Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of computer science

A Tool for Public Space Use Modeling

Hugo Mrázek

Supervisor: Ing. Martin Ledvinka, Ph.D.
Field of study: Software Engineering and Technology
May 2023

Acknowledgements

Thanks to Ing. Marting Ledvinka, Ph.D. for supervising this work and having endless patience while dockerizing and deploying the app. Thanks to my lovely girlfriend Karolína for being such a strong support. Last but not least thanks to the testers that tested my application.

Declaration

I hereby declare that I have completed this thesis independently and that I have mentioned all the used information sources in accordance with the Guideline for compliance with ethical principles in the course of writing final theses.

In Prague, 23. May 2023

Abstract

This bachelor thesis explores motivations, imaging methods, and existing software solutions for gathering feedback during public space design. Leveraging web technologies, a major part of the thesis is to develop an application that enables seamless communication between designers, stakeholders, and the public. It incorporates interactive features such as visual simulations, and interactive maps, empowering users to visualize and provide input on proposed designs. The user-centric approach encourages collaboration, supporting multiple languages, and accessibility. Provided feedback can be analyzed by professionals, providing valuable insights for decision-making and design improvements.

Keywords: public space, design of public space, feedback gathering, web application

Supervisor: Ing. Martin Ledvinka, Ph.D.
Místnost E-305,
Resslova 9,
Praha 2

Abstrakt

Tato bakalářská práce zkoumá motivace, metody zobrazování a existující softwarová řešení pro sběr zpětné vazby při návrhu veřejného prostoru. Využívajíc webové technologie se hlavní část práce zaměřuje na vývoj aplikace umožňující plynulou komunikaci mezi designéry, stakeholdery a veřejností. Aplikace zahrnuje interaktivní prvky, jako jsou vizuální simulace a interaktivní mapy, které umožňují uživatelům vizualizovat a poskytovat zpětnou vazbu k navrženým designům. Design zaměřený na uživatele podporuje spolupráci a podporuje vícejazyčnost a dostupnost. Poskytnutá zpětná vazba může být analyzována odborníky a poskytovat cenné poznatky pro rozhodování a zlepšování návrhů.

Klíčová slova: veřejný prostor, návrh veřejného prostoru, sběr zpětné vazby, webová aplikace

Překlad názvu: Nástroj pro modelování využití veřejného prostoru

Contents

Figures

Tables

Chapter 1

Introduction

Every day we pass through numerous public spaces - from parks and plazas to streets or sidewalks. When designing such public spaces, it is necessary to gather feedback from users - the general public. Obtaining feedback is not yet common as it should be - there is no tool specifically designed for such a task. Because there is no tool solely for gathering feedback, it is challenging to do it in an effective and quantifiable manner. Researchers at CTU Faculty of Architecture noticed this.

Ing. arch. Jana Zdráhalová, Ph.D. defines multiple challenges in a white paper[1]. First - there is a difference between the language used by professionals and users. Second - a difference between the intended and actual use of public space can happen. These challenges are further discussed in chapter 1.1. Such roadblocks could be tackled with a suitable application. The creation of such an application is a subject of this work.

The thesis is structured as follows: Chapter 2 provides an analysis of existing solutions with similar goals and functions and an analysis of file types in which geospatial data may be represented. It also discusses imaging methods that might be used. Chapter 3 describes the design and implementation process of the app. Chapter 4 is about user testing of the app.

[1]The whitepaper is not publicly available

1.1 Motivation

The following section explores the motivations behind this project in further detail. It describes the challenges outlined previously in the introduction.

1.1.1 Different languages

Professionals and the general public use quite different languages to describe public space. The language of the professionals is more technical. The users may not be familiar with this language and be more colloquial. This leads to the creation of more vocabularies that are different but describe the same objects, problems, and situations.

When designing public space, professionals are also more descriptive of the area and its context - they describe the materials used, the layout of the public space, and its functionality. Not all users think about public spaces in these ways - they evaluate and describe the space based on their feelings and experiences.

This work aims to create a tool that supports discovering the differences in these languages and vocabularies and makes communication between professionals and the general public clearer.

1.1.2 Usage differences

Similar to designing any product, service, or application, the creators may intend the use of the space in a different way than how the user may actually use it.

Let us discuss an example - the Virgin Mary Square in Prague. There are bollards on the Virgin Mary Square to stop cars from driving through. But because they are wide and in the shade most of the day, people use them for sitting down.

More examples can be found not far away in Old Town Square. In the Old

Town Square, there is the Virgin Mary column which is surrounded by stairs. The purpose of the stairs is to access the column but their actual use is once again for sitting down.

An excellent example can also be found in the lower half of Wenceslas Square. There is a fountain whose purpose is to decorate and bring joy. But in the hot summer months, it is used by children, pets, and sometimes adults for cooling down.

If these differences were detected earlier in the design process, it could either bring more intricate designs or uncover potential flaws in the usage of the designed spaces.

1.1.3 Shortening the feedback loop

The goal of this thesis is to create a tool for presenting early designs and gathering feedback from potential users. Providing such a tool will help with shortening the feedback loop.

A feedback loop is part of a system in which some portion (or all) of the system's output is used as input for future operations [1]. It has a minimum of four phases:

1. Input creation
2. Capturing input
3. Input analysis
4. Decision making

By shortening the feedback loop of designing public spaces one can achieve cost reduction and their designs will better meet the needs and expectations of people who will use them. Similar to motivations outlined in section 1.1.2; if all the feedback was gathered earlier it could allow for more agile development.

Chapter 2

Analysis

2.1 Existing solutions analysis

When designing an application or any solution, it is wise to research existing solutions. This research can provide us with many insights into our work - for example, features that we might be missing or validation that the features we are trying to implement in our tool are unique. The core features of this work are the following:

- Display of designs with medium scope. Designs with medium scope are designs of public spaces - plazas, streets, and parks. On the other hand, the smaller scope would be designing a house, the larger scope would be planning of development of the entire city district.
- Collaboration within a team. In today's interconnected world, where everything is designed by larger teams it is important to let the team collaborate seamlessly.
- Sharing of the designs to gather user feedback. The most important feature of this tool is to share the design to gather user feedback. This sharing must be easy and anyone from the general public must be able to accept the share and leave feedback. A future feature may also be editing the design to suit their vision. However, this editing will be very limited - for example only furnishings of the public space.

It is important to mention that the goal of this work is not to implement another Geographic Information System (GIS) or software for architectural design. This has a few reasons - as we will discover in the following sections, there are plenty of tools that allow this functionality. Also, GIS and software for architectural design is a very complex task that would require much more resources to develop. Although it may seem that the goal of this work is to develop such software from the specification of the last feature, it's not - if the tool will support editing of the designs in the future, it will be very primitive.

This feature set implies the following criteria when researching existing solution analysis:

- Scope of the design produced by the software
- Collaboration capabilities inside of the team of professionals working in design
- Support for feedback + how the design can be displayed
- Ease of use

2.1.1 Tools with the capability of designing public spaces

The following section describes software aimed to create architectural designs. I have explored them to discover what they offer - my main goal was to discover their in-team collaboration and feedback-gathering features.

SketchUp

SketchUp is a 3D design software that aims to make 3D modeling for everyone [2]. One of the main strengths of SketchUp is its simplicity and accessibility. The scope of designs SketchUp is designed to produce is much smaller than what is the scope of my tool - SketchUp is focused on the design of buildings and their immediate surroundings. Support for sharing within the team and iterative design is great. SketchUp does not offer a direct tool for gathering feedback for designs however it offers exporting to multiple formats. One thing that is slightly limiting is that you need different price plans to be able to export to different formats. augmented reality in the Go pricing plan, experience reality (XR) in the Pro pricing plan, and animations and 360 panoramas in the Studio pricing plan. Pricing ranges from free to $699/year. Each plan differs in features.

ArcGIS CityEngine

ArcGIS CityEngine is advanced 3D modeling software for creating massive, interactive, and immersive urban environments [3]. It is developed by ESRI, which makes it highly compatible with other ESRI tools. This is important because ESRI is a major player in developing GIS software and has a monopoly to an extent in this field. This may be seen as a pro, because if one is a user of ESRI products, one may use their rich ecosystem. On the other hand, if you are not a user of ESRI products, it may be seen as a negative, because as described in chapter 2.2, their ecosystem and file formats are closed and quite undocumented.

The scope of the designs created in ArcGIS CityEngine is much greater than what the tool created as part of the work will be able to display. ArcGIS CityEngine is focused on designing whole city districts.

