



Partitio 2.0

Jetzt auch als Core Razor Anwendung

Florian Buchholz
Jörg J. Buchholz

16. März 2026

1 Inhaltsverzeichnis

1	Inhaltsverzeichnis	I
2	Abbildungsverzeichnis	III
3	Bedienungsanleitung	1
3.1	Einführung	1
3.1.1	Warum sollten wir Partitio benutzen?	1
3.1.1.1	Scheidung, Trennung	1
3.1.1.2	Erben	1
3.1.2	Wie löst Partitio das Problem?	1
3.2	Geht los ...!	2
3.2.1	Anmelden	2
3.2.2	Lesezeichen erstellen	3
3.2.3	Objekte fotografieren	5
3.2.4	Objektliste	5
3.2.5	Detailfenster	6
3.2.6	Löschen	7
3.2.7	Hamburger-Menü	8
3.2.7.1	Neue Teilnehmerinnen	9
3.2.7.2	Anonymität	10
3.2.7.3	Alle zufrieden?	11
3.2.7.3.1	Drucken	11
3.2.7.4	Hilfe	12
3.2.8	Sortieren	12
3.2.9	Über- und Unterbieten	14
3.2.10	KI	17
4	Unter der Haube	20
4.1	Zusammenspiel der Programmteile	20
4.1.1	Grundstruktur der Anwendung	20
4.1.2	Datenmodelle als gemeinsames Fundament	20
4.1.3	Datenzugriff und Persistenz	20
4.1.4	Fachlogik und Service-Schicht	21
4.1.5	Verdrahtung zur Laufzeit	21
4.1.6	Gemeinsame Razor-Grundlagen	21
4.1.7	Hauptseite: PageModel und Ansicht	21
4.1.8	Fehlerbehandlung als eigener Ablauf	22
4.1.9	Zusammenfassung des Zusammenspiels	22
4.2	PartitioModels.cs	22
4.3	IProjektRepository.cs	24
4.4	XmlProjektRepository.cs	25
4.5	IPartitioService.cs	28
4.6	PartitioService.cs	30
4.7	Program.cs	38
4.8	_ViewImports.cshtml	41
4.9	_ViewStart.cshtml	41
4.10	Partitio.cshtml.cs	41
4.11	Partitio.cshtml	49
4.12	Error.cshtml.cs	58
4.13	Error.cshtml	59

5	Literatur	61
---	-----------------	----

2 Abbildungsverzeichnis

Abbildung 1	Anmeldeseite	3
Abbildung 2	Lesezeichen erstellen	4
Abbildung 3	Bild eines neuen Objekts aufnehmen	5
Abbildung 4	Objektliste	6
Abbildung 5	Detailfenster	7
Abbildung 6	Löschbestätigung	8
Abbildung 7	Hamburger Menü	9
Abbildung 8	Neue Teilnehmerinnen	10
Abbildung 9	Alle zufrieden?	11
Abbildung 10	Sortierreihenfolge: Besitzer, Objektname	12
Abbildung 11	Sortiert nach Objektname	13
Abbildung 12	Sortiert nach Gebot	14
Abbildung 13	Übernehmen durch Unterbieten klappt nicht.	15
Abbildung 14	Nicht genug Guthaben	16
Abbildung 15	Gitarre übernommen	17

3 Bedienungsanleitung

Diese Bedienungsanleitung haben wir aus der ersten Partitio-Dokumentation übernommen [1], da sich aus Sicht der Nutzerin eigentlich nichts wirklich geändert hat; bis auf die Tatsache, dass das Programm jetzt auf einem anderen Server liegt. Es ist aber immer noch erreichbar unter partitio.de.

3.1 Einführung

Wörtlich aus dem Lateinischen übersetzt heißt Partitio: „Teilung“, „Aufteilung“ oder „Verteilung“; und genau darum geht es: Partitio ist ein Programm, das Menschen beim Aufteilen gemeinsamer Objekte helfen kann; beispielsweise bei einer Scheidung oder beim Erben.

3.1.1 Warum sollten wir Partitio benutzen?

Partitio beruht auf dem Prinzip des optimalen Kompromisses, das heißt, dass die Teilnehmerinnen¹ so lange miteinander verhandeln, bis jede Einzelne mit dem Ergebnis zufrieden ist. Da die Verhandlungen aber nicht direkt, sondern über ein unparteiisches Programm geführt werden, bestimmen nicht persönliche Gefühle, sondern die wirklichen Wünsche und Ziele der einzelnen Teilnehmerinnen den ausgehandelten Kompromiss.

3.1.1.1 Scheidung, Trennung

„In dem Sessel hast du doch nie gegessen!“

Wenn sich zwei Partnerinnen trennen, stehen zwischen beiden häufig so viele negative Gefühle, dass es ihnen nicht möglich erscheint, im direkten Gespräch miteinander den gemeinsamen Besitz gerecht aufzuteilen. Jede Partnerin hat Angst, von der anderen „über den Tisch gezogen“ zu werden und gerät dabei selbst leicht in die Versuchung, die andere „auszutricksen“.

3.1.1.2 Erben

„Eigentlich brauche ich ja gar nichts aus dem Erbe meiner Mutter, aber wenn ich jetzt sehe, wie sich meine Schwestern um jeden Eierbecher streiten, ...“

Geschwister haben in ihrer Kindheit um alles konkurriert, was ihre Eltern ihnen an materiellen und ideellen Zuwendungen geben konnten. Häufig führt diese verinnerlichte Konkurrenzsituation dann auch noch im Erbfall dazu, dass die Einzelne das Gefühl hat, ihr „Anrecht“ gegen ihre Geschwister durchsetzen zu müssen, obwohl ihr die meisten zu erbenden Objekte eigentlich relativ egal sind.

3.1.2 Wie löst Partitio das Problem?

Das Ziel ist klar:

Jede Teilnehmerin bekommt am Ende genau das, was ihr am wichtigsten ist.

Dazu erhält jede Teilnehmerin insgesamt 1000 Taler (Punkte, virtuelles Spielgeld, ...) und setzt davon – wie bei einer Auktion – beliebige Summen auf die Objekte, die sie gerne bekommen möchte. Im theoretischen Idealfall gibt es am Ende keine Überschneidungen; jede Teilnehmerin bekommt das, was sie haben möchte, und alle sind zufrieden. In der Praxis funktioniert das natürlich nicht; schließlich möchte eigentlich jeder gerne die Briefmarkensammlung von Opa haben, niemand interessiert sich für den alten Fernseher und „bevor mein Bruder den ganzen Schmuck bekommt, ...“.

