1 TVORBA APLIKÁCIE

Pred začiatkom tvorby aplikácie bola vykonaná analýza, pomocou ktorej sme zistili, že problematikou spojenia internetu vecí s virtuálnou realitou sa už venuje viacero aplikácií, napríklad Virtualitics. Spojenie týchto dvoch svetov je však zatiaľ ojedinelé a existuje veľký priestor na rozšírenie poskytovaných služieb.

Aplikácia je vyvíjaná na základe spolupráce so spoločnosťou GRISSP, ktorá sa venuje IoT riešeniam. Cieľom GRISSP je vyvinutie komplexnej služby Flover.io, ktorá poskytuje softvérové aj hardvérové riešenia pre zber dát a ich následné spracovanie. Účel virtuálnej reality v produkte Flover.io je najmä prvým krokom k vytúženému spojeniu fyzického a virtuálneho sveta prostredníctvom modelu Digital Twin.

1.1 Špecifikácia problému

Požiadavky na aplikáciu sú špecifikované potrebou vytvorenia virtuálneho prostredia, ktoré bude znázorňovať dáta prijaté pomocou internetu vecí. Pred samotnou tvorbou aplikácie je potrebné uvedomiť si najzásadnejšie problémy, ktoré potrebujeme vyriešiť:

- potrebný hardvér pre virtuálnu realitu,
- potrebný softvér:
    - pre vývoj virtuálnej reality,
    - komponent pre znázornenie dát pomocou grafov vo virtuálnom prostredí,
    - komponent pre menu vo virtuálnom prostredí, pomocou ktorého sa vyberú dáta, ktoré chce konečný používateľ znázorniť,
    - komponent pre interaktivitu vo virtuálnom prostredí pomocou ovládačov pre VR.
- potrebné dáta, ktoré budú zobrazené vo virtuálnom prostredí.

Riešením problémov vymenovaných v bodoch vyššie na základe vykonanej dôkladnej analýzy bude:

- aplikácia navrhnutá pre zariadenia virtuálnej reality značky Oculus,
- prepojenie dát s prostredím VR (testovacie dáta poskytla spoločnosť GRISSP),
- využitie dát z aplikácie BackOffice o parkovacích kartách pre účely zobrazenia vo VR.

Pre realizáciu bude využitý nasledovný hardvér:

- okuliare pre virtuálnu realitu Oculus Quest a Oculus Rift S, ktoré budú pripojené na PC Lenovo Legion C730-19ICO 90JH.

Softvér potrebný pre vývoj aplikácie:

- Oculus slúžiaci na prepojenie virtuálnej reality a počítača,
- Unity v. 2019.4.15f1 pre vývoj virtuálneho prostredia, ktorého súčasťou je Unity Hub slúžiaci na prehľad a tvorbu nových projektov,
- Visual Studio pre tvorbu *scriptov* použitých v Unity,
- Curved UI, ktorý využijeme na tvorbu menu aplikácie,
- Chart and Graph, pomocou ktorého budú vizualizované dáta prijaté z databázy,
- BNG Framework, ktorého *scripty* využijeme na manipuláciu s menu (*event system).*

1.1.1 Aplikácia BackOffice

Aplikácia BackOffice slúži ako administračné rozhranie, pomocou ktorého sú spracovávané všetky informácie súvisiace s tvorbou a žiadosťami o parkovacie karty. Používateľ *BackOffice* je osoba poverená mestom, jeho časťou, či majiteľom určitého priestoru, ktorý je určený na prenájom parkovacích miest.

Hlavnými úlohami používateľa aplikácie *BackOffice* sú:

- spracovávanie žiadostí o parkovacie karty,
- manipulácia s parkoviskami – vytváranie, editácia a odstraňovanie parkovísk
- manipulácie s parkovacími kartami (vytváranie, editácia a odstraňovanie):
    - typ karty,
    - tarify karty,
    - podmienky pre vydanie karty,
    - ceny karty,
    - priradenie karty k typu osoby (fyzická, právnická, podnikateľ).

Ďalšie možnosti používateľa v aplikácii sú:

- sledovanie reportov parkovísk, platieb, počtu kariet,
- sledovanie priestupkov na parkoviskách.

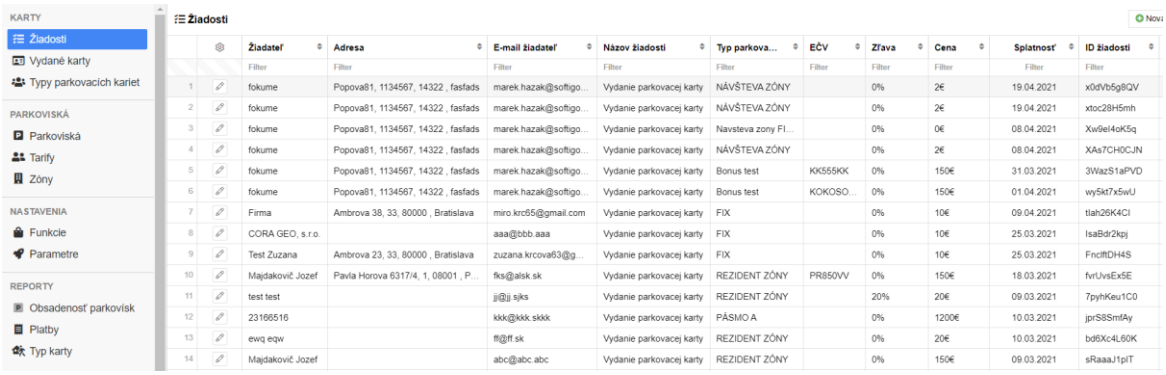
1.1.1.1 Žiadosti o parkovaciu kartu

Používateľ aplikácie má možnosť vytvoriť novú žiadosť o parkovaciu kartu alebo spracovať už existujúcu žiadosť. Všetky žiadosti sa v aplikácii nachádzajú v sekcii „Žiadosti“.

Spôsoby vytvorenia žiadosti o parkovaciu kartu:

- online, prostredníctvom webovej aplikácie pre vytvorenie žiadosti,
- osobne, na miestach na to určených, kde poverená osoba zaeviduje pomocou aplikácie *BackOffice* žiadosť do systému.

Po vyplnení formuláru s osobnými dátami žiadateľa a údajmi o parkovacej karte sa vytvorí žiadosť, ktorú používateľ *BackOffice* vidí v sekcii „Žiadosti“ a ďalej posudzuje možnosť vydania parkovacej karty pre danú osobu podľa kritérií, ktoré prináležia zvolenej karte. Ak tieto kritéria žiadateľ splní, bude mu vygenerovaný variabilný symbol pre platbu. To neplatí v prípade, ak je cena za parkovaciu kartu nulová. Po zaplatení parkovacej karty bude parkovacia karta platná od dátumu, ktorý si žiadateľ zvolil. Používateľ *BackOffice* ďalej eviduje parkovaciu kartu v sekcii „Vydané karty“.



Obrázok 1: Zoznam žiadostí v aplikácii BackOffice

1.1.1.2 Parkovací systém

Po vydaní parkovacej karty bude používateľovi zaslaný e-mail s prístupovými údajmi, pomocou ktorých mu bude umožnený prístup do webovej a mobilnej aplikácie pre parkovací systém. Používateľ sa môže do parkovacieho systému zaregistrovať aj bez žiadosti o parkovaciu kartu.