One of the notable features of ArcGIS CityEngine is its focus on iterative design and ease of sharing. But the sharing feature of ArcGIS CityEngine is not perfect. It allows publishing models on the web, but it does not offer a direct feature to input feedback from users. Next to publishing to the web ArcGIS CityEngine allows exporting to a number of different formats and platforms - it can be exported to VR and even game engines.

Another great feature of ArcGIS CityEngine is its immersivness. It allows importing of existing real-world geographic data and displaying them next to the design. This allows for an understanding of the design in the context of the area in which it will be located.

ArcGIS city engine does not share the pricing for their products quote can be obtained via request.

CityCAD

CityCAD is a city information modeling solution for exploring, testing, and communicating urban masterplans [4]. Compared to other tools it does not seem to support collaboration within a team, but it tries to sell the idea of communicating of the design with the public.

In terms of scope, CityCAD produces designs with greater scope than compared to the tool I will be developing as part of this work - similar to ArcGIS CityEngine it is designed to design whole city districts. In pricing, it is comparable to other alternatives' prices at £15 per month.

2.1.2 Tools with notable features

Because some of the features required by my application (collaboration, sharing) are similar to features of applications that are not necessarily developed to facilitate the designing of public spaces, they have also been included in the research. The following section explores a few notable features of chosen apps.

Figma

Figma is a tool for designing user interfaces (UIs). There was inspiration from this tool in two areas. First is the UI design of the tool. The second one is the ease of sharing the designs for user testing. With a click of a button, an interactive prototype can be shared with users. Users are able to leave feedback for it in the form of leaving comments.

Google office suite

Google office suite is a tool for creating text documents, spreadsheets, presentations, and more. The most notable feature in the context of this work is the sharing of documents. One can share it in three modes and with different access rights.

- **Viewing** - in this mode, users are only able to see the document and leave comments.
- **Suggesting** - in this mode, users can suggest changes, but they don't have the right to write into the document directly
- **Editing** - in this mode, users can edit the document

This concept was taken from the Google Office suite and ported to the created tool.

2.1.3 Discussion about existing solutions

As we have seen in the previous section, existing solutions often lack the feature of sharing a design with the end user seamlessly. The only software implementing this feature is ArcGIS CityEngine, but it does not support a direct feature for feedback gathering, only for sharing.

As one would expect, all the mentioned software supports architectural design. The explored software also allows easy exporting of the designs to various formats. As stated in 2.1, it would also be quite foolish and resource-consuming to develop another tool supporting the editing of architectural designs. That is the reason why the developed tool will accept designs exported from already existing software and focus solely on ease of feedback gathering. It is therefore necessary to explore file formats that the software is capable of exporting and choose the most suitable one.

2.2 File types analysis

Designs of public spaces by professionals can enter the application in various formats. There are many proprietary and open formats that can be used for the representation of geospatial data. The author of proprietary file types is mainly ESRI. Because they are closed and their documentation is often inaccessible, they are not suitable for this work. In this work, the analysis of the open standard formats and assessment of their pros and cons considering requirements was performed. The assessed requirements are:

- **F1 - geospatial data**. Formats must support the geospatial representation of public spaces
- **F2 - naming and annotations**. The format must support the naming and annotation of features in public spaces. This is required so professionals can leave their naming and intended use. On the other hand, the users then can review these annotations and discover the usages.
- **F3 - creation**. The format must be easy to create and understand.
- **F4 - web ready**. The format should be easy to display in some form - top-down or in 3D by a JavaScript library in the browser.
- **F5 - saveability**. Because users may eventually be able to edit the designs and save them, the format should be easily editable and saveable by the developer and the underlying code.

Analysis and comparison of the most often used open formats for storing geospatial data - shapefile, geopackage, and geoJSON were conducted.

2.2.1 Shapefile

Shapefile is a geospatial vector data format for GIS. It was created by ESRI, which has published its open specification. Although the term "shapefile" is common, technically shapefile is just the file with a .shp extension, it is not standalone and actually consists of a collection of files with the same prefix and multiple file extensions. Shapefile has three mandatory files and a number of optional files. Because optional files are not key for understanding shapefiles, I have omitted them. If required, they can be found online[1].

[1]`https://en.wikipedia.org/wiki/Shapefile`

Shapefile mandatory files

- .shp - This is the main file that stores the geometry for the vector features. It consists of a list of the x,y coordinates for the vertices of the features.
- .shx - This is the index file that stores the index of the geometry for the features in the .shp file. It is used to efficiently access the features in the .shp file
- .dbf - This is the attribute file that stores the attribute data for the features in the shapefile. It is a database file that contains one record for each feature, with each record containing one or more fields of attribute data

Shapefile limitations

Because shapefile is an older file format, it has many limitations and shortcomings. First, it is unable to represent topology[2] of saved geometry. Shapefiles have many geometry limitations, but they are not as important for this project as attribute limitations. Relevant attribute limitations are the following:

- Shapefile stores numeric attributes in character format, not a binary format. This may lead to rounding errors when using numbers containing decimal places
- The dBASE file standard supports only ANSI characters for field names and values
- Date fields only support date
- Field names cannot be longer than 10 characters

[2]Topology - a branch of mathematics, sometimes referred to as “rubber sheet geometry,” in which two objects are considered equivalent if they can be continuously deformed into one another through such motions in space as bending, twisting, stretching, and shrinking while disallowing tearing apart or gluing together parts.[5]

2.2.2 Geopackage

Geopackage is an open format that was created to allow the representation of raster and vector data. It was created in 2012 by Open Geospatial Consortium (OGC) as a replacement for the Shapefile. It is based on SQLite database. Compared to shapefile it allows the following:

- Storing of multiple layers - shapefile is limited to a single layer
- Store vector and raster data - shapefile is limited to vector data
- It uses SQLite which is more modern and efficient than the DBF database used by Shapefile

Geopackage is all stored in a single SQL database file with a .dpkg extension. The fact that this format is stored in a single file makes geopackage more portable than shapefile. The format is widely supported by GIS software.

Geopackage limitations

Geopackage is a relatively new format designed to address the limitations of shapefile, so it does not have many limitations itself. However geopackage is a complex file format, so the main limitation in the context of this project is the reading and writing of this file format.

2.2.3 GeoJSON

GeoJSON is an open data format building upon an already existing JSON format. It has no difference in syntax to JSON but it differs in properties required for each object.

Every geoJSON object has two required properties. Property `type` and then property `geometry` or `features` depending on the value of the `type` property. If the `type` property has the value “FeatureCollection”, it is a collection and the other property of the object is called `features`. The value of the property `features` is an array containing all the features in this collection. If the property `type` has the value of "Feature" then the other property is called `geometry`. The value of the property geometry is an object and has two properties - `type` and `coordinates`. The property `type` can have one of the following values: "Point", "MultiPoint", "LineString", "MultiLineString", "Polygon", "MultiPolygon" or "GeometryCollection". The value of the property `coordinates` is an array of positions of a given feature. Each entry in the `coordinates` array is another array. Its length is two for features in 2D space, and three for features in 3D space.

Each object in geoJSON may have a `properties` property. It can contain custom properties with custom values that are supported by JSON. `properties` contain metainformation about each Feature or FeatureCollection. An example of geoJSON file format may be seen in listing 2.1.

GeoJSON limitations

The biggest limitation of the geoJSON file format may be its file size and speed. This limitation becomes significant once the geoJSON file contains a lot of data. But because the designs of public spaces used for this project do not contain as many features, this limitation is not of concern - for example, the Virgin Mary Sqare design used as an example file for this project contains lower hundreds of features. The combined size of the whole project represented in geoJSON file format is 195.5kB.

