
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE


# NÁSTROJ PRO USNADNĚNÍ TVORBY AUTOMATIZAČNÍCH VIZUALIZACÍ TECHNOLOGIÍ MAPP VIEW

THE SAME IN ENGLISH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S


AUTOR PRÁCE — Ján Kseňak
AUTHOR

VEDOUCÍ PRÁCE — Ing. Radek Poliščuk, Ph.D.
SUPERVISOR



BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

# Zadání bakalářské práce

| | |
|---|---|
| Ústav: | Ústav automatizace a informatiky |
| Student: | **Ján Kseňak** |
| Studijní program: | Strojírenství |
| Studijní obor: | Aplikovaná informatika a řízení |
| Vedoucí práce: | **Ing. Radek Poliščuk, Ph.D.** |
| Akademický rok: | 2021/22 |

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

## Nástroj pro usnadnění tvorby automatizačních vizualizací technologií mapp View

**Stručná charakteristika problematiky úkolu:**

Trend moderních automatizačních vizualizací s využitím responzivních WWW technologií dnes vyžaduje komplexní znalosti z různých vývojářských oborů, od samotné procesní technologie, přes programátorské postupy specifické pro dané vývojové prostředí, až po znalost technologií OPC–UA, XML a tvorby uživatelsky efektivních WWW stránek. Tento vývoj dnes představuje zásadní výzvu nejen pro samotné programátory, ale i pro tvůrce samotných vývojových prostředí pro průmyslovou automatizaci, aby klientům dokázali poskytnout řešení, které by tento složitý a mnohastupňový proces dokázaly alespoň v některých ohledech zjednodušit a zpřehlednit. Jedním z kroků v tomto procesu by mohl být i návrh vizuálního nástroj pro usnadnění tvorby vizualizací pro technologii mapp View, určený pro B&R Automation Studio řady 4.

**Cíle bakalářské práce:**

Rešerše dostupných podkladů k postupům a šablonám používaným technologií mapp View, výběr vhodné programovací technologie a vytvoření uživatelsky přívětivého nástroje pro tvorbu XML kódu pro uživatelské vizualizace, využitelné v prostředí B&R Automation Studio.

**Seznam doporučené literatury:**

HOLUBOVÁ, Irena, POKORNÝ, Jaroslav: XML technologie: principy a aplikace v praxi. Praha: Grada, 2008. Průvodce (Grada). ISBN 978-80-247-2725-7.

B&R Automation Academy: TM611 - Working with mapp View [online]. Dostupné z: https://www.br-automation.com/cs/soubory-ke-stazeni/automation-academy/training-modules/visualization-and-operation/mapp-view/tm611-working-with-mapp-view/

RINALDI, John: OPC UA - Unified Architecture: The Everyman's Guide to the Most Important Information Technology in Industrial Automation, Createspace Independent Publishing Platform, 2016. ISBN 978-15-305-0511-1.

WALMSLEY, Priscilla. Definitive XML Schema. 2nd ed. Upper Saddle River, N.J.: Prentice Hall, c2013. Charles F. Goldfarb definitive XML series. ISBN 978-0-13-288672-7.

PECINOVSKÝ, Rudolf. Začínáme programovat v jazyku Python. Praha: Grada Publishing, 2020. Začínáme s.. ISBN 978-80-271-1237-1.

BANZAL, S. XML basics. Duxbury: Mercury Learning and Information, 2020. ISBN 9781683925460.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2021/22

V Brně, dne

L. S.

| | |
|---|---|
| doc. Ing. Radomil Matoušek, Ph.D.<br>ředitel ústavu | doc. Ing. Jaroslav Katolický, Ph.D.<br>děkan fakulty |

# ABSTRAKT

Cieľom bakalárskej práce je navrhnúť program, ktorý zjednoduší vývoj automatizačných vizualizácií pomocou technológie mappView vo vývojom prostredí B&R Automation studio. Teoretická časť práce sa venuje bližšiemu popisu technológie mappView a jeho neoddeliteľných súčastí. V úvode praktickej časti je spomenutý výber technológií, ktoré sa použili pri vývoji aplikácie, ktorá ma zjednodušiť vývoj mappView vizualizácií. Ďalej sa práca v empirickej časti venuje popisu architektúry aplikácie, procesu tvorby aplikácie, návrhu grafického užívateľského rozhrania aplikácie a integrácií aplikácie do vývojového prostredia B&R Automation studio. V poslednej kapitole je znázornený celý proces vývoja mappView vizualizácií vo vývojom prostredí B&R Automation studio a využitie našej aplikácie v tomto procese.

# ABSTRACT

The aim of the bachelor thesis is create a program that will simplify development of automation visualizations using mappView technology in the development environment B&R Automation studio. The theoretical part of thesis contains more detailed description of mappView technology and its necessary components. At the beginning of the practical part is mentioned a selection of technologies that were used in development of the application, which should simplify development of mappView visualizations. Furthermore, in the practical part, we describe the architecture of application, the process of creating application, the design of a graphical user interface of application and the integration of application in the development environment B&R Automation studio. The last chapter shows the whole process of developing mappView visualizations in the development environment B&R Automation studio and the use of application in this process.



# KLÚČOVÉ SLOVÁ

Automatizačné vizualizácie, HMI aplikácie, B+R Automatizace, Automation Studio, mappView, programovací jazyk Python, Xml kód

# KEYWORDS

Automation visualization, HMI application, B&R Automation, Automation Studio, mappView, programming language Python, Xml code

ÚSTAV AUTOMATIZACE A INFORMATIKY

2022

# BIBLIOGRAFICKÁ CITÁCIA

KSEŇAK, Ján. *Nástroj pro usnadnění tvorby automatizačních vizualizací technologií mapp View*. Brno, 2022. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/140167. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního nženýrství, Ústav automatizace a informatiky. Vedoucí práce Radek Poliščuk.

# POĎAKOVANIE

Rád by som poďakoval vedúcemu tejto bakalárskej práce, pánovi Dr. Radkovi Poliščukovi a Ing. Ľubomírovi Bubeníkovi zo spoločnosti B+R Automatizace, za ich odborné rady, priateľský prístup a hodnotné pripomienky k tejto bakalárskej práci.

# ČESTNÉ PREHLÁSENIE

Prehlasujem, že táto práca je mojím pôvodným dielom, vypracoval som ju samostatne pod vedením vedúceho práce a s použitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry.

Ako autor uvedenej práce danej prehlasujem, že v súvislosti s vytvorením tejto práce som neporušil žiadne autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích osobnostných práv a som si plne vedomý následkov porušenia ustanovenia § 11a nasledujúcich autorského zákona c. 121/2000 Sb., vrátane možných trestne právnych následkov.

V Brne dne 20. 5. 2022 .....

Ján Kseňak

# OBSAH

# 1 ÚVOD

Cieľom práce je navrhnúť aplikácie pre zjednodušenie vývoja automatizačných vizualizácií pomocou technológie mappView vo vývojom prostredí B&R Automation Studio. Návrh aplikácie spočíva vo výbere vhodných technológií na vývoj aplikácie, navrhnúť architektúru aplikácie a vytvoriť grafické užívateľské rozhranie pre aplikáciu. Výsledkom práce by mala aplikácia, ktorá zjednoduší procese tvorby mappView vizualizácií technológie a je plne kompatibilná s vývojovým prostredím B&R Automation Studio.

Prvá, teoretická časť práce, sa zaoberá rešeršou technológie mappView od spoločností B&R Industrial Automation. MappView je technológia určená na tvorbu automatizačných vizualizácií v prostredí B&R Automation Studio. Architektúra mappView je postavená na webových technológiách ako napríklad HTML, CSS a Javascript a zároveň poskytuje automatizačným inžinierom nástroje na tvorbu efektívnych, intuitívnych a vďaka webovým technológiám ľahko a všade prístupným HMI aplikáciám.

V praktickej časti sa opierame o vhodný výber technológií na vývoj aplikácie, ktorá ma zjednodušiť vývoj mappView vizualizácií. Nasleduje návrh vhodnej architektúry aplikácie a popis jej tvorby. Architektúra navrhnutej aplikácie pozostáva z troch častí. Z konfiguračného súboru aplikácie, z grafického rozhrania a z backendu aplikácie. Grafické rozhranie slúži na manipulácia s dátami, ktoré sa nachádzajú v konfiguračnom súbore ako, aké stránky vizualizácie má naša aplikácia vygenerovať. Backend aplikácie spracováva dáta z konfiguračného súboru. Bližší popis aplikácie a jeho časti sa nachádza v príslušných kapitolách.

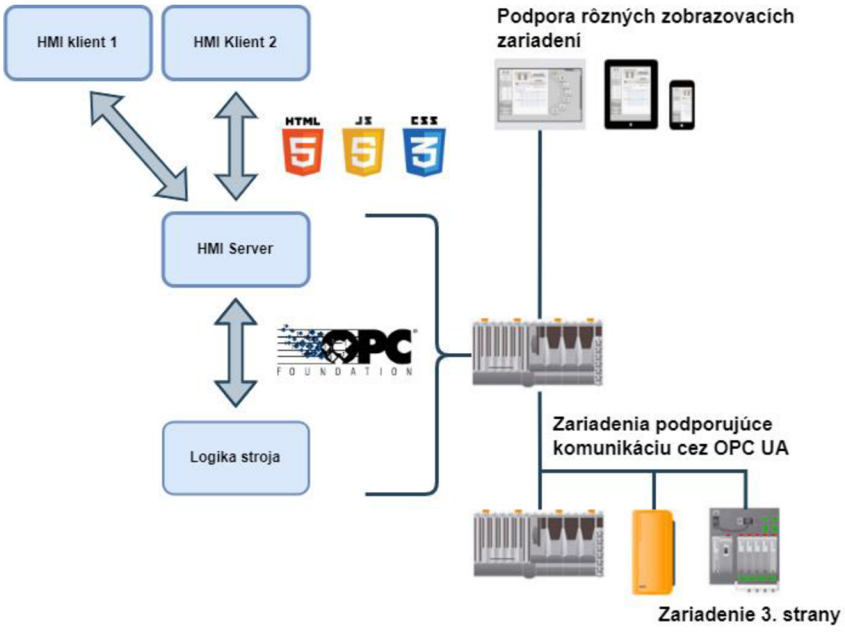
V poslednej kapitole praktickej časti bakalárskej práce je čitateľovi práce priblížený proces tvorby webovej vizualizácie pomocou technológie mappView a ukázane využitie našej aplikácie v tomto procese.

# 2 VÝVOJ HMI APLIKÁCIÍ V PROSTREDÍ MAPPVIEW B&R AUTOMATION STUDIO

MappView je integrované prostredie pre vývoj HMI aplikácií a tvorbu užívateľských rozhraní pre stroje. Toto prostredie sa nachádza v už existujúcom vývojovom prostredí B&R Automation studio. MappView ma modulárnu štruktúru, ktorá dovoľuje sledovanie a prevádzkovanie technologických procesov. V mappView je obsah vizualizácie kompletne oddelený od logiky riadenia stroja. Táto modularita ma tiež výhodu, že jednotlivé prvky vizualizácie môžeme upravovať separátne a taktiež tieto prvky využívať opakovane na rôznych častiach vizualizácie, čo výrazne redukuje čas na vývoj takejto aplikácie. MappView je postavené na webových technológiách HTML, CSS a Javascript, ale poskytuje automatizačným inžinierom nástroje na tvorbu efektívnych, intuitívnych a vďaka webovým technológiám ľahko prístupných HMI aplikácií.

**Architektúra mappView**

MappView architektúra je navrhnutá ako multi-klient/server a multi-user model. Táto architektúra pozostáva z mappView servera a jedného alebo viacerých mapView klientov. MappView klientom môže byť akékoľvek zariadenie, na ktorom funguje internetový prehliadač (mobil, tablet, počítač atď.), čo výrazne zvyšuje prístupnosť ku vizualizácií stroja a dovoľuje zobrazovať vizualizácie na viacerých zariadeniach simultánne. Stáva sa, že rôzni užívatelia vyžadujú personalizovaný obsah stránok. MappView na to využíva role, ktoré jasne definujú práva jednotlivých užívateľov. Týmto môže užívateľ vizualizácie jednotlivé prvky rozhrania konfigurovať pre práva konkrétneho užívateľa .

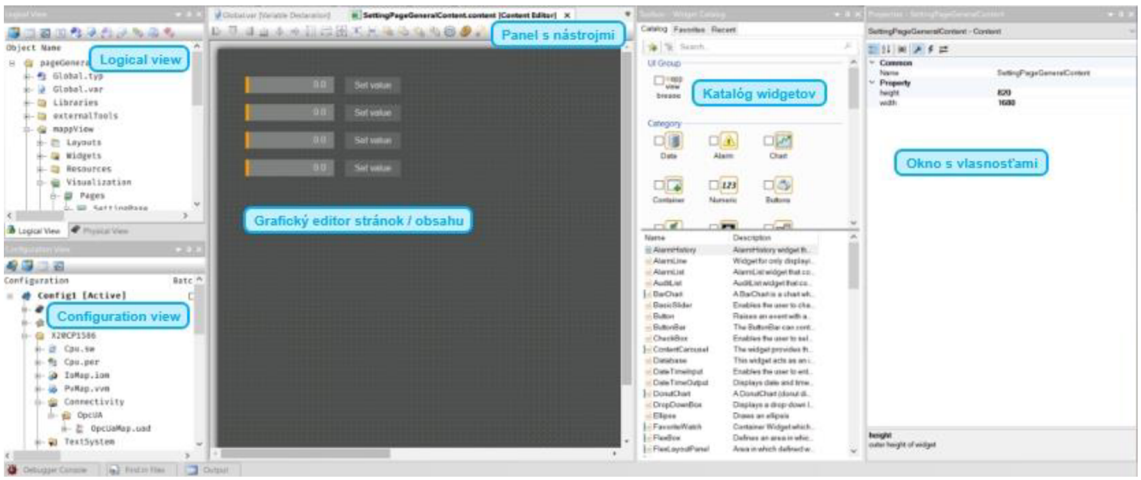



Obr. 1: MappView architektúra [1]

Prenos dát v mappView je založený na protokole OPC UA. OPC Unified Architecture je štandard pre schopnosť vzájomnej spolupráce medzi rôznymi zariadeniami, ktorý poskytuje spoľahlivú a bezpečnú výmenu dát v priemyselnej automatizácií a v iných odvetviach. Tento štandard je platformovo nezávislý, preto sa využíva na komunikáciu medzi zariadeniami od rôznych výrobcov. Jednotlivé dáta reprezentuje OPC UA uzol. Ide o procesnú premennú automatizačnej aplikácie, ktorá je dostupná cez OPC UA systém. Túto premennú môžeme zapisovať alebo čítať z OPC UA klientov ako OPC UA uzol. Tento uzol môže obsahovať dodatočné informácie o fyzikálnom rozmere sledovanej veličiny a jej interval povolených hodnôt. [1]

## 2.1 Organizácia mappView HMI aplikácie

Prostredie pre vývoj mappView HMI aplikácií je súčasťou softwaru B&R Automation studio, ktorý je určený na vývoj automatizačného softwaru od spoločnosti B&R Industrial Automation (ďalej iba AS). Jednotlivé zdrojové súbory aplikácie sa vytvárajú a upravujú v sekcii *Logical View* v prostredí AS. V *Configuration View* sa následne zostaví výsledná konfigurácia jednej alebo viacerých vizualizácií. Týmto rozdelením sa docieli rozdelenie zdrojových súborov a logiky aplikácie od výslednej konfigurácie stroja.




Obr. 2: Organizácia vývojového prostredia.

***Logical View*** - táto časť prostredia reprezentuje stromovú štruktúru prvkov automatizačného softwaru. Knižnica mappView sa nachádza v zložke s rovnakým názvom a obsahuje zložky s jednotlivými vizualizáciami. Štruktúra súborového systému rozdeľuje súbory na dve logické skupiny. Prvá skupina je globálna. Tieto súbory sa nachádzajú priamo v koreňovom adresári mappView a prístup k týmto súborom má každá vizualizácia. Druhá skupina je lokálna pre každú vizualizáciu.

***Configuration View*** - v tejto časti prostredia prebieha úprava výsledných konfigurácií jednotlivých častí automatizačného softwaru, ako napríklad rotačné osi, alarmy, ale aj konfigurácie vizualizácií a mappView servera.

**Grafický editor obsahu/stránok** - grafický editor zjednodušuje úpravu niektorých súborov (stránky, obsahy, rozloženia) a umožňuje zobrazenie vizuálnej podoby týchto súborov. Pred grafickým editorom sa súbory upravovali priamym zásahom xml kódu v textovom editore, čo pri komplikovaných xml štruktúrach bolo veľmi nepriehľadné.
**Panel s nástrojmi** – obsahuje užitočné nástroje (zarovnanie, posun po ose z, atď.) na pracovanie s widgetmi počas vývoja aplikácie.
**Katalóg widgetov** – Ak je otvorený grafický editor na úpravu obsahu, poskytuje zoznam všetkých widgetov, ktoré môžeme použiť na tvorbu aplikácie. Pomocou funkcie pretiahnutie (*drag-and-drop*) sa dajú jednotlivé widgety pridať do grafického editoru.
**Okno s vlastnosťami** – okno obsahuje zoznam vlastností konkrétneho prvku HMI aplikácie, ktoré môžeme upravovať podľa potrieb. [1]

## 2.2 Tvorba stránok

MappView HMI aplikácia zvyčajne pozostáva z viacerých stránok. Stránka je viditeľná časť HMI aplikácie, ktorá sa má zobraziť na HMI klientovi. Pre rýchlu identifikáciu sa každej stránke priradí ID v rámci danej vizualizácie. Stránka je z estetických dôvodov rozdelená do niekoľkých oblastí (*area*). Kvôli návrhu stránky je nevyhnutné definovať tieto oblasti na stránke a ich rozloženie. To sa robí pridelením existujúceho rozloženia, ktoré obsahuje požadované rozmiestnenie oblasti a ich rozmery. Každej vytvorenej oblasti je následne priradený obsah, ktorý sa má zobrazovať na danej stránke.

Súbory so stránkami majú príponu *.page* a obsahujú príslušnú xml štruktúru. Súbory so stránkami sa nachádzajú v zložke s rovnakým názvom, táto zložka môže tiež obsahovať súbory s obsahmi, ktoré sú priradené danej stránke.

**Rozloženie stránky**
Rozloženie stránky (*Layout*) je rozdelené do nami definovaných oblastí, ktoré tvoria výslednú štruktúru stránky. Rozloženie stránky je definované svojimi rozmermi a jedinečným ID. Šírka (*width*) a výška (*height*) je udávaná v pixeloch. Pomocou *layoutId* priradíme stránke požadovanú štruktúru oblastí. Každá oblasť má definovanú šírku a výšku. Poloha konkrétnej oblasti je určená atribútmi *top* a *left*, ktoré určujú polohu od horného ľavého rohu v pixeloch. Takáto vytvorená oblasť má taktiež unikátne ID, ktoré neskôr slúži na priradenie požadovaného obsahu.

