ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA DOPRAVNÍ

Bc. Jan Zarcký

Nástroj pro vytváření dat určených pro automatické ovládání infotainmentu

Diplomová práce

2023

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta dopravní
děkan
Konviktská 20, 110 00 Praha 1

K620..... Ústav dopravní telematiky

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení studenta (včetně titulů):

Bc. Jan Zarcký

Studijní program (obor/specializace) studenta:

navazující magisterský – IS – Inteligentní dopravní systémy

Název tématu (česky): **Nástroj pro vytváření dat určených pro automatické ovládání infotainmentu**

Název tématu (anglicky): Tool for creating data used for automatic control of infotainment

Zásady pro vypracování

Při zpracování diplomové práce se řiďte následujícími pokyny:

- Popište cílové aplikační prostředí vytvářeného nástroje
- Popište vstupní a výstupní data
- Analyzujte významné nástroje pro tvorbu GUI aplikace a na základě této analýzy zvolte vhodný nástroj
- Proveďte analýzu a navrhněte nástroj pro efektivní čtení a vytváření dat
- Popište a vyhodnoťte přínosy navrženého nástroj

ČVUT FD

Rozsah grafických prací:	dle požadavků vedoucích diplomové práce
Rozsah průvodní zprávy:	minimálně 55 stran textu (včetně obrázků, grafů a tabulek, které jsou součástí průvodní zprávy)
Seznam odborné literatury:	Návody k SW dostupné na: https://doc.qt.io/qtforpython/ https://kivy.org/doc/stable/ https://www.python.org/doc/

Vedoucí diplomové práce: **Ing. Martin Preisler, Ph.D.**
Ing. Jiří Růžička, Ph.D.

Datum zadání diplomové práce: **1. června 2022**
(datum prvního zadání této práce, které musí být nejpozději 10 měsíců před datem prvního předpokládaného odevzdání této práce vyplývajícího ze standardní doby studia)

Datum odevzdání diplomové práce: **15. května 2023**
a) datum prvního předpokládaného odevzdání práce vyplývající ze standardní doby studia a z doporučeného časového plánu studia
b) v případě odkladu odevzdání práce následující datum odevzdání práce vyplývající z doporučeného časového plánu studia

L. S.

Ing. Zuzana Bělinová, Ph.D.
vedoucí
Ústavu dopravní telematiky

prof. Ing. Ondřej Přibyl, Ph.D.
děkan fakulty

Potvrzuji převzetí zadání diplomové práce.

Bc. Jan Zarcký
jméno a podpis studenta

V Praze dne 1. června 2022

Poděkování

Rád bych poděkoval svému vedoucímu bakalářské práce Ing. Martinu Preislerovi, Ph.D. za odborné vedení a za trpělivost při konzultování práce. Dále bych rád poděkoval mé rodině za podporu v době studia.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů.

V Praze dne…… ……..……

Podpis autora

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta dopravní

Nástroj pro vytváření dat určených pro automatické ovládání infotainmentu

Diplomová práce

Květen 2023

Bc. Jan Zarcký

ABSTRAKT

Automatizace testování je moderní disciplína při vývoji nových technologií. V této diplomové práci je navržen nástroj, který usnadní práci vývojářům automatizovaných testů. Jsou popsány způsoby testování s ohledem na aplikační prostředí vyvíjeného nástroje a způsoby jeho využití.

KLÍČOVÁ SLOVA

Aplikace, testování, software, obrazovka, automatizace, data, infotainment

CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Transportation Sciences

Tool for creating data used for automatic control of infotainment

Master's Thesis

May 2023

Bc. Jan Zarcký

ABSTRACT

Test automation is a modern discipline in the development of new technologies. This thesis is focused on a tool, that will facilitate the work of automated test developers. Testing methods are described with respect to the application environment of the developed tool and how it can be used.

KEY WORDS

Application, testing, software, screen, automatization, data, infotainment

Obsah

Seznam použitých zkratek

GUI – Graphical user interface (Grafické uživatelské rozhraní)

ECU – Electronic control unit (Elektronická řídící jednotka)

CI/CD – Continuous integration and continuous delivery

HIL – Hardware in the loop (Hardware ve smyčce)

ADAS – Advanced driver-assistance system (Pokročilé asistenční systémy řidiče)

IDE – Integrated development environment (Vývojové prostředí)

1 Úvod

Cílem této diplomové práce je navrhnout nástroj pro vytváření dat určených k automatickému ovládání infotainmentu. Navrhovaný nástroj má sloužit jako pomocník vývojářům automatického testování funkcí infotainmentu ve firmě Digiteq Automotive s.r.o. Tato firma je členem koncernu Volkswagen a zabývá se testováním a vývojem elektrických a softwarových systémů automobilů. Autor této diplomové práce působí v oddělení testování funkcí jako vývojář automatizovaného testování.

Téma této diplomové práce vzniklo právě v oddělení testování funkcí ve výše zmiňované firmě z důvodu potřeby zjednodušení a zefektivnění práce při vytváření automatizovaných testů. Navzdory tomu, že autor pracuje ve stejném oddělení, ve kterém by měl být navržený nástroj používán, jeho současný projekt s touto tematikou nesouvisí. Z tohoto důvodu bude muset autor získat dodatečné znalosti o této problematice, které budou nezbytné pro úspěšné dokončení této diplomové práce.

Teoretická část této diplomová práce se bude věnovat detailnímu představení cílového aplikačního prostředí (tedy prostředí, ve kterém se bude navrhovaný nástroj využívat), které je specifické pro dané oddělení ve firmě. Součástí práce bude podrobný popis účelu, ke kterému bude navrhovaný nástroj využíván, s cílem získat co nejkomplexnější přehled o požadavcích na tento nástroj. V rámci práce bude provedena stručná analýza a srovnání různých nástrojů pro tvorbu grafického uživatelského rozhraní, aby bylo možné vybrat ten nejvhodnější pro daný projekt.

V praktické části bude proveden podrobný popis postupu návrhu samotného nástroje. Dále bude popsán proces tvorby grafického uživatelského rozhraní pomocí zvoleného nástroje. V neposlední řadě bude vysvětlen postup tvorby zdrojových dat prostřednictvím této aplikace, a nakonec bude navrhovaný nástroj vyhodnocen. Celkově bude tato diplomová práce obsáhlým průvodcem procesem návrhu a tvorby nástroje pro dané cílové aplikační prostředí.

2 Cílové aplikační prostředí

V této kapitole bude vysvětleno, co je to infotainment ve vozidle, jaké jsou způsoby testování obecně, ale také s ohledem na testování v cílovém aplikačním prostředí (tedy v oddělení testování funkcí, které je součástí firmy Digiteq Automotive s.r.o.). Na základě toho bude také popsáno, co jsou to testovací stavy.

Vyvíjená aplikace bude využívána v testovacím prostředí firmy Digiteq Automotive s.r.o. v oddělení testování funkcí (později jen DQRF) jako jeden z pomocných nástrojů pro účely automatizovaného testování funkcí infotainmentu (viz kapitola 2.1). Vývoj automatizovaných testů je směřován pro různé výrobce automobilů, mezi které patří například Škoda Auto, Volkswagen, Audi, Porsche a další. Nástroj tak bude sloužit jako pomocník vývojářům těchto automatizovaných testů.

Na závěr kapitoly bude podrobněji popsáno, jakým způsobem se bude navrhovaný nástroj v oddělení DQRF používat.

2.1 Infotainment

Infotainment (nebo někdy také „in-car-entertainment“) je soubor hardwarových a softwarových částí, který posádce automobilu podává informace o vozidle, ale také umožňuje ovládat různé funkce vozidla nebo poskytuje zábavu v podobě audia či videa. Typickým příkladem může být informace o venkovní teplotě, zobrazování navigačních prvků na displeji nebo dokonce zobrazování dopravních značek na head-up displeji. Poptávka, ale také snaha o zvýšení konkurenceschopnosti mezi jednotlivými výrobci automobilů pohání vývoj nových a lepších funkcionalit těchto systémů.

[1]

Testování infotainmentu je důležitou součástí celého vývoje, a to například z těchto důvodů:

- Bezpečnost: Infotainment systémy ve vozidlech mohou ovlivnit řidičovu pozornost a reakční časy. Testování je klíčové pro zajištění, že tyto systémy nesnižují bezpečnost řidiče a cestujících ve vozidle. Pro testování ovlivnění řidičovi pozornosti se používá tzv. „eye trackingový“ test, pomocí kterého se zkoumá, jak řidič věnuje pozornost silnici a infotainmentu.

- Kvalita: Testování pomáhá zajistit, že infotainment vozidla pracuje spolehlivě a bez chyb. To znamená, že uživatelé budou mít lepší zkušenost a nebudou se setkávat s problémy, které by mohly ovlivnit funkčnost systému.

- Zákaznická spokojenost: Infotainment systémy jsou důležitou součástí moderních vozidel a mohou být klíčovým faktorem pro 7. Testování pomáhá zajistit, že systémy fungují podle očekávání zákazníků a splňují jejich potřeby.

2.2 Způsoby testování

Testování nově vyvíjeného softwaru a hardwaru se zpravidla provádí buď manuálně, nebo automatizovaně. V dalších kapitolách budou tyto způsoby podrobněji popsány.

2.2.1 Automatizované testování

Velice efektivním a v dnešní době velice žádaným způsobem je testování automatizované. Tento způsob firmám šetří spoustu času, lidských zdrojů a celkově zefektivní předmět jejich podnikání. Existuje několik důvodů, proč je automatizované testování lepší než manuální:

- **Efektivita a rychlost** – Automatizované testování může být mnohem efektivnější a rychlejší než manuální testování. Automatické testování lze spustit rychleji a s menší chybou než manuální testování, což může ušetřit čas a zdroje.

- **Vícenásobné testování** – Díky automatizovaným testům je možné snadno opakovat testy a srovnávat jejich výsledky, přičemž pomocí automatizace se zajistí, že při každém opakování se test provede stejně.

- **Zaznamenávání výsledků** – Při návrhu automatizovaných testů se dbá na to, aby se při jejich běhu automaticky zaznamenávaly výsledky. Díky tomu je možné se k jednotlivým výsledkům vracet, nebo tyto výsledky prezentovat vývojářům pro účely oprav chyb.

- **Úspora nákladů** – Automatizované testování může pomoci snížit náklady na testování tím, že snižuje potřebu lidských zdrojů a času potřebného pro manuální testování.

- **Rychlost** – Jedním z hlavních přínosů automatizovaného testování je jeho rychlost. Po vytvoření testovacích scénářů dokáže počítač otestovat mnohem více funkcí a operací během krátké doby, nežli by to dokázal člověk manuálním testováním. Kýžené množství testovacích úkonů lze tedy provést za výrazně kratší dobu s použitím automatizovaného testování než s manuálním testováním.

[2]

Automatizované testování je v mnoha ohledech efektivnější, přesnější a rychlejší než manuální testování, což může vést ke snížení nákladů a zvýšení kvality testování. Není to však jednoduchý proces a vyžaduje mnoho úsilí a odbornosti. Prvním důvodem je složitost testovacích scénářů. Tyto scénáře mohou být velmi složité a pokrývat širokou škálu různých podmínek. Z tohoto důvodu je třeba pečlivě zvážit, jakým způsobem budou testovací scénáře automatizovány a jakým způsobem bude automatizovaný test spouštěn.

Dalším důvodem je technická složitost. Automatizace testování vyžaduje značné technické znalosti, zejména co se týče programování, použití automatizačních nástrojů a porozumění funkcionality testované aplikace. Automatizace testování může být složitá a vyžaduje spoustu práce při vytváření testovacích skriptů, které budou spolehlivé a efektivní.

Dalším důležitým faktorem je udržování testovacích skriptů. Testovací skripty mohou být poměrně složité a mohou se měnit spolu s vývojem testované aplikace. Protože změny aplikace mohou mít vliv na testovací scénáře, musí být testovací skripty pravidelně aktualizovány a udržovány.

Integrace do procesu vývoje je také klíčovým faktorem. Automatizované testování musí být integrováno do procesu vývoje a musí být spouštěno pravidelně a spolehlivě. To vyžaduje spolupráci mezi vývojovým týmem a týmem pro automatizované testování.

V neposlední řadě je důležitým faktorem tvorba a správa testovacích dat. Testovací data musí být pečlivě spravována a musí být snadno přístupná pro automatizované testování. To vyžaduje správné vytvoření testovacích datových sad a jejich správu. Tento proces je často zdlouhavý a časově náročný, a proto je snaha v rámci této diplomové práce vytvořit pomocný software, který usnadní tvorbu a správu těchto dat.

Ačkoli automatizované testování může být pro firmy velice lákavé pro jeho rychlost, efektivitu a úsporu času, vyžaduje také mnoho přípravy a práce, aby bylo účinné a spolehlivé. Nicméně i přes tyto nároky stále hraje manuální testování důležitou roli v testování softwaru, zejména v oblastech jako je bezpečnost, nové funkce nebo uživatelské rozhraní.

[2]

2.2.2 Manuální testování

Méně efektivním způsobem, ale také stále hojně využívaným je testování manuální. Manuální způsob testování se využívá v několika případech, jako například: testování nových funkcionalit vyvíjeného systému, které odhalí chyby nebo nedostatky v použitelnosti, které třeba ještě nejsou

pokryty automatizovanými testy. Dále také testování uživatelského rozhraní: tento typ testování se zaměřuje na ověření, zda je uživatelské rozhraní intuitivní a snadno použitelné.

Manuální testování je však pro firmy často levnější a snadněji dostupné než automatizované testování, zejména pro menší týmy nebo projekty s omezeným rozpočtem. Stále hraje klíčovou roli při zajišťování kvality softwaru a nemůže být zcela nahrazeno automatizovaným testováním. Kombinace obou metod tak vede k nejlepším výsledkům.

[2]

2.2.2.1 Testování v automobilu

Jedním z příkladů využití manuálního testování v Automotive může být například testování funkcí přímo v automobilu. Využívá se například pro zajištění bezpečnosti a minimalizaci rizik spojených s používáním infotainmentových systémů, zejména v případě, kdy se ověřuje, že infotainment neovlivňuje řidičovu pozornost. Infotainmentové systémy mohou obsahovat různé funkce, jako jsou navigace, hlasové ovládání, přehrávače hudby a možnosti hands-free telefonování. Tyto funkce mohou být pro řidiče velmi užitečné, ale zároveň mohou způsobit, že řidič ztratí pozornost na silnici, což může vést k nehodám.

Manuální testování umožňuje testerům zjistit, jaké úkoly a funkce infotainmentu mohou ovlivnit řidičovu pozornost na silnici a jaký vliv na něj mají. Testování může zahrnovat simulování různých situací, jako jsou navigace na cestách s mnoha odbočkami, hlasové ovládání při vysokých rychlostech nebo hands-free telefonování v hlučném prostředí. Testování může také zahrnovat pozorování chování řidiče při používání infotainmentového systému. Testovatelé mohou sledovat, zda řidič ovládá infotainmentový systém pohledem z očí základního umístění, což minimalizuje nutnost odvrátit pohled od silnice. Také mohou sledovat, zda řidič ztrácí pozornost na silnici během ovládání infotainmentového systému.

Krom výše zmíněného se manuálně v automobilu testují také nově navrhované funkcionality, prakticky se zkoumá, zda vše funguje tak, jak má před uvedením dané technologie do prodeje. Tento způsob testování provádějí zpravidla týmy lidí, kteří se nepodílí na vývoji automatizovaných testů.

2.3 Testování softwaru a hardwaru

Testování softwaru a hardwaru jsou dvě provázané kategorie testování v oblasti informačních technologií. Oba druhy testování jsou klíčové pro zajištění kvality a spolehlivosti informačních

technologií. Každý druh testování má své specifické vlastnosti a metody a je důležité je použít při vývoji a ověřování jak softwarových, tak hardwarových produktů.

Je důležité připomenout, že testování by mělo být prováděno průběžně v průběhu celého vývojového cyklu, a nejen na jeho konci (viz kapitola 2.3.2). To umožňuje včasné zjištění a opravu chyb a minimalizuje riziko vzniku závažných problémů a selhání produktu.

2.3.1 Testování softwaru

Testování softwaru je proces v průběhu vývoje softwaru, při kterém se ověřuje správnost, kvalita a výkonnost vyvíjeného softwaru. Testování softwaru slouží k zajištění toho, aby se vyvíjené systémy a funkce produktů chovaly podle očekávání a uživatelé se nesetkávali s chybami. Ušetří organizaci čas a peníze tím, že sníží náklady na vývoj a údržbu softwaru. Testování softwaru vytváří záruky stability při vývoji nových funkcí.

[3]

Doba vývoje nových funkcí se zkrátí stanovením sady tzv. „test casů“, kterým musí nová funkce odpovídat, aby byla považována za dokončenou. Vývojáři tak mají pevně stanovený cíl, na kterém mohou pracovat, což umožňuje přesnější odhady časového harmonogramu a snižuje výskyt nových chyb. Po zavedení těchto „test casů“ se sníží celkové náklady na údržbu. Testy se zpravidla spouští po nově dodané funkce, aby se zajistilo, že se systém stále chová podle očekávání.

[3]

Testování softwaru zahrnuje různé úrovně testů, které se používají k ověření správnosti a kvality softwarového produktu. Každá úroveň zkoumá funkčnost softwaru z jedinečného úhlu pohledu v rámci procesu vývoje. Některé z nejčastěji používaných úrovní testování softwaru jsou:

- **Testování jednotkových funkcí (Unit testing)** - Testování jednotlivých kusů kódu – tedy funkcí či metod, aby se zjistilo, zda fungují správně. Jedná se o základní úroveň testování softwaru. Měřenou jednotkou jsou v tomto případě samostatné funkce nebo metody kódu. Během testování jednotkových funkcí jsou produkční funkce kódu prováděny v testovacím prostředí se simulovanými vstupy. Výstup funkce se pak porovnává s očekávaným výstupem pro daný vstup. Pokud výstup odpovídá očekávanému, test proběhne úspěšně. Pokud ne, jedná se o chybu a je nutno jí opravit.

[3]

- **Testování integračních funkcí (Integration testing)** – Pokud „test case“ pokrývá více než jednu funkci/metodu (jednotku) v kódu softwaru, považuje se za integrační test. Zkoumá se, zda jednotlivé části softwaru fungují správně, když jsou integrovány do jednoho celku. Jinými slovy se testuje výstup při simulovaném vstupu na více funkcí či metod v kódu. Znovu platí, že se porovnává očekávaný výstup s reálným výstupem a v případě rozdílu se jedná o chybu.