Listing 2.1: An example of geoJSON file format

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [102.0, 0.5]
      },
      "properties": {
        "prop0": "value0"
      }
    }
}
```

2.2.4 Suitable file format discussion

All file formats fit the requirement for representing geospatial data.

The most suitable formats for including naming and annotation are geoJSON and geopackage, shapefile is in this area quite insufficient. Although geoJSON specification does not include rules for naming and annotations, there are specifications extending geoJSON, such as simplestyle-spec [6].

All mentioned file formats are easily exportable from commonly used GIS software - all of them fit the requirement for ease of creation. GeoJSON is the clear winner regarding the requirement for easy understanding. It is human-readable and editable. All other formats are binary, so without proper software, they are not editable by humans.

In regards to ease of display in a browser, the geoJSON is a clear winner. Since it is based on JSON, it is natively supported by JavaScript. There are JavaScript libraries for working with shapefiles[3] and geopackages[4] however they are limited. Both found libraries return geoJSON as the value after loading Shapefile which begs the question of why not start with geoJSON in the first place. The library for loading geopackage has insufficient documentation and even after laborious commissioning, it didn't work properly.

The last requirement is the option to edit and save the designs by the user. Because shapefile and geopackage are complex formats, geoJSON is a clear winner in this regard. All libraries that were found for loading shapefiles or geopackages are only one way - they are able to read the format, but not save it.

To see the requirements and the properties of each file format clearly, I have prepared the table 2.1.

From the table 2.1 can be seen that the geoJSON file format is the best for this application. Initially, an extended specification described in section 2.2.5 was created. However, further developments in the project showed that the specification is overly complex. The new discoveries are described in 2.2.6.

[3]Libraries for loading shapefile - mbostock/shapefile, calvinmetcalf/shapefile-js
[4]Library for loading geopackagengageoint/geopackage-js

	Shapefile	Geopackage	GeoJSON
F1	Yes	Yes	Yes
F2	Limited	Yes	Yes
F3	Yes	Yes	Yes
F4	Limited	Limited	Yes
F5	No	No	Yes

Table 2.1: Requirements for input file types

2.2.5 File format specification

Because the geoJSON format supports most of the requirements outlined at the beginning of chapter 2.2 it was used as the base for the file format specification. However, it does not have a standardized way of identifying feature types or their set of properties. Hence a simple yet powerful standard of doing such a thing with geoJSON was created.

Each object of type FeatureCollection adds to properties required property `classes`, which is an array describing possible features in the collection. Each object that is not of type FeatureCollection has then a special property in `properties` based on its class.

Class object

Each class object has the following required properties: `name`, `description`, `identifierKey`, `identifierType`, and `identifierValue`.

- `name` - Name is the name of the class
- `description` - Description is the description of the feature
- `identifierKey` - IdentifierKey describes the name of the property, by which it is determined what class the feature is
- `identifierType` - IdentifierType describes if the IdentifierValue is of type string or number
- `identifierValue` - IdentifierValue is the value of the IdentifierKey property which links the given feature to this class

To make things clear, take a look at listing 2.2. It is an example of geoJSON with the extended syntax representing two features of two classes. The first object is of class `Bench` that is identified by property `code` that has a value of `0`. The second object is of class `Tree` that is identified by property `uuid` that has value of `123e4567e89b12d3a456426614174000`

Listing 2.2: An example of geoJSON extended by my specification

```
{
 "type": "FeatureCollection",
 "features": [{
   "type": "Point",
   "geometry": [ 10,10 ],
   "properties": {
     "code": 0
   }
 },
   {
     "type": "Point",
     "geometry": [ 10,20 ],
     "properties": {
       "uuid": "123e4567-e89b-12d3-a456-426614174000"
     }
   }
 ],
 "properties": {
   "classes": [{
     "name": "Bench",
     "description": "This feature is for resting.",
     "identifierKey": "code",
     "identifierType": "number",
     "identifierValue": 0
   },{
     "name": "Tree",
     "description": "This feature provides shade for citizens.",
     "identifierKey": "uuid",
     "identifierType": "string",
     "identifierValue": "123e4567-e89b-12d3-a456-426614174000"
   }]
 }
}
```

2.2.6 Later discoveries

Based on discussions later in the development it was discovered that the custom file format is excessive because of multiple reasons. First, it is quite complicated to add to the files - no widely used GIS supports adding it to the geoJSON file natively. Second, the standard is more complicated than it needs to be. Third, it did not reflect the fact that another tool[5] will be used for term names and descriptions. It created unnecessary redundancy in the system.

The natural development was the introduction of the concept of identifier key and identifier value. It borrows a lot from the original concept but it is easier to implement into the designs and to understand. Now what is identifier key and identifier value?

The Identifier key is a key in each feature's properties whose value dictates the feature's purpose and function. For example, all benches will have the same identifier value. All keys of "properties" in the geoJSON file (or multiple files) are considered possible candidates for identifier keys.

The identifier key can be an arbitrary string following the JSON rules. Its values can be once again arbitrary strings or numbers. The Identifier key has to follow the following rules:

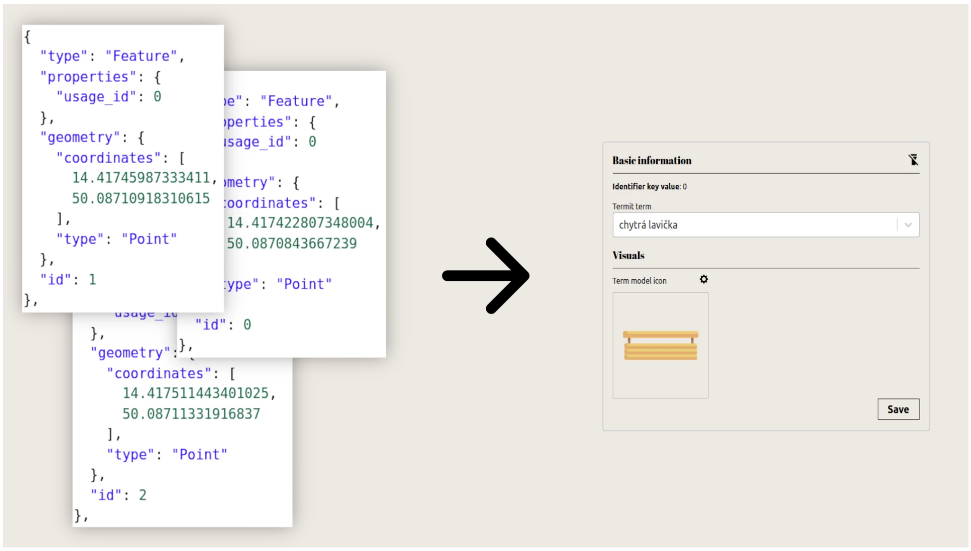
- It has to be on 100% of the project features. If it is not on 100% of the features, the design can be imported, but import results in the unwanted grouping of features into terms.

The identifier value has to follow the following rules:

- It has to be the same for features with the same purpose (Example: it was chosen that the identifier key is usage_id and the value for a bench is "0". Then all benches in the design must have usage_id "0")

An example of how the identifier key works can be seen in figure 2.1. It shows three features with the usage_id value of 0. They all get grouped into the bench term.

[5]See 3.6.3



```
{
  "type": "Feature",
  "properties": {
    "usage_id": 0
  },
  "geometry": {
    "coordinates": [
      14.41745987333411,
      50.08710918310615
    ],
    "type": "Point"
  },
  "id": 1
},
```

```
e": "Feature",
perties": {
sage_id": 0

metry": {
oordinates": [
 14.417422807348004,
 50.0870843667239

ype": "Point"

