Ekonomická fakulta
Faculty of Economics
Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v Českých Budějovicích
Ekonomická fakulta
Katedra aplikované informatiky a matematiky

Diplomová práce

Vývoj mobilní aplikace pro kontrolu 3D tiskárny s použitím jednodeskového počítače

Vypracovala: Bc. Markéta Lišková
Vedoucí práce: Mgr. Radim Remeš

České Budějovice 2021

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

Ekonomická fakulta

Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Markéta LIŠKOVÁ**
Osobní číslo:	**E19124**
Studijní program:	**N6209 Systémové inženýrství a informatika**
Studijní obor:	**Ekonomická informatika**
Téma práce:	**Vývoj mobilní aplikace pro kontrolu 3D tiskárny s použitím jednodeskového počítače**
Zadávající katedra:	**Katedra aplikované matematiky a informatiky**

Zásady pro vypracování

Cílem práce je vytvořit mobilní aplikaci pro ovládání 3D tiskárny. Aplikace bude komunikovat s jednodeskovým počítačem, který bude propojen s 3D tiskárnou.

Metodický postup:

1. Studium odborné literatury.
2. Návrh a popis vývoje a implementace výsledné aplikace.
3. Popis použitých technologií a komunikačních rozhraní.
4. Umístění aplikace do veřejnosti přístupného e-shopu.
5. Zhodnocení použitelnosti aplikace pro nasazení v reálném prostředí.
6. Závěr a doporučení.

Rozsah pracovní zprávy:	**50 – 60 stran**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování diplomové práce:	**tištěná**

Seznam doporučené literatury:

1. Bell, C. (2014). *Maintaining and Troubleshooting Your 3D Printer.* New York, NY (USA): Apress.
2. Cameron, R., & Horvath, J. (2018). *Mastering 3D Printing in the Classroom. Library, and Lab.* New York, NY (USA): Apress.
3. Molloy, D. (2016). *Exploring Raspberry Pi.* Indianapolis, Indiana (USA): John Wiley & Sons.
4. Veen, T. (2017). *Swift in Depth.* Shelter Island, NY (USA): Manning.
5. Horvath, J., & Cameron, R. (2015). *The New Shop Class: Getting Started with 3D Printing, Arduino, and Wearable Tech.* New York, NY (USA): Apress.

Vedoucí diplomové práce: **Mgr. Radim Remeš**
Katedra aplikované matematiky a informatiky

Konzultant diplomové práce: **doc. Ing. Ladislav Beránek, CSc.**
Katedra aplikované matematiky a informatiky

Datum zadání diplomové práce: **17. ledna 2020**
Termín odevzdání diplomové práce: **16. dubna 2021**

JIHOČESKÁ UNIVERZITA
V ČESKÝCH BUDĚJOVICÍCH
EKONOMICKÁ FAKULTA
Studentská 13 (26)
370 05 České Budějovice

doc. Dr. Ing. Dagmar Škodová Parmová
děkanka

doc. RNDr. Tomáš Mrkvička, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 25. března 2020

Prohlášení

Prohlašuji, že svou diplomovou práci „Vývoj mobilní aplikace pro kontrolu 3D tiskárny s použitím jednodeskového počítače“ jsem vypracovala samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to – v nezkrácené podobě/v úpravě vzniklé vypuštěním vyznačených částí archivovaných Ekonomickou fakultou – elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

……..	……..
Datum	Podpis

Poděkování

Ráda bych poděkovala vedoucímu diplomové práce Mgr. Radimu Remešovi za jeho cenné rady, připomínky a vstřícnost při vypracovávání diplomové práce.

Obsah práce

1 Úvod a cíl diplomové práce

V dnešní době je 3D tisk čím dál více populární. Jeho popularitu zvyšuje fakt, že 3D tiskárny se stávají stále více finančně dostupné a také fakt, že v období pandemie je hodně lidí nuceno trávit volný čas doma. Tisk 3D modelu může trvat několik desítek minut, ale v některých případech i několik hodin. Samozřejmě se nám může stát, že se v průběhu tisku něco pokazí a my tak strávíme několik hodin tisknutím jen chomáče plastu. Pro tyto případy bude v diplomové práci vyvinuta aplikace, která umožní uživateli kontrolovat tisk z pohodlí gauče a předejít těmto případům.

Cílem diplomové práce je vytvořit aplikaci na kontrolu průběhu tisku modelu na 3D tiskárně za pomoci jednodeskového počítače. Tato aplikace má uživatelům zpohodlnit kontrolu průběhu tisku modelu a poskytnout aktuální informace o tisku.

V teoretické části je nejdříve krátce popsána nejčastěji používaná technologie 3D tisku, pro kterou je i řešena aplikace v praktické části. Také je zde popsána nejprodávanější tiskárna na trhu. Tyto kapitoly byly do práce zahrnuty z důvodu, aby čtenář této práce získal větší povědomí o 3D tisku. V další kapitole jsou popsány základní principy návrhu aplikace a používané nástroje pro tyto účely. Dále jsou zde zmíněny využitá zařízení a technologie důležité pro komunikaci s tiskárnou. Mobilní aplikace v praktické části je vyvinuta pro operační systém iOS, a proto jsou kapitoly teoretické části týkající se vývoje aplikace přizpůsobeny pro tuto platformu. Jedná se o kapitoly: architektura mobilních aplikací, vývoj aplikace a distribuce aplikace.

Předmětem praktické části, jak již bylo zmíněno, je aplikace pro kontrolu tisku na FDM tiskárně, která je určena pro zařízení s operačním systémem iOS. Aplikace byla navržena tak, aby uživateli poskytovala základní informace o tisku, jako je například teplota tisku, čas dokončení, záběry z kamery a informace o vrstvě tisku. Také do aplikace byla zařazena možnost přerušení a zrušení tisku. Praktická část se zabývá procesem návrhu mobilní aplikace, popisem implementovaných komunikačních rozhraní, popisem zvolené architektury a procesem vývoje aplikace. Dále jsou zde kapitoly věnující se distribuci této aplikace, uživatelským hodnocením aplikace a návrhům na vylepšení.

2 Teoretická část

V teoretické části bude nejdříve popsán 3D tisk jako samotný, aby si čtenář uměl lépe představit tuto technologii a proces tisku 3D modelu. V dalších kapitolách této části se budeme více věnovat popisu kroků návrhu mobilní aplikace pro 3D tiskárnu a krokům vývoje mobilní aplikace.

2.1 3D tisk

V dnešní době je 3D tisk čím dál více populární. Jeho popularitu zvyšuje fakt, že 3D tiskárny se stávají stále více finančně dostupné a také fakt, že v období pandemie je hodně lidí nuceno trávit volný čas doma. Uživatel má na výběr z nepřeberného množství technologií tisků a výrobců tiskáren. Mezi nejpoužívanější technologii patří Fused Deposition Modeling (FDM) a Stereolitografie.

Obě tyto technologie fungují na principu vytváření 3D modelu na tiskovou základnu po jednotlivých vrstvách. Nejdříve je vytisknuta základní vrstva a na tu se dále tisknou další navazující vrstvy a model tak roste od základny do výšky. Tyto technologie vyžadují použití podporných struktur, které musí být po tisku ručně odstraněny. Podpory slouží k tisku šikmých tvarů nebo mostů, které pod sebou nemají žádný materiál základního modelu, na který by šlo tisknout.

Mezi nejvýznamnější výrobce 3D tiskáren pro náš i světový trh patří česká společnost Prusa Research. Druhým nejvýznamnějším výrobcem je čínská společnost Creality. Obě tyto společnosti se nejvíce zajímají o výrobu Fused Deposition Modeling tiskáren pro domácí využití.

Mobilní aplikace, jejíž vývoj je popsán v praktické části této diplomové práce, je převážně určena na kontrolu tisku metodou FDM a je odladěna na tiskárně od společnosti Prusa Research, proto se v této části více zaměříme na tuto metodu tisku a představíme si nejpoužívanější tiskárnu od Prusa Research.

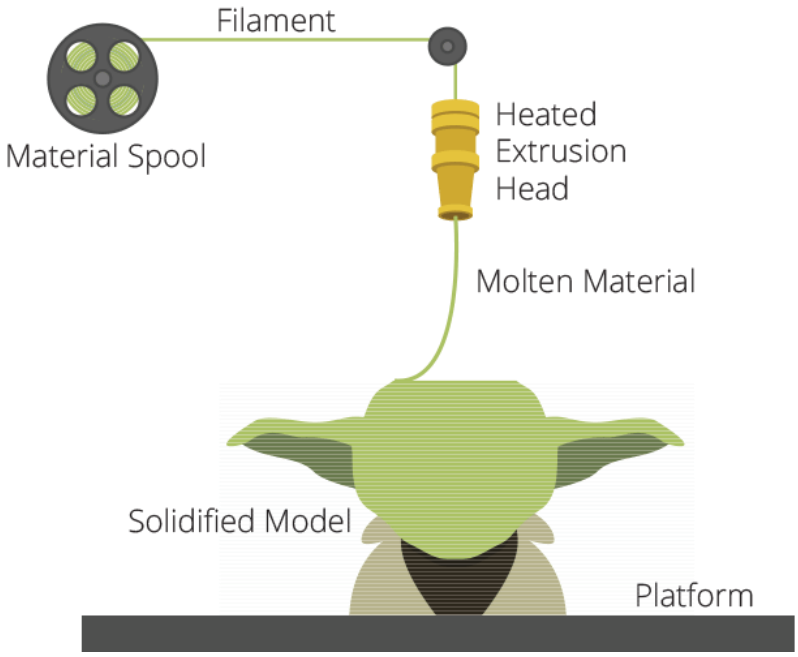
2.1.1 Fused Deposition Modeling

FDM (Fused Filament Fabrication) je technologie 3D tisku, která vytváří model tavením termoplastických polymerů, a ty jsou tiskovou hlavou po vrstvách nanášeny na stavební platformu ve tvaru modelu, který je tisknut. (Varotsis, 2018)

Tiskárna se skládá z tiskové hlavy, která taví tiskové struny (jinak označované jako filamenty), a z platformy, která se při tisku posunuje horizontálně či vertikálně, záleží na typu tiskárny.

Tisková struna je nejprve vložena do tiskárny. Jakmile tryska dosáhne teploty bodu tání tiskového materiálu, struna se přivede do tiskové hlavy, kde se roztaví. Pohyblivá tisková hlava (Obrázek 1) s pomocí posunů platformy zajišťuje pohyb v osách X, Y a Z. Roztavený materiál je nanášen tryskou v požadovaném tvaru na stavební platformu, kde zchladne a postupně tuhne. Když je vrstva dokončena, tisková hlava se posune nahoru a na již vytištěnou vrstvu je pokládána vrstva nová. Tento proces se opakuje, dokud není model zcela vytištěn. (Varotsis, 2018)

FDM tiskárny patří cenově mezi nejdostupnější tiskárny na trhu. Lze tisknout z nejrůznějších materiálů nabízených ve velkém množství barev. Tato technologie vyžaduje tvorbu podpor pro jakékoliv objekty s převislou geometrií. Podpory mohou být z výtisku jednoduše ručně odstraněny. (Jackson, et al., 2017) Starší tiskárny umožňovaly barevný tisk pouze po vrstvách, ale s vývojem modernějších tiskáren je už i možný tisk více barev v jedné vrstvě modelu, a dokonce si barvy již mění automaticky sama tiskárna.



Obrázek 1: Tisk metodou FDM

Zdroj: (Jackson, et al., 2017)

2.1.2 Prusa i3 mk3S

Prusa i3 mk3S (Obrázek 2) je model FDM tiskárny od společnosti Prusa Research, který je považován za nejlepší 3D tiskárnu po celém světě. Prvně byla představena v únoru 2019 a průběžně je každý rok vylepšována její funkcionalita. (3D tiskárna Original Prusa i3 MK3S+, 2021)

Hlavní výhodou této tiskárny jsou jednoduše vyměnitelné tiskové pláty, které jsou k tiskárně přichyceny pomocí silných magnetů. Díky této komponentě lze z tiskárny velmi jednoduše odebrat vytištěný model. Tyto pláty se prodávají ve dvou variantách s různou hrubostí povrchu, pro lepší vlastnosti tisku při použití různých materiálů. (Jennings, 2019) Na obrázku níže je na tiskovém plátu umístěn model hlavy.

Mezi další výhody patří nepřerušení tisku po výpadku elektrického proudu, rychlost a tichost tisku. Některé tisky větších modelů mohou zabrat klidně i 20 hodin tisku a pokud by vypadl proud uživatel by přišel o celý tisk. Díky této funkcionalitě tiskárna dokáže při návratu elektrické energie navázat na přerušený tisk a model v pořádku dotisknout. (Jennings, 2019)

Některé části tiskárny jsou přímo vyrobeny na jiných 3D tiskárnách. Na obrázku níže jsou to převážně oranžové části tiskárny. Výrobce nabízí modely těchto částí zdarma na své webové stránce, pokud by se tedy nějaká takováto součást tiskárny porouchala, lze si jí snadno znovu vytisknout doma.

Obrázek 2: Tiskárna Prusa i3 mk3S

Zdroj: (3D tiskárna Original Prusa i3 MK3S+, 2021)

2.2 Návrh mobilní aplikace

Návrh mobilní aplikace nebo také softwarové prototypování je vytváření neúplných verzí softwarového programu. Pojem „prototyp“ často slyšíme v mnoha různých kontextech. Z tohoto důvodu by mohlo dojít k nejasnostem ohledně jeho významu. Prototyp je především simulace toho, jak bude hotový software fungovat a vypadat. Prototypem tedy může být téměř cokoli, od série papírových skic představujících různé obrazovky nebo stavy aplikace až po plně funkční aplikaci navrženou na pixely přesně. (Takayama & Landay, 2002)

Prototypování hraje zásadní roli v procesu vytváření úspěšné uživatelské zkušenosti (UX). Umožňuje nám tedy otestovat použitelnost, pohodlnost a intuitivnost daného systému dříve, než daný systém vyvineme. Nejprve bychom tedy měli sestrojit vhodný prototyp pro koncové uživatele a tento prototyp jim dát k otestování. Ze získané zpětné vazby bychom se měli poučit, dané problémové oblasti v aplikaci předělat a znovu je dát uživatelům k testování, zdali v aplikaci nevznikl jiný problém. Pomocí tohoto procesu jsme jedině schopni navrhnout aplikaci, kterou uživatelé budou rádi používat a nebudou mít problém se v aplikaci zorientovat. (Takayama & Landay, 2002)

Při návrhu mobilních aplikací se často setkáváme s dalším pojmem wireframe. Prototyp a wireframy jsou si velmi blízké, ale největší rozdíl je zde v interaktivitě. Ve wireframech zobrazujeme pouze konkrétní rozložení stránek aplikace. V prototypu navíc zobrazíme i interakci a průchod aplikací. V prototypu nalezneme odpovědi na tyto otázky: Co se stane, když klikneme na toto tlačítko? Kam vede tento odkaz? Pokud však wireframe dokáže na tyto otázky odpovědět také, lze ho považovat za prototyp. (Pierzchała, 2018)

Prototypy lze rozdělit do dvou typů dle jejich fidelity (věrnosti). Pojem „věrnost“ odpovídá počtu podrobností prototypu. Čím blíže se prototyp podobá konečnému designu, pokud jde o jeho vzhled a funkčnost, tím vyšší je jeho úroveň. Prototypy obecně spadají do jedné ze dvou úrovní: low-fidelity (nízká věrnost) nebo high-fidelity (vysoká věrnost). (Babich, 2017)

Tyto úrovně se především liší v oblastech:

- Vizuální design
- Obsah
- Interaktivita

2.2.1 Low-fidelity prototyp

Low-fidelity prototypování je rychlý a snadný způsob, jak převést složitý design na vysoké úrovni do hmatatelných a testovatelných objektů. První a nejdůležitější rolí prototypů této úrovně je kontrolovat a testovat funkčnost spíše než vizuální vzhled aplikace. (“Low-fi prototyping: What, Why and How?”, 2016)

Zde jsou základní charakteristiky prototypů s nízkou věrností: (Babich, 2017)

- Vizuální design: Prezentovány jsou pouze některé vizuální atributy konečného produktu. Například základní tvary.
- Obsah: Zahrnuty jsou pouze klíčové prvky obsahu.
- Interaktivita: Při testování prototypem konečného uživatele provází osoba, která ručně mění stav návrhu v reálném čase. Interaktivitu lze také vytvořit propojením wireframu v příslušném programu na počítači.

Tabulka 1: Výhody na nevýhody Low-fidelity prototypu

Výhody	Nevýhody
Levné	Omezená funkcionalita
Rychlé	Velká schopnost představivosti
Vhodné pro skupinovou spolupráci	

Zdroj: (Babich, 2017)

Mezi nejpopulárnější techniky tvorby těchto prototypů patří tvorba papírových prototypů a počítačové propojování wireframů.

Papírové prototypy

Prototypování pomocí papíru vám umožňuje vytvořit rozhraní digitálního produktu bez použití digitálního softwaru. Tato technika je založena na vytváření ručních výkresů různých obrazovek, které představují uživatelská rozhraní produktu. Je velmi užitečná při prvotním zkoušení různých variant rozložení systému. (“Low-fi prototyping: What, Why and How?”, 2016)

Jednoduché obrazovky jsou nakresleny na papír (Obrázek 3) a osoba, která je seznámena s logikou průchodu produktu, simuluje počítač a přepíná náčrty podle volby testovaného uživatele.

Obrázek 3: Testování uživatele formou papírového prototypování

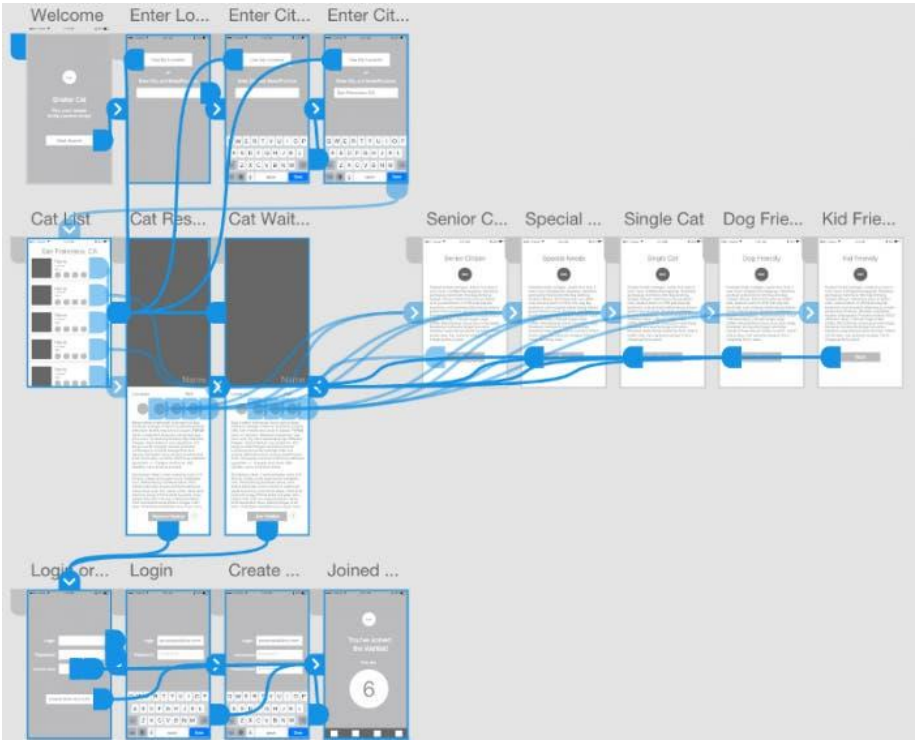
Zdroj: ("Low-fi prototyping: What, Why and How?", 2016)

Mezi hlavní výhody této metody patří rychlost sestrojení prototypu a nenáročnost na použití návrhového nástroje, protože skoro každý umí kreslit. Touto metodou lze také do prototypu velmi rychle zapracovávat další změny a nápady. Jeho hlavní nevýhodou je potřeba další osoby při uživatelském testování a grafické zpracování prototypu. ("Low-fi prototyping: What, Why and How?", 2016)

Vzhledem k výhodám a nevýhodám se doporučuje používat papírové prototypy pouze v raných fázích návrhu, když je produkt stále abstraktní nebo je v procesu formování. Čím dále se dostaneme do procesu návrhu, tím významnější bude rozdíl mezi papírovými prototypy a konečným produktem. (Babich, 2017)

Interaktivní wireframy

Tento prototyp vzniká propojením wireframů v navrhovacím nástroji (Obrázek 4). Jak již bylo řečeno výše, wireframe zobrazuje rozložení jednotlivých stránek produktu. Stejně jako papírové prototypy, interaktivní wireframy často nevypadají jako hotový produkt, ale oproti papírovým prototypům mají jednu významnou výhodu, a to že nevyžadují samostatnou osobu, která by během testování pracovala jako pomocník. Další výhodou je, že lze z tvorby low-fidelity prototypu přejít na tvorbu high-fidelity prototypu bez změny navrhovacího nástroje. (Pierzchała, 2018)



Obrázek 4: Spojené wireframy v Adobe XD

Zdroj: (Babich, 2017)

2.2.2 High-fidelity prototyp

High-fidelity prototyp se z pravidla většinou tvoří až po low-fidelity prototypu. Tato úroveň prototypu již zachycuje uživatelské rozhraní produktu (UI) z vizuálního i estetického aspektu, ale také zahrnuje i aspekt přívětivého UX. Prototyp se snaží zobrazit finální produkt v co nejvýstižnější podobě, jak z pohledu designu, tak i z pohledu průchodu celou aplikací. Prototypy této úrovně se vytvářejí ve fázi, kdy dobře známe finální produkt i jeho uživatele. (Ibragimova, 2016) Po otestování prototypů této úrovně a zanesení poznatků po testování do prototypu, již vzniká finální podoba prototypu, podle které je vyvíjen finální produkt.

Zde jsou základní charakteristiky prototypů s vysokou věrností: (Babich, 2017)

- Vizuální design: Realistický a detailní design – všechny grafické prvky rozhraní, mezery a tlačítka vypadají jako skutečná aplikace nebo web.
- Obsah: Používá se skutečný nebo velmi podobný obsah skutečnému. Prototyp zahrnuje většinu nebo veškerý obsah, který se objeví v konečném designu.
- Interaktivita: Prototyp je vytvořen v digitálním navrhovacím nástroji. Jsou zde zachyceny téměř veškeré interakce finálního produktu.

