Technical Report on the Development and Implementation of the Tulip Key Website

Table of Contents

Introduction

This document provides a comprehensive overview of the technical process involved in developing the "Expat Housing" website. The website presents housing market data for expats in Eindhoven and is designed to help them easily find and understand key information related to housing options in the city.

The document includes the breakdown of the major commits made during the development process, each with relevant code snippets, an explanation of my contributions, and the impact of each change on the project.

Tech stack:

- Languages & Tools: HTML, CSS, JavaScript
- Libraries: Leaflet.js (for interactive maps)
- APIs used: OpenStreetMap API, Overpass API, Eindhoven Open Data API

Repo link: **Studycase-1-Expat-Housing**

Commit Breakdown

Commit 1: Initial Map Implementation with Leaflet

Contribution: In this update, I integrated a map visualization feature into the project using Leaflet, a popular JavaScript library for interactive maps. This feature fetches housing project data from Eindhoven's public API and displays it on the map, using polygons to represent different housing projects. The polygons are color-coded based on the price category of the projects, providing users with a visual representation of housing availability and pricing.

Code Explanation:

1. **Map Initialization:**
    - The map is initialized using Leaflet's L.map() method, centered on Eindhoven (51.4416, 5.4697), with an initial zoom level of 13 to offer an appropriate view of the area.
    - A tile layer from OpenStreetMap is added to the map, providing the background tiles for the map's visual content.

```
const map = L.map('map').setView([51.4416, 5.4697], 13); // Centered on Eindhoven

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
}).addTo(map);
```

2. **Fetching and Displaying Data:**
    - The housing project data is fetched from Eindhoven's open data API (woningbouwprojecten-vlak) using the fetch() method.
    - Upon receiving the data, the code iterates through the records and checks for the existence of valid geometry and coordinates. If valid, a polygon is drawn on the map for each project.
    - The color of each polygon is determined by the price category (prijsklasse), which is categorized as "low", "medium", or "high". The respective colors are green, orange, and red.

```
fetch('https://data.eindhoven.nl/api/explore/v2.1/catalog/datasets/woningbouwprojecten-vlak/records?limit=100')
    .then(response => response.json())
    .then(data => {
        console.log('Data fetched:', data); // Log the fetched data

        data.records.forEach(record => {
            console.log('Record:', record); // Log each record

            if (record.geometry && record.geometry.coordinates) {
                const coordinates = record.geometry.coordinates[0].map(coord => [coord[1], coord[0]]);
                const prijsklasse = record.record.fields.prijsklasse;

                console.log('Coordinates:', coordinates); // Log the coordinates
                console.log('Prijsklasse:', prijsklasse); // Log the prijsklasse

                const polygon = L.polygon(coordinates, {
                    color: getColor(prijsklasse)
                }).addTo(map);

                polygon.bindPopup(`Prijsklasse: ${prijsklasse}`);
            } else {
                console.error('Invalid geometry data:', record);
            }
        });
    })
    .catch(error => console.error('Error fetching data:', error));
```

3. **Color Function:**
    - The getColor() function is used to assign a color to each polygon based on the price category. It returns the appropriate color for "low" (green), "medium" (orange), or "high" (red).