**Obsah stránky**
Obsah je viditeľná časť stránky. Do jednotlivých obsahov môžeme vkladať widgety z katalógu a následne ich umiestniť na požadovanú pozícií a upraviť podľa potrieb aplikácie. Widget je prvok (tlačidlo, numerický vstup atď. ), ktorý poskytuje interakciu medzi operátorom a HMI aplikáciou. Každý obsah má definovaný výšku a šírku v pixeloch. Taktiež musí mať priradené unikátne ID, ktoré slúži na rýchlu identifikáciu obsahu. Súbory, ktoré obsahujú xml štruktúru obsahu majú príponu *.content* a nachádzajú sa v priečinku stránky, ku ktorej je obsah priradený.

V mappView lokalizácia znamená prispôsobenie sa textového obsahu HMI aplikácie ku lokálnym jazykom a lokálnym normám. MappView dovoľuje tieto zmeny textov a fyzikálnych jednotiek prijať aj počas bežiaceho programu pomocou widgetu *LangugageSelector*. [1]

**Textový systém**

B&R Automation Studio disponuje textovým systémom na lokalizáciu textov. V AS je textový systém rozdelený na dve časti. Prvá časť je jazyková konfigurácia, ktorá definuje jazyky, ktoré sa budú v projekte používať. Táto konfigurácia sa nachádza v sekcii *Logical View* v súbore s príponou *.language*. Ak takýto súbor neexistuje v projekte vytvoríme ho prostredníctvom katalógu objektov.

Zo skupiny zvolených jazykov je možné vytvoriť výslednú konfiguráciu jazykov, ktorá sa prenesie na cieľové zariadenie. Táto jazyková konfigurácia sa nachádza v súbore s príponou *.textconfig* v sekcii *Configuration View* v zložke *TextSystem*. Tento súbor obsahuje zoznam všetkých textových lokalizačných súborov, ktoré využíva daná konfigurácia stroja. Taktiež je tam nakonfigurovaný systémový jazyk (*System language*) a záložný jazyk (*Fallback language*). Systémový jazyk je predvolený jazyk a slúži na zobrazovanie textov v konkrétnom jazyku. Záložný jazyk sa používa v prípade, ak neexistuje žiaden text v systémovom jazyku.

Druhá časť pozostáva z jednotlivých lokalizačných súborov s príponou *.tmx*. Tieto súbory si môžeme predstaviť ako dvojrozmernú tabuľku, kde každý riadok tabuľky obsahuje sémantický význam s textami v príslušnom jazyku. Každý riadok tabuľky je označený identifikátorom, ktorý sa nachádza v prvom stĺpci. Tieto identifikátori umožňujú reprezentovať text v rôznych jazykoch. Kvôli veľkému počtu takýchto textov sa pre organizáciu zaviedol tzv. menný priestor (*namespace*). Tento menný priestor dokáže organizovať jednotlivé identifikátory na logické skupiny. Výsledný identifikátor sa skladá z menného priestoru, oddeľovača a ID textu (napr. IAT/SettingPage/SettingPageGeneralContentTexts/text_yes). Ďalšou výhodou takého prístupu ku jednotlivým textom je opätovné využite v HMI aplikácií a úprava textov z jedného miesta. [1]

Namespace: IAT/SettingPage/SettingPageGeneralContentTexts  Text ID: text-no

| | Text ID | Czech (cs) | French (fr) | German (de) |
|---|---|---|---|---|
| 1 | text_yes | Áno | Oui | Ja |
| 2 | text-no | Nie | Non | vy |
| 3 | *New_Text_ID* | | | |

Obr. 3: Ukážka textového lokalizačného súboru.

HMI aplikácie slúžia na interakciu medzi operátorom a strojom ako už bolo spomenuté. Preto musia byť HMI aplikácie schopné manipulovať s dátami automatizačného programu ( procesné veličiny, stavy stroja atď. ). Z tohto dôvodu musí existovať prepojenie medzi prvkami HMI aplikácie a prvkami automatizačného softwaru. V mappView toto prepojenie popisuje tzv. *data binding*.

**Data binding**

Toto prepojenie jasne definuje o aký zdroj dát ide (OPC UA uzol), na aký prvok vizualizácie má byť zdroj dát napojený, a ako má táto výmena dát reagovať. MappView rozlišuje tieto typy pripojenia :

- ***Value binding*** – slúži na prepojenie jednoduchých hodnôt (1 procesná premenná).
- ***Node binding*** – slúži na prepojenie OPC UA uzlu, ktorý okrem hodnoty procesnej veličiny obsahuje aj informáciu o jej fyzikálnej jednotke a jej povolenom rozsahu.
- ***Array binding*** – Slúži na prepojenie jedného prvku poľa a prvku vizualizácie.
- ***List binding*** – slúži na výber jednej premennej z určitého zoznamu premenných. Tento výber určuje tzv. selektor – numerická hodnota podľa ktorej prebieha výber konkrétnej premennej.

Pre požadovaný smer výmeny dát medzi vizualizáciou a automatizačným softwarom mappView využíva tieto nasledovné módy:

- ***oneWay*** – mód poskytuje len čítanie procesných hodnôt do zdroja.
- ***twoWay*** – mód poskytuje čítanie aj zápis procesných hodnôt do zdroja.
- ***oneWayToSource*** – mód poskytuje len zápis procesných hodnôt do zdroja.

Prepojenia sa nachádzajú v súboroch s príponou *.binding*. V týchto súboroch sa vyskytuje xml štruktúra, ktorá jasne popisuje správanie jednotlivých prepojení. Každý takýto súbor musí mať pridelený globálne jedinečné ID. Tieto súbory sa nachádzajú v sekcii *Configuration view* AS, v priečinku mappView danej konfigurácie. [2]

Pre popis správania sa mappView HMI aplikácie sme v predošlej podkapitole opísali ako prebieha výmena dát medzi automatizačnom softwarom a vizualizáciou. *Data bidning* jasne definuje, ktoré dáta, a akým smerom sa majú tieto dáta prenášať. Pre niektoré HMI aplikácie to však môže byť nedostačujúce. MappView ponúka udalosti (*Event*) a akcie (*Action*) na individuálny popis chovania sa jednotlivých častí vizualizácie. Udalosti slúžia na zachytenie zmien (zmena hodnoty procesnej veličiny) alebo interakcie užívateľa (stlačenie tlačidla) v HMI aplikácií a umožňuje reagovať na danú zmenu pomocou akcie. Akcia je odpoveď na vyvolanú udalosť. Kombinácia vyvolania udalosti a nejaká postupnosť akcií sa označuje ako spracovanie udalostí (*Event handling*).

**Event binding - udalosti**

Udalosti slúžia na zachytenie zmien vyvolaných užívateľom alebo programom a umožňuje na tieto zmeny reagovať. MappView nám poskytuje viacero typov udalostí.

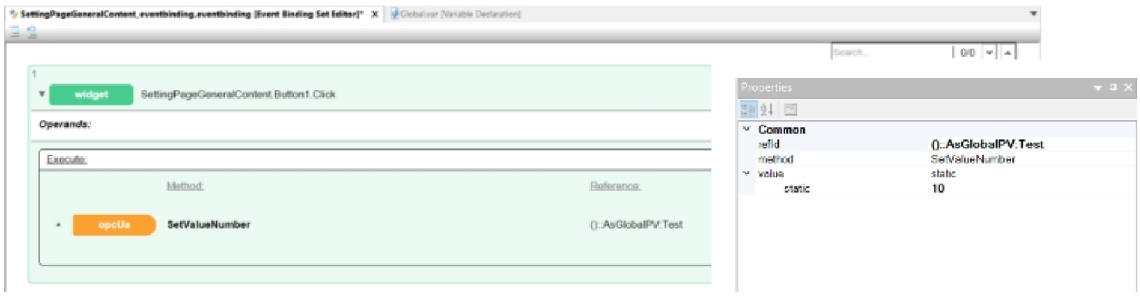
- ***OPC UA event*** – definuje udalosti, ktoré slúžia na detekciu zmien v procesných hodnotách automatizačného softwaru. Príkladom môže byť udalosť s názvom *ValueChanged*
- ***Widget event*** – Widget event poskytuje informácie o zmene nejakých vlastností konkrétneho widgetu. Rôzne skupiny widgetov poskytujú rôzne eventy. Príkladom môže byť event *Click*, ktorý vyvolá určitú akciu po stlačení tlačidla.
- ***Session event*** – tieto udalosti detegujú zmenu hodnôt *Session* premenných. Každý klient má individuálne *Session* premenné. Tie sú zároveň nezávislé na ostatných klientoch. [2]

**Action - Akcie**

Akcia reprezentuje spustiteľnú operáciu alebo metódu ako odpoveď na vyvolanú udalosť. MappView poskytuje 4 zdroje týchto akcií, každý zdroj obsahuje jednu alebo viacej akcií. Zdroje akcií sú:

- ***OPC UA*** – Táto séria akcií manipuluje s OPC UA uzlami. Takouto akciu môže byť napríklad metóda *SetValueNumber*, ktorá nastaví danú premennú na zvolenú hodnotu.
- ***Session*** – Tieto akcie slúžia na manipuláciu so *session* premennými.
- ***Client*** – Tento typ akcií poskytuje priamo HMI klient. Medzi tieto akcie patrí napríklad zobrazenie dialógu alebo funkcia prihlásenia užívateľa.
- ***Widgets*** – Táto séria akcií slúži na manipulovanie vlastnosťami jednotlivých widgetov. Príkladom môže akcia *SetVisible*, ktoré hodnota *True/False* popisuje viditeľnosť daného widgetu.

Súbory obsahujúce udalosti a akcie majú príponu *.eventbinding* a upravujú sa v grafickom alebo textovom editore. Tieto súbory sa nachádzajú v sekcií *Configuration View* v priečinku mappView danej konfigurácie stroja rovnako ako *data binding* súbory. [2]




Obr. 4: Editor udalosti a akcií.

Všetky tieto spomenuté komponenty vizualizácie je potrebné priradiť ku konkrétnej vizualizácii. Na to slúži tzv. objekt vizualizácie, ktorý ma príponou .vis a nachádza sa v *Configuration View* v zložke mappView. Objekt obsahuje zoznam komponentov z *Logical View* a *Configuration View*, ktoré sú prístupné klientom. Niektoré časti štruktúry súboru sú nasledovné:

- ***Visualization ID*** – slúži ako identifikácia HMI aplikácie na klientovi. Ak projekt obsahuje viacero objektov vizualizácie, potom ID slúži na prístup pomocou URL adresy k určitej HMI aplikácie a jej zobrazeniu.
- ***Start page*** – Obsahuje ID stránky, ktoré sa má zobraziť pri inicializácií HMI aplikácie.
- ***Pages*** – Zoznam všetkých stránok HMI aplikácie.
- ***BindingsSets*** – Zoznam všetkých *data binding* súborov HMI aplikácie.
- ***EventBindingSets*** – Zoznam všetkých *event binding* súborov HMI aplikácie.
- ***Contents*** – Zoznam obsahov, ktoré sa využívajú v rámci widgetu *Content Control*[1].

Štruktúra obsahuje aj iné časti, ktoré bližšie popisujú konfiguráciu HMI aplikácie, napr. popis správania sa virtuálnej klávesnice alebo správanie dotykových gest. [2]

Výsledná konfigurácia mappView prebieha v súbore s príponou .*mapviewcfg*. V súbore prebieha konfigurácia mappView servera a každá konfigurácia stroja môže obsahovať iba jeden takýto súbor. Medzi jednotlivé nastavanie patrí napríklad nastavenie servera (protokol, číslo portu, počet pripojiteľných klientov) alebo nastavenie klienta (predvolená vizualizácia) . [2]

| | | |
|---|---|---|
| MappViewConfiguration | | |
| Server configuration | | |
| Protocol | HTTP | |
| Port Number | 81 | |
| Maximal client connections | 3 | |
| Maximal B&R client connections (deprecated) | 0 | |
| Authentication mode | RBAC | |
| OPC-UA system | | |
| Server connection timeout | 5000 | ms |
| Sampling rate groups | | |
| default | 200 | ms |
| slow | 1000 | ms |
| fast | 100 | ms |
| Initial ValueChanged Events | TRUE | |
| Session Timer | | |
| *Timer 1* | | |
| *TimerId* | *Timer1* | |
| *Interval* | *1000* | |
| *Timer mode* | *repetitive* | |
| Client configuration | | |
| ID of default visualization | visualization | |
| Widget configuration | | |
| Rendering policy | default | |

Obr. 5: Konfigurácia mappView servera.

[1] Widget Content control slúži na dynamické zobrazovanie obsahov. Dovoľuje programátorom definovať podmienky na základe, ktorých sa zvolané obsahy zobrazujú.

# 3 TECHNOLÓGIE POUŽITÉ PRI VÝVOJI APLIKÁCIE

## 3.1 Programovací jazyk Python

Python bol vytvorený v roku 1991 holandsky počítačovým vedcom a matematikom, ktorý sa volal Guido van Rossum. Python je interpretovaný, platformovo nezávislý, objektovo orientovaný jazyk, ktorý patrí do skupiny vyšších programovacích jazykov. Dynamické typovanie a vstavané dátové štruktúry vyšších levelov robia z programovacieho jazyka Python vhodným nástrojom pre rýchly vývoj aplikácií a tvorbu automatických skriptov, ktoré vykonávajú repetitívnu úlohu. Vďaka jeho jednoduchej syntaxi sa ľahko učí a takisto jednoduchá syntax zvyšuje čitateľnosť a produktivitu vývoja. Ako bolo spomenuté ide o interpretovaný jazyk, ktorého programy budú vždy pomalšie oproti kompilovaným programom. Túto nevýhodu kompenzuje rýchlosť vývoja a kratší kód oproti kódom napísaných v jazyku C++ alebo Java. Na vývoj nášho programu bola použitá verzia Python interpreteru 3.10. [3].


Obr. 6: Logo programovacieho jazyka Python. [4]

## 3.2 Poetry

Pri vývoji aplikácií v jazyku Python je praktické používať virtuálne prostredie. Virtuálne prostredie je izolované Python vývojové prostredie, ktoré obsahuje všetky nevyhnuté spustiteľné časti – zdrojové súbory a všetky knižnice potrebné na správne fungovanie programu. Na vytvorenie týchto virtuálnych prostredí existuje viacero spôsobov. Doporučený spôsobom tvorby virtuálnych prostredí je použitie modulu *venv*. Zadávateľom práce bolo však odporúčané použiť knižnicu tretích strán s názvom Poetry. Poetry je nástroj na správu závislosti a tvorbu virtuálnych prostredí. Umožňuje vývojárom deklarovať knižnice, na ktorých je aplikácia závislá a spravuje (inštaluje, aktualizuje adť.) tieto knižnice za nás. Poetry vyžaduje verziu programovacieho jazyka Python 2.7 alebo 3.5+. [2]. [5]

[2] Python 2.7 a 3.5 nebude podporovaný s príchodom novej verzie Poetry 1.2. Preto je potrebné zvážiť prechod na novšiu verziu pythonu.

## 3.3 Lxml

V mappView je každý komponent vizualizácie súbor, ktorý obsahujú špecifickú xml štruktúru. Pre manipuláciu s xml súbormi sme vybrali Python knižnicu lxml, ktorá umožňuje jednouchú manipuláciu s týmto typom súborom. Medzi hlavné výhody patrí vysoká rýchlosť pri *parsovaní*[3] veľkých xml dokumentov, dobre spracovaná dokumentácia knižnice a podpora všetkých funkcionalít značkovacieho jazyka xml. Táto knižnica je z veľkej časti kompatibilná so známou knižnicou ElementTree API. Oproti knižnici ElementTree API ponúka špecifické funkcie xml jazyka ako XPath alebo XML Schema. Takisto poskytuje nástroje na konverziu xml dokumentov na objekty jazyka Python, čo výrazne uľahčuje prácu s xml dokumentmi. [6]

## 3.4 Pathlib

Pri manipulácií s xml dokumentmi sa často stretávame s interakciou súborového systému. Preto je dôležité správne zaobchádzať z názvami a cestami jednotlivých súborov. Python obsahuje modul s názvom Pathlib, ktorý obsahuje užitočné funkcie na vykonávanie úloh súvisiacich sú súbormi. Pathlib ďalej ponúka jednoduchší a čitateľnejší spôsob vytvárania ciest reprezentovaním ciest súborového systému ako objektov a aktivizuje písať kód, ktorý je multi-platformový. [7]

## 3.5 Pyside6

Qt je multiplatformový *framework* napísaný v programovacom jazyku C++, ktorý je určený na tvorbu grafických užívateľkách rozhraní a desktopových aplikácií. Qt sa dá volať z mnohých iných programovacích jazykov pomocou oficiálnych aj neoficiálnych API. Medzi oficiálnu API patrí tiež Pyside6. Táto knižnica je momentálne vyvíjaná priamo Qt Company. Qt obsahuje veľa preddefinovaných widgetov, funkcií, dátových modelov na tvorbu efektívnych a výkonných aplikácií, čo zvyšuje produktivitu vývoja jednotlivých aplikácií. Pri vývoji aplikácie sme tiež použili knižnicu Qt qt-material 2.10, ktorá obsahuje preddefinované grafické témy pre rozhranie, aby sme sa my mohli plne venovať vývoji funkcionalít našej aplikácie. [8]

[3] Parsovanie – syntaktická analýza textu

# 4 NÁVRH NÁSTROJA PRE ZJEDNODUŠENIE TVORBY VIZUALIZÁCIÍ V MAPPVIEW

Táto kapitola opisuje postup pri vývoji aplikácie v programovacom jazyku Python. Na začiatku je spomenutá motivácia pre vývoj takejto aplikácie. V kapitole sa ďalej venujeme celkovej realizácií aplikácie. Súčasťou je aj návrh architektúry aplikácie, návrh jednotlivých funkcií a návrh grafického rozhrania, aby užívateľ tejto aplikácie mal z používania čo najpríjemnejší zážitok.

Pri vývoji HMI aplikácií v mappView sme si všimli, že vyžaduje veľa repetitívnych krokov, ktoré nevyžaduje programátorské myslenie a je možné ich automatizovať. Preto sme sa rozhodli navrhnúť aplikáciu, ktorá bude tieto repetitívne úlohy robiť za nás, aby sme sa my mohli plne venovať programovaniu mappView HMI aplikácií. Našej aplikácií sme sa rozhodli dať meno page-generator. Názov reprezentuje hlavnú funkciu našej aplikácie a tou je generovanie prázdnych stránok vizualizácie a obsluhu jej súčastí.

