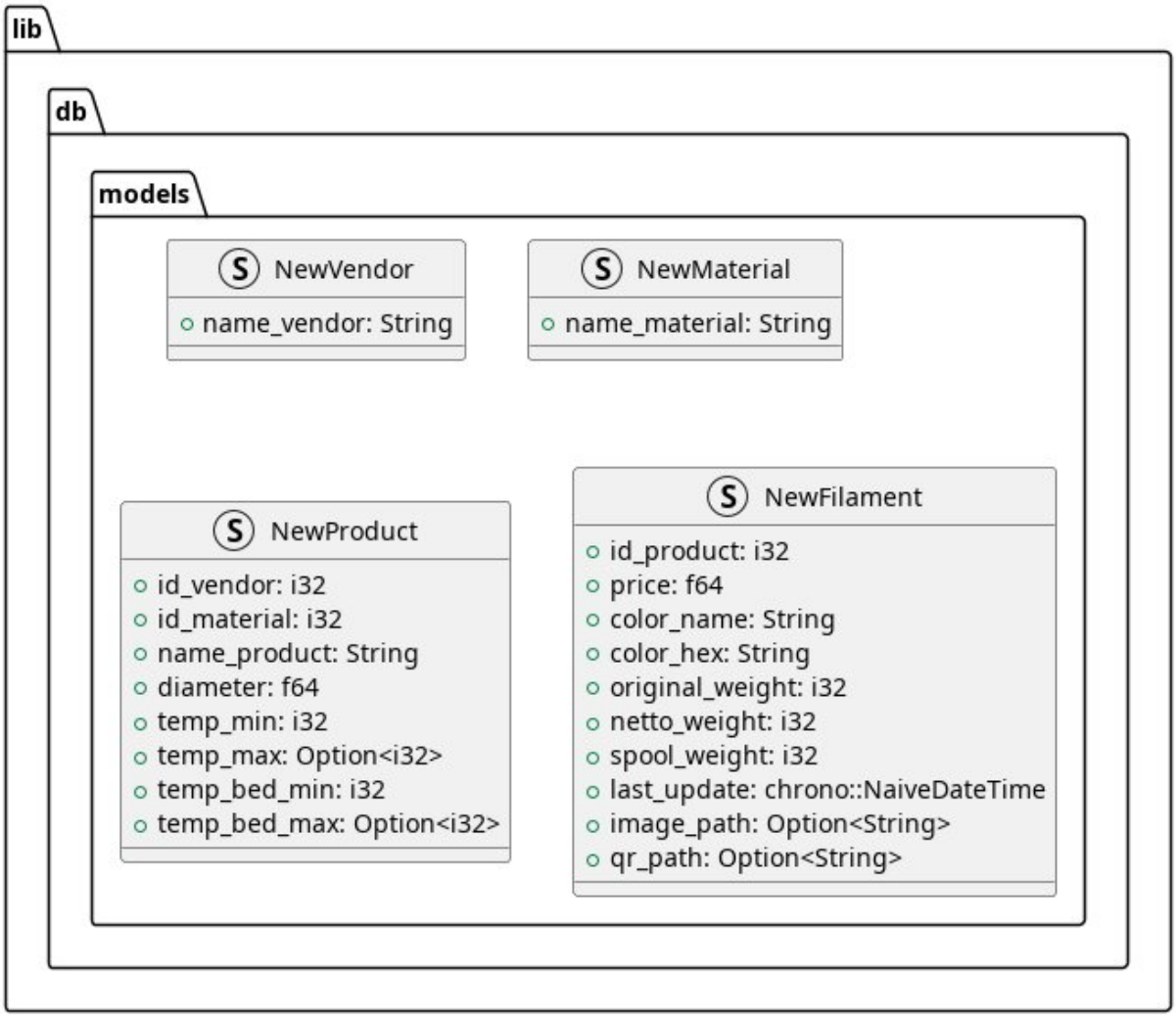
Obrázok 1 - Diagram - organizácia projektu

V súbore *lib/src/db/models.rs* sa nachádzajú štruktúry reprezentujúce výsledky databázových dotazov. Štruktúry *Vendor, Material, Product* a *Filament* priamo reprezentujú príslušné tabuľky v databáze. Štruktúry *ProductFull* a *FilamentFull* reprezentujú výsledky databázových dotazov, ktoré spájajú viaceré tabuľky (JOIN). Všetky tieto štruktúry derivujú trait *sqlx::FromRow* (pre automatické mapovanie riadku výsledku SQL dotazu na danú štruktúru), *serde::Serialize* a *serde::Deserialize* (pre serializáciu s deserializáciu štruktúr), *utoipa::ToSchema* (pre dokumentáciu OpenAPI) a *Debug* (pre pomocné výpisy)



Obrázok 2 - Diagram - štruktúry 1

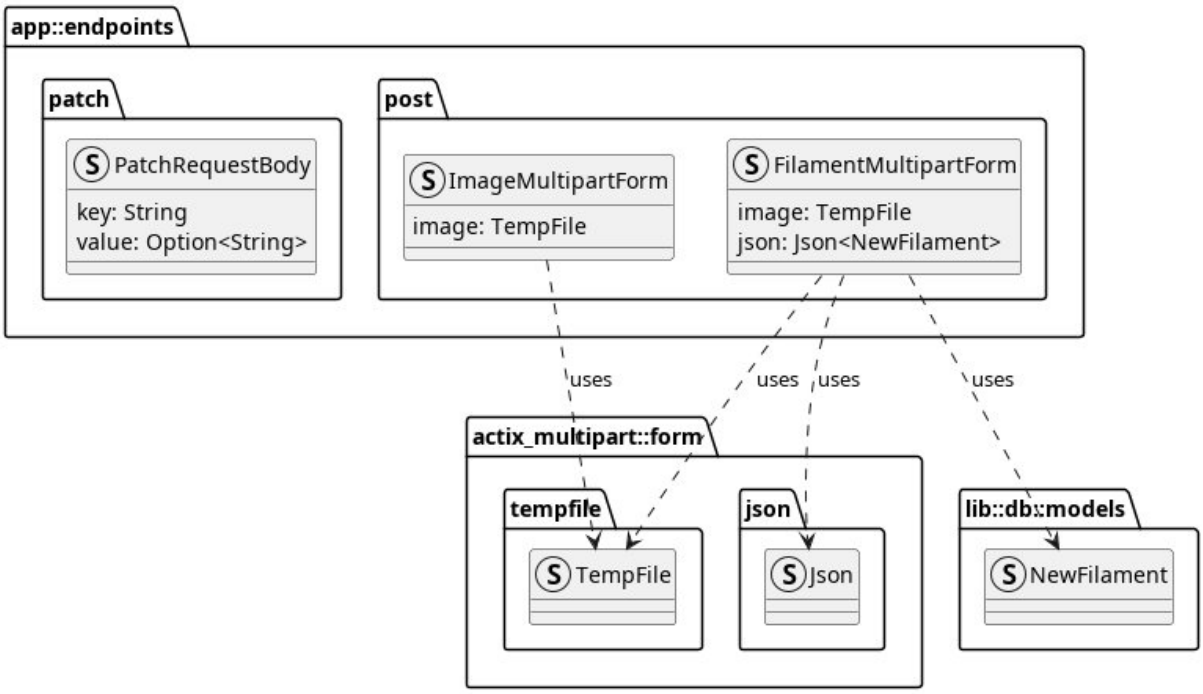
V tom istom súbore sa nachádzajú aj štruktúry *NewVendor*, *NewMaterial*, *NewProduct* a *NewFilament*, ktoré sa používajú na vkladanie dát do tabuliek databázy. Jediný rozdiel oproti predchádzajúcim štruktúram je ten, že tieto štruktúry neobsahujú primárne kľúče tabuliek, pretože tie automaticky generuje databáza (typ *serial* v PostgreSQL)