Tabulka 2: Výhody na nevýhody High-fidelity prototypu

Výhody	Nevýhody
Velká věrohodnost	Velmi pracné
Ucelený pohled na produkt	Nákladné
Atraktivní, líbivé	

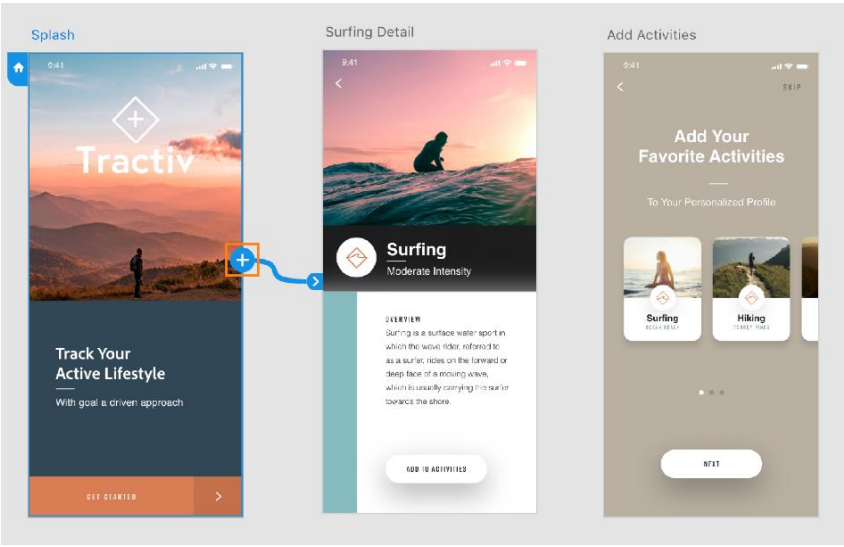
Zdroj: (Babich, 2017)

Mezi nejpopulárnější techniky tvorby těchto prototypů patří digitální prototypy vytvořené pomocí speciálních nástrojů a kódované prototypy.

Digitální prototypy

Digitální prototypy (Obrázek 5) jsou nejběžnější formou této úrovně prototypování. V dnešní době umožňuje návrhářům řada specializovaného softwaru vytvářet vizuálně bohaté a výkonné prototypy plné interaktivních efektů a složitých animací.

Mezi hlavní výhody této metody patří, že navrhovací software umožňuje návrhářům zobrazit náhled prototypu ve webovém prohlížeči nebo na jakémkoli stolním nebo mobilním zařízení. To pomáhá návrhářům UX a UI dosáhnout optimálního rozvržení na různých typech zařízení. Dále také tento prototyp osvobozuje designéra od nutnosti vysvětlovat koncovému uživateli jednotlivé koncepty během testování, což mu umožňuje soustředit se na pozorování a lépe získat zpětnou vazbu. (Babich, 2017) Do otestovaného produktu je zpravidla zanesena zpětná vazba uživatele a je vytvořena finální podoba produktu.



Obrázek 5: Figh-fidelity prototyp v Adobe XD

Zdroj: (“Create interactive prototypes”, 2021)

Kódované prototypy

Kódované prototypy jsou o něco méně častější než prototypy digitální. Takovýto prototyp je rovnou tvořen v programovacím jazyce, ve kterém bude psán front-end finálního produktu. Je zde naprogramován celkový vzhled produktu, přechody mezi jednotlivými stránkami a grafickými prvky, ale je vynechána logika na pozadí aplikace. Tento prototyp trvá nejdelší dobu vytvořit a je zde náročnější zanášet změny z uživatelského testování. Na druhou stranu takto vytvořený prototyp již může použít při finálním vývoji produktu jako naprogramovanou grafiku front-endové aplikace.

2.2.3 Navrhovací nástroje

Nástroje pro vytváření prototypů umožňují návrhářům a koncovým uživatelům lépe spolupracovat na vytváření podoby konečného produktu. Uživatelé získávají vizuální přehled o tom, jak se vlastně produkt dělá a jak se ovládá. Prototypy vytvořené v navrhovacích nástrojích tak pomáhají týmům budovat porozumění, zkoumat možnosti a bariéry, které se stanou viditelnými až když k prototypu přistoupí koncový uživatel. Největší výhodou prototypových nástrojů je brzké odhalení problémů a jejich jednoduchá oprava.

Výběr správného nástroje je klíčem k popisu myšlenek a lepší spolupráci s uživateli. V dnešní době je na výběr z nepřeberného množství prototypovacích nástrojů. V této kapitole budou popsány ty nejčastěji používané, které si v omezené míře můžeme vyzkoušet zdarma.

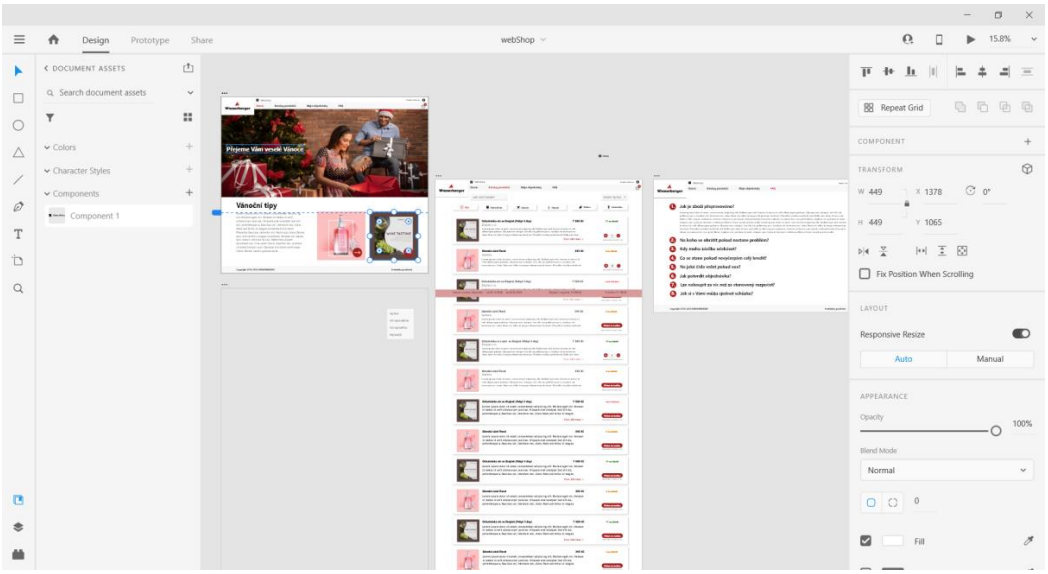
Adobe XD

Adobe XD je desktopový vektorový editor určený designérům k prototypování webových stránek, mobilních aplikací a dalších produktů. Aplikace má velmi jednoduché rozhraní a její ovládání zvládne úplný začátečník. Pro aplikaci existuje také velké množství doplňků s rozšiřujícími funkcionalitami.

V Adobe XD lze vytvářet jak low-fidelity tak i high-fidelity prototypy. Adobe nabízí různé módy tvorby prototypu, a to design, prototyp a share. V módu design (Obrázek 6) je tvořen požadovaný vzhled produktu. Jednotlivé komponenty pro návrh UI produktu jsou umístěny v pravé liště. Nalezneme zde základní tvary, možnost črtání, psaní a také možnost vložení obrázku v různých formátech. Tyto tvary jsou skládány na sebe, a tak je tvořena požadovaná úroveň vzhledu prototypu. Nastavování vlastností jednotlivých prvků nalezneme v levé navigační liště. Zde lze prvku definovat barvu, stín,

velikost, polohu a mnohem více. Samozřejmě také záleží na druhu prvku, který je editován.

V módu prototype definujeme přechody mezi jednotlivými obrazovkami. Tedy většinou vybereme jeden konkrétní prvek a jemu nadefinujeme přesun na další obrazovku. Po tomto kroku nám vzniká digitální prototyp, abychom mohli tento prototyp někomu zaslat a nemusel být vždy u našeho počítače, je zde mód share, kde vygenerujeme veřejný URL link, ze kterého lze prototyp prohlížet. Jsou zde různé možnosti nastavení tohoto linku. Lze generovat speciální link jen pro prezentaci, link pro komentování prototypu a link pro stahování jednotlivých grafických komponent prototypu.



Obrázek 6: Prostředí Adobe XD

Zdroj: Autor práce

Tento nástroj si lze vyzkoušet i zdarma. Funkcionality při vytváření prototypu nejsou vůbec nijak v této verzi omezeny, ale je zde omezeno vytvoření veřejného linku na jeden, sdílení pouze jednoho prototypu s dalším designérem a také omezené cloudové úložiště na 2 GB. (Adobe, 2021)

Tabulka 3: Výhody na nevýhody Adobe XD

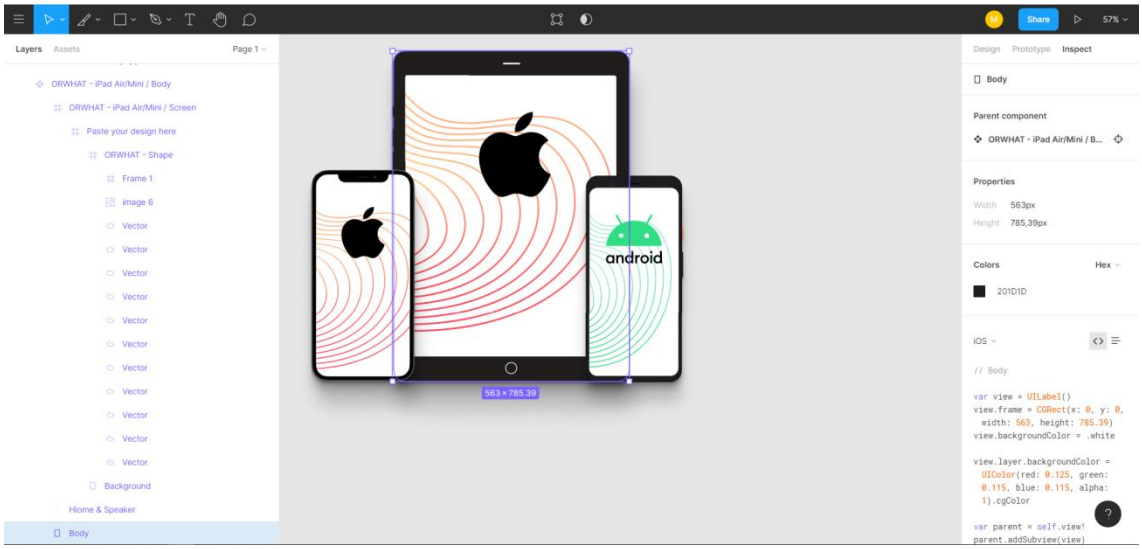
Výhody	Nevýhody
Přehledné	Kopírování textu – text v odstavcích se nakopíruje, jako jeden řádek.
Jednoduché na ovládání	Omezené animace
Opakování objektů ve zvoleném prostoru	Občas se celý projekt zasekne a je třeba jej znovu uložit.

Zdroj: (Baratashvili, 2017)

Figma

Figma je cloudový vektorový editor určený designérům k prototypování především webových stránek, ale i mobilních aplikací a dalších produktů. Aplikace se na první pohled nejeví jako úplně jednoduchá na ovládání, ale jakmile do ní uživatel pronikne, nabízí spoustu šikovných funkcionalit. Díky tomu, že celý prototyp je tvořen online, je Figma velmi vhodný nástroj na týmovou spolupráci.

V tomto navrhovacím nástroji lze tvořit prototypy všech úrovní. Základní prvky (Obrázek 7) pro tvorbu prototypu jsou umístěny v horní liště, kde nalezneme možnosti od vložení základního tvaru, textu, křivek, obrázku, možnost vložení komentáře atd. Pod touto sekcí vidíme seznam jednotlivých komponent prototypu. V pravé navigační liště se nám zobrazuje nastavení zvoleného prvku, kde můžeme přepínat mezi jednotlivými módy nastavení. V nastavení design definujeme grafické vlastnosti prvku. V módu prototype prvek napojujeme na jiné prvky a v posledním módu inspect vidíme základní informace potřebné pro vývoj prvku. Figma také nabízí ukázku kódu grafického prvku v základních jazycích pro vývoj front-endových aplikací.



Obrázek 7: Prostředí editoru Figma

Zdroj: Autor práce

Figmu si lze vyzkoušet i zdarma. Tato verze nabízí vytvoření až 3 prototypů, spolupráci s dvěma dalšími osobami a neomezené množství vytvoření veřejných linků k prototypu. (Figma pricing, 2021)

Tabulka 4: Výhody na nevýhody softwaru Figma

Výhody	Nevýhody
Spolupráce v reálném čase	Nemožnost pracovat offline
Tvorba zdrojového kódu	Z počátku složitější na ovládání
Jednoduché přecházení mezi projekty	

Zdroj: (Berezhnoi, 2021)

2.3 Komunikační rozhraní mezi aplikací a 3D tiskárnou

Pojem rozhraní v informatice označuje zařízení, program nebo formát, zajišťující správnou komunikaci a přenos dat mezi odlišnými zařízeními nebo programy. Na rozhraní lze nahlížet ze tří pohledů, a to z pohledu hardwarového, softwarového a z uživatelského pohledu. Hardwarovým rozhraním může být například USB, počítačová síť. Pod softwarovým rozhraním si můžeme představit protokoly, přes které mezi sebou komunikují jednotlivé programy. Uživatelské rozhraní jsou grafické prvky, přes které uživatel komunikuje se zařízením, toto rozhraní zachycuje také prototyp při návrhu aplikace. (Peterka, 1992) V této sekci budou více popsána jednotlivá komunikační rozhraní potřebná pro propojení mobilní aplikace s 3D tiskárnou.

2.3.1 Raspberry Pi

Raspberry Pi je název řady jednodeskových počítačů o velikosti kreditní karty vytvořených nadací Raspberry Pi Foundation, která si klade za cíl vzdělávat lidi ve výpočetní technice a usnadňovat přístup k počítačové výuce. Po celém světě lidé používají Raspberry Pi k osvojení programovacích dovedností, vytváření hardwarových projektů, a především k domácí automatizaci. Raspberry se stalo velmi oblíbené hlavně díky své ceně, která se pohybuje v rozmezí 157–1 536 Kč. Počítač běží na operačním systému Linux a obsahuje výstup pro připojení monitoru (HDMI) a přes USB je možné připojit klávesnici a myš. Výkonem lze srovnat se smartphonem. (“What is a Raspberry Pi?”, 2021)

Prodej raspberry Pi byl spuštěn v roce 2012 a od té doby bylo vydáno několik dalších variant. Původní Pi měl jednojádrový procesor 700 MHz a pouze 256 MB RAM, zatímco nejnovější model má čtyřjádrový procesor 1,4 GHz s 1 GB RAM. (“What is a Raspberry Pi?”, 2021) V následující tabulce naleznete přehled základních modelů a níže ještě popis Raspberry, která jsou vhodná pro 3D tiskárny.

Tabulka 5: Přehled specifikací jednotlivých modelů Raspberry Pi

	Model 1 B	Model 2 B	Model 3 B	Model 4 B	Zero W
Cena	35$	35$	35$	35, 45, nebo 55$ (podle modelu)	10$
Procesor	700 MHz single-core	900 MHz 32-bit čtyřjádrový	1,2 GHz 64-bit čtyřjádrový	1,5 GHz 64-bit čtyřjádrový	1 GHz single-core
Paměť	256 MiB	1 GiB	1 GiB	1, 2, 4 nebo 8 GiB	512 MiB
USB 2.0	1	4	4	2	1
USB 3.0	žádný	žádný	žádný	2	žádný
Síť	žádný	10/100 Mbit Ethernet USB adaptér	10/100 Mbit Ethernet, 802.11n Wi-Fi, Bluetooth 4.1	10/100/1000 Mbit Ethernet, 802.11ac Wi-Fi, Bluetooth 5.0	802.11n Wi-Fi, Bluetooth 4.1

Zdroj: (“Raspberry Pi”, 2021), (Raspberry Pi 3 Model B 64-bit 1GB RAM, 2021)

Raspberry Pi 3

V únoru 2016 byl uveden do prodeje model Raspberry Pi 3, který je oproti modelu 2 B vybaven WiFi a Bluetooth 4.1. Pi 3 B má navíc ve srovnání se starším Pi 2 B o 33 % vyšší frekvenci procesoru (1.2 GHz vs 0.9 GHz) a modernější jádro. Rozměry a rozložení konektorů Pi 3 B jsou shodné s předchozím Pi 2 B. Při běžném používání bude Pi 3 B asi o 50 % rychlejší než Pi 2 B se stejným software. Náskok Pi 3 B se bude dále zvyšovat s pokrokem optimalizace software na novou 64-bitovou architekturu. Celkový nárůst výkonu a integrace WiFi + Bluetooth s sebou nesou i vyšší požadavky na napájení a chlazení. (Raspberry Pi 3 Model B 64-bit 1GB RAM, 2021)

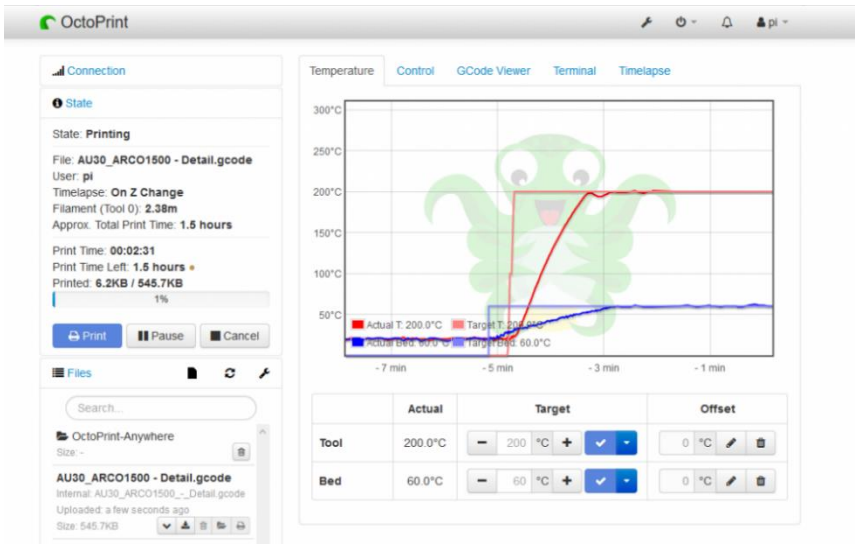
Raspberry Pi 4

V červnu 2019 byl uveden do prodeje model Raspberry Pi 4. Jeho základem je 64bitový čtyřjádrový procesor s vyšší frekvenci procesoru (1,5GHz). Oproti Raspberry Pi 3 se Raspberry Pi 4 prodává ve třech variantách podle velikosti RAM. Dále zde byly dva porty USB 2.0 nahrazeny dvěma porty USB 3.0. (“Raspberry Pi”, 2021)

2.3.2 OctoPrint

OctoPrint je webové rozhraní pro 3D tiskárny, které je open-source a je zcela zdarma. Aplikace umožňuje ovládat a sledovat tisk přímo z webového prohlížeče, stačí OctoPrint připojit k místní síti. Umožňuje poslat několik souborů G-kódu na různé tiskárny prostřednictvím jediného rozhraní. Je také kompatibilní s fotoaparáty, které umožňují sledovat tiskovou úlohu v reálném čase a vzdáleně kontrolovat, zda vše funguje správně. Většina uživatelů si ji dnes instaluje na Raspberry Pi, který je připojený k tiskárně. (Aysha, 2021)

Jak již bylo řečeno hlavním účelem aplikace OctoPrint je dálkové ovládání 3D tiskárny a sledování vývoje tisku 3D modelu. Lze sledovat vývoj teploty extruderu a tiskové podložky, samozřejmě lze také tyto parametry upravovat. Aplikace také umožňuje zahájit tisk přes webové rozhraní (Obrázek 8) a také tisk kdykoli pozastavit či zastavit. OctoPrint dále nabízí funkce pro slicování modelů, ale tato funkcionalita není moc využívána. Přece jen slicovací nástroje od výrobců tiskárny jsou o něco přesnější a optimalizované na příslušnou tiskárnu. Pokud jde o kompatibilitu, OctoPrint v současné době podporuje většinou FDM 3D tiskáren. (Aysha, 2021)



Obrázek 8: Webové rozhraní OctoPrint

Zdroj: (Škoula, 2021)

Díky tomu, že je aplikace open-source, vytvořila se komunita lidí, kteří pro zábavu vyvíjejí nové pluginy pro OctoPrint. V dnešní době je na výběr z nepřeberného množství rozšíření. Některá rozšíření počítají přesné náklady na tisk modelu, jiné zobrazují počet vrstev tisku modelu a mnoho dalších rozšíření. (Pranav, 2021)

2.3.3 API

API (Application Programming Interface) je zkratka pro aplikační programovací rozhraní. Je to způsob komunikace, který je využíván k interakci mezi jednotlivými softwarovými komponentami. API je definováno jako "specifikace všech možných interakcí se softwarovou komponentou". Co to přesně znamená? Představme si, že auto bylo softwarovou součástí. Jeho API by obsahovalo informace o tom, co auto umí – zrychlit, zabrzdit, zapnout rádio atd. Také by obsahovalo informace o tom, jak to můžete udělat. Chcete-li například zrychlit, položte nohu na plynový pedál a zatlačte. API nemusí vysvětlovat, co se děje uvnitř motoru, když sešlápneme plynový pedál, ale přesně říká, co a jak máme udělat, abychom dostali chtěnou interakci. (Freeman, 2019)

Pokud jde o software, API jsou doslova všude. Jen při objednání balíčku z Alzy se provolá nespočet API. Od stisku tlačítka si jednotlivé systémy mezi sebou musí vyměnit ID objednaných produktů, počet produktů, údaje o platbě, dodací adresu a další údaje potřebné k objednávce. Tyto informace si mezi sebou vyměňují všechny systémy potřebné k tomu, aby byl balíček doručen k nám domů. Při tomto procesu pravděpodobně používáte přímo desítky, ne-li stovky API, a každé z těchto API se spoléhá na jinou API službu. (Freeman, 2019)

REST API

REST API je rozhraní pro programování aplikací, které vychází z omezení architektury stylu REST a umožňuje interakci s webovými službami. Mezi základní principy REST patří bezestavovost, což znamená, že každý požadavek obsahuje veškeré informace pro jeho vykonání. Dále má pevně stanovený počet metod. JSON je aktuálně nejpopulárnější způsob zápisu, který se pro REST API používá, je čitelný jak pro lidi, tak pro stroje. (What is a REST API?, 2021)

REST API je tvořeno čtyřmi základními metodami:

- GET: metoda pro získání dat
- POST: metoda pro vytvoření dat

- DELETE: metoda slouží k mazání existujících dat
- PUT: metoda slouží k úpravě existujících dat

Na obrázcích (Obrázek 9 a 10) vidíte ukázku dokumentace pro použití API. První obrázek zachycuje metodu GET s potřebným informacemi pro složení dotazu. Druhý obrázek zachycuje příklad úspěšné odpovědi s daty v těle ve formátu JSON.

```
GET /api/version HTTP/1.1
Host: example.com
X-Api-Key: abcdef...
```

Obrázek 9: Metoda GET

Zdroj: (Octoprint Documentation, 2021)