 "id": 0
},
```

```
  },
  "geometry": {
    "coordinates": [
      14.417511443401025,
      50.08711331916837
    ],
    "type": "Point"
  },
  "id": 2
},
```

Basic information

Identifier key value: 0

Termit term: chytrá lavička

Visuals

Figure 2.1: Identifier key illustration (Own photo, 2023)

2.3 Analysis, feasibility, and benefits of imaging methods

When starting to work on this thesis, it seemed that the team of professionals would like to present their designs in other ways than mere floor plans of the public spaces. Therefore I've conducted an analysis of different methods of displaying said designs. Later it emerged that the imported designs do not have to include any display data and the imaging methods can be implemented through the parker app.

2.3.1 Top-down

A top-down display is like displaying designs on a map. Of all the displaying methods it is the simplest one to implement including the feature of allowing the user to modify the design. One major drawback of this method is that it is not as immersive as the other ones and the feedback provider must use their imagination a lot.

Support for top-down view in the browser

Support for top-down view on the web is quite rich and there are multiple libraries supporting it - adding map overlay to the app can be achieved using multiple libraries. An example is Google maps library or Leaflet.

2.3.2 3D

3D allows for the creation of more immersive and lifelike representations of objects and environments. The 3D view is a great compromise between the complexity of implementation, user experience, and immersiveness of the design. It is not as complex in terms of implementation compared to AR/VR, but is definitely more challenging than the top-down view.

Support for 3D in the browser

Support for displaying 3D graphics in the browser is strong. 3D content can be viewed in a web browser using Web Graphics Library (WebGL) and/or Canvas. These technologies allow users to view 3D objects in the browser without the need for specialized software.

WebGL is a JavaScript API that allows the creation of 3D graphics and visualizations using the GPU of the user's device. Compared to Canvas, WebGL has a lower-level API, which allows the rendering of primitives - points, lines, and triangles. The developer needs to add supply code for more complex features, such as surfaces, shaders, or lights [7]. Note that WebGL also renders to canvas, which Canvas API uses, but canvas element and Canvas API are different.

Canvas API is also a JavaScript API for rendering 2D and 3D graphics in the browser [8]. Compared to WebGL has a higher level of abstraction - it allows supports rendering of more complex graphical elements, such as polygons, self-intercepting shapes, gradients, and patterns. It is not hardware accelerated, so it is slower than WebGL.

To solve the problem of WebGL being such a low-level API, libraries for working with 3D graphics exist. One of these libraries is three.js. Three.js allows working with 3D graphics even for developers that have little to no experience with programming 3D graphics. It contains a set of tools such as preset cameras, lighting, and shaders.

2.3.3 VR

Virtual Reality (VR) is an advanced, human-computer interface that simulates a realistic environment. The participants can move around in the virtual world. They can see it from different angles, reach into it, grab it and reshape it. There is no little screen of symbols for manipulation nor commands to be entered to get the computer to do something. [9]

In VR the user can explore the world through a combination of visual, auditory, and haptic stimuli. This is enabled by VR headsets, which allow the user to explore virtual environments as if they were present.

Technical challenges of VR

1. Face count of input 3D models - topology and mesh of the 3D can be optimized in order to lower the count of vertices and thus lower the number of faces in the model.
2. Material and texture quality - the materials and textures of the model can be lowered. A simple method to achieve this is to lower the resolution of textures.

VR requires high framerate to avoid user dizziness. It is therefore required that 3D models are lowered in polygon count and materials and textures are downscaled. This would allow seeing the designs in VR even on devices with less performant GPUs [10][11].

Support for VR in the browser

Because libraries and technologies are practically the same for VR and AR in the browser, see subsection 2.3.4.

2.3.4 AR

We define Augmented Reality (AR) as a real-time direct or indirect view of a physical real-world environment that has been enhanced/augmented by adding virtual computer-generated information to it [12].

Augmented reality is even more immersive if it is displayed in place of a given public space. Different weather and lighting conditions could be tested, although VR could achieve this to an extent, AR is superior in this regard.

However, AR also comes with a set of challenges. Some of them are shared with VR - those are the lowering of polygon count in input 3D models and downscaling of material and texture quality to allow steady framerate and avoid dizziness of the user. But the greatest technical challenge of AR in the context of this work is finding an anchor point for the displayed 3D model. Model in AR can be anchored in two ways - using a marker (image, 3D shape, or 3D model) or SLAM technology (Simultaneous Localization And Mapping). SLAM works by calculating device position and scanning the environment and anchoring the objects on found planes. That way objects appear as if they were in the environment [13] [14]. Both methods are limiting in some way. The AR anchoring is limiting in that the anchor needs to be visible to the device, so some angles may not work at all because no markers will be in view. SLAM has very limited accuracy in terms of placing the object in space. Both methods can be optimized but the optimization is ofter space-specific - each space would need its own optimization.

Support for AR in the browser

VR/AR content can be viewed in a web browser using WebXR API. But one must take into account that for viewing VR/AR content in the browser specialized hardware is required - this hardware being a headset.

WebXR Device API is an API for displaying VR and AR on the web. Although augmented and virtual realities are now quite a buzzword and seem to be booming with the creation of metaverses and other XR[6] experiences, WebXR Device API is in an experimental stage and according to caniuse.com[7] it is supported by only 73.76% of browsers [15].

[6]XR is a term encapsulating virtual, augmented and mixed realities
[7]caniuse.com is a website that lists what percentage of devices support given javascript CSS or HTML API

That the WebXR Device API is not widely adopted can be seen from the difficulty of trying to make it work. First of all, the user needs bleeding edge version of the browser. Next, the user needs to activate this experimental feature. Last but not least user needs a dedicated hardware - a headset. The bottom line is that the user experiance (UX) of enabling AR and VR on the web is not great. The technology could be used, but its UX is not ready for mainstream use in the browser yet [16].

2.3.5 Discussion about imaging methods

In terms of immersiveness VR and AR are the best candidates. But we have to take into account development time, complexity, and end-user experience. The easiest implementation is the top-down view. 3D is achievable and because it is a de facto standard in all existing platforms, its implementation should be pursued. architects are insufficient to be displayed in another way than a top-down view.

In terms of user experience, top-down view and 3D are winners. They do not require any special hardware or software, and are widely supported by browsers. From a user experience standpoint, when implemented correctly, they "just work".

So the ideal displaying method for this work is a top-down view, with 3D being possible for near-future development. If the tool is widely used then support for VR and AR could be discussed, but it is not a priority for now.