## 4.1 Architektúra aplikácie

Pri návrhu aplikácie sme museli zohľadniť funkčné a nefunkčné požiadavky na našu aplikáciu. Funkčné požiadavky predstavujú správanie sa aplikácie a popisujú, ako by mal systém reagovať na vstupy od užívateľa. Funkčné požiadavky pre aplikáciu sú :

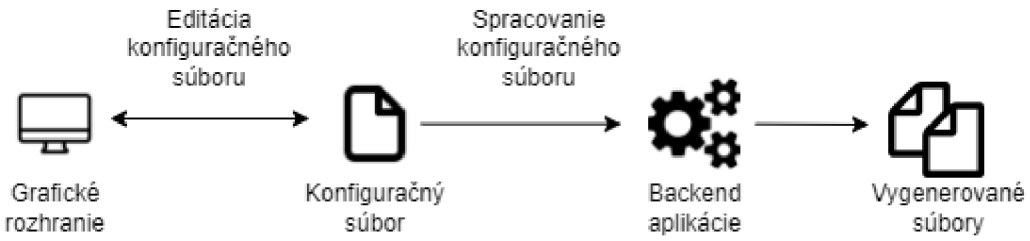
- Tvorba stránok
  - Vytvorenie prázdnej stránky
  - Vytvorenie a priradenie obsahov podľa rozloženia stránky
- Vytvorenie prázdnych data binding, event binding a lokalizačných súborov
- Obsluha súborového systému AS
  - Pridávanie a mazanie záznamov o obsahu určitého priečinka do tzv. súboru package
- Obsluha textového systému AS
  - Pridávanie a mazanie záznamov o lokalizačných súboroch projektu AS.
  - Pridávanie a mazanie záznamov do objektu vizualizácie
- Grafické rozhranie
  - Zobrazenie informácie z konfiguračného súboru
  - Tvorba konfiguračného súboru
  - Editácia konfiguračného súboru
    - Editácia stránok
    - Editácie obsahov
    - Úprava nastavení skriptu na generovanie stránok

Nefunkčné požiadavky sú požiadavky, ktoré kladú podmienky na aplikáciu. Nefunkčné požiadavky pre našu aplikáciu sú:

- Grafické rozhranie je nezávisle od backendu aplikácie. V budúcnosti sa predpokladá prechod na webovú platformu
- Modularita aplikácie
- Jednoduchá implementácia do AS

Po zohľadnení všetkých požiadaviek na našu aplikáciu sme navrhli vhodnú architektúru aplikácie. Navrhnutá architektúra našej aplikácie je postavená na troch hlavných častiach

- **Grafické rozhranie** – Grafické rozhranie je nezávislá časť aplikácie a poskytuje užívateľovi jednoduché prostredie na editovanie konfiguračného súboru, ktorý je aj výstupom grafického rozhrania. Konfiguračný súbor je naďalej spracovaný backendom našej aplikácie.
- **Konfiguračný súbor** – konfiguračný súbor slúži ako komunikačná vrstva medzi grafickým rozhraním a backendom aplikácie. Súbor obsahuje podrobný opis stránok, ktoré backend po spustení programu vygeneruje. Konfiguračný súbor taktiež obsahuje určité nastavenia backendu ako napríklad povolenie obsluhy súborového systému AS
- **Backend** – backend je samostatná časť aplikácie, ktorá spracováva konfiguračný súbor. Na základe informácií z konfiguračného súboru vygeneruje stránky a ich potrebné súbory. Backend taktiež obsluhuje aj niektoré časti AS ako napríklad súborový systém.




Obr. 7: Architektúra aplikácie.

Modularita programu je zabezpečená objektovo orientovaným programovaním. Objektovo orientované programovanie (OOP) je programovacia paradigma, ktorá využíva takzvané objekty. Objekty sú dátové štruktúry, ktoré majú vlastnosti, metódy a udalosti, ktoré popisujú správanie sa daného objektu. Tento prístup ku návrhu aplikácie výrazne zlepšuje opakované využitie kódu.

## 4.2 Príprava projektu

Pre beh aplikácie je kľúčové vytvorenie virtuálneho prostredia na vývoj Python aplikácií, tj. inštalácia jednotlivých knižníc a popis príkazov na prácu v takomto prostredí.

Úplne na začiatku je potrebné mať na počítači nainštalovaný správny Python interpreter. Odporúčaná verzia programovacieho jazyka Python je 2.7 alebo 3.5+ .My sme v projekte používali verziu 3.10.1. Inštalácia je veľmi jednoduchá a prebieha pomocou inštalačného okna, ktoré obsahuje popis všetkých krokov pre úspešnú inštaláciu. Inštalátor pridá taktiež pridá programovací jazyk Python do Windows premennej PATH, čo sprístupní volanie príkazov pomocou príkazového riadka.

Po nainštalovaní požadovanej verzie jazyka Python interpreteru sa nám sprístupní manažér knižníc s názvom pip[4]. Pomocou neho nainštalujeme lepší manažér knižníc s názvom Poetry príkazom:

- **pip install --user poetry.**

Poetry oproti pip ponúka lepšie funkcie na správu knižníc ako virtuálne prostredie, hromadné aktualizácie knižníc a oveľa viac.

**Vytvorenie projektu a virtuálneho prostredia**

Vytvorenie nového projektu prebieha cez príkazový riadok pomocou príkazu ***poetry new názov-projektu.*** Tento príkaz vygeneruje základu štruktúru súborov na vývoj aplikácie. Najdôležitejším je súbor s názvom *pyproject.toml.* Tento súbor obsahuje zoznam knižníc, ktoré sú nevyhnutné pre správne fungovanie aplikácie. Najjednoduchšia cesta ako aktivovať virtuálne prostredie je príkaz ***poetry shell.*** Tento príkaz vytvorí skrytý priečinok s názvom *.venv*, ktorý obsahuje zdrojové súbory interpreteru a knižníc. Taktiež vytvorí interaktívny príkazový riadok, pomocou ktorého priebehu ďalšia práca s týmto manažérom knižníc.

**Popis základných príkazov**

- **poetry new „názov-projektu“** – príkaz slúži na vytvorenie nového pracovného adresára
- **poetry shell** – aktivuje virtuálne prostredie a spustí interaktívny príkazový riadok
- **poetry add „názov-knižnice“** – pridá knižnicu do súboru *pyproject.toml*, a nainštaluje potrebné závislosti na správne fungovanie knižnice
- **poetry remove „názov-knižnice“** – odstráni knižnicu z projektu
- **poetry install** – nainštaluje všetky knižnice, ktoré obsahuje súbor *pyproject.toml*
- **poetry update –** aktualizuje všetky knižnice, ktoré sú používane v projekte a nachádzajú sa v súbore *pyproject.toml*.
- **poetry run „názov-súboru“** – spustí Python skript

[4]Pip - Package installer for python

- **poetry run python.exe „názov-súboru –flag“** – spustí Python skript s argumentmi, ktoré užívateľ zadal cez príkazový riadok. [5]

Po nainštalovaní všetkých nevyhnutných knižníc, ktoré sú popísané v kapitole 3, sme pripravení na vývoj našej aplikácie.

## 4.3 Návrh konfiguračného súboru

Ako už bolo spomenuté tento konfiguračný súbor slúži ako komunikačná vrstva medzi backendom aplikácie a grafickým rozhraním aplikácie. Ide o xml súbor, ktorého štruktúra je navrhnutá, aby spĺňala naše požiadavky. Štruktúra je rozdelená na dve časti. Prvá časť obsahuje určité nastavenia backendu a druhá obsahuje zoznam stránok, ktoré backend po spracovaní vygeneruje. Tento súbor je editovateľný priamym zásahom do xml štruktúry alebo pomocou grafického rozhrania, ktoré je na to určené. Xml štruktúra súboru môže vyzerať nasledovne :

```
<?xml version='1.0' encoding='UTF-8'?>
<configuration>
  <scriptSettings>
     <packageWatcher allow="True"/>
     <textConfigWatcher allow="True"/>
     <configPath path="C:/Projects/personal/page
generator/Physical/Config1/X20CP1586"/>
  </ scriptSettings >
  <pages>
     <page id="abcd" layoutId="MainLayout" visuId="Visualization">
       <content id="MainContent" areaId="MainArea">
         <dataBinding id="MainContent"/>
         <eventBinding id="MainContent"/>
         <tmxFile id="MainContentTexts"/>
       </content>
     </page>
     <page id="xyz" layoutId="MainLayout" visuId="Visualization">
       <content id="GlobalHeader" areaId="GlobalHeaderArea">
         <dataBinding id="GlobalHeader"/>
         <eventBinding id="GlobalHeader"/>
         <tmxFile id="GlobalHeader"/>
       </content>
     </page>
   </pages>
</configuration>
```

Jednotlivé elementy a ich atribúty, ktoré sa vyskytujú v xml štruktúre konfiguračného súboru sú popísane v nasledujúcej tabuľke.

| **Názov elementu** | **Popis** |
|---|---|
| configuration | Definuje priestor, v ktorom sa nachádza daná konfigurácia. |
| scriptSettings | Element slúži na oddelenie nastavení skriptu od zoznamu generovaných stránok. |
| packageWatcher | Element definuje pomocou svojho atribútu *allow* aktiváciu alebo deaktiváciu obsluhy súborového systému AS. Atribút *allow* ma dátový typ boolean. |
| textConfigWatcher | Element definuje pomocou svojho atribútu *allow* aktiváciu alebo deaktiváciu obsluhy textového systému AS. Atribút *allow* má dátový typ boolean. |
| configPath | Element obsahuje informáciu o ceste zložky konfigurácie stroja, ktorú má skript obsluhovať. |
| pages | Element slúži na oddelenie zoznamu stránok od nastavení skriptu. Reprezentuje zoznam stránok, ktoré vygeneruje skript. |
| page | Element, ktorý definuje samostatnú stránku. Element pages môže obsahovať viac takýchto elementov. Atribút *id* je unikátny názov súboru. Atribút *layoutId* je názov existujúceho layoutu, ktorý sa má priradiť stránke. Atribút *visuId* je názov vizualizácie, ku ktorej daná stránka patrí. |
| content | Element, definuje obsah príslušnej stránky. Element page môže obsahovať viac takýchto elementov. Atribút *id* je unikátny názov súboru. Atribút je *areaId* názov oblasti ku, ktorej je daný obsah následne priradený . |
| dataBinding | Element reprezentuje databinding súbor príslušného obsahu. Atribút *id* je unikátny názov súboru. |
| eventBinding | Element reprezentuje eventbinding súbor príslušného obsahu. Atribút *id* je unikátny názov súboru. |
| tmxFile | Element reprezentuje lokalizačný textový súbor príslušného obsahu. Atribút *id* je unikátny názov súboru. |

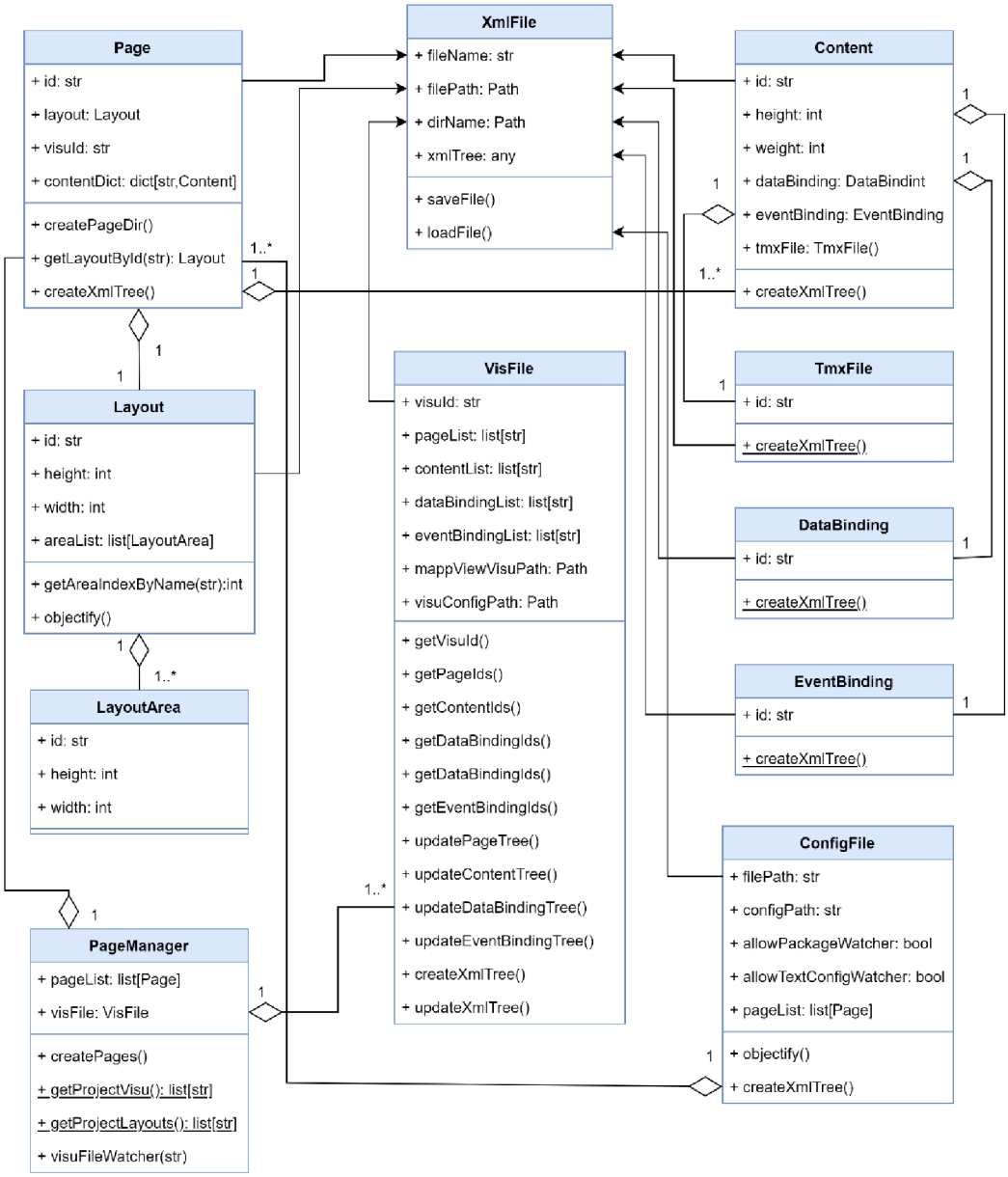
Tab. 1: Popis jednotlivých elementov konfiguračného súboru.

## 4.4 Návrh backendu aplikácie

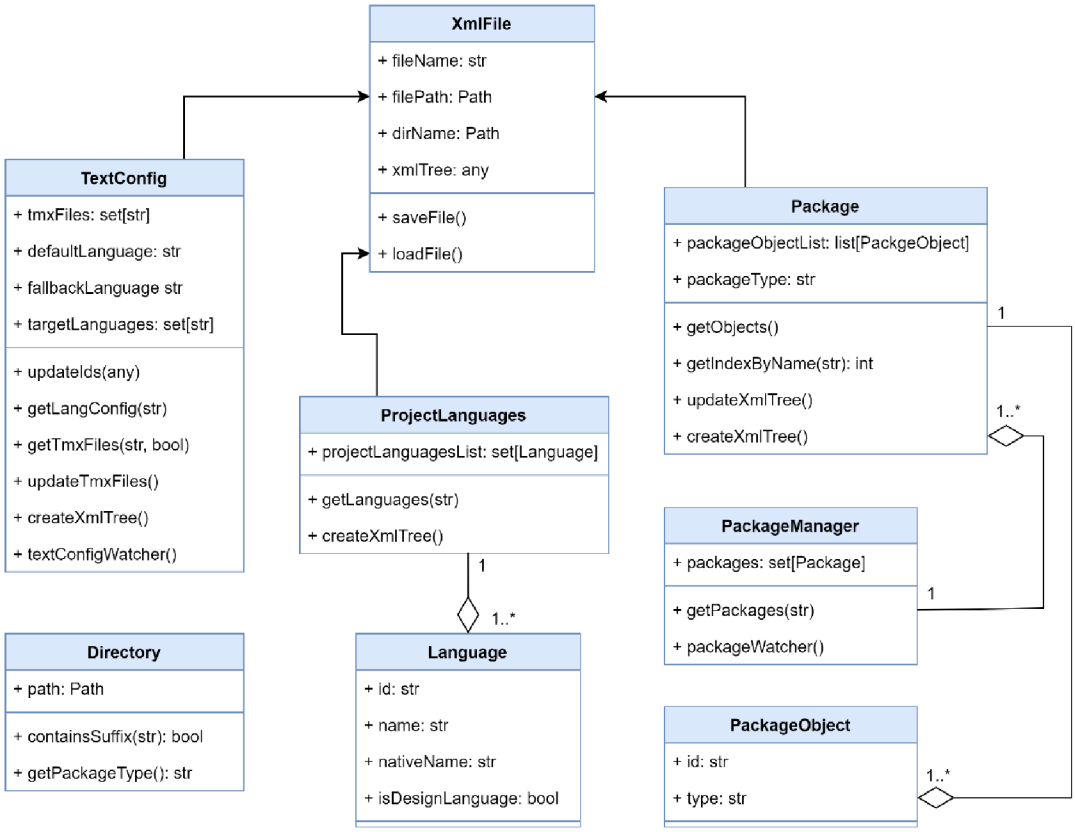
Backend predstavuje automatický skript, ktorý môžeme rozdeliť na tri časti.

- Generovanie stránok a príslušných súborov
- Obsluha textového systému AS
- Obsluha súborové systému AS

Nasledujúci návrhový diagram zobrazuje štruktúru a vzťahy medzi jednotlivými triedami.




Obr. 8: Prvá časť triedneho diagramu




Obr. 9: Druhá časť triedneho diagramu

### 4.4.1 Generovanie stránok

Každý súbor, z ktorého sa skladá mappView vizualizácia je v podstate xml súbor so špecifickou príponou. Na manipuláciu so xml štruktúrami sme použili knižnicu lxml, ktorú sme bližšie popísali v kapitole 3.3. Ďalšou nevyhnutnou časťou práce so súbormi sú úlohy spojené s ich cestami v súborovom systéme. Medzi tieto úlohy môžu napríklad patriť úlohy ako rekurzívne vyhľadávanie súborov v zložkách, zistenie príponou súborov, overenie či prvok v zložke je zložka alebo súbor. Na tieto úlohy sme použili vstavanú knižnicu s názvom Pathlib.

**XmlFile**

Ako je z triedneho diagramu zrejmé, všetky triedy, ktoré reprezentujú špecifické súbory mappView vizualizácie dedia vlastnosti a metódy zo základnej triedy *XmlFile*. Trieda je navrhnutá tak, aby predstavovala obecný xml súbor. Pre manipuláciu s týmito typmi súborov sme museli implementovať vlastnosti a metódy, ktoré nám s tým pomôžu.
Hlavnými vlastnosťami triedy *XmlFile* je vlastnosť *filePath* a *xmlTree*. *FilePath* je inštancia triedy *Path* a predstavuje cestu ku kontraktnému súboru v súborovom systéme počítača. Vlastnosť *xmlTree* obsahuje xml štruktúru daného súboru. Trieda *XmlFile* taktiež obsahuje metódy na manipuláciu s touto xml štruktúrou, ako načítanie a uloženie xml štruktúry do súboru.