¹Dieses Dokument verwendet das generische Femininum. Männer sind automatisch mitgemeint.

Dann wird es spannend. Angenommen, Kainina hat 600 Taler für die Briefmarkensammlung geboten; dann hat sie natürlich nur noch 400 Taler für die restlichen Objekte zur Verfügung. Abella hat jetzt zwei Alternativen: Entweder sie überbietet Kainina (z.B. mit 700 Talern); dann hat Kainina wieder 1000 und Abella nur noch 300 Taler für die übrigen Objekte. Oder aber Abella sagt sich, dass ihr die Briefmarkensammlung nicht mehr als 600 Taler wert ist, überlässt Kainina die Briefmarken und kann nun ihrerseits 1000 Taler für andere Objekte setzen.

Interessant ist nun, dass es – anders als bei einer klassischen Auktion – bei Partitio auch jederzeit möglich ist, Gebote wieder zu erniedrigen, um auf diese Weise wieder mehr Taler auf vielleicht doch ganz interessante andere Objekte setzen zu können. Natürlich geht frau durch eine Erniedrigung immer das Risiko ein, dass eine andere Teilnehmerin das dann günstiger gewordene Angebot übernimmt; aber schließlich kann frau sich ja jederzeit entscheiden, doch wieder mehr zu setzen.

Durch die Möglichkeit, Gebote beliebig oft zu erhöhen oder zu erniedrigen, wechseln einzelne Objekte vielleicht mehrere dutzendmal die Besitzerin und ihr „Preis“ steigt und sinkt; am Ende des Projekts bekommt sie aber doch die Teilnehmerin, der sie insgesamt am meisten wert sind.

Das Projekt ist daher genau dann im Gleichgewichtszustand und damit zu Ende, wenn alle Teilnehmerinnen zufrieden sind. Oder anders gesehen: Solange noch eine Teilnehmerin unzufrieden mit dem Ergebnis ist, wird sie es einfach ihren Vorstellungen gemäß anpassen. Darauf reagiert dann möglicherweise eine andere Teilnehmerin wieder, so dass frau selber auch wieder ein paar Taler verschieben möchte, bis ... ja ... bis eben alle zufrieden sind.

Klingt kompliziert? Probier's einfach mal aus: partitio.de

3.2 Geht los ...!

3.2.1 Anmelden

Durch den Aufruf von partitio.de startet Erika automatisch ein neues Projekt und landet auf der Anmeldeseite (Abbildung 1)

The image shows a web application interface. At the top, there is a teal header bar with a hamburger menu icon and the text "Partitio -". Below this, a white modal dialog box is displayed with the title "Neue Teilnehmerin". Inside the dialog, there is a form with a label "Name" and a text input field containing the name "Erika". Below the input field is a button labeled "OK". To the right of the dialog box, there is a green circular button with a white plus sign. At the bottom of the screen, there is a teal bar containing the text "Guthaben:" on the left and "Aktualisieren" on the right.

Abbildung 1: Anmeldeseite

3.2.2 Lesezeichen erstellen

Nachdem Erika ihren Namen eingetragen hat, gelangt sie auf eine Informationsseite (Abbildung 2)

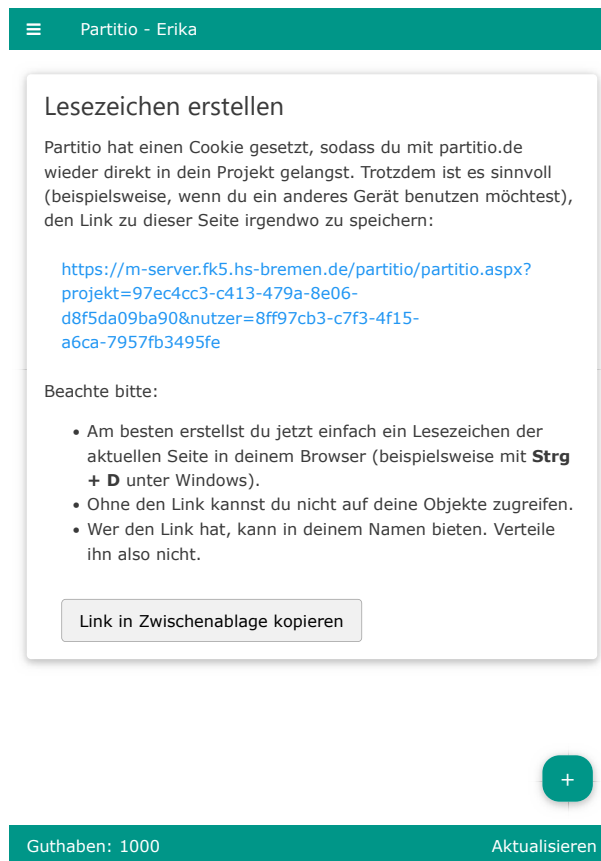


Abbildung 2: Lesezeichen erstellen

auf der ihr Name in der Titelzeile („Partitio - Erika“) eingetragen ist und auf der sie darüber informiert wird, dass Partitio einen Cookie im aktuellen Browser gesetzt hat, sodass sie beim nächsten Aufruf von `partitio.de` automatisch in ihr bestehendes Projekt gelangt. Um sicher zu gehen, kann Erika zusätzlich (beispielsweise unter Windows mit Strg + D) ein Lesezeichen in ihrem Browser setzen.

Wenn sie ihr Projekt auch in einem anderen Browser oder auf einem anderen Gerät öffnen möchte, kann sie den dazu notwendigen Link durch Anklicken des gleichnamigen Buttons in die Zwischenablage kopieren und irgendwie (E-Mail, Whatsapp, ...) auf das andere Gerät übertragen.

Der Link, der in diesem Fall²

```
https://m-server.fk5.hs-bremen.de/partitio/partitio.aspx?
projekt=97ec4cc3-c413-479a-8e06-d8f5da09ba90&
nutzer=8ff97cb3-c7f3-4f15-a6ca-7957fb3495fe
```

lautet, beinhaltet sowohl die Adresse des Programmes

```
https://m-server.fk5.hs-bremen.de/partitio/partitio.aspx
```

als auch eine eindeutige aber nicht erratbare³ Kennung des aktuellen Projekts

```
projekt=97ec4cc3-c413-479a-8e06-d8f5da09ba90
```

und die ebenfalls nicht erratbare Kennung der aktuellen Teilnehmerin:

²Partitio liegt seit 2026 nicht mehr auf dem Server der Hochschule Bremen, sondern auf dem privaten Server des Autors. Alle Links beginnen jetzt daher mit `https://webserver.kee9wtun8rfetirg.myfritz.net/partitio` statt mit `https://m-server.fk5.hs-bremen.de`. Aus „Effizienzgründen“ (alias Faulheit) passen wir die Bilder in dieser Dokumentation nicht auf diese neuen Links an.

