Praktikumsbericht

Aras Ahmad

Matrikelnummer: 2036174
Studiengang: Informatik (B.Sc.)

Bergische Universität Wuppertal

Betreuerin: Frau Dr. Seehagen-Marx

Praktikumszeitraum: 02.04.2025 – 22.05.2025

Unterschrift Studierender

Wuppertal, 20. Oktober 2025

1 Vorstellung der Praktikumsstelle

Mein Praktikum absolvierte ich an der Bergischen Universität Wuppertal, einer der großen öffentlichen Universitäten des Landes Nordrhein-Westfalen. Die Universität wurde 1972 gegründet und zählt heute über 20.000 Studierende in mehr als 100 Studiengängen. Mein Praktikum fand im Bereich der Informatik statt und wurde betreut von Frau Dr. Seehagen-Marx.
Die Universität bietet eine praxisnahe, forschungsorientierte Ausbildung in verschiedenen technischen und geisteswissenschaftlichen Disziplinen. Besonderes Augenmerk wird auf moderne Softwareentwicklung, didaktische Vermittlung digitaler Kompetenzen sowie innovative IT-Projekte gelegt.

Meine Praktikumstätigkeit fand im Rahmen eines universitätsinternen Projekts statt, das sich auf die Entwicklung von webbasierten Anwendungen mit modernen Technologien wie ASP.NET Core MVC, D3.js, Entity Framework und CKEditor fokussierte. Die Projektarbeit erfolgte in engem Austausch mit weiteren Praktikanten und wurde durch regelmäßige Besprechungen mit der Betreuerin begleitet.

Einleitung

Dieser Bericht dokumentiert meine Praktikumszeit an der Bergischen Universität Wuppertal unter der Betreuung von Frau Dr. Seehagen-Marx. Von Anfang April bis Ende Mai 2025 habe ich zwei praxisnahe Entwicklungsprojekte mit ASP.NET Core MVC umgesetzt. Dabei arbeitete ich intensiv mit D3.js für Datenvisualisierung, Entity Framework Core für Datenbankzugriffe und modernen Frontend-Technologien wie Bootstrap und CKEditor. Der folgende Bericht stellt Woche für Woche dar, welche Aufgaben ich bearbeitet habe, welche Fortschritte ich gemacht habe und welche Lösungen ich für verschiedene Herausforderungen gefunden habe.

2 Zusammenfassende Aufgabenübersicht

Im Verlauf meines Praktikums konnte ich an zwei umfangreichen Softwareprojekten mitarbeiten und tiefgreifende Erfahrungen in der praktischen Webentwicklung sammeln. Nachfolgend ist eine Übersicht der durchgeführten Aufgaben und bearbeiteten Themenfelder zusammengestellt:

- Einarbeitung in das .NET-Framework sowie ASP.NET Core MVC
- Analyse von Softwarearchitekturen mit dem MVC-Muster
- Nutzung von Entity Framework Core für Datenbankzugriffe
- Implementierung einer Webanwendung zur Witzverwaltung (`JokesWebApp`)
- Entwicklung einer interaktiven Stadtkarte mit Haltestellen (D3.js + ASP.NET)
- Modellierung relationaler Datenstrukturen (Haltestellen + Verbindungen)
- Visualisierung von Netzwerken und Graphstrukturen mit D3.js
- Erstellung von API-Controllern für Haltestellen- und Verbindungsdaten
- Aufbau einer modularen JavaScript-Komponente (`map.js`)
- Umsetzung von CRUD-Funktionalitäten (Create, Read, Update, Delete)
- Fehleranalyse, Debugging und Verbesserung der Datenintegrität
- Start eines zweiten Projekts: Lernpfad-Plattform mit dynamischer UI
- Gestaltung von Benutzeroberflächen mit Bootstrap 5
- Integration des CKEditor 5 für Rich-Text-Content
- Dynamische Lernschritte: Hinzufügen, Vorschau, Bearbeitung im Frontend
- Implementierung von Datei-Uploads (z.B. Vorschaubilder)
- Erweiterung um Filter-, Such- und Vorschaufunktionen
- Erstellung von Migrations, Seeds und Datenbankmodellen
- Versionsverwaltung, Wiederherstellung und Aktualisierung der Datenbank
- Teilnahme an regelmäßigen Besprechungen und Teamabsprachen

3 Woche 1 (02.04 – 04.04.2025)

02.04.2025

Mein erster Praktikumstag begann mit einem persönlichen Einführungsgespräch mit Frau Dr. Seehagen-Marx. In diesem Gespräch wurde mir erläutert, dass der Schwerpunkt des Praktikums auf der Arbeit mit der Softwareentwicklungsplattform `.NET` liegen würde. Es wurde vereinbart, dass ich mich sowohl theoretisch als auch praktisch mit den Technologien im Microsoft-Umfeld beschäftigen und erste Webanwendungen umsetzen würde.
Noch am selben Tag begann ich mit einem umfassenden Videotutorial zum Thema `.NET`, das aus insgesamt elf Videos bestand. Die Reihe wurde auf dem YouTube-Kanal „Edureka!“ veröffentlicht:
`https://www.youtube.com/watch?v=ZNf0t0G2d1g&list=PL9ooVrP1hQOGwV9LIF_QJQz9Knp-JHx4p&ab_channel=edureka%21`

Ich konnte bereits am ersten Tag drei dieser Videos durcharbeiten. Darin lernte ich die grundlegende Architektur und Komponenten von .NET kennen:

- **.NET Runtime**: Laufzeitumgebung (z.B. CLR – Common Language Runtime)
- **.NET SDK**: Tools, Compiler, Bibliotheken zum Entwickeln von Anwendungen
- **Unterstützte Sprachen**: C#, F#, Visual Basic
- **.NET Libraries**: Große Sammlung an Framework-Funktionen (z.B. für Dateioperationen, Web, Sicherheit)
- **Plattformunabhängigkeit**: Moderne Versionen ab .NET 5 laufen auf Windows, Linux und macOS

Ich lernte zudem, wie sich `.NET Core` zur heutigen `.NET`-Plattform weiterentwickelt hat, während das `.NET Framework` als Windows-exklusive Legacy-Variante existiert. Weitere Varianten sind `ASP.NET` für Webentwicklung und `Xamarin/.NET MAUI` für mobile Apps.

03.04.2025

Am zweiten Tag besprach ich mit Frau Dr. Seehagen-Marx mögliche zukünftige Aufgaben. Ein zentrales Thema war die Beschäftigung mit dem Framework „Next Skills“ für die Erstellung interaktiver Webseiten.

Im Anschluss setzte ich das Tutorial fort. Ich arbeitete mich in folgende Konzepte ein:

- Delegates in C#
- Unterschiede zwischen `C#` und `.NET`

- Konzepte wie Index, Controller, Views und Migrations

Ein ergänzendes Video zu Delegates nutzte ich hier:
`https://www.youtube.com/watch?v=jYjNigSmPE8&t=1s&ab_channel=InterviewHappy`

Parallel begann ich mit einem praktischen Mini-Projekt: eine einfache Webanwendung mit dem Namen `JokesWebApplication`, die es erlaubt, verschiedene Witze mit Antworten anzuzeigen. Benutzer mit einem Login sollen in der Lage sein, neue Witze hinzuzufügen oder vorhandene zu löschen.

Das genutzte Tutorial stammt von freeCodeCamp.org:
`https://www.youtube.com/watch?v=BfEjDD8mWYg&t=1814s&ab_channel=freeCodeCamp.org`

Ich installierte die benötigte Entwicklungsumgebung `Visual Studio Community` sowie das `ASP.NET`-Paket.

04.04.2025

Ich setzte meine Arbeit an der `JokesWebApplication` fort. Ich entschied mich, das Projekt mit dem **MVC-Muster** (Model-View-Controller) zu strukturieren. Das Framework stellt bereits viele vordefinierte Klassen, Datenbank- und Verbindungsstrukturen bereit, sodass sich der Entwicklungsaufwand deutlich reduzierte.
Ich startete die Webanwendung erfolgreich mit `IIS Express` in Visual Studio und testete die erste Benutzeroberfläche.
Anschließend versuchte ich, die Anwendung mit einer Datenbank zu verbinden. Hierfür nutzte ich die `Package Manager Console` von Visual Studio. Dabei traten zunächst Probleme auf:

- Die Konsole war schwer auffindbar – ich fand sie erst über die Suchfunktion.
- Die erste Eingabe des .NET-Migrationsbefehls war fehlerhaft.

Nach Recherche und Tests konnte ich schließlich erfolgreich die Migration anlegen:

```
dotnet ef migrations add InitialCreate
```

Damit war die Verbindung zur Datenbank hergestellt und das Projekt migrationsfähig.
Zum Abschluss der Woche fand ein gemeinsames Treffen mit Frau Dr. Seehagen-Marx und weiteren Praktikanten statt, bei dem wir über mögliche Kooperationen, Herausforderungen und unsere Arbeitsweise sprachen. Die Teamdynamik war von Anfang an offen und kollegial.

4 Woche 2 (07.04 – 11.04.2025)

07.04.2025

Die Woche begann mit einer Besprechung mit einem Praktikumsassistenten, in der erste fachliche Aufgaben vorgestellt wurden. Dabei wurde empfohlen, mich mit den folgenden vier Schlüsselthemen der modernen Webentwicklung auseinanderzusetzen:

- **D3.js** – JavaScript-Bibliothek für interaktive Datenvisualisierung
- **Entity Framework (EF)** – Object-Relational Mapper für .NET
- **ASP.NET Identity** – Benutzer-Authentifizierung und -Verwaltung
- **CRUD** – Grundlagen der Datenbearbeitung: Create, Read, Update, Delete

Ich begann mit einer systematischen Recherche zu diesen Themenbereichen.