**ConfigFile**

Trieda *ConfigFile* slúži na spracovanie nami navrhnutého konfiguračného súboru, ktorého návrh je popísaný v kapitole 4.3. Trieda *ConfigFile* obsahuje vlastnosti, ktoré reprezentujú dáta, ktoré sa nachádzajú v konfiguračnom súbore.

Po načítaní a parsovaní konfiguračného súboru sa dáta tohto súboru priradia ku príslušným vlastnostiam. Na to slúži metóda *objectify()*, ktorá transformuje xml štruktúru dát na vlastnosti objektu *ConfigFile*. Prostredníctvom toho vieme v programe pristupovať ku dátam z konfiguračného súboru ako ku Python premenným. Medzi tie dáta patrí aj zoznám stránok, ktorý sa následne predá triede *PageManager,* aby vytvoril tieto stránky, ak neexistujú.

Trieda *ConfigFile* taktiež obsahuje metódu s názvom *createXmlTree()*, ktorá z vlastnosti inštancie triedy vygeneruje požadovanú xml štruktúru konfiguračného súboru a priradí je vlastnosti *xmlTree*, ktorú zdedila z triedy *XmlFile*.

**PageManager**

Tejto triede, pre správne fungovanie, je nutné predať pri vytvorení nového objektu typu *PageManger* zoznam stránok, ktoré ma manažér vygenerovať. Ako už bolo spomenuté v kapitole 4.1, zoznam stránok sa získa parsovaním konfiguračného súboru pomocou triedy *ConfigFile*.

Zoznam stránok prestavuje vlastnosť *pageList*, je to dátový typ *list*[5], ktorého prvky sú objekty typu *Page*. Objekt *Page* reprezentuje stránku mappView vizualizácie a obsahuje podrobné informácie o generovanej stránke a jej obsahoch, tieto informácie budú podrobnejšie rozobraté v nasledujúcich odstavcoch.

Ak súbor s konkrétnou stránkou neexistuje, manažér najprv vytvorí zložku danej stránky. Potom vytvorí xml štruktúru z vlastností objektu *Page* a vytvorí súbor s príponou .page, ktorý obsahuje základnú xml štruktúru stránky.

Ako už bolo spomenuté v kapitole 2.2 každá stránka sa skladá z obsahov. Každý objekt typu Page obsahuje vlastnosť s názvom *contentDic*t. Táto vlastnosť má dátový typ *dictionary (dict)*[6], kde kľúč je identifikátor oblastí a hodnota je objekt typu *Content.* Objekt *Content* predstavuje obsah stránky. Bližšie informácie o triede *Content* budú popísané v nasledujúcich odstavcoch. Ak súbor s obsahom neexistuje, manažér pre každý obsah z vlastnosti *contentDict* vytvorí príslušnú základnú xml štruktúru a súbor uloží do zložky stránky, ku ktorej obsah patrí. Taktiež pre každý novo vytvorený obsah vytvorí *data binding, event binding* a textové lokalizačné súbory do príslušných zložiek.

Trieda *PageManger* taktiež obsahuje vlastnosť *visFile*, ktorá predstavuje súbor s objektom vizualizácie, ktorý sa nachádza v *Configuration View* AS. Na obsluhu tohto súboru sme navrhli metódu *visFileWatcher()*, ktorá vytvorí nový súbor s objektom vizualizácie, ak neexistuje. A ak existuje, vyvolá metódy objektu *VisFile*, ktoré

[5] list – kolekcia dát, ktorá je zoradená, menná, indexovaná a povoľuje duplicitné hodnoty. V programovacom jazyku Python predstavuje dynamické pole. [11]

[6] dictionary (dict) – kolekcia dát, ktorá ukladá dáta v dvojici kľuč-hodnota. Kolekcia je zoradená, menná a nepovoľuje duplicitné hodnoty. [7]

aktualizujú xml štruktúru tohto súboru. Trieda *VisFile* bude podrobnejšie popísaná ďalej v tejto kapitole.

Trieda obsahuje aj statické metódy *getProjectVisu()* a *getProjectLayouts(),* ktoré prehľadajú súborový systém projektu v AS a poskytnú nám zoznam všetkých vizualizácií a rozložení, ktoré sa nachádzajú v danom projekte. Tieto metódy sa využívajú najmä pri grafickom rozhraní, kde užívateľ potrebuje vedieť, aké vizualizácie a rozloženia existujú v konkrétnom projekte AS a následne im poskytuje možnosť si vybrať z existujúcich možností.

**VisFile**

Trieda *VisFile* je navrhnutá na obsluhu objektu vizualizácie, ktorý sa nachádza v *Configuration View* AS. Tento súbor ma príponu *.vis* a obsahuje xml štruktúru, ktorá jasne definuje aktuálne zloženie mappView vizualizácie. Každý takýto súbor ma v xml štruktúre jedinečný identifikátor na rozlíšenie týchto objektov, ak by ich bolo viacej. Pri tvorbe nového súboru objektu vizualizácie sa ako identifikátor použije názov vizualizácie, ku ktorej patrí.

Obsluhou myslíme pridávanie a mazanie záznamov a komponentov, ako napríklad zoznam stránok, zoznam obsahov, zoznam *data binding* súbor a podobne. Tieto súbory sú po registrovaní do objektu vizualizácie prístupne všetkým HMI klientom. Trieda obsahuje patričné vlastnosti, ktoré predstavujú zoznam identifikátorov týchto súborov. Súbory, ktoré vygeneroval backend aplikácie majú zhodný identifikátor a názov súboru. To nám zjednoduší prístup ku identifikátorom stránok a nemusíme zakaždým otvárať a čítať súbor, aby sme zistili daný identifikátor stránky.

Trieda obsahuje aj metódy, ktoré tieto súbory vyhľadajú v súborovom systéme AS a poskytnú nám aktuálny zoznam identifikátorov komponentov vizualizácie, ktoré môže podľa našich potrieb ďalej využívať. Každý z týchto komponentov, ktorý je potrebný registrovať má vlastnú metódu na aktualizáciu príslušnej časti xml štruktúry, ako napríklad *updatePageTree()*.

V prípade, ak by tento súbor ešte neexistoval, metóda *createXmlTree()* vytvorí príslušnú xml štruktúru s potrebnými zoznamami identifikátorov súborov a pomocou zdedenej metódy *saveFile()* vytvorí a uloží túto xml štruktúru do novo vytvoreného súboru s príponou *.vis*.

**Page**

Trieda *Page* reprezentuje jednu stránku mappView vizualizácie. Stránka má unikátny identifikátor v rámci vizualizácie a jej rozmer a rozloženie je definované priradením rozloženia stránky. Stránka taktiež obsahuje vlastnosť *visuId*, ktorá reprezentuje identifikátor vizualizácie, ku ktorej stránka patrí.

Na získanie informácie o rozložení sme navrhli metódu *getLayoutById(),* ktorá ako parameter vezme id rozloženie a vráti objektovú formu xml štruktúry daného rozloženia. Informácie o rozložení stránky sú prístupné pomocou vlastnosti *layout*.

Medzi hlavné informácie o rozložení stránky je rozloženie jednotlivých oblastí. Každej oblasti tejto stránky je nutné priradiť príslušný obsah. Na to slúži vlastnosť

s názvom *contentDict*, ktorá ukladá dáta v dvojici kľúč-hodnota, kde kľúč je identifikátor oblasti a hodnota objekt typu *Content*. Táto vlastnosť má dátový typ *dictionary* alebo v preklade slovník. Na tvorbu xml štruktúry stránok je navrhnutá metóda *createXmlTree()*, ktorá z prístupných vlastností vytvorí požadovanú xml štruktúru a uloží ju do zdedenej vlastnosti *xmlTree*. Uloženie do súboru prebieha ako u iných tried pomocou zdedenej metódy *saveFile()*. Spomenuté vlastnosti a metódy dedí trieda *Page* od triedy *XmlFile*.

**Layout**

Trieda *Layout* reprezentuje rozloženie stránky. Tento súbor naša aplikácia negeneruje, ale iba načítava a transformuje jeho xml štruktúru do formy Python premenných pomocou metódy *objectify()*. Rozloženia sa vytvárajú v grafickom alebo textovom editore vo vývojovom prostredí AS.

Rozloženie priamo definuje jeho identifikátor a jeho výška a šírka v pixeloch. Trieda *Layout* taktiež obsahuje vlastnosť *areaList*, ktorá zastupuje zoznam jednotlivých oblastí daného rozloženia. Táto vlastnosť má dátový typ *list* a jeho osobitné prvky sú objekty typu *LayoutArea,* ktoré predstavujú jednotlivé oblasti.

Trieda LayoutArea obsahuje iba nevyhnutné dáta o konkrétnej oblasti a to identifikátor, šírku a výšku v pixeloch.

**Content**

Trieda *Content* ako je zreteľné, predstavuje jeden obsah stránky. Obsah je definovaný jedinečným identifikátorom a svojou šírkou a výškou v pixeloch. Pomocou získaným informáciám o rozložení stránky pomocou triedy *Page* vieme šírku a výšku jednotlivých oblastí. Tento rozmer následne prevezme aj obsah podľa priradenia konkrétnej oblasti. Trieda taktiež obsahuje metódu na generovanie základnej xml štruktúru obsahu s názvom *createXmlTree()*. Obsahy, ktorých identifikátor začína predponou *Global* sa vygenerujú do zložky s názvom *AreaContents*. Táto zložka obsahuje všetky globálne obsahy, ktoré sa nachádzajú v mappView vizualizácií. Globálne obsahy sú také, ktoré využíva viac ako jedná stránka. Ostatné súbory s obsahmi sa generujú do zložky s príslušnou stránkou.
V odstavci, ktorý sa venuje triede *PageManager* bolo spomenuté že, naša aplikácia pre každý obsah stránky vygeneruje jeden *databinding*, *eventbidnig* a textový lokalizačný súbor do určených zložiek. Preto trieda *Content* obsahuje aj vlastnosti, ktoré reprezentujú tieto súbory. V práci sa nebudeme venovať popisu týchto tried, pretože tieto triedy len vytvárajú základnú xml štruktúru týchto súborov a následne ju zapíše do nových súborov.

### 4.4.2 Obsluha textového systému

Obsluha textového systému znamená automatickú aktualizáciu textového konfiguračného súboru, ktorý má príponu *.textconfig*. Tento súbor obsahuje informácie o používaných jazykoch v HMI aplikácií aj zoznam všetkých textových lokalizačných súborov. Tieto súbory majú príponu *.tmx* . Všetky nevyhnutné textové lokalizačné súbory musí vývojár registrovať do textového konfiguračného súboru pre správne zobrazenie textov, čo môže

byť niekedy zdĺhavé, ak ich je viac alebo sa môže na to zabudnúť. Zo spomenutých dôvod sme navrhli triedu *TextConfigManager*.

**TextConfigManager**

Trieda *TextConfigManager* je navrhnutá na automatickú obsluhu textového konfiguračného súboru s príponou *.textconfig*. Trieda obsahuje metódy, ktoré rekurzívne vyhľadajú všetky textové lokalizačné súbory s príponou *.tmx* a registrujú ich do xml štruktúry textového konfiguračného súboru.

Na udržiavanie aktuálneho zoznamu textových lokalizačných súborov slúži vlastnosť *tmxFileSet*. Táto vlastnosť má dátový typ *set*[7] a jeho prvky sú textové reťazce, ktoré predstavujú cestu v súborovom systéme ku jednotlivým lokalizačným súborom v požadovanom formáte. Ak nejaké súbory pribudli v súborovom systéme a v konfiguračnom súbore ešte o nich nie je záznam, tak ich metóda *updateTmxFiles()* registruje do konfiguračného súboru. V opačnom prípade teda, ak sa textový lokalizačný súbor odstránil zo súborového systému a je registrovaný v textovom konfiguračnom súbore, táto metóda vymaže záznam o ňom.

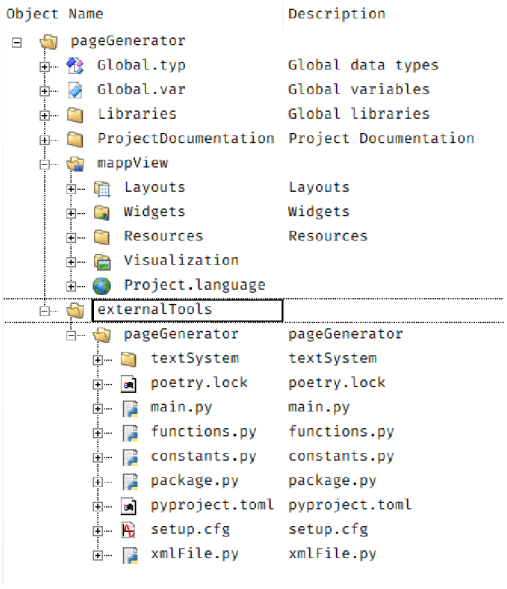
Ak textový konfiguračný súbor neexistuje, pomocou metódy *createXmlTree()* vieme vytvoriť novú xml štruktúru tohto súboru s aktuálnym zoznamom lokalizačných súborov a nami zvolenou základnou konfiguráciu použitých jazykov. Ako základnú konfiguráciu jazykov sme zvolili anglický jazyk ako pôvodný jazyk a český jazyk ako záložný. Zoznam základných jazykových lokalizácií je anglický, český a nemecký jazyk.

V triednom diagrame, ktorý je znázornený na obr.8 a 9 sa nachádzajú aj triedy *ProjectLanguage* a *Language*. Tieto triedy predstavujú jazykovú konfiguráciu projektu v AS. Pri vývoji aplikácií sme zistili, že nie sú potrebné a na prácu s týmito súbormi stačia nástroje, ktoré poskytuje AS. Pri nových verziách aplikácie plánujeme implementovať zmeny, ktoré odstránia tieto triedy.

[7] Set – kolekcia dát, ktorá je nezoradená, nemenná, neindexovaná a nedovoľuje duplicitné hodnoty. [11]

### 4.4.3 Obsluha súborového systému AS

Ako každý programátorsky projekt tak aj projekt v AS ma určitý súborový systém. Pri vývoji aplikácie v prostredí AS, pomáha tento organizovať tieto súbory (štruktúry, súbory s premennými, súbory s programom atď.) do takzvaných POU (*Program organization unit*). POU si vieme predstaviť ako zložku, ktorá organizuje aplikačné súbory podľa funkcie do logických zoskupení. V AS sa tieto zložky nazývajú *Package*. Tieto zložky sú voľne spojené s ostatnými zložkami v aplikácií. Zložky sú zapuzdrené a majú definované rozhranie pre každú položku zložky. AS dovoľuje definovať zložky v *Logical* a *Configuration View*.




Obr. 10: Súborový systém vývojového prostredia B&R Automation Studio.

Každá takáto zložka obsahuje súbor s názvom *Package.pkg* (existuje aj pár výnimočných prípadov, kedy sa súbor volá inak, ale obsah a prípona ostáva rovnaká). Každý súbor *Package.pkg* jasne definuje zložku, typ zložky a súbory, ktoré sa v nej nachádzajú. Pre viditeľnosť súborov v súborovom systéme AS je nutné zaznamenávať obsah týchto zložiek do jeho xml štruktúry. Avšak existuje pár obmedzení tohto súborového systému:

- Mená jednotlivých zdrojových súborov musia byť vždy unikátne.
- Maximálny počet znakov podadresárov a zdrojových súborov je obmedzený vlastnosťami operačného systému.
- Zložky s taskom[8] nemôžu obsahovať iné tasky.
- Akcie[9] v tasku sú deklarované globálne, preto musia mať unikátny názov v celom adresári aj podadresári .
- Zložky s taskom, knižnicami a s dátovými objektami nesmú obsahovať žiadne podadresáre s predponou SG.

[8] Task – Program bežiaci na PLC.
[9] Akcia (Action) – programová organizačná jednotka. [2]

- Adresáre a podadresáre, ktoré obsahujú knižnice a tasky aplikácie nesmú obsahovať súbory s príponou *.typ*, *.var.* a *.fun*.
- Všetky adresáre, lokálne funkcie a funkčné bloky sa musia nachádzať v koreňovom adresári (*Logical* alebo *Config)* projektu AS.

Tým, že my v aplikácií generujeme množstvo nových súborov a AS nie je schopné tieto externe vytvorené súbory zaznamenávať do spomenutých súborov *Package.pkg*, museli sme navrhnúť program, ktorý bude tieto zmeny v súborovom systéme AS monitorovať za nás. Túto časť aplikácie obsluhujú triedy *Package* a *PackageManager*.

**Package**

Trieda *Package* predstavuje jeden súbor *Package.pkg*, ktorý ako bolo už spomenuté slúži na organizáciu súborového systému AS. Trieda *Package* má dve vlastnosti. Prvá je typ súboru *Package.pkg* s názvom *packageType* a druhá je zoznam súborov, ktoré daná zložka obsahuje. Jej názov je *packageObjectList*. Zistenie o aký typ zložky ide, prebieha volaním statickej metódy *getPackageType* triedy *Directory*, ktorá na základe obsahu zložky alebo načítaní xml štruktúry súboru *Package.pkg*, zistí konkrétny typ zložky. Vlastnosť *packageObjectList* má dátový typ *list* a obsahuje dátové objekty *PackageObject*, ktoré reprezentujú jeden súbor v zložke. Objekt *PackageObject* nesie informácie o názve súboru, o type súboru a jeho popise.

Trieda taktiež obsahuje metódu na zistenie obsahu zložky, ktorú predstavuje. Táto metóda sa volá *getObjects()* a po jej zavolaní sa vlastnosti *packageObjectList* priradí príslušný zoznam objektov typu *PackageObject*. Na základe spomenutých vlastností sa môže vytvoriť nová xml štruktúra ak daný súbor neexistuje alebo sa len aktualizuje existujúca xml štruktúra. Na to slúžia metódy *createXmlTree()* a *updateXmlTree()*

**PackageManager**

Trieda *PackageManager* je navrhnutá tak, aby aplikácie vedeli pracovať so všetkými *Package.pkg* súbormi, ktoré existujú v projekte AS z jedného miesta. Trieda má iba jednu vlastnosť a tou je *packageSet*. Táto vlastnosť má dátový typ *set* a jeho jednotlivé prvky sú objekty typu *Package*.

Na skenovanie súborového systému AS a získanie všetkých *Package.pkg* súborov alebo zložiek, kde je nutné vytvoriť nový súbor *Package.pkg* sme navrhli metódu s názvom *getPackages()*. Táto metóda rekurzívne prehľadá povolené priečinky súborového systéme AS a získa informácie o existujúcich súborov *Package.pkg* alebo o zložkách, kde je nutné tieto súbory vytvoriť.