³Die Kennungen sind sogenannte Globally Unique Identifier (GUID). Sie bestehen aus 32 zufälligen Zahlen und Buchstaben (a bis f) und lassen sich daher praktisch nicht erraten. Aus dem gleichen Grund geht auch die Wahrscheinlichkeit, dass nochmals die gleiche GUID erzeugt wird, gegen null.

3.2.3 Objekte fotografieren

Durch Anklicken des Plus-Buttons (rechts unten in Abbildung 2) gelangt Erika zu einer Abfrage, in der sie den Zugriff auf die Kamera abnickt⁴.

Als dann zeigt die Seite das Bild der rückwärtigen Kamera des Smartphones oder der Webcam⁵ in einer Auflösung von 300×300 Pixeln (Abbildung 3).

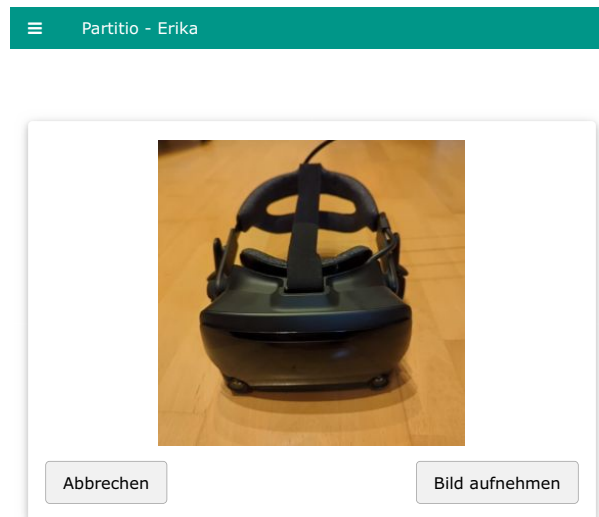


Abbildung 3: Bild eines neuen Objekts aufnehmen

Durch Anklicken des Bild aufnehmen-Buttons wird der aktuelle Kameraausschnitt auf den Server hochgeladen und sofort in der Objektliste angezeigt (Abbildung 4).

3.2.4 Objektliste

Die Objektliste (Abbildung 4) zeigt alle Objekte des Projekts, egal, von wem sie fotografiert wurden.

⁴Wenn Erika den Zugriff auf ihre Kamera verbietet, kann sie keine eigenen Objekte fotografieren und hochladen. Sie kann aber trotzdem auf Objekte bieten, die andere Teilnehmerinnen hochgeladen haben.

⁵Üblicherweise läuft Partitio auf einem Smartphone (oder Tablet), da es damit sehr einfach ist, die zu teilenden Objekte direkt vor Ort zu fotografieren. Bei Bedarf kann Erika Objekte aber auch beispielsweise am PC oder Notebook mit einer Webcam fotografieren. Das Hochladen von bereits gespeicherten Bildern ist aus Sicherheitsgründen nicht möglich.

Partitio - Erika		
Objekt	Gebot	Besitzer
 ...	0	Erika

+

Guthaben: 1000

Aktualisieren

Abbildung 4: Objektliste

Jede Zeile der Liste beschreibt ein Objekt und besteht aus folgenden Spalten:

Bild In der Liste wird das Bild – das mit 300×300 Pixeln aufgenommen wurde – aus Gründen der Übersichtlichkeit mit 40×40 Pixeln dargestellt. Durch Zoomen oder Anklicken des Bildes (Abschnitt 3.2.5) wird aber die volle Auflösung sichtbar.

Objekt In Abbildung 4 werden drei Punkte als Objektname angezeigt, damit das neue, „unbekannte“ Objekt ganz oben in der nach dem Objektnamen sortierten Liste erscheint. Nach dem Eintragen eines Namens im Detailfenster (Abschnitt 3.2.5 und Abschnitt 3.2.10) wird dieser in der Liste angezeigt (Abbildung 5).

Gebot Für das neue Objekt wurde noch kein Gebot abgegeben. In der gleichnamigen Spalte wird daher in Abbildung 4 eine Null angezeigt. Erst wenn Erika im Detailfenster (Abschnitt 3.2.5) ihr Gebot für das Objekt eingetragen hat, wird auch dieses in der Liste angezeigt (Abbildung 5).

Besitzer Da Erika das Objekt aufgenommen hat und noch niemand ein Gebot abgegeben hat, wird sie in der entsprechenden Spalte als Besitzerin aufgelistet.

3.2.5 Detailfenster

Erika kann jetzt in der Objektliste (Abbildung 4) das Bild oder „...“ oder „0“ oder „Erika“ anklicken, um das Detailfenster über der Objektliste zu öffnen (Abbildung 5).

Partitio - Erika

Index

Besitzer: Erika

Gebot: 42

Bestätigen

Löschen

Objekt	Gebot	Besitzer
Index	42	Erika

+

Guthaben: 958
Aktualisieren

Abbildung 5: Detailfenster

Nachdem sie das Objekt mit einem Namen versehen („Index“), ihr Gebot („42“) für das Objekt abgeben und den Bestätigen-Button angeklickt hat, erscheinen diese Daten in der Objektliste und das Detailfenster wird geschlossen⁶. Alternativ kann Erika das × rechts oben im Detailfenster anklicken, um es ohne Änderungen wieder zu schließen.

Erikas Guthaben (links unten in Abbildung 5) reduziert sich durch ihr Gebot von 42 jetzt natürlich auf $1000 - 42 = 958$.

3.2.6 Löschen

Jede Teilnehmerin kann Objekte – auch solche, die sie selbst nicht aufgenommen hat – löschen, indem sie im Detailfenster auf den Löschen-Button klickt. Es erscheint ein Löschbestätigungsfenster (Abbildung 6) und durch Anklicken des dortigen Löschen-Buttons wird das Objekt aus der Liste gelöscht⁷.

⁶In Abbildung 5 haben wir das Detailfenster nochmals geöffnet, um die Informationen im Detailfenster und der Objektliste gemeinsam darstellen zu können.

⁷Wir klicken in Abbildung 6 auf den Abbrechen-Button, sodass das Objekt nicht gelöscht wird.