Obrázok 3 - Diagram - štruktúry 2

V bedničke *app* sú ďalšie pomocné štruktúry

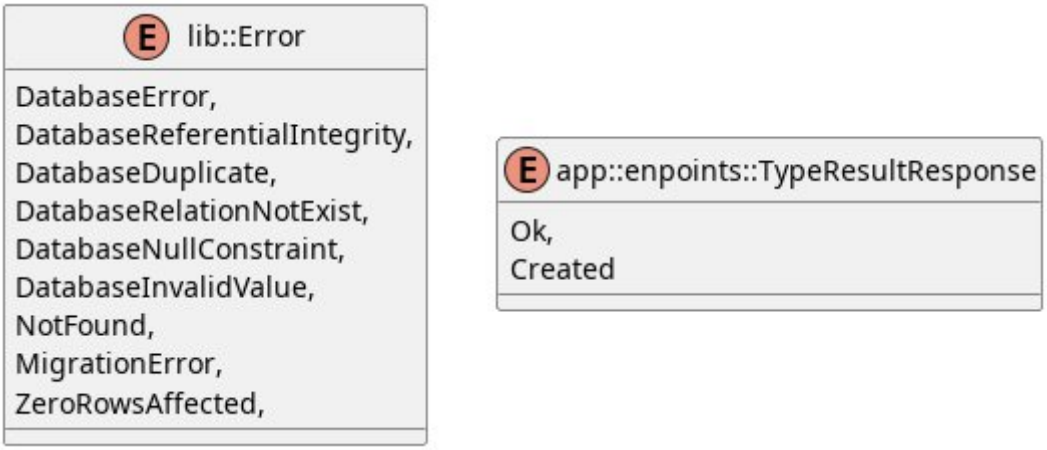
- Modul *endpoints::patch* používa štruktúru *PatchRequestBody* definovanú v súbore *endpoints/patch.rs*. Táto štruktúra reprezentuje telo PATCH požiadaviek. Obsahuje názov stĺpca databázy (*key*) a novú hodnotu (*value*)
- Modul *endpoints::post* používa dve štruktúry – *ImageMultipartForm* a *FilamentMultipartForm*. Obe tieto štruktúry sa používajú pri požiadavkách typu POST, ktoré obsahujú obrázok – teda endpointy */api/v2/images* a */api/v2/filaments-with-image* s metódami typu POST. Tieto endpointy prijímajú požiadavky s formátom *multipart/form-data*. Tieto štruktúry (a teda aj ich príslušné endpointy) obsahujú tieto polia
  - *ImageMultipartForm* – jedno pole s názvom *image* typu *TempFile*
  - *FilamentMultipartForm* – pole *image* typu *TempFile* a pole *json* s typu *Json<NewFilament>*



Obrázok 4 - Diagram - štruktúry 3

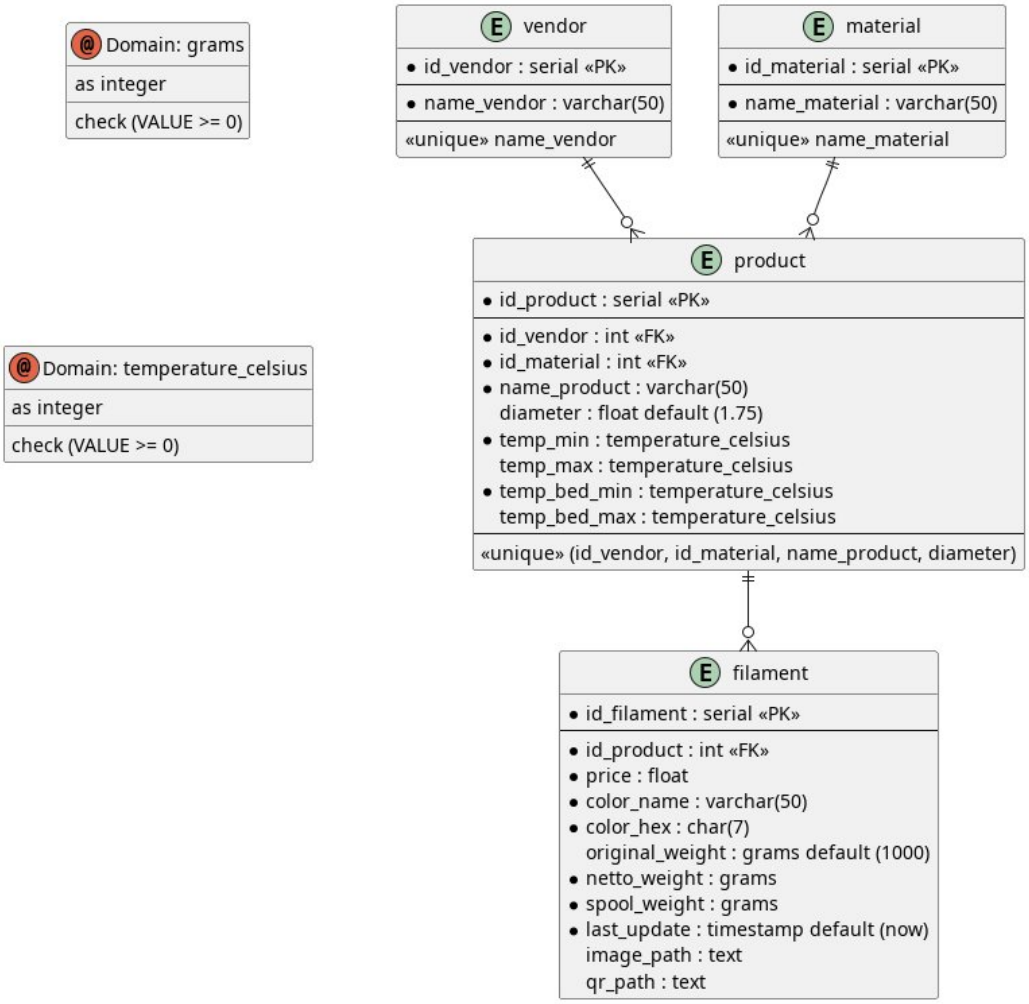
V projekte sa nachádzajú dva enumy

- *lib::Error* – používa sa v *lib/src/db/mod.rs*, kde sa naň mapuje typ *sqlx::Error*. Následne sa v module *app::endpoints* pri chybnej HTTP požiadavke zašle vhodný chybový kód na základe typu tohto enumu
- *app::endpoints::TypeResultResponse* – pomocný enum pre funkciu *app::endpoints::handle_type_result*. Táto funkcia sa využíva vo viacerých endpointoch, pričom niekedy je vhodné vrátiť kód 200 a niekedy 201, pričom tento enum určuje ktorý z nich bude použitý



Obrázok 5 - Diagram enumov

Databáza obsahuje štyri tabuľky – *material*, *vendor*, *product*, *filament* (a tabuľku *_sqlx_migrations*, ktorú automaticky generuje SQLx). Jednotlivé polia tabuliek sú znázornené v diagrame nižšie. Databáza je normalizovaná do 3NF, primárne kľúče sú automaticky generované.