[3]

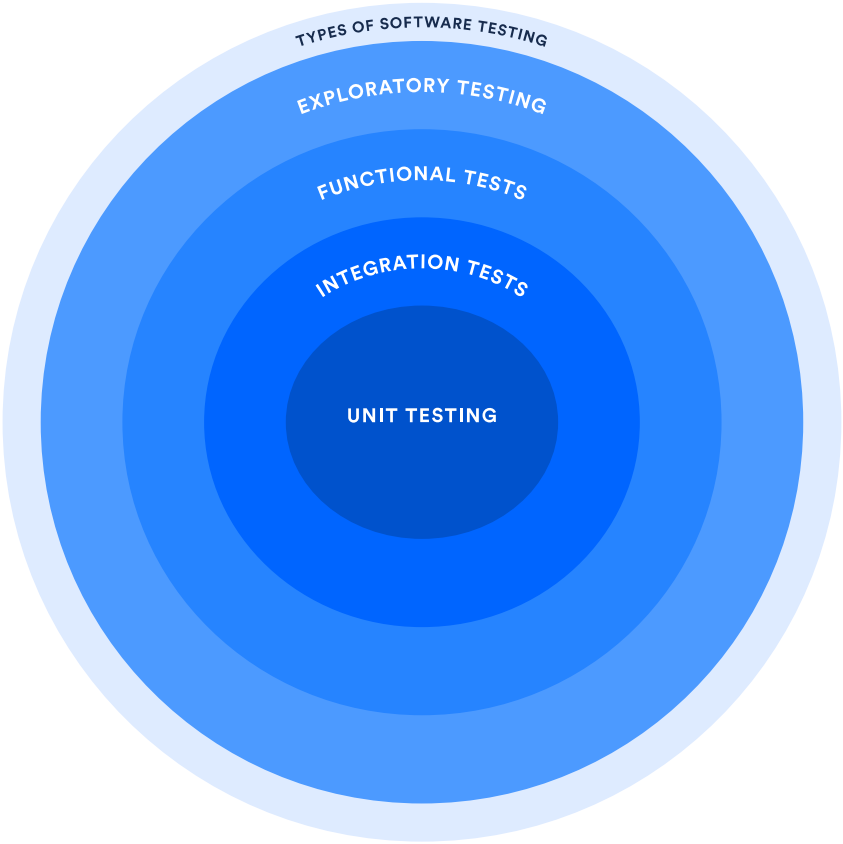
- **Funkční nebo end-to-end testování** – Testování celého softwarového systému, aby se zjistilo, zda splňuje požadavky a specifikace. Tyto testy simulují komplexní uživatelskou zkušenost. End-to-end testy používají nástroje, které simulují skutečné chování uživatele. Díky těmto testům se ověří správná funkčnost softwaru ve všech vrstvách.

[3]

- **Testování učením (Exploratory testing)** - Průzkumné testování je testování, při kterém je testerům zadán volně definovaný úkol, který mají splnit pomocí testovaného softwaru. Díky tomu se může vývojář dozvědět mnoho o způsobu, jakým lidé používají váš produkt v praxi a na základě toho provést změny tak, aby byl produkt uživatelsky přívětivější.

[3]

- **Testování výkonu (Performance testing)** - Testování, zda softwarový produkt pracuje dostatečně rychle a efektivně, a zda je schopen zpracovat požadavky a zátěž v reálném provozu.



*Obrázek 1 - Typy testování softwaru. – zdroj **[3]***

2.3.2 Testování softwaru při kontinuálním dodávání (Continuous testing)

Kontinuální testování je proces provádění automatizovaných testů jako součást procesu dodávání softwaru s cílem získat okamžitou zpětnou vazbu o vyvíjeném softwaru. Původně bylo kontinuální testování navrženo jako způsob, jak zkrátit dobu čekání na zpětnou vazbu pro vývojáře zavedením testů spouštěných vývojovým prostředím a tradičnějších testů spouštěných vývojářem/testerem.

Tento druh testování využívá tzv. „CI/CD[1] pipeline“, což představuje sérii kroků, které jsou nutné provést pro správné dodání nové verze softwaru. Automatizované testy při kontinuálním testování využívají všechny typy testování popsané v kapitolách 2.3.1 a 2.3.2. Optimální nastavení této „CI/CD pipeline“ umožňuje vývojáři odeslat nedávno dokončený kód do této „CI/CD pipeline“ k vyhodnocení. Tento „pipeline“ by nový kód otestoval jednotlivými úrovněmi testování. Pokud kód testováním projde, bude automaticky sloučen do hlavní vývojové větve a nasazen do produkce. Pokud však kód testy neprojde, kód bude odmítnut a vývojář bude automaticky upozorněn na kroky k nápravě.

[3]

2.3.3 Testování hardwaru

Testování hardwaru je důležitý proces, který se provádí při vývoji nových zařízení nebo při údržbě a opravě již existujících. Cílem testování hardwaru je zajištění toho, že zařízení funguje přesně tak, jak by mělo a splňuje požadavky uživatele. Při testování hardwaru se používají podobné praktiky jako při testování softwaru. Na začátku je nutné vytvořit tzv. test plán, představující sadu „test casů“, které ověřují správnost fungování dle předem definovaných požadavků. Testování hardwaru může být prováděno manuálně i automaticky. Mezi nejčastější typy testování hardwaru patří:

- **Funkční testování** – K tomuto testování dochází během výroby ještě předtím, než díl opustí továrnu nebo je srolován do sestavy. Takové testování se označuje jako funkční testování, kdy se s dílem zachází jako s černou skříňkou a testuje se, aby se prokázalo, že díl splňuje své požadavky.
- **Testování spolehlivosti** – Hardware musí prokázat svou odolnost při opakovaném a/nebo dlouhodobém používání. Testování spolehlivosti se používá buď k ověření, že očekávaná životnost výrobku bude splněna, nebo k určení životnosti návrhu, aby bylo možné sladit případné záruční doby a předejít tak nadměrnému vracení výrobků zákazníky. Testování spolehlivosti obvykle vystavuje několik jednotek hardwaru opakovaným činnostem a/nebo cyklickým podmínkám, dokud každá jednotka neselže.

V současné době jsou však napříč vývojáři i testery hojně využívány emulátory. Emulátory jsou softwarové programy, které napodobují chování určitých hardwarových součástí nebo celých zařízení. Tyto emulátory umožňují testovat zařízení v různých prostředích bez nutnosti fyzického připojení k reálnému hardwaru. Díky tomu lze provádět testování rychleji a efektivněji, a přitom zachovat přesnost a spolehlivost.

[4][5]

2.4 Testování v cílovém aplikačním prostředí

V oddělení firmy, pro kterou bude navrhovaný softwarový nástroj vyvíjen, se používá kombinace automatizovaných testů s testy manuálními. V případě manuálního testování se jedná hlavně o testování v automobilu, kdy jednotliví testeři testují nové funkcionality přímo ve vozidle. Tento způsob je však spíše minoritní.

Více využívaným způsobem testování je právě to automatizované. V případě testů jednotkových a integračních, které jsou popsány v kapitole 2.3.1, není potřeba dalších prvků či zařízení pro to, aby se tyto testy mohly spouštět.

Vyvíjená aplikace však bude využívána vývojáři automatizovaných testů, které pokrývají uživatelské testování funkcí. Takové testy pokrývají kombinaci testů softwarových i hardwarových. Aby bylo možné provádět tento druh automatizovaných testů (funkční nebo end-to-end testování), je nutné mít k dispozici sadu hardwarových i softwarových zařízení, která simulují reálné prostředí vozidla, či alespoň jeho částí. Nejčastěji je využívána kombinace obojího. Těmto zařízením (nebo skupinám zařízení), simulujícím prostředí, se říká testovací stavy. Testovací stavy budou blíže popsány v kapitole 2.5.

Další nedílnou součástí jsou samotné automatizované testy ve formě skriptů, které vykonávají předem stanovené úkony. Tyto skripty pak využívají své vlastní softwarové prostředí, jehož funkcionality se následně využívají v jednotlivých krocích automatizovaných testů. Část těchto skriptů bude právě generovat vyvíjená aplikace. Tyto skripty se píšou v programovacím jazyce Python.

2.5 Testovací stavy

Testovací stavy jsou fyzická zařízení, která se používají k ověření správnosti nebo spolehlivosti návrhu nebo modelu. Obvykle jsou umístěny v testovacích laboratořích firem, které mají testování na starosti. Tyto laboratoře mohou být přímo v prostorách firem, které automobily vyrábí, ale i v externích firmách, které si výrobci najímají a které mají testování na starosti.

Testovací stavy jsou obvykle navrženy tak, aby napodobovaly scénáře a prostředí z reálného světa, kde bude testovaný software a hardware nasazen. Umožňují tak vývojářům vyhodnocovat výkon součástek a systémů za různých podmínek. Vývojáři či testeři tak mohou identifikovat potenciální problémy, ověřit změny v návrhu a optimalizovat výkon ještě před uvedením výrobku na trh.

Jednou z hlavních výhod používání testovacích stavů je, že umožňují vývojářům identifikovat potenciální problémy v rané fázi vývojového procesu, ještě před uvedením výrobku na trh. To pomáhá snižovat riziko nákladných chyb a zpoždění a zajišťuje, že výrobky jsou zákazníkům dodávány v nejvyšší možné kvalitě a spolehlivosti.

Další velkou výhodou používání testovacích stavů je, že umožňují vývojářům simulovat širokou škálu scénářů a podmínek, které nemusí být možné nebo praktické zopakovat při testování v

reálném světě. Inženýři tak mohou vyhodnocovat výkonnost součástí a systémů v široké škále podmínek a podle toho optimalizovat jejich výkon.

Celkově jsou testovací stavy základním nástrojem pro vývojáře a testery v široké škále průmyslových odvětví, včetně leteckého, automobilového, elektronického a telekomunikačního průmyslu. Poskytují nákladově efektivní a účinný způsob testování a ověřování elektronických systémů a komponent, který zajišťuje, že výrobky jsou zákazníkům dodávány v nejvyšší možné kvalitě a spolehlivosti.

[6][7]

Testovací stavy (nebo také tzv. „test benche“) používané při testování infotainmentu ve vozidle jsou fyzická zařízení, která simulují reálné prostředí v automobilu, kde bude infotainment používán. Tyto stavy obvykle obsahují počítač, který slouží jako centrální zařízení pro všechny další komponenty, které jsou součástí testovacích stavů. Tyto komponenty obvykle představují řídící jednotky, displeje, reproduktory, mikrofony, ovládací prvky atd. Na centrálním počítači se také spouští automatizované testy.

Během testování mohou být pomocí testovacích stavů prováděny různé typy testů, které byly popsány v kapitolách 2.3.1 a 2.3.3. Například mohou být testovány různé funkce infotainmentu, jako jsou navigace, přehrávání hudby, hands-free volání a další. Dále mohou být testovány různé výstupy, jako jsou zvukové výstupy a obrazové výstupy.[1]

Díky testovacím stavům mohou být identifikovány chyby a nedostatky v infotainmentu dříve, než se dostanou do reálného prostředí, což může vést k výraznému zlepšení bezpečnosti a spokojenosti zákazníků s výsledným produktem.

2.5.1 Hardware in the loop

Hardware-in-the-loop (HIL) je druh testovacího stavu, pomocí kterého se provádí testování složitých systémů reálného času, jako jsou elektronické řídicí jednotky (ECU), pomocí integrace skutečných zařízení do simulační smyčky. Simulace HIL umožňuje ověřování řídicích algoritmů a systémů v reálném prostředí a zajišťuje vysokou úroveň bezpečnosti a robustnosti. Simulace HIL je v automotive hojně využívána a lze ji použít pro různé systémy, jako je elektronický posilovač řízení, odpružení, řízení baterií, ADAS, ale i testování infotainmentu. V Automotive ve své podstatě HIL představuje reálné prvky a jejich propojení v automobilu.

[1] CI/CD – Continuous integration and continuous deployment

[8]

2.6 Funkce navrhované aplikace

Jak již bylo stručně nastíněno v předchozích kapitolách, vyvíjená aplikace bude sloužit jako podpůrný nástroj vývojářům automatizovaných testů. Nástroj bude schopen načítat a zobrazovat screenshot obrazovky infotainmentu, a následně pomocí dalších vstupních dat a znalostí uživatelů této aplikace generovat data výstupní, která jsou potřebná pro automatické ovládání infotainmentu.

Automatické ovládání infotainmentu je realizováno pomocí nástroje, který se orientuje na základě předem definovaných skriptů popisujících zobrazovanou obrazovku infotainmentu. Aby byla možná navigace napříč celým systémem, je nutné pro každou obrazovku infotainmentu vytvořit skript, který popisuje všechny prvky na dané obrazovce. Tento skript se odkazuje na další zdroje informací, jako jsou například souřadnice daných prvků na obrazovce (tzv. masky – maska představuje souřadnice zkoumaného prvku na obrazovce), ikony prvků, či jejich textové popisky. Všechny tyto informace (masky, ikony a texty) jsou ukládány jako samostatné soubory, a skripty popisující obrazovky se na tyto soubory odkazují.

Kvůli častým změnám v uživatelském rozhraní infotainmentu v průběhu vývoje nejsou masky a soubory s ikonami poskytovány vývojáři, tudíž si je musí testeři vytvářet sami. Jedním z výstupů navrhované aplikace budou tedy soubory s ikonami a maskami. Dalším výstupem bude samotný skript popisující obrazovku, přičemž tento skript se bude následně odkazovat na nově vytvořené soubory, nebo na již předem existující pro případ, že již existují.

Nástroj, který pracuje s těmito skripty, a díky kterému je realizováno automatické ovládání infotainmentu, byl již vyvinut firmou Digiteq Automotive s.r.o. Funkcemi tohoto nástroje jsou jednak automatické ovládání, tak také ověření, že se na dané souřadnici obrazovky nachází prvek se správnou ikonou a popiskem tak, jak bylo definováno v požadavcích zadavatele.

3 Nástroje pro tvorbu GUI

3.1 Využití GUI pro účely generování zdrojových dat

Navrhované grafické uživatelské rozhraní bude usnadňovat vývojářům generování zdrojových dat (viz kapitola 2.6). Díky grafickým prvkům a intuitivnímu ovládání pomocí kombinace kreslení a zapisování bude možno jednotlivé prvky na obrazovce definovat velice přehledně a rychle. Ušetří to tak vývojářům čas, usnadní práci a odstraní případné chyby, ke kterým často dochází při ručním psaní skriptů. Další výhodou bude i to, že tvorbu těchto skriptů bude moci díky této aplikaci provádět i člověk, který nerozumí programování.

3.2 Porovnání nástrojů pro tvorbu GUI

Výběr správného nástroje pro tvorbu GUI závisí na řadě faktorů, jako jsou požadavky aplikace, cílová platforma, cílové aplikační prostředí, použité programovací jazyky a další. Vývojáři by měli pečlivě zvážit všechny tyto faktory a vybrat nástroj, který nejlépe vyhovuje jejich potřebám.

Vzhledem k tomu, že je v rámci této diplomové práce tvořena aplikace, která má sloužit vývojářům jako pomocný nástroj při vývoji automatizovaných testů, které se spouští výhradně na počítačích s operačním systémem Windows, bude se v této analýze uvažovat pouze s takovými metodami a technologiemi, které umožňují tvorbu aplikací na platformě Windows.

V neposlední řadě je potřeba zmínit, že existuje obrovské množství nástrojů určených pro vytváření grafického uživatelského rozhraní (GUI) pro desktopové aplikace. Nicméně, v rámci této diplomové práce budou zmíněny pouze některé z nich.

3.2.1 Qt

Qt je multiplatformní framework pro vývoj aplikací pro počítače, vestavěné systémy a mobilní zařízení. Mezi podporované platformy patří Linux, MacOS, Windows, VxWorks, QNX, Android, iOS, BlackBerry, Sailfish OS a další. Qt není samostatný programovací jazyk.

Jedná se o framework napsaný v jazyce C++. K rozšíření jazyka C++ o funkce, jako jsou signály a sloty, se používá preprocesor MOC (Meta-Object Compiler). Signály a sloty slouží k propojení uživatelského rozhraní aplikace s business logikou.

Vývoj Qt zahájili v roce 1990 norští programátoři Eirik Chambe-Eng a Haavard Nord pod společností Trolltech. Dnes se bývalý Trolltech jmenuje The Qt Company a je dceřinou společností finské společnosti Digia Plc. Ačkoli je The Qt Company hlavním tahounem Qt, Qt nyní vyvíjí větší aliance: The Qt Project. Ta se skládá z mnoha společností a jednotlivců z celého světa.

Aplikace používající Qt jsou převážně psané v programovacím jazyce C++, nicméně existují tzv. jazykové vazby na jiné programovací jazyky, mezi které patří například Python, JavaScript, Rust, C# a další. Rozhraní Qt API je implementováno v jazyce C++ a poskytuje další funkce pro snadnější vývoj napříč platformami.

Qt je dodáváno s vlastním integrovaným vývojovým prostředím (IDE) s názvem Qt Creator. Běží na operačních systémech Linux, MacOS a Windows. Součástí Qt je také interaktivní grafický nástroj Qt Designer, který funguje jako generátor kódu pro grafická uživatelská rozhraní založená na widgetech. Qt Designer lze používat samostatně, ale je také integrován do Qt Creatoru.

[9]

Jednou z nevýhod Qt může být jeho komplexita. Qt nabízí velké množství funkcí a nástrojů, které mohou být pro začínající vývojáře složité ovládat. Další nevýhodou může být cena. Zatímco Qt je zdarma pro open source projekty a malé projekty, pro větší projekty je nutné zakoupit licenci, což může některé vývojáře motivovat k tomu, aby se rozhodli využít jiný konkurenční nástroj, který je zdarma.

Vcelku však lze konstatovat, že Qt je výkonný a flexibilní nástroj pro tvorbu uživatelských rozhraní. Jeho vlastnosti, jako jsou cross-platformová podpora, široká paleta widgetů a vysoká úroveň interaktivity umožňují vývojářům rychle a efektivně vytvářet moderní a vzhledově přívětivé aplikace pro různé platformy a zařízení.

3.2.2 wxWidgets

wxWidgets založil v roce 1992 Julian Smart na univerzitě v Edinburghu. Původně vznikl jako projekt pro vytváření aplikací přenositelných mezi systémy Unix a Windows, ale postupně se rozrostl na podporu systémů MacOS a mnoha dalších sad nástrojů a platforem.

Nástroj wxWidgets poskytuje jediné, snadno použitelné rozhraní API pro psaní aplikací grafického uživatelského rozhraní na různých platformách, které využívají ovládací prvky a nástroje nativní platformy.

Přestože je wxWidgets napsán v jazyce C++, je možné ho používat v různých programovacích jazycích jako je například Pythonu, Perlu a C#.

wxWidgets funguje s několika IDE jako je například CLion, Code::Blocks, Microsoft Visual C++, ale také Xcode.

[10]

Podobně jako Qt, wxWidgets nabízí open-source GUI designer s názvem wxFormBuilder, ve kterém lze vytvářet grafické prvky vyvíjené aplikace, ale také spravovat zdrojový kód. Oproti Qt ale nenabízí své vlastní vývojové prostředí.

3.2.3 JavaFX

JavaFX je platforma pro tvorbu moderních aplikací s grafickým uživatelským rozhraním pro desktopové, mobilní a vestavěné systémy v programovacím jazyce Java. JavaFX poskytuje mnoho nástrojů pro tvorbu uživatelského rozhraní, včetně kontejnerů, widgetů, efektů a animací. JavaFX také podporuje CSS pro stylování rozhraní.