Používateľ parkovacieho systému dokáže:

- meniť svoje osobné údaje – údaje budú automaticky zosynchronizované s údajmi v *BackOffice*. V prípade vytvárania novej parkovacej karty budú tieto dáta automaticky vopred vyplnené vo formulári pre žiadosť o parkovaciu kartu,

- vytvárať, editovať a odoberať svoje vozidlo (prípadne dočasné vozidlo),
- zobraziť prehľad:
    - platných parkovacích kariet,
    - žiadostí o parkovaciu kartu,
    - histórie parkovacích lístkov,
    - nastávajúcich parkovacích lístkov,
- zakúpiť si parkovací lístok,
- nabiť kredit pre platbu parkovacích lístkov.

1.2 Návrh aplikácie

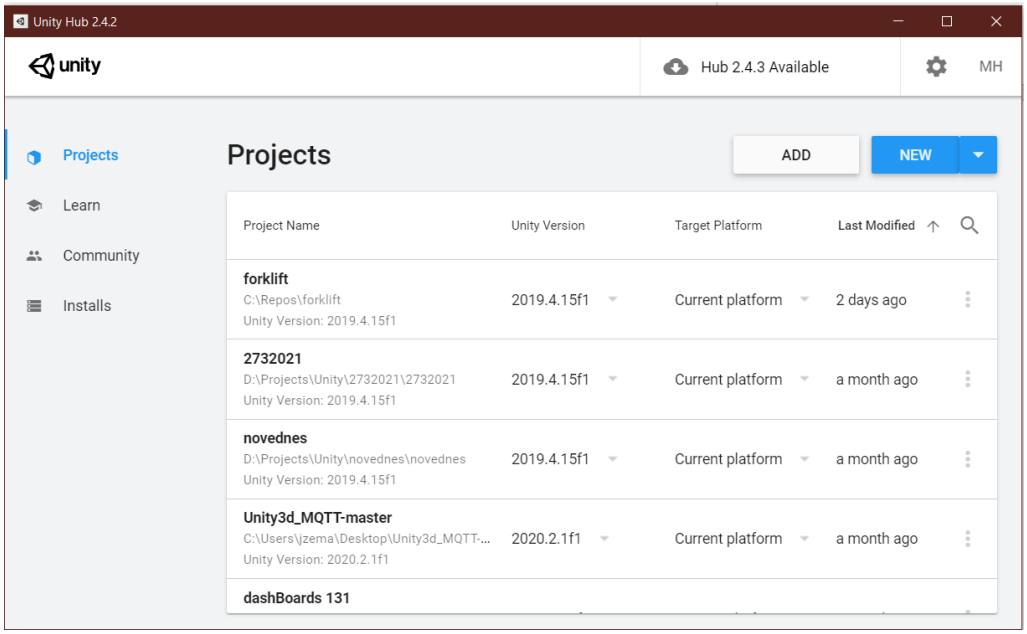
V tejto časti začína vývoj aplikácie. Na základe dát prijatých z aplikácie BackOffice chceme, aby sa detaily a štatistiky zobrazovali rozdelené do obrazoviek:

- menu s výberom mesta, kde si používateľ môže vybrať mesto, podľa ktorého budú selektované dáta v ďalších obrazovkách,
- menu mesta, ktoré zobrazí:
    - 5 najnovších žiadostí,
    - Položky menu:
        - „Parkovacie karty“,
        - „Prehľad výšky platieb“,
        - „Využitie parkovísk“,
        - „Mesačné tržby“.

Každá z položiek menu mesta bude interaktívna a po potvrdení stlačením tlačidla na ovládači VR sa zobrazí štatistika alebo detail pre zvolenú položku. Používateľ sa bude môcť z každého menu vrátiť pomocou kliku na ikonu na to určenú. Dáta do aplikácie budú prenesené vďaka využitiu REST API HTTP protokolu.

1.3 Postup pri tvorbe aplikácie

Po úspešnej inštalácii softvérov zadefinovaných v sekcii Špecifikácia problému, spustíme aplikáciu *Unity Hub*, ktorá používateľovi umožňuje vytvoriť nový projekt alebo listovať v zozname existujúcich projektov. Aplikácia taktiež umožňuje stiahnuť a nainštalovať viacero verzií Unity. Používateľ si pri vytváraní nového projektu vyberie verziu, v ktorej chce vyvíjať softvér.



Obrázok 2: Unity Hub

Po kliknutí na tlačidlo *New* sa zobrazí formulár pre vytvorenie nového projektu. Po voľbe názvu projektu, cieľového umiestnenia, 3D prostredia, sa otvorí prostredie Unity.

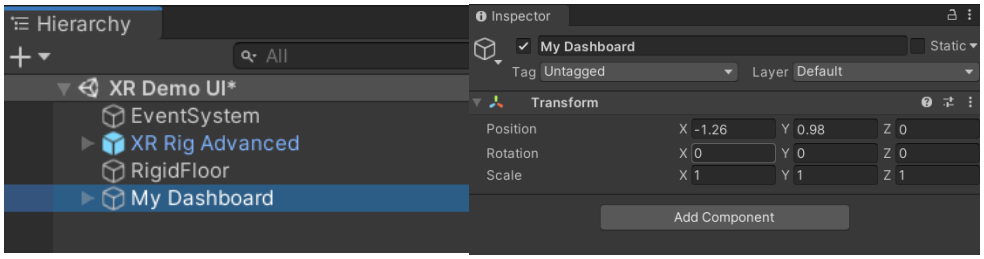
1.3.1 Používateľské rozhranie Unity

Pre dobrú orientáciu v projekte je potrebné zoznámiť sa s používateľským rozhraním Unity. Používateľské rozhranie alebo *layout* je možné zmeniť podľa potreby vývojára. Pre tvorbu záverečnej práce budeme používať používateľské rozhranie s nasledovným rozložením:

- Vrchná časť menu slúžiacu na:
  - vytváranie objektov, komponentov a *scriptov* (časť menu),
  - pohyb vo virtuálnom prostredí scény,
  - pridávanie *assetov*,
  - konfiguráciu Unity,
  - spúšťanie a vypínanie aplikácie.