08.04.2025

Am zweiten Tag vertiefte ich mein Wissen über D3.js und Entity Framework anhand von Videotutorials. Besonders hilfreich waren:

- `https://www.youtube.com/watch?v=bp2GF8XcJdY` (Fireship – Überblick über D3.js)
- `https://www.youtube.com/watch?v=TOJ9yjvlapY` (Academind – Einstieg in D3.js)

Ich lernte, dass D3.js es erlaubt, DOM-Elemente an Daten zu binden und mit SVG und CSS visuell darzustellen. Besonders interessant fand ich die Flexibilität, selbst komplexe Visualisierungen wie Linien-, Balken-, oder Kreisdiagramme zu erzeugen.
Parallel dazu befasste ich mich mit dem **Entity Framework (EF)** und den Konzepten *Code First* und *Database First*. Dazu nutzte ich folgende Playlist:
`https://www.youtube.com/watch?v=SryQxUeChMc&list=PLdo4fOcmZ0oX7uTkjYwvCJDG2qhcSzwZ6`

09.04.2025

Ich setzte meine Recherchen mit Fokus auf **User Authentication mit ASP.NET Identity** und **CRUD-Funktionalität** fort. Die wichtigsten Erkenntnisse:

- Identity-Framework ermöglicht Registrierung, Login, Rollenverwaltung und Zwei-Faktor-Authentifizierung
- Beispiel für Benutzerregistrierung in C#:

```
var user = new ApplicationUser { UserName = model.Email,
    Email = model.Email };
var result = await _userManager.CreateAsync(user, model.
    Password);
```

- CRUD-Operationen stehen in direktem Zusammenhang mit HTTP-Methoden: POST (Create), GET (Read), PUT/PATCH (Update), DELETE (Delete)

Ich begann wieder an meinem Projekt `JokesWebApp` zu arbeiten. Ich implementierte ein Modell namens `Joke`, das die Felder `JokeQuestion` und `JokeAnswer` enthielt, und erstellte den zugehörigen Controller `JokesController.cs`. Ich erzeugte Datenbanktabellen für Witze und Benutzeraccounts.

10.04.2025

Ein zentrales Ereignis des Tages war die Besprechung mit Frau Dr. Seehagen-Marx und anderen Praktikanten. Dabei wurden Ideen zu einer interaktiven Webseite ausgetauscht, u.a. mit Inspiration durch:
`https://nextskills.org/de/playground/future-skills-finder/`

Ich entwickelte eine neue Projektidee: Eine interaktive Karte mit Knoten, bei denen jeder Knoten eine Bushaltestelle darstellt. Features sollten sein:

- Visualisierung der Haltestellen über D3.js
- Anklickbare Knoten mit Infos und Bild
- Option zur Suche und Navigation
- Kontaktseite für Entwicklerinformationen

Ich entschied mich für ein Fullstack-Projekt mit:

- **Frontend**: HTML, CSS, D3.js
- **Backend**: Node.js mit Express
- **Datenbank**: MySQL

Ich startete die Entwicklung über das Bash-Terminal, richtete Datenbank und Server ein und versuchte, das Projekt auszuführen. Allerdings scheiterte der erste Startversuch des Servers.

11.04.2025

Ich führte weitere Debugging-Schritte am Projekt „Wuppertal Map“ durch. Besonders bei der `index.html`-Datei stellte ich Syntaxfehler fest:

- Falscher Link zu Leaflet-CSS
- Fehlerhafte Marker-Erstellung
- Problem beim Laden der Bushaltestellen-Daten (API-Aufruf)

Ich behob diese Fehler, startete den Server mit `node server.js` neu und öffnete die Webseite im Browser – erfolgreich.
Die Seite zeigte nun:

- Eine Karte von Wuppertal mit zehn interaktiven Haltestellen
- Info-Fenster und Bilder beim Klicken auf Knoten
- Ein Suchfeld für Haltestellen
- Eine verlinkte Kontaktseite mit Bild und Angaben des Entwicklers

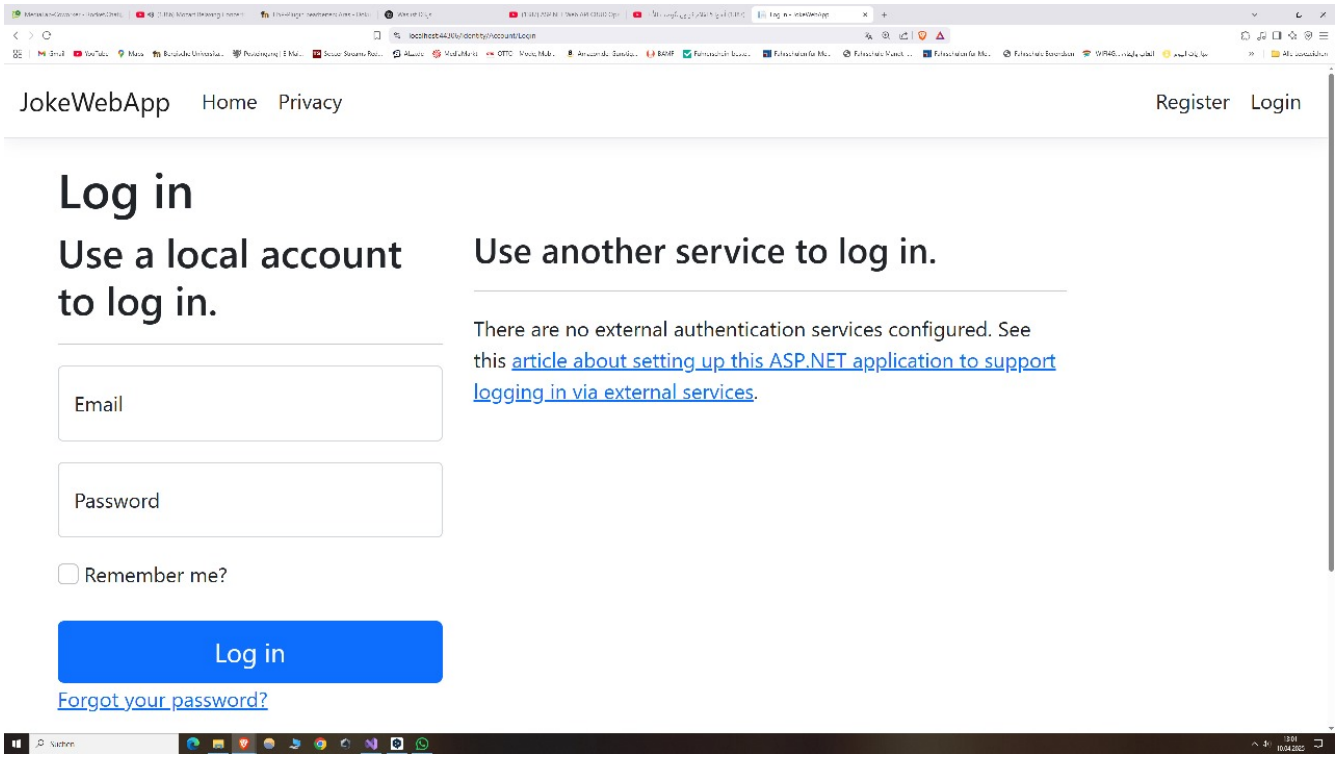

Abbildung 1: Login-Seite der JokeWebApp mit lokalem Authentifizierungsformular

Abbildung 2: Erste Version der Wuppertal-Karte mit visualisierten Haltestellen

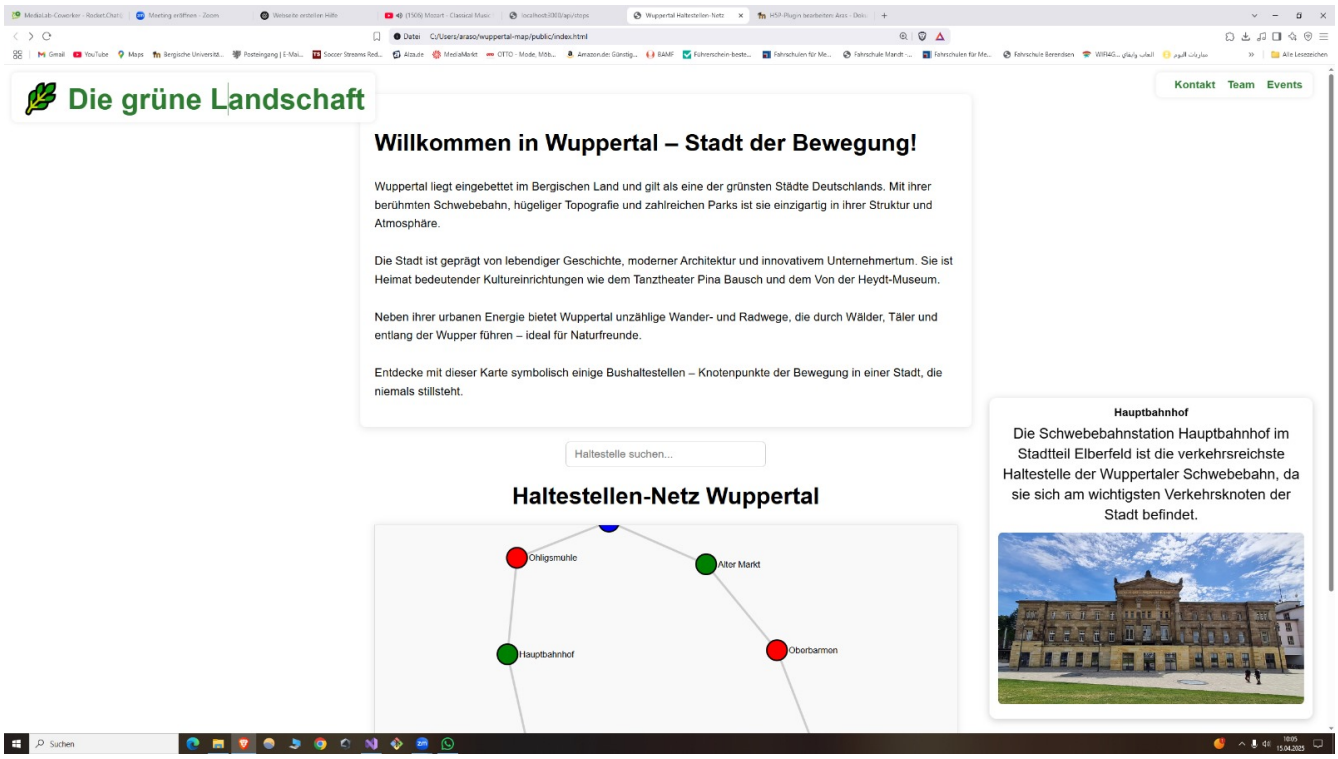

Abbildung 3: Informationsbereich über die Stadt Wuppertal in der Webanwendung

5 Woche 3 (14.04 – 17.04.2025)

14.04.2025

Ich begann mit dem Aufbau der Webanwendung `WuppertalMapMvc` auf Basis von `ASP.NET Core MVC`. Ziel war es, eine interaktive Karte mit Bushaltestellen zu entwickeln, die über ein Frontend (D3.js) visualisiert und mit einer SQL-Datenbank über Entity Framework Core verknüpft wird.
Zu Beginn richtete ich die folgenden Grundkomponenten ein:

- **Controller** – zur Verarbeitung von HTTP-Anfragen
- **Modelle** – z.B. `Stop.cs` zur Beschreibung einer Haltestelle (Name, Position, Linien)
- **Migrations** – zur automatischen Generierung und Aktualisierung der Datenbankstruktur
- **Entity Framework Core** – als ORM zur Datenbankkommunikation

Die vorbereitenden Schritte:

- Neues Projekt `WuppertalMapMvc` angelegt
- Datei `Stop.cs` in `Models/` erstellt
- Datei `AppDbContext.cs` in `Data/` erstellt
- Datei `Program.cs` angepasst, um Datenbankkontext zu konfigurieren

Die Datenbankstruktur wurde initial durch folgende Befehle erzeugt:

```
dotnet tool install --global dotnet-ef
dotnet ef migrations add InitialCreate
dotnet ef database update
```

Das EF Core-Paket installierte ich über NuGet. Daraufhin wurde automatisch die Datenbank `stops.db` mit der Tabelle `Stops` erzeugt.
Ich erstellte den `StopController.cs`, der eine API-Schnittstelle bereitstellt, über die Haltestellen an das Frontend unter `/api/stop` ausgeliefert werden.
Zusätzlich implementierte ich die Klasse `DbInitializer.cs`, mit der Seed-Daten automatisch in die Datenbank eingefügt werden können. Die Methode `DbInitializer.Initialize(db)` wurde im `Program.cs` aufgerufen.
Beim ersten Buildversuch trat ein Fehler auf, der sich an diesem Tag nicht beheben ließ.

15.04.2025

Zunächst fand eine kurze Besprechung mit Frau Dr. Seehagen-Marx statt. Anschließend tauschte ich mich mit einem anderen Praktikanten aus, der ebenfalls an einer interaktiven Webanwendung arbeitete. Wir diskutierten mögliche Umsetzungen, Designideen und technische Ansätze.
Ich untersuchte im Projekt `WuppertalMapMvc` die Fehlerursache weiter. Nach intensiver Analyse stellte sich heraus, dass in der Datei `Program.cs` ein Semikolon in der Import-Anweisung fehlte:

```
using WuppertalMapMvc.Data
```

Nach Korrektur (Semikolon ergänzt) konnte das Projekt erfolgreich kompiliert und gestartet werden.

Allerdings traten neue Probleme in der Oberfläche auf:

- Beim Klick auf Haltestellen erschienen keine Bilder oder Informationen.
- Schriftarten und Styles im unteren Bereich waren fehlerhaft.
- Die Kontakt-Seite öffnete sich nicht korrekt.

Die Fehlerursachen waren:

- Syntaxfehler bei der Bildreferenz:

```
<img src="@Url.Content("~/mein-bild.jpg")" alt="Mein Bild" />
```

- Ein fehlendes Komma nach der Eigenschaft `info` in einem Datenobjekt in JavaScript:

```
info: "Beschreibung",
image: "bild.jpg"
```

Nach Korrekturen funktionierte die Darstellung wieder korrekt.

16.04.2025

Ich tauschte mich mit einem weiteren Praktikanten aus, der ein ähnliches Kartenprojekt mit Flughäfen umsetzte. Wir verglichen unsere Lösungsansätze für Datenmodelle, Benutzeroberflächen und Verbindungslogik.

Anschließend begann ich mit der Implementierung von CRUD-Funktionalitäten für Haltestellen. Ich erstellte den `StopController.cs`, um POST-Anfragen zur Erstellung neuer

Haltestellen zu verarbeiten. Das dazugehörige HTML-Formular `Create.cshtml` wurde ebenfalls erstellt. Es enthält Eingabefelder für:

- Name
- Linien (Lines)
- Latitude / Longitude
- Beschreibung
- Bild

Die Felder sind direkt an das `Stop`-Modell gebunden. Beim Absenden wird das Objekt an die `Create`-Methode im Controller gesendet und in die Datenbank eingefügt. Danach aktualisierte ich das Datenbankschema:

```
dotnet ef migrations add UpdateStopModel
dotnet ef database update
```

Abschließend fügte ich in die Oberfläche eine Option „Haltestellen verwalten“ ein. Diese wurde als Button oben rechts platziert und öffnet eine Verwaltungsseite. Beim Erstellen einer Haltestelle wurde diese erfolgreich auf der Karte dargestellt – allerdings ohne Verbindung zu bestehenden Knoten.

17.04.2025

Zu Beginn fand eine Rücksprache mit Frau Dr. Seehagen-Marx über den bisherigen Fortschritt statt.
Anschließend widmete ich mich dem Feinschliff der Benutzeroberfläche:

- Einführung moderner Schriftarten und Farben (z.B. `font-family: 'Poppins'`)
- Styling der Knoten mit Hover-Animation:

```
circle:hover {
    stroke: black;
    stroke-width: 2px;
    cursor: pointer;
    opacity: 0.85;
}
```

- Integration von Drag-and-Drop:

```
.call(
    d3.drag()
      .on("start", dragstarted)
      .on("drag", dragged)
      .on("end", dragended)
)
```

Ich ergänzte außerdem Tooltips statt fester Infoboxen und passte das Layout für mobile Geräte per `@media`-Queries an.
Um neue Knoten mit bestehenden zu verbinden, erweiterte ich den Code:

- Dropdown-Menü mit allen vorhandenen Knoten befüllen:

```
// Nach .map(...) in startMap()
const dropdown = document.getElementById("connect-to");
nodes.forEach(n => {
   const opt = document.createElement("option");
   opt.value = n.id;
   opt.textContent = n.id;
   dropdown.appendChild(opt);
});
```

- Beim Absenden des Formulars: neuer Knoten wird gespeichert, auf der Karte hinzugefügt und mit dem gewählten verbunden

Ich überprüfte abschließend folgende Punkte:

- Lädt die App erfolgreich?
- Ist die Karte sichtbar?
- Funktionieren die Klick-Events und die API (`/api/stop`)?
- Können Haltestellen hinzugefügt und gelöscht werden?

Ein Problem trat jedoch auf: Nach dem Hinzufügen eines Knotens verschwanden plötzlich alle Knoten. Trotz gültiger Eingaben konnte ich die Ursache an diesem Tag noch nicht vollständig beheben.