„JavaFX je moderní framework pro tvorbu bohatých okenních aplikací. Bohatých je zde myšleno vizuálně. JavaFX přináší podporu obrázků, videa, hudby, grafů, CSS stylů a dalších technologií, které zajistí, že výsledná aplikace je opravdu líbivá. Zároveň je kladen důraz na jednoduchost tvorby. Všechny zmiňované věci má JavaFX v základu. Platformu JavaFX můžeme použít pro desktopové aplikace, webové applety nebo mobilní aplikace.“

[11]

Jednou z nevýhod JavaFX je, že má nízkou popularitu mezi vývojáři. To znamená, že je k dispozici méně dokumentace, tutoriálů a komunitní podpory. Další nevýhodou JavaFX může být jeho omezená podpora pro některé funkce. JavaFX neumožňuje snadné vytváření vlastních vzhledů aplikací, což může být omezující pro vývojáře, kteří potřebují vytvořit unikátní a přizpůsobené uživatelské rozhraní. Další nevýhodou může být nutnost instalace Javy na počítači uživatele. To může způsobit problémy v případech, kdy uživatel nemá nejnovější verzi Javy nebo ji vůbec nemá nainstalovanou.

Celkově lze konstatovat, že nevýhody JavaFX spočívají zejména v nízké popularitě mezi vývojáři a omezené podpoře pro některé funkce.

3.2.4 Electron

Electron je open-source framework pro tvorbu desktopových aplikací pomocí webových technologií, jako je HTML, CSS a JavaScript. Tento framework umožňuje vývojářům vytvářet aplikace s moderním uživatelským rozhraním a s přístupem k nativnímu API operačního systému, což umožňuje přístup k mnoha funkcím operačního systému, jako jsou například notifikace, dialogy a další, aniž by vývojáři museli mít zkušenosti s nativním vývojem. Electron využívá Chromium pro zobrazování webového obsahu.

„Electron je framework pro vytváření desktopových aplikací pomocí JavaScriptu, HTML a CSS. Díky tomu, že do své binární verze zabudoval Chromium a Node.js, umožňuje Electron udržovat jednu kódovou základnu JavaScriptu a vytvářet multiplatformní aplikace, které fungují v systémech Windows, macOS a Linux.“

[12]

Jednou z nevýhod Electronu je jeho náročnost na výkon počítače. Protože využívá technologie webových prohlížečů, může být náročný na procesor a paměť, což může vést k pomalejšímu spouštění a odezvě aplikace. Kromě toho je například oproti Qt a wxWidgets i omezující v tom, že nemá API na jiné programovací jazyky a aplikace musí být vytvářeny pouze v programovacím jazyce JavaScript.

3.2.5 Windows Forms

Windows Forms je technologie pro tvorbu GUI pro desktopové aplikace na platformě Windows. Byl vyvinut společností Microsoft. Windows Forms poskytuje vývojářům vizuální návrhář, pomocí kterého mohou vývojáři jednoduše tvořit grafické rozhraní pouhým umisťováním již předem připravených prvků do vyvíjeného okna aplikace. Tyto prvky si pak mohou vývojáři graficky upravovat dle svých preferencí. Windows Forms je postaven na .NET Frameworku a používá se v programovacích jazycích C# nebo Visual Basic.

„Windows Forms je framework na tvorbu uživatelského rozhraní aplikací na desktopovém prostředí operačního systému Windows. Poskytuje jeden z nejproduktivnějších způsobů vytváření desktopových aplikací na základě vizuálního návrháře poskytovaného ve Visual Studiu. Funkce, jako je umisťování vizuálních ovládacích prvků přetažením, usnadňují vytváření desktopových aplikací.

Pomocí Windows Forms můžete vyvíjet graficky bohaté aplikace, které lze snadno nasadit, aktualizovat a pracovat s nimi v režimu offline nebo při připojení k internetu. Aplikace Windows Forms mohou přistupovat k místnímu hardwaru a souborovému systému počítače, na kterém je aplikace spuštěna.“

[13]

Jednou z nevýhod Windows Forms je omezená přenositelnost mezi různými operačními systémy, protože je navržen pro použití na platformě Windows. To může být problém pro vývojáře, kteří chtějí vytvářet aplikace pro více operačních systémů. Další nevýhodou může být i to, že Windows Forms nemá takovou škálu funkcí jako jiné frameworky pro tvorbu GUI, jako je například výše

zmiňovaný wxWidgets nebo Qt. Tyto frameworky nabízejí větší možnosti pro tvorbu moderního uživatelského rozhraní a podporují pokročilejší funkce, jako jsou například animace, efekty nebo práce s multimédii.

3.2.6 Tkinter

Tkinter je výchozí toolkit pro tvorbu GUI desktopových aplikací v jazyce Python. Je součástí standardní instalace Pythonu pro všechny platformy a je k dispozici zdarma a open-source. Využívá Tk GUI toolkit, který poskytuje různé ovládací prvky a komponenty pro tvorbu GUI aplikací, jako jsou tlačítka, textová pole, seznamy, okna a dialogová okna. Tkinter také umožňuje vytvářet vlastní widgety a přizpůsobovat vzhled a chování existujících widgetů. Je podporovaný na různých operačních systémech včetně Windows, macOS a Linux. Vizuální prvky jsou vykreslovány pomocí nativních prvků operačního systému, takže aplikace vytvořené pomocí nástroje Tkinter vypadají jako na platformě, na které jsou spuštěny.

Tkinter nemá tak široké spektrum funkcí jako jiné výše zmiňované frameworky, což může být v některých případech omezující. Navíc, vzhled Tkinteru není tak moderní a atraktivní jako u konkurenčních toolkitů, neumožňuje například animace nebo práce s multimédii a může být někdy pomalý a málo výkonný v porovnání s jinými frameworky, zejména při práci s velkým množstvím dat nebo při vykreslování grafických prvků.

Oproti Qt nemá vlastní vývojové prostředí ani designer, nicméně existují nástroje třetích stran, které umožňují vytvářet design aplikace. Příkladem takových nástrojů je například Visual TK nebo Tkinter Designer.

[14]

3.2.7 Swing

Swing je knihovna pro tvorbu grafického uživatelského rozhraní pro desktopové aplikace v jazyce Java. Knihovna Swing poskytuje aplikační rozhraní pro tvorbu a obsluhu klasického grafického uživatelského rozhraní. Pomocí Swingu je možno vytvářet okna, dialogy, tlačítka, rámečky, rozbalovací seznamy atd. Je součástí Java Foundation Classes (JFC).

Swing byl navržen s ohledem na platformovou nezávislost a umožňuje vytvářet aplikace, které běží na různých operačních systémech. Využívá technologie Java Virtual Machine (JVM), která umožňuje spouštět Java aplikace na mnoha platformách.

Swing nabízí velké množství komponentů, které jsou k dispozici pro tvorbu GUI. Tyto komponenty jsou rozděleny do několika skupin, včetně základních komponent (např. tlačítka, pole pro vstup

textu), pokročilých komponent (např. tabulky, stromy, seznamy), komponent pro zpracování událostí a další.

Vývoj aplikací v Swing je poměrně snadný díky množství nástrojů, které jsou k dispozici. Například, vývojáři mohou využívat tzv. "GUI builder", což je nástroj pro tvorbu GUI, který umožňuje vytvářet GUI aplikací přetahováním a nakládáním komponentů. Swing má také velkou komunitu a množství dokumentace, což usnadňuje učení a používání tohoto frameworku.

Mezi nevýhody patří jeho komplexnost – Swing má mnoho funkcí a možností, ale vývojáři musí mít hluboké znalosti jazyka Java a architektury aplikace, aby mohli využít jeho plného potenciálu. Dále také i fakt, že aplikace postavené na Swingu nemají vzhled a chování nativní pro daný operační systém, což může ovlivnit uživatelskou zkušenost. V neposlední řadě je potřeba zmínit i fakt, že Swing byl vyvinut v roce 1997 a od té doby nebyl zásadně aktualizován. Proto mnoho vývojářů přechází na modernější technologie pro tvorbu GUI, jako je JavaFX nebo Qt.

Celkově však lze říct, že Swing je velmi výkonný a flexibilní framework pro tvorbu GUI desktopových aplikací v jazyce Java.

[15]

3.3 Výběr nástroje

Nástroj	Vývojové prostředí grafického rozhraní	Programovací jazyky	Zkušenosti týmu	Zkušenosti autora
Qt	Ano	C++, C, Ruby, Python, Java, C# a další…	Ano	Ano
wxWidgets	Ano	C++, Python, Perl, C#, Java a další…	Ne	Ne
JavaFX	Ano	Java, Kotlin, Scala, Groovy	Ne	Ne
Electron	Ne	JavaScript (TypeScript)	Ne	Ne
Windows Forms	Ano	C++, C#, Visual Basic	Ano	Ne
Tkinter	Ano (3. strany)	Python	Ano	Ano
Swing	Ano	Java	Ne	Ne

Tabulka 1 - porovnání nástrojů pro tvorbu GUI

V tabulce výše (Tabulka 1 - porovnání nástrojů pro tvorbu GUI) byly porovnány jednotlivé nástroje na základě parametrů, které jsou důležité pro finální rozhodnutí ohledně toho, jaký nástroj bude autorem k tvorbě navrhované aplikace využit. Parametr vývojového prostředí grafického rozhraní byl vybrán z důvodu snadnějšího vývoje, kdy si vývojář může díky takovému nástroji vytvořit design aplikace vizuálně, velmi jednoduše pomocí tzv. „drag and drop“. Tuto funkcionalitu mají téměř všechny popisované nástroje, tudíž tento parametr není při výběru rozhodující. Dalším velice důležitým parametrem byl zvolen programovací jazyk, pomocí kterého se dají aplikace v daném nástroji vytvářet. U tohoto parametru je nutné zmínit, že autor této diplomové práce má nejhlubší zkušenosti s programovacím jazykem Python, nicméně nějaké zkušenosti má i s jazyky JavaScript a C++. Díky tomuto parametru se již tedy výběr zúžil o nástroj Swing a JavaFX. Dalším parametrem byly zvoleny zkušenosti týmu s daným nástrojem. Slovem tým se v tomto kontextu myslí oddělení, ve kterém se navrhovaná aplikace bude používat. Tento parametr je důležitý především proto, aby v budoucnu mohla být aplikace udržována, ale i vylepšována lidmi z daného oddělení. Po vyhodnocení tohoto parametru se výběr zúžil pouze na nástroje Qt, Windows Forms a Tkinter. S ohledem na poslední parametr, kterým je zkušenost samotného autora aplikace s daným nástrojem vyplývá, že pro vývoj aplikace je nejoptimálnější nástroj Qt a Tkinter. Na základě provedené analýzy v kapitolách 3.2.1 a 3.2.6 však bylo rozhodnuto, že se k vývoji aplikace použije nástroj Qt.

Jak bylo řečeno výše, nástroj Qt nabízí spoustu funkcionalit, které usnadňují a urychlují vývoj desktopových aplikací. Tyto aplikace je možné vyvíjet pomocí programovacího jazyka Python (díky knihovně PyQt – viz kapitola 3.3.2), který se používá v prostředí, ve kterém se navrhovaný nástroj bude používat. Díky tomu se zajistí, že aplikace bude v budoucnu snadněji rozšiřitelná a spravovatelná i dalšími kolegy v oddělení, kde se bude používat.

Vzhledem k tomu, že pro vývoj navrhovaného nástroje byl zvolen framework Qt v podobě PyQt, budou další podkapitoly věnovány programovacímu jazyku Python a knihovně PyQt.

3.3.1 Python

Python je vysokoúrovňový interpretovaný programovací jazyk, který byl vytvořen v roce 1991 Guido van Rossumem. Jedná se o velmi oblíbený jazyk, který se používá k řešení různých úloh, včetně web developmentu, datové analýzy, vývoji desktopových aplikací, automatizace úloh a mnoha dalších. Jeho popularita začala nejvíce stoupat po roce 2015, přičemž v roce 2018 se zařadil mezi nejoblíbenější programovací jazyky.

Jazyk Python se vyznačuje jednoduchou a čitelnou syntaxí, který usnadňuje jeho použití. Kódy napsané v Pythonu jsou srozumitelné i pro začátečníky. Python také nabízí širokou škálu knihoven a frameworků, které usnadňují vývoj a zajišťují efektivní a rychlý vývoj.

Python má dynamickou typovou kontrolu a podporuje více paradigmat programování, včetně procedurálního, objektově orientovaného a funkcionálního programování. Python také poskytuje integrovanou konzoli, která umožňuje snadné testování kódu, což usnadňuje práci hlavně v případě, že si vývojář potřebuje otestovat pouze nějakou část kódu.

Python je multiplatformní jazyk, který běží na různých operačních systémech, včetně Windows, Linux a macOS. Je také volně dostupný a open-source, což znamená, že je k dispozici zdarma a vývojáři mohou upravovat jeho zdrojový kód podle svých potřeb.

Python je vynikající volbou pro vývojáře, kteří chtějí efektivně vytvářet aplikace s čitelným kódem a snadným použitím, ať už pro jednodušší úlohy v rámci automatizace, tak i pro složité a komplexní projekty.

[16]

3.3.2 PyQt

PyQt je knihovna, která umožňuje vytvářet grafické uživatelské rozhraní (GUI) pomocí Qt frameworku v jazyce Python. PyQt je jednou z nejoblíbenějších vazeb jazyka Python pro multiplatformní framework Qt C++. Qt je sada multiplatformních knihoven C++, které implementují vysokoúrovňové rozhraní API pro přístup k mnoha aspektům moderních desktopových a mobilních systémů. Patří mezi ně lokalizační a polohové služby, multimédia, připojení NFC a Bluetooth, webový prohlížeč založený na Chromiu a také pro tuto diplomovou práci nejdůležitější – tradiční vývoj uživatelského rozhraní. PyQt vyvinula společnost Riverbank Computing Limited. Samotné Qt je vyvíjeno jako součást projektu Qt.

[17]

V době psaní této diplomové práce existují 3 hlavní verze PyQt – PyQt4, PyQt5 a PyQt6.

Vývoj aplikací pomocí PyQt spočívá v psaní Python kódu, který využívá Qt funkce a třídy pro práci s GUI prvky. PyQt5 poskytuje přístup k většině Qt modulů, jako jsou QtCore, QtGui, QtWidgets, QtNetwork, QtMultimedia atd. PyQt5 také podporuje signály a sloty, což jsou mechanismy pro komunikaci mezi objekty pomocí událostí.

PyQt aplikace se skládají z několika částí. První část je hlavní skript, který importuje potřebné moduly, vytváří instanci QApplication a spouští hlavní smyčku událostí. Druhá část je třída pro okno aplikace, která dědí od QWidget nebo QMainWindow a definuje vzhled a chování okna. Třetí část je návrh GUI, který může být vytvořen pomocí Qt Designeru nebo ručně psaným kódem. Návrh GUI se pak načte do třídy pro okno aplikace pomocí nástroje pyuic5 nebo funkce „uic.loadUi“. Blíže je tento způsob načítání popsán v kapitole 4.7. Čtvrtá část je logika aplikace, která obsahuje funkce a metody pro reakci na uživatelské akce a provádění požadovaných operací.

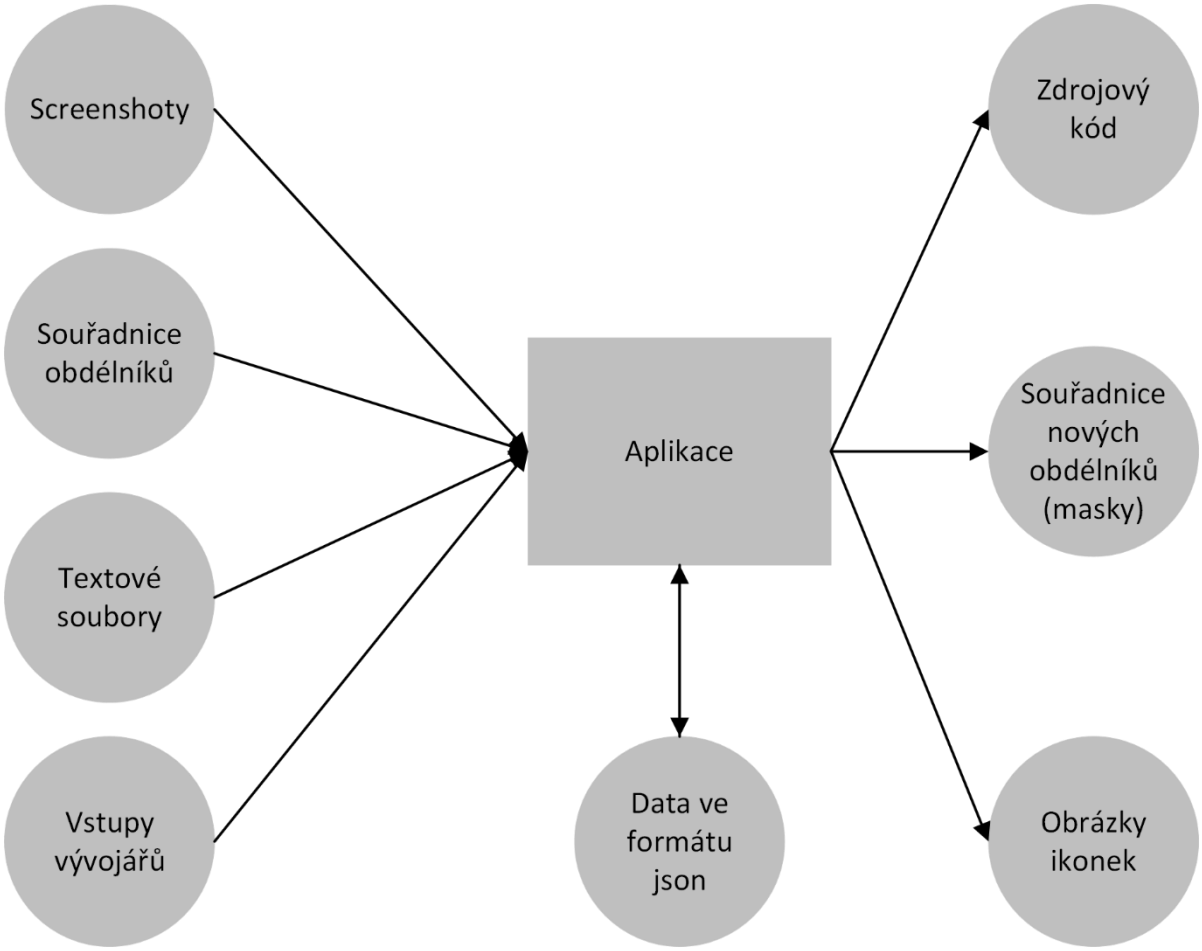
Pro účely vývoje navrhovaného nástroje byl zvolen PyQt ve verzi 5. Důvodem je vysoká stávající podpora, kterou tento framework nabízí v porovnání s PyQt4. Navíc je k dispozici větší množství dokumentace, což je zohledněno skutečností, že PyQt ve verzi 6 je poměrně nový.

4 Nástroj pro vytváření dat určených pro automatické ovládání infotainmentu

Na úvod této kapitoly je potřeba připomenout, že veškerá data (a tedy i finální zdrojový kód), která budou vytvářený nástroj generovat, musí být ve formátu tak, aby se dala použít a zasadit do již vytvořeného prostředí pro testování. Prostředí, ve kterém bude generovaný zdrojový kód používán, bylo vytvořeno firmou Digiteq Automotive s.r.o. (viz. kapitola 2.6), tudíž je nutné tyto výstupy tomuto prostředí přizpůsobit.

V rámci této kapitoly budou v první řadě popsána vstupní a výstupní data, se kterými bude navrhovaný nástroj pracovat. Dále bude popsáno, jak vypadá zdrojový kód, který bude generován a se kterým pracuje již vytvořený nástroj. V neposlední řadě bude popsán postup vývoje navrhovaného nástroje.