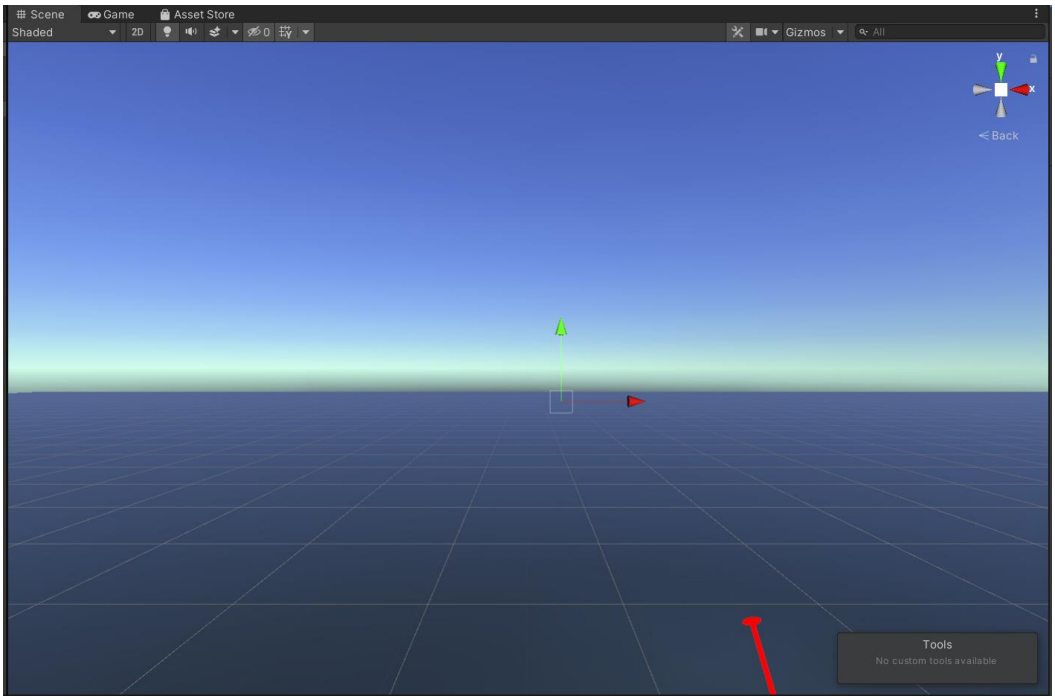
Obrázok 3: Vrchná časť používateľského rozhrania v Unity

- ľavá časť používateľského rozhrania Unity s názvom *Hierarchy* slúži na zobrazenie všetkých objektov nachádzajúcich sa vo virtuálnom prostredí,
- pravá časť používateľského rozhrania Unity s názvom *Inspector* slúži na zobrazenie všetkých komponentov patriacich pod objekt, ktorý je zvolený v časti *Hierarchy*,



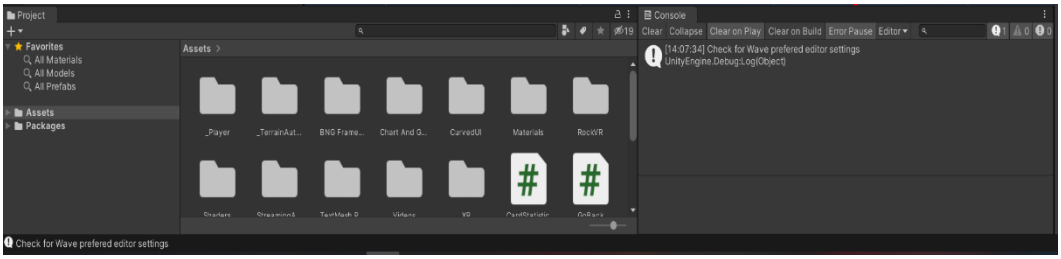
Obrázok 4 Hierarchia projektu a *Inspector*

- hlavná časť, nachádzajúca sa medzi panelmi *Inspector* a *Hierarchy* zobrazuje model virtuálneho prostredia alebo v prípade spustenej aplikácie používateľovu aktivitu vo virtuálnom prostredí. Pomocou hlavnej časti je tiež možné využívať Asset Store,



Obrázok 5 Default scéna v Unity

- spodná časť, ktorá je rozdelená na dve časti:
    - súbory, ktoré sú importované, vytvárané a využívané v projekte ako napríklad *assety*, *scripty* alebo komponenty,
    - konzola, ktorá pri vývoji upozorní na rôzne chyby, prípadne môže byť využívaná na *debugovanie*.

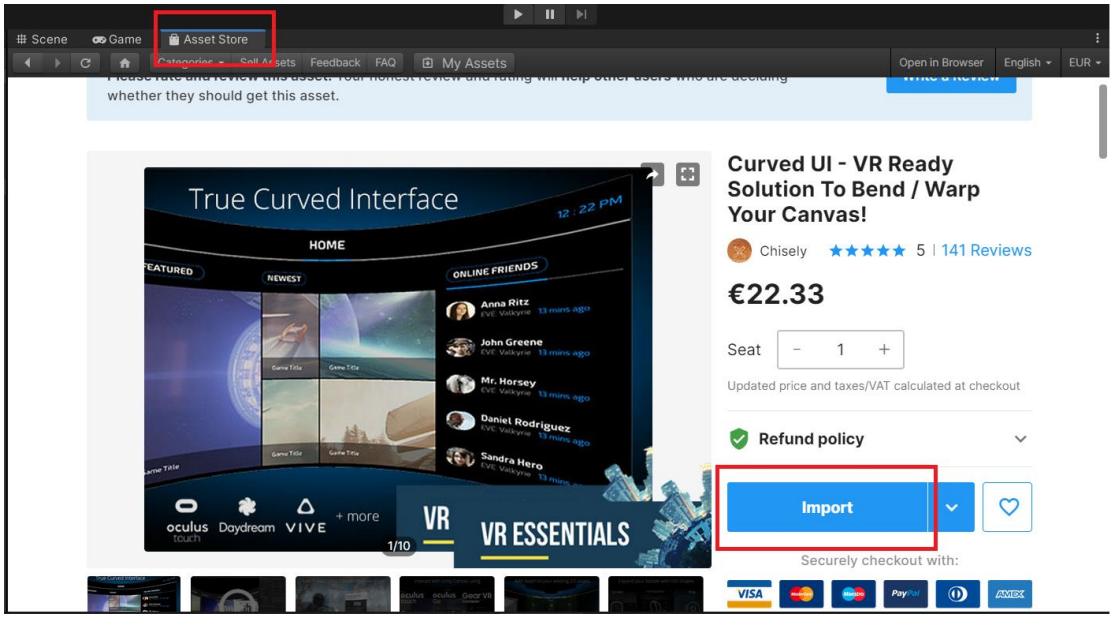


Obrázok 6 Súborová hierarchia projektu a konzola v Unity

1.3.2 Implementácia *assetov* z *Unity Asset Store*

Unity Asset Store je internetový obchod, vďaka ktorému je možné importovať *assety* do projektu. Predtým než budeme schopní importovať *assety* do projektu, je potrebné vytvoriť si účet v Unity a prihlásiť sa. *Assety* je možné kupovať, sťahovať a importovať priamo vo vývojovom prostredí Unity. *Asset store* sa zobrazí po kliknutí na vrchnú časť menu, na položku *Window* → *Asset Store*. Následne je potrebné zadať do vyhľadávania *asset*, ktorý chceme importovať do projektu.

Assety, ktoré sme zadefinovali v časti Špecifikácia problému importujeme do projektu stlačením tlačidla „Import“. V Unity sa následne zobrazí kompletný zoznam *scriptov*, modelov, demo verzií, či ďalšieho obsahu, ktorý sa nachádza v *assete*. Používateľ má možnosť vybrať si, čo chce skutočne importovať alebo naopak, čomu sa chce vyhnúť. Pre vývoj tejto aplikácie importujeme kompletne celý obsah *assetov*.