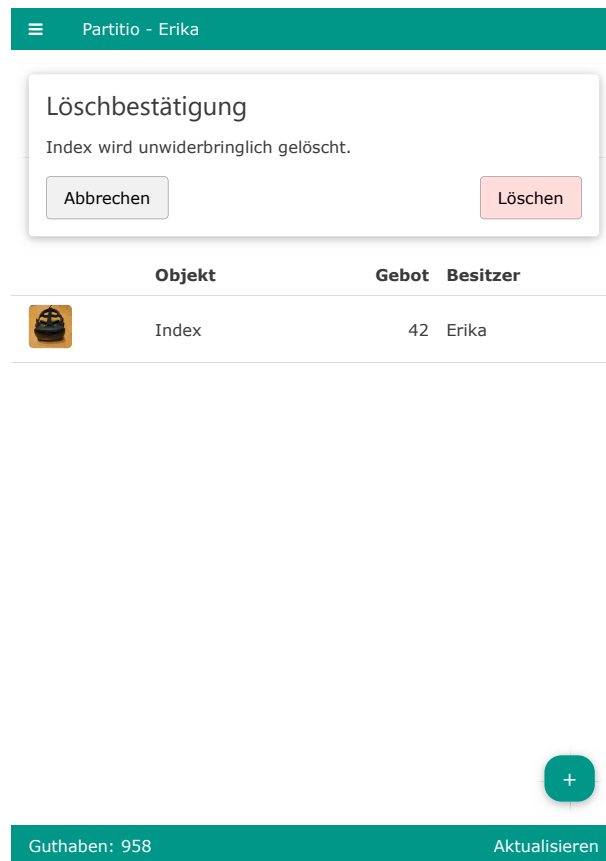


Abbildung 6: Löschbestätigung

3.2.7 Hamburger-Menü

Nach Anklicken des Hamburger-Buttons ($\equiv$ ganz links oben) öffnet sich das Hamburger-Menü (Abbildung 7).

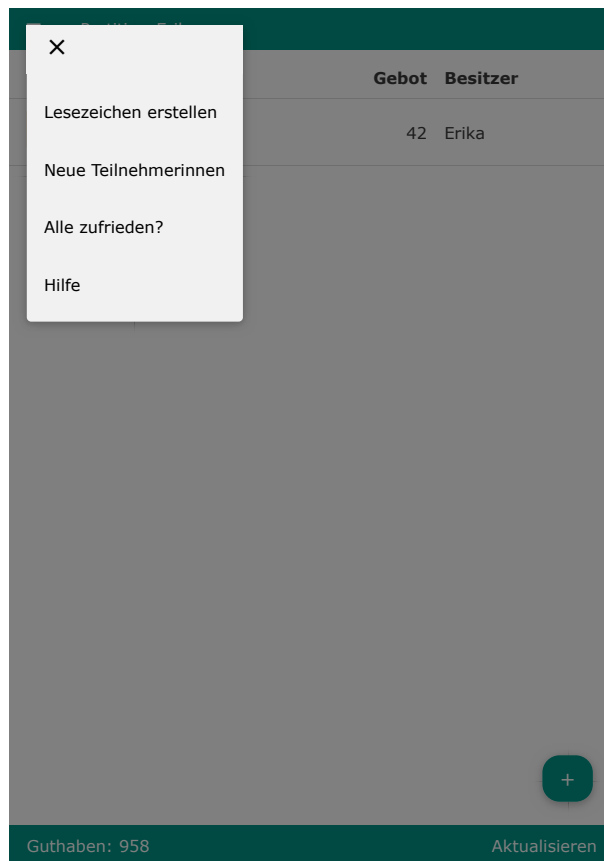


Abbildung 7: Hamburger Menü

Erika kann das Hamburger-Menü wieder schließen, indem sie auf das × im ersten Eintrag oder irgendwo außerhalb des Menüs klickt.

Ein Klick auf den zweiten Eintrag öffnet das schon in Abschnitt 3.2.2 beschriebene Lesezeichen erstellen-Fenster.

3.2.7.1 Neue Teilnehmerinnen

Ein Klick auf den dritten Eintrag des Hamburger-Menüs öffnet ein Fenster, in dem Erika weitere Teilnehmerinnen zu ihrem Projekt einladen kann (Abbildung 8).

Partitio - Erika

Neue Teilnehmerinnen

Um neue Teilnehmerinnen zu deinem Partitio-Projekt einzuladen, sendest du ihnen einfach den folgenden Link per Whatsapp, Signal, Telegram, SMS, ... oder E-Mail:

<https://m-server.fk5.hs-bremen.de/partitio/partitio.aspx?projekt=97ec4cc3-c413-479a-8e06-d8f5da09ba90&nutzer=42>

Beachte bitte:

- Wer den Link hat, kann an dem Projekt teilnehmen.
- Viele Menschen scheuen sich sinnvollerweise, auf Links in Nachrichten zu klicken. Eine E-Mail könnte im Spam-Ordner landen. Vielleicht möchtest du zusätzlich anrufen?

Link in Zwischenablage kopieren

	Objekt	Gebot	Besitzer
	Index	42	Erika

+

Guthaben: 958

Aktualisieren

Abbildung 8: Neue Teilnehmerinnen

Um beispielsweise ihren Bruder Heinz als weiteren Teilnehmer einzuladen, schickt Erika ihm den im Fenster angegebenen Link. Sie kann dazu im Fenster auf „E-Mail“ klicken, woraufhin sich ihr E-Mail-Programm mit einer vorbereiteten E-Mail öffnet. Sie muss nur noch Heinz’ E-Mail-Adresse eintragen und die E-Mail abschicken. Alternativ kann sie natürlich auch jeden anderen Kommunikationsweg (Whatsapp, SMS, Telefon, ...) nutzen, um Heinz den Link zukommen zu lassen.

Der Link unterscheidet sich von dem in Abschnitt 3.2.2 beschriebenen nur durch die angegebene Nutzer-Kennung:

Lesezeichen erstellen: Mit diesem Link kann sich Erika wieder als Erika in ihrem Projekt anmelden. Die Nutzer-Kennung ist ihre eigene Kennung:

nutzer=8ff97cb3-c7f3-4f15-a6ca-7957fb3495fe

Neue Teilnehmerinnen: Mit diesem Link lädt Erika neue Teilnehmerinnen zu ihrem Projekt ein. Die Nutzer-Kennung ist eine Kennung, die im Projekt ganz sicher nicht vorkommt:

nutzer=42

Auf diese Weise wird Heinz, sobald er den Link aufruft, automatisch zur Anmeldeseite umgeleitet, wo er sich anmeldet und eine eigene Nutzer-Kennung erhält.