4.1 Vstupní a výstupní data



Obrázek 2 - Diagram vstupních a výstupních dat – zdroj vlastní

Na obrázku výše (Obrázek 2 - Diagram vstupních a výstupních dat) jsou graficky znázorněna vstupní a výstupní data, se kterými bude navrhovaný nástroj pracovat. Jednotlivá vstupní a výstupní data znázorněna na obrázku budou popsána v kapitolách 4.2 a 4.3.

4.1.1 Datové formáty JSON a XML

Vzhledem k tomu, že v dalších podkapitolách budou používány zkratky datových formátů JSON a XML, je nutné si připomenout, co tyto zkratky představují.

4.1.1.1 JSON

*„**JSON** (JavaScript Object Notation) je odlehčený formát pro výměnu dat. Je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojově. Je založen na podmnožině programovacího jazyka JavaScript, Standard ECMA-262 3rd Edition – December 1999. JSON je textový, na jazyce zcela nezávislý formát, využívající však konvence dobře známé*

programátorům jazyků rodiny C (C, C++, C#, Java, JavaScript, Perl, Python a dalších). Díky tomu je JSON pro výměnu dat opravdu ideálním jazykem.

JSON je založen na dvou strukturách:

- *Kolekce párů název/hodnota. Ta bývá v rozličných jazycích realizována jako objekt, záznam (record), struktura (struct), slovník (dictionary), hash tabulka, klíčový seznam (keyed list) nebo asociativní pole.*
- *Seřazený seznam hodnot. Ten je ve většině jazyků realizován jako pole, vektor, seznam (list) nebo posloupnost (sequence).*

Jedná se o univerzální datové struktury a v podstatě všechny moderní programovací jazyky je v nějaké formě podporují. Je tedy logické, aby na nich byl založen i na jazyce nezávislý výměnný formát.“

[18]

4.1.1.2 XML

XML je zkratka pro „Extensible Markup Language“. Jedná se o textový značkovací jazyk odvozený ze standardního zobecněného značkovacího jazyka (SGML). Značky XML identifikují data a slouží k jejich uložení a uspořádání, nikoli k určení způsobu jejich zobrazení jako značky HTML, které se používají k zobrazení dat.

Existují tři důležité vlastnosti XML, díky nimiž je XML užitečný v různých systémech a řešeních:

- XML je rozšiřitelný – XML umožňuje vytvářet vlastní popisné značky nebo jazyk, který vyhovuje vaší aplikaci.
- XML přenáší data, neprezentuje je – XML umožňuje ukládat data bez ohledu na to, jak budou prezentována.
- XML je veřejný standard – XML vyvinula organizace World Wide Web Consortium (W3C) a je k dispozici jako otevřený standard.

[19]

4.2 Vstupní data

Jak je znázorněno na obrázku výše (Obrázek 2 - Diagram vstupních a výstupních dat), aplikace uvažuje s několika druhy vstupních dat. Pro to, aby bylo možné vygenerovat data výstupní, je nejprve nutné mít k dispozici a shromáždit data vstupní. V této podkapitole budou jednotlivá vstupní data vysvětlena.

- **Screenshoty** – obrázky screenshotů jednotlivých obrazovek, které si bude aplikace načítat jako vstup a bude je zobrazovat. Uživatel aplikace bude pak s těmito obrázky interagovat pomocí kreslení obdélníků, které budou ohraničovat prvky na obrazovce a budou nositeli informací o těchto prvcích.

Obrázek 3 - Screenshot obrazovky infotainmentu – zdroj vlastní

- **Souřadnice obdélníků** – souřadnice obdélníků, ohraničující prvky na obrazovce, které uživatel nakreslí. V případě, že daný prvek na obrazovce již souřadnice má (ve formě tzv. masky), uživatel po nakreslení obdélníku bude mít možnost vybrat soubor se souřadnicemi. Obdélník, který uživatel na obrazovku nakreslil se po načtení souboru (masky) přesune do pozice, která bude specifikována tímto vybraným souborem.
- **Textové soubory** – textové soubory, které obsahují informace o všech textových popiscích, které se vyskytují napříč celým operačním systémem infotainmentu. Tyto textové soubory jsou ve formátu XML a každý jednotlivý soubor reprezentuje jednu jazykovou verzi infotainmentu. Tyto textové soubory jsou vývojářům test automatizace poskytovány od zákazníka.

```
<LangID ID="LANG_CAR_STATUS_Warn_TextDEnglish_a683_Labels" Text="Asistent vyparkování je nedostupný. Senzor nemá výhled."/>
<LangID ID="LANG_CAR_STATUS_Warn_TextDEnglish_a684_Labels" Text="Přívěs: asistent vyparkování je nedostupný."/>
<LangID ID="LANG_CAR_STATUS_Warn_TextDEnglish_a685_Labels" Text="Side Assist a výstraha vystupování nyní nedostupné."/>
<LangID ID="LANG_CAR_STATUS_Warn_TextDEnglish_a686_Labels" Text="Závada: Side Assist a výstraha vystupování"/>
<LangID ID="LANG_CAR_STATUS_Warn_TextDEnglish_a687_Labels" Text="Side Assist a výstraha vystupování: senzor nemá výhled."/>
<LangID ID="LANG_CAR_STATUS_Warn_TextDEnglish_a688_Labels" Text="Side Assist a výstraha vystupování nedostupné."/>
<LangID ID="LANG_CAR_STATUS_Warn_TextDEnglish_a689_Labels" Text="Side Assist a asistent vyparkování jsou nedostupné."/>
<LangID ID="LANG_CAR_STATUS_Warn_TextDEnglish_a68a_Labels" Text="Závada: Side Assist a asistent vyparkování"/>
```

Obrázek 4 - Příklad části textového souboru ve formátu XML – zdroj vlastní

- **Vstupy vývojářů** – pro to, aby bylo možné vygenerovat zdrojová data správně, bude nutné, aby aplikace umožňovala uživatelům doplnit všechny potřebné informace, které budou součástí generovaného skriptu. Mezi tyto informace patří například názvy jednotlivých proměnných či argumenty funkcí ve výstupním zdrojovém kódu, nebo také popis jednotlivých prvků na obrazovce. Blíže budou tyto vstupy popsány v kapitole 4.6.4.

4.3 Výstupní data

Jak již bylo stručně popsáno v kapitole 2.6, hlavním funkcionalitou navrhovaného nástroje bude schopnost generování dat v podobě skriptů, které popisují jednotlivé obrazovky infotainmentu. Kromě těchto skriptů bude navrhovaný nástroj umět generovat i několik dalších dat.

- **Data ve formě zdrojového kódu v programovacím jazyce Python** – cílová výstupní data budou ve formě zdrojového kódu v programovacím jazyce Python, jejichž účelem je popis zobrazovaných obrazovek na displeji infotainmentu. Tento zdrojový kód bude sloužit jako vzor pomocí kterého automatizované testy rozpoznají jednotlivé elementy na obrazovce, čímž je v rámci automatizovaného testování umožněna navigace napříč infotainmentem.

- **Data ve formátu XML** – jedná se o souřadnice obdélníků, které uživatel nakreslí na screenshot obrazovky. V případě, že se bude na zkoumané obrazovce nacházet prvek, pro který ještě nebyl vytvořen XML soubor ve formě tzv. „masky“, popisující jeho souřadnice, aplikace tento strukturovaný soubor („masku“) na základě uživatelského vstupu vytvoří tak, aby byl následně použitelný pro stejný prvek na jiné obrazovce.

```xml
<?xml version="1.0" encoding="utf-8"?>
<MASK name="acc">
    <ELEMENTS>
        <ELEMENT type="Rectangle" index="0">
            <OffsetX>333</OffsetX>
            <OffsetY>72</OffsetY>
            <SizeX>141</SizeX>
            <SizeY>120</SizeY>
        </ELEMENT>
    </ELEMENTS>
</MASK>
```

Obrázek 5 - Ukázka souboru XML, který reprezentuje souřadnice (masku) prvku na obrazovce – zdroj vlastní

- **Data ve formátu JSON** – k tomu, aby bylo možné vygenerovat správný a pro automatizační nástroj použitelný zdrojový kód popisující obrazovku infotainmentu, bude potřeba shromáždit veškeré informace o všech zobrazovaných prvcích na zkoumané obrazovce. K tomu bude sloužit navrhovaná aplikace, která bude na základě uživatelských vstupů tyto informace shromažďovat. Nejprve se budou strukturovaně zapisovat do datového formátu „JSON“, který bude sloužit jako pomocník pro modul, který bude mít na starost generování zdrojového kódu. Tento modul je samozřejmě součástí navrhovaného nástroje, tudíž aplikace bude tato data jak generovat, tak také zpracovávat a tato data nebudou využita jinak, než touto aplikací.

4.4 Popis a vlastnosti generovaného kódu

Jak již bylo zmíněno v předchozích kapitolách, generovaný zdrojový kód má reflektovat popis elementů na obrazovce infotainmentu tak, aby vytvořené prostředí pro automatizaci s danou obrazovkou a jejími elementy dokázalo pracovat. Níže je přiložena ukázka zdrojového kódu, který se již používá:

```
BT_BROWSER_GENRE_LIST = Screen("BT_BROWSER_GENRE_LIST") \
    .add_identification(
    TextToolTag(texttool="LANG_MediaMyTab_FilterCriteria_Genre",
mask="screen_title_media%d"),
    IconTag(icon="music_genre", mask="teleport_menu_row%d")) \
    .add_element(Button(name="MediaBrowser",
identification_tag=IconTag(icon="source_my_bluetooth",
mask="screen_title_menu")), "BT_BROWSER_MUSIC") \
    .add_element(SetBox(name="TestGenre",
identification_tag=TextTag(text="[0-9]{2,5}Hz", method="REGEX",
mask="teleport_menu_browser_row%d"))) \
    .add_element(Button(name="Unknown",
identification_tag=TextToolTag(texttool="LANG_MediaBrowser_FilterCri
teria_UnknownGenre", mask="teleport_menu_browser_row%d",
threshold=0.75)), "BT_UNKNOWN_GENRE") \
    .add_element(SetBox(name="PlaybackActive",
identification_tag=IconTag(icon="playback_active",
mask="teleport_menu_row%d")))
```

Obrázek 6 - Příklad zdrojového kódu popisující obrazovku infotainmentu – zdroj vlastní

Na obrázku výše (Obrázek 6 - Příklad zdrojového kódu popisující obrazovku infotainmentu) lze vidět příklad zdrojového kódu, který se používá při spouštění automatizovaných testů. Do proměnné se ukládá instance třídy „Screen“, která je součástí již vytvořeného prostředí. Název proměnné, do které se informace o obrazovce ukládají je vždy velkými písmeny. Třída „Screen“ má 2 hlavní metody, které se používají pro přidání definice prvků na obrazovce.

První metoda „add_identification“ má za úkol přidat (namapovat pro automatizaci) prvky, které se klasifikují jako tzv. identifikace. Argumenty této metody jsou instance tříd, které popisují jednotlivé prvky, které se můžou vyskytovat na obrazovce. Příkladem může být (jak je vidět na obrázku výše) třída „TextToolTag“ nebo „IconTag“. Instance těchto tříd se pak následně vytvářejí s použitím proměnných (tzv. „instance variables“ nebo také atributy objektu), které jsou vždy specifické pro

každý prvek. Všechny možné třídy (tedy identifikace), které lze v rámci skriptu použít společně s atributy vytvářeného objektu lze vidět níže na obrázku (Obrázek 7 - Identifikace a jejich atributy), který dané kombinace zobrazuje ve formátu JSON:

```
{
    "TextTag": ["text", "method", "mask", "threshold"],
    "IconTag": ["icon", "mask"],
    "TextToolTag": ["texttool", "mask", "threshold"],
    "PlaceHolderTag": ["key", "mask", "preprocessing_mode"]
}
```

Obrázek 7 - Identifikace a jejich atributy – zdroj vlastní

Druhá metoda „add_element“ přidává na obrazovku prvky, které se klasifikují jako elementy. Stejně jako u metody „add_identification“ se jako argument vkládají instance tříd, jejichž kombinace a atributy jsou popsané na obrázku níže (Obrázek 8 - Elementy a jejich atributy) (ukázka na obrázku je opět ve formátu JSON):

```
{
    "Button": ["name", "identification_tag", "mask"],
    "SetBox": ["name", "identification_tag", "mask"],
    "CheckBox.create": ["name", "element_identification_tag",
"false_identification_tag", "true_identification_tag"],
    "ParamCircular.create": ["name", "mask",
"element_identification_tag", "values", "value_identificators",
"threshold"]
}
```

Obrázek 8 - Elementy a jejich atributy – zdroj vlastní

Třídy elementů v sobě však obsahují atributy, které se inicializují pomocí instancí identifikací. Mezi tyto atributy patří například v elementu „Button“ atribut „identification_tag“. Příklad použití:

```
.add_element(Button(name="MediaBrowser",
identification_tag=IconTag(icon="source_my_bluetooth",
mask="screen_title_menu")), "BT_BROWSER_MUSIC")
```

Obrázek 9 - Příklad použití metody add_element a jejich argumentů – zdroj vlastní

Jak lze vidět na obrázku (Obrázek 9 - Příklad použití metody add_element a jejich argumentů) výše, argument „identification_tag“ se inicializuje s použitím třídy „IconTag“, která je součástí prvků identifikací. Argumenty této třídy pak zůstávají stejné, jak je popsané na obrázku (Obrázek 7 - Identifikace a jejich atributy) výše (v tomto případě se tedy pro třídu „IconTag“ používají atributy „icon“ a „mask“).

Předmětem této diplomové práce není analyzovat jednotlivé třídy či argumenty funkcí a metod, které již vytvořená automatizace využívá. Z tohoto důvodu nebude zpracována podrobnější analýza těchto parametrů.

4.5 Mapovací tabulky

S ohledem na stručně provedenou analýzu zdrojového kódu, který má aplikace generovat, je na začátek nutné si vytvořit tzv. „mapovací tabulky“, které budou specifikovat všechny možné parametry těchto metod. Pomocí těchto mapovacích tabulek se bude řídit business logika navrhované aplikace. Díky tomu bude aplikace vědět, jaké parametry má uživatel při zvolení příslušného prvku vyplnit a zamezí se tak nechtěnému vyplnění parametrů někam, kam tyto parametry nepatří.

Pro jednoduchost tvorby těchto mapovacích tabulek a jejich následného čtení pomocí programovacího jazyka Python byl zvolen formát JSON. V Pythonu lze totiž použít vestavěnou knihovnu „json“ pro práci s JSON formátem. Knihovna obsahuje funkce pro převod datových struktur na řetězce JSON a naopak, což výrazně autorovi usnadní práci.

4.5.1 Mapovací tabulka všech prvků a jejich atributů

První mapovací tabulka bude obsahovat informace o všech prvcích (tedy identifikacích a elementech) a všech jejich možných atributů. Tato mapovací tabulka je znázorněna na obrázku (Obrázek 10 - Mapovací tabulka identifikací a elementů včetně jejich atributů ve formátu JSON) níže.

```
{
    "TextTag": ["text", "method", "mask", "threshold"],
    "IconTag": ["icon", "mask"],
    "TextToolTag": ["texttool", "mask", "threshold"],
    "PlaceHolderTag": ["key", "mask", "preprocessing_mode"],
    "Button": ["name", "identification_tag", "mask"],
    "SetBox": ["name", "identification_tag", "mask"],
    "CheckBox.create": ["name", "element_identification_tag",
"false_identification_tag", "true_identification_tag"],
    "ParamCircular.create": ["name", "mask",
"element_identification_tag", "values", "value_identificators",
"threshold"]
}
```

Obrázek 10 - Mapovací tabulka identifikací a elementů včetně jejich atributů ve formátu JSON – zdroj vlastní

Klíče jsou třídy, které se ve zdrojovém kódu používají a jejich hodnoty jsou listy obsahující všechny inicializační atributy vytvářených objektů. Pomocí této mapovací tabulky se bude řídit business logika aplikace. Například, pokud uživatel v aplikaci zvolí „TextTag“, aplikace pomocí této mapovací tabulky nabídne uživateli jen ta pole, která jsou uvedena v této mapovací tabulce (vyplňovaná pole pro tento případ budou tedy text, method, mask a threshold).

4.5.2 Mapovací tabulka identifikací a elementů

Druhá mapovací tabulka bude specifikovat, zda při se při zvolení daného prvku jedná o identifikaci či element. Ukázka této mapovací tabulky je na obrázku níže (Obrázek 11 - Mapovací tabulka ve formátu JSON rozlišující prvky na identifikace a elementy).

```
{
    "Identification": ["TextTag", "IconTag", "TextToolTag",
"PlaceHolderTag"],
    "Element": ["Button", "SetBox", "CheckBox.create",
"ParamCircular.create", "name"]
}
```

Obrázek 11 - Mapovací tabulka ve formátu JSON rozlišující prvky na identifikace a elementy – zdroj vlastní

4.5.3 Mapovací tabulka řídících jednotek

Poslední mapovací tabulka bude specifikovat seznam řídících jednotek a cesty k souborům, pro které se bude vytvářet zdrojový kód. Tato mapovací tabulka se následně promítne do grafického rozhraní aplikace tak, že uživateli nabídne ze seznamu pouze ty jednotky, které jsou uvedeny v této mapovací tabulce. Ukázka této mapovací tabulky je znázorněna na obrázku níže (Obrázek 12 - Ukázka mapovací tabulky ve formátu JSON pro fiktivní řídící jednotku).

V ukázce obrázku jsou cesty k jednotlivým souborům pouze imaginární, neboť ty skutečné vedou k datovým úložištím v rámci společnosti Digiteq Automotive s.r.o. Pro správné fungování aplikace na počítačích, které nemají k těmto úložištím přístup, je nutné tyto cesty upravit. Z důvodu obchodního tajemství společnosti Škoda Auto se nebudou uvádět přesné názvy, ale jen fiktivní jména.

```
{
"Unit1": {
    "textool" : "C:\\Users\\U5Q80PK\\Desktop\\Texttool\\Unit1",
    "icons": "C:\\Users\\U5Q80PK\\Desktop\\Icons\\Unit1",
    "masks": "C:\\Users\\U5Q80PK\\Desktop\\Masks\\Unit1"
},
```

Obrázek 12 - Ukázka mapovací tabulky ve formátu JSON pro fiktivní řídící jednotku – zdroj vlastní

4.6 Tvorba GUI aplikace

Pro tvorbu uživatelského rozhraní vyvíjeného nástroje (tedy desktopové aplikace) byl vybrán open source nástroj Qt Designer. Tento program umožňuje rychle a snadno vytvořit veškeré prvky, které budou tvořit vzhled a funkčnost vyvíjené aplikace.