2.3.6 Analysis summary

The analysis brought many insights into the development. Exploration of existing solutions helped with shaping the function requirements. The evaluation of the file types helped uncover that **geoJSON** will be the go-to file type when importing files. Last but not least, the study of the imaging methods showed that the initial imaging method will be a **top-down** view with 3D being a near-future possibility. With this acquired expertise, the implementation could begin.

Chapter 3

Design and implementation

The application that was created as a part of this work is intended to be used by architects and members of the general public. It is therefore necessary to be intuitive and accessible.

It was decided that the application will be a web application. The reasoning is twofold - first, we wanted to allow the usage of the application by as many people as possible. Second, to allow for inputting feedback it is necessary to make the designs of public spaces easily shareable.

The following chapter describes the design and implementation of the web app. It was chosen to call the app “parker” and it is referred to it in the work by this term.

3.1 Functional requirements

The following section describes functional requirements. They have been divided into groups based on their importance for the app. The groups are identified by color squares (🟩,🟧,🟥) and the meaning of the squares is following:

- 🟩 **Must have**. Requirements in this group are absolutely necessary for the functioning and completeness of the application.
- 🟧 **Nice to have**. Requirements in this group are not necessary for the functioning of the application. The difference between nice to haves and perks is their achievability. Nice to haves are easier to implement than perks.
- 🟥 **Perk**. Requirements in this group are more plans for future development than goals to be achieved within this work. It is however important to list them so this thesis can be worked upon in the future.

FREQ 1 🟩 Authentication. Users will be able to perform tasks expected by a web application, meaning registration and logging in. Since authentication will be provided by login using google, password reset functionality is not needed.

FREQ 2 🟩 Model import. Professionals will be able to import public space models in geoJSON format.

FREQ 2.1 🟩 Model versioning. One model is able to have more versions. This allows for iterative approach to designing public spaces.

FREQ 3 🟩 Model editing by professionals

FREQ 3.1 🟩 Features editing. Users will be able to edit imported features of the design. The properties of the features that are editable are name, description, Termit URI, image, and 3D model.

FREQ 3.2 🟧 Collaboration. Professinals will be able to collaborate on a project. This allows for large teams to work on one design and gather feedback easier and quicker

FREQ 4 🟥 Model editing by general public

FREQ 4.1 🟥 Top-down editing. In this mode, users will be able to edit the design to suit their needs in a top-down view of the design.

FREQ 4.2 🟥 3D editing. In this mode, users will be able to edit the design to suit their needs in a 3D view of the design.

FREQ 4.3 🟥 AR editing. In this mode, users will be able to edit the design to suit their needs in an AR view of the design.

FREQ 5 🟩 Model sharing. Professionals will be able to share the design with the general public. It is possible in three sharing modes:

FREQ 5.1 🟩 Feedback providing mode. In feedback providing mode users are allowed to add comments to the shared design. They can see their comments left on the model previously and are able to see replies left to their comments.

FREQ 5.2 🟩 Editing mode. In editing mode, users are able to do the same as in feedback providing mode. This mode also allows seeing all the comments left on the design, editing the terms, adding and deleting project versions and sharing the project to more users.

FREQ 5.3 🟩 Owner mode. In owner mode, users are able to do the same as in editing mode. This mode also allows deletion of the project.

FREQ 6 🟩 Model viewing.

FREQ 6.1 🟩 Topdown viewer. Users will be able to view public space designs by professionals in topdown view.

FREQ 6.2 🟧 3D viewer. Users will be able to view public space designs by professionals in 3D view.

FREQ 6.3 🟥 AR viewer

FREQ 7 🟧 Notifications. Users will receive notification when an action in the app occurs - feedback will be left or design will be shared with them.

3.2 Non-functional requirements

The following section describes non-functional requirements. They have been divided into three groups the same way functional requirements have been. For more information about the division, check out 3.1.

NREQ 1 🟧 Documentation. Because the application could be extended by other people, it should be documented well.

NREQ 2 🟩 Internationalization. To allow more users to use the app the app will have support for internalization. First version will support Czech and English.

NREQ 3 🟩 Dockerization. The app will run on the school provided server. To avoid problems with different environment and runtime versions, and problems of “it works on my machine” in general, the app must be dockerized.

3.3 Use cases

As a part of this work use case diagrams were modeled. In figure 3.1 actors of this application can be seen. The rest of the use cases can be found in the attachments.

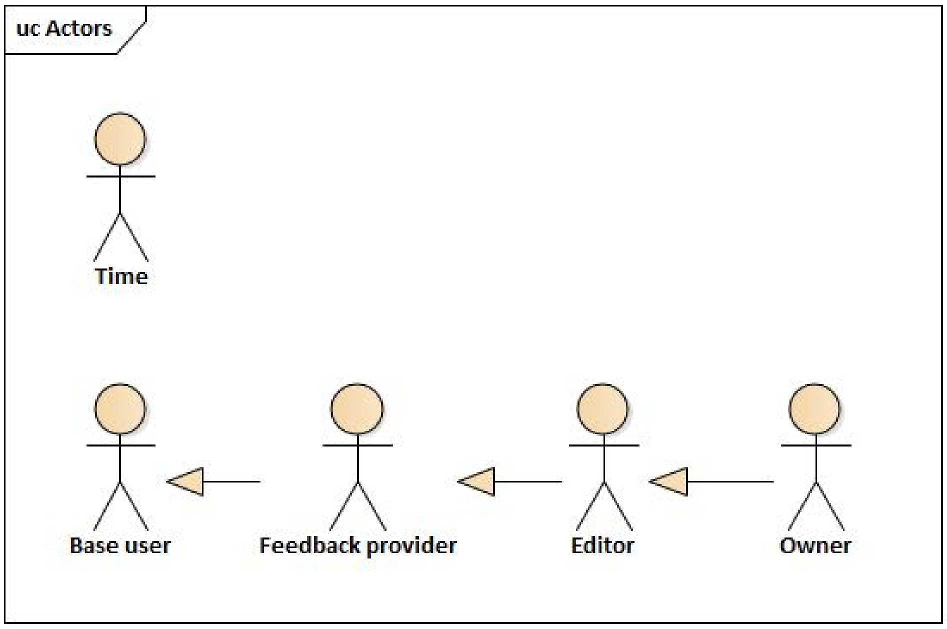


Figure 3.1: Actors in parker (Own image, 2023)

3.4 User interfaces

The following section describes the processes and ideas behind parker's user interfaces. The main idea behind the user interfaces was for the professionals and feedback providers not to be overwhelmed by it so they can be solely focused on the design and feedback for public spaces. The user interface is minimalistic with elements of brutalism.

3.4.1 Design system

A design system is a set of standards to manage design at scale by reducing redundancy while creating a shared language and visual consistency across different pages and channels [17].

It can provide many benefits - it streamlines and speeds up the design process. In figure 3.2 design system of the parker app can be seen. Once the design system of the project was established, work on the screens could begin.

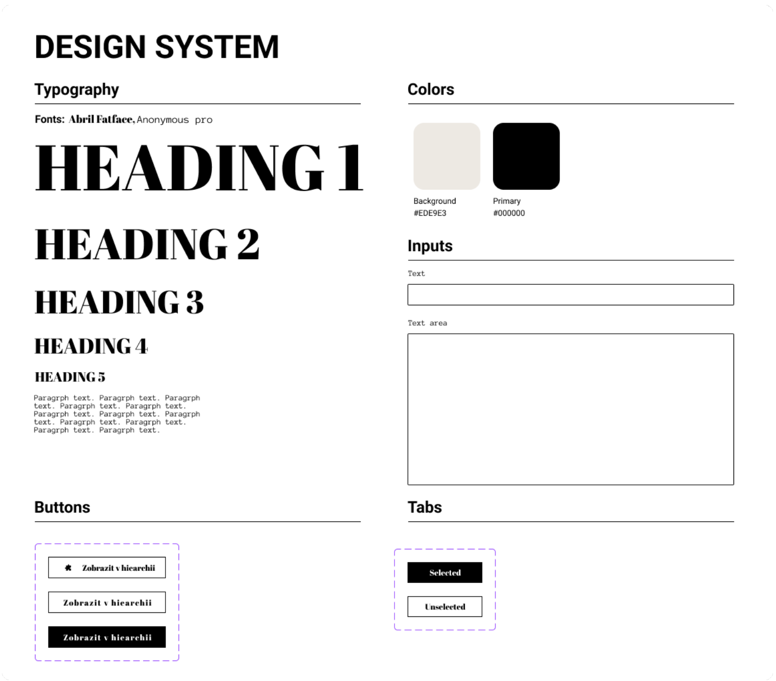


Figure 3.2: Parker design system (Own image, 2023)

3.4.2 Screens

The following section describes the key screens of the application. Because of legibility, the user interface design can be found in the attachments. The screens of the application are the following:

- **Landing screen** - screen with basic information about the app
- **Login screen** - screen where the user can log in
- **Dashboard** - screen with projects the user can edit or give feedback to
- **Import flow** - a set of screens that allows professionals to import their designs. The screens in the import flow are:
  - **File selection and identifier key selection screen** - in this screen, the file can be chosen and the identifier key selected
  - **Term mapping screen** - in this screen, the features in the map are mapped to terms
  - **Import complete screen** - this is an import summary screen. In this screen, a project can be shared or the user can navigate to the imported version.
- **Project detail** - screen with detail of a project. Editors and owners can edit and share the project, feedback providers can review the project and its versions
- **Project version detail** - screen with detail of a project version. Editors and owners can edit the version, its term groups and terms, and feedback providers can browse the version and its terms
- **Designer** - the screen where everyone can leave their feedback

The diagram 3.3 maps the screen hierarchy for logged-in user. For its primitiveness, the screen hierarchy for logged-out users is included as an attachment. Upon logging in the user gains access to the dashboard where he or she may manage his or her projects.