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "api": "0.1",
  "server": "1.3.10",
  "text": "OctoPrint 1.3.10"
}
```

Obrázek 10: Úspěšná odpověď na zvolený dotaz v jazyce JSON

Zdroj: (Octoprint Documentation, 2021)

2.4 Architektura mobilních aplikací

V procesu vytváření jakékoli mobilní nebo webové aplikace bychom se měli ujistit, že je každá součást aplikace dobře sestavena a tvoří logický celek. Dokonce i ten nejmenší problém v architektuře aplikace může velmi zhoršit celkovou kvalitu a znesnadnit vývoj. Aplikace s dobrou architekturou se zpravidla v pozdějších fázích vývoje vyvíjejí rychleji a lépe se udržují. (Basics: Mobile App Architecture & How to Start Building One in 2020, 2020)

Architektura mobilních aplikací je sada vzorů a technik, které vývojáři dodržují při vytváření plně strukturované mobilní aplikace. Dobrá architektura rozděluje jednotlivé komponenty aplikace do více vrstev, kde má každá vrstva trochu jinou funkcionalitu. Jednou z nejpopulárnějších vícevrstvých architektur je třívrstvá architektura, kterou tvoří vrstva prezentační, aplikační a datová. Prezentační vrstva slouží k zobrazení informací uživateli a k interakci s uživatelem. Aplikační vrstva obsahuje logické jádro aplikace, tedy funkce pro zacházení s daty. Datová vrstva představuje

úložiště dat. Mezi nejčastěji používané vícevrstvé architektury mobilních aplikací patří MVC a MVVM, které budou více vysvětleny v této kapitole. (Čápka, 2017)

2.4.1 MVC

MVC architektura (Obrázek 11) rozděluje komponenty aplikace do 3 typů, hovoříme tak o Modelu, View a Controlleru. Z tohoto rozdělení vychází i název celé architektury MVC = Model-View-Controller. (Čápka, 2013) Tento typ architektury se nejdříve začal používat při vývoji webů, ale dnes je i hodně rozšířen v mobilním vývoji.

Model

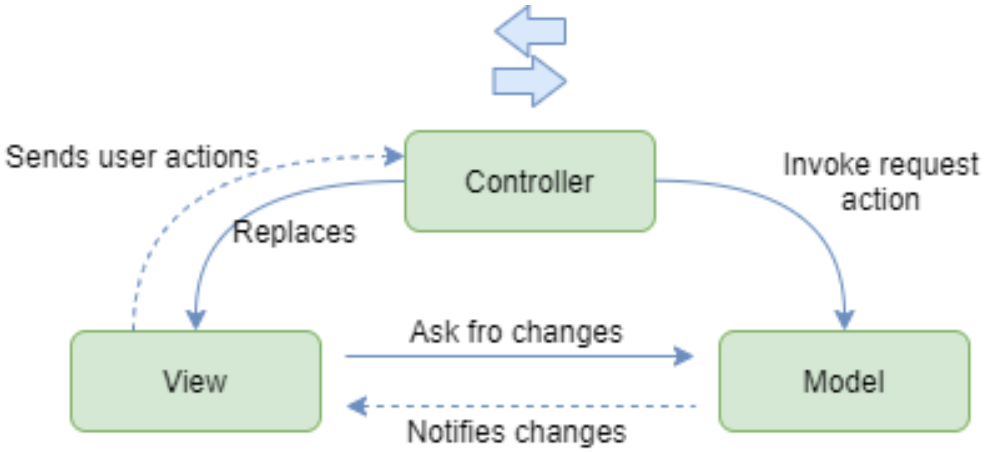
Vrstva modelu obsahuje logiku aplikace a vše, co do ní spadá. Obsahuje potřebné výpočty, databázové dotazy, validace povinných polí atd. Jeho hlavní funkce spočívá v přijetí parametrů zvenku a vydání ven. Model vůbec neví ani se nestará o to, odkud data přišla a jak budou výsledná data zformátována a vypsána. (Čápka, 2013)

View

Tato vrstva se stará o zobrazování výstupů uživateli. Uživateli zobrazuje design, grafiku příslušné stránky aplikace. Zobrazuje tedy hlavně to, co bylo navrhnuto na jednotlivých stránkách při prototypování aplikace. (Čápka, 2013)

Controller

Controller spojuje View a Model, ale Contoller není prostředník mezi nimi. Controller mění Model na základě aktivity uživatele ve View a aktualizuje i View. (Evaluating Design Patterns for Mobile Development, 2013)



Obrázek 11: Interakce mezi vrstvami modelu

Zdroj: (Evaluating Design Patterns for Mobile Development, 2021)

Jak ukazuje diagram výše, View může přímo přistupovat k Modelu, ale jen v režimu read-only, View by nemělo nijak měnit stav Modelu. To může Controller, ale ten však nereaguje přímo na View. Odpovědností View je dotázat se na stav změněného Modelu, o to se Controller nestará. Controller hraje roli, až když je třeba poskytnout nové View. (Evaluating Design Patterns for Mobile Development, 2021)

Tabulka 6: Výhody a nevýhody architektury MVC

Výhody	Nevýhody
Snadnější paralelní vývoj	Těžší orientace v kódu
Jednodušší provádění změn v kódu	Trochu jiné přístup k této architektuře na obou mobilních platformách
Nízké propojení mezi vrstvami	

Zdroj: ("Why MVC Architecture?", 2017)

2.4.2 MVVM

MVVM (Obrázek 12) stejně jako MVC rozděluje komponenty aplikace do tří vrstev, a to do vrstvy Modelu, View a vrstvy View Modelu. Začáteční písmena těchto vrstev tvoří název této architektury. Současně je to jedna z nejpoužívanějších architektur na mobilní platformě iOS i Android. (Mathur, 2020)

Model

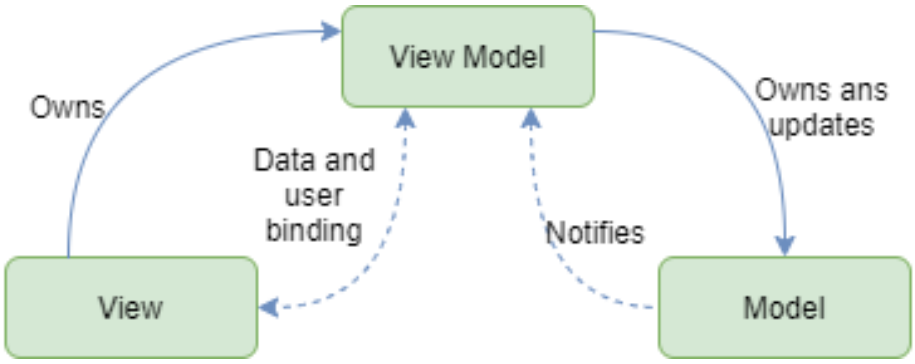
Model představuje jednoduchá data. Jednoduše uchovává data a nemá nic společného s logikou aplikace. Můžeme jednoduše říci, že jde o prostou strukturu dat, která získáváme prostřednictvím služeb API. (Mathur, 2020)

View

Tato vrstva zobrazuje, co uživatel vidí, a jejím účelem je informovat View Model o akci uživatele.

View Model

View Model slouží jako prostředník mezí View a Modelem. View Model iniciuje změny v Modelu a také se sám aktualizuje. Představuje vrstvu aplikační logiky aplikace. (Evaluating Design Patterns for Mobile Development, 2021)



Obrázek 12: Interakce mezi vrstvami modelu MVVM

Zdroj: (Evaluating Design Patterns for Mobile Development, 2021)

Jak je znázorněno na diagramu (Obrázek 12) jedinou zodpovědností View je vlastnit View Model, delegovat akce od uživatele do View Modelu a poslouchat změny, které mu právě View Model poskytuje. Mezi akce od uživatele může patřit například stisknutí tlačítka, přechod na další obrazovku apd. O veškerou logiku se stará pouze View Model. Ten současně nic neví o samotném View, které ho vlastní. Teoreticky může jeden View Model sloužit pro různé typy View. Například může existovat jedno View pro telefon a druhé View pro tablet. Každé View může data zobrazovat trochu jinak, ale obě View mohou být napojena na ten samý View Model. View Model se kromě aplikační logiky stará také o to, aby si držel Model s daty, která mohou být získávána z různých zdrojů jako API nebo databáze. Stejně jako View Model nic neví o View, tak Model nic neví o View Modelu.

Tabulka 7: Výhody a nevýhody architektury MVVM

Výhody	Nevýhody
Snadnější paralelní vývoj	Pro jednoduchá UI je příliš složitá
Snižuje množství bussiness logiky aplikace	U velkých aplikací je obtížné správně navrhnout View Model
Jednodušší provádění změn v kódu	

Zdroj: (MVVM pros and cons, 2021)

2.5 Technologie a nástroje pro vývoj mobilní aplikace

Vývojovým nástrojem je počítačový program, který vývojáři používají k vytváření, úpravám, údržbě a ladění aplikací při vývoji. Vývojové nástroje mohou mít mnoho podob, jako jsou například kompilátory, formátovače kódu, debugger, nástroje pro analýzu výkonu a mnoho dalších. Při výběru vhodného nástroje je třeba brát v potaz,

že i tyto prostředky budou mít vliv na kvalitu vytvořeného produktu a je dobré jejich výběr nepodcenit. Navíc prací v těchto systémech se při vývoji stráví velké množství času, a je tak dobré vybrat nástroj, který je uživatelsky přívětivý a dobře se s ním pracuje.

Samozřejmě největší vliv na výběr vývojového nástroje má sám druh aplikace, kterou chceme vytvořit. Záleží, zdali vyvíjíme aplikaci na desktop nebo na mobilní zařízení, na jakou platformu je aplikace určena atd. Všechny tyto faktory, a ještě další, mají vliv na výběr vhodného nástroje. V této kategorii budou popsány často používané nástroje pro vývoj aplikace na mobilní zařízení s operačním iOS, protože vývoji na toto zařízení se budeme více věnovat v praktické části.

2.5.1 Swift

Swift je kompilovatelný programovací jazyk pro aplikace iOS, macOS, watchOS, tvOS. Byl vytvořen v roce 2014 společností Apple. Cílem tvůrců jazyku Swift bylo vytvořit nejlepší volně dostupný jazyk pro programování aplikací na mobilní telefon, tak i na desktopová zařízení. Swift je navržen tak, aby vývojářům co nejvíce usnadnil psaní a údržbu programů v tomto jazyce. Tvůrci jazyku si zakládají na těchto třech hodnotách jazyku: (About Swift, 2021)

Bezpečnost – jeho syntaxe nutí k psaní čistého a konzistentního kódu. Swift se snaží předejít chybám a zlepšuje čitelnost kódu.

Rychlost – Swift byl vytvořen s ohledem na výkon. Swift je 2,6x rychlejší než Objective-C a 8,4x rychlejší než Python.

Oblíbenost – tvůrci si zakládají na tom, aby se uživateli s jazykem dobře pracovalo a byl to jejich oblíbený jazyk.

Historie

Společnost Apple s vývojem tohoto jazyka začala od roku 2010, o čtyři roky později Swift představila na konferenci Worldwide Developers Conference, kterou společnost sama pořádala. První reakce na tento jazyk byly velmi smíšené. Někteří vývojáři byli nadšeni z jeho funkcí, flexibilitou a jednoduchostí, zatímco jiní ho kritizovali. Většina z nich přesto souhlasila s tím, že je příliš brzy na to, aby byl Swift používán pro produkční aplikace. Jazyk se ale rychle vyvíjel a s každou další verzí byl zlepšován. (The Good and the Bad of Swift Programming Language, 2021)

Zásadní zlom přišel v roce 2015, kdy se Apple rozhodl jazyk udělat open-source, Po tomto kroku byl jeho růst obrovský a během tří let se stal desátým nejoblíbenějším programovacím jazykem. (The Good and the Bad of Swift Programming Language, 2021)

V březnu 2019 byla oficiálně vydána nejnovější verze Swift 5.0. Součástí této verze bylo i SwiftUI, což je v podstatě knihovna ovládacích, grafických prvků a jejich rozvržení, které lze použít k návrhu UI pro různá zařízení Apple. (The Good and the Bad of Swift Programming Language, 2021)

Vlastnosti

Swift byl vyvinut jako náhrada za dřívější programovací jazyk používaný Applem, Objective- C. Swift je alternativou k tomuto jazyku, která využívá modernější koncepty programovacího jazyka a snaží se o jednodušší syntaxi. Ukázku kódu v jazyce Swift naleznete na ukázce níže (Ukázka kódu 1). (The Good and the Bad of Swift Programming Language, 2021)

```swift
//definice proměnné, nasledovaný jménem proměnné, typem a hodnotou
var str: String = "Hello, playground"

//pokud je typ vynechán, je použit typ proměnné
var number = 10

//definice konstanty let, nasledovaný jménem proměnné, typem a hodnotou
let siblings = ["Ellie", "Jon"]

//definice podmínky
if siblings.isEmpty {
    print("I am an only child.")
} else {
    print("I have \(siblings.count) siblings.")
}