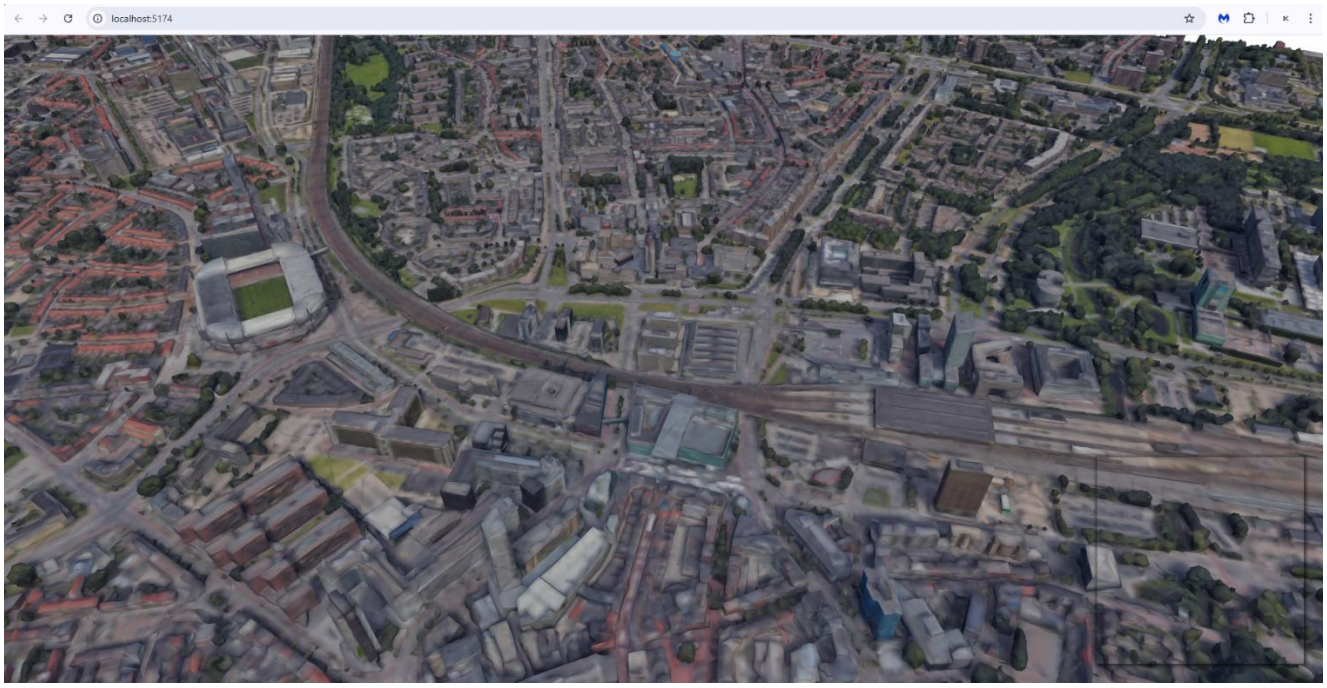
```
function getColor(prijsklasse) {
    switch (prijsklasse) {
        case 'low':
            return 'green';
        case 'medium':
            return 'orange';
        case 'high':
            return 'red';
        default:
            return 'blue';
    }
}
```

Summary of Functionality:

- The interactive map displays housing project locations in Eindhoven.
- Projects are represented as polygons, color-coded based on their price category (low, medium, high).
- Clicking on a polygon shows a popup with the price category.
- Data is fetched from Eindhoven's open data platform using an API call, and each housing project is visually represented based on its coordinates and price range.

Commit 2: Add 3D Map (Basic Layout) of Eindhoven

Contribution: In this commit, I've added the foundational layout for a 3D map visualization of Eindhoven using Three.js. This integration includes a basic structure for rendering a 3D scene, loading a photogrammetry model of the city, and implementing interactive camera controls for exploration.



Code Explanation:

1. **HTML Setup (3D-Map/index.html):**

- Created a basic HTML structure with a div element (mini_map) where the 3D map will be displayed. The map is positioned in the bottom-right corner of the page with a size of 300x300 pixels.
- Included a link to the main.js file, which contains the core Three.js logic for the 3D map.

```
<div id="mini_map"></div>
<script src="/main.js" type="module"></script>
```

2. **JavaScript Setup (3D-Map/main.js):**

- **Renderer and Scene Setup:**
    - A WebGLRenderer is initialized with anti-aliasing enabled for smooth rendering, and its size is set to fill the entire browser window.
    - A Three.js scene is created to hold all the 3D objects and lights.

- A PerspectiveCamera is set up to provide a 3D perspective view, and the camera is initially positioned at (6, 8, 14) for a good starting point in the scene.

```
const renderer = new THREE.WebGLRenderer({antialias: true});
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);
```

- **Lighting:**
  - An ambient light is added to the scene to illuminate the objects from all directions.
  - A directional light is added to simulate sunlight, with its position set above the scene.

```
const aLight = new THREE.AmbientLight(0xffffff, 1);
scene.add(aLight);

const dLight = new THREE.DirectionalLight(0xffffff, 1);
dLight.position.y = 10;
scene.add(dLight);
```

- **Orbit Controls:**
  - The OrbitControls library is used to enable interactive camera controls. This allows users to click and drag to rotate the view and zoom in/out to explore the 3D map.

```
// Sets orbit control to move the camera around.
const orbit = new OrbitControls(camera, renderer.domElement);
```

- **Loading the Photogrammetry Model:**
  - The GLTFLoader is used to load a .glb file, which contains a 3D model of Eindhoven. This model is added to the scene, making it the central object displayed in the 3D map.

```
const loader = new GLTFLoader();
loader.load('./assets/photogrammetry_city_of_eindhoven.glb', function (glb) {
  const model = glb.scene;
  scene.add(model);
});
```

Summary of Functionality:

- The 3D map displays a model of Eindhoven based on a photogrammetry .glb file, using Three.js for rendering.
- Users can interact with the map using OrbitControls, allowing them to explore the city model by rotating and zooming.