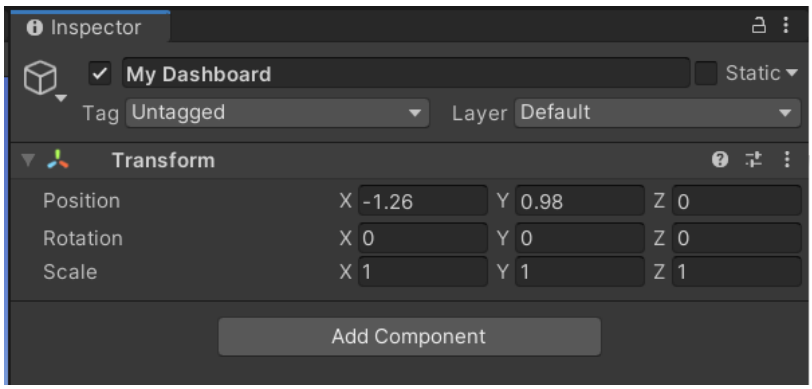


Obrázok 7 Import assetu do Unity

1.3.3 Vytváranie nového objektu

Vytváranie nových objektov je možné vo vrchnej časti používateľského rozhrania *GameObject → Create Empty*. V hierarchii projektu využívame vzťahy stromovej štruktúry. Vzťah medzi hlavným objektom a objektom nachádzajúcim sa o úroveň nižšie nazývame *parent* a *child.* Vlastnosti objektu a jeho komponenty upravujeme v časti *Inspector*. Objekt má automaticky po vytvorení priradený komponent *Transform*, pomocou ktorého dokážeme meniť:

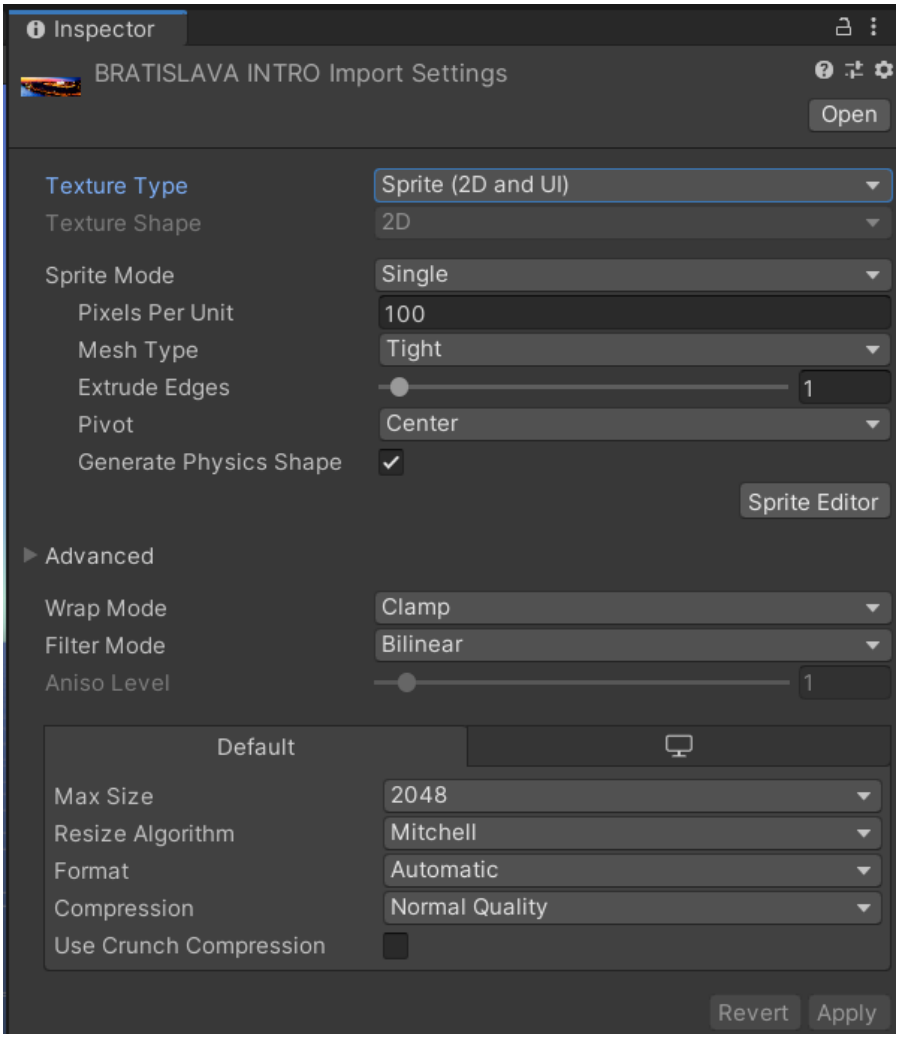
- pozíciu – *Position* šírky (x), výšky (y) a hĺbky (z),
- rotáciu – Rotation x, y, z,
- mierku – *Scale* x, y, z.



Obrázok 8 Komponent Transform v Unity

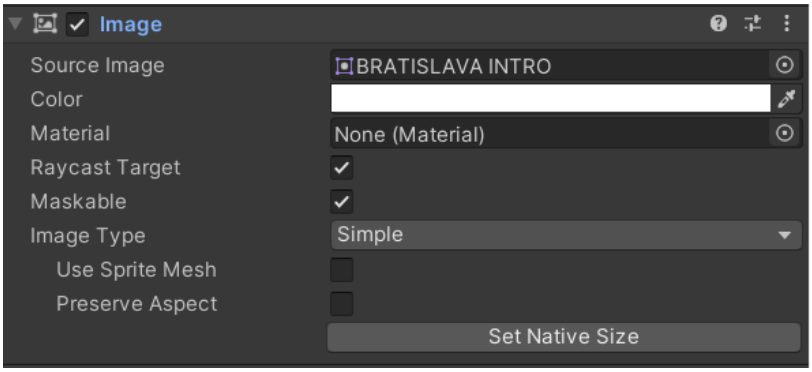
1.3.4 Implementácia obrázkov

Pre úspešnú implementáciu obrázku do projektu vytvoríme nový objekt *canvas*, do ktorého pridáme komponent *Image*. Obrázok, ktorý chceme zobraziť v aplikácii je potrebné importovať priamo do projektu a zmeniť mu textúru na *Sprite (2D and UI).* Zmenu textúry vykonáme v sekcii *Inspector* a potvrdíme pomocou tlačidla *Apply.*



Obrázok 9 Úprava textúry obrázku v Unity

Po zmene textúry spôsobom *drag and drop* presunieme obrázok do vytvoreného komponentu *Image* a obrázok sa zobrazí ako pozadie *canvasu.*