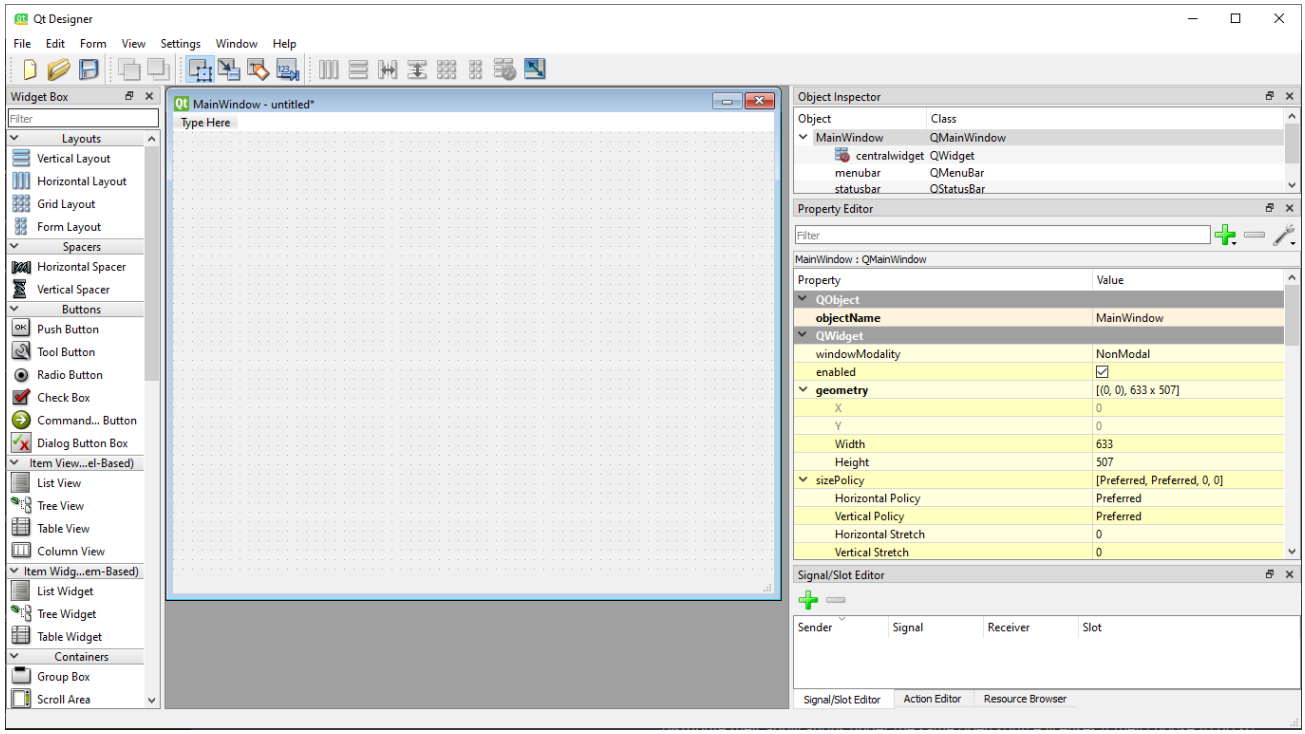
4.6.1 Qt Designer

Qt Designer disponuje širokou paletou předdefinovaných prvků, včetně tlačítek, formulářů, tabulek a mnoho dalších, a také umožňuje přizpůsobit vzhled těchto prvků podle potřeb vývojářů.

Díky jednoduché drag and drop funkčnosti lze snadno a rychle vytvářet layouty, do kterých je možné následně vkládat jednotlivé prvky. Tento postup značně usnadňuje práci při tvorbě uživatelského rozhraní a šetří čas.

Další výhodou Qt Designeru je, že umožňuje pracovat s kódem v jazyce Python. Po vytvoření designu se design ukládá ve formě tzv. „ui“ souborů, které jsou formátované jako XML soubory. Tyto soubory se pak pomocí knihovny PyQt jednoduše načtou a lze s jednotlivými prvky aplikace (tlačítky, combo boxy atd.) následně pracovat přímo v kódu.

Použití Qt Designeru umožňuje efektivní tvorbu uživatelského rozhraní a usnadňuje práci při vývoji aplikace.

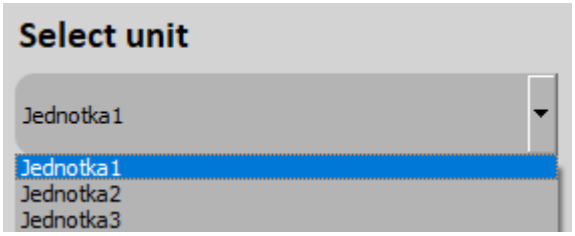


Obrázek 13 - Rozhraní aplikace Qt Designer – zdroj vlastní

4.6.2 Tvorba designu aplikace v Qt Designeru

Hlavní okno aplikace je jedna z nejdůležitějších částí celého aplikačního rozhraní. Uživatel hlavní okno uvidí okamžitě po spuštění aplikace. Jeho návrh je tedy kritickou částí při tvorbě tohoto nástroje.

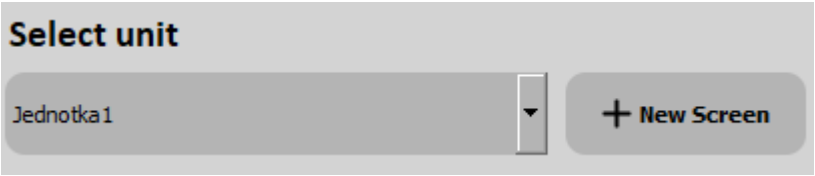
Po spuštění aplikace bude uživatel nucen vybrat ze seznamu řídící jednotku, pro kterou daný skript chce vytvářet. Tento seznam bude odpovídat seznamu z mapovací tabulky pro řídící jednotky (4.5.3). Opět je potřeba zmínit, že z důvodu obchodního tajemství firmy Škoda Auto jsou na obrázku níže (Obrázek 14 - Ukázka combo boxu) použity pouze fiktivní názvy řídících jednotek.



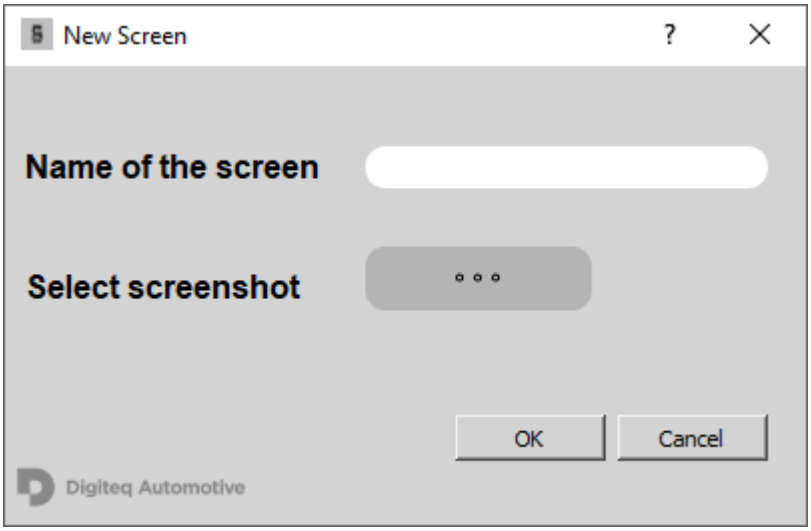
Obrázek 14 - Ukázka combo boxu – zdroj vlastní

Součástí hlavního okna aplikace budou také tlačítka, které budou mít různé funkcionality a budou nastaveny tak, aby byly zobrazovány jen v případě, že budou moci být použity. Mezi tyto tlačítka patří:

- „New Screen“ tlačítko, které po stisknutí otevře dialogové okno, ve kterém uživatel vyplní název obrazovky a vybere screenshot obrazovky, pro kterou bude generovat zdrojový kód. Toto tlačítko se zobrazí pouze v případě, že uživatel v combo boxu zvolí řídící jednotku, pro kterou bude chtít generovat zdrojový kód.



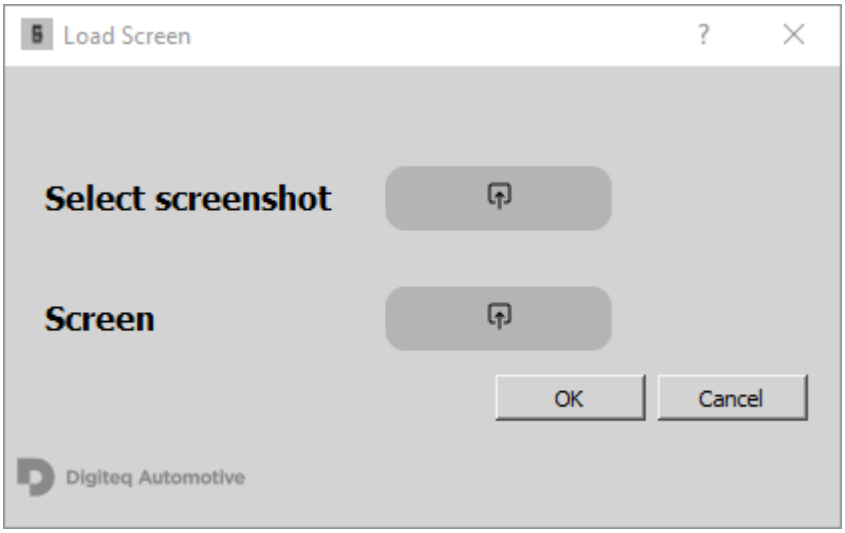
Obrázek 15 - Ukázka tlačítka "New Screen" – zdroj vlastní



Obrázek 16 - Ukázka dialogového okna zobrazovaného po stisknutí tlačítka "New Screen" – zdroj vlastní

- „Load Screen“ tlačítko, které uživateli umožní otevřít již předem rozpracovaný projekt. Po stisknutí tohoto tlačítka se otevře dialogové okno, ve kterém uživatel vybere screenshot obrazovky a již předem vytvořený JSON soubor s informacemi o obrazovce, pro kterou bude chtít generovat zdrojový kód.

Obrázek 17 - Ukázka tlačítka "Load Screen" – zdroj vlastní

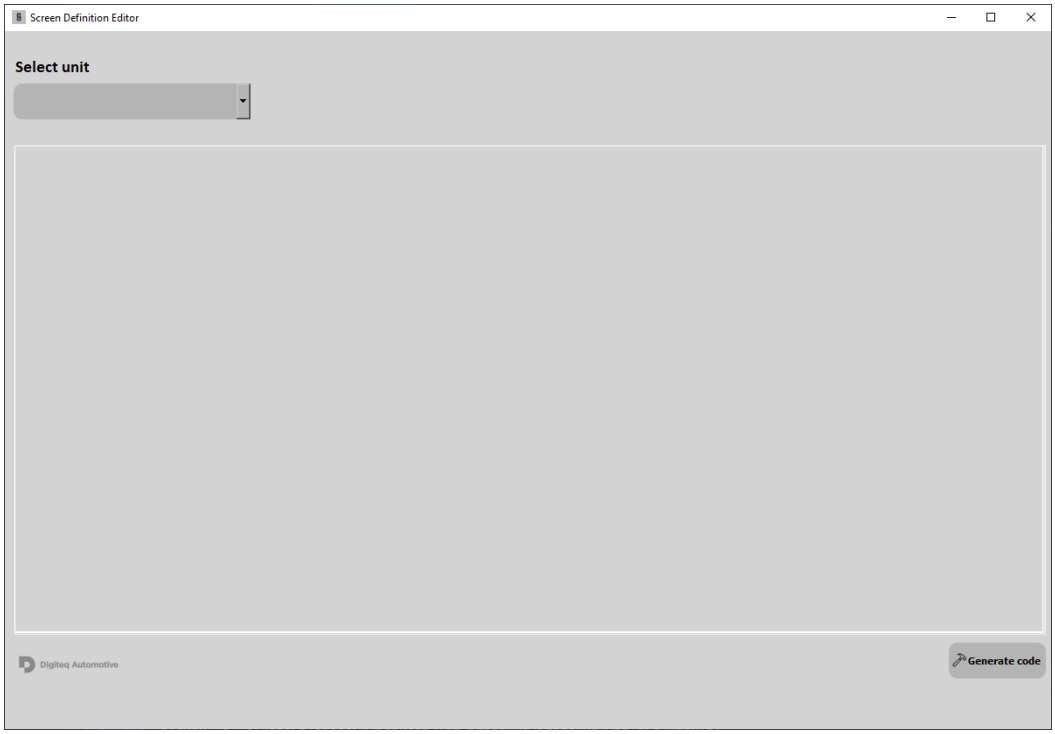


Obrázek 18 - Ukázka dialogového okna zobrazovaného po stisknutí tlačítka "Load Screen" – zdroj vlastní

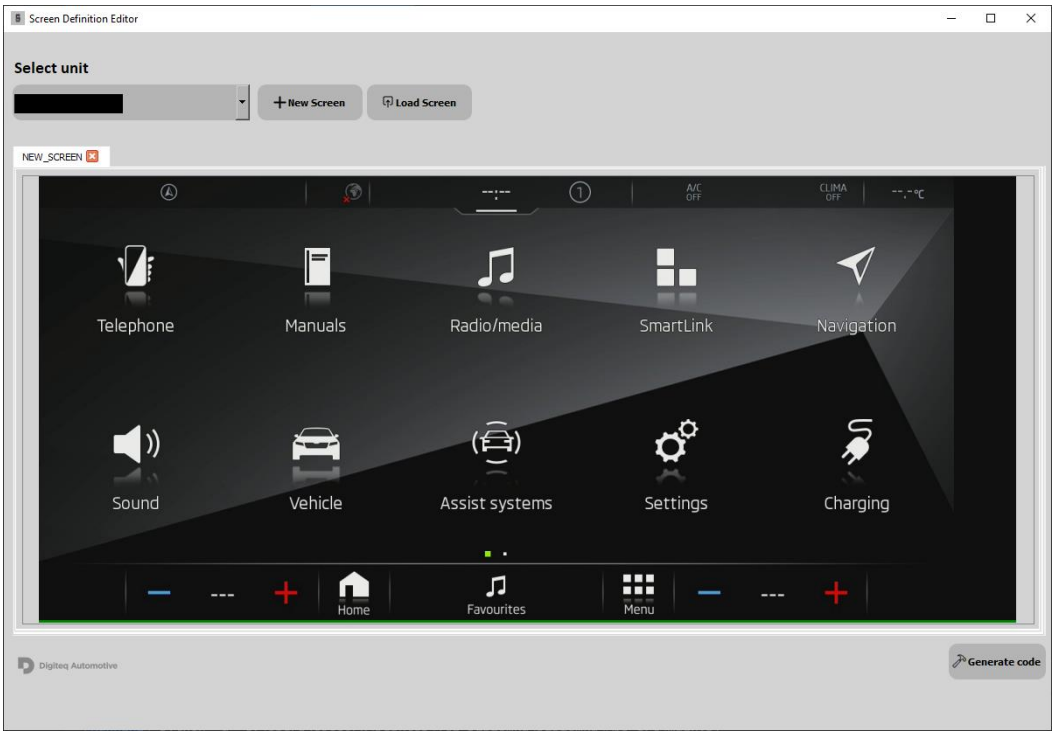
- „Generate code“ tlačítko, které po stisknutí otevře dialogové okno, ve kterém uživatel vybere vygenerovaný soubor ve formátu JSON popisující danou obrazovku a místo, kam bude chtít vygenerovaný skript uložit.

Obrázek 19 - Ukázka dialogového okna zobrazovaného po stisknutí tlačítka "Generate Code" – zdroj vlastní

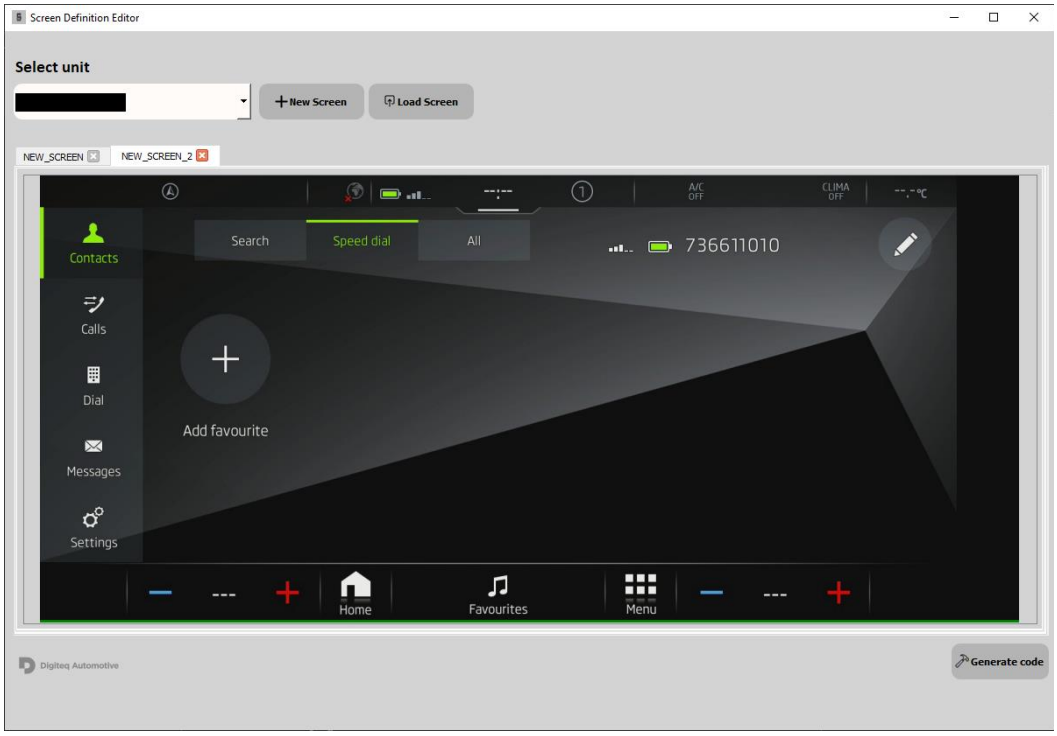
- Po spuštění aplikace se otevře „prázdné“ okno, ve kterém bude viditelný pouze combobox pro vybrání příslušné jednotky, tlačítko „Generate Code“ a prázdné „Tab View“ okno, které později bude sloužit jako hlavní zobrazovací pole screenshotů. Na obrázku (Obrázek 20 - Ukázka celého "Main Window" okna aplikace po spuštění aplikace) lze vidět prázdné hlavní okno aplikace. Na obrázku (Obrázek 21 - Ukázka "Main Window" okna aplikace po vybrání screenshotu) lze pak vidět hlavní okno s vybraným a zobrazeným screenshotem. Obrázek se zobrazuje v poli „Tab View“. Díky tomu je možné načíst v aplikaci několik obrázků (a tedy obrazovek) najednou. Takové zobrazení lze vidět na obrázku (Obrázek 22 - Ukázka "Main Window" okna aplikace po vybrání dvou screenshotů a dvou záložek v rámci "tab view"), kde jsou v aplikaci otevřené 2 záložky s obrazovkami.