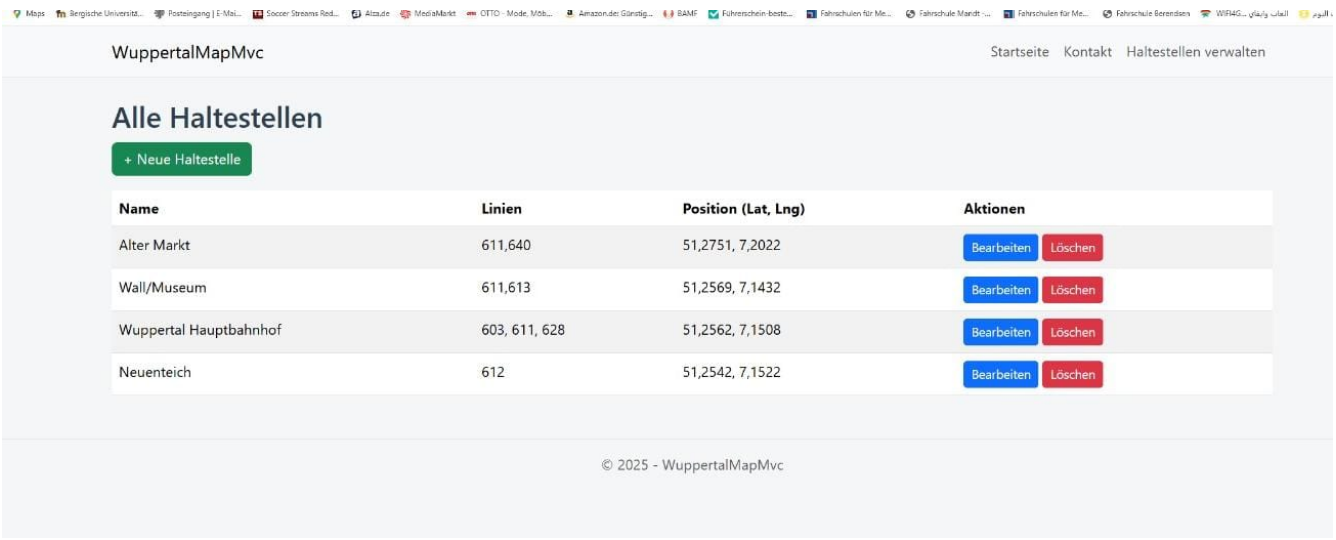

Abbildung 4: Übersicht der Haltestellen in der Webanwendung „WuppertalMapMvc“

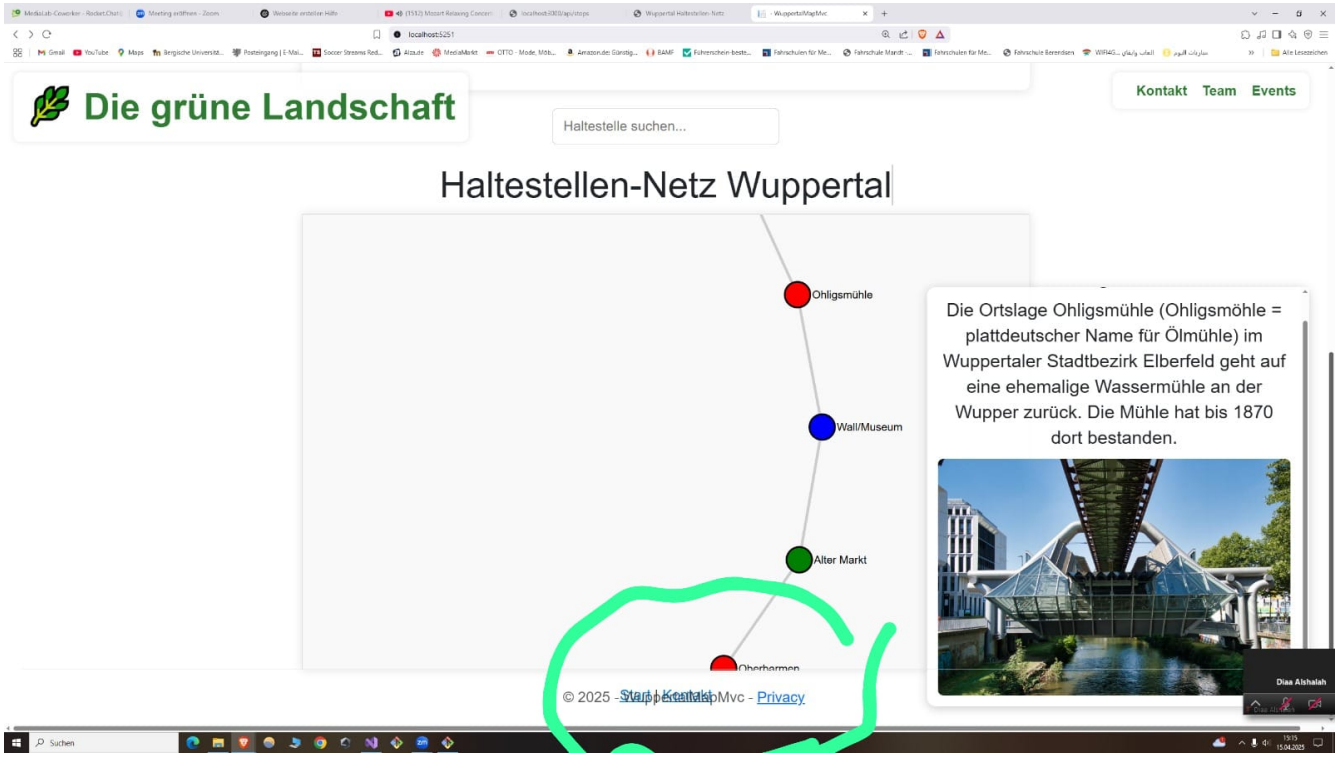

Abbildung 5: D3.js Visualisierung einer Haltestelle mit Beschreibung und Bild

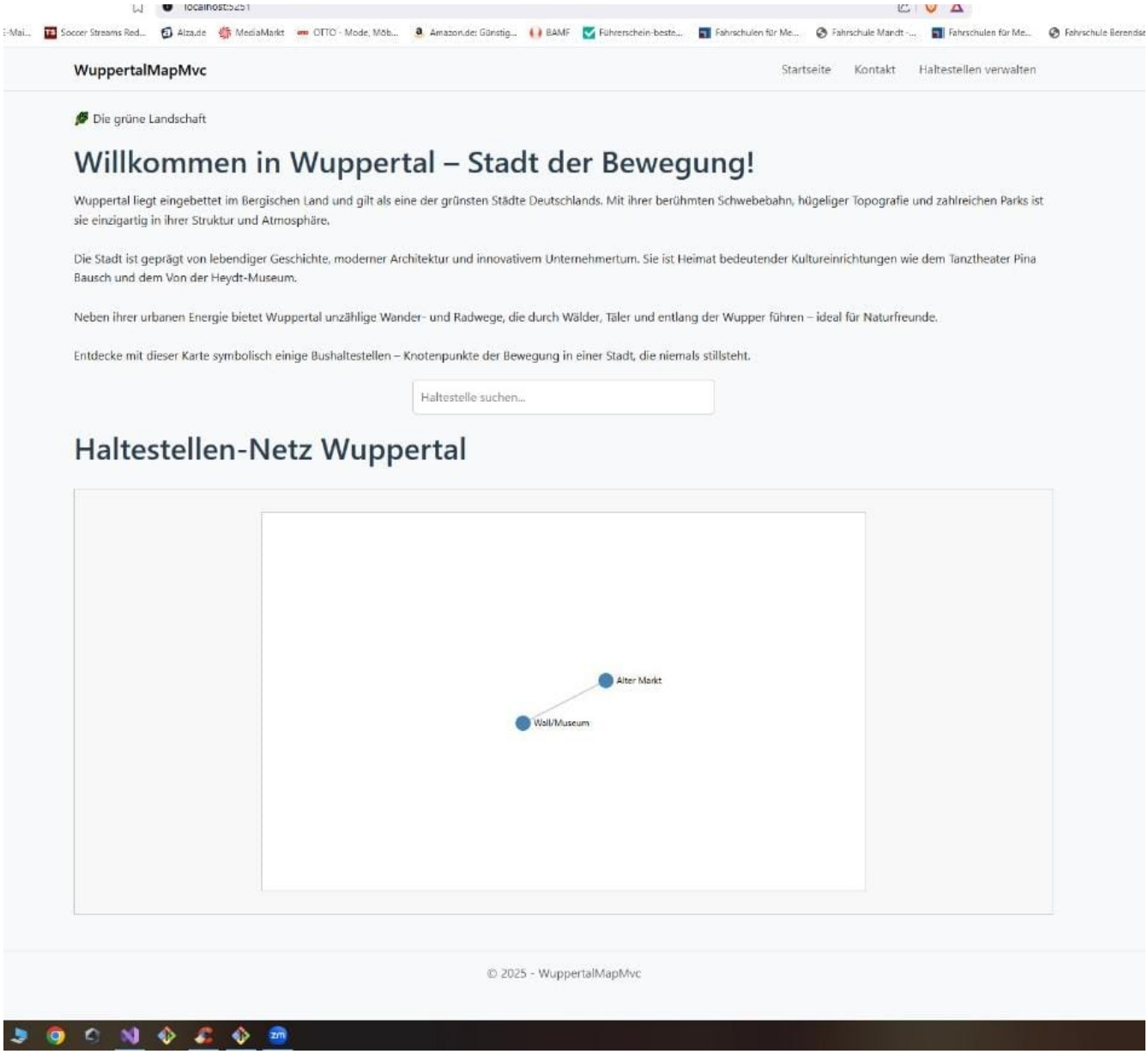

Abbildung 6: Früher Entwicklungsstand der interaktiven Karte mit zwei Knoten

6 Woche 4 (22.04 – 25.04.2025)

22.04.2025

Am 22. April stellte ich fest, dass beim Hinzufügen eines neuen Knotens alle bestehenden verschwanden. Der Grund lag in falschen Koordinaten: Ich hatte Kommas statt Punkte bei Latitude und Longitude verwendet.
Um Verbindungen zwischen Knoten (Bushaltestellen) möglich zu machen, bereitete ich neue Datenbankstrukturen vor:

- Modell `StopConnection`.
- Ergänzung in `AppDbContext`:

```
public DbSet<StopConnection> StopConnections { get; set; }
```

- Migration:

```
dotnet ef migrations add AddStopConnections
dotnet ef database update
```

Ich erweiterte den Seed in `DbInitializer.cs`:

```
var hbf = context.Stops.FirstOrDefault(s => s.Name == "Hauptbahnhof");
var zoo = context.Stops.FirstOrDefault(s => s.Name == "Zoo");

if (hbf != null && zoo != null)
{
    var connections = new List<StopConnection>
    {
        new StopConnection { FromStopId = hbf.Id, ToStopId = zoo.Id }
    };
    context.StopConnections.AddRange(connections);
    context.SaveChanges();
}
```

Im JS befüllte ich das Dropdown für Verbindungen dynamisch:

```
// Dropdown automatisch f llen
const connectSelect = document.getElementById("connect-to");
nodes.forEach(n => {
    const opt = document.createElement("option");
    opt.value = n.id;
```

```
    opt.textContent = n.id;
    connectSelect.appendChild(opt);
});
```

Formularverarbeitung:

```
document.getElementById("add-form").addEventListener("submit",
    async function (e) {
    e.preventDefault();
    // neuer Stop wird erstellt...
    if (res.ok) location.reload();
});
```

23.04.2025

An diesem Tag traf ich Frau Dr. Seehagen-Marx und andere Praktikanten. Ich optimierte die Hover-Glitches: Ich gruppierte Circle + Text in einem gemeinsamen ‘<g>‘-Element:

```
const nodeGroup = svg.append("g")
    .selectAll("g")
    .data(nodes)
    .enter()
    .append("g")
    .call(d3.drag()
        .on("start", dragstarted)
        .on("drag", dragged)
        .on("end", dragended));

nodeGroup.append("circle") ...
nodeGroup.append("text") ...
```

So funktionierte Dragging stabiler.