Obrázok 10 Pridávanie obrázku do komponentu Image

1.3.5 Tvorba scriptov

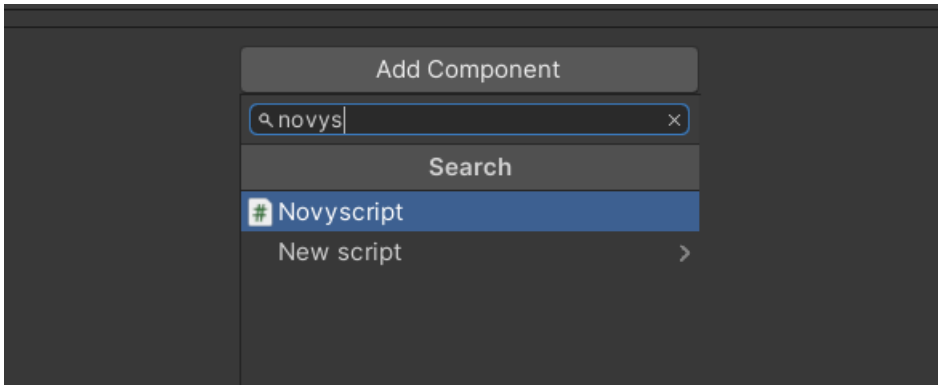
Scripty vytvárame v hornej hlavnej lište menu *Assets* → *Create* → *C# script.* Všetky *scripty* v našom projekte nájdeme v hlavnom priečinku *Assets.* Keď je *script* vytvorený, môžeme ho otvoriť dvojklikom, čo nás automaticky presmeruje do vývojového prostredia *Visual Studio.* Unity pri vytváraní nového scriptu automaticky vygeneruje verejnú triedu s názvom *scriptu,* ktorá obsahuje funkcie:

- *Start*, ktorá sa vykoná 1x, vždy pri zavolaní daného *scriptu,*
- *Update,* ktorá sa vykonáva pravidelne každé 0,02 sekundy.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script | 0 references
public class novyscript : MonoBehaviour
{
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }

    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {

    }
}
```

Ukážka kódu 1 Vytvorený script

Script je po vytvorení a prípadných úpravách dostupný medzi komponentami a pridávať ho môžeme jednoducho v rámci projektu pod ktorýkoľvek objekt v paneli *Inspector* kliknutím na tlačidlo *Add Component*.



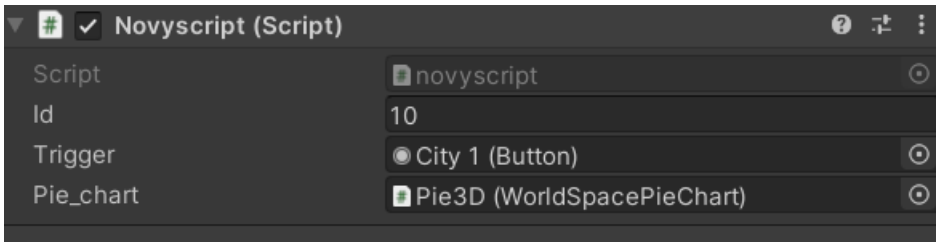
Obrázok 11 Pridávanie nového komponentu do objektu

Veľkou výhodou pri tvorbe *scriptov* v Unity je možnosť využitia *public* premenných. Pri vytváraní nových premenných prvý parameter označuje dostupnosť premennej, v tomto prípade *public*, typ a názov premennej. Po zadefinovaní nového typu premennej bude potrebné implementovať jej knižnicu v hlavičke *scriptu* pomocou príkazu *using*, za ktorým nasleduje jej názov.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using ChartAndGraph;

Unity Script | 0 references
public class novyscript : MonoBehaviour
{
    public int id;
    public Button trigger;
    public PieChart pie_chart;
```

Ukážka kódu 2 Definícia globálnych premenných v scripte

Používateľ si tak vie jednoducho cez panel *Inspector* nastaviť ID alebo objekt, na ktorom sa vykoná požadovaná akcia. Túto funkcionalitu budeme využívať pri zmene obrazoviek, zobrazovaní dát pomocou grafov a načítavaní dát z databázy.



Obrázok 12 Pridaný script ako komponent objektu v *Inspectore* Unity

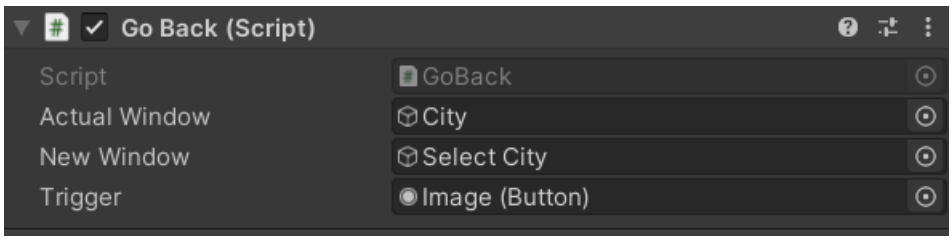
1.3.6 Script na zmenu viditeľnosti objektov

Viditeľnosť objektov v Unity nastavujeme pomocou funkcie *SetActive*, ktorá očakáva parameter typu *boolean*. Ak je hodnota *true*, objekt sa stáva viditeľným.

Vytvoríme nový *script* s názvom *Go Back* a pridáme ho do objektu, ktorého viditeľnosť potrebujeme zmeniť. Referenciu k tomuto objektu, získame zadefinovaním *public* premennej typu *GameObject*. Keďže cieľom je zmeniť viditeľnosť dvom objektom, zadefinujeme si dve tieto premenné. Spustenie celého procesu zmeny viditeľnosti zabezpečí premenná *Trigger* odkazujúca na komponent *Button*. Na *Trigger* nastavíme *EventListener onClick*, ktorý čaká na stlačenie *Buttonu*, po ktorom sa vykoná funkcia.

```
public class GoBack : MonoBehaviour
{
    public GameObject ActualWindow, NewWindow;
    public Button Trigger;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        Trigger.onClick.AddListener(SwitchWindows);
    }

    1 reference
    public void SwitchWindows()
    {
        ActualWindow.SetActive(false);
        NewWindow.SetActive(true);
    }
}
```

Ukážka kódu 3 *Script* pre zmenu viditeľnosti objektu



Obrázok 13 *Script* Go Back v *Inspectore*

1.3.7 Tvorba HTTP *requestu*

Pre získanie dát z databázy využijeme HTTP *request* volaním REST API typu POST. Tento typ *requestu* sa využíva v situáciách, keď do databázy vkladáme nové dáta,

aktualizujeme ich alebo z nej potrebujeme získať dáta na základe poslaného identifikátora – vrátia sa formou *response*. Pre prácu s HTTP protokolom budeme využívať jeho knižnicu a taktiež knižnicu pre prácu s typom JSON. Predtým, než zavoláme *request* je potrebné zadefinovať objekt modelu, ktorý zašleme a taktiež ktorý chceme prijať. Na to slúži *DataContract,* v ktorom je potrebné zadefinovať všetky premenné patriace do objektu.

```
[DataContract]
5 references
public partial class TblRequest
{
    [DataMember(Name = "pk_request_id")]
    0 references
    public int PkRequestId { get; set; }

    [DataMember(Name = "fk_car_id")]
    0 references
    public int FkCarId { get; set; }

    [DataMember(Name = "carlicense")]
    1 reference
    public string CarLicense { get; set; }
```

Ukážka kódu 4 Časť definície premenných, ktoré budú prijaté v *response* z HTTP *Requestu*