Značka	Vysvetlenie
●	Povinné pole (not null)
«PK»	Primárny kľúč
«FK»	Cudzí kľúč
«unique»	Položka musí byť jedinečná

Obrázok 6 - Diagram - databáza

3. Zoznam použitých knižníc

Bednička *app*

- *actix-web* – implementácia REST API a spracovanie HTTP požiadaviek
- *actix-cors* – definovanie pravidiel CORS (Cross-Origin Resource Sharing) pre prístup z iných lokalít
- *actix-files* – obsluha (odosielanie) statických súborov – obrázkov filamentov a QR kódov
- *actix-multipart* – spracovanie (prijatie) obrázkov od používateľa
- *serde* – serializácia a deserializácia Rust štruktúr pre komunikáciu s frontendom a databázou
- *utoipa* – generácia OpenAPI dokumentácie
- *utoipa-swagger-ui* – vizualizácia OpenAPI dokumentácie

Bednička *lib*

- *chrono* – manipulácia s dátumom a časom
- *dotenvy* – načítanie premenných prostredia do programu
- *qrcode* – generovanie obrázkov QR kódov
- *image* – ukladanie vygenerovaných obrázkov QR kódov
- *serde* – serializácia a deserializácia Rust štruktúr pre komunikáciu s databázou
- *sqlx* – komunikácia s databázou
- *uuid* – generovanie unikátnych ID pre názvy obrázkov
- *utoipa* – derivácia traitov pri niektorých štruktúrach, potrebné pre generovanie dokumentácie OpenAPI

4. Používateľská príručka

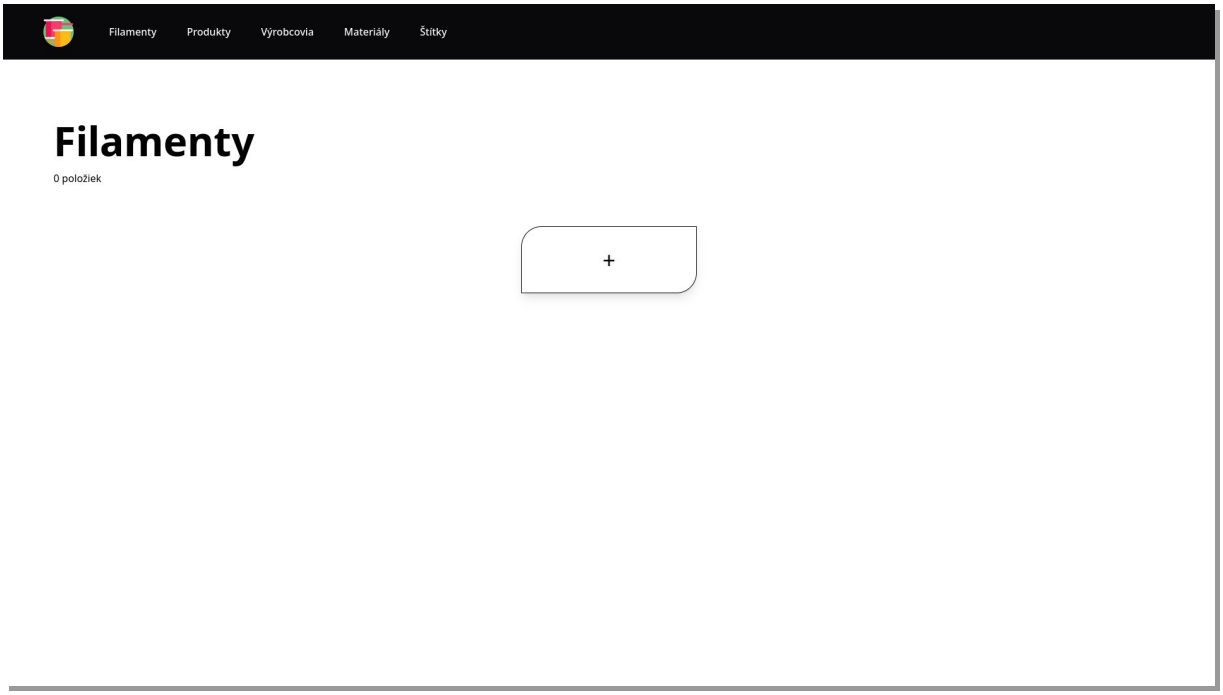
Požiadavky na spustenie – *git*, *docker* a *docker compose*

Spustenie aplikácie

1. Naklonujte git repozitár so zdrojovými súbormi (*git clone https://github.com/cyprich/filamenty-rs*) a presuňte sa do tohto priečinku
2. Premenujte súbor *example.env* na *.env*
3. Podľa potreby môžete upraviť niektoré hodnoty v súbore *.env*
   - *POSTGRES_USER*, *POSTGRES_PASSWORD*, *POSTGRES_DB* sú prihlasovacie údaje do databázy. Tieto hodnoty nie je potrebné meniť
   - *HOST* je doménové meno alebo IP adresa zariadenia, kde beží aplikácia. Ak si prajete, aby bola aplikácia dostupná pre ostatné zariadenia v lokálnej sieti, môžete túto hodnotu zmeniť na IP adresu vášho zariadenia. Túto hodnotu nemusíte meniť ak budete k aplikácií pristupovať iba zo zariadenia, na ktorom je aplikácia spustená
   - *FRONTEND_PORT*, *BACKEND_PORT* sú sieťové porty frontendu a backendu. Tieto hodnoty nemusíte meniť, ak na vašom zariadení nebežia iné služby používajúce tieto porty
4. Spustite aplikáciu príkazom *docker compose up -d*. Prvé spustenie môže trvať niekoľko minút. Ak upravíte nejaké hodnoty v súbore *.env*, tak je potrebné spustiť aplikáciu príkazom *docker compose up -d --build*
5. Po spustení bude aplikácia dostupná na predvolenej adrese, napr. *http://localhost*
6. Aplikácia bude bežať na pozadí až kým nezadáte príkaz *docker compose down* v priečinku s naklonovaným repozitárom

Používanie aplikácie

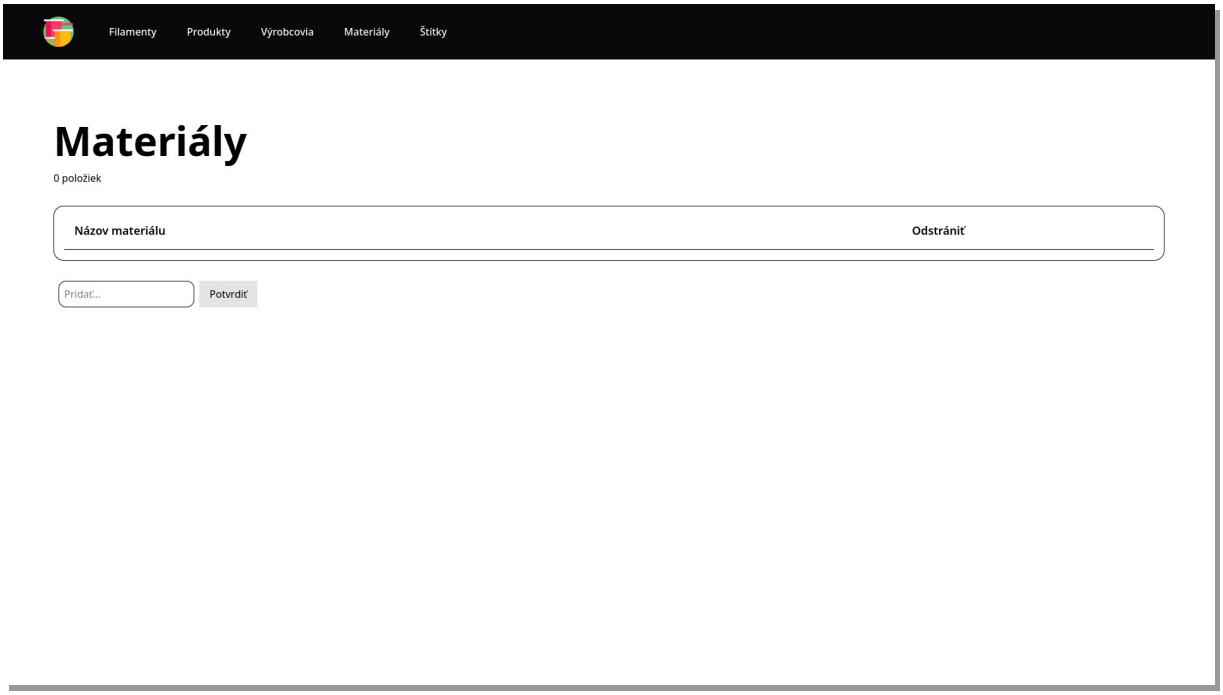
Po spustení aplikácie, a otvorení príslušného linku (napr. *http://localhost*) vo webovom prehliadači môžeme vidieť obrazovku s filamentami, ktorá bude na začiatku prázdna