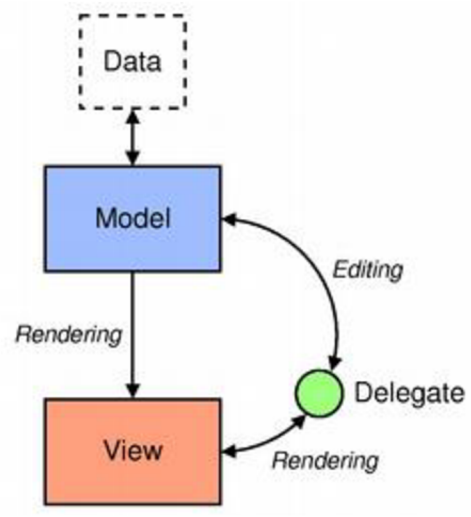
Metóda *packageWatcher(),* slúži na to, aby manipulovala so súbormi *Package.pkg*, ktoré sa nachádzajú vo vlastnosti *packageSet*. Manipuláciou myslíme vytvorenie súboru, ak ešte neexistuje v zložke, ktorá vyžaduje súbor *Package.pkg* alebo aktualizáciu xml štruktúry už existujúceho *Package.pkg* súboru.

## 4.5 Návrh grafického rozhrania aplikácie

Grafické užívateľské rozhranie poskytuje rýchlejší a užívateľsky prívetivejší spôsob editovania a tvorby konfiguračných súborov pre backend našej aplikácie. Užívateľ našej aplikácie už nemusí nastavenia backendu a stránky pre mappView vizualizácie definovať priamou úpravou xml štruktúry konfiguračného súboru, ale poskytneme mu na to vhodné nástroje.

Grafické rozhranie je naprogramované pomocou multiplatformového *frameworku Qt*, vyvíjané v jazyku C++, ktorý slúži na tvorbu grafických rozhraní v rôznych programovacích jazykoch. V programovacom jazyku *Python*, existuje API s názvom Pyside6, ktorými autormi sú taktiež Qt Company, čo môže byť záruka dlhej a oficiálnej podpory tejto knižnice. Qt taktiež ako mappView poskytuje rôzne skupiny widgetov, ktoré slúžia na interakciu s užívateľmi.

Grafické rozhrania vytvorené v Qt frameworku najčastejšie využívajú architektúru *model-view*. Tento návrhový vzor oddeľuje dátový model aplikácie od spôsobu ako sú tieto dáta prezentované užívateľovi. Toto oddelenie umožňuje zobrazovať rovnaké dáta na rôznych miestach aplikácie a dáva nám možnosť implementovať nové dátové typy bez zmeny základných dátových modelov. Na flexibilné spracovanie vstupov od užívateľa Qt predstavuje takzvaného delegáta (*delegate*). Delegát umožňuje prispôsobiť spôsob zobrazenia dát a spôsob ako tieto dáta editovať. Triedy, ktoré popisujú architektúru *model-view*, vieme rozdeliť do troch skupín - modely, zobrazenia a delegáti. Každý skupina je definovaná abstraktnou triedou, ktoré ponúkajú spoločné rozhrania a v niektorých prípadoch aj základnú implementáciu funkcionalít. Modely, zobrazenia a delegáti komunikujú medzi sebou pomocou signálov[10] a slotov[11].




Obr. 11: Architektúra model-view. [9]

[10] Signál – Notifikácia, ktorú vyvolá widget keď sa niečo udeje (napr. kliknutie na tlačidlo). [12]
[11] Slot – Slúži na zachytávanie signálov. Väčšinou sa jedná o metódu, ktorá ma nejakú funkciu [12]

**Models – Modely**

Všetky prvky modelov majú za základ triedu *QtAbstractItemModel*. Trieda *QtAbstractItemModel* definuje rozhranie, ktoré používajú zobrazenia a delegáti ku prístupu k dátam aplikácie. Toto rozhranie je natoľko flexibilné, že rozhrania ho vedia prezentovať vo forme tabuliek, zoznamov alebo vo forme stromovej štruktúry. Samotné dáta nemusia byť uložené v modeli, ale môžu byť uložené aj v súbore, databáze alebo v inom aplikačnom komponente.

**Views -Zobrazenia**

Qt ponúka kompletné implementácie, pripravené na použitie pre rôzne druhy zobrazení. *QListView* zobrazuje dátový model ako zoznam, *QTableView* ako tabuľku a *QTreeView* zobrazuje prvky dátového modelu ako stromovú štruktúru. My sme pri návrhu využili zobrazenia *QListView* a *QTableView* pre zobrazenie zoznamu stránok a konfiguráciu obsahov. Po naštudovaní si tejto architektúry sme začali s návrhom grafického užívateľského rozhrania. Najprv sme definovali požadované funkcie a po zohľadnení týchto funkcií sme navrhli vhodný grafický dizajn a dátový model.

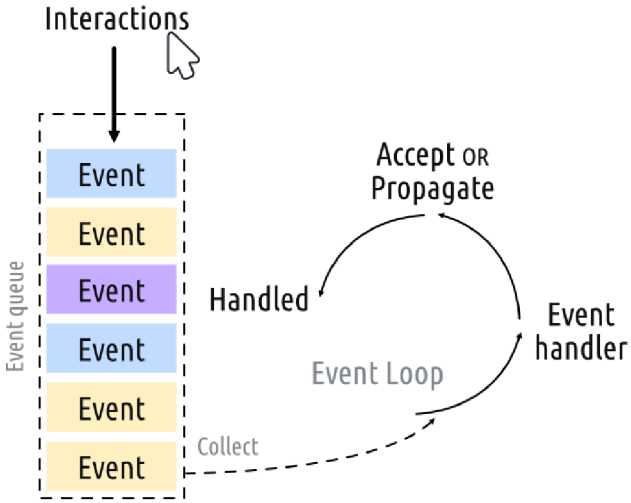
Dátový model je navrhnutý tak, aby odzrkadľoval štruktúru konfiguračného súboru. Po načítaní dát z konfiguračného súboru sa tieto dáta uložia do dátového modelu, s ktorým následne rozhranie aplikácie ďalej pracuje. Dátový model sa nachádza v objekte *ConfigModel*. Náš dátový model pozostáva z piatich položiek. Tie sa nazývajú *allowPkgWatcher* a *allowTextConfigWather* a značia či povolenie pre obsluhu súborového systému AS a obsluhu textového konfiguračného súboru konfigurácie stroja.

Ďalšie predstavujú cestu v súboroch počítača samotného konfiguračného súboru, s ktorým rozhranie pracuje a cestu ku aktuálnej konfigurácií stroja.

Poslednou a najdôležitejšou je zoznam stránok. Aby nami zvolené zobrazenie mohlo komunikovať s naším dátovým modelom, museli sme zoznam stránok definovať pomocou objektu *QStandartItemModel*. Tento model je univerzálny a orientuje sa na samotné položky modelu. Model ku jednotlivým položkám pristupuje pomocou ich indexu. Aby sme mohli ukladať do tohto modelu informácie o stránkach, čiže objekty typu *Page*, museli sme reimplentovať triedu *QStandardItem*, ktorá popisuje správanie sa položky modelu *QStandartItemModel*.

**Popis rozhrania**

Po importovaní všetkých potrebných knižníc sme sa mohli pustiť do tvorby rozhranie pre našu aplikáciu. Základom každej Qt aplikácie je inštancia triedy *QApplication*. Tento objekt drží takzvaný cyklus udalostí (*event loop)*. Tento cyklus udalostí riadi všetky interakcie používateľa rozhrania. Každá interakcia s aplikáciou či už stlačenie klávesy, kliknutie myšou na tlačidiel, vygeneruje udalosť, ktorá sa vloží do rady udalosti. V cykle sa pri každej iterácií skontroluje táto rada, a ak sa nájde nejaká udalosť, ktorú treba obslúžiť, vyvolá sa funkcia, ktorá obslúži túto udalosť. Po vysporiadaní sa danou udalosťou sa kontrola nad programom vráti opäť k cyklu udalosti [10]




Obr. 12: Cyklus udalosti. [10]

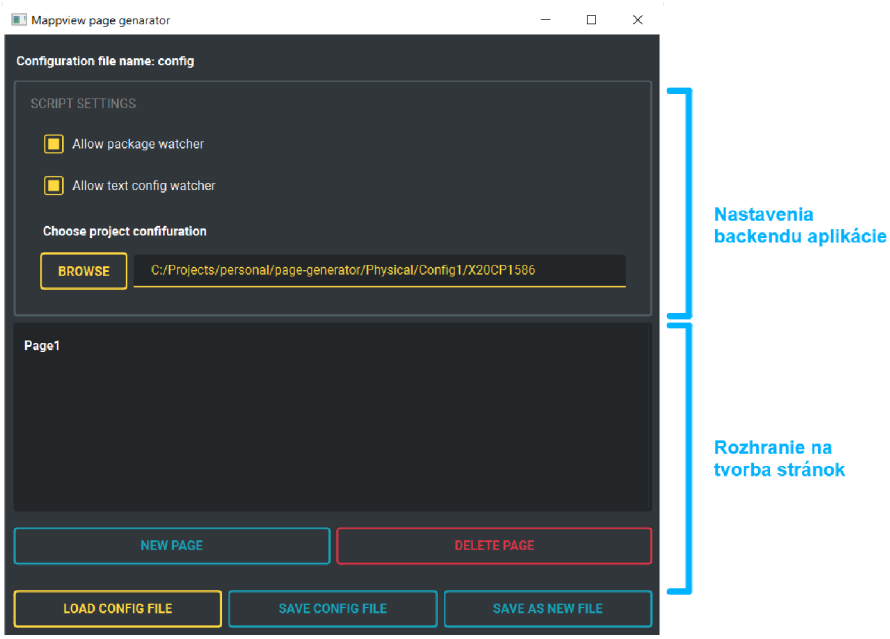
Naše grafické rozhranie pozostáva z troch častí. Prvá slúži na úpravu nastavení skriptu, druhá sa venuje tvorbe stránok a tretia slúži na zobrazenie a úpravu detailu stránky. Tieto tri časti bližšie popíšeme v nasledujúcich odstavcoch.

Ako widget najvyššej úrovne, ktorý bude zaobaľovať všetky ostatné widgety, Qt ponúka preddefinovanú triedu *QMainWidnow*, ktorá predstavuje zobraziteľné grafické okno. My sme túto triedu použili ako základ pre našu triedu, ktorá predstavuje hlavné okno našej aplikácie s názvom *MainWindow*. Trieda *MainWindow* rozširuje základnú triedu o nami definované vlastnosti a metódy. Metódy tejto triedy vieme rozdeliť na metódy, ktoré generujú prvky rozhrania metódy a metódy, ktoré slúžia ako sloty. Napríklad metóda *scriptSettingsSection()* generuje widgety, z ktorých sa skladá prvá časť nášho rozhrania a tou sú nastavenie backendu našej aplikácie. Časť s nastaveniami backendu sa skladá z dvoch zaškrtávacích polí a jedného textové vstupu. Funkcia zaškrtávacích polí je, že graficky zobrazujú logickú pravdu a nepravdu. Na vytvorenie týchto zaškrtávacích polí slúžia Qt objekty typu *QCheckBox*. My sme tieto zaškrtávacie polia napojili na náš dátový model, ktorý obsahuje hodnoty, ktoré povoľujú obsluhu súborového systému AS a obsluhu textových konfiguračných súborov. Tento dátový model predstavuje vlastnosť model, ktorej dátový typ je objekt *ConfigModel*. Dátovému modelu sa ale budeme venovať v inej časti našej práce. Prepojenie na dátový model prebieha pomocou signálu *toggled* a príslušných slotov (napr. *allowPkgWatcherCheckBoxToggled*), ktorý uloží novú hodnotu získanú od užívateľa do nášho dátového modelu.

Ďalší prvok, ktorý sa nachádza v nastaveniach backendu je dialógové okno na výber aktuálnej zložky konfigurácie stroja v AS. Dialógové okno reprezentujte objekt *QFileDialog* a slúži na zobrazenie a možnú manipulácií so súbormi nášho počítača. Tento objekt tiež obsahuje vlastnosti a metódu, ktoré slúžia na bližšiu špecifikáciu súborov, ako napríklad či ide o súbor alebo zložku až po konkrétnu príponu súboru. Na vyvolanie tohto okna slúži tlačidlo *BROWSE*. Tlačidlá a ich správanie sú definované objektom *QPushButton*. Objekt *QPushButton* obsahuje signál s názvom *click*, ktorý vyvolá slot po

kliknutí na dané tlačidlo. V našom prípade vyvolá slot s názvom *configDirDialog*, ktorá vyvolá spomenuté dialógové okno na výber zložky. Potom, čo užívateľ úspešne zvolí zložku aktuálnej konfigurácie stroja, sa cesta ku tejto zložke zapíše do dátového modelu a z dátového modelu sa načíta cesta na textový výstup. Textový výstup ukazuje cestu ku zvolenej zložke a slúži len na čítanie. Textové vstupy sa vytvárajú pomocou objektov *QLineEdit*.

Rozhranie na tvorbu stránok sa nachádza hneď pod nastaveniami skriptu. Užívateľ po načítaní konfiguračného súboru do dátového modelu môže jednotlivé stránky upravovať, vytvárať nové alebo ich mazať. Prostredie, kde sa jednotlivé stránky zobrazujú, sa vytvorili prostredníctvom implementácie *QlistView* a vytvorením vlastného delegáta na editáciu detailu stránky. *QlistView* nám ponúka pristupovať ku dátovému modelu ako ku zoznamu prvkov. V našom prípade sme toto zobrazenie použili na zobrazenie zoznamu stránok, ktoré užívateľ získal po načítaní konfiguračného súboru alebo užívateľ vytvoril nový zoznam stránok, ktorý sa ma exportovať. Toto zobrazenie taktiež ponúka rôzne preddefinované metódy a vlastnosti ku prístupu jednotlivým prvkom. Napríklad označením nejakého prvku, nám toto zobrazenie ponuka index prvku, ktorý využívame pri úprave alebo mazaní dát z dátového modelu.

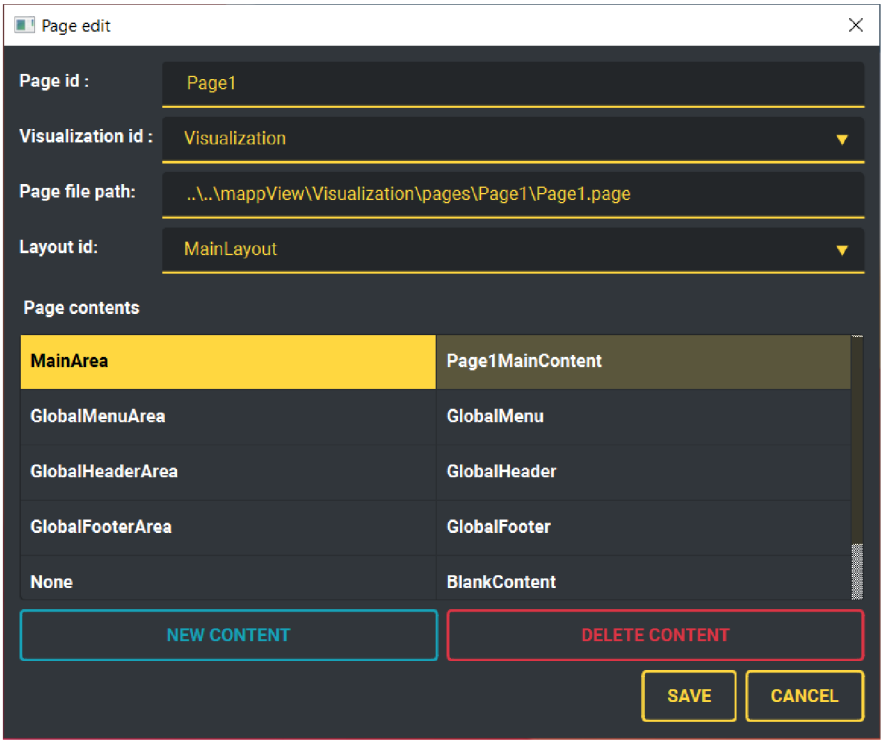



Obr. 13: Hlavné okno grafického rozhrania aplikácie.

Otvorenie dialógového okna na editovanie detailu stránky sa zobrazí po dvojitým klikom na určitá stránku. Na vytvorenie vlastného delegáta sme museli vytvoriť novú triedu s názvom *PageListViewItemDelagate*, ktorá dedí vlastnosti a metódy od abstraktnej Qt triedy *QStyledItemDelegate*.

Trieda *QStyledItemDelegate* ponúka 3 triedy, ktoré definujú správanie editora dát tohto zobrazenia. Prvou je *createEditor()*, ktorá popisuje aký widget alebo prvok rozhrania sa má použiť na editáciu dát. Druhou je *setEditorData()*, ktorá vytvorí rozhranie na úpravu dát a nastaví požadované hodnoty na vstupy editora, ktoré chceme upravovať.

Poslednou je metóda *setModelData()*, ktorá po prevzatí signálu, ktorý predstavuje potvrdenie zmeny, uloží nové hodnoty do dátové modelu a aktualizuje grafické rozhranie aplikácie. Všetky tieto tri zdedené metódy sme museli prepísať, aby spĺňali naše požiadavky. Ako editor detailu stránok sme použili nami vytvorené dialógové okno, ktoré je na to určené. Toto dialógové okno predstavuje objekt typu *PageDialog*. Taktiež slúži na vytvorenie novej stránky. Po stlačení tlačidla *NEW PAGE*, ktoré sa nachádza pod zoznamom stránok, sa otvorí okno s prázdnymi vstupmi a užívateľ si môže vytvoriť novú stránku. Ak užívateľ chce zmazať konkrétnu stránku musí ju najprv označiť v zozname stránok a vymazať ju stlačením tlačidla *DELETE PAGE*.




Obr. 14: Dialógové okno na editáciu stránok

V dialógovom okne na úpravu alebo vytvorenie stránky sa nachádzajú všetky potrebné vstupy na správnu definíciu stránky. Okno obsahuje textový vstup na identifikátor stránky, dva rozbaľovacie zoznamy, ktoré poskytujú identifikátory vizualizácií a rozložení, ktoré existujú v aktuálnom projekte AS a jeden textový výstup, ktorý ukazuje cestu k danej stránke. Na vytvorenie rozbaľovacích zoznamov slúži Qt trieda *QComboBox*. Na vyplnenie týchto zoznamov boli navrhnuté statické metódy triedy *PageManager* s názvami *getProjectLayouts* a *getVisFiles*. Tieto metódy boli bližšie opísane v kapitole 4.4.1 v odstavci PageManager.