24.–25.04.2025

Ich programmierte den `StopConnectionController.cs`:

```
[HttpPost]
public async Task<IActionResult> CreateConnection([FromBody]
    StopConnection connection)
{
    if (!ModelState.IsValid)
        return BadRequest(ModelState);

```

```
    _context.StopConnections.Add(connection);
    await _context.SaveChangesAsync();

    return Ok(connection);
}
```

Ich ergänzte auch GET für Verbindungen:

```
[HttpGet]
public IActionResult GetConnections()
{
    var connections = _context.StopConnections
        .Select(sc => new {
            From = sc.FromStop.Name,
            To = sc.ToStop.Name
        })
        .ToList();

    return Ok(connections);
}
```

Im JS sorgte ich dafür, dass Verbindungen direkt in der Simulation sichtbar werden:

```
// Verbindung visualisieren
svg.append("line")
    .datum(links[links.length - 1])
    .attr("stroke", "#aaa")
    .attr("stroke-width", 2);

simulation.force("link").links(links);
simulation.alpha(1).restart();
```

Ich stellte fest:

- Neue Knoten hatten manchmal keine Verbindung.
- Nach dem Reload verschwanden sie.

Die Ursache lag in der falschen Integration in die `links`-Daten.

—

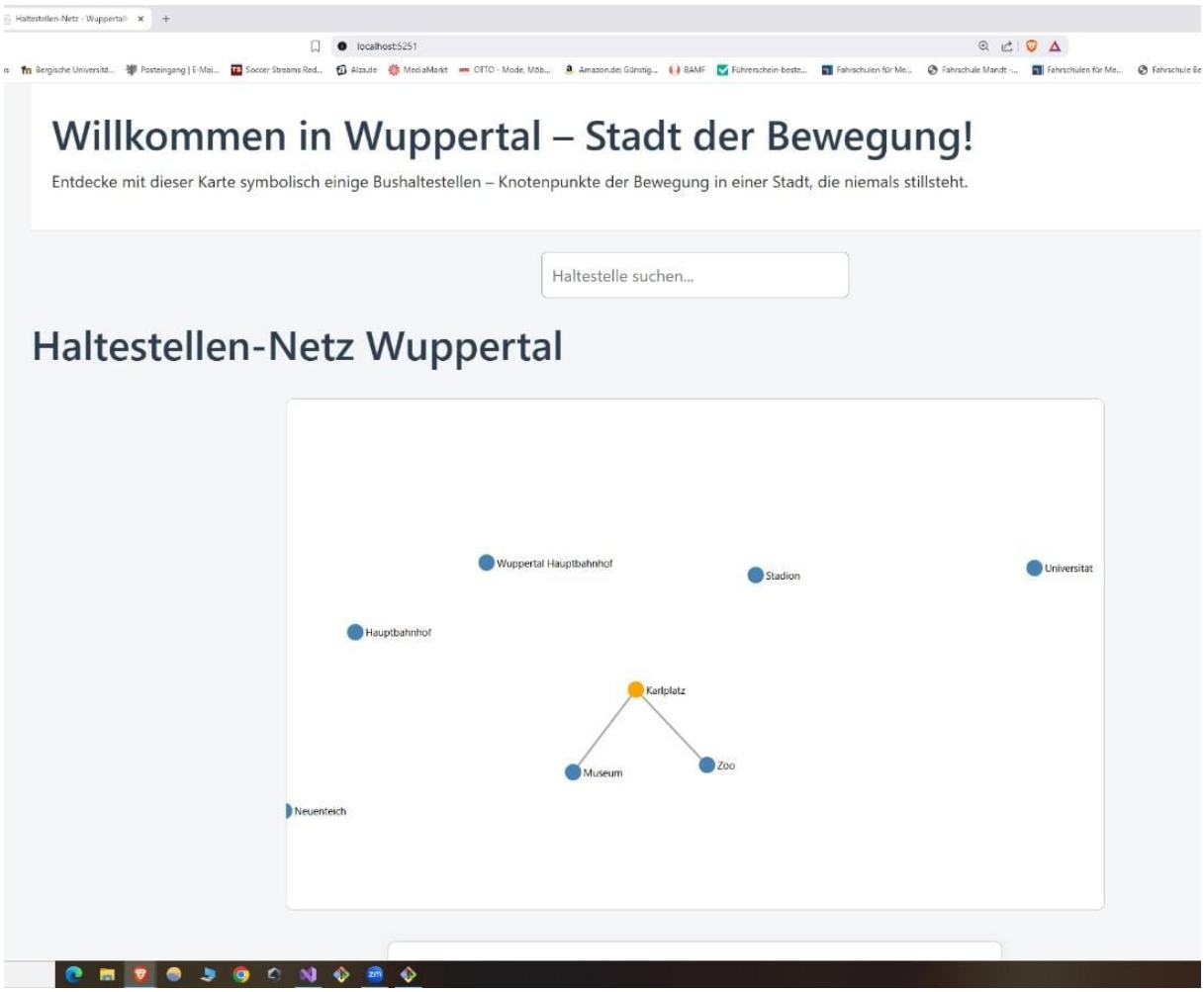

Abbildung 7: Visualisierung des Haltestellen-Netzes in Wuppertal mit farbigen Knoten und verbundenen Linien (Projektstand: erweitert)

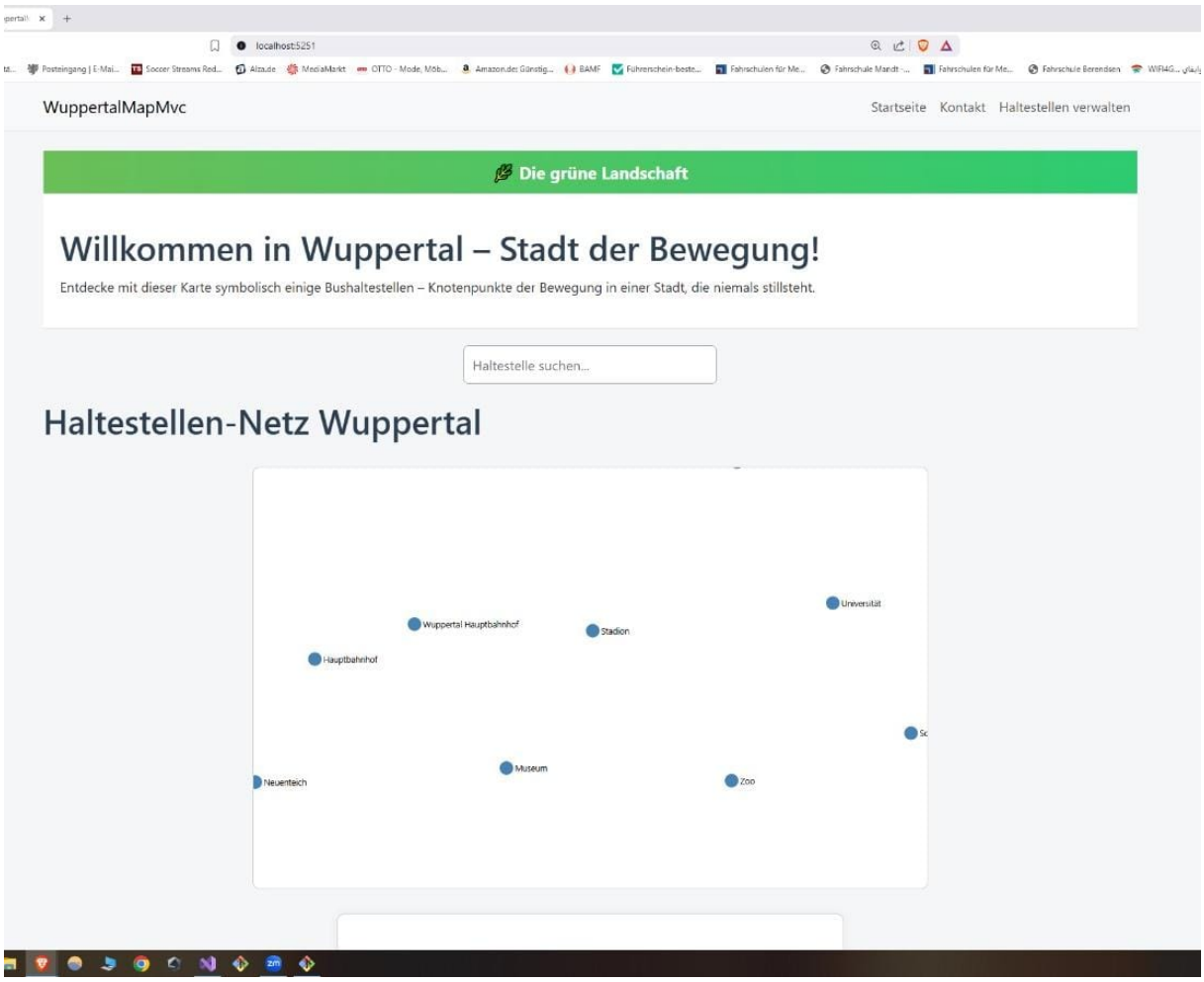

Abbildung 8: Finale Kartenstruktur nach dem Verbinden von Haltestellen im MVC-Projekt

7 Woche 5 (28.04 – 02.05.2025)

28.04.2025

Ich traf mich mit Frau Dr. Seehagen-Marx.
Ich stellte fest: In meiner `map.js` hatte ich ‘<script>‘-Tags direkt im JS – unzulässig. Ich räumte auf:

```
<script src="https://d3js.org/d3.v7.min.js"></script>
<script type="module" src="~/js/map.js"></script>
```

Nun wurden Name, Links und Verbindungen korrekt geladen.