Obrázek 20 - Ukázka celého "Main Window" okna aplikace po spuštění aplikace – zdroj vlastní



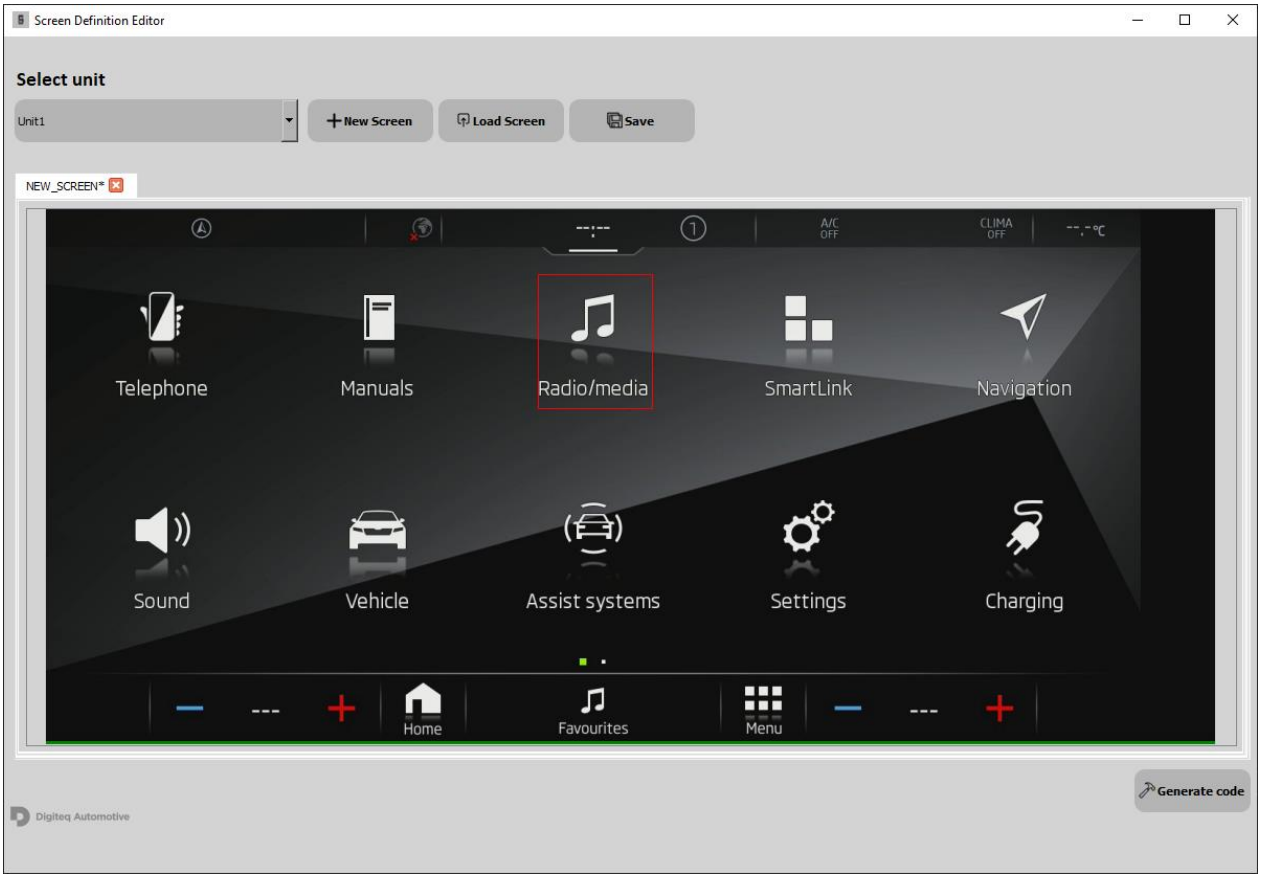
Obrázek 21 - Ukázka "Main Window" okna aplikace po vybrání screenshotu – zdroj vlastní



Obrázek 22 - Ukázka "Main Window" okna aplikace po vybrání dvou screenshotů a dvou záložek v rámci "tab view" – zdroj vlastní

4.6.3 Práce s obrázky a definování prvků

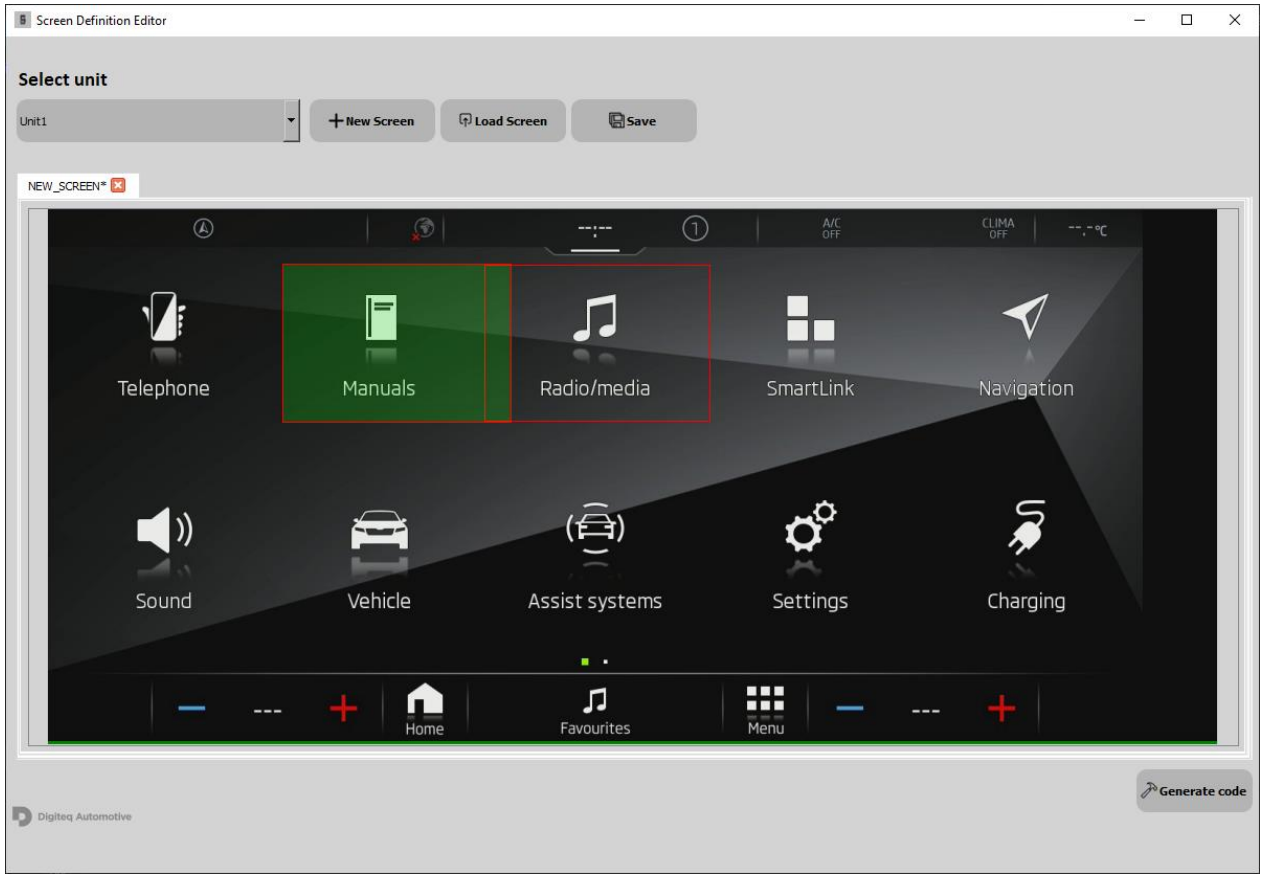
V kapitole 4.6.2 bylo popsáno, jak bude vypadat hlavní okno aplikace. Jakmile uživatel načte obrázky do vyhrazeného pole aplikace, může začít pracovat na vytváření zdrojových dat. Pro definici jednotlivých prvků na obrazovce bude k dispozici nástroj "kreslení myší", který umožní vytvořit obdélníkové pole ohraničující prostor daného prvku na obrazovce. Proces se spustí stisknutím a podržením levého tlačítka myši a následným tažením. Po uvolnění tlačítka myši se ohraničení pole obarví červeně a otevře se dialogové okno. Toto dialogové okno bude sloužit jako „sběratel“ informací o ohraničeném prvku na obrazovce. Na obrázku (Obrázek 23 - Ukázka ohraničeného elementu na obrazovce) níže je ukázka takového ohraničeného pole v podobě obdélníku. Dále lze nově vidět i nové tlačítko „Save“, které po stisknutí otevře dialogové okno souboru, ve kterém uživatel vybere místo, kam chce svou práci uložit. Pokud již uživatel dříve nastavil umístění souboru, tlačítko uloží změny do této předem definované cesty. Pokud dojde k uložení souboru, hvězdička v názvu záložky, která dříve indikovala, že práce nebyla uložena, zmizí.



Obrázek 23 - Ukázka ohraničeného elementu na obrazovce – zdroj vlastní

Po nakreslení obdélníku a uvolnění levého tlačítka myši se automaticky otevře dialogové okno, které bude blíže popsáno v kapitole 4.6.4.

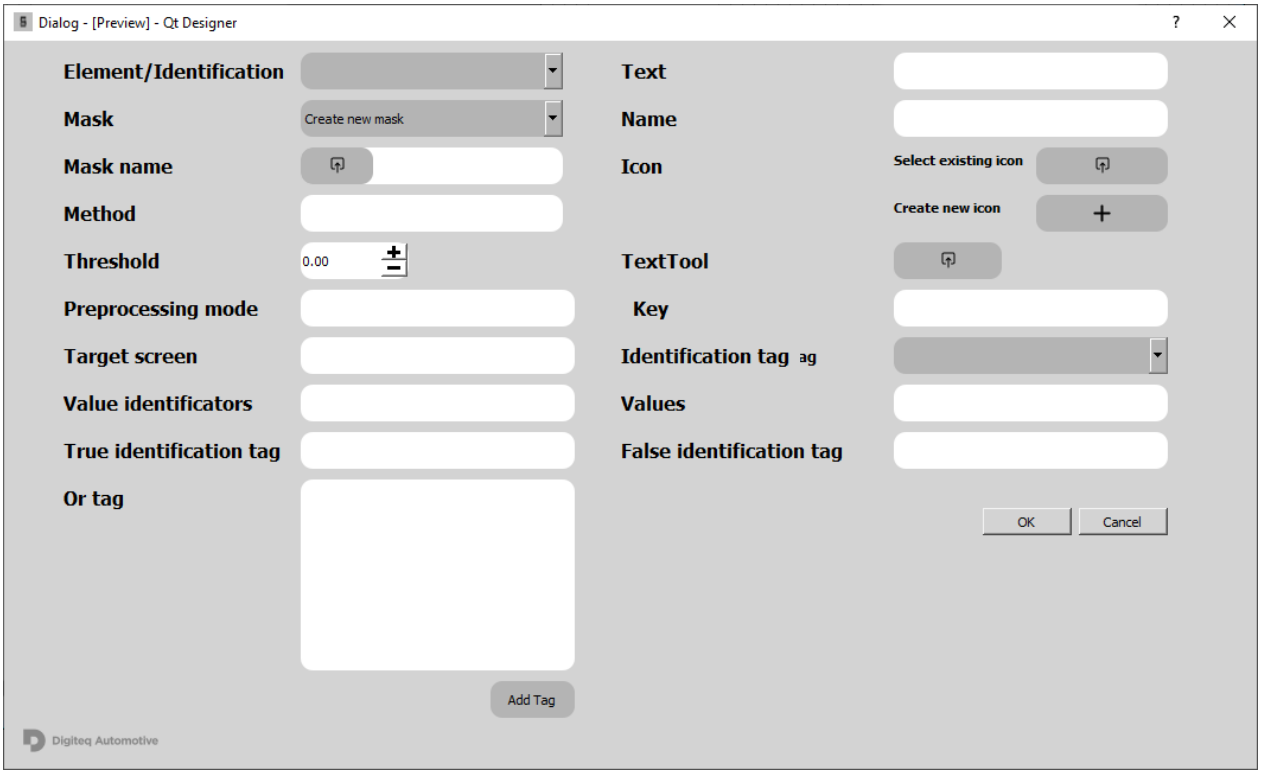
Jak již bylo zmíněno výše, obdélníky uchovávají informace o prvku na obrazovce. Je tudíž nutné uživateli umožnit dané pole (obdélník) znovu otevřít tak, aby mohl informace upravovat dle libosti. K tomu se uživatel dostane tak, že dvakrát klikne dovnitř vyhraničeného prostoru. Prvním kliknutím se prostor podbarví zeleně tak, aby si uživatel mohl ověřit, že klikl opravdu do prostoru toho obdélníku, který chce upravovat. Tato funkcionalita přijde vhod hlavně v momentě, kdy se budou 2 nebo více obdélníků překrývat. Tuto situaci lze vidět na obrázku (Obrázek 24 - Zobrazení označeného pole a překryvu obdélníků) níže.



Obrázek 24 - Zobrazení označeného pole a překryvu obdélníků – zdroj vlastní

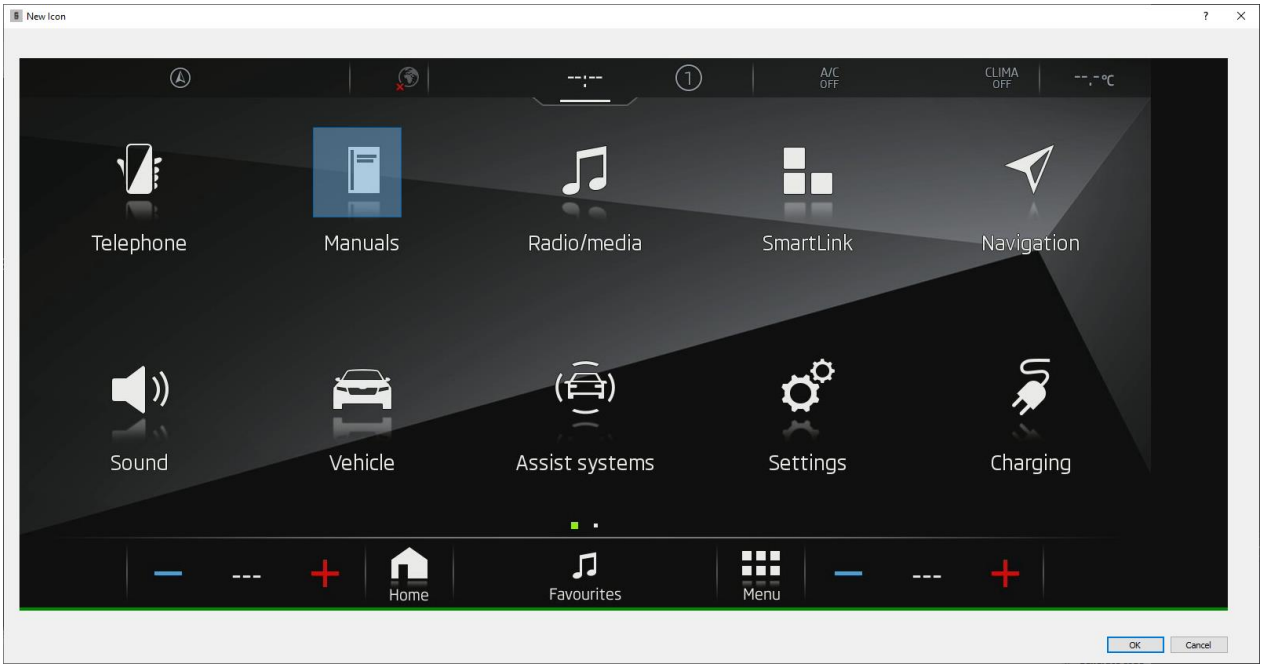
4.6.4 Dialogové okno s informacemi o ohraničeném prvku

Jak bylo řečeno v předchozích kapitolách, po nakreslení obdélníku a uvolnění levého tlačítka myši se otevře dialogové okno. V tomto dialogovém okně bude uživatel aplikace vyplňovat informace o daném prvku. Informace se vyplňují prostřednictvím textu, vybráním souboru nebo hodnot. Každý prvek na obrazovce může nabývat několika předem specifikovaných hodnot. Tyto hodnoty jsou znázorněny v mapovacích tabulkách v kapitolách 4.5.1 a 4.5.2. Na obrázku (Obrázek 25 - Ukázka dialogového okna uchovávající informace o prvku na obrazovce) níže lze vidět ukázku dialogového okna jako „preview“ v Qt Designeru, na kterém lze vidět všechny atributy, které bude moci uživatel vyplnit. Důvod, proč byl screenshot vytvořen jako „preview“ v Qt Designeru je ten, že v běžící aplikaci uživatel nikdy neuvidí všechna tato pole k vyplnění, protože pro každý prvek budou viditelná jen ta pole, která dle mapovací tabulky lze k danému prvku přiřadit.



Obrázek 25 - Ukázka dialogového okna uchovávající informace o prvku na obrazovce – zdroj vlastní

Jak již bylo zmíněno v kapitole výše (Výstupní data), aplikace bude umět ukládat také nové ikony na obrazovce. K tomu slouží tlačítko „Create new icon“ na obrázku výše (Obrázek 25 - Ukázka dialogového okna uchovávající informace o prvku na obrazovce). Po stisknutí tohoto tlačítka se uživateli zobrazí nové dialogové okno, ve kterém bude zobrazen stejný screenshot jako na hlavní obrazovce s tím rozdílem, že v tomto okně bude mít uživatel možnost udělat výstřižek, který bude moci uložit jako ikonu. Výstřižek se bude tvořit podobně jako v případě kreslení obdélníků na screenshotu obrazovky v hlavním okně. Stisknutím levého tlačítka myši a následným tažením myši se na obrazovce začne vykreslovat obdélník, který bude ohraničovat pole výstřižku, které se bude následně ukládat jako ikona. Po uvolnění levého tlačítka myši se uživateli zobrazí dialogové okno, které bude uživatele vyzývat k tomu, aby vybral umístění ukládaného souboru. Praktická ukázka takového okna společně se zobrazeným obdélníkem ohraničující ikonu je znázorněna na obrázku Obrázek 26 - Tvorba výstřižku) níže.



Obrázek 26 - Tvorba výstřižku – zdroj vlastní

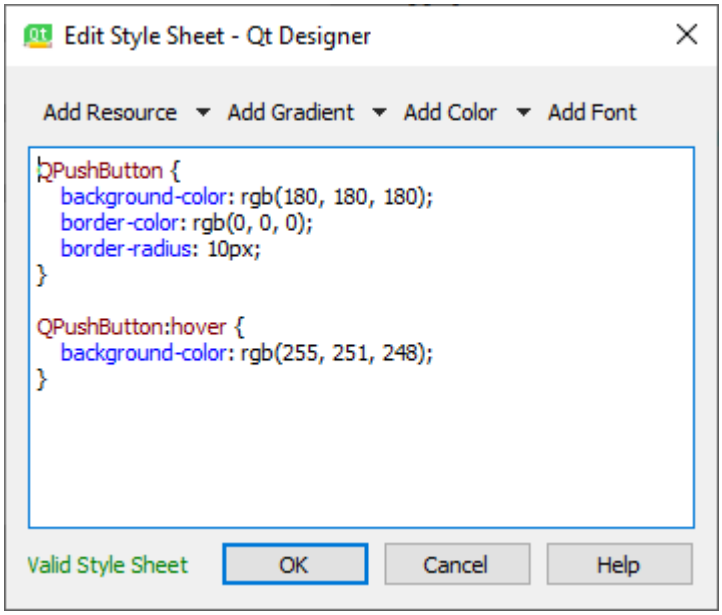
4.6.5 Design aplikace

Autor této diplomové práce vytvořil veškerý design aplikace s využitím snadno ovladatelného a intuitivního nástroje Qt Designer. Díky této efektivní metodě bylo možné v relativně krátkém čase vytvořit návrh aplikace, který je poměrně intuitivní a pro někoho dokonce vizuálně atraktivní.