Obrázok 7 - Aplikácia - úvod

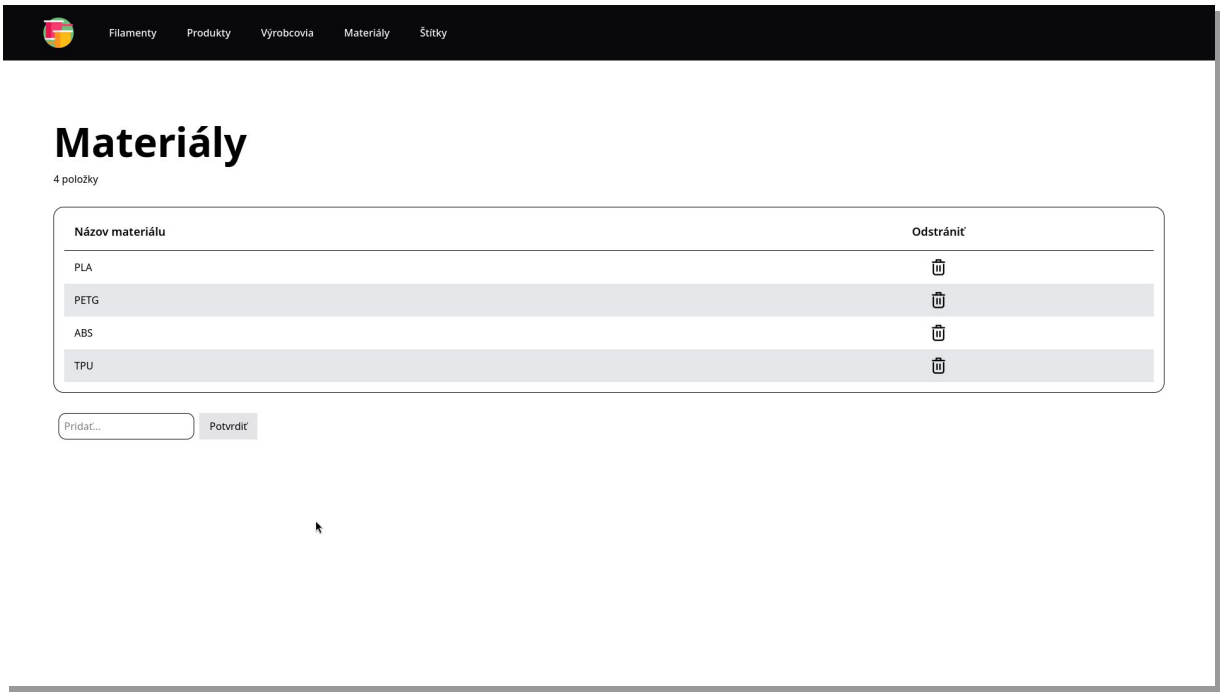
Ešte pred samotným pridaním filamentu je potrebné pridať produkt, výrobcu a materiál. Začneme pridaním materiálu.

Kliknutím na “Materiál“ v hornej lište sa dostaneme na podstránku s materiálmi. Na začiatku bude taktiež prázdna.



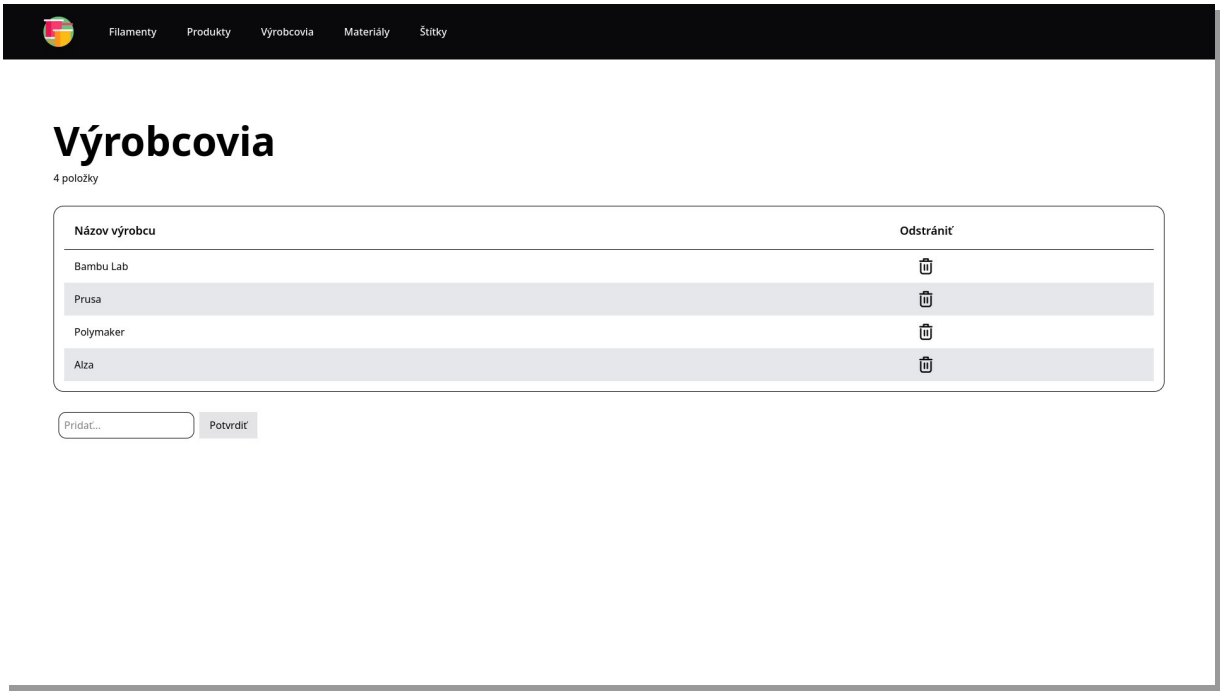
Obrázok 8 - Aplikácia - materiály

Pre pridanie materiálu napíšeme do políčka s označením “Pridať...“ názov daného materiálu, napr. PLA. Následne klikneme na tlačidlo “Potvrdiť“, čim pridáme daný materiál do tabuľky.



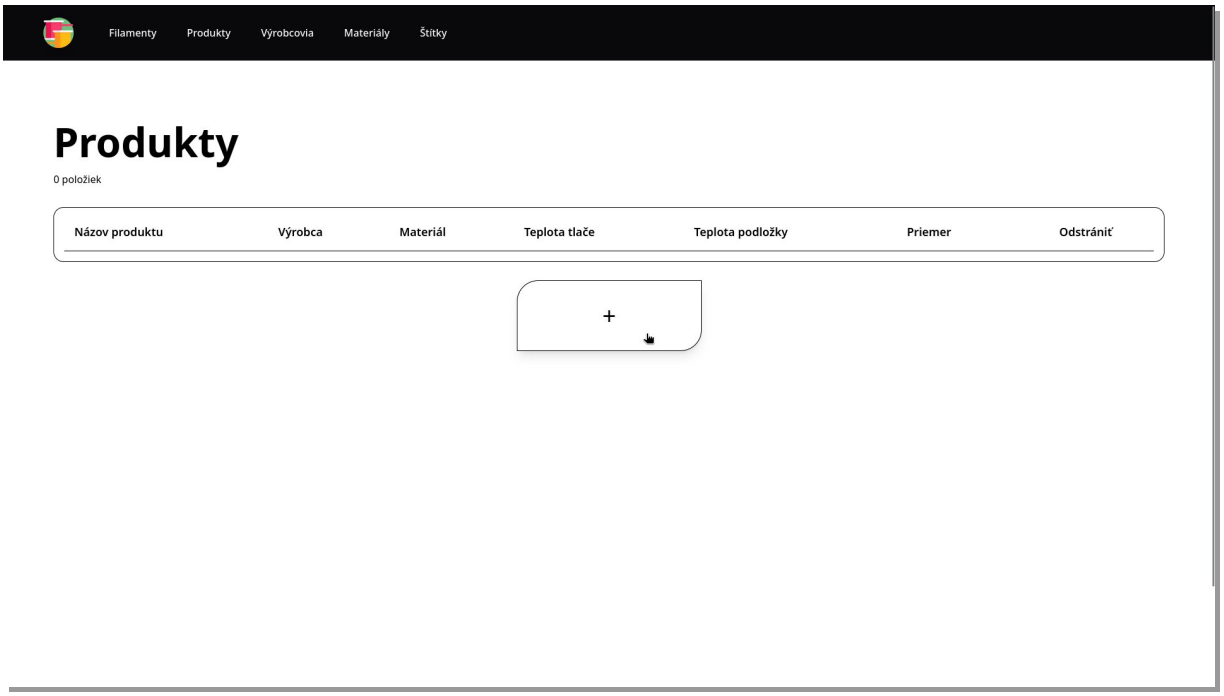
Obrázok 9 - Aplikácia - vytvorené materiály