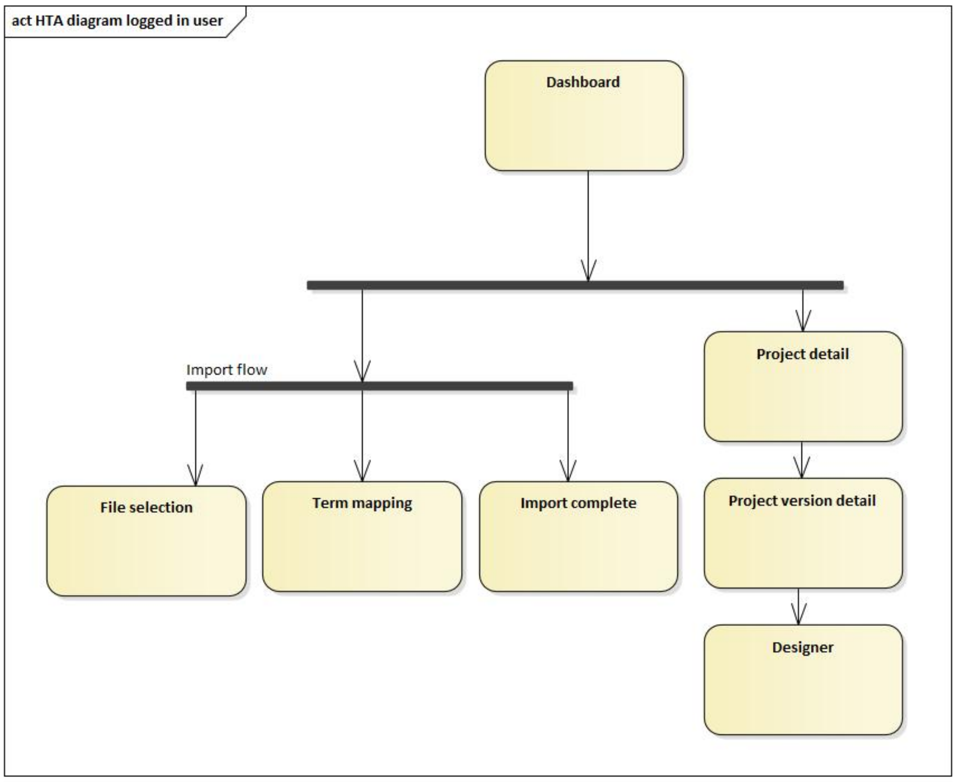


Figure 3.3: Screen map for logged-in user. (Own image, 2023)

3.5 Model

The following section describes the key entities of the model. The model diagram overviewing all entities can be seen in figure 3.4. A detailed diagram can be found in attachements.

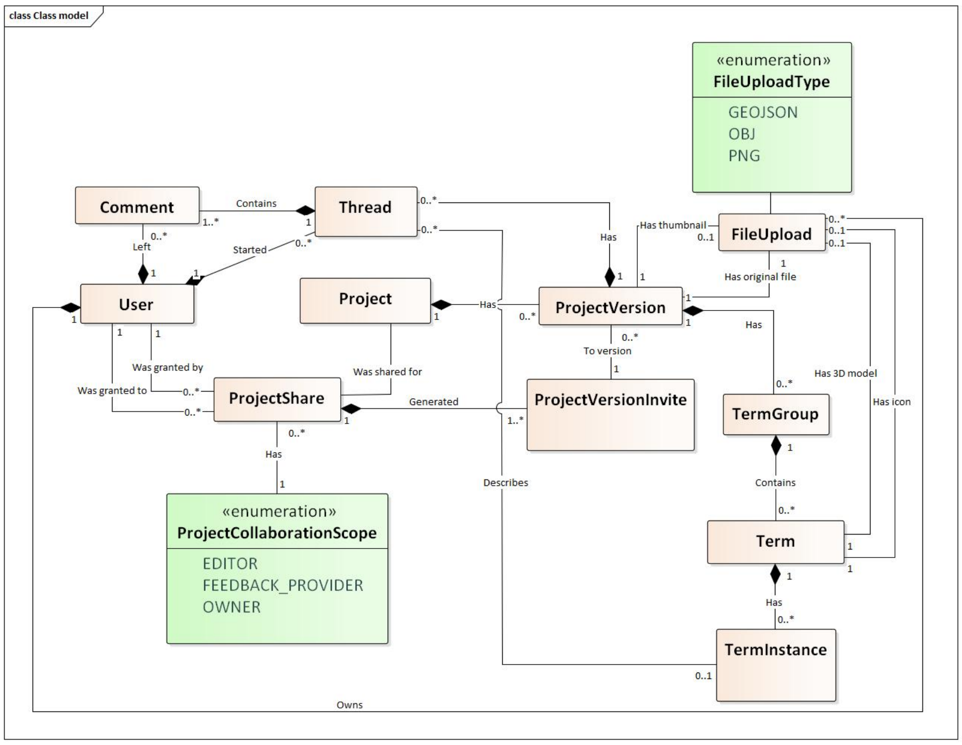


Figure 3.4: Parker class model (Own image, 2023)

The key entities of the model are as follows:

- **User** represents the people using the application. It contains information such as email, name, and other vital information about the user.
- **Project** represents architectonic project. The project contains information such as name and description. The project contains a collection of versions. This allows for easy versioning. The project also can be shared.
- **Project version** represents the individual project version. It contains information such as name and description. The version also has multiple term groups and threads.
- **Term group** is a grouping of terms. Upon import, every file imported gets transformed into a term group. It contains a collection of terms occurring in the design.
- **Term** is to a term instance is what class is to an instance in programming. Each term represents a parent feature for features with the same identifier key value. The term can have annotation from an external service, an icon, and a 3D model. It also has multiple instances of it.

- **Term instance** is what every feature in the imported geoJSONs maps onto. It contains information such as latitude and longitude.
- **Thread** is a collection of comments on coordinates.
- **Comment** is content left by a feedback provider or a response from the project editor or owner.

3.6 Technologies

The following section discusses technologies chosen for the implementation of this work.

3.6.1 Backend and frontend

In today's world, the technologies offered for web development feel almost endless. It was chosen to go with full-stack next.js for a few reasons:

1. Author of the work wanted to try building a full-stack production app using next.js
2. Author of the work has currently the most experience building with next.js
3. The whole stack is JavaScript (or TypeScript). There is therefore no need to switch contexts completely when jumping between frontend and backend
4. Next.js is a React framework. React is the most widely used JavaScript framework for building user interfaces on the web today. It has a large community and support [18]
5. Most importantly, it solves some of the most difficult cross-cutting concerns out of the box - most notably authentication in the form of next-auth.js library
6. The prototype was written using this technology. The prototype provided a solid base for future development

Overall the choice to use the next.js framework as a full-stack framework leads to one key result: The development time will be significantly reduced, so user feedback can be gathered as soon as possible and the viability of the project can be evaluated.

3.6.2 Data storage

This application stores data in two forms - it stores its own business data in a database and it will be connected to TermIt service[1] for term annotations.

The database for business logic was chosen as MySQL. It was chosen because the author of the app has the most experience with the technology. An argument can be made why PostgreSQL was not used for the database with its support for JSON storage and indexing [19]. However, there is no need for such functionality in this app - JSON will be stored as files on the server, and searching within them will not be needed.

3.6.3 External services

The application will be working with two external services - TermIt and Google auth service.

TermIt is a tool to manage vocabularies, terms, and resources in which are the terms defined and used [20]. Inside TermIt, a vocabulary of terms will be created. In the parker application, this vocabulary will be used to annotate terms. This allows the standardization of the annotations which can later be used to provide an analysis of features of the public space. The backend part of the app will communicate with TermIt via its REST API.

Google auth server will be used as an authorization server in the app's login flow. The auth flow here is a classic OAuth2.0 flow.

[1]See 3.6.3 for more information

3.7 Architecture

The following section describes how the technologies described in the previous chapter work together. The architecture of the backend part of the application is a traditional three-layered monolithic web application [21]. It was not chosen randomly, but the decision has solid reasoning behind it:

1. The team behind this work is a solo developer. This favors monolithic architecture since it does not create an organizational and maintenance overhead.
2. Three-layered architecture is the go-to architecture for smaller projects. It offers a fair balance between the maintainability of the code and the boilerplate code required to keep the structure of the project [22].
3. The app will have one very specific function. It does not make sense to split it into something complex eg. microservices.

The backend exposes REST API that is consumed by the frontend application. REST API was chosen because the author of this work is the most familiar with it. The architecture of the entire app can be seen in figure 3.5.

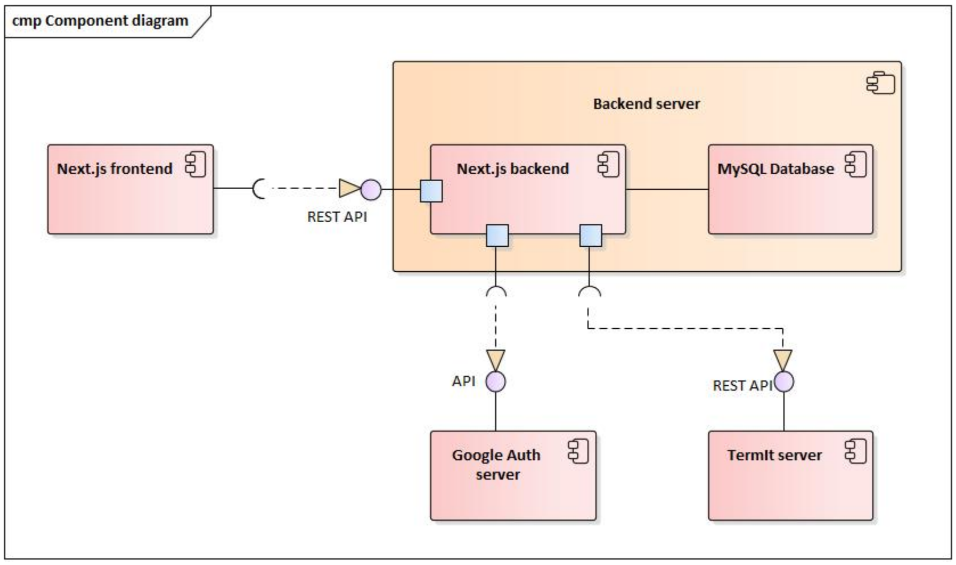


Figure 3.5: Diagram of the components of the app (Own image, 2023)

3.7.1 Deployment diagram

The following figure 3.6 describes the deployment structure of the parker app.

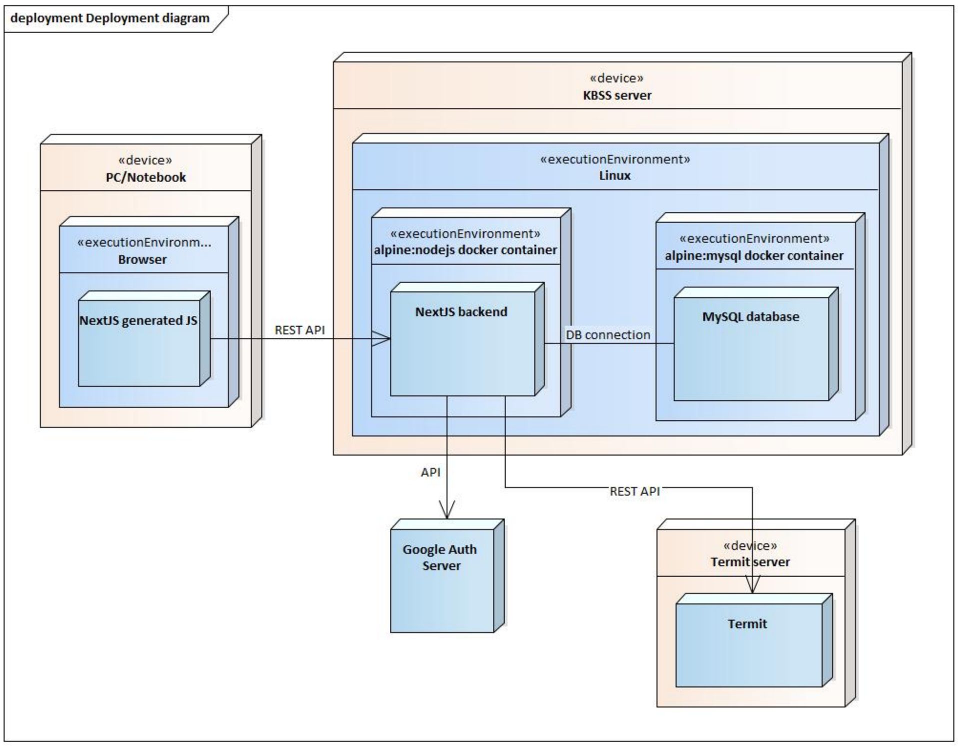


Figure 3.6: Deployment diagram (Own image, 2023)

Chapter 4

User Testing

User testing is essential for the design process of user-centered applications. By user testing authors of digital products gather valuable insight into how users interact with prototypes and discover potential usability issues. In the following chapter various types of user testing will be discussed and one will be chosen. Test groups will be established and test scenarios will be introduced. Finally, the feedback will be gathered and evaluated and the next steps will be decided.

4.1 Choosing the right strategy

Dimensions in which user testing can be divided can be numerous, for example, qualitative vs. quantitative testing, and on-site vs. off-site testing, or moderated and unmoderated testing. this work will discuss only qualitative vs. quantitative because other types usually depend on the choice of qualitative or quantitative testing.

4.1.1 Qualitative testing

Qualitative testing evaluates each participant's needs, emotions, and expectations [23]. It usually takes more time and effort, but in return, the person conducting the test gets detailed information and feedback. Tools of this type of testing are interviews focus groups and observation.

4.1.2 Quantitative testing

Quantitave testing on the other hand focuses on numerical data. Its aim is to analyze large groups of users, gather the data, and focus on the trends of the data.
Tools for gathering quantitative testing are questionnaires and polls [23].

4.1.3 Right strategy discussion

For the purposes of this work, a questionnaire was created to ease the gathering of feedback. The test is therefore mostly quantitative however it contains qualitative elements - open-ended questions where participants can add anything to their experience.

4.2 User groups

Users and scenarios are divided into following two groups.

- **Professionals.** In this group are mainly architects and designers of the public space. The task for the professionals was to import their design and share it for feedback gathering.
- **General public.** Members of the general public is anyone who would like to give feedback to public spaces. Their task was to simulate giving feedback to the provided design.

4.2.1 Screening

As a part of the test each participant filled in a short set of questions to get basic information about them. Information gathered includes age, role in testing (professional or general public), professional background, and whether they have even basic knowledge about the tool and its function.