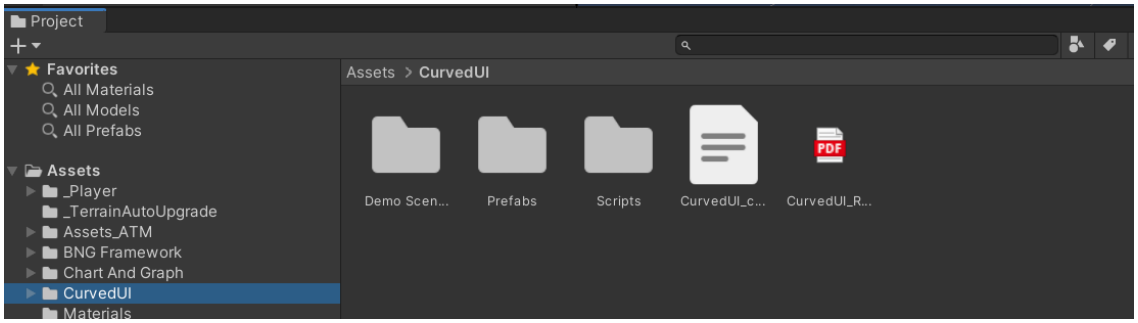
Z bezpečnostných dôvodov je infraštruktúra navrhnutá tak, aby nebol *backend* napájajúci sa na databázu a náš projekt na jednom mieste. Preto nebudeme volať priamo databázu, ale *backend,* na ktorý sa napojíme pomocou URL adresy a po úspešnom volaní nám vráti požadované dáta. Z dôvodu ochrany pred neželanými útokmi na databázu sa využíva autorizácia typu *Bearer*. Po zavolaní *requestu* podľa Ukážka kódu 5 HTTP *request* dostaneme z databázy požadované dáta, ktoré transformujeme do typu JSON.

```
HttpClient client = new HttpClient();
client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", Bearer);

CityId cityId = new CityId();
cityId.FkUserIdCity = cityid;
selectedID = cityid;

string json = JsonConvert.SerializeObject(cityId);
StringContent data = new StringContent(json, Encoding.UTF8, "application/json");
HttpResponseMessage response = await client.PostAsync(url, data);
string result = response.Content.ReadAsStringAsync().Result;
res = JsonConvert.DeserializeObject<List<TblRequest>>(result);
```

Ukážka kódu 5 HTTP *request*

1.3.8 Implementácia menu

Pre vytvorenie menu je použitý *asset Curved UI,* ktorý si importujeme a zobrazí sa v zozname *assetov* projektu.



Obrázok 14 Zobrazenie *assetu* v hierarchii projektu

Importovaný *asset* obsahuje demo scény, ktoré však nie je možné importovať ako celok, pretože sú vo formáte .unity. Riešením tohto problému je otvorenie demo scény s názvom *06 VRMenuCustomRay,* ktorá je súčasťou *assetu Curved UI*. Z tejto demo scény potrebujeme exportovať objekt s názvom *canvas*, ktorý obsahuje základné prvky menu. Každé menu sa štandardne skladá z dvoch hlavných objektov *Panel* a *Top.* V časti Top sa nachádza titul zarovnaný na stred, v ľavej časti *button* pre návrat späť a v pravej časti aktuálny čas. Na vertikálne usporiadanie objektov sa využíva komponent *Vertical Layout Group* a na horizontálne *Horizontal Layout Group*. *Canvas* vložíme ako *child* objekt do hlavného objektu *My Dashboard*.

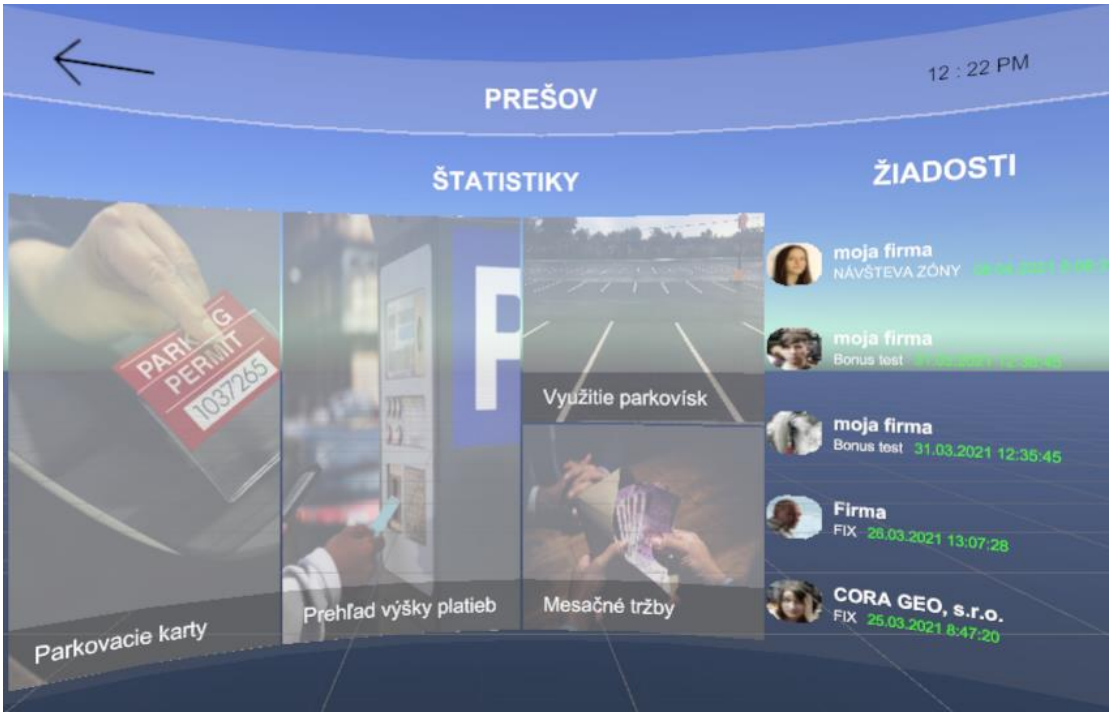
1.3.9 Menu Mesta

Podľa postupu v kapitole 1.3.8 implementujeme *canvas* do projektu a premenujeme ho na „*City*“. V objekte *City → Panel → Columns* pridáme komponent *Vertical Layout Group* a vytvoríme štyri *child* objekty:

- *Parking Cards* pre štatistiku predajnosti parkovacích kariet,
- *Pay Stats* pre štatistiku výšky platieb za parkovací lístok,
- *Stats Container*, pre štatistiku využitia parkovísk a štatistiku mesačných tržieb,
- *Request Panel,* ktorý zobrazí 5 posledných žiadostí o parkovaciu kartu.

V objektoch, ktoré využijeme na zobrazenie štatistiky pridáme komponenty *Image* a *Text.* Do *Image* vložíme pozadie a do *Textu* popis objektu. Všetky objekty budú obsahovať *scripty* pre vizualizáciu dát, ktoré získame z databázy.

6 *child* objektov vytvoríme v *Request Paneli*. Do prvého objektu pridáme komponent *Text* a hodnotu nastavíme na „ŽIADOSTI“. Do ďalších objektov pridáme komponenty *Text* a *Image*, pomocou ktorých neskôr zobrazíme 5 žiadostí.



Obrázok 15 Menu mesta

1.3.10 Menu Zvoľte mesto

Podľa postupu v kapitole 1.3.8 implementujeme *canvas* do projektu a premenujeme ho na „*Select City“*. V objekte *Select City → Panel → Big Panel* vytvoríme dva nové objekty, ktoré nazveme „*City 1“* a „*City 2“* a do oboch pridáme komponent *Image*, do ktorého vložíme pozadie. Do objektu *Big Panel* pridáme komponent *Vertical Layout Group*.