Ikony použité v aplikaci jako grafické prvky pro jednotlivá tlačítka jsou k dispozici zdarma. Přidání ikony do tlačítka je velmi snadné – stačí vybrat tlačítko v návrhu a pomocí nabídky vybrat požadovanou ikonu nebo vybrat soubor s ikonou z disku. Pro přidání obrázku jako ikony okna aplikace byl použit atribut "windowIcon", který umožňuje importovat obrázek z disku počítače a nastavit ho jako ikonu okna.

V aplikacích vytvořených v Qt Designeru je také možné přidat interaktivitu na tlačítka pomocí funkce "hover", která změní vzhled tlačítka při přejetí myší nad něj. Tato funkce umožňuje uživatelům rychleji identifikovat aktivní prvky uživatelského rozhraní a zvýšit tak interaktivitu aplikace.

Pro přidání funkce "hover" na tlačítko je potřeba vybrat tlačítko v návrhu, a v editoru vlastností vybrat položku "StyleSheet". Poté je třeba kliknout na tlačítko "Edit..." a vložit CSS kód, který nastaví nový vzhled tlačítka při přejetí myší nad něj. V případě vyvíjené aplikace byly pro všechna tlačítka nastaveny stejné styly. Ukázka takového nastavení lze vidět ne obrázku (Obrázek 27 - CCS kód k nastavení designu tlačítka) níže:



Obrázek 27 - CCS kód k nastavení designu tlačítka – zdroj vlastní

4.7 Načtení vytvořených souborů z Qt Designeru do Pythonu

V minulých podkapitolách bylo popsáno, jakým způsobem byl tvořen design vyvíjené aplikace. Vzhledem k tomu, že se při návrhu v Qt Designeru jedná pouze o vzhled uživatelského rozhraní, veškerá tlačítka jsou pouze znázorněna graficky bez jakýkoliv funkcionalit. V této podkapitole bude popsán způsob, jak se s designem vytvořeném v Qt Designeru pracuje v Pythonu.

Qt Designer ukládá návrhy do souborů s příponou „.ui.“ Tyto soubory jsou ve formátu XML, který popisuje vlastnosti a uspořádání prvků v okně aplikace. Soubory „.ui“ jsou nezávislé na platformě a mohou být použity pro různé verze Qt.

Soubory „.ui“ je možné použít v PyQt5 dvěma způsoby. První způsob je převést soubor „.ui“ na soubor „.py“ pomocí nástroje „pyuic5“. Tento nástroj převede XML kód na Python kód, který definuje třídu pro okno aplikace. Tuto třídu je možné importovat do hlavního skriptu a vytvořit instanci okna.

Druhý způsob je načíst soubor „.ui“ do třídy pomocí funkce „uic.loadUi“. Tato funkce načte XML kód a dynamicky vytvoří objekty pro prvky v okně aplikace. Tuto funkci je možné použít v konstruktoru třídy, která dědí od QWidget nebo QMainWindow.

Oba způsoby mají své výhody a nevýhody. Převod souboru „.ui“ na soubor .py umožňuje vidět a upravovat Python kód pro okno aplikace, ale vyžaduje znovu spustit nástroj pyuic5 při každé změně návrhu.

Pro účely vývoje dané desktopové aplikace byl ale vybrán druhý způsob načítání návrhu GUI pomocí funkce „uic.loadUi“. Tento způsob má několik výhod oproti převodu souboru „.ui“ na soubor „.py“ pomocí nástroje „pyuic5“.

Jednou z hlavních výhod je snadná aktualizace okna aplikace při změně návrhu GUI. To znamená, že pokud je nutné dělat grafické úpravy aplikace, nemusí se vůbec zasahovat do kódu, který definuje logiku aplikace. V případě použití nástroje pyuic5 by se pokaždé zdrojový Python kód měnil. To souvisí s další výhodou, kterou je zachování původního formátu souboru „.ui“. V případě práce přímo s generovanými „.ui“ soubory a s použitím funkce „uic.loadUi“, soubor „.ui“ zůstane nezměněn. Z toho vyplývá fakt, že je díky tomu kompletně oddělen zdrojový kód popisující logiku aplikace a zdrojový kód ve formátu „.ui“ popisující design.

Načítání „.ui“ souborů pomocí Pythonu a metody „uic.loadUi“ je znázorněno na obrázku (Obrázek 28 - Ukázka třídy „MainWindow“ a načtení .ui soboru do atributu třídy v konstruktoru) níže:

```
class MainWindow(QtWidgets.QMainWindow):
    def __init__(self) -> None:
        super().__init__()

        self.ui = uic.loadUi("UI_FILES/Main_Window.ui", self)
        self.ui.setWindowTitle("Screen Definition Editor")
```

Obrázek 28 - Ukázka třídy „MainWindow“ a načtení .ui soboru do atributu třídy v konstruktoru – zdroj vlastní

Metoda „uic.loadUi“ načte všechny vytvořené elementy (například tlačítka, widgety apod.) ze souboru vytvořeným z Qt Designeru a přiřadí je do proměnných instancí pod názvem tak, jak byly pojmenovány v Qt Designeru. Dalším způsobem, jak pracovat s předem vytvořenými prvky (například tlačítkům, widgetům atd) je pomocí metody findChild. Tato metoda umožňuje vyhledat objekt podle jeho typu a jména a lze pak tento prvek uložit do proměnné instance. Tento způsob není v případě použití metody „uic.loadUi“ nutné použít, protože jak již bylo řečeno výše, jednotlivé prvky se automaticky ukládají do proměnných instancí tříd. Autor této diplomové práce tak využívá funkcionalit „uic.loadUi“, tudíž metoda „findChild“ není využívána. Šetří tím operační paměť, protože se díky tomu nenačítá jeden prvek do paměti dvakrát (jednou pomocí metody „uic.loadUi“ a podruhé pomocí metody „findChild“).

Příklad použití lze metody „findChild“ lze však vidět na obrázcích (Obrázek 29 - Ukázka načtení atributu z .ui souboru) a (Obrázek 30 - Ukázka načtení atributu z .ui souboru).

```
def create_tab_view(self):
    self.tab_widget = self.parent.findChild(QTabWidget, "TabForScreens")
    self.tab_widget.clear()
    self.tab_widget.tabCloseRequested.connect(self.close_tab)
    self.tab_widget.tabBarClicked.connect(self.handle_tabbar_clicked)
```

Obrázek 29 - Ukázka načtení atributu z .ui souboru – zdroj vlastní

```
def new_screen_button(self):
    self.new_screen_button = self.parent.findChild(QPushButton, "NewScreen")
    self.new_screen_button.setVisible(False)
```

Obrázek 30 - Ukázka načtení atributu z .ui souboru – zdroj vlastní

Dané prvky se uloží do atributů třídy a lze s nimi pak pracovat v dalších funkcích. Každý prvek má v knihovně PyQt sadu metod, které se dají použít pro práci s nimi. Příkladem takové metody je použití „setVisible“ na obrázku (Obrázek 30 - Ukázka načtení atributu z .ui souboru), kdy se tlačítko při zavolání této funkce zneviditelní. To je napříč navrhovaným softwarem hojně využívané, aby se uživateli aplikace zobrazovala pouze ta pole, která jsou relevantní ke zrovna zvoleným prvkům.

4.8 Tvorba business logiky aplikace

V přechozích podkapitolách bylo popsáno, jakým způsobem bylo tvořeno grafické rozhraní vyvíjené aplikace. Bylo nastíněno, jak s tímto rozhraním následně pracovat pomocí Pythonu v rámci frameworku PyQt5 společně s praktickými ukázkami použití.

V této kapitole bude popsáno, jak byla řešena tvorba tzv. „business logiky“ vyvíjené aplikace.

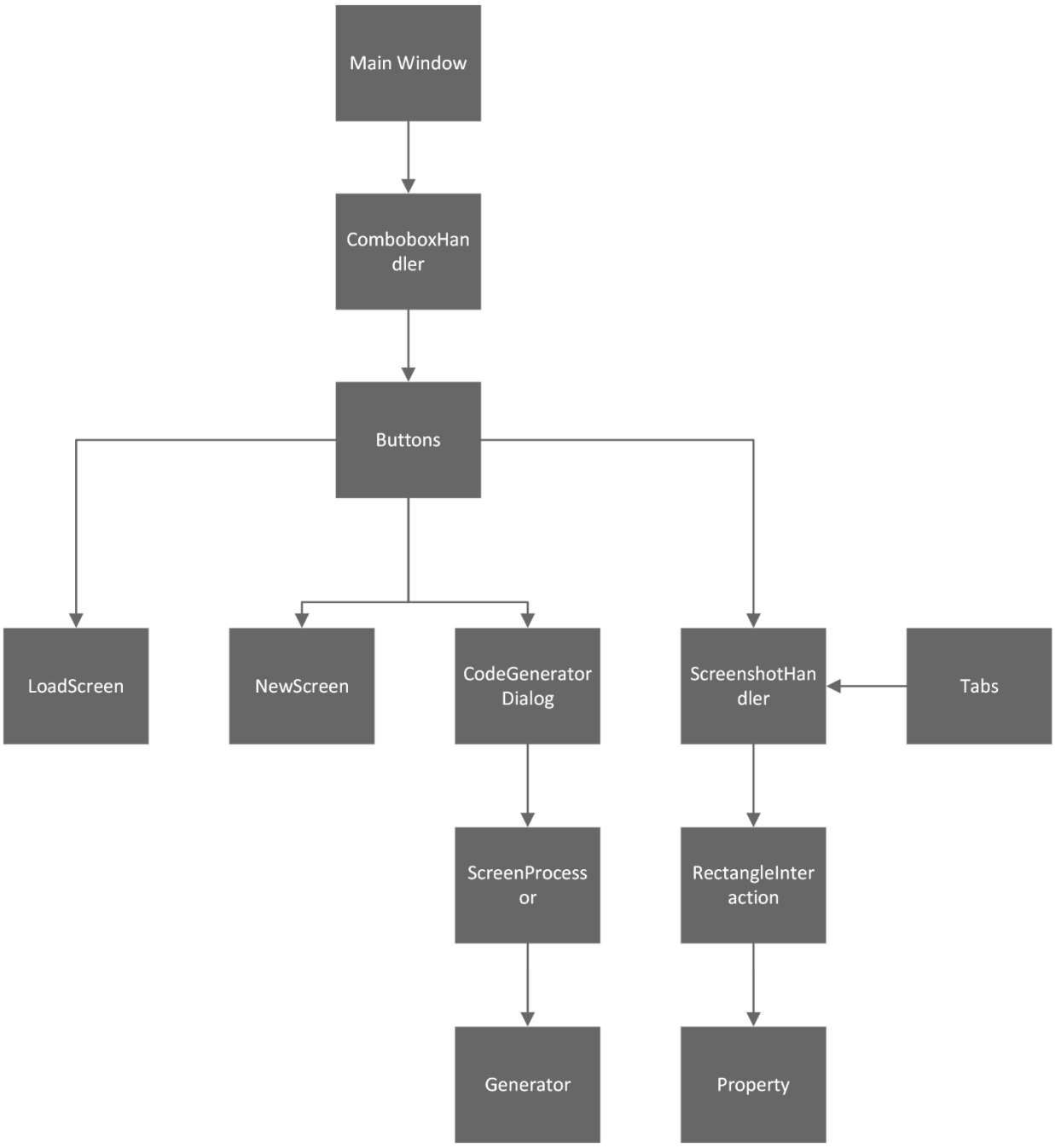
Kód, který řeší logiku aplikace, se někdy nazývá backend kód nebo business logika. Tento kód zajišťuje funkčnost aplikace, jako je zpracování dat, komunikace s databází, výpočty, validace atd. Business logika je, jak bylo řečeno v předchozí kapitole, oddělena od kódu, který řeší vzhled a interakci s uživatelem, který se nazývá frontend kód nebo uživatelské rozhraní.

4.8.1 Signály a sloty

Uživatelské rozhraní aplikací se ve frameworku PyQt propojují s business logikou pomocí tzv. signálů a slotů. Signály a sloty jsou základním mechanismem pro komunikaci mezi objekty v PyQt. Signál je zpráva, kterou objekt může vyslat, když se něco stane. Slot je funkce, která může být spuštěna, když je signál vyslán. Signály a sloty jsou propojeny pomocí metody *„connect()“*. Na obrázku Obrázek 29 - Ukázka načtení atributu z .ui souboru) výše lze vidět ukázku takového propojení, kdy se signál proměnné „tab_widget“ (v tomto případě se jedná o signál, který říká, že

uživatel stiskl tlačítko pro zavření záložky) propojuje se slotem, který reprezentuje metoda „close_tab“.

4.8.2 Funkční schéma business logiky



Obrázek 31 - Architektura business logiky aplikace – zdroj vlastní

Na obrázku (Obrázek 31 - Architektura business logiky aplikace) výše lze vidět navrženou architekturu zdrojového kódu vyvíjené aplikace. Jednotlivé obdélníky znázorňují názvy tříd, které jsou v softwaru použity.

Třída „MainWindow“ je nadřazená všem třídám a obsahuje pouze načtení hlavní okna a volá třídu „ComboboxHandler“, která má na starosti logiku comboboxu, ve kterém si uživatel volí jednotku, pro kterou chce zdrojová data generovat.

Třída „Buttons“ načítá z generovaných „.ui“ souborů všechna tlačítka, které se na hlavní obrazovce po zvolení jednotky zobrazují. V této třídě se následně vytvářejí instance tříd „LoadScreen“, „NewScreen“, „CodeGeneratorDialog“ a „ScreenshotHandler“, jsou propojeny s jednotlivými tlačítky.

```
def generate_code_button(self):
    self.parent.generate_code_button.clicked.connect(lambda: CodeGeneratorDialog(self.parent))
```

Obrázek 32 - Ukázka kódu funkce, která propojuje tlačítko na generování kódu s třídou "CodeGeneratorDIalog" – zdroj vlastní

Na obrázku (Obrázek 32 - Ukázka kódu funkce, která propojuje tlačítko na generování kódu s třídou "CodeGeneratorDIalog") lze vidět ukázku metody ve třídě „Buttons“, která má na starosti napárování tohoto tlačítka s třídou „CodeGeneratorDialog“. Pomocí této metody je docíleno toho, že po stisknutí tohoto tlačítka se provede nadefinovaná operace.

Ve třídách NewScreen“, „LoadScreen“ „a „GenerateCode“ je pak samotná implementace pro správné fungování dialogových oken, jejichž ukázky jsou na obrázcích výše (Obrázek 16 - Ukázka dialogového okna zobrazovaného po stisknutí tlačítka "New Screen", Obrázek 18 - Ukázka dialogového okna zobrazovaného po stisknutí tlačítka "Load Screen" a Obrázek 19 - Ukázka dialogového okna zobrazovaného po stisknutí tlačítka "Generate Code").

Třída „ScreenshotHandler“ dědí ze třídy Tabs a společně tyto třídy obsahují implementaci zobrazování obrázku v daném QTabWidgetu, dále také logiku vytváření obdélníků, které ohraničují prvky na zobrazované obrazovce a všechny s tímto spjaté funkcionality. Třída „RectangleInteration“ je pomocná třída uchovávající informace o kresleném obdélníku. Jak již bylo řečeno v předchozích kapitolách, obdélníky ohraničující prvky na obrazovce mají tyto prvky reprezentovat a uchovávat v sobě veškeré informace o tomto prvku. Třída „Property“ implementuje funkcionality dialogového okna na obrázku (Obrázek 25 - Ukázka dialogového okna uchovávající informace o prvku na obrazovce) v kapitole 4.6.4.

Třída „CodeGeneratorDialog“ implementuje funkcionality dialogového okna po stisknutí tlačítka „Generate Code“. Ukázka tohoto dialogového okna je na obrázku (Obrázek 19 - Ukázka dialogového okna zobrazovaného po stisknutí tlačítka "Generate Code") v kapitole 4.6.2.

V neposlední řadě třídy „ScreenProcessor“ a „Generator“ implementují funkcionality spojené s čtením vytvořených JSON souborů a následné generování zdrojového kódu v programovacím jazyce Python.

Všechny třídy, které mají na starosti implementaci funkcionalit dialogových oken, dále dědí z třídy „QDialog“, která je součástí frameworku PyQt5. Oproti tomu třída „MainWindow“, jak již název dané třídy napovídá, dědí ze třídy QMainWindow, která je opět implementována v rámci frameworku PyQt5.

Společně všechny tyto třídy obsahují implementaci veškeré business logiky v celé vyvíjené aplikaci.

4.9 Generování masek prvků

Jak bylo zmíněno v předchozích kapitolách, jedním z výstupů navrhované aplikace jsou soubory ve formátu XML, kterým se v kontextu automatického ovládání infotainmentu říká masky. Tyto masky budou generovány automaticky v případě, že uživatel v dialogovém okně, které je znázorněno na obrázku (Obrázek 25 - Ukázka dialogového okna uchovávající informace o prvku na obrazovce) výše, zvolí možnost „Create new mask“. Aplikace nejprve provede normalizaci těchto souřadnic, a poté na základě předem definované struktury tyto souřadnice uloží jako samostatný XML soubor reprezentující masku daného prvku. K tvorbě takového XML souboru autor této diplomové práce využil knihovnu „xml“, která je součástí standartní instalace v Pythonu.

4.10 Generování ikon

Jedním z dalších výstupů, který bude aplikace schopna generovat jsou ikony prvků zobrazené na obrazovce. V případě, že se bude na zkoumané obrazovce vyskytovat prvek, pro který ještě ve firemním úložišti neexistuje příslušný „.png“ soubor, bude si moci uživatel takový soubor vytvořit. Způsob, jakým se takový soubor vytvoří byl popsán v kapitole (4.6.4) výše. K tomu, aby uživatel mohl výstřižek pořídit byl využita metoda „pixmap“, kterou nabízí framework PyQt5.

4.11 Generování zdrojového kódu

Samotné generování zdrojového kódu bude zpracováváno pomocí modulu, který bude pracovat s regulárními výrazy.