3.2.7.2 Anonymität

Das Hantieren mit Projekt- und Nutzerkennungen mag auf den ersten Blick umständlich erscheinen, garantiert aber jeder Teilnehmerin absolute Anonymität. Auf diese Weise wird auf dem Partitio-Server keine einzige E-Mail-Adresse oder eine sonstige Information gespeichert, mit der die Objekte eines Projekts einer real existierenden Teilnehmerin zugeordnet werden könnten. Wer beispielsweise Omas

Goldbarren fotografiert und das Foto auf einen Server hoch lädt, möchte vermutlich nicht wirklich, dass die Goldbarren mit seiner „Adresse“ verknüpft werden, oder?

3.2.7.3 Alle zufrieden?

Ein Klick auf den vierten Eintrag des Hamburger-Menüs öffnet eine Seite (Abbildung 9)

Objekt	Gebot	Besitzer
 Index	42	Erika

Abbildung 9: Alle zufrieden?

auf der eine Teilnehmerin durch Anklicken des Zufrieden-Buttons eintragen kann, dass sie mit dem aktuellen Zustand des Projekts zufrieden ist und dass ihrer Meinung nach das Projekt jetzt beendet werden könnte⁸.

Wenn alle Teilnehmerinnen zufrieden sind, kann das Projekt beendet werden. In der aktuellen Version wird das Projekt nicht automatisch beendet, wenn alle Teilnehmerinnen zufrieden sind. Es ist also sinnvoll, ab und zu mal auf der Zufriedenheitsseite vorbeizuschauen, um sich über den aktuellen Zufriedenheitsstand des Projekts zu informieren. Natürlich kann die Übereinkunft zum Abschluss des Projekts und die dann folgende reale Aufteilung der Objekte auch über einen beliebigen anderen Kommunikationsweg erfolgen.

Wenn Heinz ein Objekt von Erika übernimmt, wird Erikas Zustand automatisch auf „unzufrieden“ gesetzt und das Projekt geht weiter.

3.2.7.3.1 Drucken

Am Ende des Projekts, wenn alle zufrieden sind, kann jede Teilnehmerin aus dem Browser heraus die Webseite mit der Objektliste ausdrucken, um ihre „ersteigerten“ Objekte zu dokumentieren; auf Papier oder in ein PDF-Dokument.

⁸Wir haben zu Demonstrationszwecken für Abbildung 9 vorher schon bei Erika auf den Zufrieden-Button geklickt.

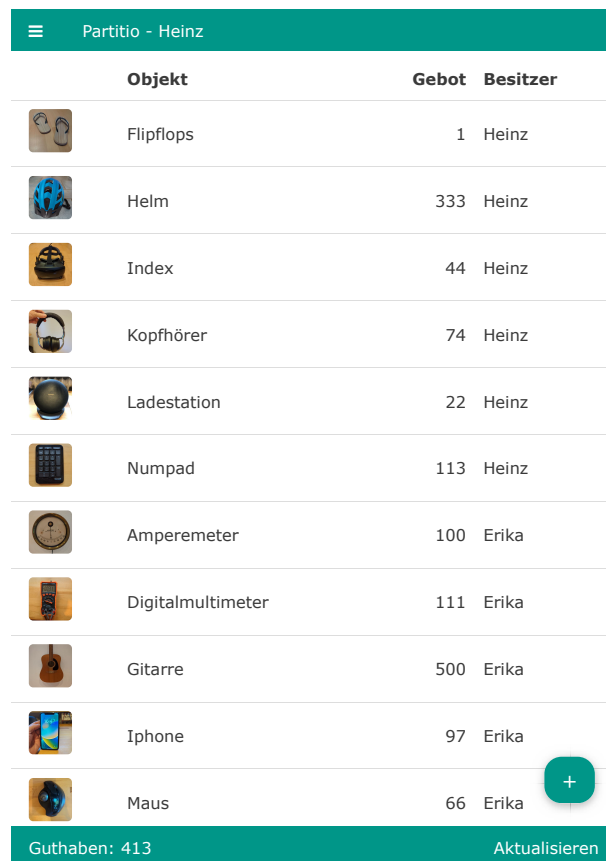
In Firefox gibt es nach einem Rechtsklick außerdem den Eintrag „Bildschirmfoto aufnehmen“, um die gesamte Seite als Screenshot abzuspeichern.

3.2.7.4 Hilfe

Ein Klick auf den fünften Eintrag des Hamburger-Menüs öffnet dieses PDF-Dokument.

3.2.8 Sortieren

In Abbildung 10 haben Erika und Heinz schon fleißig Objekte aufgenommen und Gebote abgegeben.



Partio - Heinz			
	Objekt	Gebot	Besitzer
	Flipflops	1	Heinz
	Helm	333	Heinz
	Index	44	Heinz
	Kopfhörer	74	Heinz
	Ladestation	22	Heinz
	Numpad	113	Heinz
	Amperemeter	100	Erika
	Digitalmultimeter	111	Erika
	Gitarre	500	Erika
	Iphone	97	Erika
	Maus	66	Erika

Guthaben: 413 Aktualisieren

Abbildung 10: Sortierreihenfolge: Besitzer, Objektname

Dabei ist anfänglich das erste Sortierkriterium in der Objektliste der Besitzer, wobei die Objekte des aktuellen Besitzers immer ganz oben angezeigt werden. Aus diesem Grund erscheinen seine Objekte in der Liste vor allen anderen, wenn Heinz das Projekt unter seinem Namen aufruft (Abbildung 10).

Das zweite Sortierkriterium innerhalb eines Besitzers ist der Objektname, sodass bei Heinz „Flipflops“ vor „Helm“ und bei Erika „Amperemeter“ vor „Digitalmultimeter“ gelistet sind.

In langen Listen kann es sinnvoll sein, alle Objekte – egal, wem sie gehören – alphabetisch zu sortieren. Dies geschieht durch Anklicken⁹ der Spaltenüberschrift „Objekt“ (Abbildung 11).

⁹Ein erneutes Anklicken der Spaltenüberschrift „Objekt“ sortiert die Objekte in umgekehrter alphabetischer Reihenfolge. Das „Numpad“ würde dann als erstes Objekt gelistet.