29.–30.04.2025

Ich sah Lernvideos zu ASP.NET:
`https://www.youtube.com/watch?v=qBTe6uHJS_Y&list=PL82C6-O4XrHfrGOCPmKmwTO7MOavXyQKc`
Ich behob:

- Hover-Glitch mit ‘<g>‘-Gruppe.
- Knoten wie „Karlplatz“ nur als Label – jetzt mit Circle.

02.05.2025

Ich erstellte eine neue `map.js`:

```
const svg = d3.select("#graph").append("svg")
    .attr("width", 800).attr("height", 500);

const x = d3.scaleLinear().domain([7.12, 7.22]).range([0, 800]);
const y = d3.scaleLinear().domain([51.23, 51.28]).range([500, 0]);

async function loadStops() {
    const response = await fetch("/api/stopapi");
    const data = await response.json();

    const nodes = data.stops.map(stop => ({
        id: stop.name,
        x: x(stop.longitude),
        y: y(stop.latitude)
    }));

    svg.selectAll("circle")
        .data(nodes)
```

```
        .enter()
        .append("circle")
        .attr("cx", d => d.x)
        .attr("cy", d => d.y)
        .attr("r", 10);
}
loadStops();
```

Die neue Karte funktionierte, aber das Verbinden blieb schwierig.

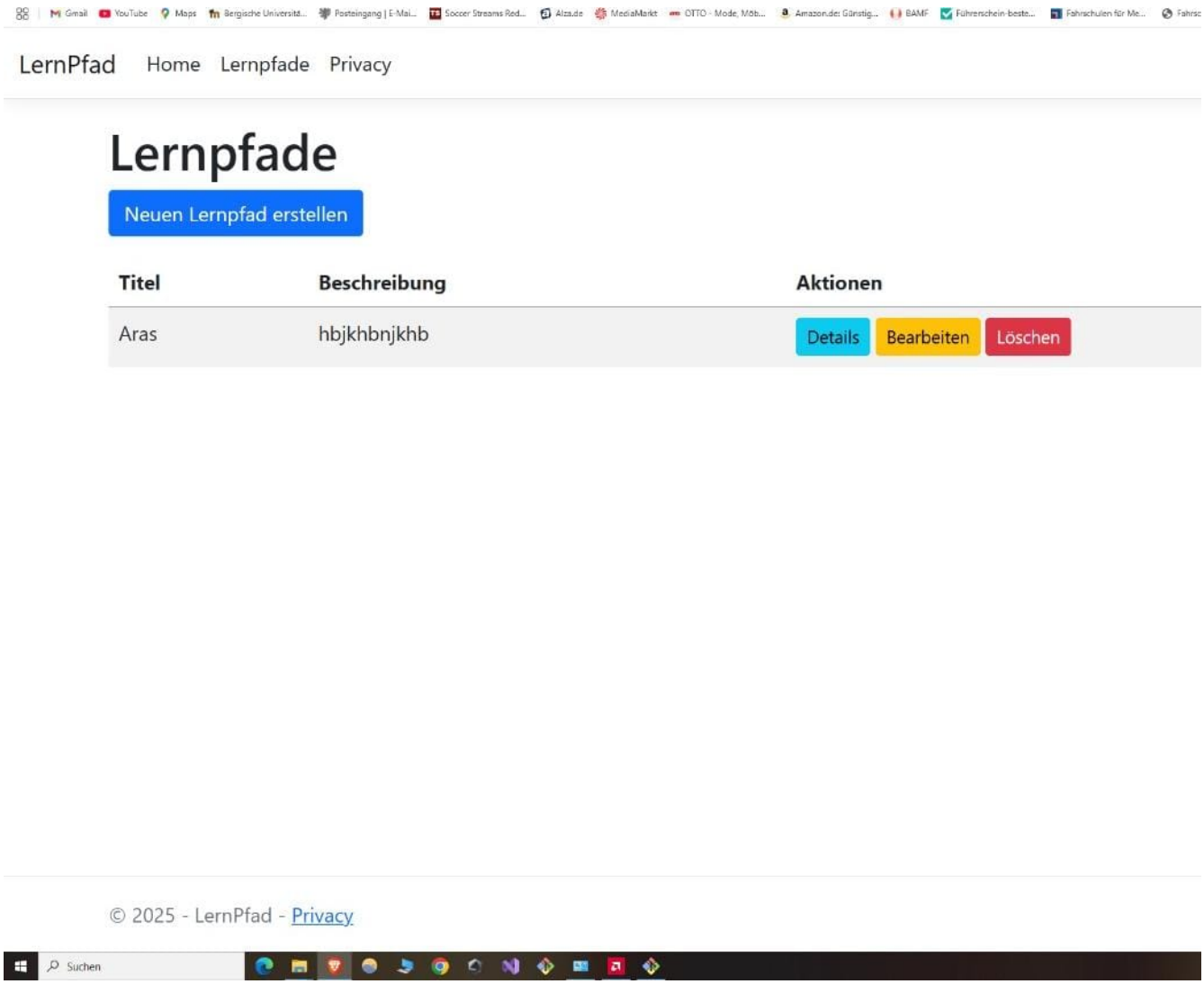

Abbildung 9: Lernpfadübersicht mit Aktionen wie Details, Bearbeiten und Löschen

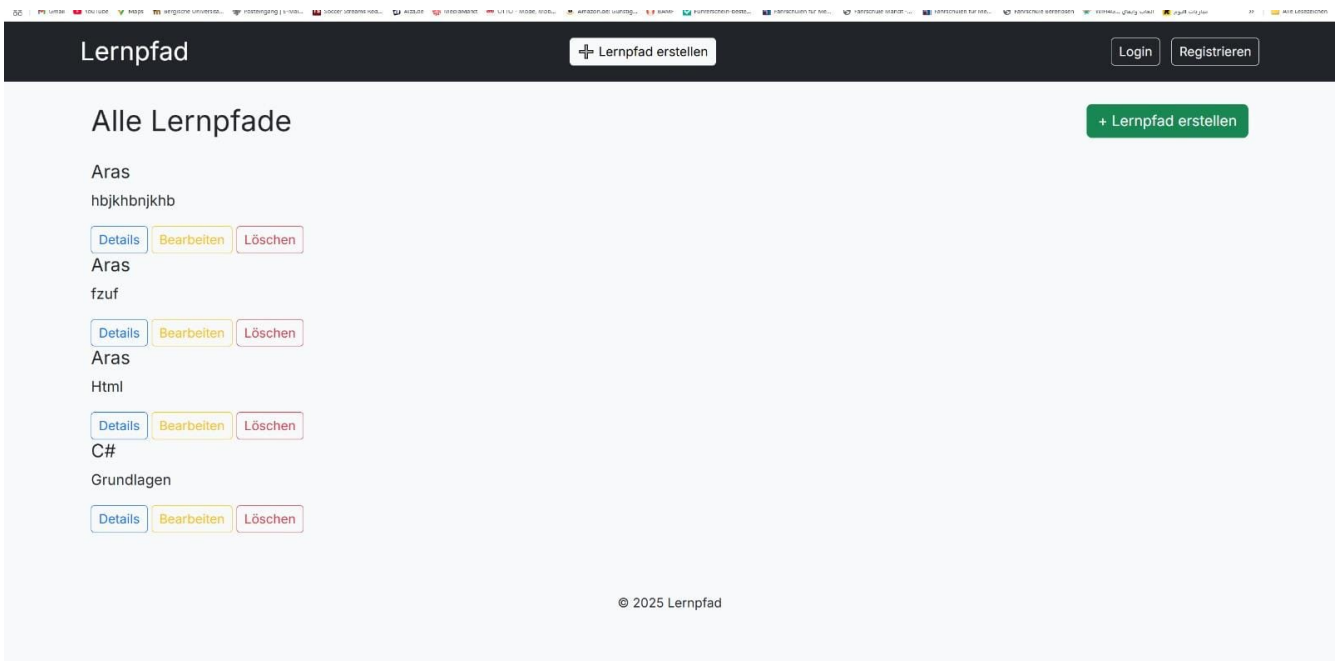

Abbildung 10: Seite mit allen erstellten Lernpfaden – Ansicht als Liste

8 Woche 6 (05.05 – 09.05.2025)

05.05.2025

Zu Beginn der sechsten Woche besprach ich mit Frau Dr. Seehagen-Marx, dass ich nun an meiner zweiten Aufgabe arbeiten würde. Diese bezieht sich auf das Konzept der Visualisierung in der Webseitenentwicklung – ähnlich wie bei der interaktiven Karte, nur diesmal mit dem Schwerpunkt **Lernpfad**.
Ich untersuchte die Plattform `https://lernpfad.ch/pfad/htrjdrbjz5b3/vorschau`. Ich stellte fest: Ein Lernpfad ist eine strukturierte Abfolge von Schritten, die Lernmaterialien wie Texte, Videos oder Aufgaben enthalten. Lehrkräfte können Lernpfade erstellen und teilen.
Ich startete die neue ASP.NET-Core-Anwendung und legte folgende Projektstruktur an:

- Controller: `AccountController.cs`, `LernpfadController.cs`, `InhaltController.cs`.
- Models: `ApplicationUser.cs`, `Lernpfad.cs`, `Lernschritt.cs`, `Inhalt.cs`.
- Views: `Lernpfad/Index.cshtml`, `Details.cshtml`, `Create.cshtml`.
- `ApplicationDbContext.cs` für EF Core.