Obrázok 16 Menu zvoľte mesto

Pre načítanie dát po zvolení mesta vytvoríme *script* s názvom „*Select City"* a pridáme ho do objektov *City 1* a *City 2*. Na začiatku *scriptu* si zadefinujeme premenné, ktoré budeme využívať ako referencie pre objekty.

```
Unity Script | 21 references
public class SelectCity : MonoBehaviour
{
    public GameObject SelectCityWindow, CityWindow;
    public Button Trigger;
    public int CityID;
    public static List<TblRequest> res;
    private Text ordererName, ordererCardTypeName, orderDate;
    public static DateTime ActualTime;
    public static int selectedID;
```

Ukážka kódu 6 Premenné v *scripte* pre voľbu mesta

- *GameObject SelectCityWindow* a *CityWindow* tvoria referenciu na objekty menu,
- premenná typu *Button* tvorí referenciu na komponent objektu, pomocou ktorej sa vykoná *script*,
- premenná *CityID* je celočíselného typu. Mesto Prešov má ID 3, Bratislava 12. Identifikátory sú určené podľa primárneho kľúča v databáze,
- *List<TblRequest>* označuje pole objektov, do ktorého vložíme dáta prijaté z databázy,
- premenné *ordererName, ordererCardTypeName, orderDate* tvoria referenciu na komponent *Text*,

- *DateTime ActualTime* je časová premenná, do ktorej vložíme aktuálny čas,
- *selectedID* využijeme v ďalších obrazovkách pre informáciu o zvolenom ID mesta.

Pomocou HTTP *requestu* získame dáta z databázy, ktoré vložíme do menu mesta.

1.3.11 Vizualizácia dát

Dáta budeme vizualizovať pomocou *assetu Chart and Graph*, ktorý si implementujeme do projektu a exportujeme z neho dva druhy grafov *Pie chart* a *Line chart*. Podľa návrhu aplikácie si vytvoríme nový objekt pre každý typ štatistiky. Dáta do grafov budeme získavať z databázy vďaka volaniu REST API s využitím HTTP protokolu. Prijaté dáta budeme triediť pomocou knižnice *Linq,* ktorú využíva C#.

Nastavovať hodnoty v stĺpcoch je možné pomocou funkcie *SetValue*, ktorá očakáva ako parametre názov kategórie, názov skupiny a hodnotu. V prípade *Pie chart* sú parametre názov kategórie a hodnota. Pre nastavovanie názvu skupiny, minimálnu a maximálnu hodnotu v grafe, materiály v grafe využijeme referenciu *Datasource*.

V ukážke kódu 7 je príklad triedenia dát, ktoré sme získali pomocou REST API a uložili do premennej counts. Premenná *pieChart* je referenciou pre objekt grafu v Unity. Funkcia *Where* slúži na nájdenie hodnoty v premennej *counts* (pole). Do parametru funkcie vložíme hodnotu, ktorú má *Where* nájsť. Cieľom však nie je nájsť jednu hodnotu, ale celkový počet hodnôt v zadanom rozpätí, na čo využijeme funkciu *Count.*

```
counts = counts.OrderByDescending(c => c.Count).ToList();
pieChart.DataSource.SetValue("Category 1", counts.Where(x => x.Price > 1 && x.Price < 2).Count());
pieChart.DataSource.SetValue("Category 2", counts.Where(x => x.Price > 2 && x.Price < 5).Count());
pieChart.DataSource.SetValue("Category 3", counts.Where(x => x.Price > 5 && x.Price < 10).Count());
pieChart.DataSource.SetValue("Category 4", counts.Where(x => x.Price > 10).Count()); // 4
pieChart.DataSource.SetValue("Category 5", counts.Where(x => x.Price < 1).Count()); // 8
```

Ukážka kódu 7 Nastavovanie hodnoty v grafe *PieChart*

Do objektu, v ktorom sa nachádza graf vytvoríme *canvas* obsahujúci komponent *Button* pre návrat do menu mesta a komponent *Text*, ktorý využijeme ako nadpis. Legendu ku grafu vytvoríme použitím objektu menu, ktorý nazveme *Graph Detail*. Pre zobrazenie farby použitej v grafe využijeme komponent *Image* a pre popis hodnoty komponent *Text*.

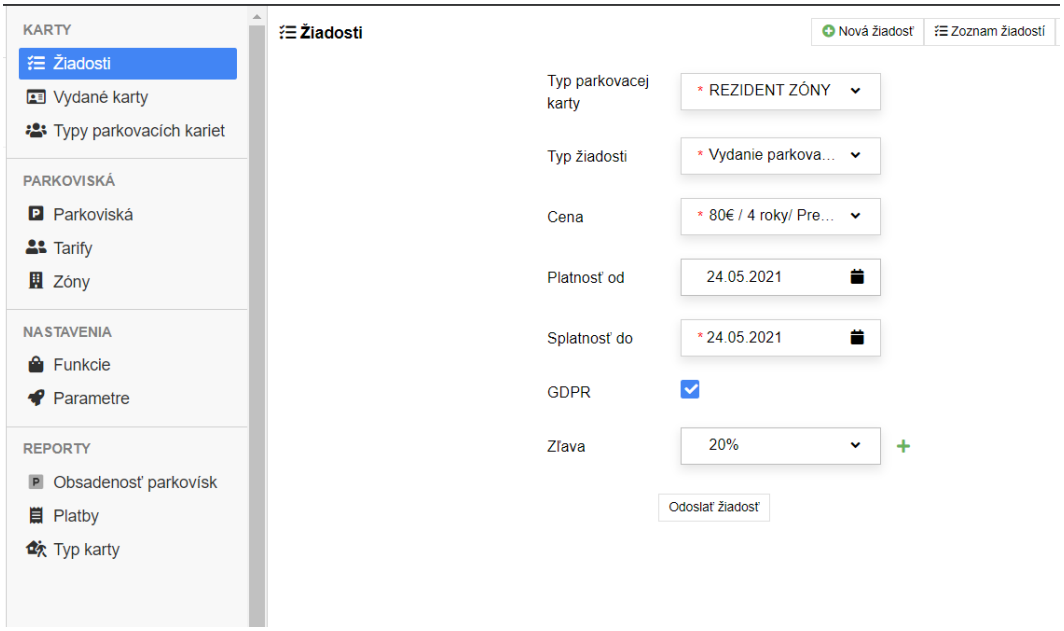
Po vykonaní všetkých krokov dostaneme výsledok vo forme koláčového grafu, zobrazujúceho štatistiku cien parkovacích lístkov.

Obrázok 17 Vizualizácia štatistiky cien parkovacích lístkov pomocou *Pie Chart*

Pre vytvorenie ďalších grafov postup opakovane aplikujeme.