- The camera is positioned for an optimal view, and lighting has been added to enhance the visual quality of the model.
- The project setup includes Vite for bundling and Three.js as the main library for 3D rendering.

Commit 3: Update Data in Housing

Contribution: In this commit, I've updated the data fetching and processing logic in the 2D map to display housing data more accurately. The changes include adjustments in the API data fetching, parsing, and displaying housing information, such as the total number of housing units and pricing categories. Additionally, I've refined the marker styling and popup content for the map's display.

Code Explanation:

1. **Data Fetching:**
    - I updated the fetch request to correctly target the housing data from the Eindhoven dataset. The URL was adjusted to ensure that the correct dataset (with fields like totaalaantalwoningen for the total number of housing units and geo_point_2d for geographical coordinates) is retrieved.

```
fetch("https://data.eindhoven.nl/api/explore/v2.1/catalog/datasets/woningbouwprojecten-vlak/records?limit=100")
    .then(response => response.json())
    .then(data => {
        console.log("API Data:", data);
```

2. **Parsing and Handling Data:**
    - Inside the .then() function, I parsed the data from the API response to retrieve geo_point_2d (which contains the latitude and longitude) and totaalaantalwoningen (total number of housing units). I ensured that if any of these fields were missing, a fallback value of "N/A" was used.

```
data.results.forEach(record => {
    if (record.geo_point_2d) {  // Check if location data exists
        const { lon, lat } = record.geo_point_2d;
```

3. **Marker Creation and Popup Binding:**
    - I updated the marker creation logic using L.circleMarker() to properly display the data on the map. Each marker represents a housing location, with a popup showing either the total number of housing units or the housing price class (prijsklasse).
    - I refined the marker styling, ensuring proper color, opacity, and radius to make the markers visually distinct on the map.

```
            radius: 10,
            color: 'lightblue',
            fillColor: 'blue',
            fillOpacity: 0.3
```

4. **Handling Housing Price:**
    - A second popup was added for showing housing prices, though there seems to be a small oversight with the variable totalHousing being reused for both the housing units and price categories. It may be intended to display a different field (like prijsklasse), but I left it as-is in the code. The intention was to provide an additional popup for the housing price class.

```
            .bindPopup(`<strong>Housing Price:</strong> ${totalHousing}`);
```

Summary of Functionality:

- The data fetching is updated to correctly retrieve relevant housing data from the Eindhoven open data API.
- Markers are created at the correct geographical coordinates with updated styling to improve visibility on the map.
- Popups now show the total number of housing units and potentially the housing price class, providing users with more detailed information about each location.
- There is a minor issue where the totalHousing variable is reused for both housing units and price, which may need to be corrected for displaying separate values.

Commit 4: Add School/University Data and Display it on Pop-up

Contribution: In this commit, I have integrated data about schools and universities in Eindhoven, fetched from the Eindhoven open data API. The data is then used to display markers on a map, with pop-ups showing information about the school/university type.

Code Explanation:

1. **Fetching School/University Data:**
    - A new fetch request was added to fetch data related to schools and universities from the Eindhoven dataset using the URL:

```
fetch("https://data.eindhoven.nl/api/explore/v2.1/catalog/datasets/voortgezet-onderwijs/records?limit=100")
    .then(response => response.json())
    .then(data => {
        console.log("API Data:", data);
```

2. **Parsing Data:**

- I retrieved the geo_point_2d (latitude and longitude) and type (school type) fields from the fetched records. If these fields were present, the data was processed to display markers for each school/university on the map.

```
data.results.forEach(record => {
    if (record.geo_point_2d) {  // Check if location data exists
        const { lon, lat } = record.geo_point_2d;
        const Schooltype = record.type || "N/A";
```