Na zobrazenie a úpravu obsahov, z ktorých stránka pozostáva sme použili rozloženie s názvom *QTableView*. V prvom stĺpci tabuľky sa nachádza identifikátor oblasti rozloženia, v druhom stĺpci sa nachádza obsah, ktorý sa má vložiť do danej oblasti. Aby sme užívateľovi zjednodušili úpravu obsahov, navrhli sme rozhranie tak, aby sa po kliknutí na bunky z prvého stĺpca zobrazil rozbaľovací zoznam, ktorý obsahuje oblasti zvoleného rozloženia. V bunkách druhého sa zase nachádza textový vstup pre názov

obsahu. Toto prispôsobenie tabuľky sme taktiež museli spraviť pomocou úpravy vstaveného delegáta pre objekt *QTableView*. Aby nedochádzalo k problémom s oblasťami rozloženia, sa vždy po voľbe nového rozloženia tabuľka vyčistí od definovaných hodnôt.

Tlačidlom *NEW CONTENT* sa pridá nový riadok do tabuľky, ktorý predstavuje dvojicu oblasť rozloženia a obsah stránky. Po označení akékoľvek bunky riadku a nasledovnom stlačení tlačidla *DELETE CONTENT* prebehne zmazanie riadku z tabuľky. Po úspešnej úprave alebo vytvorení stránky si užívateľ môže vybrať či chce požadovaná zmenu uložiť alebo zahodiť. To prebieha pomocou tlačidiel *SAVE* alebo *CANCEL*. Tieto tlačidlá po stlačení pošlú signál s odpoveďou metódy *setModelData()* objektu *QListView*, ktorá uloží alebo zahodí zmeny dátového modelu nášho rozhrania.

Importovanie a exportovanie konfiguračného súboru z dátového modelu prebieha pomocou skupiny tlačidiel, ktoré sa nachádzajú v spodnej časti hlavného okna aplikácie. Na každé tlačidlo zo skupiny je na signál *click* napojená odpovedajúca metóda. Na tlačidlo s názvom *LOAD CONFIG* je napojená metóda *loadConfigDialog()*. Táto metóda vytvorí dialógové okno na výber súboru zo súborového systému počítača a zo zvoleného konfiguračného súboru načíta xml štruktúru, ktorú následne transformuje na dátový model aplikácie. Tlačidlo s názvom *SAVE CONFIG* slúži na uloženie zmien vybraného konfiguračného súboru. Na uloženie týchto zmien sme navrhli metódu *saveConfig()*, ktorá z dátového modelu zostaví novú xml štruktúru konfiguračného súboru a potom ju uloží. Na export a uloženie xml štruktúry sa využíva metóda *createXmlTree()* a *saveFile()* objektu *ConfigFile*. Aby sa uložené zmeny prejavili aj v rozhraní aplikácie musíme znova načítať konfiguračný súbor, ale už s novou štruktúrou. Ak by užívateľovi aplikácie nestačilo používať jeden konfiguračný súbor, pridali sme do aplikácií možnosť exportovať konfiguračný súbor ako nový súbor s menom, ktoré mu definoval užívateľ. Táto funkcia prebehne stlačením tlačidla *SAVE AS NEW FILE*. Po stlačení tlačidla sa zobrazí dialógové okno, kde si užívateľ môže vybrať zložku, kam sa má tento nový konfiguračný súbor exportovať a môže mu definovať akékoľvek meno. Export a uloženie prebieha identicky, ako sme ho popísali pri obyčajnom uložení súboru.

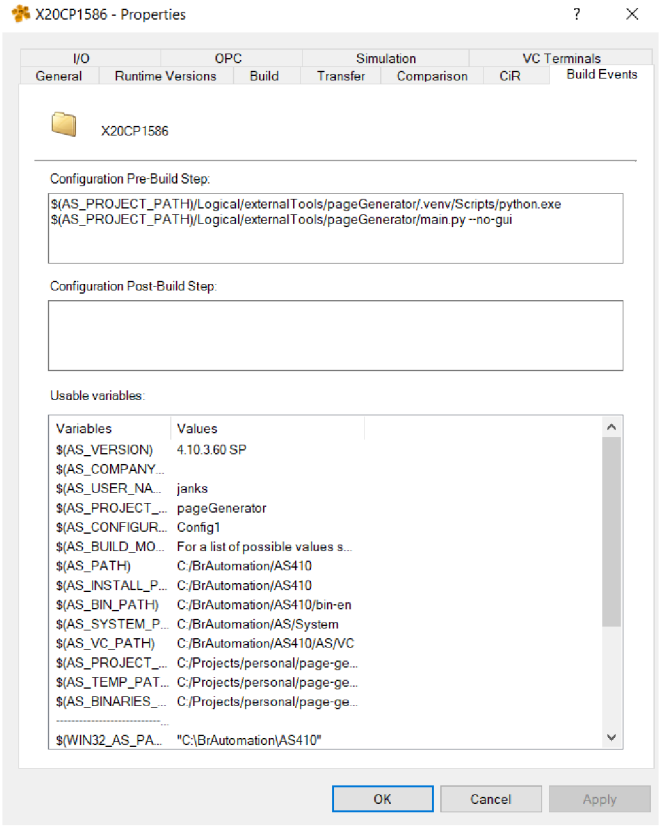
## 4.6 Implementácia do vývojového prostredia B&R Automation Studio

Vývojové prostredie B&R Automation studio obsahuje nástroje na prácu s príkazovým riadkom. Túto výhodu sme využili, pretože vďaka tomu sme schopní automaticky volať príkazy na spustenie našej aplikácie vždy pred zostavením projektu a užívateľ aplikácie nemusí používať externý príkazový riadok. Nastavenie príkazov príkazového riadka AS nájdeme v zložke - *Project → Change runtime version → Builds event*. Okno obsahuje vstupy na príkazy príkazového riadka AS, ktoré sa majú spustiť v špecifickej fáze zostavovania projektu. Taktiež okno obsahuje pomocné premenné (napr. cestu ku aktuálnemu projektu) na tvorbu príkazov .

Príkazy, ktoré sa majú vždy spustiť predtým, ako sa zostaví projekt, sa zapisujú do textového poľa s názvom *Configuration Pre-build Step* ako je zrejme z obrázku 15. Zápis príkazu je nasledovný :

- **$(AS_PROJECT_PATH)/Logical/externalTools/pageGenerator/.venv/Scripts/python.exe $(AS_PROJECT_PATH)/Logical/externalTools/pageGenerator/main.py --no-gui**

Prvý riadok príkazu definuje aký Python interpreter má využívať naša aplikácia. Druhý riadok spustí hlavný zdrojový súbor nášho projektu, ktorý sa nachádza v súbore *main.py*. Argument *–no-gui*, slúži na spustenie backendu aplikácie.




Obr. 15: Konfigurácia zostavovacích udalostí projektu AS.

# 5 POUŽÍVANIE APLIKÁCIE

V tejto kapitole ukážeme a popíšeme ako našu aplikáciu využiť pri programovaní automatizačných vizualizácií technológie mappView vo vývojom prostredí B&R Automation studio. Ukážka zahrňuje používanie našej aplikácie a jednoduchý demonštračný program vizualizácie.

Správne fungovanie našej aplikácie vyžaduje určité predpoklady, ktoré musia byť splnené. Tieto predpoklady sú:

- Zložka s aplikáciou sa musí nachádzať v druhej úrovni podadresáru *Logical* (*Logical View*). V našom projekte sa to nachádza v zložke *externalTools/pageGenerator*.
- Užívateľ našej aplikácie si musí nainštalovať všetky potrebné knižnice, ktoré vyžaduje naša aplikácia. Knižnice a ich verzie obsahuje súbor s názvom *pyproject.toml* a príkazu na inštaláciu sú popísane v kapitole 4.2.

Spustenie backendu aplikácie môže prebiehať dvoma spôsobmi. Automatickým volaním pred zostavovaciou udalosťou projektu AS alebo manuálnym volaním cez príkazový riadok. Návod ako integrovať automatické volanie backendu aplikácie je popísane v kapitole 4.6 . Pri používaní príkazového riadka je samozrejmosťou, aby sa užívateľ nachádzal v pracovnom adresári, ktorý obsahuje zdrojové súbory. Ak užívateľ aplikácie nechce z nejakého dôvodu implementovať automatické volanie skriptov, môže náš backend aplikácie spustiť cez príkazový riadok príkazmi:

- **Poetry shell** – Aktivácia interaktívneho príkazové riadku Poetry.
- **Poetry run python.exe main.py --gui** – Spustenie grafického rozhrania na úpravu konfiguračného súboru.

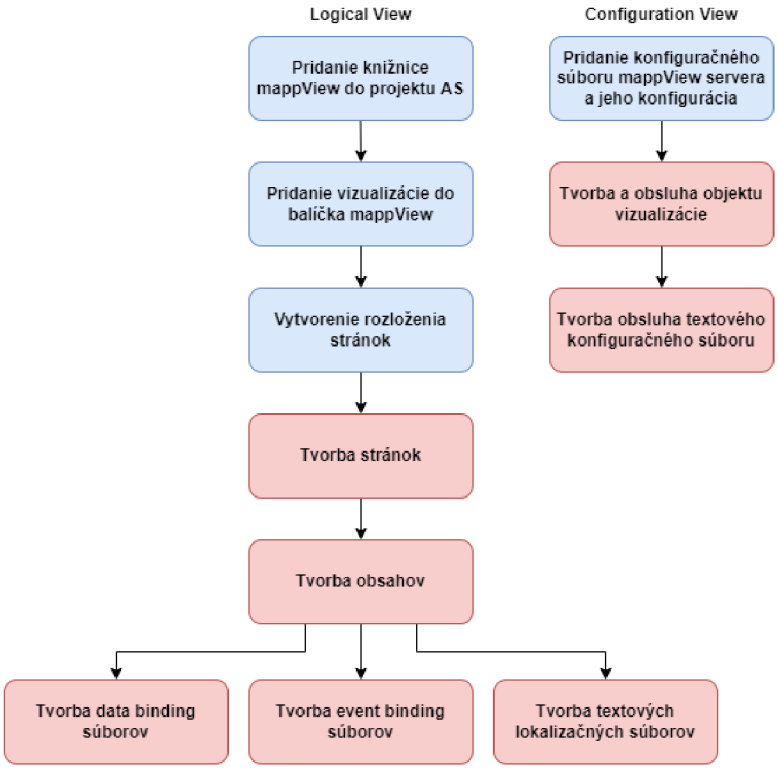
Grafické rozhranie aplikácie otvoríme dvojklikom priloženého batch súboru s názvom *GUI.bat*. Súbor obsahuje zoznam príkazov, ktoré sa majú postupne vykonať.

- **start %~dp0\\.venv\Scripts\python.exe main.py –gui** – Spustenie grafického rozhrania na úpravu konfiguračného súboru pomocou interpreteru nachádzajúceho sa v virtuálnom vývojovom prostredí.

Rovnako ako backend aplikácie aj grafické rozhranie aplikácie vieme spúšťať cez príkazový riadok pomocou príkazu:

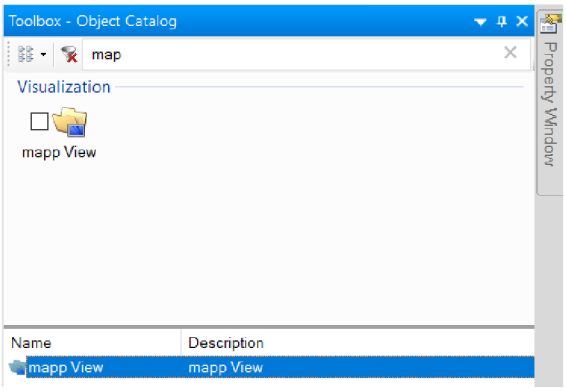
- **Poetry run python.exe main.py –no-gui** – Spustenie backendu aplikácie, ktorý spracuje konfiguračný súbor.

Ako už bolo spomenuté, naša aplikácia má zjednotiť prácu pri vývoji mappView vizualizácií. Na obrázku 16. je znázornený postup krokov vývoja vizualizácie. Červenou farbou sú znázornené kroky, ktoré backend našej aplikácie spraví automaticky za nás. Na jednoduchom príklade si ukážme všetky kroky vývoja s použitím našej aplikácie.

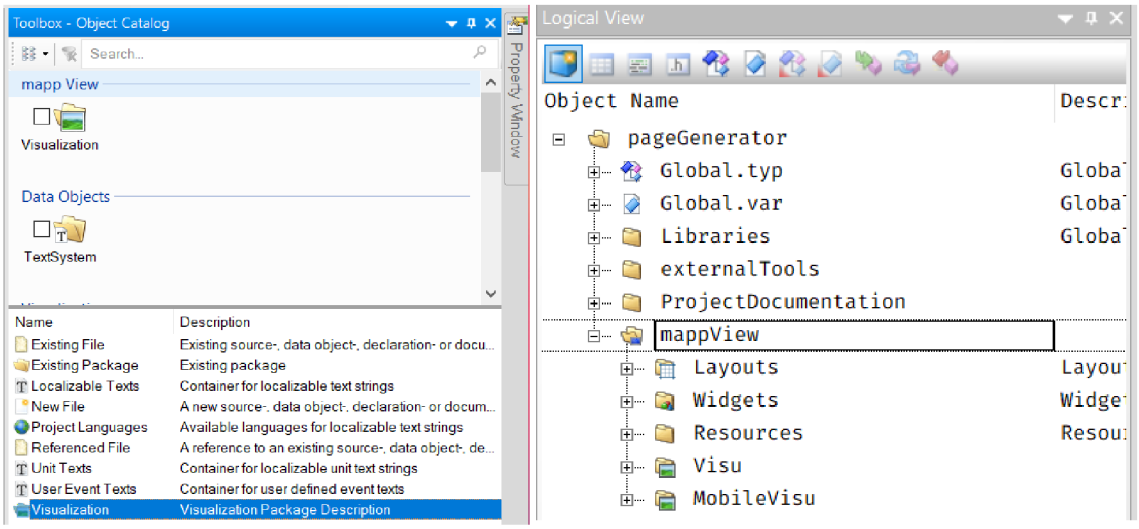



Obr. 16: Algoritmus tvorby mappView vizualizácií.

Knižnica mappView sa vkladá do projektu AS pomocou panelu katalóg objektov, dvojitým kliknutím na položku mappView alebo ale pretiahnutím do *Logical View*. Ak sa tam mappView nenachádza, je možné, že ešte táto knižnica nie je nainštalovaná vo vývojovom prostredí. Inštalácia knižníc prebieha cez okno *Tools* → *Upgrades* alebo cez oficiálne stránky B&R Automation. Ak máme mappView vložené v projekte, je na rade vytvoriť konkrétnu vizualizáciu. My sme vytvorili jednu vizualizáciu na širokom uhle obrazovky a jednu určenú pre mobilné zariadenia. Tvorba vizualizácie taktiež prebieha cez panel katalóg objektov. Položka vizualizácie sa volá *Visualization* a objaví sa až po označení zložky mappView v *Logical view*. Vizualizácie sme pomenovali podľa potreby. Vizualizácia s názvom *Visu* je určená pre širokouhlé obrazovky a *Mobile* pre mobilné zariadenia. Finálny výsledok vidieť na obrázku 18.

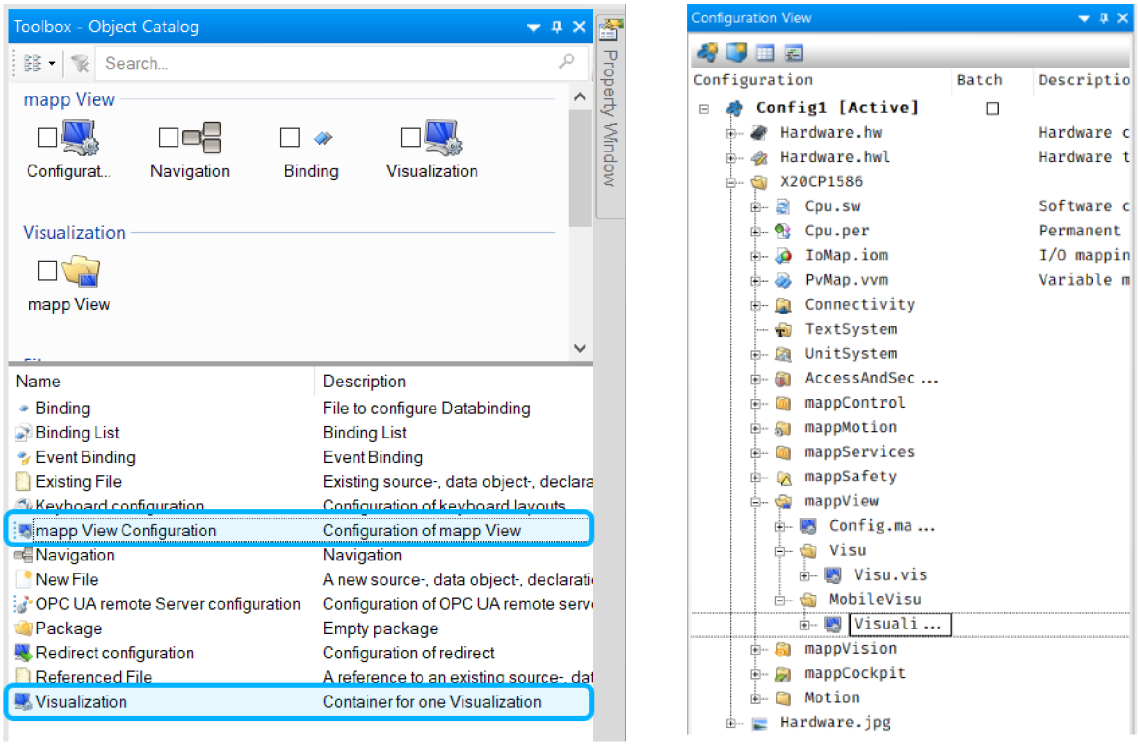



Obr. 17: Katalóg objektov.



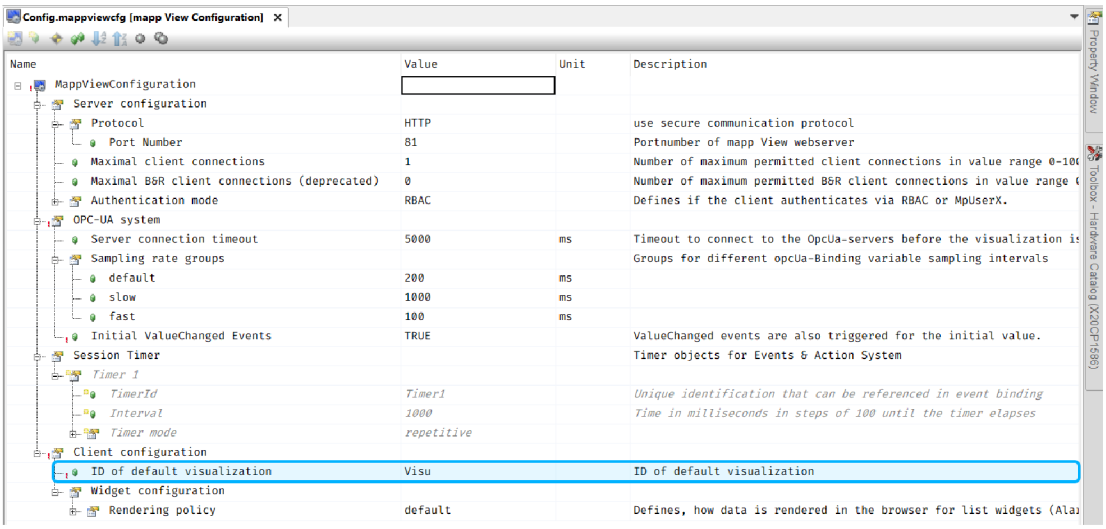

Obr. 18: Priadanie mappView knižnice do projektu AS.