//definice funkce s jejími parametry a typem, co vrací
func myAge (name: String, age: Int) -> String {
    let result = "\(name) is \(age) old."
    return result
}
```

Ukázka kódu 1: Definice základních prvků v jazyce Swift

Zdroj: Autor práce

SwiftUI

SwiftUI je inovativní a výjimečně jednoduchý způsob budování uživatelských rozhraní napříč všemi platformami Apple. Umožňuje vytvářet uživatelská rozhraní pro

jakékoli zařízení Apple, pomocí pouze jedné sady nástrojů. Díky stávající syntaxi Swift je kód SwiftUI velmi srozumitelný, dobře čitelný a UI se v něm píše velmi rychle. Díky automatické podpoře lokalizace, tmavého režimu a rychlosti vytváření UI je SwiftUI moderní a mocný nástroj novodobého vývoje aplikací na zařízení Apple. Na mobilních zařízeních je SwiftUI podporováno od verze iOS 13. (SwiftUI: Better apps. Less code., 2021)

Tabulka 8: Výhody a nevýhody jazyka Swift

Výhody	Nevýhody
Rychlý vývojový proces	Jazyk je stále docela mladý
Čitelný kód	Špatná spolupráce s nástroji třetích stran
Moderní jazyk, neustále se vyvíjející	
Automatická správa paměti	

Zdroj: (The Good and the Bad of Swift Programming Language, 2021)

2.5.2 Xcode

Xcode je vývojové prostředí společnosti Apple, které obsahuje profesionální vývojářské nástroje pro vývoj softwarových aplikací pro různé platformy Apple zařízeních. Nejnovější dostupná verze je Xcode 12, který se snaží co nejvíce přizpůsobit vývoji prostřednictvím SwiftUI. Xcode je obrovská aplikace, kde lze nastavit počáteční nastavení projektu, projekt debutovat, zobrazovat na různých zařízení, provést nastavení před publikací projektu a mnoho dalšího. Proto si v této kapitole popíšeme jen jeho část, kde se při vývoji aplikace nejčastěji pohybujeme (Obrázek 13). (Knaster et al., 2012)

Panel nástrojů – zde je umístěna většina ovládacích prvků a také vidíme místo projektu, kde se aktuálně nacházíme.

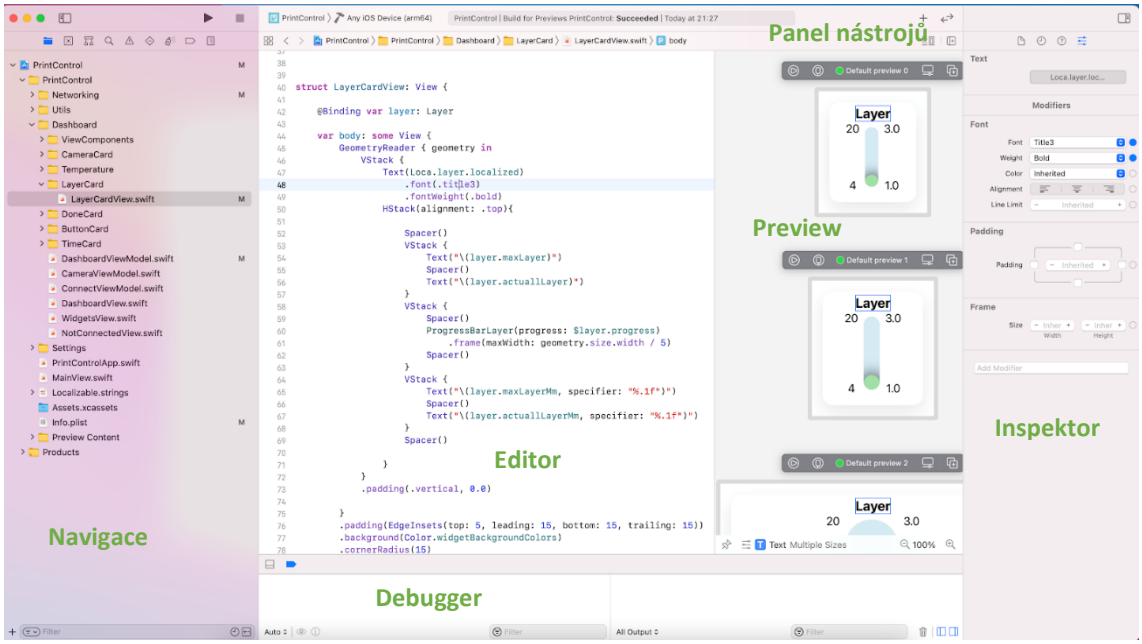
Navigace – toto podokno obvykle obsahuje seznam souborů projektu. Zde vytváříme strukturu, jak chceme mít projekt logicky rozčleněn.

Editor – v tomto podokně upravujeme zdrojový kód projektu.

Debugger – zde je umístěno podokno debuggeru, kam se vypisují informace potřebné k ladění aplikace.

Preview – zde se zobrazují náhledy vytvořeného View, u kterých si lze nastavovat různé parametry pro lepší představu o zobrazení prvku na různých zařízeních.

Inspektor – v této oblasti se vpravo zobrazují informace a ovládací prvky ke zvolenému objektu v editoru. Zde lze objekt v omezené míře editovat.



Obrázek 13: Uživatelské rozhraní Xcode

Zdroj: Autor práce

2.5.3 Git

Git je open-source systém na správu a kontrolu verzí systému, který je dostupný zdarma. S jeho vývojem v roce 2005 začal známý finský programátor Linus Torvalds, protože systém BitKeeper, který dosud používal k verzování, ukončil bezplatné používání. A tak se Linus rozhodl vytvořit podobný, ale lepší a rychlejší systém. Systém za celou dobu své životnosti prošel velkými změnami a dnes nabízí i další aplikace, které jsou postavené na jeho repozitářích, například jako GitLab, GitHub a mnoho dalších. (Brown, 2018)

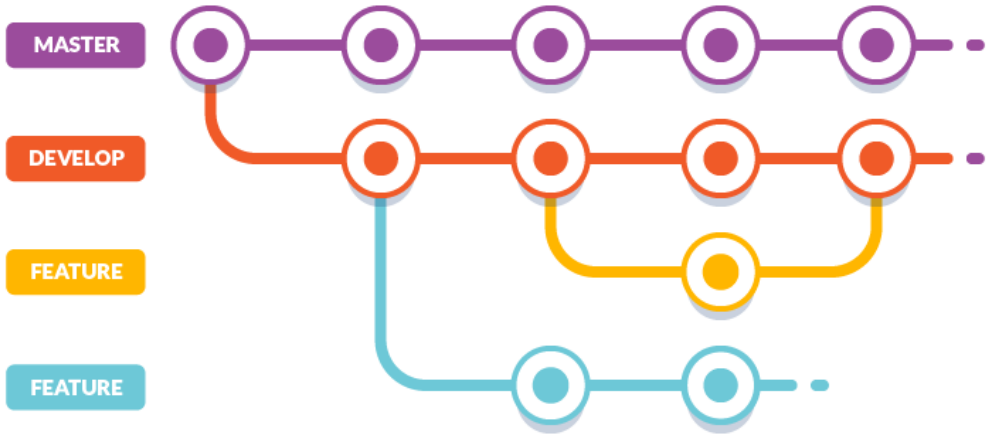
Systém Git dělá velmi užitečný fakt, že na reálném projektu vždy pracuje více lidí najednou a je tedy zapotřebí systém pro správu verzí, jako je Git, aby se zajistilo, že mezi vývojáři nedojde ke konfliktům v kódu. Git má vzdálené úložiště, které je uloženo na serveru, a místní úložiště, které je uloženo v počítači každého vývojáře. To znamená, že kód není uložen pouze na centrálním serveru, ale úplná kopie kódu je k dispozici ve všech počítačích vývojářů, a tak každý vývojář může paralelně pracovat na projektu s jiným vývojářem. Navíc se dnes požadavky na vyvíjený systém v těchto projektech často mění, díky Gitu se vývojáři mohou snadno vrátit ke starším verzím kódu. (Sridhar, 2018)

Branching and Merging (větvení a slučování) jsou dvě funkcionality Gitu, které tento systém dělají jedinečný od ostatních systémů na správu verzí systému. Git umožňuje mít více vývojových větví, které jsou zcela nezávislé na ostatních. Princip zacházení s větvemi by mohl být následující, samozřejmě záleží na konkrétním projektu. Programátor si může ve své vývojové větvi zkoušet nejrůznější věci, až když je s úpravami spokojený, nahraje tuto verzi do testovací větve, kde funkcionality vyzkouší tester, a pokud je vše v pořádku, tak až v tuto dobu budou změny nahrány do master (hlavní, produkční) větve. Tento proces je zachycen na obrázku níže (Obrázek 14). Funkcionality branching and merging dále umožňují: (About: Branching and Merging, 2021)

Větve dle rolí – Git umožňuje založit větve, které obsahují pouze funkcionality pro produkční systém, větve pro testování nových funkcionalit a několik menších větví pro každodenní vývoj.

Větve dle funkcionalit – lze založit větvi pro každou novou funkcionalitu a jednoduše mezi větvemi přepínat tam a zpět.

Jednorázové experimentování – lze vytvořit větev, ve které budete experimentovat. Až si uvědomíte, že experiment nefunguje, jednoduše ji smažete a ani nikdo jiný nikdy neuvidí, o co jste se pokoušeli.



Obrázek 14: Systém větví v Gitu

Zdroj: (“5 Git workflows and branching strategy you can use to improve your development process”, 2020)

2.5.4 Postman

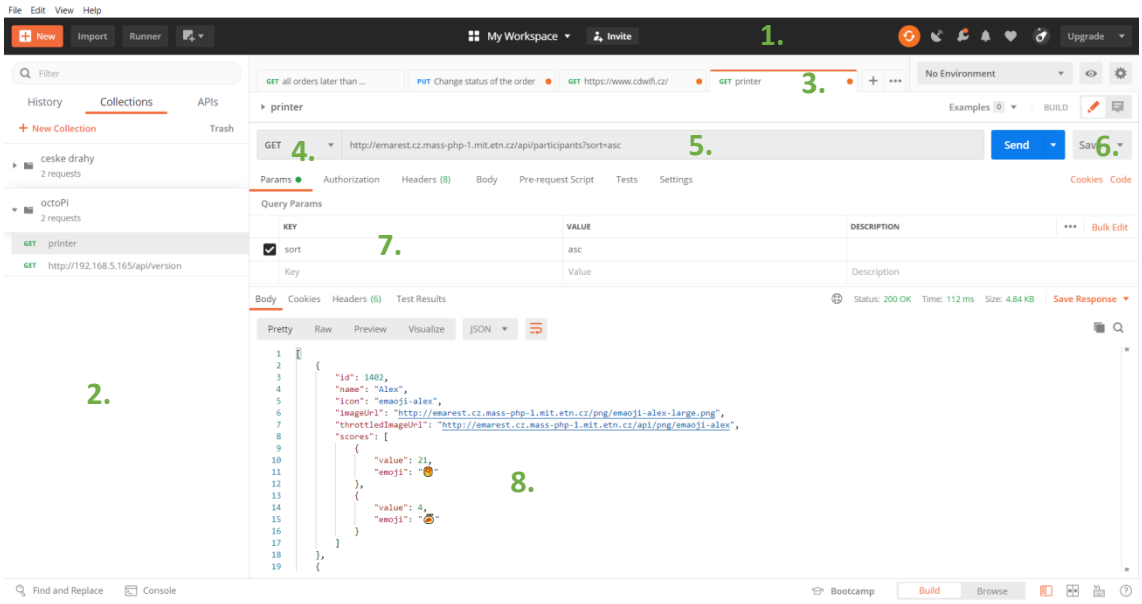
Postman je nástroj pro testování API, který začal v roce 2012 vyvíjet jako svůj vedlejší projekt Abhinava Asthany. Jeho cílem bylo vyvinout systém, který by usnadňoval práci a testování API při vývoji. (Postman Tutorial: How to use Postman Tool for API Testing, 2021)

Postman vývojářům usnadňuje vytváření, sdílení, testování a dokumentování API. Funguje na principu, že uživatelům umožní vytvářet a ukládat jednoduché i složité dotazy a také číst jejich odpovědi. (Sarir, 2020) Ke komunikaci s většinou systému existuje dokumentace k jejich API, ale mnohdy jsou v dokumentaci chyby a není uvedená úplná struktura odpovědí. Proto je dobré, než začneme vyvíjet, si ověřit, co daná služba ve skutečnosti vrací, a k tomu se právě používá Postman. Díky němu je celý proces vývoje rychlejší a efektivnější.

Uživatelské rozhraní je tvořeno z těchto částí (Obrázek 15):

1. Navigační lišta –zde nalezneme základní ovládací prvky, možnosti založení nového dotazu, možnost importování souboru dotazů a mnoho dalších nastavení Postmana.
2. Uložené dotazy – v této sekci se nachází přehled kolekcí našich uložených dotazů a můžeme si je zpětně procházet a upravovat.
3. Otevřené dotazy – tato část je tvořena záložkami, podobně jako ve webovém prohlížeči můžeme jednoduše procházet z prohlížení jednoho dotazu na prohlížení jiného dotazu.
4. Výběr metody – na obrázku je zachycena metoda GET, která se používá pro získání odpovědi
5. Adresa dotazu – do tohoto pole uživatel vkládá URL adresu aktuálního dotazu.
6. Možnosti provolání a uložení – Zde se nachází možnost na provolání dotazu a následně i možnost na uložení dotazu do příslušné kolekce.
7. Další parametry dotazu – Zde uživatel nalezne další možnosti nastavení parametru dotazu jako je například API key.

8. Odpověď – V této sekci se uživateli zobrazí odpověď na jeho dotaz, lze si přepínat mezi formáty odpovědi. Na obrázku je zachycena odpověď ve formátu JSON.



Obrázek 15: Uživatelské rozhraní Postmanu

Zdroj: Autor práce

2.6 Distribuce mobilní aplikace

Tato kapitola bude zcela zaměřena jen na distribuci aplikací pro mobilní zařízení platformy iOS. Na rozdíl od platformy Android, zde není distribuce tak zcela jednoduchá, je to docela složitý proces. Balíček aplikace si nelze jen tak stáhnout z počítače a nainstalovat do svého zařízení. Je vždy potřeba aplikaci instalovat do zařízení přes nějaký distribuční kanál.

Každá aplikace pro iOS zařízení navíc musí obsahovat provisioning profile, který je určen typem distribuce. Tento profile musí být podepsán platným certifikátem od společnosti Apple. Certifikát lze vytvořit přímo ve vývojovém prostředí Xcode nebo ve webovém rozhraní Apple Developer, ve kterém lze navíc i nastavit mobilní zařízení pro přístup k jednotlivým aplikacím a také zde lze uhradit platby spojené s některými druhy distribuce aplikace, nastavit klíče pro notifikace a mnoho dalšího. Všechny vytvořené certifikáty na podepsání aplikací musí mít speciální nastavení odpovídající zvolenému druhu distribuce příslušné aplikace. Pokud se tato nastavení nebudou shodovat, aplikaci nebude možné na zařízení spustit. Apple rozlišuje čtyři druhy distribuce aplikací: distribuce pro App Store, Ad Hoc distribuce, Enterprise a Developer distribuce. Dále

v této kategorii budou více popsány jednotlivé druhy distribuce a kanály, přes které lze aplikace na iOS distribuovat.

App Store distribuce

Tento druh distribuce je vhodný, pokud chceme, aby k naší aplikaci měla přístup široká veřejnost. Je to nejčastěji používaný typ distribuce na iOS zařízeních. Aplikace s tímto druhem distribuce můžeme distribuovat přes distribuční službu App Store, kterou v sobě mají všechny iOS zařízení. Lze také zpoplatnit stažení naší aplikace, z čehož si Apple bere určitá procenta. (Marsiglia, 2021)

Tabulka 9: Výhody a nevýhody distribuce pro App Store

Výhody	Nevýhody
Veřejný přístup ke stažení aplikace	Aplikace musí projít review
Jednoduché vydání dalších verzí	
Lze na aplikaci vydělat	

Zdroj: (Marsiglia, 2021)

Distribuce Ad Hoc

Způsob distribuce aplikace na omezené množství zařízení, aniž by byla aplikace zpřístupněna široké veřejnosti. Maximální možný počet zařízení je 100 a musíme znát všechny UDID čísla zařízení, na kterých chceme aplikaci používat. UDID je jedinečný identifikátor pro jedno iOS zařízení, který zjistíme po připojení zařízení k MacBooku. Dále aplikace bude na zařízeních fungovat až 90 dní a po této době je třeba znovu vygenerovat nový provisioning profile, opět s platným certifikátem. (Marsiglia, 2021)

Tabulka 10: Výhody a nevýhody distribuce Ad Hoc

Výhody	Nevýhody
Nemusí projít review	Omezené množství zařízení
	Doba funkčnosti aplikace závisí na délce platnosti provisioning profilu
	Složité vydání dalších verzí

Zdroj: (Marsiglia, 2021)

Enterprise distribuce

Tato metoda distribuce má smysl, pokud chcete aplikaci zpřístupnit pouze lidem ve vaší společnosti. Široká veřejnost k ní nebude mít přístup a nejsou zde ani žádná

omezení, jako byla u předchozího druhu distribuce. Tento typ je také zpoplatněn a společnost musí splnit určité podmínky stanovené společností Apple, například musí dosahovat určitého počtu zaměstnanců. (Apple Developer Enterprise Program, 2021)

Tabulka 11: Výhody a nevýhody distribuce Enterprise

Výhody	Nevýhody
Velké množství zařízení	Placené
Vhodné pro distribuci vnitrofiremní aplikace	Aplikace musí projít review

Zdroj: Autor práce

Developer distribuce

Tato metoda distribuce je velmi podobná distribuci Ad Hoc s tím rozdílem, že aplikace zůstává v testovacím módu a lze na ní detekovat chyby v aplikaci.

Tabulka 12: Výhody a nevýhody distribuce Developer

Výhody	Nevýhody
Snadnější debugging	Omezené množství zařízení
	Doba funkčnosti aplikace závisí na délce platnosti provisioning profilu

Zdroj: Autor práce

2.6.1 App Store

App Store je distribuční služba, kterou každý může najít na svém iPhonu. Zde si lze stahovat aplikace, které jsou určené pro veřejnost. Najdeme zde jak aplikace zdarma, tak i s malým či velkým poplatkem. Lze zde distribuovat aplikace s nastavením distribuce pouze na App Store.

Aby se námi vyvinutá aplikace objevila v App Storu musí daná aplikace projít přes review. Při review je celá aplikace testována testovacím oddělením Applu, které kontroluje, zdali aplikace neporušuje určitá pravidla, je bezpečná, je správně otextovaná a další požadavky ze strany Applu. Veškerá tyto doporučení lze nalézt ve zmiňovaném portálu Apple Developer. Apple na svých stránkách uvádí, že review může trvat 24-48 hodin dle aplikace. Z mých zkušeností takové review může trvat i několik týdnů v závislosti na velikosti a složitosti aplikace. Pokud se rozhodneme, že za svoji aplikaci budeme chtít, aby nám uživatelé zaplatili, Apple si bere určité procento i z této částky. (App Store Review Guidelines, 2021)

2.6.2 TestFlight

TestFlight je služba pro distribuci aplikací k internímu či externímu testování. TestFlight umožňuje vývojářům získávat zpětnou vazbu od testerů a prohlížet zaznamenané pády aplikace. Lze zde distribuovat aplikace s nastavením distribuce pouze na App Store.

Při interním testovaní lze aplikaci distribuovat na testery, kteří mají vytvořený účet Developer pro přístup do rozhraní Apple Developer. Na mobilní zařízení si testeři nainstalují mobilní aplikaci TestFlight a zde uvidí aplikaci k testování, také se jim zde budou zobrazovat nové verze aplikace k testování.

Pro veřejné testování musí nejdříve aplikace projít přes review Applu, pokud bude schválena, získáme veřejný link pro distribuci aplikace. Tento link zašleme osobám, co chceme, aby se podíleli na testování naší aplikace. Tyto osoby vůbec nemusí mít účet Developer. Stačí, když si stáhnou aplikaci TestFlight, zde vloží link a aplikace se jim zobrazí ke stažení. Jakmile bychom chtěli distribuovat další verzi aplikace k testování tímto způsobem, musela by aplikace znovu projít přes review a testerům bychom posílali další link pro stažení nové verze.

Aplikace distribuované tímto způsobem jsou pro testování dostupné 90 dní od buildu aplikace a po této době již aplikace nepůjde otevřít. (Testing Apps with TestFlight, 2021) Přes aplikaci TestFlight mohou testeři také zasílat feedback k testovaným aplikacím.

2.6.3 Visual Studio App Center

Visual Studio App Center je služba pro aplikace na iOS, Android, macOS a Windows zařízení, která nabízí velkou škálu funkcionalit. Lze zde koordinovat celý životní cyklus aplikace od automatických buildu, testování, distribuce, analýzy pádů a monitoringu užívání aplikace. (Introducing App Center: Build, Test, Distribute and Monitor Apps in the Cloud, 2017) V této kapitole se pouze zaměříme na možnost distribuce iOS aplikace přes tento kanál.

Pře App Center lze distribuovat aplikace s nastavením distribuce na Enterprise, Developer a Ad Hoc. Aplikaci s distribucí App Store distribuovat nelze. Služba je pro distribuci aplikací zdarma. (Zpřístupnili jsme centrum aplikací pro všechny, 2021) U některých ostatních funkcionalit jsou zde účtovány poplatky.

Pro distribuci je zapotřebí balíček aplikace nahrát do App Centra a zvolit skupinu pro distribuci aplikace. Skupina může mít definované členy, kteří se k aplikaci dostanou přes svůj e-mail anebo můžeme vytvořit skupinu veřejnou. Po zvolení skupiny nám App Center vygeneruje link, který když otevřeme v prohlížeči v telefonu, tak si můžeme do telefonu stáhnout příslušnou aplikaci. Přes veřejný link si bude moct aplikaci do telefonu nainstalovat každý, ale pozor, aplikace půjde spustit jen těm, kteří mají UDID svého telefonu přidaný v nastavení provisioning profilu aplikace.

3 Praktická část

Praktická část se bude zabývat návrhem a vývojem iOS aplikace na kontrolu 3D tiskárny, která tiskne na principu technologie FDM. Budou zde popsány využité komponenty a nástroje pro vývoj aplikace, pro propojení aplikace s tiskárnou a pro distribuci aplikace.

3.1 Návrh mobilní aplikace

Tisk 3D modelu na tiskárně může trvat několik minut ale také i několik hodin. Během celého procesu tisku je dobré model občas zkontrolovat, zdali tisk správně probíhá a nedošlo k nějaké chybě, jako například zdali se tisk neodtrhl od podložky. Uživatel tedy většinou musí osobně dojít k tiskárně a výtisk zkontrolovat. S tím by ale již měla pomoci vyvíjená aplikace v této diplomové práci, která umožní tisk pohodlně kontrolovat prostřednictvím telefonu. Aplikace by měla umět zobrazovat základní informace potřebné pro kontrolu tisku. Měla by umět zobrazit záběry z kamery, pokud je k tiskárně připojena, a také by zde měla být možnost tisk přerušit.

Vyvíjenou aplikaci bude možné nainstalovat pouze na zařízení platformy iOS. Celý proces návrhu a designu aplikace bude tak optimalizován pro tato zařízení. Aplikace bude navrhnuta tak, aby byla na těchto zařízeních uživatelsky přívětivá a podporovala nové funkcionality této platformy.

Aplikace bude fungovat pro tiskárny, kterou tisknout technologií FDM. Tato technologie byla zvolena z důvodu, že je mezi tiskaři nejvíce rozšířena a tyto tiskárny jsou velmi rozšířeny pro domácí použití. Mezi nejvíce rozšířené tiskárny této technologie patří tiskárna Prusa i3 mk3S, pro kterou je tato aplikace nejvíce přizpůsobena.

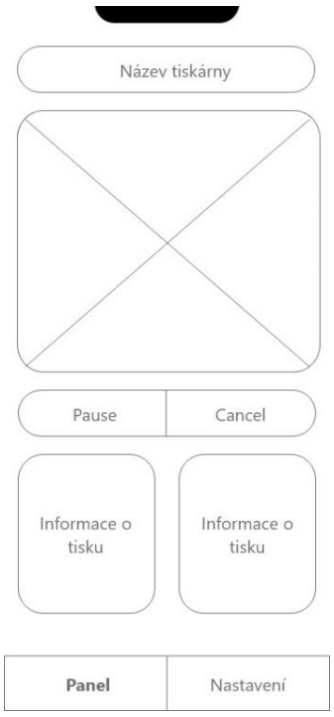
Většina 3D tiskáren není připojena k internetu, a proto bylo nutné vyřešit, jak z tiskárny získat potřebné údaje pro kontrolu tisku. Při prozkoumání konkurenčních řešení, kterých nebylo mnoho, bylo zjištěno, že většina aplikací pro kontrolu tisku využívá službu OctoPrint, která je nahraná na Raspberry Pi připojený k tiskárně. Byly prověřeny i jiné možnosti propojení aplikace s 3D tiskárnou jako například za pomoci cloudové služby Firebase, ale tyto možnosti nebyly zcela tak vyhovující, jako komunikace napřímo se službou OctoPrint. Proto byla i do grafického návrhu aplikace zapracována sekce nastavení, potřebná pro připojení k této službě.

3.1.1 Low-Fidelity prototyp

Nejdříve bylo potřeba vyřešit prvotní návrh aplikace, kde bude zachycen přibližný obsah aplikace. Byly určeny nezbytné údaje potřebné pro kontrolu tisku, mezi které patří: informace o času tisku a o teplotě tisku, náhled kamery, možnost přerušit tisk a nastavení pro připojení k OctoPrintu. Jelikož ne všechny tyto údaje má každý uživatel k dispozici, například kameru k tiskárně nemá připojený každý, bylo potřeba UI aplikace navrhnout modulárně tak, aby se vždy zobrazovaly jen ty údaje, které má daný uživatel k dispozici.

Prvotní návrh wireframů byl načrtnut na papír, kde bylo zobrazení jednotlivých druhů informací rozděleno do widgetů. Tento návrh byl překreslen do digitální formy pomocí nástroje Adobe XD. Na obrázkách (Obrázek 16 a 17) níže můžete vidět, že navigace v aplikaci byla rozdělena do dvou tabů. Do tabu s informacemi o tisku a tabu pro nastavení. Obrazovka s informacemi o tisku v horní části nese název tiskárny. Dále zde je prostor pro zobrazení kamery. Pod tímto widgetem se nachází widget pro přerušení tisku a pod ním widgety s informacemi o tisku jako je teplota a čas tisku. Prototyp si můžete vyzkoušet prostřednictvím odkazu uvedeného níže.

Odkaz na low-fidelity prototyp: https://xd.adobe.com/view/3535a211-a673-4b9e-be18-3fc880bd51d2-6066/?fullscreen



Obrázek 16: Obrazovka s informacemi o tisku

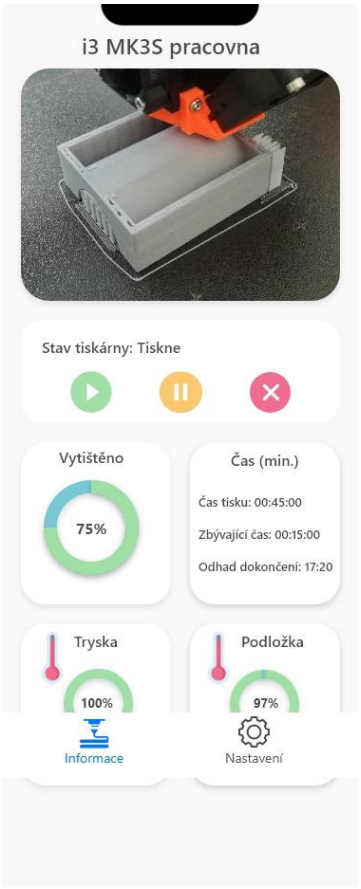
Zdroj: Autor práce

Obrázek 17: Obrazovka nastavení

Zdroj: Autor práce

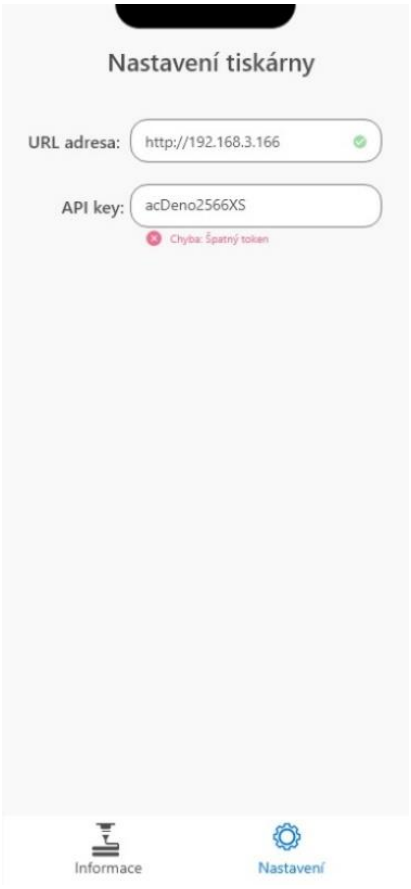
3.1.2 High-Fidelity prototyp

Low-fidelity prototyp byl dále překreslen na high-fidelity prototyp, opět prostřednictvím navrhovacího nástroje Adobe XD. V tomto návrhu (Obrázek 18 a 19) jsou již podrobněji rozkresleny jednotlivé widgety s informacemi o tisku. V horní části obrazovky zůstal název tiskárny, pod kterým je prostor pro záběry z kamery. Dále se nachází widget s tlačítky pro zrušení, přerušení a pokračování v tisku. V tomto widgetu je i obsažen současný stav tiskárny. V dalším widgetu jsou zachycena procenta dokončeného tisku. Velmi důležitý widget je s informacemi o času tisku, kde je důležité zachytit zbývající čas a odhadovaný čas dokončení tisku. Jako poslední jsou widgety, které nesou informace o teplotách podložky a trysky, ve kterých jsou procenta nahřátí na cílovou teplotu a současná teplota. Tab nastavení je již také více rozkreslen a zde jsou zachyceny potřebné údaje pro připojení aplikace k OctoPrintu.



Obrázek 18: Původní verze obrazovky s informacemi o tisku

Zdroj: Autor práce



Obrázek 19: Obrazovka nastavení

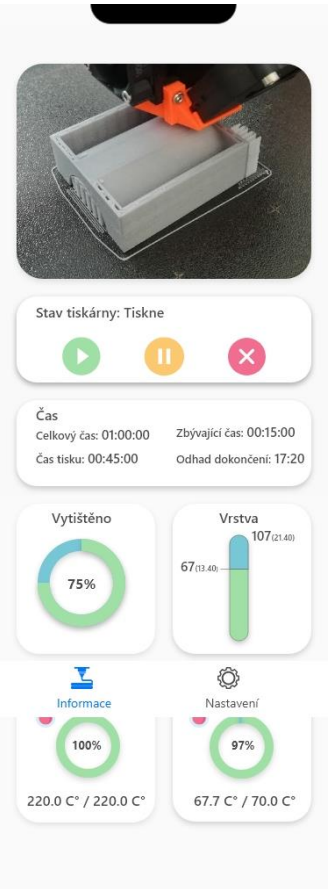
Zdroj: Autor práce

Tento prototyp byl předložen dvěma uživatelům k testu. Z pozorování vyplynulo několik postřehů. Uživatelé název tiskárny brali jako zcela zbytečný údaj, který jen

zabíral místo. Dále měli problém v orientaci ve widgetu s informacemi o času tisku. Jednomu z testovaných uživatelů v aplikaci chyběly informace o výšce a čísle aktuálně tisknuté vrstvy. S obrazovkou nastavení neměl problém žádný z uživatelů. Veškeré tyto poznámky byly znovu zapracovány do prototypu.

Z hlavní obrazovky byl odstraněn název tiskárny (Obrázek 20). Widget s informacemi o čase tisku byl rozšířen a dále byl přidán widget s informacemi o aktuální vrstvě tisku. Widgety se záběry kamery a informacemi o vrstvě se zobrazují jen pokud tyto informace o tisku aplikace má k dispozici. Obrazovka s nastavením zůstala beze změny. Dále byl prototyp doplněn o obrazovku (Obrázek 21), kdy je tiskárna vypnutá a nemáme o ní žádné informace. Tento prototyp byl již finální a dle něho bylo programováno UI aplikace.

Odkaz na high-fidelity prototyp: https://xd.adobe.com/view/2df1a1b1-eda3-481a-8d12-62e9a15f87a7-78a2/



Obrázek 20: Finální verze obrazovky s informacemi o tisku

Zdroj: Autor práce

Obrázek 21: Obrazovku pro odpojenou tiskárnu

Zdroj: Autor práce

3.1.3 Název, jazyk a ikona aplikace

Bylo navrženo, že aplikace bude podporovat dva jazyky, a to jazyk český a anglický. S ohledem na vybrané jazyky byl zvolen název aplikace jako PrintControl. Dále byla vyřešena ikona aplikace na plochu. Na pozadí ikony byla vybrána modrá barva, která je i v předchozím návrhu aplikace, a doprostřed byla umístěna ikona 3D tiskárny, která byla zdarma dostupná z online databáze. Na obrázku níže (Obrázek 22) můžete vidět ikonu a název aplikace na ploše telefonu.

Obrázek 22: Zobrazení ikony aplikace na telefonu

Zdroj: Autor práce

3.2 Komunikační rozhraní

V této kapitole bude více rozebrána komunikace mezi tiskárnou a aplikací. Jelikož tiskárna nemá možnost připojení k internetu, musel k ní být připojen Rasberry Pi, který tuto možnost má.

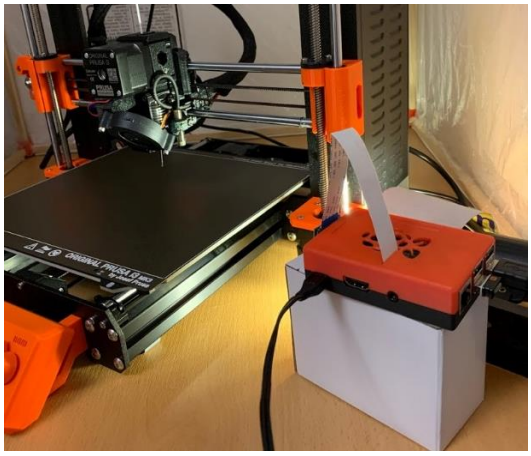
Dále bylo potřeba vyřešit, jak z tiskárny dostat potřebná data. Po prozkoumání různých možností, jak komunikovat s tiskárnou napřímo, bylo vybráno jako nejspolehlivější řešení komunikace s tiskárnou skrze OctoPrint pomocí API. Aplikace Octoprint byla nainstalována na jednodeskový počítač. Díky tomuto aplikace s tiskárnou může komunikovat v rámci jedné sítě, pokud bychom chtěli s tiskárnou komunikovat odkudkoliv, musel by jednodeskový počítač mít veřejnou IP adresu.

3.2.1 Raspberry Pi

Nejdříve bylo potřeba vybrat vhodný model Raspberry Pi, aby na něm OctoPrint v pořádku fungoval. Vybrán byl model 3B z důvodu již dřívější koupě, ale v komunitě tiskařů je také oblíben model 4. Model Zero není doporučován, jelikož disponuje poměrně pomalým procesorem.

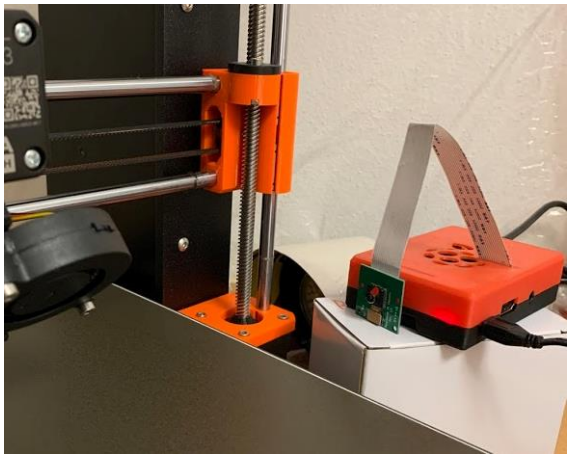
K modelu 3B byla dále připojena kamera a jednodeskový počítač byl propojen s tiskárnou přes USB kabel. Na Raspberry Pi byl dále nainstalován program OctoPrint.

Na obrázku (Obrázek 23 a 24) můžete vidět jednodeskový počítač schovaný v krabičce, která byla vytištěna na 3D tiskárně. K počítači je dále připojena kamera, která zabírá tiskovou podložku 3D tiskárny Prusa i3 mk3S.



Obrázek 23: Připojený jednodeskový počítač k tiskárně

Zdroj: Autor práce



Obrázek 24: Snímek počítače s kamerou zabírající tiskovou plochu tiskárny

Zdroj: Autor práce

3.2.2 OctoPrint

Do webového rozhraní OctoPrintu se dostaneme zadáním IP adresy OctoPrintu do našeho prohlížeče. V tomto rozhraní lze nastavit potřebná nastavení OctoPrintu, například jako přidání kamery a popřípadě lze zde stáhnout další rozšiřující doplňky této aplikace. Samozřejmě zde také zjistíme potřebné autorizační údaje pro používáni služeb API této aplikace. V dokumentaci OctoPrintu byl popsán způsob, jak se lze k používání metod API autorizovat. Je k tomu potřeba IP adresa OctoPrintu a API klíč, který je ve webovém rozhraní uveden ve formě textu, ale také i jako QR kód. Oba tyto údaje tedy byly zahrnuty do sekce nastavení aplikace PrintControl, aby je uživatel správně vyplnil.

Dále bylo potřeba vybrat vhodné API metody, abychom získali potřebná data, co chceme v aplikaci zobrazovat. Níže budou uvedeny použité metody v aplikaci a důvod jejich použití.

Stav připojení k OctoPrintu je ověřován pomocí metody GET api/version (Obrázek 25), která, když vrátí hodnotu 200, tak je uživatel správně připojen a může se pokusit o připojení tiskárny.