08.05.2025

Ich installierte NuGet-Pakete und das Design-Paket für Migrationen. In `Index.cshtml` setzte ich eine erste View auf:

```
@model IEnumerable<LernPfad.Models.Lernpfad>

<h1>Lernpfade</h1>
<p><a asp-action="Create" class="btn␣btn-primary">Neuen Lernpfad
   erstellen</a></p>
<table class="table">
  <thead>
    <tr><th>Titel</th><th>Beschreibung</th><th>Aktionen</th></tr>
  </thead>
  <tbody>
  @foreach (var item in Model) {
    <tr>
      <td>@item.Titel</td>
      <td>@item.Beschreibung</td>
      <td>
        <a asp-action="Details" asp-route-id="@item.Id">Details</a>
        <a asp-action="Edit" asp-route-id="@item.Id">Bearbeiten</a>
```

```
17          <a asp-action="Delete" asp-route-id="@item.Id">L schen</a>
18        </td>
19      </tr>
20    }
21    </tbody>
22  </table>
```

Die CRUD-Funktionalität funktionierte, nur das Design war noch minimal.

09.05.2025

Ich erstellte `Create.cshtml` mit Formular für Titel und Beschreibung. Im Controller baute ich zwei Methoden:

```
1  public IActionResult Create() => View();
2
3  [HttpPost]
4  [ValidateAntiForgeryToken]
5  public async Task<IActionResult> Create(Lernpfad.Models.Lernpfad
       lernpfad)
6  {
7    if (ModelState.IsValid) {
8      _context.Lernpfade.Add(lernpfad);
9      await _context.SaveChangesAsync();
10     return RedirectToAction(nameof(Index));
11   }
12   return View(lernpfad);
13 }
```

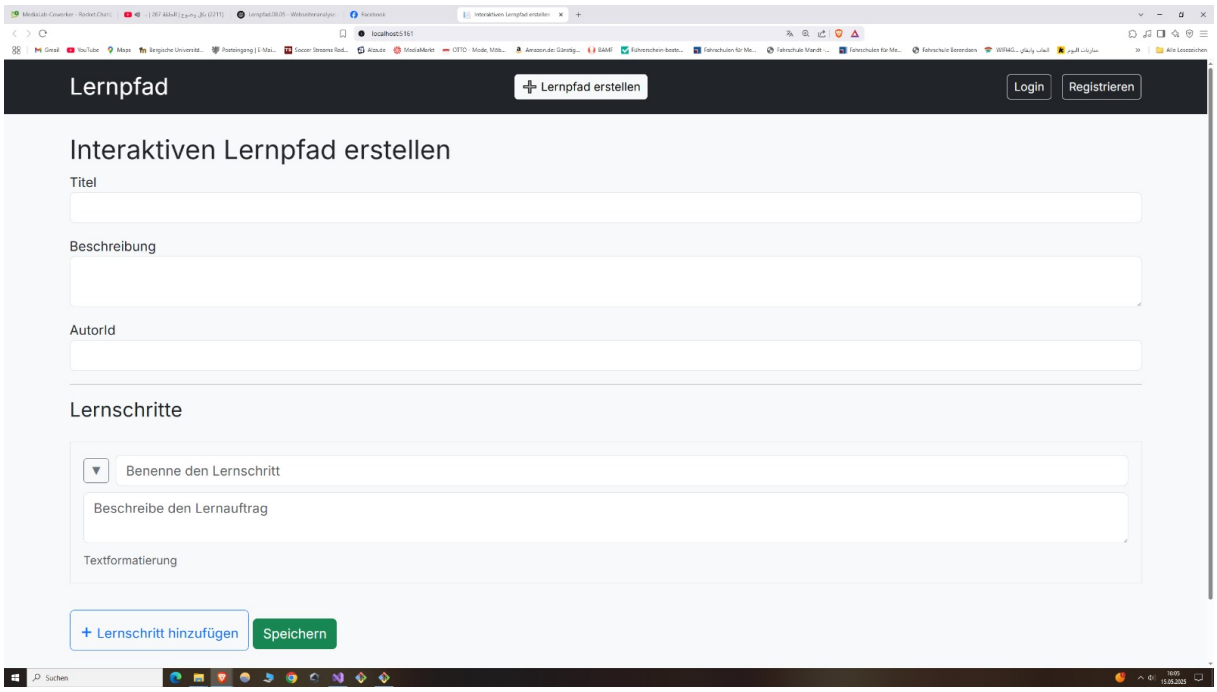

Abbildung 11: Übersicht aller gespeicherten Lernpfade mit Titel, Beschreibung und Aktionsbuttons

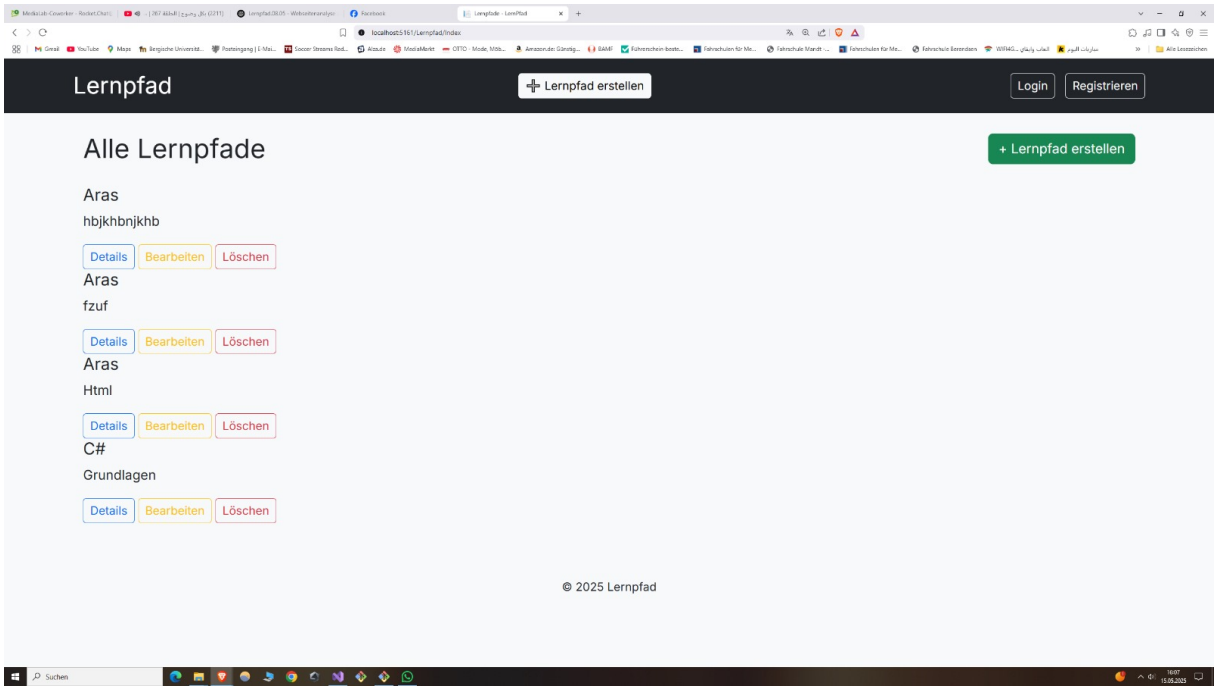

Abbildung 12: Formular zum Erstellen eines interaktiven Lernpfads mit dynamisch hinzufügbaren Lernschritten

9 Woche 7 (12.05 – 16.05.2025)

12.05.2025

Ich sprach mit Frau Dr. Seehagen-Marx. Ich programmierte `Edit.cshtml` mit GET/-POST:

```
// GET
public async Task<IActionResult> Edit(int id) {
  var lernpfad = await _context.Lernpfade.FindAsync(id);
  if (lernpfad == null) return NotFound();
  return View(lernpfad);
}

// POST
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, Lernpfad lernpfad) {
  if (id != lernpfad.Id) return NotFound();
  if (ModelState.IsValid) {
    try {
      _context.Update(lernpfad);
      await _context.SaveChangesAsync();
    } catch (DbUpdateConcurrencyException) {
      if (!_context.Lernpfade.Any(e => e.Id == lernpfad.Id)) return
          NotFound();
      else throw;
    }
    return RedirectToAction(nameof(Index));
  }
  return View(lernpfad);
}
```

Ich behebe Fehler im Hidden-Input:

```
<input type="hidden" asp-for="Id" />
```

13.–15.05.2025

Ich baute `Details.cshtml` und `Delete.cshtml`. Ich erweiterte das Layout mit Bootstrap:

```
<link rel="stylesheet"
 href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/
    bootstrap.min.css" />
```

Header, Footer, moderne Schrift `Inter`. Cards für Lernpfade: Titel, Beschreibung, Buttons.

16.05.2025

Ich schrieb `Preview.cshtml`. Im Controller:

```
public async Task<IActionResult> Preview(int id = 1)
{
  var lernpfad = await _context.Lernpfade
    .Include(lp => lp.Schritte)
    .FirstOrDefaultAsync(lp => lp.Id == id);
  if (lernpfad == null) return NotFound();
  return View(lernpfad);
}
```

Ich passte `Lernpfad.cs` und `Lernschritt.cs` an, um die Beziehungen abzubilden.

