## Vergleichstabelle

|  |  |  |  |
| --- | --- | --- | --- |
| Kriterium | ASP.NET Core MVC | Blazor (Server/WebAssembly) | React (mit ASP.NET Backend) |
| Entwicklungsaufwand | Gering bis moderat – gute Integration mit EF | Mittel – komplexer bei Komponenten | Hoch – separate Projekte, API-Kommunikation nötig |
| Lernkurve | Mittel – etwas neuere Technologie | Mittel – neue Denkweise mit Komponenten | Flach – gewohnte Technologie |
| Wartbarkeit | Hoch – serverbasiert, klare Trennung von Logik & UI | Mittel – bei vielen Komponenten steigt Komplexität | Mittel – bei guter Struktur sehr wartbar |
| Performance | Gut – server-rendered, zuverlässig | Gut (Server) / schwankend (WebAssembly, je nach App-Größe) | Sehr gut – clientseitig, schnell und reaktiv |
| Mobile Tauglichkeit | Gut | Gut | Sehr gut – viele UI-Frameworks (z.B. MUI, Tailwind) |
| Tooling & Debugging | Sehr gut – Visual Studio, Razor Support | Gut – speziell im .NET-Ökosystem | Gut – Chrome DevTools, VS Code etc. |

Auf Basis des Projektumfangs wurde die Methode ASP.NET Core MVC als passend empfunden. Dadurch kann eine vollständige Entwicklung mit .NET gewährleistet werden. Ausserdem bietet die .NET-Umgebung eine sehr gute Tool-Unterstützung und es existiert so keine Overhead mit JavaScript-Frameworks.

## Begründung der Entscheide

Das Projekt wurde in Folgende Ordner untergliedert, um eine saubere MVC-Struktur zu gewährleisten. Für die Entwicklung wurde zudem ein MacBook Pro verwendet. Da auf MacOS Visual Studio nicht mehr unterstützt wird, wurde auf den Editor Visual Studio Code zurückgegriffen, doch dazu mehr in der README-Dokumentation.

1. Controller: Die beiden Controller steuern lediglich den Datenfluss und erledigen die CRUD-Tasks des Projektes
2. Models: Hier werden die Datestrukturen sowie die Validierung mit den Tags angelegt
3. ViewModels: Hier wurde auf ein ViewModel für *SpesenEintragCreateViewModel.cs* gesetzt, da es eine bessere Übersicht von benötigten Daten bietet.
4. Wwwroot/spesen-form.js: In dieser findet lediglich die Live-Berechnung der Reisekosten statt und hat für das Projekt sonst keinen Einfluss
5. Data: Mit dem AppDbContext wird die Datenbankintegration gewährleistet mit EF Core. Die Factory Datei wurde zu Beginn gebraucht, damit EF Core einen Weg hatte den Kontext manuell zu bauen. Damit wurde eine Fehlermeldung behoben.