4.3 Scenarios

The following section maps out the scenarios that users will be subjected to. Each test scenario has its unique code (UTS-P<number> - user test scenario for professionals and UTS-GP<number> - user test for the general public). It has also a name summarizing the activities in the scenario. The scenarios for the professionals can be found online[1].

4.3.1 Professionals scenarios

All the participants were asked to do the following tasks in order:

1. **UTS-P1 Basics**
   Log into the app and change the language to your preferred language.
2. **UTS-P2 Project creation**
   Import your design as a new project. The design contains three terms - benches, bushes and bollards. Add annotations to the terms, add their icons, and modify the dimensions of the icons. Change the name of the project and add a short description of it. Then give a name to the version and add a short description as well. Finish the import and return to the dashboard.
3. **UTS-P3 Project editing**
   You found out that there was a miscommunication between the urbanist that prepared the file and you. Change the properties of the terms.
4. **UTS-P4 Sharing for collaboration and adding feedback providers**
   Navigate to any of your projects and add other users. Add one as editor and add another as feedback provider. Ask the feedback provider to leave feedback on the design.
5. **UTS-P5 Finding the feedback**
   Find the feedback left by the feedback provider. Reply anything in any of the threads in the design.

[1] `https://forms.gle/fihCmE6KP9MtFp9F7`

6. **UTS-P6 Adding project version**
   Navigate to any of the projects and add a new version to it. The version is much simpler, only buildings were left. Finish the import and return to project version editing.
7. **UTS-P7 New version notification**
   Notify the feedback providers about the new version of the project.

4.3.2 General public scenarios

All the participants were asked to do the following tasks in order:

1. **UTS-GP1 Basics**
   Log into the app and change the language to preferred language.
2. **UTS-GP2 Leaving feedback**
   Find a project that was shared with you for feedback providing. The feedback can be left anywhere in the design and its content can also be anything. Return to the dashboard.
3. **UTS-GP3 Leaving more feedback**
   After a while, you remembered that you forgot something. Add another comment to the same thread you left before. The content can again be anything.

4.4 Test report

This section describes the user experience of the participants.

Participant 1

Role: Professional
The test: Participant One had multiple issues with the app. He expected the terms to be highlighted upon clicking on the map in the import flow. He suggested that there should be an explicit "Save" button in project edit and project version edit because users tend not to trust auto-save features.

Participant 2

Role: Professional
The test: The participant mentioned that the language switch is hard to find in preferences. Also, this participant found it slightly confusing that the "Save" button of the edit term form does not disappear after a successful save. Moreover, he noticed a few flaws in the Czech translation. This led to participants thinking that some of the functionality of the app did not work. Once this was sorted out, all functionality worked as expected.
The second participant also came with interesting quality of-life improvement - new versions of the same project would remember the identifier key. Also if the identifier key would be the same in the new version, term instances with the same identifier values across versions would annotate with annotations from the previous version.

Participant 3

Role: Professional
The test: This participant also mentioned the location of the language switcher - according to the testee it could be moved to the main page. This participant did not see or did not hear about how the app should work beforehand. This is important because it provided a view of how users might perceive the app seeing it for the first time. It brought many key insights. He had trouble understanding and finding stuff in general:

- When asked to edit the imported project, he edited the version instead of the project.
- He had trouble locating how to edit term icon dimensions.
- The option to edit the rotation of term instances seemed missing for him. He discovered it at the end of the testing.
- He did not understand what the "Notify users" meant. He tried clicking it before he was asked to do so to test the functionality.

It is clear from this user's input that the app needs to be explained more clearly. There is a tutorial for leaving feedback but this is not enough. The user eventually gets the app, but a description would help him to understand it the first time.

Participant 4

Role: Professional
The test: This participant had a few issues with the import flow. The partaker noticed that with a large enough project, a problem with long scrolling in the import flow could occur, suggesting that the term containers could be collapsible. The test taker also noted that the highlight feature of the terms is slightly confusing stating that the purpose of this button is often completely opposite - clicking a similar icon in Figma for example hides the item. In the project version window, the participant found it confusing that the edit icon is so far away from the edited term.

Participant 5

Role: General public
The test: This participant was overall confused with the navigation to the feedback - namely with the versioning of the project. The participant also found a few UX-related problems:

- Replying to a thread using enter is not possible and that is confusing - it is possible in various other places in the app.
- The participant would expect the basic controls of the designer to be displayed constantly

Participant 6

Role: General public
The test: This participant pointed out a few flaws and misunderstandings in the designer. First, it was noted that when a comment is added it may hide the features under it. It was not clear to the partaker what the outlines of the buildings are for. During this test, the app was accessed from Google Chrome which uncovered a platform-specific flaw - the scrollbars in the right bar in the designer are visible at all times, even when the container is not scrollable. The participant would also expect the arrows next to the project name and project version to be explained more clearly.

4.4.1 Participant 7

Role: General public
The test: This participant did the testing on an iPad. Although the work did not aim to implement responsive design, it brought insights that can be applicable to the desktop as well.
The partaker's test was really thorough. The feedback started on landing where the illegibility of the font was brought up in the "How it works section". In the project detail and project version detail the participant noted that in the information hierarchy in the left bar is unclear, suggesting adding more margin to differentiate between information.
In the designer, the partaker would expect the information about the project and the terms to be collapsible. This makes sense because on iPad there is not enough real estate to accommodate all of the UI. This is a feature that should be implemented in the desktop version as well.
The participant did not expect the behavior of the arrows next to the project and project version names. Since the arrows are pointing right the expected behavior was to obtain more information, not get redirected to the project or project version.
As one of the previous participants, this participant also found the highlight feature confusing.
The last note by the partaker was to add a text when the user has no projects assigned to them - now it may be slightly confusing.

4.4.2 Evaluating the results