Podobným spôsobom môžeme pridať aj výrobcov. Kliknutím na “Výrobcovia“ v hornej lište sa dostaneme na podstránku s výrobcami. Nového výrobcu pridáme rovnakým spôsobom ako sme pridávali materiál. Po pridaní môže tabuľka vyzerať nejak takto:

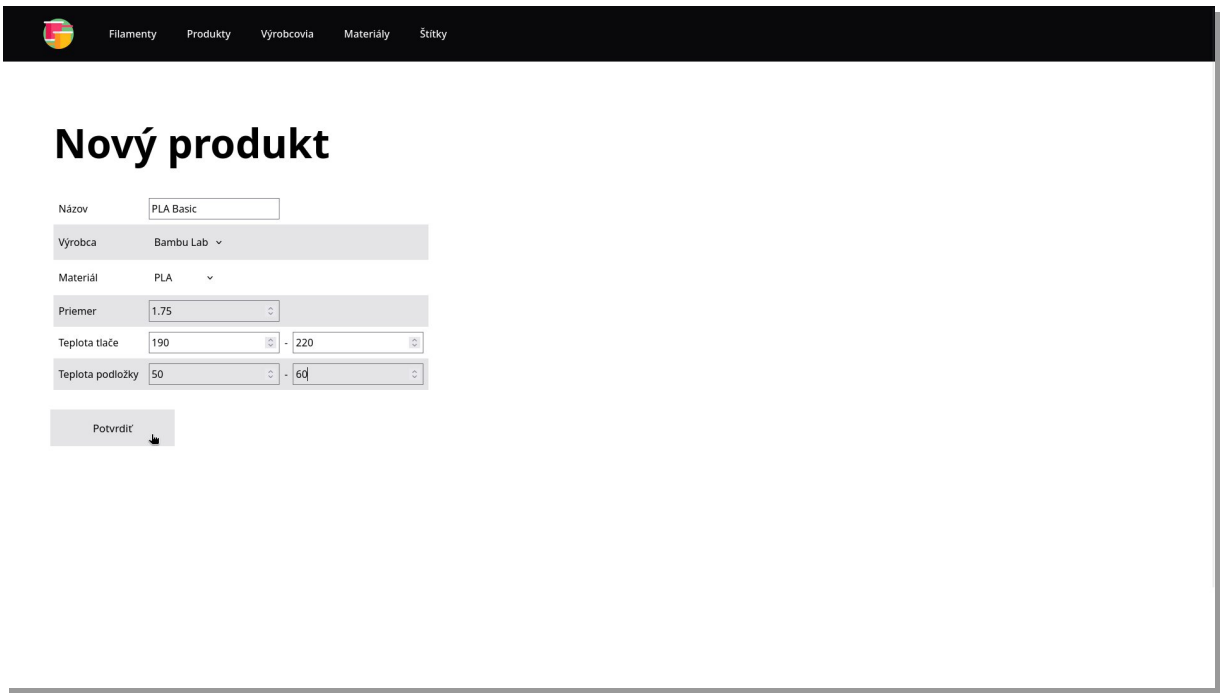


Obrázok 10 - Aplikácia - vytvorení výrobcovia

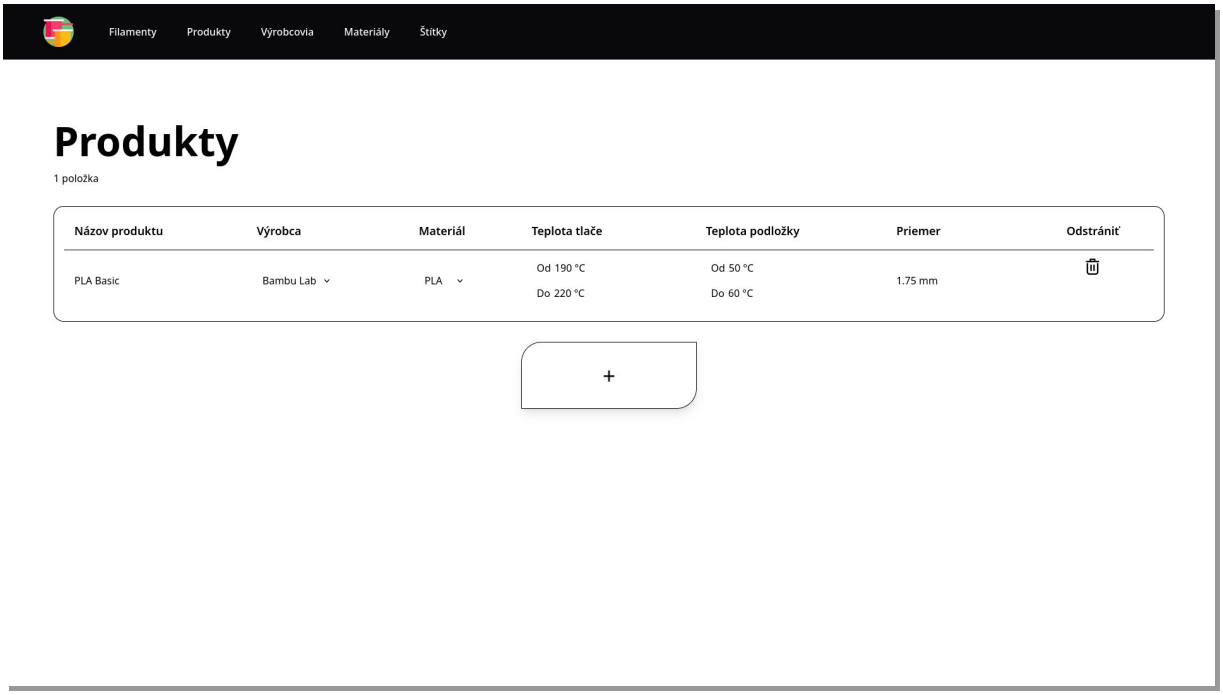
Ďalej môžeme pridať produkt. Kliknutím na “Produkty“ v hornej lište sa dostaneme na podstránku s produktami. Táto stránka bude na začiatku prázdna. Kliknutím na tlačidlo s označením “+“ sa dostaneme na podstránku pre pridávanie produktov. Vyplníme všetky polia, zvolíme výrobcu a materiál a klikneme na tlačidlo “Potvrdiť“. Následne môžeme vidieť pridaný produkt v tabuľke