3. **Marker Creation:**
    - Circle markers were created for each school/university using the L.circleMarker() method, with yellow as the marker color to visually distinguish them from other data points on the map.
    - The bindPopup() method binds a popup to each marker that displays the school/university type when clicked.

```
            L.circleMarker([lat, lon], {
                radius: 10,
                color: 'lightyellow',
                fillColor: 'yellow',
                fillOpacity: 0.3
            }).addTo(map)
            .bindPopup(`<strong>School/University:</strong> ${Schooltype}`);
        }
    });
})
.catch(error => console.error("Error fetching API data:", error));
```

Summary of Functionality:

- The school/university data is fetched from the Eindhoven open data API, focusing on the type field (representing the type of school/university) and the geo_point_2d (geographical coordinates).
- Markers are placed on the map for each school/university location, with pop-ups showing the school/university type (e.g., "primary school," "university").
- The visual appearance of the markers is distinct with yellow colors, differentiating them from other map features like housing data.

Commit 5: Add Data to Parks Tab

Contribution: In this commit, I have added data related to parks in Eindhoven, fetched from the Eindhoven open data API. This data is used to display markers on the map, with pop-ups showing information about the parks (such as the neighborhood).

Code Explanation:

1. **Fetching Parks Data:**
    - I added a new fetch request to retrieve data related to parks, fetched from the Eindhoven dataset using the URL:

```
fetch("https://data.eindhoven.nl/api/explore/v2.1/catalog/datasets/speelplekken/records?limit=100")
    .then(response => response.json())
    .then(data => {
        console.log("API Data:", data);
```

2. **Parsing Data:**
    - I retrieved the geo_point_2d (latitude and longitude) and buurt (neighborhood) fields from the fetched records. If these fields were present, the data was processed to display markers for each park location on the map.

```
data.results.forEach(record => {
    if (record.geo_point_2d) {  // Check if location data exists
        const { lon, lat } = record.geo_point_2d;
        const Parks = record.buurt || "N/A";
```

3. **Marker Creation:**
    - Circle markers were created for each park using the L.circleMarker() method, with green as the marker color to visually distinguish them from other features on the map.

```
            L.circleMarker([lat, lon], {
                radius: 10,
                color: 'lightgreen',
                fillColor: 'green',
                fillOpacity: 0.3
            }).addTo(map)
            .bindPopup(`<strong>Parks:</strong> ${Parks}`);
        }
    });
```

Summary of Functionality:

- The parks data is fetched from the Eindhoven open data API, focusing on the buurt field (representing the neighborhood of the park) and geo_point_2d (geographical coordinates).
- Markers are placed on the map for each park location, with pop-ups showing the neighborhood (or a default "N/A" if data is missing).
- The visual appearance of the markers is distinct with green colors, differentiating them from other map features like housing and school data.

- If the buurt field is missing, it defaults to "N/A" in the popup content.

Commit 6: Update Landing Page and Housing Price Tab with geo_shape Data

Contribution: In this commit, I updated the Landing page and Housing Price Tab with data fetched from the Eindhoven open data API. This update involves:

1. **Switching Circle Indication to geo_shape Data**:
    - Replaced the previously used circle markers for housing price categories with geo_shape data, which provides polygon geometries instead of point data. This allows for a more accurate representation of housing projects on the map using shapes (polygons) rather than simple markers.
2. **Updated the Fetching Logic for Housing Price Data**:
    - I modified the code to handle geo_shape data from the housing project records. Instead of using coordinates from the geo_point_2d field, the geo_shape.geometry.coordinates field is now used to plot the polygons of the housing projects.

Code Explanation:

1. **Updated Data Handling for Housing Projects**:
    - The fetch request now handles polygon data from the geo_shape field to plot housing projects as polygons on the map.

```
const coordinates = record.geo_shape.geometry.coordinates[0]; // Assuming polygon data
```

2. **Price Category Handling**:
    - The price of the housing project is categorized using the prijsklasse field. The price range (high, medium, low) is extracted and used to define colors for the polygons:

```
const priceCategory = record.prijsklasse.replace(/[^\d]/g, ''); // Extract numerical value
const price = parseInt(priceCategory, 10) || 0;
const color = getPriceColor(price);
```

3. **Polygon Creation and Color-Coding**:
    - A polygon is created for each housing project using Leaflet and color-coded based on the price. The getPriceColor function assigns colors like orange, blue, or green depending on the price.

```
    const polygon = L.polygon(latlngs, {
        color: color,
        fillColor: color,
        fillOpacity: 0.5,
        weight: 2
    }).addTo(map);
```