1.4 Testovanie aplikácie

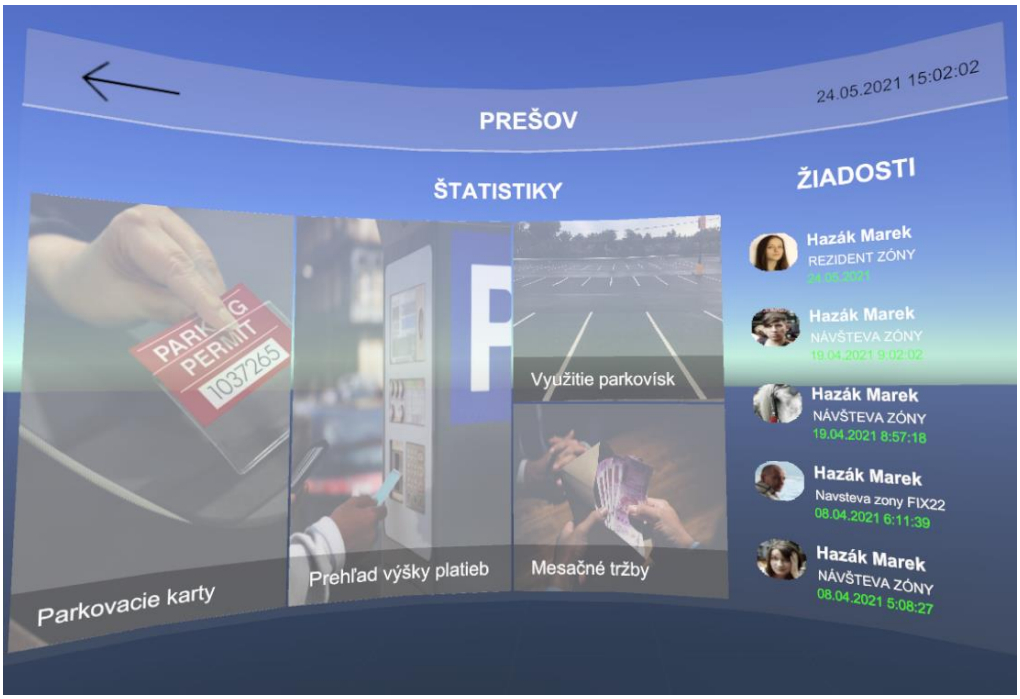
Najzásadnejšie pri testovaní aplikácie bolo overenie jej funkčnosti v praxi. Cieľom tohto testu bolo primárne overiť prenos novo-vytvorenej žiadosti z aplikácie BackOffice pre mesto Prešov (Obrázok 18) do virtuálneho prostredia.



Obrázok 18 Tvorba žiadosti o parkovaciu kartu v BackOffice

Po úspešnom vytvorení žiadosti v aplikácii BackOffice sme spustili naše virtuálne prostredie a v úvodnom menu zvolili mesto Prešov. Po zvolení mesta sa nám zobrazí ponuka

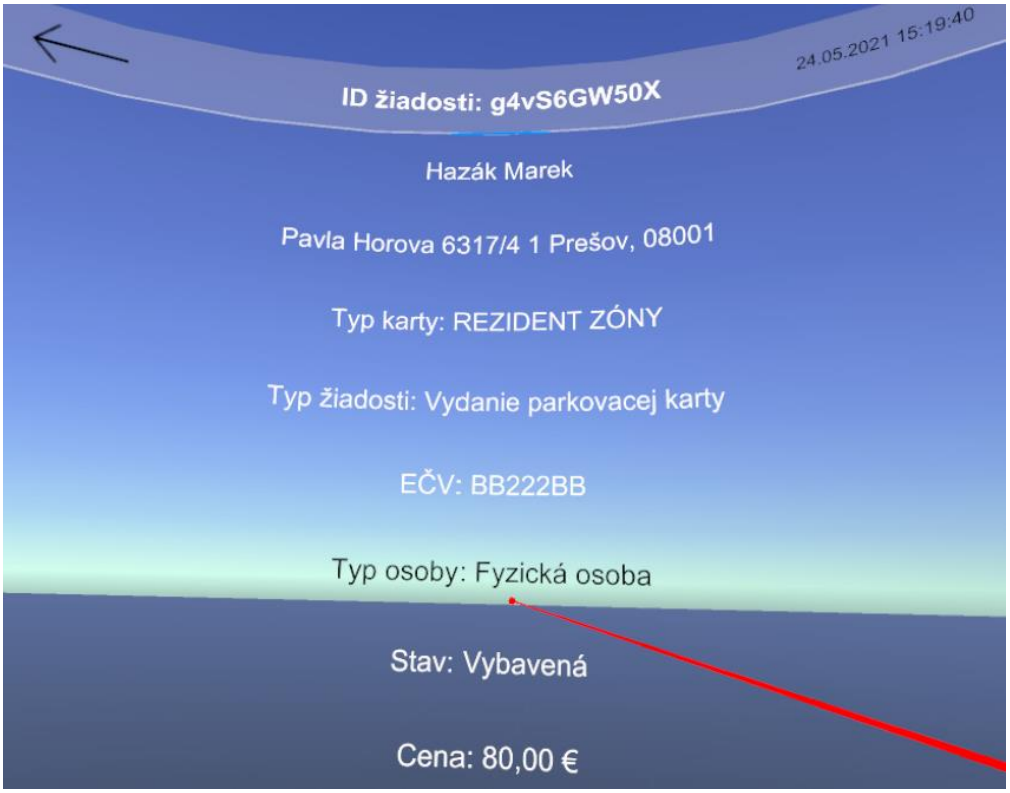
položiek menu pre toto mesto, medzi ktorými sa nachádza aj zoznam piatich posledných žiadostí o parkovaciu (Obrázok 19 Testovanie zobrazenia žiadosti v aplikácii).



Obrázok 19 Testovanie zobrazenia žiadosti v aplikácii

Pre skontrolovanie vstupných dát, ktoré sme zadali do žiadosti sme označili najnovšiu žiadosť a ovládačom VR potvrdili náš výber. Po tejto akcii sa nám zobrazí detail zvolenej žiadosti. Podľa ilustrácie (Obrázok 20 Detail žiadosti vo virtuálnom prostredí) môžeme vidieť, že dáta, ktoré sme zadávali do aplikácie BackOffice (Obrázok 18 Tvorba žiadosti o parkovaciu kartu v BackOffice) sú zhodné s tými, ktoré sa nám zobrazili vo virtuálnom prostredí. Tento test potvrdzuje, že skutočne dostávame do našej aplikácie dáta prostredníctvom internetu vecí v reálnom čase. Dokázali sme teda spojiť odvetvie internetu vecí s virtuálnou realitou.

Ako ďalší sme vykonali test používateľského rozhrania. Cieľom bolo otestovať prehľadnosť a jednoduchosť aplikácie a odhaliť jej prípadné chyby, či nežiadúce účinky.



Obrázok 20 Detail žiadosti vo virtuálnom prostredí

Aplikáciu sme dali otestovať viacerým používateľom. V prvom kole testovania na vzorke troch ľudí bol odhalený problém s rozložením menu a grafov. Pôvodný návrh vytvorenia štatistík, ktoré boli rozložené po celom 360° zornom poli virtuálneho prostredia vyžadoval príliš mnoho zmien polohy a stal sa pre používateľov nepraktickým a ťažkým pre zorientovanie. Tento problém bol vyriešený umiestnením všetkých objektov do jedného zorného poľa používateľa s funkciou na zmenu viditeľnosti objektu, čím sa aplikácia zbavila aj prípadných nežiaducich účinkov z *motion sickness*.

Používatelia v druhom kole testovania prejavili nadšenie z prehľadnosti a jednoduchosti aplikácie.