```
//metoda GET pro identifikaci správného připojení
GET /api/version HTTP/1.1
Host: example.com
X-Api-Key: abcdef...

//úspěšná odpověď
HTTP/1.1 200 OK
Content-Type: application/json

{
  "api": "0.1",
  "server": "1.3.10",
  "text": "OctoPrint 1.3.10"
}
```

Obrázek 25: Metoda GET pro ověření připojení

Zdroj: Autor práce

Pro připojení k tiskárně byla použita metoda POST /api/connection (Obrázek 26), kde byl využit atribut command s hodnotou connect.

```
//metoda POST pro připojení k OctoPrintu
POST /api/connection HTTP/1.1
Host: example.com
Content-Type: application/json
X-Api-Key: abcdef...

//potřebné hodnoty pro připojení
{
  "command": "connect",
}
```

Obrázek 26: Metoda POST pro připojení

Zdroj: Autor práce

Pro získání aktuálního stavu tiskárny, informaci o čase tisku a progresu tisku byla použita metoda GET api/job (Obrázek 27), která poskytuje veškeré potřebné časy pro dopočítání dalších hodnot ve widgetu s odhady časů.

```
//dokumentace metody na získání informací o čase
GET /api/job HTTP/1.1
Host: example.com
X-Api-Key: abcdef...

HTTP/1.1 200 OK
Content-Type: application/json

//úspěšná odpověď
{
  "progress": {
    "completion": 0.2298468264184775,
    "filepos": 337942,
    "printTime": 276,
    "printTimeLeft": 912
  },
  "state": "Printing"
}
```

Obrázek 27: Ukázka metody GET pro získání informací o čase tisku

Zdroj: Autor práce

K získání údajů o teplotě tiskové podložky a trysky, byla použita metoda GET api/printer (Obrázek 28), která poskytovala hodnotu aktuální teploty a cílové teploty nahřátí. Lze tak jednoduše i zjistit procenta nahřátí.

```
//metoda na získání dat i teplotě
GET /api/printer HTTP/1.1
Host: example.com
X-Api-Key: abcdef...

HTTP/1.1 200 OK
Content-Type: application/json

//struktura úspěšné odpovědi
{
  "temperature": {
    "tool0": {
      "actual": 214.8821,
      "target": 220.0,
      "offset": 0
    },
    "bed": {
      "actual": 50.221,
      "target": 70.0,
      "offset": 5
    }
  }
}
```

Obrázek 28: Metoda GET pro získání informací o teplotě

Zdroj: Autor práce

Pomocí metody POST api/job (Obrázek 29) lze ovládat tisk. Do atributu command musí být vložena hodnota akce, kterou chceme provést. Na ukázce vidíme dotaz pro zrušení tisku.

```
//ukázka dokumentace metody POST pro ovládání tiskárny
POST /api/job HTTP/1.1
Host: example.com
Content-Type: application/json
X-Api-Key: abcdef...

{
  "command": "cancel"
}
```

Obrázek 29: Metoda POST pro zrušení tisku

Zdroj: Autor práce

Aby se uživateli v aplikaci zobrazil widget s informacemi o tisknuté vrstvě, je potřeba stáhnout do OctoPrintu plugin DisplayLayerProgress, který umí aplikaci tyto údaje skrze API (Obrázek 30) poskytnout. Data o vrstvách budeme mít k dispozici pouze pokud uživatel zadá tisk do tiskárny skrze OctoPrint a ne z SD karty.

```
//metoda na získání dat o tisknuté vrstvě
GET /plugin/DisplayLayerProgress/values HTTP/1.1
Host: example.com
X-Api-Key: abcdef...

HTTP/1.1 200 OK
Content-Type: application/json

//struktura úspěšné odpovědi
{
  "height": {
    "current": "8.00",
    "currentFormatted": "8"
    "total": "15.00",
    "totalFormatted": "15",
    "totalWithExtrusion": "10.0",
    "totalWithExtrusionFormatted": "10"
  },
  "layer": {
    "averageLayerDuration": "0h:01m:03s",
    "averageLayerDurationInSeconds": 63,
    "current": "39",
    "lastLayerDuration": "0h:00m:58s",
    "lastLayerDurationInSeconds": 58,
    "total": "49"
  }
}
```

Obrázek 30: Metoda GET na získání informací o vrstvě tisku

Zdroj: Autor práce

Použití metody GET /webcam/, na získání obrázku z kamery připojené k tiskárně, bylo nalezeno v nastavení OctoPrintu. Metoda s úspěšnou odpovědí vrací obrázek.

3.3 Architektura aplikace

Pro aplikaci byla zvolena architektura MVVM, která rozděluje aplikaci do tří základních vrstev, a to vrstva Model, vrstva View a vrstva View Model.

3.3.1 Vrstva View

Vrstva View mohla být řešena pomocí frameworku UIKit nebo pomocí frameworku SwiftUI. S ohledem na to, že aplikace je určena pro nejnovější iOS zařízení, tak byla použita technologie SwiftUI, protože se v něm rychleji a jednodušeji vyvíjí. Z důvodu přehlednosti a variability byla větší View rozdělena do menších celků zvaných Subview. Výhoda menších View je možnost znovu je použít a vyvarujeme se duplicitního kódu.

3.3.2 Vrstva Modelu

V modelové vrstvě nalezneme datové modely reprezentující například teplotu, vrstvy tisku, stav tisku a další. Dále byla do této vrstvy zahrnuta i komunikační část, která získává data skrze službu API. Tato data jsou deserializována z formátu JSON do datových modelů aplikace. Do této vrstvy byla zahrnuta i logika získávání dat z lokálního

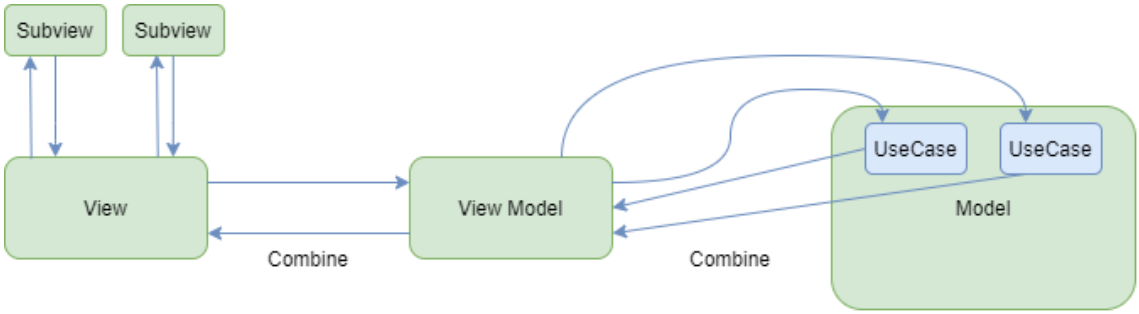
úložiště aplikace. Tedy data, která chceme, aby si aplikace pamatovala, jako je například URL adresa OctoPrintu a API klíč. Podobně jako je rozdělena vrstva View, je i tato vrstva rozdělena do menších celků, které řeší jednotlivé problémy. Tyto jednotlivé části jsou v aplikaci nazývány jako UseCase, například CancelUseCase slouží k zrušení tisku 3D modelu.

3.3.3 Vrstva View Modelu

View Model se v aplikaci stará o zprostředkování dat z Modelu do View a i opačným směrem. Současně se stará i o další logiku, jako je například frekvence obnovování dat v aplikaci. Dále si také drží stavy jednotlivých obrazovek v aplikaci. Například pokud se nepodaří připojit k tiskárně, nastaví stav, podle kterého View vrstva pozná jakou obrazovku má zobrazit. V tomto případě zobrazí obrazovku s hláškou "Tiskárna není připojena".

3.3.4 Komunikace mezi vrstvami

Pro komunikaci mezi jednotlivými vrstvami (Obrázek 31) se používá framework Combine, který zajišťuje, aby data byla doručena napříč všemi vrstvami. View si drží jednotlivá Subview, kterým deleguje data z View Modelu. Samo View si drží View Model, aby mohlo předávat akci uživatele do dalších vrstev aplikace. View Model si drží jednotlivé UseCase z Modelu, aby do nich mohlo předat akci uživatele, pokud je potřeba. Také si drží jednotlivé stavy aplikace a na základě nich, se dále rozhoduje, jaké akce je třeba provést.



Obrázek 31: Komunikace mezi vrstvami aplikace

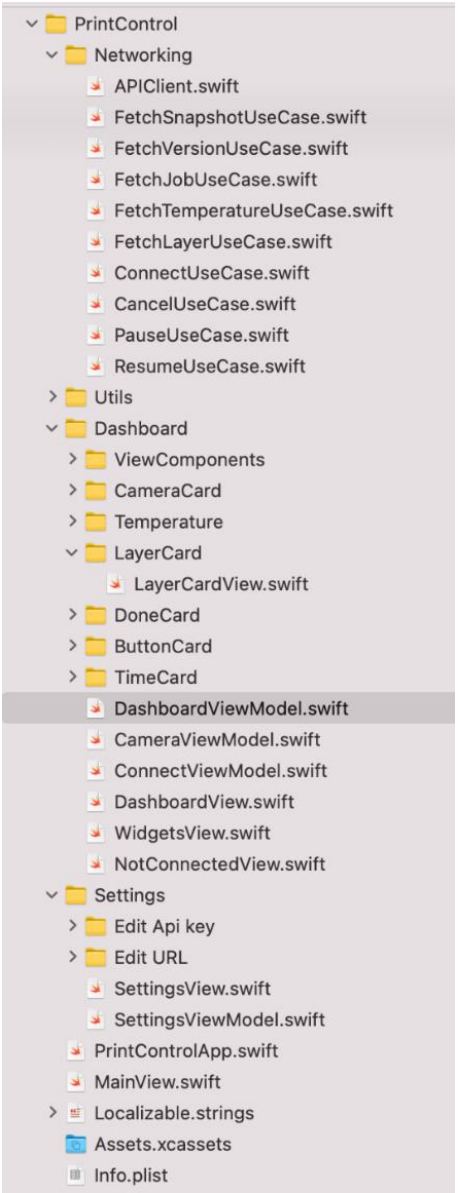
Zdroj: Autor práce

3.3.5 Struktura aplikace

Při vývoji byla snaha rozčleňovat do složek (Obrázek 32) jednotlivé prvky aplikace dle jejich souvislosti s vrstvami a funkcionalitou, kvůli lepší orientaci v kódu. Do složky Networking byli ukládány části aplikace, které souvisejí s vrstvou modelu a

zajišťují získávání dat. Zde většina souborů nese název své funkcionality a označení UseCase. Složky Dashboard a Settings představují jednotlivé obrazovky v aplikaci. Uvnitř složek nalezneme jak jednotlivá View tak i View Modely.

Složka Utils, jak již název vypovídá, obsahuje sadu pomocných metod pro definici některých prvků aplikace. V souboru Localizable jsou uloženy překlady jednotlivých textů, které se vyskytují v uživatelském rozhraní aplikace. Adresář Assets obsahuje obrázky, ikonky a barvy, použité v aplikaci. Info.plist obsahuje základní informace o aplikaci, jako je verze aplikace nebo na jaké zařízení je aplikace určena a mnoho dalších nastavení.



Obrázek 32: Struktura aplikace

Zdroj: Autor práce

3.4 Vývoj aplikace

Celá aplikace byla naprogramována v nativním programovacím jazyce Swift. Pro vývoj bylo využíváno vývojové prostředí Xcode. K zálohování kódu a verzovaní aplikace byl využit nástroj Git. Jak již bylo zmíněno výše, aplikace byla strukturována do logických celků. Jako první si více popíšeme jednotlivé komponenty, které tvoří výsledné View aplikace.

3.4.1 Jednotlivá View aplikace

View vrstava aplikace byla naprogramována v nativním frameworku SwiftUI. Celkové View je tvořeno z menších jednotlivých View, která budou nejdříve v této části popsána a dále bude ukázáno, jak je z nich složeno finální View aplikace.

Widget se snímky z kamery

Toto View (Ukázka kódu 2) slouží k zobrazování snímku z kamery připojené k 3D tiskárně. Je zde umístěn obrázek z kamery Image, který je nadefinován tak, aby zaplňoval co největší část View nad ním a přitom zůstal zachován poměr stran. Dále je zde i definován rotationEffect, který s obrázkem otočí o 180° při poklikání na obrázek. Tato funkcionalita zde byla zahrnuta z důvodu, že velmi často je kamera k tiskárně připojena hlavou dolů. U konce ZStack je nastavena barva widgetu, stín a zaoblení rohů widgetu. Toto nastavení bylo použito u všech widgetů.

```
var body: some View {
    ZStack {
        Image(uiImage: image ?? UIImage())
            .resizable()
            .aspectRatio(contentMode: .fit)
            .rotationEffect(.degrees(-rotation))
        .background(Color.widgetBackgroundColors)
        .cornerRadius(15)
        .shadow(color: Color.widgetShadowColors, radius: 15, x: 0, y: 10)
    }
    .onTapGesture {
        rotation += 180
    }
}
```

Ukázka kódu 2: Camera View

Zdroj: Autor práce

Widget se stavem tiskárny a tlačítky pro ovládání tiskárny

V tomto widgetu je uveden aktuální stav tiskárny a tlačítka na ovládání tiskárny. Jednotlivé objekty jsou v tomto View uspořádány pomocí VStack a HStack. VStack slouží k umístěním objektům nad sebe a HStack umisťuje objekty vedle sebe. V ukázce

kódu můžete vidět, že jednotlivá tlačítka a text je vedle sebe rozmístěn pomocí HStack. Tyto HStack jsou umístěny v jednom VStack, který zajistí, aby se prvky zobrazovaly nad sebou.

V ukázce kódu (Ukázka kódu 3) vidíte řešení tlačítek pro znovu spuštění tisku a pro pozastavení tisku, není zde uvedeno tlačítko pro zrušení tisku, které je řešeno velmi podobně jako tlačítko pro pozastavení tisku. Každé tlačítko je tvořeno ikonkou z volné knihovny ikonek, která byla upravena do barev vyhovující designu aplikace. Dle aktuálního stavu tiskárny jsou zablokována tlačítka, která nemají v příslušném stavu tiskárny logické použití. Pokud uživatel klikne na tlačítko pro pozastavení a zrušení tisku, aplikace mu zobrazí alert upozorňující na jeho akci, který, když potvrdí, tak se příslušná akce provede.

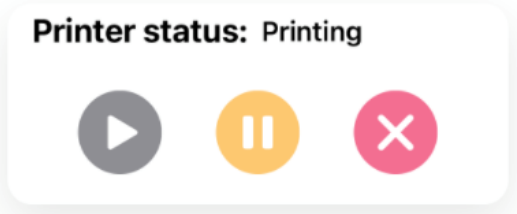
```
var body: some View {
        VStack(alignment: .leading) {
            HStack {
                Text(Loca.printerState.localized)
                    .font(.title3)
                    .fontWeight(.bold)
                Text(printerState)
                    .fontWeight(.semibold)
            }
            .padding(.bottom, 13.0)
            HStack {
                Spacer()
                Button(action: resumeAction, label: {
                    Image(systemName: "play.circle.fill")
                        .font(.system(size: 52))
                        .foregroundColor(isPlayActive ? Color.lightGreen : Color.gray)
                })
                .disabled(!isPlayActive)

                Spacer()
                Button(action: {
                    self.showPauseAlert = true
                }, label: {
                    Image(systemName: "pause.circle.fill")
                        .font(.system(size: 52))
                        .foregroundColor(isPauseActive ? Color.lightYellow : Color.gray)
                })
                .disabled(!isPauseActive)
                .alert(isPresented: $showPauseAlert) {
                    Alert(title: Text(Loca.pauseTitle.localized),
                          message:   Text(Loca.pauseText.localized),
                          primaryButton: .destructive(Text(Loca.yes.localized), action: pauseAction),
                          secondaryButton: .cancel(Text(Loca.no.localized)))
                }
```

Ukázka kódu 3: View s ovládacími prvky a stavem tiskárny

Zdroj: Autor práce

Na obrázku (Obrázek 33) s náhledem View ve vývojovém prostředí Xcode, můžete vidět tento widget zachycující stav tiskárny při jejím tisku, kdy uživatel může stisknout tlačítko pro pozastavení tisku a zrušení tisku. Tlačítko pro znovu obnovení tisku je zablokováno.



Obrázek 33: Widget s ovládacími prvky a stavem tiskárny

Zdroj: Autor práce

Widget s informacemi o čase tisku

Tento widget (Obrázek 34) slouží k zobrazení informací o čase tisku modelu. K uspořádání jednotlivých objektů je zde využit HStack a VStack (Ukázka kódu 4). Veškeré položky jsou umístěny ve VStacku, ve kterém se dále nachází HStack, který slouží k zarovnání všech čtyřech časů vedle sebe. Tento HStack dále obsahuje VStack, který nad sebe zarovnává zbývající čas a uplynulý čas tisku. Aby se název a jednotky času zobrazovaly vedle sebe, jsou umístěny ještě v jednom HStacku. V ukázce kódu je zachyceno řešení prvních dvou časů ve widgetu. Zbývající dva jsou řešeny stejným způsobem. Pro většinu výpisů časů byla využita možnost zvolit si z připravených variant formátování data. Konkrétně byl využit styl timer. Pro další byla napsána funkce timeCorrect, která zajišťovala výpis jen potřebných jednotek času. Tedy nebudou se vypisovat nuly na místě hodin, pokud tisk trvá jen řády minut.

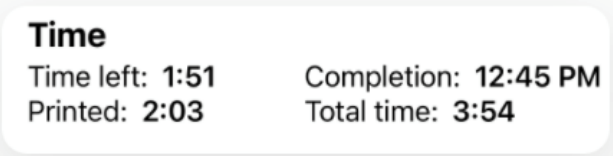
```
var body: some View {
    VStack(alignment: .leading, spacing: 2.0){
        Text(Loca.time.localized)
            .font(.title3)
            .fontWeight(.bold)

        HStack {
            VStack(alignment: .leading) {
                HStack{
                    Text(Loca.timeLeft.localized)
                    (time.timeLeft != 0 ?
                        Text(Date().addingTimeInterval(time.timeLeft), style: .timer) :
                        Text("0:00"))
                        .fontWeight(.semibold)
                }

                HStack {
                    Text(Loca.printedTime.localized)
                    (time.timeLeft != 0 ?
                        Text(Date().addingTimeInterval(-time.printedTime), style: .timer) :
                        Text(timeCorrect(for: time.printedTime)))
                        .fontWeight(.semibold)
                }
            }
```

Ukázka kódu 4: View s informacemi o čase tisku

Zdroj: Autor práce



Obrázek 34: Widget s informacemi o čase tisku

Zdroj: Autor práce

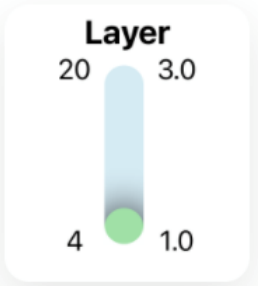
Widget s informacemi o tisknuté vrstvě

Toto View (Ukázka kódu 5) slouží k zobrazení celkového počtu tisknutých vrstev, konečné výšce tisku v milimetrech a také pro zobrazení aktuálního čísla vrstvy a její výšky (Obrázek 35). Veškeré objekty jsou uvnitř GeometryReader. Je to z důvodu, aby šířka sloupcového grafu byla přesně pětina celkové šířky View komponenty. Uspořádání objektů ve View je řešeno pomocí Stack. Pro zobrazení grafického indikátoru výšky tisknuté vrstvy byla napsána vlastní View komponenta ProgressBarLayer. Uváděné milimetry výšky tisku jsou zaokrouhlovány na jedno desetinné místo.