V *Configuration View* je nutné vložiť konfiguračný súbor mappView servera taktiež cez katalóg objektov a môžeme tam vložiť aj zložky, ktoré oddelenia súbory jednotlivých vizualizácií, do ktorých sa následne vloží objekt vizualizácie. Jeden pre každú vizualizáciu. Ak sa objekty vizualizácie pridávajú manuálne, je potrebné im prideliť identifikátor vizualizácie, ku ktorej patria. Manuálne pridanie objektu vizualizácie nie je nutné, pretože to vie spraviť aj náš automatický skript.



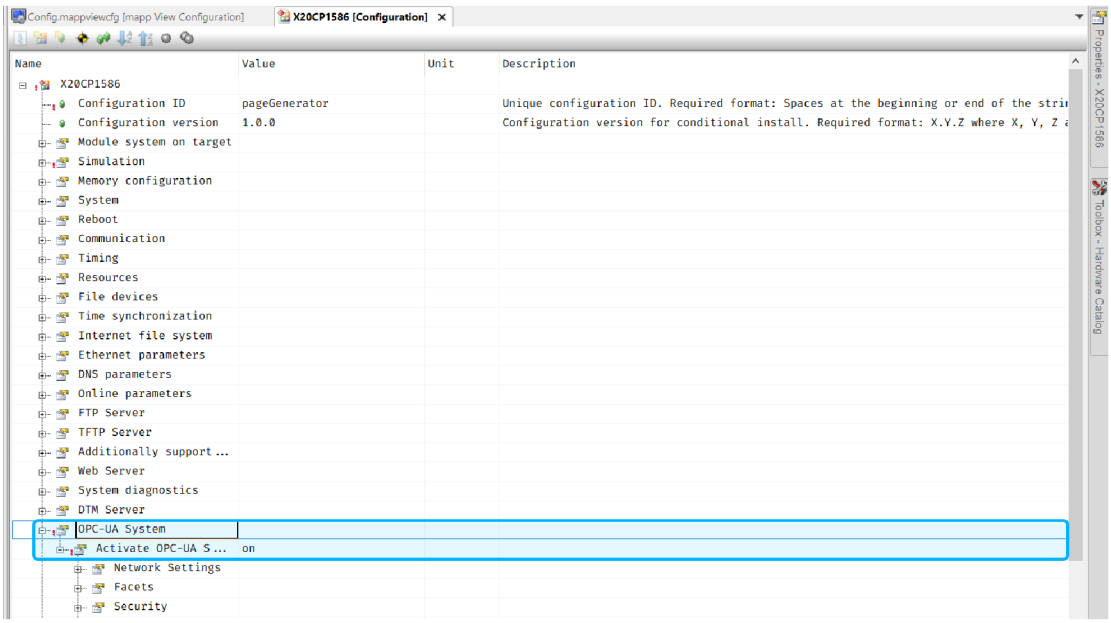

Obr. 19: Pridanie konfiguračných súborov.

Ďalším krokom je konfigurácia mappView servera a OPC UA systému na cieľovom zariadení. V našom prípade je to PLC X20CP1586. Konfigurácia mappView servera prebieha v súbore s príponou *.mappviewcfg*. Základná konfigurácia je veľmi jednoduchá a pozostáva z nastavenia predvolenej vizualizácie. Ku tejto vizualizácií sa bude dať pripojiť cez internetový prehliadač pomocou URL adresy - *IpAdresaZariadenia:81/index.html*. Ku ostatným vizualizáciám sa pristupuje podobne, len s tým rozdielom, že treba špecifikovať identifikátor objektu vizualizácie napríklad *IpAdresaZariadenia:81/index.html?visuId=Mobile*.




Obr. 20: Konfigurácia mappView servera.

Konfigurácia OPC UA systéme prebieha v konfigurácia cieľového zariadenia. K tejto konfigurácií sa dostaneme cez *Physical View*, kde pravým kliknutím myšou na cieľové zariadenie sa nám otvorí zoznam s možnosťami. Zo zoznamu vyberieme položku *Configuration*. Na fungovanie mappView servera stačí aktivovať OPC UA systém.

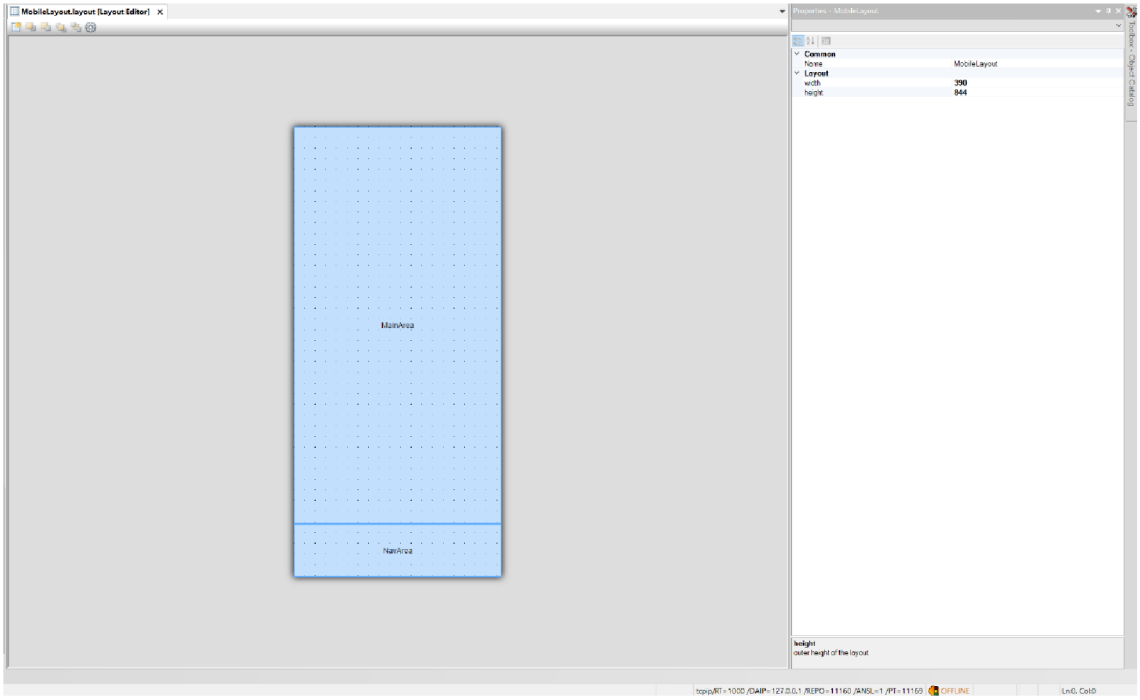



Obr. 21: Konfigurácia OPC UA systému na cieľovom zariadení.

Jazyková konfigurácia mappView vizualizácie prebieha na dvoch miestach. Prvým je jazykový konfiguračný súbor, ktorý sa vkladá do *Logical View* pomocou katalógu objektov. Tento súbor ma príponu *.language* a obsahuje jazykovú konfiguráciu celého projektu nie len mappView vizualizácie. V súbore sme ponechali predvolenú jazykovú konfiguráciu, ktorá pozostáva z anglického, nemeckého a francúzskeho jazyka. Druhým miestom je textový konfiguračný súbor, ktorý sa nachádza v *Configuration View* v zložke *TextSystem*. Ako bolo spomenuté v teoretickej časti, tento súbor obsahuje jazyky a textové lokalizačné súbory, ktoré sa majú preniesť a používať na cieľovom zariadení. Textový konfiguračný súbor nie je potrebné manuálne vytvoriť, pretože to dokáže aj náš backened aplikácie, avšak, programátorovi nič nebráni si ho manuálne vytvoriť a backend aplikácie len aktualizuje jeho obsah,

V tomto štádiu už sme mali všetko správne nakonfigurované a mohli sme sa pustiť do tvorby stránok pre naše vizualizácie. Ešte predtým ako sme využili našu aplikáciu page-generator sme museli vytvoriť rozloženia stránok v grafickom editore AS. Tieto rozloženia stránok budú potom prístupne v aplikácií page-generator pri tvorbe stránok

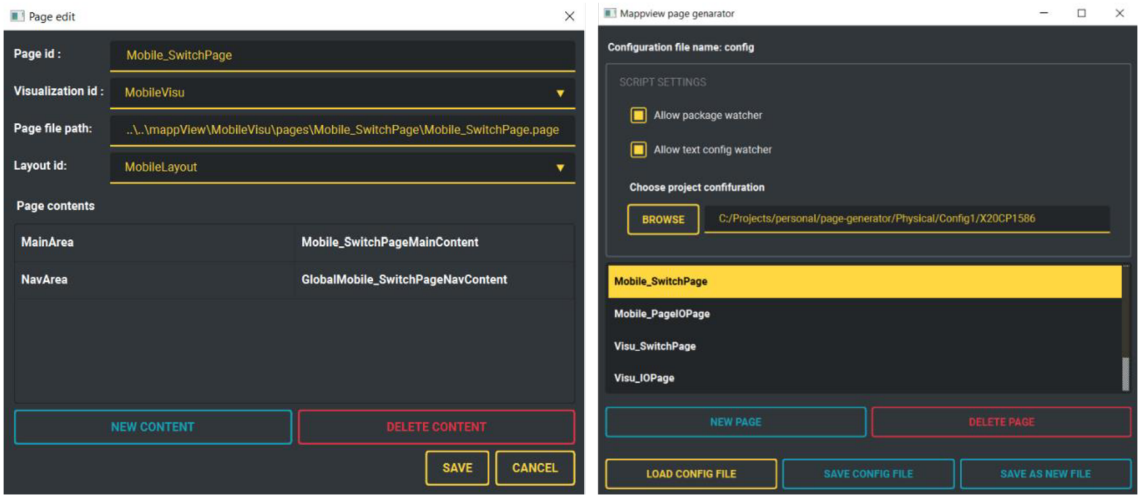
Ku každému rozloženiu pristupujeme lokálne. To znamená, že rozloženie sa nachádza v zložke *Layouts* vizualizácie, ku ktorej patrí. Nami navrhnuté rozloženia pozostávajú z dvoch časti. Jedná oblasť pre navigáciu a druhá je hlavná oblasť, kde sa bude zobrazovať obsah danej stránky.



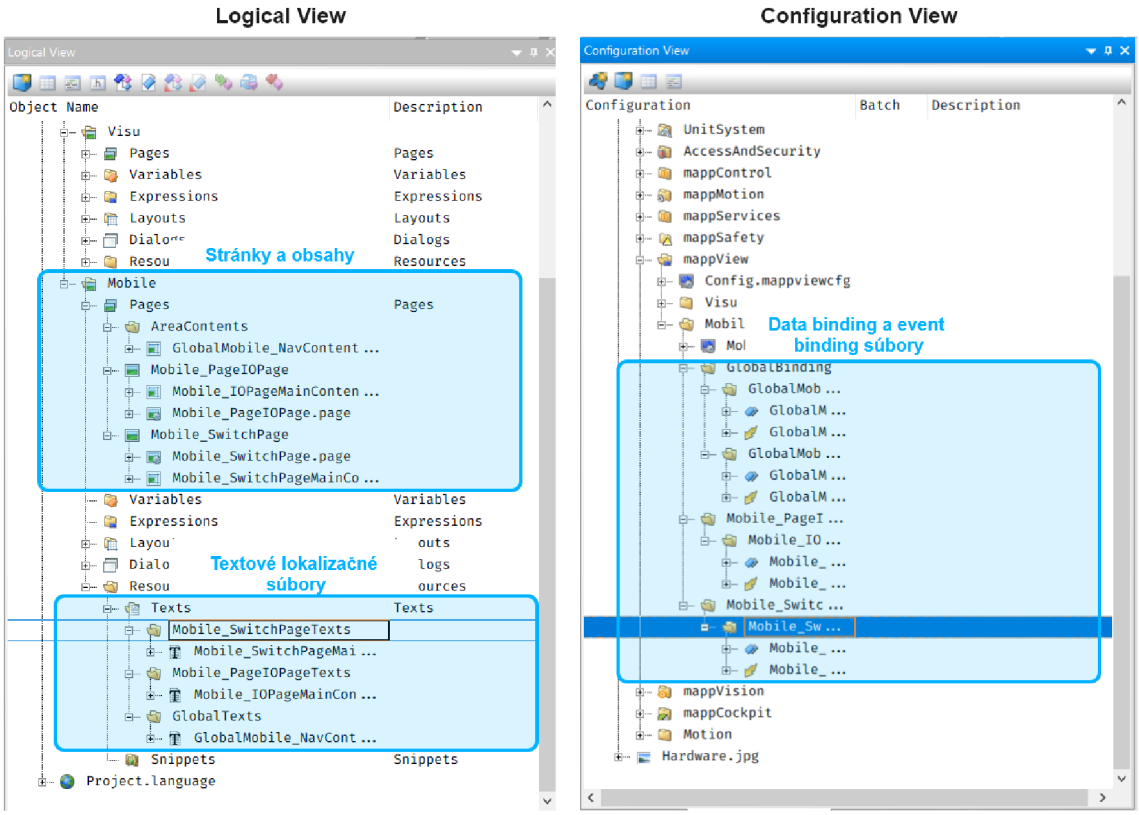

Obr. 22: Rozloženie stránky pre mobilné zariadenia.

Na tvorbu stránok, obsahov a príslušných súborov využijeme našu aplikáciu. V aplikácii page-generator sme definovali štyri stránky na generovanie. Jednotlivým stránkam sme priradili vizualizáciu, názov a rozloženie. Podľa rozloženia sme nasledovne každej oblasti podľa potrieb priradili obsah. Celú konfiguráciu stránok sme vyexportovali do

konfiguračného súboru, ktorý potom spracoval náš backend aplikácie. Backend aplikácie sme volali automaticky pred zostavovanou udalosťou v AS. Backend aplikácie vygeneroval štyri súbory so stránkami, šesť súborov s obsahmi, šesť *data binding* súborov, šesť *event binding* súborov, šesť textových lokalizačných súborov. V súčte teda backend vygeneroval dvadsaťosem súborov, ktoré následne referencoval do konfiguračných súborov podľa potrieb mappView vizualizácie.




Obr. 23: Tvorba stránok v aplikácií page-generator.




Obr. 24: Vygenerované súbory.

```
<?xml version='1.0' encoding='UTF-8'?>
<vdef:Visualization xmlns:vdef="http://www.br-automation.com/iat2015/visualizationDefinition/v2" id="Mobile">
  <StartPage pageRefId="Mobile_SwitchPage"/>
  <Pages>
    <!--<Page refId=""/>-->
    <Page refId="Mobile_PageIOPage"/>
    <Page refId="Mobile_SwitchPage"/>
  </Pages>
  <Navigations>
    <!--<Navigation refId="" />-->
  </Navigations>
  <BindingsSets>
    <!--<BindingsSet refId="" />-->
    <BindingsSet refId="GlobalMobile_NavArea_binding"/>
    <BindingsSet refId="Mobile_IOPageMainContent_binding"/>
    <BindingsSet refId="Mobile_SwitchPageMainContent_binding"/>
    <BindingsSet refId="GlobalMobile_NavContent_binding"/>
  </BindingsSets>
  <EventBindingsSets>
    <!--<EventBindingsSet refId="" />-->
    <EventBindingsSet refId="GlobalMobile_NavContent_eventbinding"/>
    <EventBindingsSet refId="Mobile_SwitchPageMainContent_eventbinding"/>
    <EventBindingsSet refId="GlobalMobile_NavArea_eventbinding"/>
    <EventBindingsSet refId="Mobile_IOPageMainContent_eventbinding"/>
  </EventBindingsSets>
  <Dialogs>
    <!--<Dialog refId="" />-->
```

Obr. 25: Ukážka súboru objektu mobilnej vizualizácie.

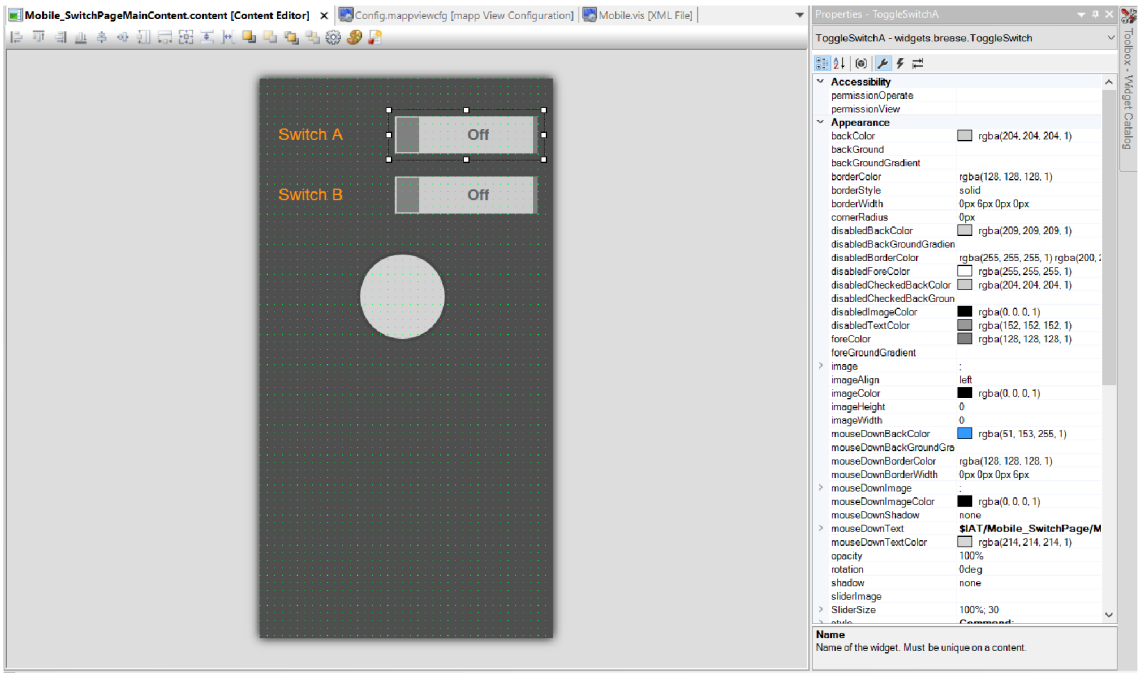
Po vygenerovaní všetkých súborov sme začali návrhom demonštračných obsahov. Do obsahov sme vložili prvky, ktoré majú simulovať schodišťové zapojenie spínačov a žiarovku. Tieto prvky sú dve prepínacie tlačidlá (*ToggleSwitch*) a grafický prvok elipsa (*Ellipse*), ktorá bude signalizovať vypnutú a zapnutú žiarovku.