„Regulární výraz je textový řetězec, který slouží jako vzor pro vyhledávání textu. Regulární výraz tvoří soubor dvou typů znaků – literálů, které musí hledaný řetězec obsahovat, a pomocných znaků, které umožňují pokročilé možnosti vyhledávání. Například pokud chce uživatel v textu vyhledat řetězec, který nezná přesně nebo který může mít více variant, umožňuje mu použití regulárního výrazu popsat podmínky vyhledávání („regule“), které musí hledaný řetězec splňovat. Regulární výraz se zapisuje regulárním jazykem, který zavedl americký matematik Stephen Cole Kleene.“

[20]

Tyto regulární výrazy budou představovat jakousi šablonu zdrojového kódu, který bude následně generován. Příklad takového regulárního výrazu je zobrazen na obrázku (Obrázek 33 - Regulární výraz pro metodu TextTag) níže, kdy tento regulární výraz je uložen do proměnné „text_tag“ jako datový typ string. Veškeré prvky, které jsou ohraničeny tagy (<>) budou později nahrazovány podle toho, co do daných polí vyplnil uživatel v rámci práce s aplikací. Nahrazování se provádí pomocí knihovny „re“, která je standardní a je součástí instalace Pythonu. Tato knihovna obsahuje funkce pro práci s regulárními výrazy. Konkrétně pak metoda „sub“, která prohledává text, a slouží k nahrazování textu v řetězci na základě zadaných pravidel. Ukázka jednoduchého použití je vidět na obrázku (Obrázek 34 - Ukázka použití knihovny re a metody sub) níže.

```
def generate_text_tag(self, dictionary, identification_or_element):
    text_tag: str = """TextTag(text="<<text>>", method="<<method>>", mask="<<mask>>", threshold=<<threshold>>)"""
```

Obrázek 33 - Regulární výraz pro metodu TextTag – zdroj vlastní

```
1  import re
2
3  text = "Hello, World!"
4  new_text = re.sub("World", "Python", text)
5
6  print(new_text) # Hello, Python!
```

Obrázek 34 - Ukázka použití knihovny re a metody sub – zdroj vlastní

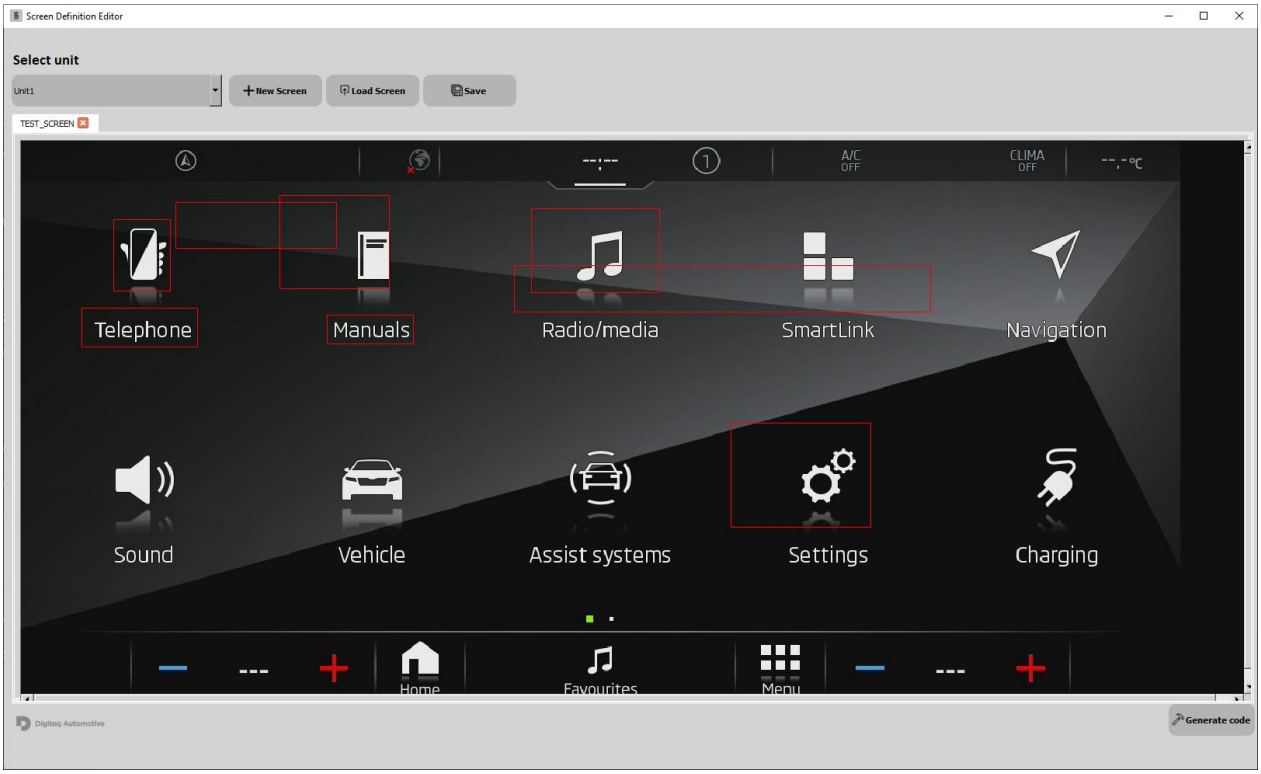
5 Vyhodnocení

V kapitole 4 byl podrobně popsán návrh a tvorba desktopové aplikace. Po dokončení všech částí byl celý návrh otestován na testovacích vstupech a datech, kdy cílem bylo otestovat všechny možné kombinace a zkontrolovat, zda aplikace vygeneruje zdrojový kód tak, jak má. Tento test proběhl v pořádku a automatizační nástroj, který má na starosti automatické ovládání infotainmentu s těmito daty dokázal pracovat. Vzhledem k tomu, že tento nástroj je majetkem firmy Digiteq Automotive s.r.o., není možné v rámci této diplomové práce ukazovat podrobnější chování tohoto nástroje.

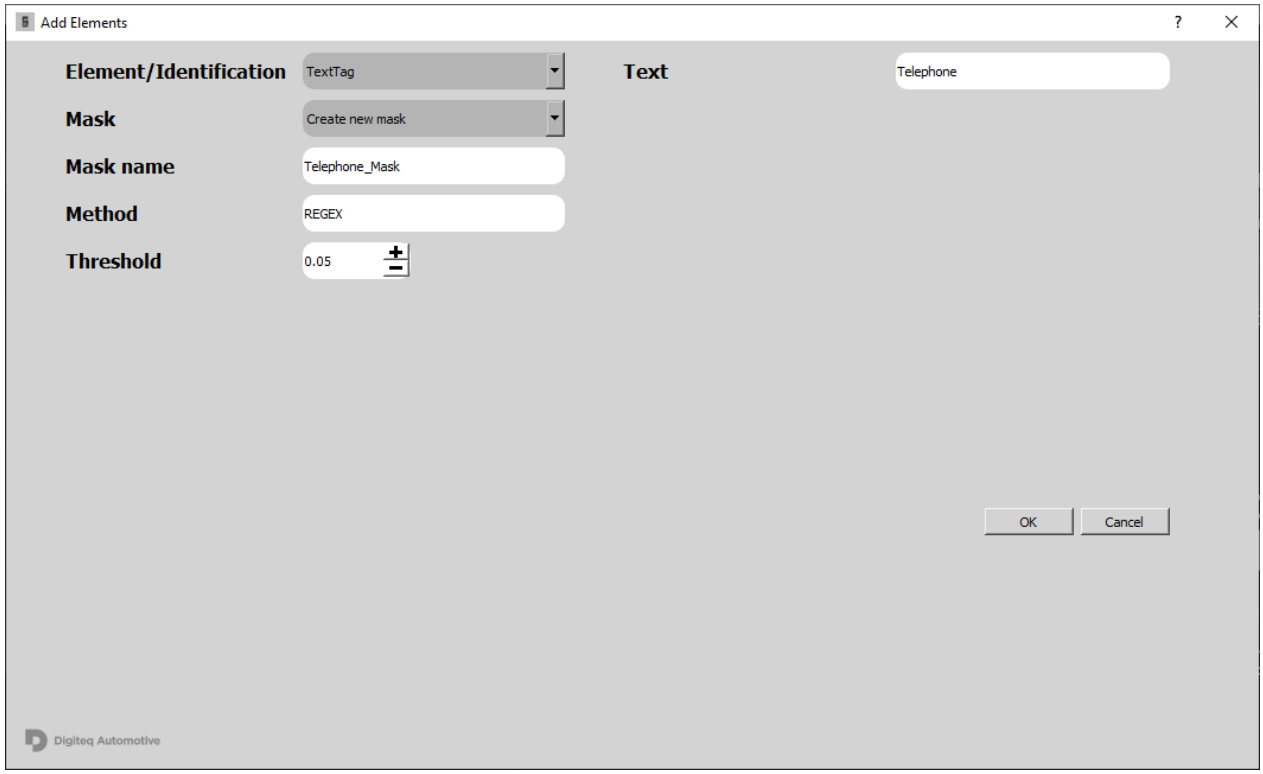
Aplikaci však lze otestovat na datech testovacích (imaginárních), a pomocí toho alespoň ukázat vygenerovaný zdrojový kód. Po naplnění aplikace různými testovacími vstupy byl vygenerován tento zdrojový kód:

```
TEST_SCREEN = Screen("TEST_SCREEN") \
    .add_identification(
        TextTag(text="Telephone", method="REGEX", mask="Telephone_Mask",
threshold=0.05),
        IconTag(mask="Telephone_Mask", icon="telephone.png"),
        TextToolTag(threshold=0.05, mask="Manuals_Mask", texttool="en_US.xml"),
        PlaceHolderTag(mask="acc", key="SearchedDeviceName",
preprocessing_mode="NUMBERS")) \
    .add_element(Button(name="Radio/Media",
identification_tag=IconTag(mask="Radio_Media_Button", icon="Media_Icon.png")),
"Radio/Media"
        ) \
    .add_element(SetBox(name="Smart", identification_tag=IconTag(mask="active_track",
icon="smartlink.png")), "SmartlinkScreen"
        ) \
    .add_element(CheckBox.create(name="Settings",
element_identification_tag=TextTag(text="SETTINGS_CHCECKBOX", method="REGEX",
mask="SETTINGS_CHECKBOX", threshold=0.01))
        , false_identification_tag="_BROWSER_SHUFFLE_OFF_TAG",
true_identification_tag="_BROWSER_SHUFFLE_OFF_TAG", threshold=0.01) \
    .add_element(ParamCircular.create(name="Charging",
element_identification_tag=PlaceHolderTag(mask="android_title",
key="SearchedDeviceName", preprocessing_mode="NUMBERS"))
        , values=['off', 'track', 'all'], value_identificators=['repeat_off',
'repeat_on'], threshold=0.02)
```

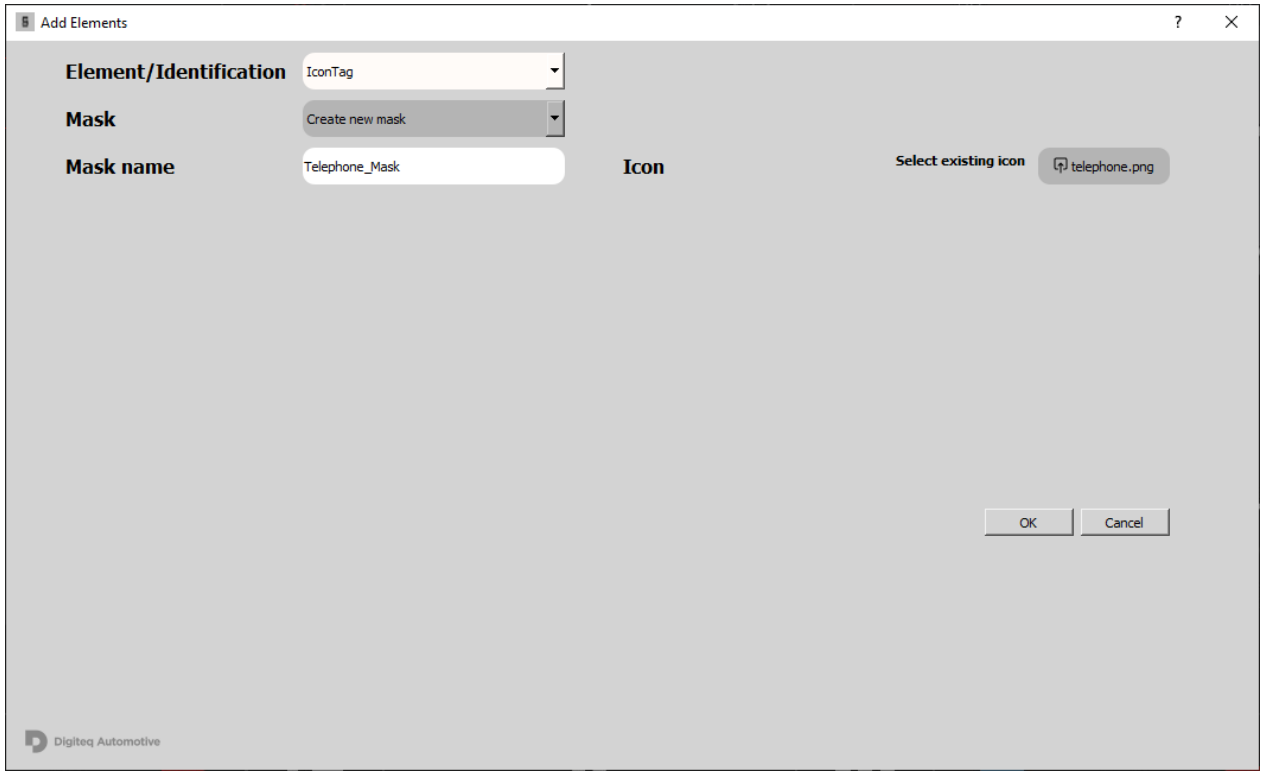
V ukázce výše lze vidět zdrojový kód, který byl vygenerován pomocí aplikace po předání různých testovacích vstupů. Na obrázcích níže si lze ověřit, že výstupy, které byly aplikací vygenerovány, odpovídají vstupům a jsou správné. Pro ověření si lze všimnout textových vstupů či textů zobrazovaných v jednotlivých tlačítkách a combo boxech a porovnat je s parametry ve zdrojovém kódu. Na obrázku níže (Obrázek 36 - Dialogové okno po vytvoření obdélníku s testovacími vstupy) si lze může všimnout, že v combo boxu Element/Identification byl zvolen „TextTag“ s parametry „Mask name“ = „Telephone_Mask“, „Method“ = „REGEX“, „Threshold“ = 0.05 a „Text“ = „Telephone“. Tyto parametry jsou pak předány do vygenerovaného zdrojového kódu, kde si jich lze všimnout od druhého řádku ukázkového zdrojového kódu.



Obrázek 35 - Aplikace s testovacími vstupy – zdroj vlastní



Obrázek 36 - Dialogové okno po vytvoření obdélníku s testovacími vstupy – zdroj vlastní



Obrázek 37 - Dialogové okno č.2 po vytvoření obdélníku s testovacími vstupy – zdroj vlastní

6 Závěr

Cílem této diplomové práce bylo navrhnout a vytvořit nástroj pro vytváření dat určených k automatickému ovládání infotainmentu, který bude sloužit jako pomocník vývojářům automatického testování funkcí infotainmentu ve firmě Digiteq Automotive s.r.o.

V teoretické části této diplomové práce bylo podrobně představeno cílové aplikační prostředí, v němž je navržený nástroj používán. Byly zde popsány metody testování, včetně podrobného vysvětlení automatizovaného a manuálního testování, jejich výhod a nevýhod a také jejich uplatnění v cílovém aplikačním prostředí navrženého nástroje. Dále bylo podrobně popsáno, jak bude navrhovaná aplikace využívána a jaký přínos bude mít. V závěru teoretické části byla provedena stručná analýza a srovnání různých nástrojů a knihoven na tvorbu grafických uživatelských rozhraní. Na základě této analýzy byl pro tvorbu GUI vybrán nástroj Qt v nadstavbové podobě pro programovací jazyk Python, které se říká PyQt.

V praktické části byl především důkladně popsán postup návrhu této aplikace, a to jak po stránce vývoje grafického uživatelského rozhraní, vzhledu aplikace, tak zde byla popsána i business logika. Dále byl kladen důraz na popis vstupních a výstupních dat, kde bylo vytvořeno jednoduché schéma, pomocí kterého si může čtenář udělat ještě lepší představu, k čemu se navrhovaná aplikace využívá. V neposlední řadě byl popsán způsob, kterým autor přistoupil ke generování všech výstupů, tedy zdrojového kódu, XML souborů v podobě masek a ikon v podobě obrázků.

Závěrem lze říct, že se autorovi podařilo naplnit předem stanovené cíle. Díky tomu, že se diplomová práce věnovala tématu, které vzniklo v pracovním prostředí firmy Digiteq Automotive s.r.o. (konkrétně v oddělení testování funkcí), navrhované řešení má uplatnění v praxi. Hlavním přínosem navrženého nástroje je zefektivnění práce vývojářů automatizovaných testů. Výhodou také je, že byl nástroj vytvořen pomocí nástrojů, které jsou dalším kolegům v rámci oddělení blízké, tudíž do budoucna nebude problém s údržbou či dalším rozšiřování aplikace.

7 Zdroje

[1] In-car entertainment. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-05-01]. Dostupné z: https://en.wikipedia.org/wiki/In-car_entertainment#cite_note-1

[2] Test Automation: Delivering Business Value. AppLabs White Paper [online]. 2008, 2-4 [cit. 2023-05-01]. Dostupné z: https://web.archive.org/web/20100106191031/http://www.applabs.com/internal/app_whitepaper_test_automation_delivering_business_value_1v00.pdf

[3] Software testing in continuous delivery. ATLASSIAN [online]. [cit. 2023-05-01]. Dostupné z: https://www.atlassian.com/continuous-delivery/software-testing

[4] Hardware Testing Process – How to test products during production. VIEWPOINT SYSTEMS [online]. [cit. 2023-05-01]. Dostupné z: https://www.viewpointusa.com/TM/ar/hardware-testing-process/

[5] 7 Fundamental Steps for Testing (with) Hardware. HTEC GROUP [online]. [cit. 2023-05-01]. Dostupné z: https://htecgroup.com/insights/all/7-fundamental-steps-for-testing-with-hardware/

[6] HARRIS, Sarah L. a David HARRIS. Digital Design and Computer Architecture. 2021. ISBN 978-0-12-820064-3.

[7] Test bench. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-05-01]. Dostupné z: https://en.wikipedia.org/wiki/Test_bench

[8] ELLIS, George. Control System Design Guide. 4. 2012. ISBN 978-0-12-385920-4.

[9] About Qt [online]. [cit. 2023-05-01]. Dostupné z: https://wiki.qt.io/About_Qt

[10] WxWidgets Overview [online]. [cit. 2023-05-01]. Dostupné z: https://www.wxwidgets.org/about/

[11] Lekce 1 - Úvod do JavaFX. Itnetwork.cz [online]. [cit. 2023-05-01]. Dostupné z: https://www.itnetwork.cz/java/javafx/uvod-do-javafx

[12] Electron [online]. [cit. 2023-05-01]. Dostupné z: https://www.electronjs.org/docs/latest/

[13] Desktop Guide (Windows Forms .NET). Microsoft [online]. [cit. 2023-05-01]. Dostupné z: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-7.0

[14] Graphical User Interfaces with Tk. Python [online]. [cit. 2023-05-01]. Dostupné z: https://docs.python.org/3/library/tk.html

[15] Swing (Java). In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-05-01]. Dostupné z: https://cs.wikipedia.org/wiki/Swing_(Java)

[16] Python. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-05-01]. Dostupné z: https://cs.wikipedia.org/wiki/Python

[17] PyQt5 5.15.9 [online]. [cit. 2023-05-01]. Dostupné z: https://pypi.org/project/PyQt5/

[18] Úvod do JSON [online]. [cit. 2023-05-01]. Dostupné z: https://www.json.org/json-cz.html

[19] XML – Overview. Tutorialspoint [online]. [cit. 2023-05-01]. Dostupné z: https://www.tutorialspoint.com/xml/xml_overview.htm

[20] Regulární výraz. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-05-01]. Dostupné z: https://cs.wikipedia.org/wiki/Regul%C3%A1rn%C3%AD_v%C3%BDraz#:~:text=Regul%C3%A1rn%C3%AD%20v%C3%BDraz%20(zkratky%20regexp%2C%20regex,kter%C3%A9%20umo%C5%BE%C5%88uj%C3%AD%20pokro%C4%8Dil%C3%A9%20mo%C5%BEnosti%20vyhled%C3%A1v%C3%A1n%C3%AD

8 Seznam tabulek

9 Seznam obrázků

10 Seznam příloh

Příloha č. 1 – Zdrojový kód programu