```
GeometryReader { geometry in
            VStack {
                Text(Loca.layer.localized)
                    .font(.title3)
                    .fontWeight(.bold)
                HStack(alignment: .top){
                     Spacer()
                    VStack {
                        Text("\(layer.maxLayer)")
                        Spacer()
                        Text("\(layer.actuallLayer)")
                    }
                    VStack {
                        Spacer()
                        ProgressBarLayer(progress: $layer.progress)
                            .frame(maxWidth: geometry.size.width / 5)
                        Spacer()
                    }
                    VStack {
                        Text("\(layer.maxLayerMm, specifier: "%.1f")")
                        Spacer()
                        Text("\(layer.actuallLayerMm, specifier: "%.1f")")
                    }
                    Spacer()
                }
```

Ukázka kódu 5: View s informacemi o tisknuté vrstvě

Zdroj: Autor práce



Obrázek 35: Widget s informacemi o vrstvě tisku

Zdroj: Autor práce

Komponenta ProgressBarLayer (Ukázka kódu 6) využívá GeometryReader. Je to z důvodu, aby bylo možné vypočíst správnou výšku sloupcového grafu, který reflektuje aktuální hodnotu tisknuté vrstvy. Komponenty, ze kterých se skládá progress bar, byly zobrazovány nad sebou. Jako první byl zobrazen modrý tvar Capsul, který, jak již název napoví, odpovídá tvaru kapsle. Ten objekt byl překryt zeleným tvarem Capsul, jehož výška byla nastavována dle počtu vytištěných vrstev. Tomuto objektu byl také definován stínek. Pro zobrazení pohybu přírůstku vytištěných vrstev byl zvolen typ animace easeInOut.

```
GeometryReader{ geometry in
            ZStack (alignment:.bottom){
                Capsule()
                    .frame(width: geometry.size.width, height: geometry.size.height)
                    .foregroundColor(Color.lightBlue)
                Capsule()
                    .frame(
                        width: geometry.size.width,
                        height: CGFloat(geometry.size.height * CGFloat(progress) + geometry.size.width)
                    )
                    .padding(EdgeInsets.init(top: -geometry.size.width, leading: 0, bottom: 0,
trailing: 0))
                    .foregroundColor(Color.lightGreen)
                    .offset(y: geometry.size.width)
                    .shadow(color: .black, radius: geometry.size.width/2, x: 0, y: 0)
            }
            .clipShape(
                Capsule()
            )
            .animation(.easeInOut)
        }
```

Ukázka kódu 6: Tělo metody ProgressBarLayer

Zdroj: Autor práce

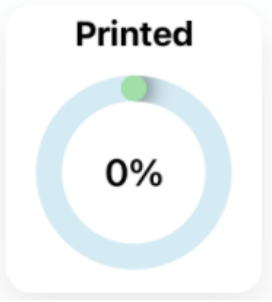
Widget s procenty dokončeného tisku

View slouží pro zobrazení dokončených procent tisku (Obrázek 36). Jak vidíte z ukázky kódu (Ukázka kódu 7), je tvořeno nadpisem View a metodou pro zobrazení koláčového grafu. View komponenta PieChart je velmi podobná předchozí View komponentě ProgressBarLayer. Tato komponenta jako základní tvar využívá tvar Circle. Uprostřed View se zobrazují procenta v textové podobě.

```
var body: some View {
        VStack{
            Text(Loca.done.localized)
                .font(.title3)
                .fontWeight(.bold)
            ZStack {
                VStack {
                    PieChart(progress: $done.progress)
                }
            }
        }
        .padding(EdgeInsets(top: 5, leading: 15, bottom: 15, trailing: 15))
        .background(Color.widgetBackgroundColors)
        .cornerRadius(15)
        .shadow(color: Color.widgetShadowColors, radius: 15, x: 0, y: 10)
    }
```

Ukázka kódu 7: View s procenty dokončeného tisku modelu

Zdroj: Autor práce



Obrázek 36: Widget s procenty dokončeného tisku

Zdroj: Autor práce

Widgety s informacemi o teplotě

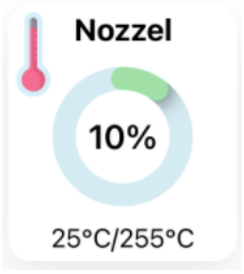
Toto View (Ukázka kódu 8) slouží pro zobrazení informací o aktuální teplotě a cílové teplotě nahřátí (Obrázek 37). V aplikaci je použito 2x, jednou pro teplotu tiskové podložky a podruhé pro teplotu tiskové trysky. Nadpis widgetu se ve View nastavuje přes parametr title. Ikonka teploměru byla stažena z volné knihovny a upravena pro design aplikace. Hlavní komponentou widgetu je koláčový graf, pro který byla opět využita metoda PieChart.

```
var body: some View {
        VStack(alignment: .center) {
            ZStack(alignment: .top) {
                HStack {
                    Image("thermometer")
                    Spacer()
                }
                .padding(.leading, -10.0)
                HStack {
                    Text(title)
                        .font(.title3)
                        .fontWeight(.bold)
                        .multilineTextAlignment(.center)
                }
            }
            .padding(.bottom, -20.0)
            PieChart(progress: $temperature.progress)

            Text("\(temperature.actuallTemperature,specifier: "%.0f°C")/\
(temperature.targetTemperature,specifier: "%.0f")°C")
                .multilineTextAlignment(.center)
                .padding(.top, 10.0)
        }
```

Ukázka kódu 8: View s informacemi o teplotě

Zdroj: Autor práce



Obrázek 37: Widget s informacemi o teplotě trysky

Zdroj: Autor práce

3.4.2 Spojení jednotlivých View do celku

Zmíněná jednotlivá View v kapitole výše byla ve View nesoucí název WidgetsView pospojována dohromady.

WidgetsView

Veškerá View jsou umístěna do ScrollView (Ukázka kódu 9), které zajistí vertikální scrollování obsahu, který se nevejde na obrazovku. Jako první je zobrazován widget s kamerou, ale jen pokud máme data z kamery. Další se vždy zobrazuje widget se stavem tiskárny a s ovládacími prvky. Jakmile jsou nenulová procenta vytištěného modelu, zobrazí se vedle sebe i widget s těmito procenty a widget s informacemi o vrstvě. Avšak widget s informacemi o vrstvě se zobrazuje jen tehdy, pokud máme data k vrstvám tisku. Pokud data nemáme, celý tento prostor v aplikaci vyplní widget s procenty dokončeného tisku. Dále se vždy vedle sebe zobrazí informace o teplotě tiskové podložky a trysky. V ukázce kódu je zobrazena logika jen po View s informacemi o vrstvách.

```
var body: some View {
        GeometryReader { geometry in
            ScrollView {
                VStack {
                    if cameraViewModel.image != nil {
                        CameraCardView(image: $cameraViewModel.image)
                            .padding()
                    }

                    ButtonCardView(
                        printerState: $viewModel.printerStateText,
                        printerStateforButton: $viewModel.printerState,
                        cancelAction: {
                            viewModel.cancelJob()
                        },
                        pauseAction: {
                            viewModel.pauseJob()
                        },
                        resumeAction: {
                            viewModel.resumeJob()
                        }
                    )
                        .padding()

                    if viewModel.done.progress != 0.0 {
                    TimeCardView(time: $viewModel.time)
                        .padding()

                    HStack{
                        DoneCardView(done: $viewModel.done)
                            .frame(
                                width: (viewModel.layer.maxLayer != 0 ?
                                        abs(geometry.size.width/2 - 25) : abs(geometry.size.width -
37.5)),
                                height: geometry.size.width/2 + 5
                            )
                        if viewModel.layer.maxLayer != 0 {
                            Spacer()
                            LayerCardView(layer: $viewModel.layer)
                                .frame(width: abs(geometry.size.width/2 - 25), height:
geometry.size.width/2 + 5)
                            }
                    }
                    .padding()
                    }
```

Ukázka kódu 9: View pro spojení jednotlivých View

Zdroj: Autor práce

Dále bylo potřeba vyřešit stav, kdy o tiskárně nemáme žádné informace a také stav, kdy je potřeba tiskárnu připojit k Octoprintu. Pro tyto účely slouží NotConnectedView.

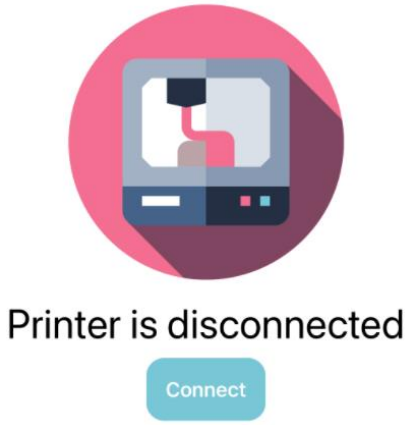
NotConnectedView

Toto View (Ukázka kódu 10) je tvořeno obrázkem, pod kterým je uveden text (Obrázek 38). Jakmile je tiskárna ve stavu, ve kterém jí můžeme připojit k OctoPrintu, zobrazí se pod textem i tlačítko pro připojení.

```
VStack(alignment: .center){
    Image("bigPrinter")
    Text(Loca.noConnection.localized)
        .font(.largeTitle)
    if printerState == .offline {
        Button(action: {
            connectViewModel.connectPrinter()
        }, label: {
            Text(Loca.connectButton.localized)
                .foregroundColor(.white)
                .font(.headline)
                .padding()
                .background(Color.mediumBlue)
                .cornerRadius(15)
        })
    }
}
```

Ukázka kódu 10: View pro spojení jednotlivých View

Zdroj: Autor práce



Obrázek 38: Stránka, pokud není tiskárna připojena

Zdroj: Autor práce

Jelikož se WidgetsView a NotConnectedView budou v aplikaci zastupovat a budou se zobrazovat na stejné stránce, bylo je potřeba spojit do jednoho View. Pro tyto účely slouží v aplikaci DashboardView.

DashboardView

Veškerá View v tomto View (Ukázka kódu 11) byla zabalena do NavigationView, díky kterému lze definovat nadpis stránky, který je typický pro iOS zařízení. Nadpis se zobrazuje v horní části stránky a jakmile se začne v aplikaci scrollovat, nadpis se zmenší a zarovná na střed stránky.

```
NavigationView {
            if viewModel.isEmpty
            {
                AnyView(NotConnectedView(printerState: $viewModel.printerState))
                    .navigationBarTitle(Loca.information.localized)
            } else {
                AnyView(WidgetsView(with: viewModel, cameraViewModel: cameraViewModel))
                    .navigationBarTitle(Loca.information.localized)
            }
        }
```

Ukázka kódu 11: Logika DashboardView

Zdroj: Autor práce

3.4.3 SettingsView

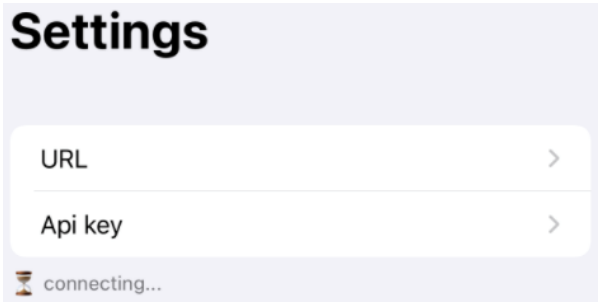
Dále bylo potřeba v aplikaci implementovat obrazovky (Obrázek 39) pro nastavení atributů potřebných pro připojení k OctoPrintu. Pro tyto potřeby bylo vytvořeno SettingsView.

SettingsView (Ukázka kódu 12) je zabaleno do NavigationView a je implementováno jako formulář, který při kliknutí na jeden jeho řádek otevře View pro zadání příslušné hodnoty, které je definováno v atributu destination. Byly nastaveny dvě položky formuláře, a to jedna pro zadání URL adresy a druhá pro zadání API klíče. Jako patička formuláře bylo nastaveno zobrazení stavu připojení k OctoPrintu.

```
Form {
    Section(footer: Text(connectionStatusText)) {
        NavigationLink(
            destination:
          EditURLView(url: $viewModel.url, urlState: $viewModel.urlState),
            label: {
                HStack {
                    Text(Loca.url.localized)
                    Spacer()
                    Text(viewModel.url)
                        .foregroundColor(.gray)
                }
            }
        )
        NavigationLink(
            destination:
          EditApiKeyView(apiKey: $viewModel.apiKey),
            label: {
                HStack {
                    Text(Loca.apiKey.localized)
                    Spacer()
                    Text(viewModel.apiKey)
                        .lineLimit(1)
                        .truncationMode(.middle)
                        .foregroundColor(.gray)
                }
            }
        )
    }
}
```

Ukázka kódu 12: Implementace SettingsView

Zdroj: Autor práce



Obrázek 39: Stránka nastavení

Zdroj: Autor práce

EditApiKeyView

Toto View (Ukázka kódu 13) slouží pro zadání API klíče. Je tvořeno textovým polem pro zadání klíče z klávesnice a tlačítkem pro otevření QR skeneru, prostřednictvím kterého je možné kód naskenovat a do pole se nám vypíše příslušný API klíč, který byl obsažen v QR kódu. Toto řešení bylo implementováno pomocí knihovny třetí strany. Tlačítko pro otevření skeneru je tvořeno systémovým obrázkem QR kódu.

```
VStack {
    Form {
        HStack {
            TextField(Loca.apiKey.localized, text: $apiKey)
            Button(action: {
                isPresentingScanner = true
            }, label: {
                Image(systemName: "qrcode")
            })
            .sheet(isPresented: $isPresentingScanner) {
               self.scannerSheet
            }
        }
    }
```

Ukázka kódu 13: Implementace EditApiKeyView

Zdroj: Autor práce

Obrazovka pro zadání URL adresy byla řešena podobně, s tím rozdílem, že neobsahuje komponentu pro skenování QR kódu.

3.4.4 MainView

MainView je v kódu na pár řádků, ale je to velmi důležité View (Ukázka kódu 14). Spustí se jako první při spuštění aplikace a dá se díky němu pohybovat v aplikaci pomocí tab View. V tab bar jsou nadefinované dvě položky. Jedna pro DashboardView a druhá pro SettingsView. K těmto položkám byly vybrány vhodné ikonky, jedna ze setu systémových ikonek a druhá z volné online knihovny.

```
        TabView {
            DashboardView()
                .tabItem {
                    Image("printer")
                    Text(Loca.information.localized)
                }
            SettingsView()
                .tabItem {
                    Image(systemName: "gear")
                    Text(Loca.settings.localized)
                }
        }
```

Ukázka kódu 14: Implementace MainView

Zdroj: Autor práce

3.4.5 API klient

API klient slouží k zjednodušení získávání dat ze služeb OctoPrintu, které byly popsány výše. Pro přístup k službě je nezbytné se autorizovat pomocí API klíče a dotaz je potřeba směřovat na správnou adresu. Tyto údaje je potřebné použít při každém provolání API. Díky této struktuře nevytváříme v aplikaci duplicitní kód pro tyto účely. Ošetření chyb, vytváření URL dotazů a přidání API klíče do dotazu je řešeno právě v API klientovi (Ukázka kódu 15).

API klient je potřeba inicializovat s API klíčem uživatele a adresou serveru, na kterém běží OctoPrint. API klient obsahuje pomocné struktury ApiKey a Response (Ukázka kódu 16). Struktura ApiKey nám usnadňuje práci s API klíčem při vkládání do http požadavku. Struktura Response drží kromě odpovědi ze serveru i deserializovaný objekt, neboli připravená data pro další použití v aplikaci.

```
private let baseURL: URL
private let apiKey: ApiKey

init(baseURL: URL, apiKey: ApiKey) {
    self.baseURL = baseURL
    self.apiKey = apiKey
}
```

Ukázka kódu 15: Struktura API klienta

Zdroj: Autor práce

```
struct ApiKey {
    let value: String
    let field: String = "X-Api-Key"

    init(value: String) {
        self.value = value
    }
}

struct Response<T> {
    let value: T
    let response: URLResponse
}
```

Ukázka kódu 16: Pomocné struktury uvnitř API klienta

Zdroj: Autor práce

API klient obsahuje tři důležité funkce. Jednu pro vytváření requestu (Ukázka kódu 17), která se stará o sestavení URL adresy, přidání API klíče do hlavičky requestu a nastavení potřebné http metody, jako například metoda POST a GET. Tato funkce s nazývá makeURLRequest. Metoda vrací objekt se složeným requestem.

```
func makeURLRequest(
        path: String,
        queryItems: [URLQueryItem]? = nil,
        httpMethod: Method = .get
    ) -> URLRequest
```

Ukázka kódu 17: Definice funkce makeURLRequest

Zdroj: Autor práce

Druhá a třetí funkce jsou si velmi podobné. Obě slouží k vytváření takzvaných publisherů, což jsou objekty z frameworku Combine. Publisher zajišťuje provolání požadavku na server. Následně se ověří, že požadavek proběhl úspěšně. Ve funkci je navíc kontrolován status kód, který případně vyvolá skrze Publisher chybu. V případě úspěchu funkce zajišťuje deserializaci dat a přesun běhu kódu na hlavní vlákno, jak je patrné z ukázky kódu (Ukázka kódu 18). Poslední funkce slouží pro requesty, ze kterých nepotřebujeme získat deserializovaná data, ale jen informaci, zdali byly úspěšně provedeny.

```
func run<T: Decodable>(_ request: URLRequest) -> AnyPublisher<Response<T>, Error> {
        return URLSession.shared
            .dataTaskPublisher(for: request)
            .tryMap { result -> Response<T> in
                guard let httpResponse = result.response as? HTTPURLResponse else {
                    throw URLError(.badServerResponse)
                }

                switch httpResponse.statusCode {
                case 403:
                    throw URLError(.userAuthenticationRequired)
                case 409:
                    throw URLError(.badServerResponse)
                case 200:
                    break
                default:
                    throw URLError(.badServerResponse)
                }

                let value = try JSONDecoder().decode(T.self, from: result.data)
                return Response(value: value, response: result.response)
            }
            .receive(on: RunLoop.main)
            .eraseToAnyPublisher()
}
```

Ukázka kódu 18: Funkce, které vrací deserializovaná data skrze publisher

Zdroj: Autor práce

3.4.6 Usecase struktury

V aplikaci bylo nadefinováno několik UseCase struktur, které využívají API klienta. Každý UseCase zastává jeden specifický účel. Některá jsou pro získávání dat a další pro provedení požadavku na OctoPrintu. V této části bude popsán jeden UseCase od každého druhu.

UseCase označované Fetch slouží pro získávání dat. V ukázce kódu je uveden UseCase pro získání informací o teplotě. Tento UseCase (Ukázka kódu 19) obsahuje strukturu, ve které je definován objekt reprezentující potřebná data o teplotě. Tento objekt musí implementovat protokol Codable, aby API klient byl schopný převést data ze strojové čitelné podoby do námi dané struktury. Dále je zde funkce fetchTemperature (Ukázka kódu 20), která využívá API klient a předává mu potřebné informace.

```
struct TemperatureResponse: Codable {
    let temperature: Temperature

    struct Temperature: Codable {
        let bed: Bed
        let tool0: Tool

        struct Bed: Codable {
            let actual: Double
            let target: Double
        }

        struct Tool: Codable {
            let actual: Double
            let target: Double
        }
    }
}
```

Ukázka kódu 19: Struktura pro teplotu

Zdroj: Autor práce

```
func fetchTemperature() -> AnyPublisher<TemperatureResponse, Error> {
       let request = apiClient.makeURLRequest(path: "/api/printer", httpMethod: .get)

       return apiClient.run(request)
           .map(\.value)
           .eraseToAnyPublisher()
}
```

Ukázka kódu 20: Funkce fetchTemperature

Zdroj: Autor práce

UseCase pro provedení požadavků v OctoPrintu na začátku svého názvu nesou označení své akce. ConnectUseCase (Ukázka kódu 21) slouží k připojení tiskárny k OctoPrintu. Obsahuje strukturu ConnectRequest, která má v sobě uloženou informaci o akci, kterou chceme vykonat. Tuto strukturu vložíme do API klienta, který provede serializaci dat. Dále je zde funkce (Ukázka kódu 22), která využívá API klienta a nastavuje zde potřebná data pro cílový dotaz.