As expected, the user testing provided valuable insight into how different users might use and understand the app. It also brought many suggestions from the users and offered a few ideas for future development of the application. The suggestions that will be worked in the app as a part of this work are:

- When a user clicks on a feature in the map, the term will be highlighted in the term list
- Explicit "Save" button in the project edit form and project version edit forms will be added.
- The save button of a term will appear when the term is edited

Plans for future development based on user testing:

- When importing a project with the same possible identifier key, the identifier key will be selected by default
- If the same identifier key will be selected, the user will get asked if he wants to map the terms with the same identifier key value shall be to the same TermIt terms
- More tutorials will be added. This includes but is not limited to:
  - Tutorial for term mapping
  - Tutorial describing key project and project version concepts - the versioning aspect itself, sharing and notifications
- The thread bubbles will be hideable
- Term overview and project information will be collapsible
- The highlight feature will be probably kept for the import flow - in this use case it makes sense, because the professional may be overwhelmed by the number of undefined terms, but in the designer the feature is confusing. The fact that the icon looks like a thrash icon and not a flashlight only helps to deepen the problem
- A test for different browsers must be conducted. None of the problems with different browsers was deal breaking but it should be improved to provide the same experience for everyone
- A language switch will be moved to a location that is easier to find

Chapter 5

Summary

The work described the problem of the inability of professionals to gather user feedback for user spaces efficiently and on a large scale. It explored the options of representing geospatial data digitally. It then selected a file type suitable for importing the designs of the professionals.

A thorough analysis of existing solutions shaped and extended function requirements. Because most of the available software displays designs in 3D and professionals required more complex imaging methods, it was obvious that there is a need for analysis of imaging methods.

A proof-of-concept app was developed. The app fully covers the functional requirements marked as “must haves”. It implements one “Nice to have” feature. The work included deploying the app into a real-world environment. The app was thoroughly user tested. This uncovered a few flaws in the user design which were swiftly fixed or were noted for future development.

Overall the work served as a great introduction to geospatial software and provides a solid base for future development in the area of gathering feedback on public spaces.

Appendix A

Bibliography

[1] L. Fitzgibbons, “What is feedback loop?: Definition from techtarget.” https://www.techtarget.com/searchitchannel/definition/feedback-loop, 2019. Accessed 22. December 2022.

[2] Trimble, “3D Design Software | 3D Modeling on the Web.” http://www.sketchup.com/page/homepage. Accessed: 10. January 2023.

[3] ESRI, “GIS Mapping Software, Location Intelligence & Spatial Analytics.” https://www.esri.com/en-us/home. Accessed: 10. January 2023.

[4] “CityCAD 3.0 - a conceptual masterplanning solution for urban designers.” https://www.holisticcity.co.uk/services/citycad/. Accessed: 1. October 2022.

[5] S. C. Carlson, “Topology.” https://www.britannica.com/science/topology, 2022. Accessed: 14. December 2022.

[6] Mapbox team, “Mapbox/simplestyle-spec: A simple styling convention for geojson data.” https://github.com/mapbox/simplestyle-spec. Accessed 8. January 2023.

[7] MDN Contributors, “WebGL tutorial - Web APIs | MDN.” https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial. Accessed: 5. January 2023.

[8] MDN Contributors, “Canvas API - Web APIs | MDN.” https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API. Accessed: 5. January 2023.

[9] J. Zheng, K. Chan, and I. Gibson, “Virtual reality,” *IEEE Potentials*, vol. 17, no. 2, 1998. doi: 10.1109/45.666641.

[10] N. Fernandez, "10 best practices when optimizing 3D files for VR." `https://meshmatic3d.com/technical/optimize-3d-files-ar-vr/`, 2020. Accessed 5. January 2023.

[11] P. Šimek, *Prohlížečka 3D modelů ve virtuální realitě*. Czech Technical University, 2022.

[12] J. Carmigniani, *Augmented Reality: An Overview*. Springer New York, 2011. doi: 10.1007/978-1-4614-0064-6_1.

[13] J. Batt, "Understanding anchors in augmented reality experiences: Learning solutions." `https://www.learningguild.com/articles/understanding-anchors-in-augmented-reality-experiences/`. Accessed: 5. January 2023.

[14] Blippar, "What is slam?." `https://www.blippar.com/blog/2021/10/01/what-is-slam`. Accessed 5. January 2023.

[15] MDN Contributors, "WebXR Device API - Web APIs | MDN." `https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API`. Accessed: 5. January 2023.

[16] MDN Contributors, "Starting up and shutting down a WebXR session - Web APIs | MDN." `https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API/Startup_and_shutdown`. Accessed: 5. January 2023.

[17] Therese Fessenden, "Design systems 101." `https://www.nngroup.com/articles/design-systems-101/`, 2021. Accessed 27. April 2023.

[18] tkrotoff, "Front-end frameworks popularity (React, Vue, Angular and Svelte)." `https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190`. Accessed 18. May 2023.

[19] PostgreSQL team, "PostreSQL JSON types." `https://www.postgresql.org/docs/15/datatype-json.html`, 2023. Accessed 6. May 2023.

[20] Knowledge-based and Software Systems Group, "Termit." `https://kbss-cvut.github.io/termit-web/`. Accessed 18. May 2023.

[21] IBM, "What is Three-Tier Architecture." `https://www.ibm.com/topics/three-tier-architecture`. Accessed 18. May 2023.

[22] V. Ostrenko, "Comparing Three-Layered and Clean Architecture for Web Development." `https://betterprogramming.pub/comparing-three-layered-and-clean-architecture-for-web-development-533bda5a` Accessed 18. May 2023.

[23] R. Costa, "Quantitative vs qualitative testing: validating our steps." `https://www.justinmind.com/blog/qualitative-quantitative-testing/`, 2021. Accessed 25. April 2023.

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Mrázek Hugo**	Personal ID number:	**478429**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Software Engineering and Technology**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

A Tool for Public Space Use Modeling

Bachelor’s thesis title in Czech:

Nástroj pro modelování využití veřejného prostoru

Guidelines:

1. Become familiar with the topic of representing geographical data in software systems.
2. Analyze the possibilities of visualization of map data and storing geographical models using these data.
3. Design a software tool that will allow authorized users to publish digital models of public spaces and annotate various architectural elements in them (such as benches, street lamps, etc.). The general public should be able to view and provide feedback to these digital models.
4. Implement the tool.
5. Evaluate the tool with user testing. Select two groups of subjects and test the tool both from the side of public space creators and users.

Bibliography / sources:

[1] Egenhofer, M. J. (2002). Toward the Semantic Geospatial Web. Proceedings of the Tenth ACM International Symposium on Advances in Geographic Information Systems - GIS ’02, 1–4.
[2] Christopher, A. et al. (1977). A Pattern Language: Towns, Buildings, Construction. New York: Oxford University Press.
[3] Michal Med, Petr Křemen: Context-based ontology for urban data integration. iiWAS 2017: 457-461

Name and workplace of bachelor’s thesis supervisor:

Ing. Martin Ledvinka, Ph.D. Knowledge-based Software Systems FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **26.01.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Martin Ledvinka, Ph.D.	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature		Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.
Date of assignment receipt | Student’s signature