Obrázok 11 - Aplikácia - produkty

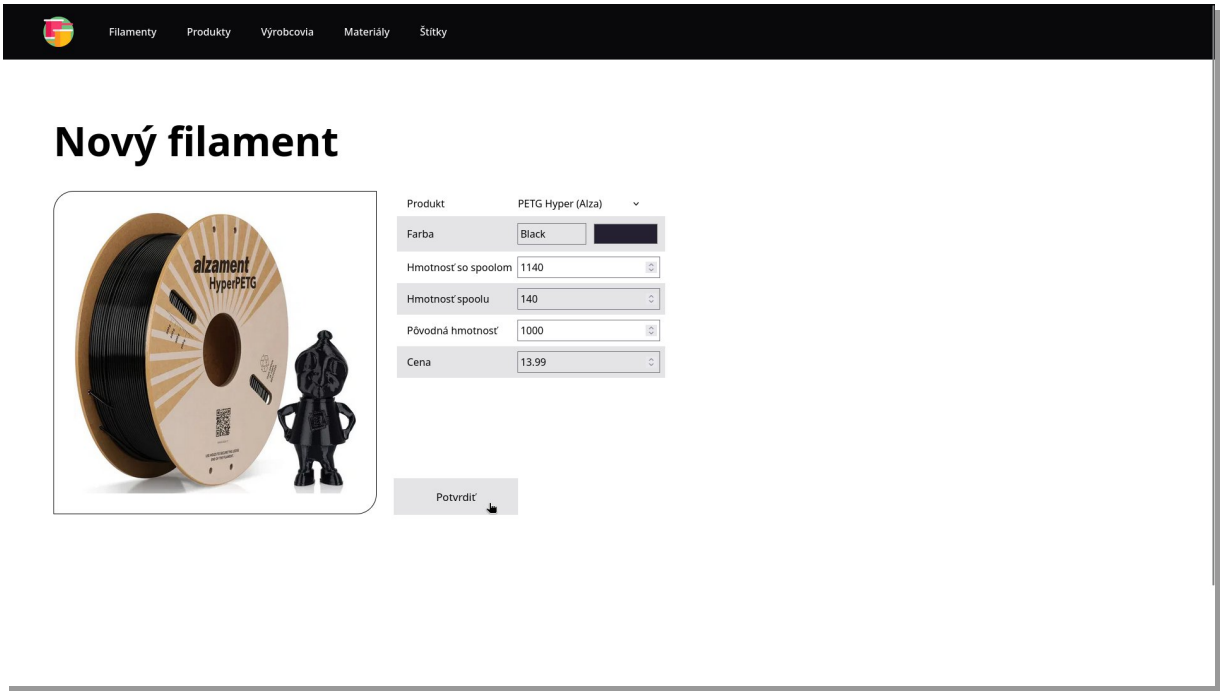


Obrázok 12 - Aplikácia - nový produkt

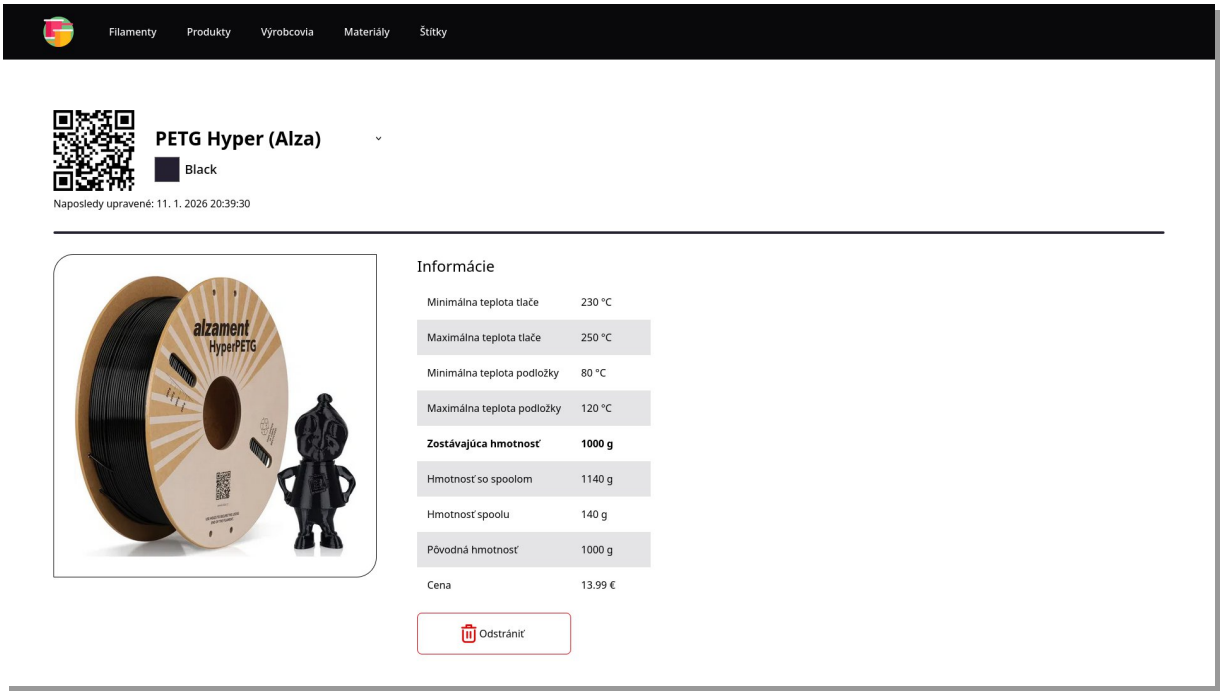


Obrázok 13 - Aplikácia - vytvorený produkt

Po pridaní produktu môžeme pridať filament. Na hornej lište klikneme na “Filamenty“ a následne klikneme na tlačidlo s označením “+“. Po presunutí na podstránku pre vytvorenie filamentu zvolíme príslušný produkt, vyplníme všetky polia a zmeníme obrázok (kliknutím na aktuálny obrázok). Teraz môžeme kliknúť na tlačidlo “Potvrdiť“, čím sa vytvorí nový filament a aplikácia nás presmeruje na stránku s vytvoreným filamentom

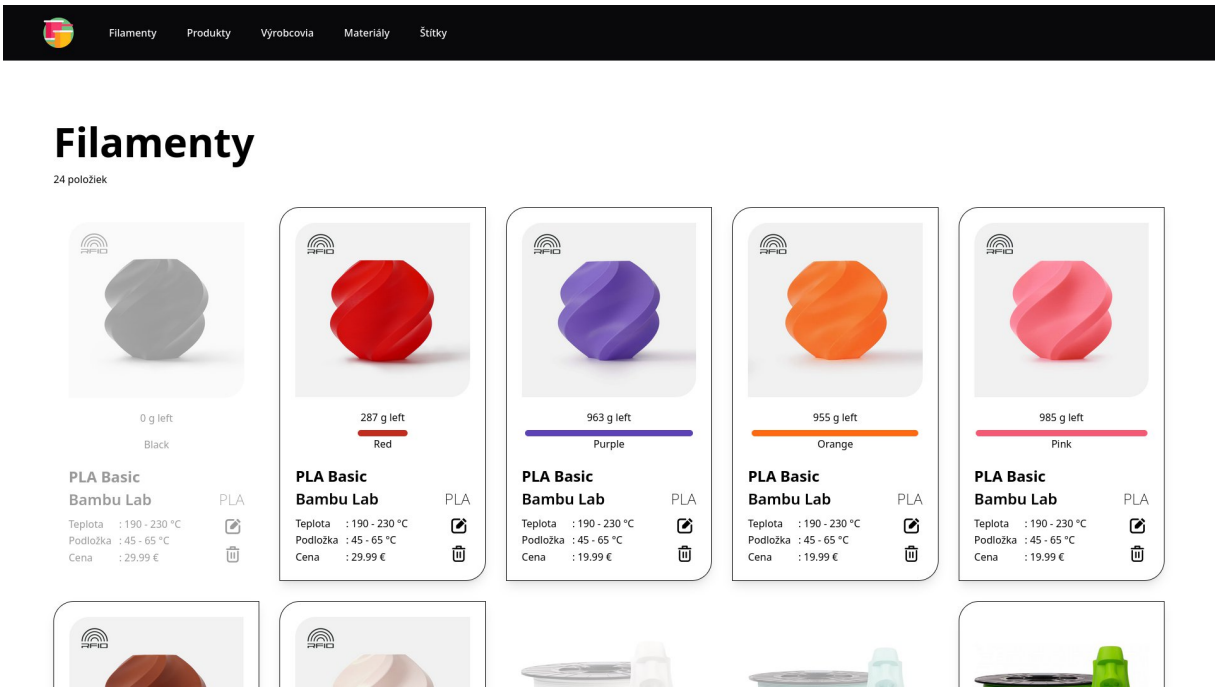


Obrázok 14 - Aplikácia - nový filament



Obrázok 15 - Aplikácia - vytvorený filament

Takýmto spôsobom môžeme pridávať ďalšie materiály, výrobcov, produkty a filamenty. Keď pridáme viac filamentov, tak úvodná stránka môže vyzerať nejak takto:



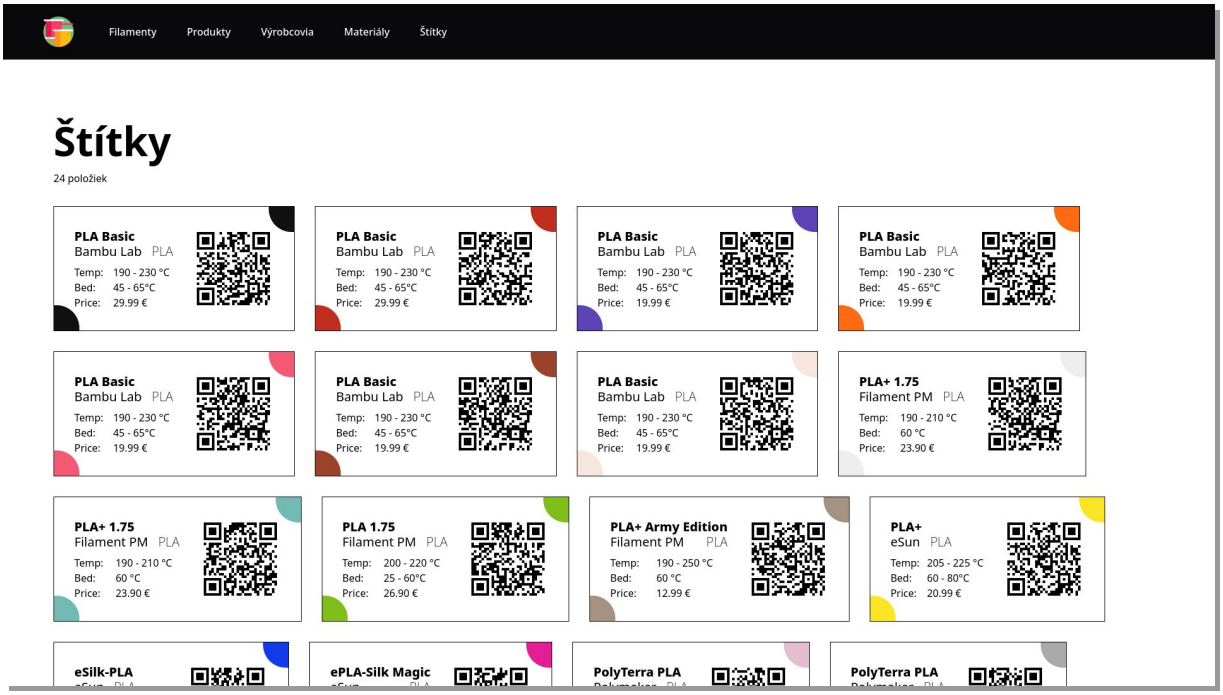
Obrázok 16 - Aplikácia - ukážka filamentov

Môžeme vidieť, že filamenty, ktorých zostávajúca hmotnosť je nulová sú vizuálne odlíšené od ostatných filamentov (menej sýte farby)

Ak chceme upraviť nejakú hodnotu filamentu, môžeme kliknúť na ikonku ceruzky (✎) alebo na obrázok filamentu. Produkty, výrobcovia a materiály sa dajú upravovať na ich príslušných podstránkach.

Ak chceme vymazať nejakú hodnotu, tak klikneme na ikonu koša (🗑). Pred samotným vymazaním vyskočí dialógové okno, kde môžeme buď potvrdiť alebo zrušiť vymazanie. Aby bolo vymazanie nejakej hodnoty úspešné, tak na ňu nemôže existovať žiadna iná referencia. Napríklad ak máme produkt z materiálu PLA, tak materiál PLA nie je možné vymazať.

Pri pridávaní filamentov sa automaticky generujú štítky. Môžeme ich vidieť na podstránke “Štítky“. Štítky môžu byť vytlačené na papier a nalepené napríklad na krabicu filamentu, na spool a podobne. Naskenovaním QR kódu cez mobilný telefón budeme presmerovaní na stránku daného filamentu (za predpokladu správnej konfigurácie, ako je uvedené v sekcií *Spustenie aplikácie*)



Obrázok 17 - Aplikácia - ukážka štítkov

5. Programátorská príručka

Backend je implementovaný ako Rust workspace, ktorý pozostáva z dvoch bedničiek

- *app* – spustiteľná aplikácia, ktorá zabezpečuje REST API pomocou knižnice Actix-web
- *lib* – knižnica, ktorá obsahuje spoločnú logiku pre prácu s databázou, dátové modely, QR kódy a ďalšie

Bednička *app* obsahuje súbor *main.rs*, ktorý predstavuje vstupný bod pre program, a moduly *endpoints* a *api_docs*.

V súbore *main.rs* sa najprv vytvorí tzv. pool pre databázu – skupina pripojení k databáze, ktoré program následne používa na vykonávanie databázových dotazov. Po vytvorení tohto pool-u sa vykonajú migrácie databázy, vytvoria sa potrebné adresáre v súborovom systéme pre obrázky, vygenerujú sa prípadné chýbajúce QR kódy filamentov, zistí sa číslo sieťového portu pre backend z premennej prostredia *BACKEND_PORT*, a spustí sa HTTP server. HTTP server je obalený do *middleware::TrailingSlash::Trim*, ktorý automaticky odstraňuje koncové lomítko z požiadaviek (GET */filaments/* → GET */filaments*), nastavuje CORS pravidlá, a poskytuje všetkým endpointom prístup k databázovému poolu. Všetky endpointy sa definujú pod */api/v2* (kvôli kompatibilite). Dodatočne bola pridaná služba SwaggerUI na vizualizáciu OpenAPI dokumentácie, dostupná na adrese *http://localhost:5000/swagger-ui/* (v prípade zmeny premenných v súbore *.env* sa bude táto adresa líšiť. Pozor, na konci adresy je potrebné zachovať lomítko)

Modul *endpoints* je rozdelený na štyri podmoduly

- *get* – obsahuje všetky GET endpointy. Funkcie v tomto module volajú príslušnú funkciu z *lib::db* (napr. endpoint */api/v2/vendors* zavolá funkciu *lib::db::get_vendors*) a spracujú výsledok pomocou funkcie *handle_type_result* (detaily funkcie sú popísané v ďalšom odseku). Výnimkou je posledná funkcia, ktorá reprezentuje *endpoint /api/v2/images/{:.*}* – tento endpoint vracia obrázok zo súborového systému
- *post* – obsahuje všetky POST endpointy. Tento modul by sa dal pomyselne rozdeliť na dve časti:
  - “Bežné funkcie“ – prijímajú JSON s príslušným typom (napr. funkcia *post_vendors* prijíma *Json<NewVendor>*), ktorý sa následne deserializuje a pošle do príslušnej metódy v databáze. Výsledok je taktiež spracovaný funkciou *handle_type_result*
  - “Multipart funkcie“ – funkcie, ktoré slúžia na nahratie obrázku. Jedná sa o funkcie *post_images* a *post_filaments_with_image* s ich príslušnými endpointami. Funkcie prijímajú požiadavky typu *multipart/form-data* ktoré obsahujú obrázok, ktorý uložia do súborového systému

- *delete* – obsahuje všetky DELETE endpointy. Funkcie v tomto module volajú funkciu *lib::db::general_delete_by_id* s konkrétnymi parametrami, a následne spracúvajú výsledok pomocou funkcie *handle_general_result*
- *patch* – obsahuje všetky PATCH endpointy. Funkcie v tomto module volajú funkciu *lib::db::general_patch* s konkrétnymi parametrami a následne spracúvajú výsledok pomocou funkcie *handle_general_result*