Partio - Heinz			
	Objekt	Gebot	Besitzer
	Amperemeter	100	Erika
	Digitalmultimeter	111	Erika
	Flipflops	1	Heinz
	Gitarre	500	Erika
	Helm	333	Heinz
	Index	44	Heinz
	Iphone	97	Erika
	Kopfhörer	74	Heinz
	Ladestation	22	Heinz
	Maus	66	Erika
	Numpad	113	Heinz
Guthaben: 413		Aktualisieren	

Abbildung 11: Sortiert nach Objektname

Nach Anklicken der Spaltenüberschrift „Gebot“ wird die Objektliste aufsteigend nach der Höhe des Gebots sortiert (Abbildung 12).

Partitio - Heinz		
Objekt	Gebot	Besitzer
 Flipflops	1	Heinz
 Ladestation	22	Heinz
 Index	44	Heinz
 Maus	66	Erika
 Kopfhörer	74	Heinz
 Iphone	97	Erika
 Amperemeter	100	Erika
 Digitalmultimeter	111	Erika
 Numpad	113	Heinz
 Helm	333	Heinz
 Gitarre	500	Erika
Guthaben: 413		Aktualisieren

Abbildung 12: Sortiert nach Gebot

Auf diese Weise lassen sich in längeren Listen vielleicht günstige „Schnäppchen“ finden, die von anderen Teilnehmerinnen als nicht so „wertvoll“ angesehen werden.

Des Weiteren sortiert ein Klick auf die Spaltenüberschrift „Besitzer“ natürlich nach dem Namen des Besitzers¹⁰.

Gibt es irgendein sinnvolles Kriterium, um nach den Bildern in der ersten Spalte zu sortieren ... Farbmittelwert ...?

3.2.9 Über- und Unterbieten

In einer üblichen Auktion gibt es für Gebote nur eine Richtung: nach oben. In Partitio kann der Preis eines Objektes hingegen sowohl steigen als auch sinken.

Konkret kann Heinz beispielsweise das Iphone, das sich gemäß Abbildung 12 momentan mit 97 in Erikas Besitz befindet, von Erika übernehmen, indem er in der Detailansicht als Gebot 98 einträgt und den Bestätigen-Button anklickt oder einfach die Enter-Taste drückt.

Wenn er versucht, das Iphone mit 96 zu übernehmen, bekommt er die Information (98 ... 413 in Abbildung 13), dass er mindestens 98 bieten muss und maximal 413 als Guthaben zur Verfügung hat. Er kann dann entweder ein Gebot von mindestens 98 abgeben, das Detailfenster durch Anklicken von × in der rechten oberen Ecke schließen oder einfach ein anderes Objekt auswählen.

¹⁰Um die anfängliche Sortierreihenfolge „Meine zuerst“ wieder herzustellen, ist in der aktuellen Partitio-Version ein erneutes Aufrufen des Projekts (partitio.de) notwendig.

Partitio - Heinz

Besitzer: Erika

Gebot: 98 ... 413

Bestätigen

Löschen

Objekt	Gebot	Besitzer
Flipflops	1	Heinz
Helm	333	Heinz
Index	44	Heinz
Kopfhörer	74	Heinz

Guthaben: 413

Aktualisieren

Abbildung 13: Übernehmen durch Unterbieten klappt nicht.

Wenn Heinz versucht, die Gitarre von Erika zu übernehmen, für die Erika ja momentan 500 geboten hat (Abbildung 12), wird er daran erinnert, dass sein aktuelles Guthaben (413) dazu nicht ausreicht (Abbildung 14).



Besitzer: Erika

Gebot: Nicht genug Guthaben

Objekt	Gebot	Besitzer
 Flipflops	1	Heinz
 Helm	333	Heinz
 Index	44	Heinz
 Kopfhörer	74	Heinz

Guthaben: 413

Abbildung 14: Nicht genug Guthaben

Wenn Heinz jetzt aber doch sehr an der Gitarre interessiert ist, kann er beispielsweise sein aktuelles Gebot für den Helm von 333 auf 200 **reduzieren**, um damit wieder 546 zur Verfügung zu haben und damit Erika die Gitarre für 501 abzunehmen (Abbildung 15).

Partio - Heinz			
	Objekt	Gebot	Besitzer
	Flipflops	1	Heinz
	Gitarre	501	Heinz
	Helm	200	Heinz
	Index	44	Heinz
	Kopfhörer	74	Heinz
	Ladestation	22	Heinz
	Numpad	113	Heinz
	Amperemeter	100	Erika
	Digitalmultimeter	111	Erika
	Iphone	97	Erika
	Maus	66	Erika
Guthaben: 45		Aktualisieren	

Abbildung 15: Gitarre übernommen

Durch die Reduktion seines Gebotes auf den Helm von 333 auf 200 geht Heinz natürlich das Risiko ein, dass Erika im Gegenzug den Helm beispielsweise für 201 übernimmt; aber schließlich ist Heinz die Gitarre ja offensichtlich wichtiger als der Helm, und er kann andererseits ja jederzeit sein Gebot für die Gitarre wieder erniedrigen und das für den Helm erhöhen.

Lange Rede, kurzer Sinn:

**Übernehmen geht nur durch Überbieten.
Gebote können aber jederzeit wieder erniedrigt werden.**

Ja, dadurch ist es möglich, ein Objekt durch Überbieten zu übernehmen und das Gebot gleich darauf wieder auf das alte Gebot (oder sogar noch weiter) zu erniedrigen und ja, das können natürlich auch andere Teilnehmerinnen machen. Auf diese Weise ist es möglich, dass ein Objekt pingpong-artig mehrfach den Besitzer wechselt, ohne dass sein Gebot faktisch steigt.

In der Spieltheorie gibt es für solche Situationen Regeln: So darf ein Kō¹¹ beispielsweise erst beim **übernächsten** Zug wieder genommen werden. Wir haben in dieser Partio-Version bewusst darauf verzichtet, komplizierte Kō-Regeln einzuführen und vertrauen darauf, dass Teilnehmerinnen automatisch andere Objekte ins Visier nehmen, wenn sie merken, dass es bei einzelnen Objekten zu sinnlosen Pingpong-Kämpfen kommt.

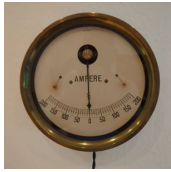
3.2.10 KI

In einer der nächsten Partio-Versionen ist geplant, den Objektnamen von einer lokalen KI direkt aus dem Bild vorschlagen zu lassen. Erste Versuche, die marktführende ChatGPT 4o [2] mit dem Prompt

¹¹Kō (dt. etwa: „Ewigkeit“) bezeichnet eine Stellung beim Brettspiel Go, in der beide Spieler wechselseitig einzelne Steine des Gegners schlagen könnten, wodurch die jeweils vorherige Situation wiederhergestellt würde.