10 Woche 8 (19.05 – 22.05.2025)

19.05.2025

Ich erweiterte `Create.cshtml` mit CKEditor 5:

```
<script src="https://cdn.ckeditor.com/ckeditor5/41.4.2/classic/
   ckeditor.js"></script>
```

Dynamische Lernschritte:

```
ClassicEditor.create(document.querySelector('textarea[name="
   Beschreibung"]'))
```

21.–22.05.2025

Ich programmierte Upload:

```
public string? BildPfad { get; set; }
```

Formular:

```
<input type="file" name="BildDatei" class="form-control" accept="
   image/*" />
```

Upload-Logik:

```
if (BildDatei != null) {
  var uploadsFolder = Path.Combine(_env.WebRootPath, "uploads");
  if (!Directory.Exists(uploadsFolder)) Directory.CreateDirectory(
     uploadsFolder);
  var uniqueName = Guid.NewGuid() + Path.GetExtension(BildDatei.
     FileName);
  var filePath = Path.Combine(uploadsFolder, uniqueName);
  using var stream = new FileStream(filePath, FileMode.Create);
  await BildDatei.CopyToAsync(stream);
  lernpfad.BildPfad = "/uploads/" + uniqueName;
}
```

Die Suchfunktion:

```
query = query.Where(lp => lp.Titel.Contains(suchtext));
```

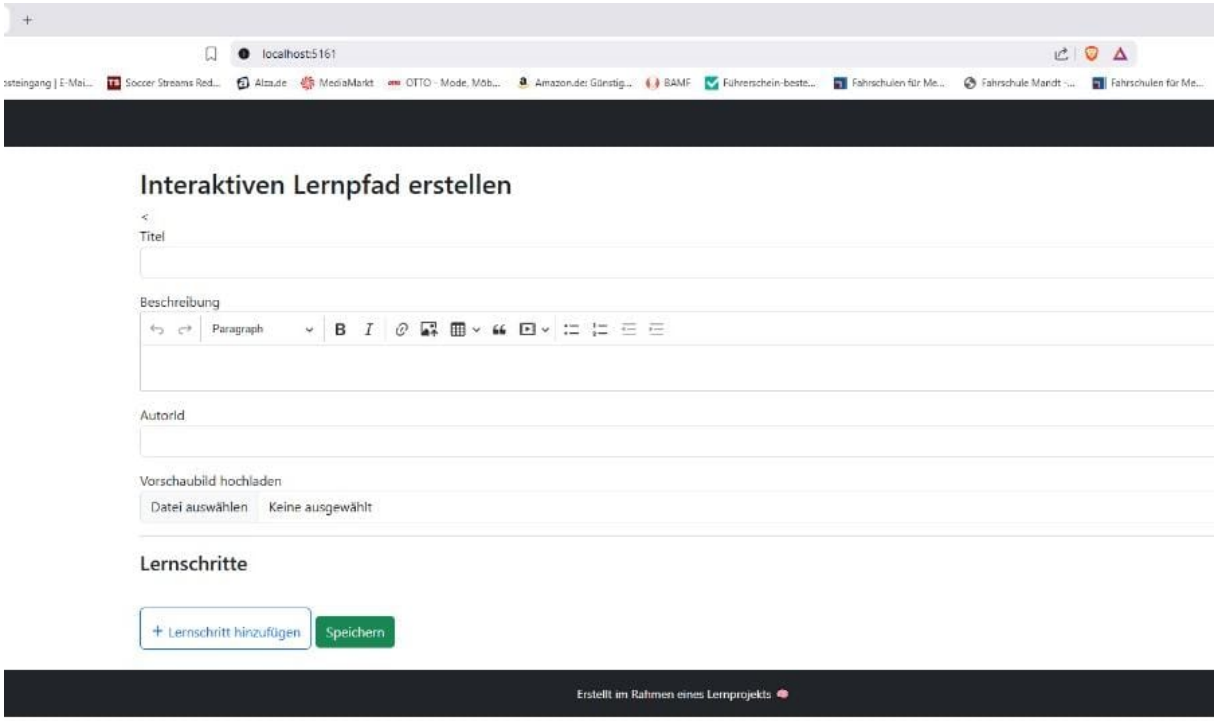

Abbildung 13: Formular zum Erstellen eines interaktiven Lernpfads mit Titel, Beschreibung und CKEditor

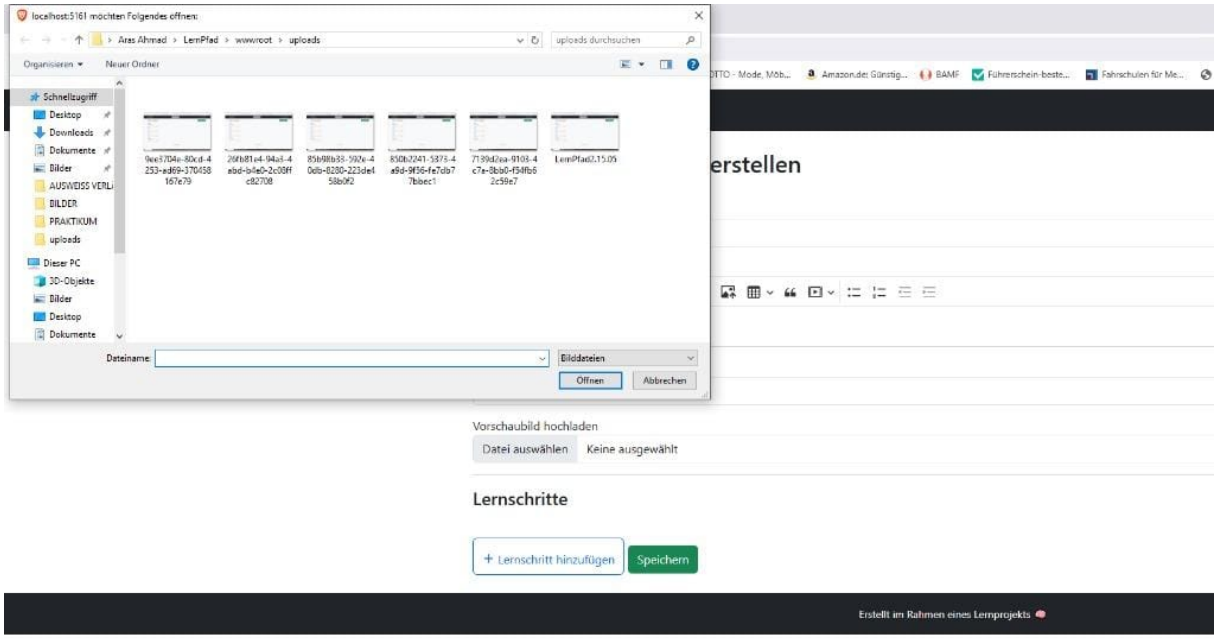

Abbildung 14: Erweiterte Ansicht des CKEditor-Formulars zur Erstellung von Lernpfaden

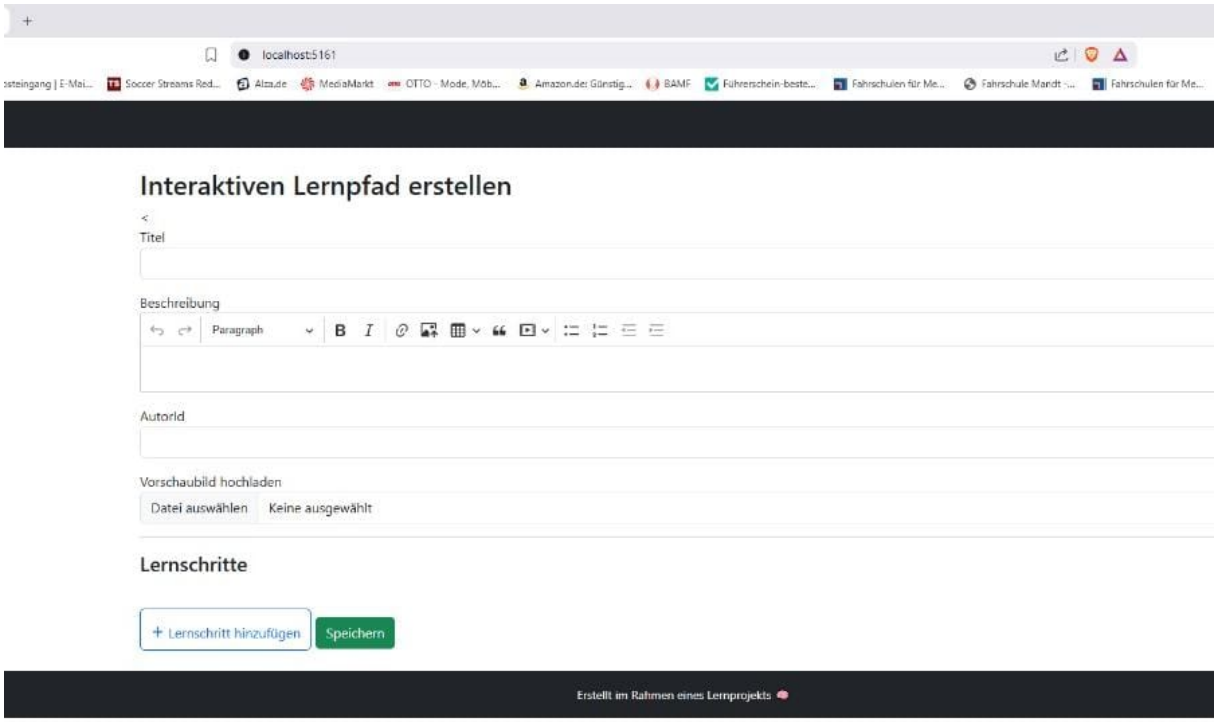

Abbildung 15: Upload-Dialog zum Hochladen eines Vorschaubilds für den Lernpfad

Wuppertal, 20. Oktober 2025 Aras Ahmad

Fazit

Rückblickend habe ich in diesen acht Wochen ASP.NET Core MVC, EF Core, D3.js, CKEditor und moderne Frontend-Technologien praktisch erlernt. Ich konnte Fehler selbstständig lösen, Features erweitern und Design mit Funktionalität verbinden. Ich danke Frau Dr. Seehagen-Marx für die Betreuung und freue mich, dieses Wissen in künftigen Projekten anzuwenden.

Wuppertal, 20. Oktober 2025 Aras Ahmad

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich den vorliegenden Bericht eigenständig und ohne unzulässige Hilfe Dritter angefertigt habe. Alle aus anderen Quellen direkt oder indirekt übernommenen Inhalte sind als solche kenntlich gemacht. Der Bericht wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Wuppertal, 20. Oktober 2025

Unterschrift des Studierenden