Modul má dve pomocné funkcie, ktoré sú definované v súbore *endpoints/mod.rs*:

- Funkcia *handle_general_result* – ako parameter má premennú typu *Result<(), lib::Error>*. Ak je táto premenná Ok, tak vráti stavový kód 200 OK, ak je Err tak vráti príslušný chybový kód podľa typu chyby
- Funkcia *handle_type_result* – ako parameter má premennú typu *Result<T, lib::Error>*, kde *T* je typový parameter ktorý implementuje *serde::Serialize*. Táto funkcia sa používa pri endpointoch ktoré vracajú nejaké údaje (väčšinou GET endpointy). Ak je premenná v parametri typu Ok, tak vráti stavový kód 201 Created spolu s príslušnými dátami. Ak je táto premenná Err, tak sa využije funkcia *handle_general_result* na spracovanie chyby

Prehľad všetkých endpointov, ich parametrov a možných odpovedí je zdokumentovaný pomocou OpenAPI dokumentácie, predvolene dostupné na adrese *http://localhost:5000/swagger-ui*, alebo v priloženom súbore *openapi.pdf*. Táto dokumentácia je generovaná v module *app::api_docs* pomocou knižnice *utoipa*

Bednička *lib* obsahuje súbor *lib.rs a* moduly *db*, *qr* a *uuid*

Modul *lib::db* obsahuje funkcionalitu potrebnú na prácu s databázou. Súbor *lib/src/db/mod.rs* obsahuje samotnú logiku pre prácu s databázou a pomocné funkcie. Súbor by sa dal pomyselne rozdeliť do 5 kategórií:

- Všeobecné a pomocné funkcie:
  - Funkcia *create_pool* vytvorí databázový pool s piatimi pripojeniami
  - Funkcia *handle_general_result* berie ako parameter typ *Result<PgQueryResult, sqlx::Error>*, a podľa hodnoty tohto parametra namapuje výsledok buď na *Ok(())*, alebo enum *lib::Error()* podľa hodnoty chybového kódu (duplicitné hodnoty v databáze, porušenie referenčnej integrity a pod.)
  - Funkcia *run_migrations* spustí migrácie databázy – k je to potrebné tak vytvorí v databáze požadované tabuľky
- Spracovanie HTTP GET požiadaviek – pošle na databázu dotaz typu SELECT na príslušnú tabuľku (*vendor*, *material*, ...). Pri funkciách *get_products_full* a *get_filaments_full* robí aj JOIN s ostatnými potrebnými tabuľkami, aby výsledok obsahoval aj názov výrobcu (nie len jeho ID) a podobne.

- Spracovanie HTTP POST požiadaviek – pošle na databázu dotaz typu INSERT pre vloženie daného typu do príslušnej tabuľky. Vo funkcií *add_filaments* sa navyše generuje QR kód pre daný filament
- Spracovanie HTTP DELETE požiadaviek – pošle na databázu dotaz typu DELETE. Jedná sa o jednu všeobecnú funkciu pre všetky tabuľky, ktorá "vyskladá" databázový dotaz z parametrov *table*, *id_name*, *id_value*, kde:
  - *table* je názov tabuľky z ktorej sa ide mazať (DELETE from {table} ...)
  - *id_name* a *id_value* je názov a hodnota primárneho kľúča v danej tabuľke (... WHERE {id_name} = {id_value})
- Spracovanie HTTP PATCH požiadaviek – pošle na databázu dotaz typu UPDATE. Tiež sa jedná o jednu všeobecnú funkciu, podobne ako v predchádzajúcej odrážke. Parametre funkcie sú *table*, *variable_name*, *variable_value*, *id_name*, *id_value* a *valid_variable_names*.
  - *table*, *id_name*, *id_value* majú zhodnú funkcionalitu s premennými v predchádzajúcej funkcií
  - *variable_name* a *variable_value* vyjadrujú názov stĺpca ktorý sa ide aktualizovať a jeho novú hodnotu
  - *valid_variable_names* (typ *Vec<&str>*) je zoznam platných stĺpcov v danej tabuľke. Napríklad v tabuľke *vendor* môže používateľ modifikovať iba stĺpec *name_vendor*, preto ak bude chcieť modifikovať stĺpec *id_vendor*, alebo neexistujúci stĺpec *last_update*, tak funkcia vráti chybu
  - Vo funkcií sa ešte zistí typ premennej (*variable_type*), pretože funkcia prijíma iba typ *&str*, ale databáza potrebuje konkrétny typ (napr. *text*, *char*, *null* a pod.). Následne je v dotaze premenná *variable_value* pretypovaná na typ *variable_type* pomocou PostgreSQL operátora ::
  - Celkový dotaz má takúto formu: UPDATE {table} SET {variable_name} = {variable_value}::{variable_type} WHERE {id_name} = {id_value}
  - Pri vkladaní do tabuľky *filament* sa tiež aktualizuje stĺpec *last_update* (...SET last_update = now()...)

Podmodul *lib::db::models* obsahuje štruktúry, ktoré som popisoval v sekcií UML Diagramy.

Modul *lib::qr* obsahuje funkcionalitu pre generovanie QR kódov. Nachádzajú sa tu tri funkcie:

- Funkcia *generate_manually* – vygeneruje QR kód zo zadaného obsahu (parameter *content*) a uloží ho ako obrázok na zadanú cestu (parameter *path*)
- Funkcia *generate_for_filament_id* – vygeneruje názov a obsah QR kódu pre filament (podľa parametra *id*), a následne využije funkciu *generate_manually* na generáciu a uloženie QR kódu. Funkcia vráti názov súboru, v ktorom je nový QR kód uložený

- Funkcia *prepare_missing* – v prvom kroku zistí z databázy ID filamentov, ktoré nemajú QR kódy. Následne vygeneruje chýbajúce QR kódy a aktualizuje hodnoty v databáze

Modul *lib::uuid* je veľmi jednoduchý modul, ktorý obsahuje funkciu na generovanie unikátnych identifikátorov (UUID), ktoré sa používajú ako názov súboru pre vygenerované QR kódy a obrázky nahraté od používateľa

6. Záver

V rámci semestrálnej práce sa podarilo navrhnúť a implementovať fullstack webovú aplikáciu na správu filamentov pre 3D tlač. Aplikácia umožňuje prehľadnú evidenciu filamentov a súvisiacich údajov, čo značne zjednodušuje ich správu. Ciele práce boli splnené a výsledná aplikácia je funkčná, rozšíriteľná a použiteľná v praxi.

Do budúcnosti by bolo možné aplikáciu rozšíriť o niektoré funkcionality, napríklad:

- Implementácia používateľského systému
- Šifrovanie – nasadenie HTTPS a šifrované spojenie s databázou
- Rozšírené informácie o filamentoch – napr. tolerancia, odporúčaná rýchlosť tlače, typ podložky a podobne

7. Použité zdroje

Materiály a zdrojové kódy z cvičení

Dokumentácie a príklady Rust knižníc

- https://actix.rs/docs/
- https://github.com/actix/actix-web/blob/5c6a29f4/actix-multipart/README.md
- https://docs.rs/actix-cors/latest/actix_cors/
- https://docs.rs/utoipa/latest/utoipa/
- https://docs.rs/utoipa/latest/utoipa/attr.path.html#actix_extras-feature-support-for-actix-web
- https://github.com/launchbadge/sqlx
- https://crates.io/crates/qrcode
- https://crates.io/crates/uuid

Online články

- https://www.w3tutorials.net/blog/react-js-replace-img-src-onerror/
- https://eastondev.com/blog/en/posts/dev/20251217-docker-multistage-build/#rust-applications-minimal-deployment
- https://dev.to/imzihad21/deploying-a-react-app-with-docker-using-nginx-4in4

Môj predchádzajúci projekt - https://github.com/cyprich/filamenty