Ich lade dir ein paar Bilder hoch. Beschreibe das Objekt auf dem jeweiligen Bild mit einem Wort.

auf die Bilder loszulassen, waren schon mal ziemlich erfolgversprechend:



Amperemeter



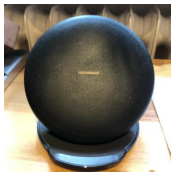
Headset



Gitarre



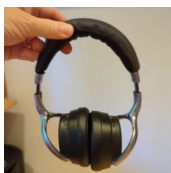
Nummernblock



Ladegerät



Flip-Flops



Kopfhörer



Multimeter



Helm



Smartphone



Trackball

Natürlich kann Chatty auf Nachfrage

Um welches Smartphone handelt es sich?

weitere Vermutungen anstellen

Das hochgeladene Smartphone scheint ein Apple iPhone zu sein, basierend auf dem abgebildeten Design. Aufgrund der Displayanzeige und des Kameraausschnitts könnte es sich um ein iPhone X¹² oder ein neueres Modell handeln. Genauer könnte es ein iPhone 12 oder iPhone 13 sein, da diese Modelle das gezeigte Design teilen. Um das Modell exakt zu bestimmen, wären genauere Informationen wie die Rückseite oder technische Details erforderlich.

aber wir hatten ja ganz ausdrücklich nur nach **einem** Wort als Beschreibung gefragt.

„Trackball“ und „Multimeter“ charakterisieren die Objekte vielleicht sogar besser als „Maus“ und „Digitalmultimeter“¹³.

¹²Es ist tatsächlich ein iPhone X.

¹³Ja, es gibt auch heute noch analoge Multimeter ...

4 Unter der Haube

Mittlerweile sind die künstlichen Intelligenz-Copiloten [3] so gut geworden, dass wir das Umwandeln der alten ASP.NET Webforms in ASP.NET Core Razor Pages komplett Chatty [2] überlassen haben. [4] Weiterhin haben wir die KI aufgefordert, jede einzelne von ihr erstellte Programmzeile ausführlich zu kommentieren und das Zusammenspiel der einzelnen Programmteile in Abschnitt 4.1 mit ihren Worten zu beschreiben. Mit anderen Worten alles von Abschnitt 4.1 bis Abschnitt 4.13 stammt aus Chatty's Feder.

4.1 Zusammenspiel der Programmteile

4.1.1 Grundstruktur der Anwendung

Die Anwendung ist in mehrere klar voneinander abgegrenzte Schichten unterteilt, die erst im Zusammenspiel das vollständige Verhalten des Systems erzeugen. Die Grundlage bilden die Datenmodelle aus Abschnitt 4.2. Auf ihnen bauen der Datenzugriff über Abschnitt 4.3 und Abschnitt 4.4 sowie die fachliche Verarbeitung über Abschnitt 4.5 und Abschnitt 4.6 auf. Die Laufzeitverdrahtung dieser Bestandteile erfolgt anschließend in Abschnitt 4.7. Erst danach greifen die Razor-spezifischen Dateien wie Abschnitt 4.8 und Abschnitt 4.9 sowie die eigentlichen Seitenpaare aus Abschnitt 4.10 und Abschnitt 4.11 beziehungsweise Abschnitt 4.12 und Abschnitt 4.13 ineinander.

Diese Reihenfolge ist nicht nur für die Darstellung im Dokument sinnvoll, sondern spiegelt auch die Architektur der Anwendung wider: Zunächst wird festgelegt, mit welchen Datenobjekten gearbeitet wird, danach, wie diese gespeichert und fachlich verarbeitet werden, anschließend, wie alle Teile zur Laufzeit miteinander verbunden werden, und zuletzt, wie die Ergebnisse im Browser dargestellt werden.

4.1.2 Datenmodelle als gemeinsames Fundament

Die in Abschnitt 4.2 definierten Klassen stellen das gemeinsame fachliche Vokabular der Anwendung bereit. Sie beschreiben die Struktur der Daten, mit denen alle anderen Programmteile arbeiten. Das Repository nutzt diese Modelle, um XML-Inhalte in anwendungsinterne Objekte zu überführen. Der Service verarbeitet dieselben Objekte weiter, ergänzt oder verändert sie, und die PageModels reichen sie schließlich an die Razor-Seiten zur Darstellung weiter.

Gerade diese gemeinsame Modellschicht ist für die Wartbarkeit entscheidend. Ohne sie müssten Persistenz, Fachlogik und Oberfläche jeweils eigene Datenstrukturen verwalten und laufend ineinander übersetzen. Durch die zentrale Definition in Abschnitt 4.2 bleiben die Verantwortlichkeiten sauber getrennt, während die beteiligten Schichten trotzdem mit denselben fachlichen Einheiten arbeiten.

4.1.3 Datenzugriff und Persistenz

Wie auf Projektdaten zugegriffen werden darf, wird in Abschnitt 4.3 festgelegt. Dieses Interface definiert die vertragliche Grenze zwischen Fachlogik und technischer Speicherung. Die eigentliche Implementierung dieses Vertrags erfolgt in Abschnitt 4.4. Dort übernimmt `XmlProjektRepository.cs` das Lesen, Interpretieren und Speichern der XML-basierten Projektdaten.

Das Repository ist damit die einzige Schicht, die unmittelbar mit dem konkreten Speicherformat arbeitet. Für die übrigen Teile der Anwendung ist nicht entscheidend, wie die Daten intern auf der Festplatte oder in XML organisiert sind, sondern nur, dass sie über das Repository in Form der in Abschnitt 4.2 beschriebenen Objekte bereitgestellt werden. Diese Entkopplung sorgt dafür, dass Änderungen an der Persistenzschicht nicht automatisch Änderungen an der Fachlogik oder der Oberfläche erzwingen.

4.1.4 Fachlogik und Service-Schicht

Die fachlichen Operationen, die der Webanwendung zur Verfügung stehen, werden zunächst abstrakt in Abschnitt 4.5 beschrieben. Die konkrete Umsetzung dieser Operationen liegt in Abschnitt 4.6. `PartitioService.cs` bildet damit die zentrale Verarbeitungsschicht der Anwendung: Es nimmt Anforderungen aus der Weboberfläche entgegen, nutzt das Repository für den Zugriff auf die Daten und führt darauf die eigentliche Geschäftslogik aus.