```
struct ConnectRequest: Codable {
    let command: Command

    enum Command: String, Codable {
        case connect = "connect"
    }
}
```

Ukázka kódu 21: Struktura ConnectRequest

Zdroj: Autor práce

```
func fetchConnect() -> AnyPublisher<Int, Error> {
        var request = apiClient.makeURLRequest(path: "/api/connection", httpMethod: .post)
        request.timeoutInterval = 5
        request.setValue("application/json", forHTTPHeaderField: "Content-Type")

        let jsonData = try! JSONEncoder().encode(ConnectRequest(command: .connect))
        request.httpBody = jsonData

        return apiClient.runNoContent(request)
            .map(\.value)
            .eraseToAnyPublisher()
}
```

Ukázka kódu 22: Funkce fetchConnect

Zdroj: Autor práce

3.4.7 ViewModely

ViewModely v aplikaci slouží k propojení View s UseCase. Vzhledem k velikosti aplikace máme jen několik ViewModelů. CameraViewModel, který zajišťuje pravidelné obnovování snímku z kamery. ConnectViewModel slouží k tomu, aby uživateli umožnil skrze View provolat akci na připojení tiskárny. SettingsViewModel drží informace o stavu připojení k tiskárně a zajišťuje ukládání uživatelských dat. DashboardViewmodel zajišťuje data pro jednotlivá View, kterých je na hlavní obrazovce mnoho. Jsou zde funkce pro formátování času, pro obnovu dat.

Komunikace mezi View a ViewModelem je zajištěna pomocí frameworku Combine. Proměnné označené klíčovým slovem Published, jak můžete vidět v ukázce kódu (Ukázka kódu 23), je možné díky frameworku Combine propojit přímo s konkrétním View. Hlavní vlastností je, že při změně hodnoty ve ViewModelu se změna propíše až na přiřazené View. Na první ukázce (Ukázka kódu 23 a 24) je proměnná ve ViewModelu a na další ukázce je vidět přiřazení na konkrétní View.

```
@Published var bedTemperature = Temperature(
        progress: 0.0,
        targetTemperature: 0,
        actuallTemperature: 0
    )
@Published var nozzelTemperature = Temperature(
        progress: 0.0,
        targetTemperature: 0,
        actuallTemperature: 0
    )
```

Ukázka kódu 23: Proměnné označené Published

Zdroj: Autor práce

```
PieChart(progress: $temperature.progress)
Text("\(
     temperature.actuallTemperature,
     specifier: "%.0f°C"
    )/\(
     temperature.targetTemperature,
     specifier: "%.0f")°C")
```

Ukázka kódu 24: Přiřazení na konkrétní View

Zdroj: Autor práce

3.4.8 Ukládání dat

V aplikaci bylo potřeba uchovat některá data. Pro tento účel byla zvolena jednoduchá databáze UserDefaults. Ta umožnuje ukládat data ve formátu klíč, hodnota. Pro naše účely postačilo ukládání hodnot ve formátu String. V aplikaci ukládáme URL a API klíč. O ukládání hodnot do databáze se stará SettingsViewModel skrze metody uvedené na ukázce kódu (Ukázka kódu 25). K položkám z databáze přistupujeme na několika místech, například při inicializaci API klienta, jak je také uvedeno v ukázce kódu (Ukázka kódu 26).

```
@Published var apiKey: String {
    didSet {
        print("set: \(apiKey)")
        UserDefaults.standard.set(apiKey, forKey: "KEY_API")
        checkVersion()
    }
}
```

Ukázka kódu 25: Metoda pro uložení do UserDefaults

Zdroj: Autor práce

```
guard let urlString = UserDefaults.standard.string(forKey: "KEY_URL") else { return }
        guard let apiKey = UserDefaults.standard.string(forKey: "KEY_API") else { return }

        guard let url = URL(string: urlString) else {
            return
        }

        let apiClient = APIClient(baseURL: url, apiKey: .init(value: apiKey))

        let useCase = FetchLayerUseCase(apiClient: apiClient)
```

Ukázka kódu 26: Získání dat z databáze

Zdroj: Autor práce

3.4.9 Lokalizace

Pro aplikaci byly navrženy dva jazyky, a to čeština a angličtina. Každý text v aplikaci musel být přeložen do příslušného jazyka. Tyto překlady se nacházejí v souborech s názvem Localizable. Jazyk aplikace se automaticky nastaví podle jazyka prostředí telefonu. Pro snadnější přístup k překladům byl použit číselník s odpovídajícím překlady (Ukázka kódu 27).

```
//překlad slova ve složce Localizable
"layer" = "Vrstva";

// enum pro usnadnění načtení z Localizable
enum Loca: String {

   case layer = "layer"
   var localized: String {
        self.rawValue.localized
    }
}