Každému textu, ktorý sa nachádza v obsahu stránky sme vytvorili samostatné textové pole v príslušnom textovom lokalizačnom súbore. Identifikátor textového poľa bol následne priradený popisom a tlačidlám pomocou vlastností *text* a *mouseDownText*. Tieto vlastnosti dokážu zobrazovať texty v rôznych jazykoch na základe zvoleného jazyka vo vizualizácii.

Textsystem Configuration: TextConfig

| Name | Value | Description |
|---|---|---|
| TextConfig | | |
| Languages | | |
| System language | en | |
| Fallback language | de | |
| Target languages | | |
| Target language 1 | en | |
| Target language 2 | de | |
| Target language 3 | fr | |
| *Target language 4* | | |
| Tmx files for target | | |
| Tmx file 1 | mappView.MobileVisu.Resources.Texts.Mobile_PageIOPageTexts.Mobile_IOPageMainContentTexts.tmx | |
| Tmx file 2 | mappView.MobileVisu.Resources.Texts.Mobile_SwitchPageTexts.Mobile_SwitchPageMainContentTexts.tmx | |
| Tmx file 3 | mappView.Visu.Resources.Texts.GlobalTexts.GlobalVisu_NavContentTexts.tmx | |
| Tmx file 4 | mappView.Visu.Resources.Texts.Visu_IOPageTexts.Visu_IOPageMainContentTexts.tmx | |
| Tmx file 5 | mappView.Visu.Resources.Texts.Visu_SwitchPageTexts.Visu_SwitchPageMainContentTexts.tmx | |
| Tmx file 6 | mappView.MobileVisu.Resources.Texts.GlobalTexts.GlobalMobile_NavContentTexts.tmx | |
| *Tmx file 7* | | |

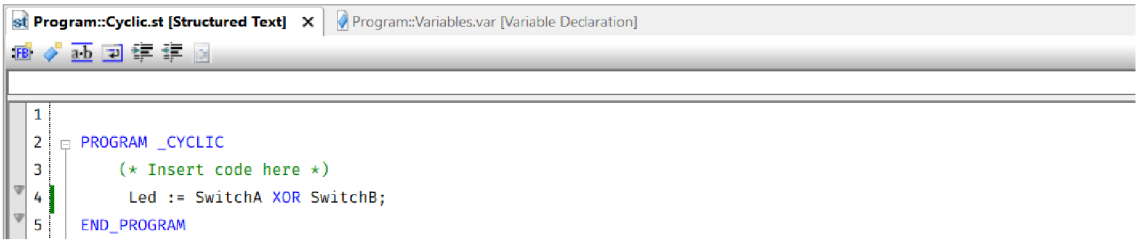
Obr. 26: Ukážka textového konfiguračného súboru.




Obr. 27: Návrh obsahu na simuláciu schodišťového zapojenia spínačov.

Vizualizácia ma zmysel vtedy keď niečo riadi alebo je riadená programom. Nami navrhnutý program bude na základe vstupov získaných z vizualizácie ovládať program, ktorý je znázornený na obrázku 27. Výsledok tejto logickej operácie sa uloží do premennej *Led*. Na základe pravdivostnej hodnoty premennej *Led* sa bude pomocou udalostí meniť farba prvku, ktorý signalizuje žiarovku.

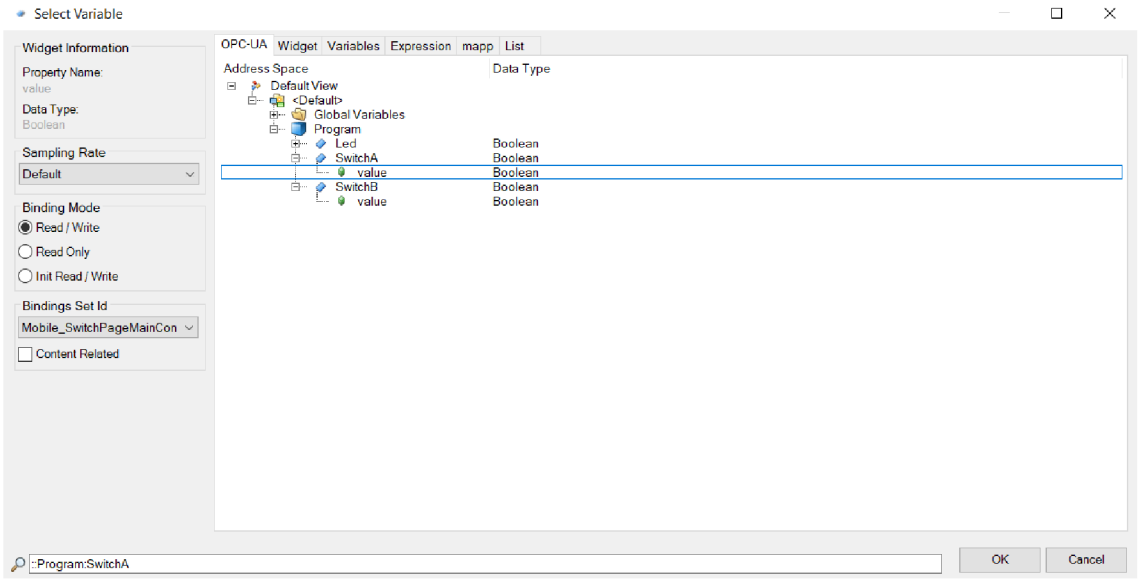
Program je naprogramovali v jazyku ST, ktorý sa najviac podobá bežným textovým programovacím jazykom a vychádza z jazyka Pascal. Jednotlivé programy sa nachádzajú v programových jednotkách, ktoré sa nazývajú *Task*. Týmto programovým jednotkám sa následne v softwarovej konfigurácií pridelí podľa požiadaviek cyklická trieda, v ktorej majú bežať. My sme zvolili predvolenú 100ms cyklickú triedu. Program




Obr. 28: Program schodišťového zapojenia spínačov.

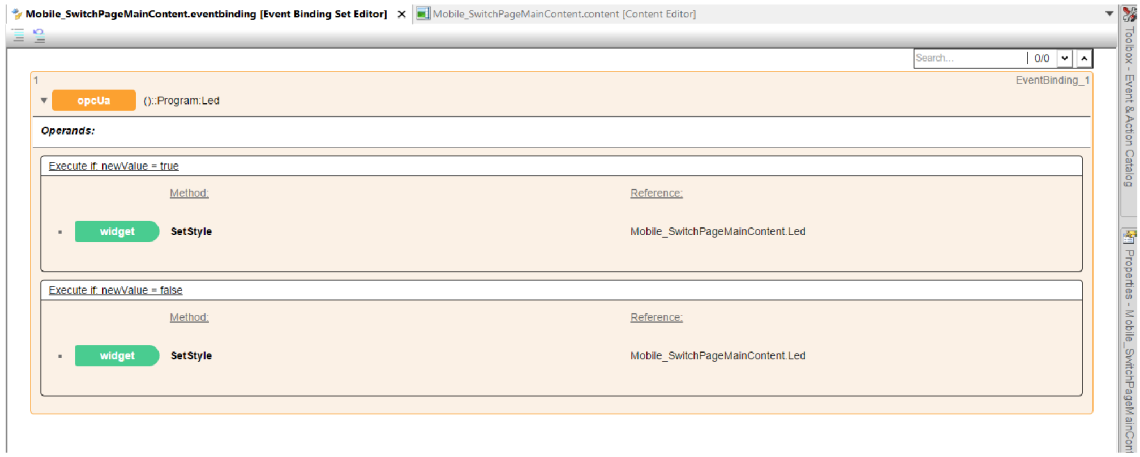
Predtým ako sme začali tvoriť *data binding* štruktúry sme museli povoliť premenné programu, ktoré majú byť prístupne protokolu OPC UA na výmenu dát. To sa robí v súbore *OpcUaMap.uad*, ktorý sa nachádza v zložke *Connectivity* v *Configuration View*. Po tomto kroku sú všetky potrebné premenné dostupné cez protokol OPC UA a my sme mohli prejsť na tvorbu *data binding* štruktúr. Skoro každý widget mappView vizualizácie obsahuje vlastnosť, na ktorú sa dá pripojiť na premenné programu, ktorý beží na

cieľovom zariadení. U prepínacích tlačidiel sa táto vlastnosť volá *value*. Obidvom tlačidlám sme povolili čítanie aj zápis logickej hodnoty do príslušných premenných programu.




Obr. 29: Príklad tvorby data binding štruktúr.

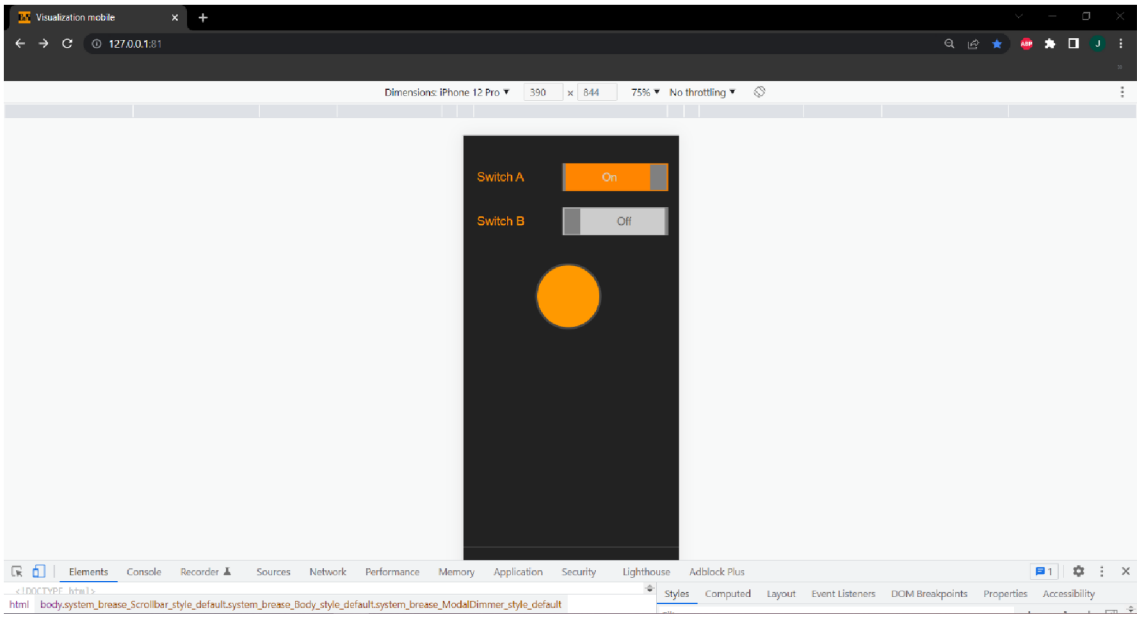
Na signalizáciu či žiarovka svieti alebo nesvieti sme použili grafický prvok elipsa, ktorej na základe hodnoty premennej získanej z programu budeme meniť farbu. Z tohto dôvodu musí byť vizualizácia schopná detegovať zmenu hodnoty premennej v programe. Presne na to slúžia udalosti. Udalosť s názvom *ValueChanged* deteguje zmenu hodnoty, konkrétneho OPC UA uzlu. Akcia, ktorá sa zavolá pri zmene hodnoty *Led* sa nazýva *SetStyle*, a podľa hodnoty premennej nastaví požadovaný grafický štýl. Ak hodnota premennej *Led* nadobudne logickú jednotku, elipse sa nastaví štýl s názvom *Orange*. V opačnom prípade bude mať elipsa štýl *default*.



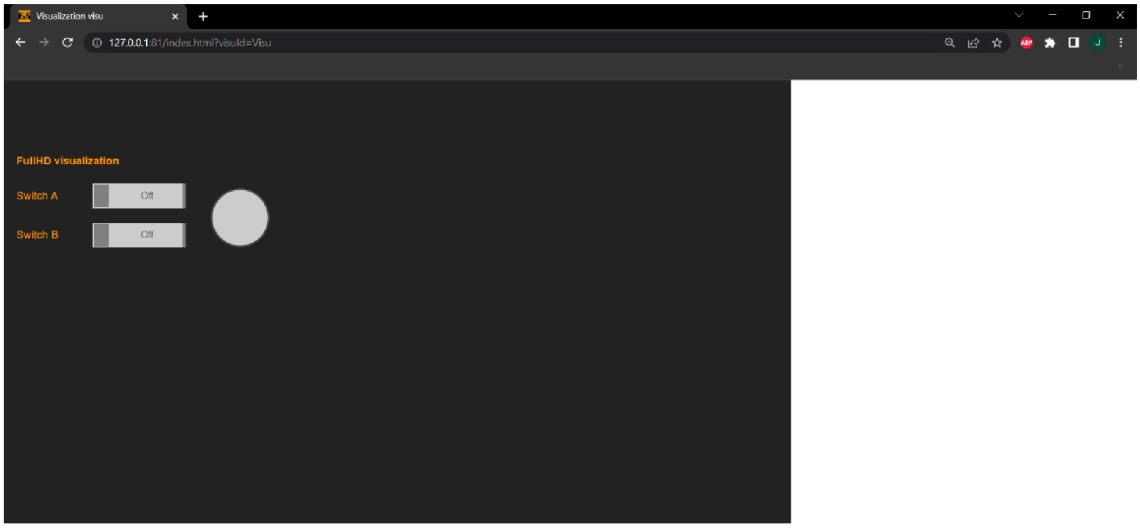

Obr. 30: Príklad tvorby udalostí.

V tomto bode je naša vizualizácia na simuláciu zapojenia schodišťových spínačov hotová a dostupná cez internetový prehliadač, kde URL adresu môžeme zadať v dvoch možných formátoch :

- *IpAdresaZariadenia:81/index.html?visuId=IdViuzalizácie*
- *IpAdresaZariadenia:81/index.html*




Obr. 31: Mobilná verzia vizualizácie.




Obr. 32: Širokouhlá verzia vizualizácie

# 6 ZÁVER

Na základe naštudovaných znalostí a odborných materiálov bola úspešne vytvorená pilotná verzia aplikácie, ktorá zjednoduší vývoj webových automatizačných vizualizácií pomocou technológie mappView vo vývojovom prostredí B&R Automation Studio.

Začiatok práce je venovaný samotnej architektúre technológie mappView od spoločnosti B&R Industrial Automation. MappView je technológia určená na tvorbu webových automatizačných vizualizácií v prostredí B&R Automation Studio. Architektúra mappView je postavená na webových technológiách, ako napríklad HTML, CSS a Javascript. MappView zároveň poskytuje automatizačným inžinierom nástroje na tvorbu efektívnych, intuitívnych a vďaka webovým technológiám ľahko prístupným HMI aplikáciám.

V praktickej časti je popísaný celkový priebeh vývoja aplikácie od návrhu architektúry aplikácie až po samotnú realizáciu. Navrhnutá architektúra aplikácie pozostáva z troch hlavných častí. Prvá a najdôležitejšia časť je konfiguračný súbor, ktorý obsahuje informácie o stránkach vizualizácie, ktoré sa majú vygenerovať. Tento konfiguračný súbor taktiež obsahuje určite nastavenia backendu našej aplikácie, ako napríklad či má aplikácia povolenie obsluhovať súborový systém vývojového prostredia B&R Automation Studio. Druhou časťou je užívateľské rozhranie na úpravu konfiguračného súboru. Poslednou časťou je backend našej aplikácie, ktorý po načítaní konfiguračného súboru spracuje obsiahnuté dáta. V praktickej časti je aj spomenutý návod ako implantovať vlastné programy alebo skripty do vývojového prostredia B&R Automation Studio. Záver praktickej časti má čitateľovi práce priblížiť a ukázať proces tvorby webových automatizačných vizualizácií pomocou technológie mappView a nami vytvorenej aplikácie.

V budúcnosti je plánované pokračovanie vývoja tohto projektu. Nasledujúci vývoj by zahŕňal prechod užívateľského grafického rozhrania do internetového prehliadača za pomocí technológií HTML, CSS a Javascript. Ďalšou možnou funkciou našej aplikácie by bolo generovanie šablónovitých stránok a ich súčastí, čo ešte viac urýchli proces tvorby automatizačných vizualizácií pomocou technológie mappView vo vývojovom prostredí B&R Automation Studio.

# ZOZNAM POUŽITEJ LITERATÚRY


[1] B&R INDUSTRIAL AUTOMATION GMBH. *B&R Automation Academy: TM611 - Working with mapp View* [online]. 2020 [cit. 2022-04-23]. Dostupné z: https://www.br-automation.com/cs/soubory-ke-stazeni/automation-academy/training-modules/visualization-and-operation/mapp-view/tm611-working-with-mapp-view/

[2] B&R INDUSTRIAL AUTOMATION. *B&R Automation Studio 4.10: Help* [software]. 2021 [cit. 2022-05-16]. Dostupné z: https://www.br-automation.com/cs/soubory-ke-stazeni/#categories=Software-1344987434933/Automation+Studio-1344987435049/Automation+Studio+4.10-1622214236094

[3] *Learn Python Programming: Third Edition*. 2021. Birmingham, UK: Packt Publishing Ltd., 2021. ISBN 978-1-80181-509-3.

[4] The Python Logo. In: *Python* [online]. 2022 [cit. 2022-05-16].

[5] *Poetry* [online]. 2022 [cit. 2022-04-23]. Dostupné z: https://python-poetry.org/docs/

[6] RICHTER, Stephan. *Lxml: XML and HTML with Python* [online]. 2022 [cit. 2022-04-23]. Dostupné z: https://lxml.de/tutorial.html

[7] PYTHON SOFTWARE FOUNDATION. *Pathlib: Object-oriented filesystem paths* [online]. 2022 [cit. 2022-04-23]. Dostupné z: https://docs.python.org/3/library/pathlib.html

[8] THE QT COMPANY LTD. *Qt for Python Documentation: Pyside6* [online]. 2022 [cit. 2022-04-23]. Dostupné z: https://doc.qt.io/qtforpython/contents.html

[9] THE QT COMPANY LTD. *Model/View Programming* [online]. 2022 [cit. 2022-05-01]. Dostupné z: https://doc.qt.io/qtforpython/overviews/model-view-programming.html

[10] FITZPATRICK, Martin. *Creating your first app with PySide6: A simple Hello World! application with Python and Qt* [online]. 2022 [cit. 2022-05-02]. Dostupné z: https://www.pythonguis.com/tutorials/pyside6-creating-your-first-window/

[11] W3SCHOOLS. *Python Collections (Arrays)* [online]. 2022 [cit. 2022-04-26]. Dostupné z: https://www.w3schools.com/python/python_lists.asp

[12] FITZPATRICK, Martin. *PySide6 Signals, Slots & Events: Triggering actions in response to user behaviors and GUI events* [online]. 2022 [cit. 2022-05-01]. Dostupné z: https://www.pythonguis.com/tutorials/pyside6-signals-slots-events/