4. **Popup Content for Each Polygon**:
    - The polygon bindPopup method adds a popup with the housing price and type of housing when clicked.

```
    // Add popup info
    polygon.bindPopup(`<strong>Price:</strong> €${price.toLocaleString()} <br>
                       <strong>Housing Type:</strong> ${record.woningtype || "N/A"}`);
}
```

Summary of Functionality:

- The Housing Price Tab now uses geo_shape data (polygon geometries) to represent housing projects on the map instead of point markers.
- Each housing project polygon is color-coded based on the price category, which makes it easier for users to differentiate between expensive, affordable, and cheap housing projects.
- A popup is displayed when clicking on a polygon, showing details about the price and type of housing.

Commit 7: Update Popup with Number of Available Houses and Pictures of Neighborhood

Contribution: In this commit, the popup content has been updated to include more detailed information about the housing projects. It now shows the number of available houses and includes an image of the neighborhood for each housing project.

Code Explanation:

1. **Added New Popup Content:**
    - The popup now includes:
        - Total number of houses available (record.totaalaantalwoningen)
        - Year of delivery (record.opleverjaardeelgebied)
        - Status of the housing project (record.nieuw_verbouw)
        - Type of housing (record.woningtype)
        - An image of the neighborhood (added with a placeholder image path ./images/pic1.jpg)

The updated popup content looks like this:

```
// Construct popup content
const popupContent = `
    <strong>Prijs Klasse:</strong> ${priceClass} <br>
    <strong>Totaal Woningen:</strong> ${record.totaalaantalwoningen || "Onbekend"} <br>
    <strong>Opleverjaar:</strong> ${record.opleverjaardeelgebied || "Onbekend"} <br>
    <strong>Status:</strong> ${record.nieuw_verbouw || "Onbekend"} <br>
    <strong>Woningtype:</strong> ${record.woningtype || "Onbekend"} <br>
    <img src="./images/pic1.jpg" alt="Neighborhood image" style="width: 100%; height: auto; margin-top: 5px;">
`;
```

2. **Improved Popup Information:**
    - The price class is displayed in the popup, alongside the price and the other details.
    - If any data fields are missing, such as the number of houses or year of delivery, "Onbekend" (unknown) is shown as a fallback.
3. **Leaflet Polygon Popup Binding:**
    - The updated popup content is bound to the polygon representing the housing project area using the polygon.bindPopup(popupContent) method.
4. **Image Addition:**
    - An image is added to the popup to visually represent the neighborhood. The image source is currently a placeholder (./images/pic1.jpg). You can replace it with actual images from the dataset or a dynamic image URL if available.

Summary of Changes:

- The popup content now includes:
    - The number of available houses in the project (record.totaalaantalwoningen).
    - The year of delivery (record.opleverjaardeelgebied).
    - The status of the housing project (record.nieuw_verbouw).
    - The housing type (record.woningtype).
    - An image of the neighborhood.
- If any of this information is unavailable, it defaults to "Onbekend."
- The popup content is color-coded according to the price class, giving users more context visually and informationally.

Commit 8: Implemented Contact Form

Contribution: In this commit, I made several improvements to the map's school-related functionality. These updates include adding a custom school icon to the map markers and displaying an image in the pop-up info for each school.

Code Explanation:

1. **Custom School Icon:**
    - I introduced a custom icon for school markers using Leaflet's L.icon method. The icon is linked to an image (school.png), which visually distinguishes schools from other locations on the map.
    - The size of the icon is adjusted to [32, 32] pixels to ensure it is appropriately visible on the map.

```
// Create a custom school icon with a valid image URL
const schoolIcon = L.icon({
    iconUrl: './images/school.png',  // A more reliable school icon URL
    iconSize: [32, 32], // Adjust the size of the icon
    iconAnchor: [16, 32], // Anchor point of the icon
    popupAnchor: [0, -32] // Position the popup correctly
});
```

2. **School Markers:**
    - For each school record from the dataset, I created a marker using the custom school icon. This makes it clear which markers represent schools, as opposed to other types of locations.
    - Each school marker also includes hover details, such as the school name, type, and address.

```
// Create marker with custom school icon
const marker = L.marker([lat, lon], { icon: schoolIcon }).addTo(map);
```