//použiti překladu ve View
Text(Loca.layer.localized)
```

Ukázka kódu 27: Řešení překladů v aplikaci

Zdroj: Autor práce

3.4.10 Barvy, dark mode

V souboru ColorKit jsou definované použité barvy v aplikaci. Pro barvy použité na pozadí aplikace a widgetů bylo potřeba definovat i barvu pro dark mode. Z ukázky

kódu můžete vidět (Ukázka kódu 28), že pokud atribut isDart nabývá hodnoty ne, použije se standardní bílá barva. Pokud má hodnotu ano, použije se definovaná černá barva.

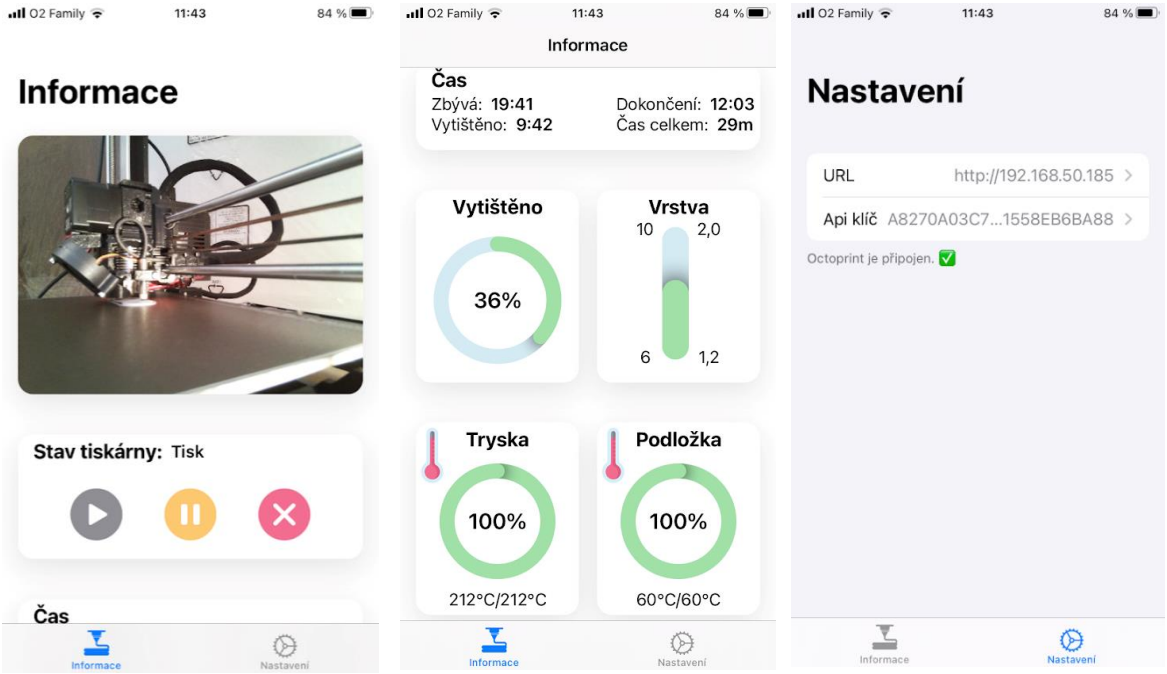
```
static var widgetBackgroundColors: Color {
        isDark ? Color.init(red: 28/255, green: 28/255, blue: 30/255) : .white
}
```

Ukázka kódu 28: Definice barvy

Zdroj: Autor práce

3.5 Nainstalovaná aplikace na fyzických zařízení

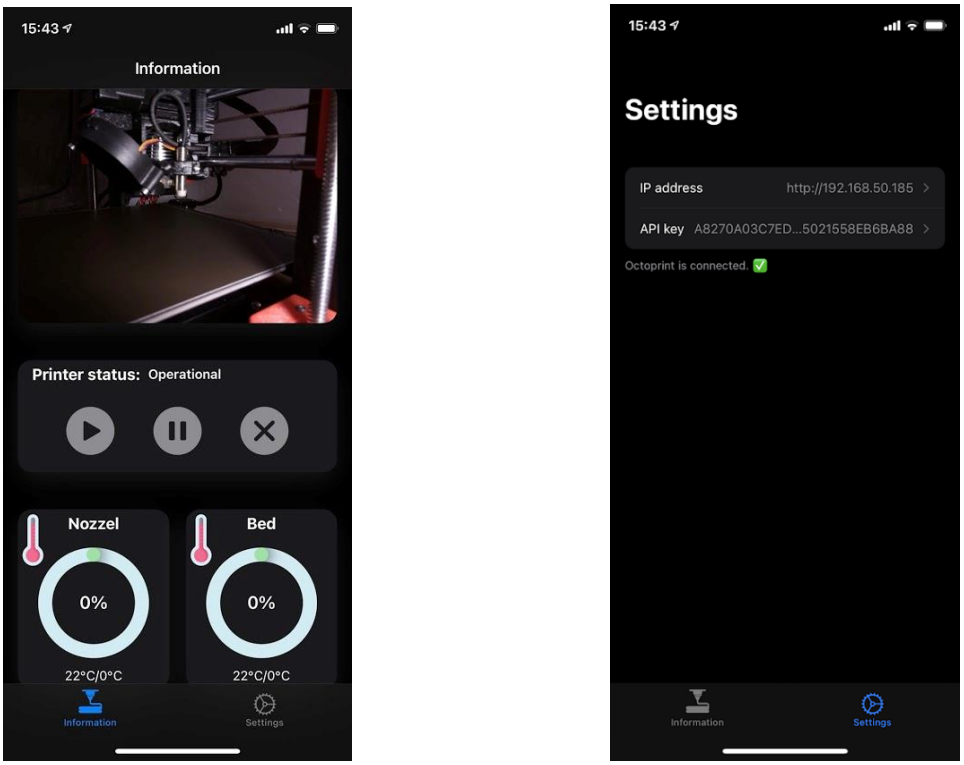
Na ukázkových obrázcích (Obrázek 40) je aplikace nainstalována na zařízení iPhone 7 se systémovým jazykem nastaveným na češtinu. Aplikace je zde ve stavu, kdy probíhá tisk na tiskárně.



Obrázek 40: Aplikace na zařízení iPhone7

Zdroj: Autor práce

Druhé zařízení, kde je aplikace nainstalována, je iPhone XS se systémovým jazykem angličtinou. Aplikace (Obrázek 41) je zde ve stavu, kdy je připojena, ale neprobíhá na ní žádný tisk. Proto se zobrazuje jen část widgetů. Na telefonu je zapnut dark mód, proto se aplikace zobrazuje v černých barvách.



Obrázek 41: Aplikace na zařízení iPhone XS

Zdroj: Autor práce

3.6 Distribuce aplikace

Pro aplikaci byla zvolena distribuce Ad Hoc. Z App Store distribuce bylo odstoupeno z důvodu, že pro schválení aplikace do obchodu by Apple potřeboval veřejný přístup k OctoPrintu, aby si ji mohl vyzkoušet, a ten nebylo možné poskytnout.

Za účelem distribuce aplikace byl vytvořen distribuční certifikát, který umožní distribuovat aplikaci metodu Ad Hoc. V nastavení projektu dále bylo zvoleno automatické podepisování, které zajistí automatické vytváření provisioning profilu, je nutného pro nahrávání aplikace do fyzického zařízení. Aby výsledný balíček s příponou ipa byl spustitelný na konkrétním zařízení, bylo potřeba do prostředí Developer portálu přidat UDID čísla těchto zařízení. Výsledná ipa dále byla nahrána do prostředí App Centra a zde byl vytvořen link ke stažení aplikace. V případě, že by bylo potřeba aplikaci testovat na novém zařízení, muselo by být UDID číslo opět přidáno do Developer portálu a musela by být sestavena nová ipa, která bude opět nahrána do App Centra.

3.7 Uživatelské hodnocení aplikace a návrhy na zlepšení

V této sekci budou shrnuty dojmy z aplikace uživatelů, kteří byli vybráni pro odzkoušení aplikace. Vybraní uživatelé museli disponovat FDM 3D tiskárnou s Raspberry Pi a telefonem s operačním systémem iOS. Dle zpětné vazby uživatelů byly navrženy další změny v aplikaci, které budou shrnuty v sekci Návrhy na zlepšení.

3.7.1 Uživatelské hodnocení

Aplikace k testování byla zpřístupněna čtyřem uživatelům. Tři z nich disponovali tiskárnou Prusa i3 mk3S a jeden tiskárnou značky Creality.

První z uživatelů s tiskárnou Prusa hodnotil aplikaci velmi kladně. Líbil se mu svěží design aplikace a podporovaná možnost tmavého režimu. Kladně také hodnotil zobrazení všech důležitých parametrů na hlavní stránce a snadné napojení k Octoprintu. Jako návrh pro zlepšení navrhoval implementaci push notifikací, které by upozornily na dokončení tisku modelu.

Další uživatel testoval aplikaci s tiskárnou od firmy Creality. Dle jeho názoru je aplikace velmi povedená pro rychlý náhled do informací o tisku. Uživateli v aplikaci chyběla možnost pro nastavení teploty trysky a také se mu zdálo zmatečné zadání URL adresy v nastavení aplikace.

Třetí uživatel testoval aplikaci na tiskárně značky Prusa. Uživatel aplikaci hodnotí jako velmi přehlednou a chválí zobrazení pouze dostupných informací. Také jako užitečnou vnímá funkci dark mode. Velmi kladně také hodnotí otočení snímků kamery na kliknutí a zde by ještě doplnil funkcionalitu, aby si aplikace pamatovala toto otočení a uživatel nemusel znovu klikat po každém dalším vypnutí aplikace.

Poslední uživatel testoval aplikaci také na tiskárně Prusa. Velmi pozitivně hodnotí, to že aplikace umožní rychlý náhled na stav tisku, jelikož má tiskárnu umístěnou ve sklepě a její kontrola není tak pohodlná. Dále také kladně hodnotí líbivý design aplikace.

3.7.2 Návrhy na zlepšení

Z uživatelského hodnocení vyplynulo hned několik postřehů, které by bylo dobré do dalších verzí aplikace zapracovat, aby se aplikace stala ještě více uživatelsky přívětivá.

Jako velmi užitečné byly zhodnoceny chybějící notifikace v aplikaci, které by uživatele upozornily na dokončení tisku 3D modelu. Zde je třeba rozhodnout, zdali

notifikace budou řešeny jen uvnitř aplikace, anebo by byl implementován systém třetí strany, který by se staral o zasílání notifikací z OctoPrintu na koncové zařízení uživatele.

Dále by bylo dobré upravit zadání URL adresy, které mohlo uživatele zmást. Zde by změny nebyly tak náročné. Pole by se přejmenovalo na název IP adresa a uživatel by mohl zadat jak adresu s hlavičkou http, tak i bez ní. Aplikace by si sama kontrolovala, zdali adresa http obsahuje a pokud ne sama by jí doplnila.

Navrhovanou úpravu s implementací ovládání teploty trysky by bylo ještě potřeba promyslet, jelikož tato úprava je spíše pro tiskárny značky Creality. Prusa tiskárny tuto úpravu moc nevyužijí. Zde by bylo potřeba nejdříve zjistit, jaký druh tiskáren uživatelé s touto aplikací používají nejčastěji, a pak až se pouštět do úpravy.

Jako poslední věc by bylo velmi vhodné prověřit více možnosti publikace aplikace na App Store. Zde přichází v úvahu zařídit Applu vzdálený přístup k tiskárně anebo vytvořit mockovací data, na kterých by si aplikaci mohl odzkoušet. U obou možností je velmi pravděpodobné, že i takové přístupy jim nebudou stačit a aplikaci by při testování zamítli.

4 Závěr

Cílem diplomové práce bylo vytvořit aplikaci na kontrolu průběhu tisku modelu na 3D tiskárně za pomoci jednodeskového počítače. Účelem této aplikace bylo uživatelům zpohodlnit kontrolu průběhu tisku a poskytnout aktuální informace o tisku modelu.

V teoretické části diplomové práce byla stručně popsána FDM technologie tisku a nejprodávanější tiskárna trhu, z důvodu, aby čtenář získal větší povědomí o 3D tisku. Dále je zde popsán proces tvorby návrhu mobilní aplikace a využitá komunikační rozhraní pro komunikaci s 3D tiskárnou. Následující kapitoly byly obsahem přizpůsobeny pro vývoj aplikace na platformu iOS. V kapitole architektura mobilní aplikace byla teoreticky popsána struktura aplikace a rozsah funkcionalit jednotlivých vrstev architektury. Kapitola o vývoji aplikace se věnovala jednotlivým nástrojům pro vývoj aplikací pro operační systém iOS a programovacím jazykům pro tuto platformu. V poslední kapitole jsou popsány jednotlivé možnosti distribuce aplikace.

Předmětem praktické části diplomové práce byla aplikace pro kontrolu tisku na FDM tiskárně, která byla určena pro zařízení s operačním systémem iOS. Aplikace byla navržena tak, aby uživateli poskytovala základní informace o tisku, jako je například teplota tisku, čas dokončení tisku, záběry z kamery a informace o vrstvě tisku. Také do aplikace byla zařazena možnost přerušení a zrušení tisku. Praktická část se nejdříve zabývala procesem návrhu mobilní aplikace, kde byl vytvořen funkční prototyp v nástroji Adobe XD. Na základě navržených funkcionalit v prototypu byly zmapovány možnosti komunikačního rozhraní a byly vybrány příslušné komunikační metody služby OctoPrint, které poskytly potřebná data pro mobilní aplikaci. Dále zde byla popsána implementovaná architektura aplikace a také celý proces vývoje aplikace v jazyku Swift, kde jsou uvedeny ukázky kódu se základní logikou v aplikaci. V dalších kapitolách jsou uvedeny finální obrázky z aplikace na koncových zařízeních, popsán způsob distribuce aplikace a také je zde uvedeno uživatelské hodnocení s návrhy na zlepšení aplikace. Jako další úpravy aplikace do budoucna se nabízí implementovat notifikace upozorňující na dokončení tisku a upravit způsob zadáváni URL adresy v nastavení aplikace.

Summary and keywords

The aim of this diploma work is to develop a mobile application for controlling printing on a 3D printer. The application should make the printing process more comfortable for users.

The theoretical part of the diploma work presents a brief description of the FDM printing technology, process of designing a prototype of an application, communication interfaces, mobile application architecture, editors for an application development and distribution of applications.

The application was developed for the iOS platform in a native programming language Swift. The communication between a 3D printer and the application is held by OctoPrint's API. The basic features of the application are webcam streaming, temperature control, printing time control and an option to pause or cancel the printing of a model. At the end of this paper is presented user feedback and some suggestions for improvements.

Keywords: printer, 3D printing, application, iOS, Swift, SwiftUI, OctoPrint

Seznam literatury

5 Git workflows and branching strategy you can use to improve your development process. (2020). In Zepel. https://zepel.io/blog/5-git-workflows-to-improve-development/

3D tiskárna Original Prusa i3 MK3S+. (2021). Prusa 3D. Retrieved April 02, 2021, from https://www.prusa3d.cz/original-prusa-i3-mk3/

About: Branching and Merging. (2021). Git. Retrieved February 20, 2021, from https://git-scm.com/about

About Swift. (2021). Swift.org. Retrieved February 19, 2021, from https://swift.org/about/#swiftorg-and-open-source

Adobe. (2021). Retrieved February 08, 2021, from https://www.adobe.com/cz/products/xd/pricing/individual.html

App Store Review Guidelines. (2021). Apple Developer. Retrieved February 25, 2021, from https://developer.apple.com/app-store/review/guidelines/

Apple Developer Enterprise Program. (2021). Apple Developer. Retrieved February 25, 2021, from https://developer.apple.com/programs/enterprise/

Aysha, M. (2021). OctoPrint, the application to control your 3D printer. In 3D natives. https://www.3dnatives.com/en/octoprint-260820205/#!

Babich, N. (2017). Prototyping 101: The Difference between Low-Fidelity and High-Fidelity Prototypes and When to Use Each. Adobe Blog. https://blog.adobe.com/en/publish/2017/11/29/prototyping-difference-low-fidelity-high-fidelity-prototypes-use.html#gs.s8jdms

Baratashvili, T. (2017). Adobe XD Review — Pros & Cons. Medium.com. https://medium.com/@temology/adobe-xd-review-pros-cons-aba98c16bc7

Basics: Mobile App Architecture & How to Start Building One in 2020. (2020). Basics: Mobile App Architecture & How to Start Building One in 2020. Retrieved February 18, 2021, from https://www.intellectsoft.net/blog/mobile-app-architecture/

Berezhnoi, R. (2021). Figma: pros and cons. F5 Studio. Retrieved February 10, 2021, from https://f5-studio.com/articles/figma-pros-and-cons/

Brown, Z. (2018). A Git Origin Story. Linuc journal. Retrieved March 20, 2021, from https://www.linuxjournal.com/content/git-origin-story

Čápka, D. (2017). Lekce 3 - Třívrstvá architektura a další vícevrstvé architektury. ITnetwork.cz. Retrieved February 18, 2021, from https://www.itnetwork.cz/navrh/architektury-a-dependency-injection/trivrstva-architektura-a-dalsi-vicevrstve-architektury

Čápka, D. (2013). MVC architektura. ITnetwork.cz. Retrieved February 18, 2021, from https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor

Evaluating Design Patterns for Mobile Development. (2021). Clarion technologies. Retrieved February 18, 2021, from https://www.clariontech.com/blog/evaluating-design-patterns-for-mobile-development

Figma pricing. (2021). Figma. Retrieved February 10, 2021, from https://www.figma.com/pricing/

Freeman, J. (2019). What is an API? Application programming interfaces explained. In InfoWorld. What is an API? Application programming interfaces explained

Ibragimova, E. (2016). High-fidelity prototyping: What, When, Why and How?. Prototyper.io. Retrieved February 05, 2021, from https://blog.prototypr.io/high-fidelity-prototyping-what-when-why-and-how-f5bbde6a7fd4

Introducing App Center: Build, Test, Distribute and Monitor Apps in the Cloud. (2017). Retrieved February 26, 2021, from https://devblogs.microsoft.com/appcenter/introducing-visual-studio-app-center/

Jackson, B., Iftikhar, U., Vialva, T., Lord, B., & Petch, M. (c2020). The Free Beginner's Guide [Online]. In *3D Printing Industry*. Retrieved December 27, 2020, from https://3dprintingindustry.com/3d-printing-basics-free-beginners-guide#04-processe

Jennings, A. (2019). Original Prusa i3 MK3S review. Techradar [Online]. Retrieved January 29, 2021, from https://www.techradar.com/reviews/original-prusa-i3-mk3s

Knaster, S., Malik, W., & Dalrymple, M. (2012). More About Xcode. In Learn Objective-C on the Mac. Apress, Berkeley, CA. https://doi.org/https://doi.org/10.1007/978-1-4302-4189-8_7

Low-fi prototyping: What, Why and How? (2016). Prototyper.io. https://blog.prototypr.io/low-fi-prototyping-what-why-and-how-24f77d9f4995

Marsiglia, M. (2021). IOS Distribution Methods. Atomic Object. Retrieved February 25, 2021, from https://spin.atomicobject.com/2010/12/29/ios-distribution-methods/

Mathur, A. (2020). MVVM in iOS Swift. Flawless iOS. https://medium.com/flawless-app-stories/mvvm-in-ios-swift-aa1448a66fb4¨

MVVM pros and cons. (2021). Oreilly. Retrieved February 19, 2021, from https://www.oreilly.com/library/view/learning-javascript-design/9781449334840/ch10s07.html

Octoprint Documentation. (2021). Retrieved February 18, 2021, from https://docs.octoprint.org/en/master/api/version.html

Peterka, J. (1992). Interface. EArchiv.cz. Retrieved April 02, 2021, from https://www.earchiv.cz/a92/a212c120.php3

Pierzchała, B. (2018). Low Fidelity vs High Fidelity Prototypes. 7ninjas, 2018. https://medium.com/7ninjas/low-fidelity-vs-high-fidelity-prototypes-903a7befaa5a

Postman Tutorial: How to use Postman Tool for API Testing. (2021). Guru99. Retrieved February 20, 2021, from https://www.guru99.com/postman-tutorial.html

Pranav, G. (2021). 10 Reasons to Use OctoPrint. In All3DP. https://all3dp.com/2/reasons-to-use-octoprint/

Raspberry Pi. (2021). In Wikipedia: the free encyclopedia. Wikimedia Foundation. https://cs.wikipedia.org/wiki/Raspberry_Pi

Raspberry Pi 3 Model B 64-bit 1GB RAM. (2021). RPishop.cz. Retrieved February 11, 2021, from https://rpishop.cz/raspberry-pi-3b/283-raspberry-pi-3-model-b-64-bit-5060214370028.html

Sarir, J. (2020). How to Use Postman to Manage and Execute Your APIs. Blazemeter. Retrieved February 20, 2021, from https://www.blazemeter.com/blog/how-use-postman-manage-and-execute-your-apis

Sridhar, A. (2018). An introduction to Git: what it is, and how to use it. FreeCodeCamp. Retrieved February 20, 2021, from https://www.freecodecamp.org/news/what-is-git-and-how-to-use-it-c341b049ae61/

SwiftUI: Better apps. Less code. (2021). Developer. Retrieved February 19, 2021, from https://developer.apple.com/xcode/swiftui/

Škoula, M. (2021). Jak s pomocí OctoPrintu ovládat 3D tiskárnu. In Michal Škoula. https://www.skoula.cz/jak-s-pomoci-octoprintu-ovladat-3d-tiskarnu/

Takayama, L., & Landay, J. (2002). High-Fidelity or Low-Fidelity, Paper or Computer Choosing Attributes When Testing Web Prototypes. Human Factors and Ergonomics Society Annual Meeting Proceedings. https://doi.org/10.1177/154193120204600513

Testing Apps with TestFlight. (2021). Retrieved February 26, 2021, from https://testflight.apple.com/

The Good and the Bad of Swift Programming Language. (2021). Altexsoft. Retrieved February 19, 2021, from https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-swift-programming-language/

Varotsis, A. B. (c2020). Introduction to FDM 3D printing [Online]. Retrieved December 27, 2020, from https://www.3dhubs.com/knowledge-base/introduction-fdm-3d-printing

What is a Raspberry Pi? (2021). Opensource.com. https://opensource.com/resources/raspberry-pi

What is a REST API?. (2021). RedHat. Retrieved February 18, 2021, from https://www.redhat.com/en/topics/api/what-is-a-rest-api

Why MVC Architecture? (2017). In Medium.com. Socratic Solution. https://medium.com/@socraticsol/why-mvc-architecture-e833e28e0c76

Zpřístupnili jsme centrum aplikací pro všechny. (2021). Visual Studio. Retrieved February 26, 2021, from https://visualstudio.microsoft.com/cs/app-center/pricing/

Seznam tabulek, obrázků a zkratek

Seznam tabulek

Seznam obrázků

Seznám ukázek kódu

Seznam zkratek

3D – trojrozměrný / trojdimenzionální

API – Application Programming Interface

FDM – Fused Deposition Modeling

IP – Internet Protocol

JSON – JavaScript Object Notation

MVVM – Model, View, ViewModel

MVC – Model, View, Controller

OR – Quick Response

REST – Representational State Transfer

UDID – Unique Device Identifier

UI – design uživatelského rozhraní

URL – Uniform Resource Locator

USB – Universal Serial Bus

UX – uživatelská zkušenost

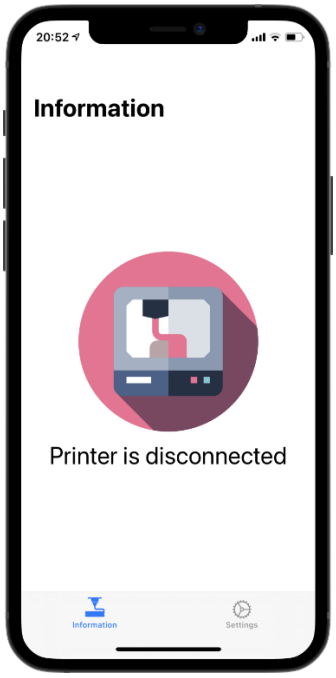
Wi-Fi – Wirelles Fidelity

Seznam příloh

Přílohy

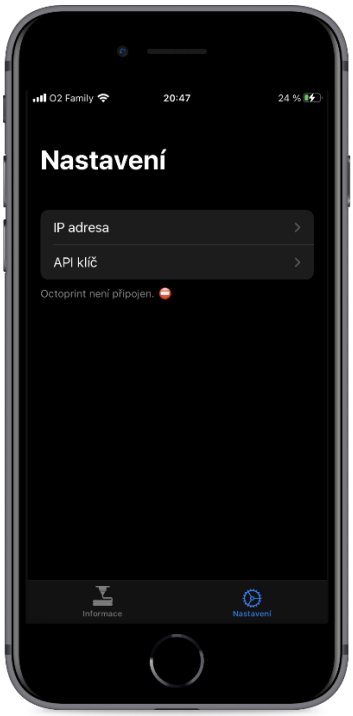
Příloha 1: Tiskárna není připojena, tmavý režim

Zdroj: Autor práce



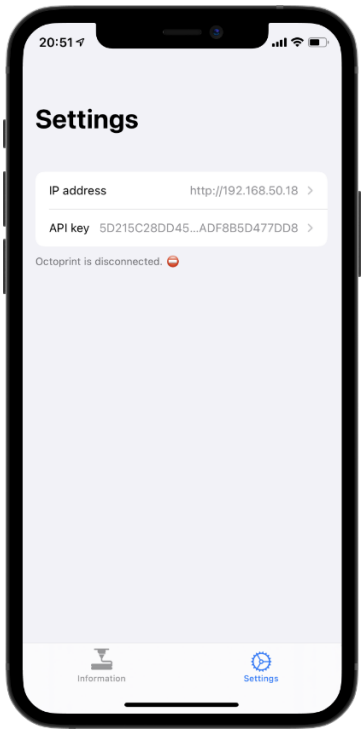
Příloha 2: Tiskárna není připojena

Zdroj: Autor práce



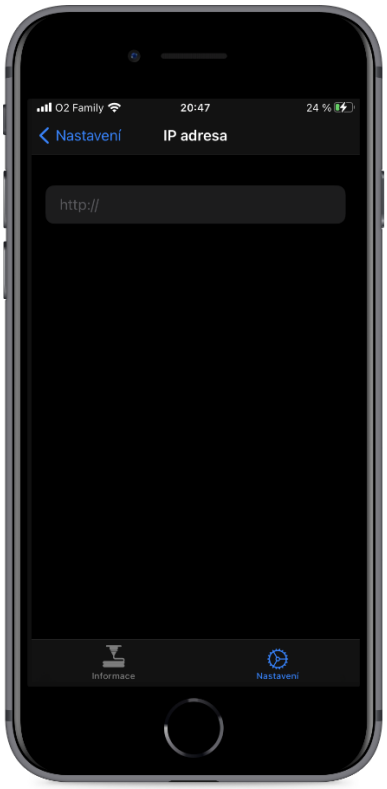
Příloha 3: Sekce nastavení, tmavý režim

Zdroj: Autor práce



Příloha 4: Sekce nastavení

Zdroj: Autor práce

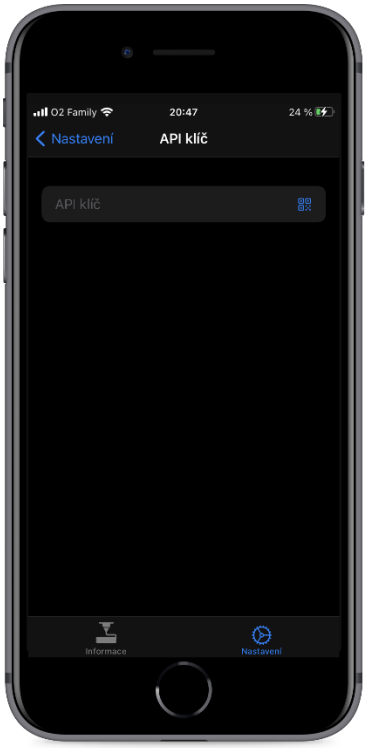


Příloha 5: Zadání URL adresy, tmavý režim

Zdroj: Autor práce

Příloha 6: Zadání URL adresy

Zdroj: Autor práce

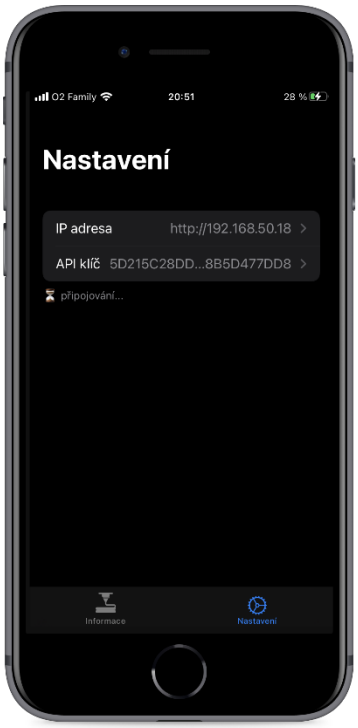


Příloha 7: Zadání API klíče, tmavý režim

Zdroj: Autor práce

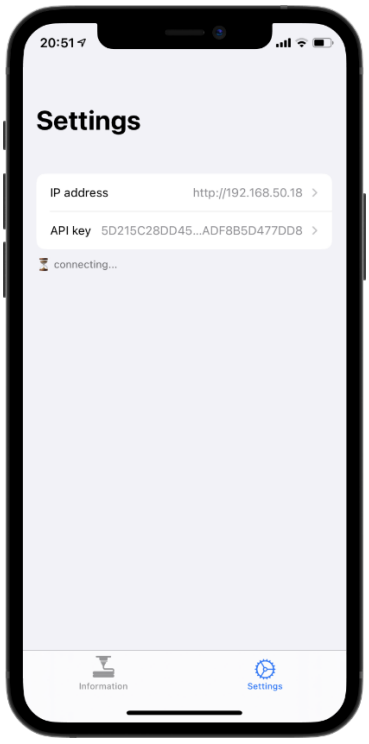
Příloha 8: Zadání API klíče

Zdroj: Autor práce



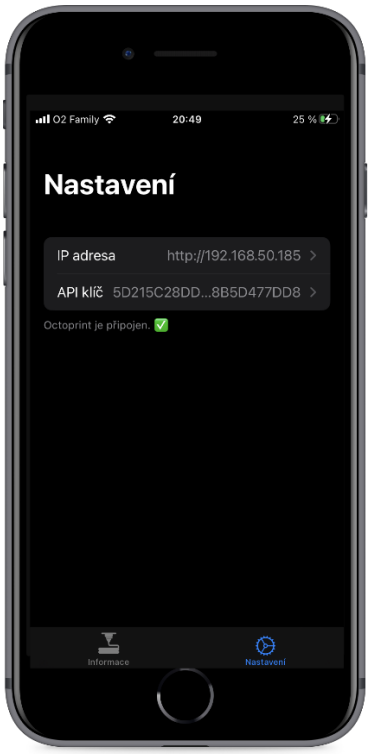
Příloha 9: Sekce nastavení ve stavu připojování, tmavý režim

Zdroj: Autor práce



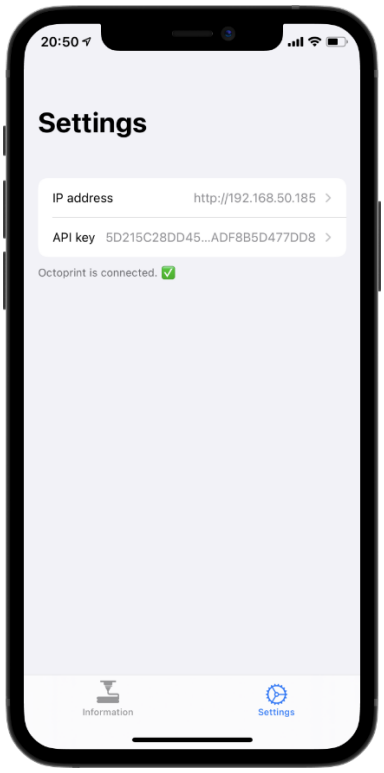
Příloha 10: Sekce nastavení ve stavu připojování

Zdroj: Autor práce



Příloha 11: Sekce nastavení ve stavu připojeno, tmavý režim

Zdroj: Autor práce

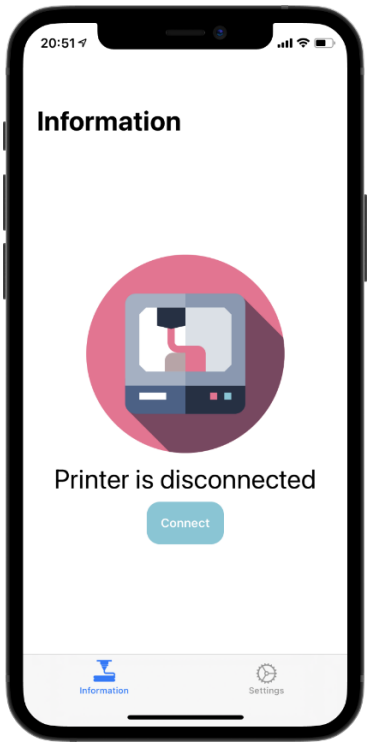


Příloha 12: Sekce nastavení ve stavu připojeno

Zdroj: Autor práce

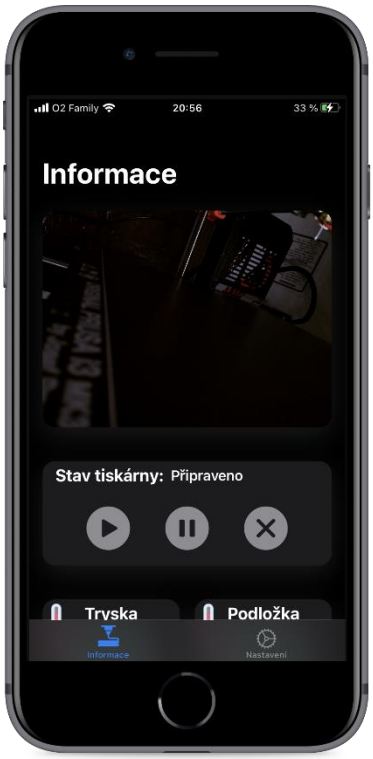
Příloha 13: Připojení tiskárny, tmavý režim

Zdroj: Autor práce



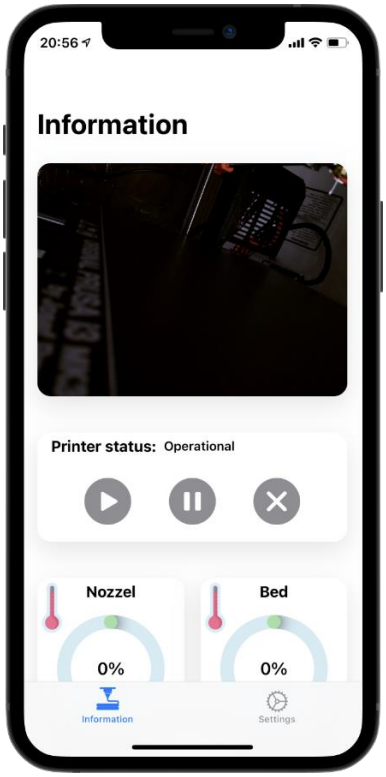
Příloha 14: Připojení tiskárny

Zdroj: Autor práce



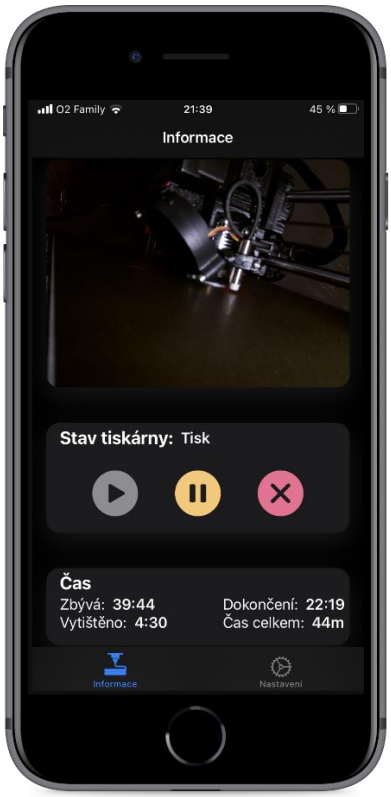
Příloha 15: Informace o tiskárně ve stavu připraveno, tmavý režim

Zdroj: Autor práce



Příloha 16: Informace o tiskárně ve stavu připraveno

Zdroj: Autor práce

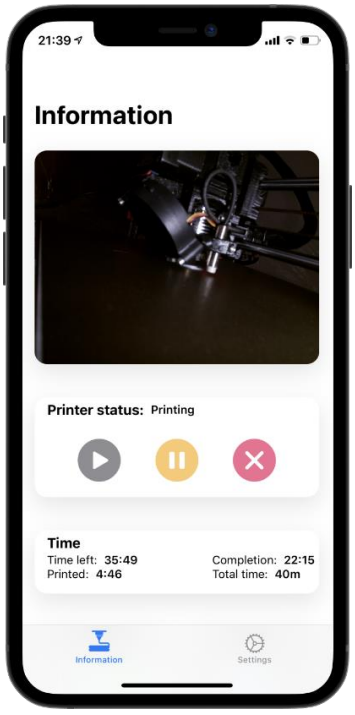


Příloha 17: Informace o tiskárně ve stavu tiskne, tmavý režim

Zdroj: Autor práce

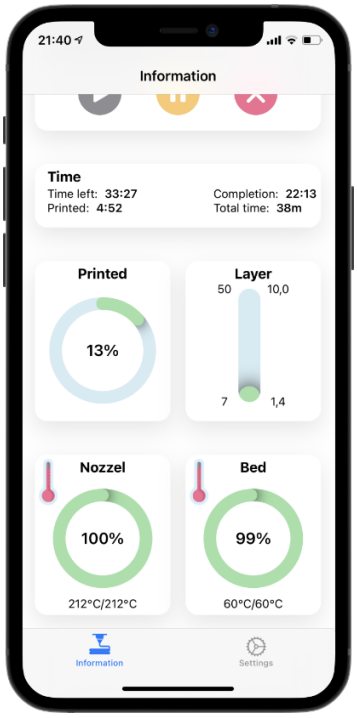
Příloha 18: Informace o tiskárně ve stavu tiskne, tmavý režim

Zdroj: Autor práce



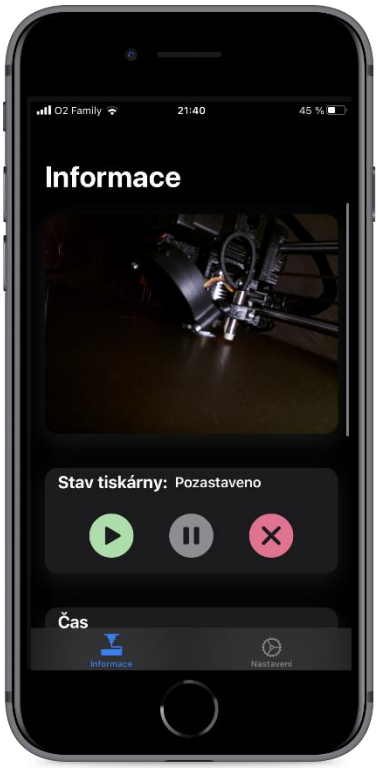
Příloha 19: Informace o tiskárně ve stavu tiskne

Zdroj: Autor práce



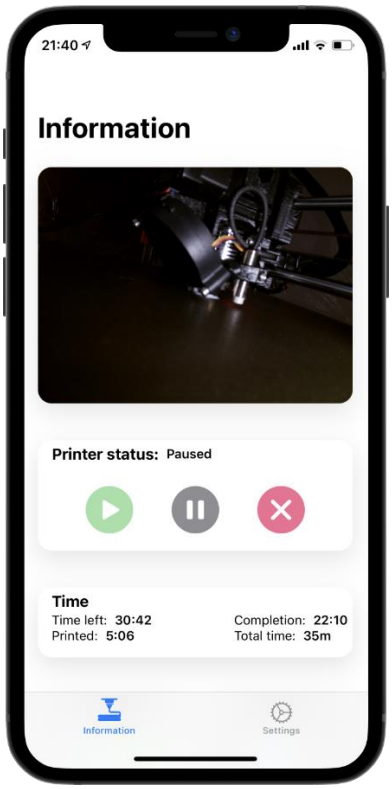
Příloha 20: Informace o tiskárně ve stavu tiskne

Zdroj: Autor práce



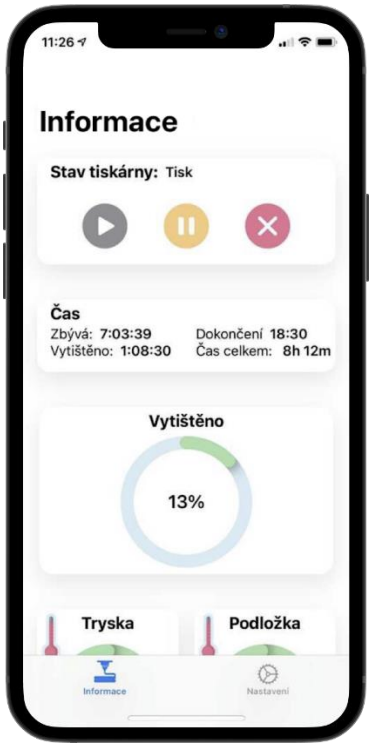
Příloha 21: Informace o tiskárně ve stavu pozastaveno, tmavý režim

Zdroj: Autor práce



Příloha 22: Informace o tiskárně ve stavu pozastaveno

Zdroj: Autor práce



Příloha 23: Informace o tiskárně bez informací o vrstvách tisku

Zdroj: Autor práce