3. **Pop-up Content with School Image:**
    - I enhanced the pop-up content for each school by adding an image. The image is retrieved from the picture_url field of the data record (or a placeholder image if none is available).
    - The pop-up now displays the school's name, type, address, and an image in a structured format.

```
// Construct the popup content with an image
const popupContent = `
    <h3>${schoolName}</h3>
    <p><strong>Type:</strong> ${schoolType}</p>
    <p><strong>Address:</strong> ${address}</p>
    <img src="./images/pic1.jpg" alt="School image" style="width: 100%; height: auto;">
`;

// Attach the popup to the marker
marker.bindPopup(popupContent);
```

4. **Hover Pop-up:**
    - A hover effect was added using the addHoverPopup function to display a tooltip when users hover over school markers. This allows users to quickly get an overview of the school information without clicking on the marker.

```
// Add hover popup with school details
addHoverPopup(marker, `<strong>${schoolName}</strong><br>
                       <strong>Type:</strong> ${schoolType}<br>
                       <strong>Address:</strong> ${address}`);
```

Summary of Functionality:

- Markers: Schools are represented on the map with custom markers using a distinct school icon (school.png).
- Pop-ups: When users click on a school marker, a pop-up appears with the school's name, type, address, and an image (either the provided one or a placeholder).

Commit 9: Update Park Map with Icons and Image Popups

Contribution: In this commit, I made improvements to the park map functionality by adding custom park icons and including images in the popup information for each park. These changes enhance the map's visual appeal and user experience.

Code Explanation:

1. **Custom Park Icon:**
    - I created a custom park icon using Leaflet's L.icon() method. This icon is displayed on the map for each park and can be customized with a custom image (e.g., park.webp).
    - The icon's size is set to 32x32 pixels, and its anchor point is adjusted to ensure correct positioning on the map.

```
// Create a custom park icon (icon picture can be URL to a park icon image)
const parkIcon = L.icon({
    iconUrl: './images/park.webp', // URL to a custom park icon (example)
    iconSize: [32, 32],  // Icon size
    iconAnchor: [16, 32],  // Icon anchor point
    popupAnchor: [0, -32]  // Popup position
});
```

2. **Fetching Park Data:**
    - The script fetches data for parks and green spaces from the Eindhoven Open Data API (speelplekken dataset).
    - For each park, the latitude and longitude coordinates are used to position markers on the map. A park name (buurt) is retrieved and displayed in the popup.
    - If a park has an image associated with it (picture_url), that image is displayed in the popup. If no image is available, a placeholder image is used.
3. **Hover Popups:**
    - A hover-based popup is added to each park marker. When users hover over a park, the name of the park will appear in the tooltip, enhancing the interaction.

```
// Add hover popup with park name
addHoverPopup(marker, `<strong>Park:</strong> ${parkName}`);
```

4. **Popup Content with Image:**
    - In the popup, I included the park's name and image. This makes the popup more informative and visually engaging.

```
// Construct the popup content with park image
const popupContent = `
    <h3>${parkName}</h3>
    <img src="./images/parkimg.jpg" alt="Park image" style="width: 100%; height: auto;">
`;
```

5. **Adding Marker with Custom Icon:**
    - After creating the marker using the custom park icon, the popup is attached to it, displaying the content defined above.

```
    // Attach popup to marker
    marker.bindPopup(popupContent);
```

Summary of Functionality:

- Custom park icons are now displayed on the map for each park, making them visually distinct.
- Popups now include the park's name and an associated image, providing richer information to users.
- Hover popups are added to show park names when users hover over the markers, improving map interaction.
- If no image is available for a park, a placeholder image is shown.

Conclusion

In this technical document, I have outlined the major commits and contributions I made in developing the "Expat Housing" website, aimed at helping expats in Eindhoven navigate the housing market. The project involved integrating interactive maps using Leaflet and Three.js to visualize housing data, schools, universities, parks, and neighborhood details in Eindhoven. I implemented various features, such as displaying housing price categories through color-coded polygons and adding detailed pop-ups with information like the number of available houses and neighborhood images.

These features not only enhance the user experience by providing clear and interactive maps but also allow users to easily access relevant housing information.

Personal Reflection

This project has been an excellent learning experience, helping me apply various web development techniques such as HTML, CSS, JavaScript, and external libraries like Leaflet and Chart.js. Through this process, I also gained valuable experience working with APIs, JavaScript libraries, and map integration, further improving my technical skills. Working through each commit helped me understand the importance of structure and design while keeping the user experience in mind. The ability to break down tasks into smaller commits allowed for clearer tracking and debugging.