Der Service steht also zwischen Oberfläche und Persistenz. Er sorgt dafür, dass die Razor Pages nicht selbst XML-Dateien lesen, Datenstrukturen manipulieren oder fachliche Regeln implementieren müssen. Stattdessen konzentrieren sich die Seitenmodelle auf die Verarbeitung von Anfragen, während der Service die inhaltliche Arbeit übernimmt. Diese Aufteilung verhindert, dass Oberflächenlogik und Fachlogik unübersichtlich miteinander vermischt werden.

4.1.5 Verdrahtung zur Laufzeit

In Abschnitt 4.7 werden die zuvor getrennt definierten Bestandteile zu einer lauffähigen ASP.NET-Core-Anwendung zusammengesetzt. `Program.cs` registriert die Razor Pages, bindet Repository und Service in die Dependency Injection ein, konfiguriert die Middleware-Pipeline und legt fest, welche Endpunkte und Verarbeitungswege für eingehende HTTP-Anfragen zur Verfügung stehen.

`Program.cs` ist damit die zentrale Verdrahtungsstelle der Anwendung. Während die anderen Dateien beschreiben, **was** einzelne Teile fachlich oder technisch leisten, bestimmt Abschnitt 4.7, **wie** diese Teile im laufenden Betrieb zusammenfinden. Dazu gehört auch die Konfiguration der Fehlerbehandlung, der statischen Dateien, der zusätzlichen Upload-Endpunkte und der Weiterleitungen auf die eigentliche Hauptseite.

4.1.6 Gemeinsame Razor-Grundlagen

Bevor konkrete Razor-Seiten gerendert werden, wirken die gemeinsamen Grundlagen aus Abschnitt 4.8 und Abschnitt 4.9. `_ViewImports.cshtml` stellt Namespaces, Razor-Direktiven und Tag Helpers zentral bereit. Dadurch müssen diese Grundlagen nicht in jeder einzelnen `.cshtml`-Datei erneut notiert werden. `_ViewStart.cshtml` legt darüber hinaus gemeinsame Render-Einstellungen wie das Standardlayout fest und beeinflusst damit die Darstellung aller untergeordneten Seiten.

Diese beiden Dateien sind keine eigenständigen Inhaltsseiten, aber sie prägen das Verhalten der eigentlichen Ansichten wesentlich. Sie bilden somit eine technische Zwischenschicht zwischen der allgemeinen Anwendungsverdrahtung aus Abschnitt 4.7 und den konkret sichtbaren Razor-Seiten wie Abschnitt 4.11 und Abschnitt 4.13.

4.1.7 Hauptseite: PageModel und Ansicht

Die zentrale Benutzerinteraktion der Anwendung wird durch das Paar aus Abschnitt 4.10 und Abschnitt 4.11 realisiert. In Abschnitt 4.10 verarbeitet das PageModel eingehende Anfragen an die Hauptseite, ruft dafür die benötigten Methoden aus Abschnitt 4.6 auf und bereitet die Ergebnisse für die Darstellung vor. Es bildet also die Verbindung zwischen dem ASP.NET-Core-Request-Lebenszyklus und der fachlichen Logik des Systems.

Die eigentliche Ausgabe im Browser erfolgt dann in Abschnitt 4.11. `Partitio.cshtml` enthält das Razor-Markup, die HTML-Struktur und die benutzerseitigen Interaktionselemente. Die Datei greift auf die vom PageModel vorbereiteten Daten zu und stellt sie in der vorgesehenen Form dar. Dadurch bleibt die Aufgabenverteilung klar: Das PageModel steuert den Ablauf und die Datenbeschaffung, die Ansicht übernimmt die Präsentation.

4.1.8 Fehlerbehandlung als eigener Ablauf

Neben der Hauptseite besitzt die Anwendung auch für Fehlerfälle eine eigene, sauber getrennte Struktur. In Abschnitt 4.7 wird festgelegt, dass unbehandelte Ausnahmen kontrolliert auf die Fehlerseite umgeleitet werden. Die serverseitige Vorbereitung dieser Fehleranzeige erfolgt in Abschnitt 4.12. Dort werden insbesondere Diagnoseinformationen wie die Request-ID ermittelt und für die Ausgabe bereitgestellt.

Die eigentliche Darstellung dieser Informationen findet dann in Abschnitt 4.13 statt. Auch hier zeigt sich dieselbe Struktur wie auf der Hauptseite: Das `.cshtml.cs`-Modell verarbeitet die serverseitigen Informationen, während die `.cshtml`-Datei diese Daten in sichtbares Razor-Markup übersetzt. Damit folgt selbst die Fehlerdarstellung derselben architektonischen Logik wie der restliche Aufbau der Anwendung.

4.1.9 Zusammenfassung des Zusammenspiels

Insgesamt entsteht das Verhalten der Anwendung also durch ein gestuftes Zusammenwirken aller Teile. Abschnitt 4.2 definiert die gemeinsamen Datenobjekte. Abschnitt 4.3 und Abschnitt 4.4 regeln den Zugriff auf diese Daten. Abschnitt 4.5 und Abschnitt 4.6 kapseln die fachliche Verarbeitung. Abschnitt 4.7 verbindet diese Komponenten zur Laufzeit zu einer funktionierenden Webanwendung. Abschnitt 4.8 und Abschnitt 4.9 schaffen die gemeinsamen technischen Voraussetzungen für die Razor-Darstellung. Schließlich bilden Abschnitt 4.10 und Abschnitt 4.11 sowie Abschnitt 4.12 und Abschnitt 4.13 die sichtbaren Seitenpaare für Normal- und Fehlerfall.

Gerade diese klare Schichtung macht das System nachvollziehbar: Jeder Programmteil besitzt eine eigene Verantwortung, und erst das koordinierte Zusammenspiel aller Dateien erzeugt das vollständige Verhalten der Anwendung.

4.2 PartitioModels.cs

```
/*
 * Diese Datei bündelt die zentralen Datenmodelle der Anwendung Partitio.
 * Sie beschreibt damit die in-memory-Strukturen, mit denen Projektzustände,
 * Nutzer, Objekte und verschiedene Ergebnis- oder Anzeigeformen innerhalb der
 * Anwendung transportiert werden. Ein Teil dieser Klassen repräsentiert den
 * eigentlichen fachlichen Zustand eines Projekts, also das, was gespeichert
 * und wieder geladen wird. Andere Klassen dienen als speziell zugeschnittene
 * Rückgabeformen für Service-Methoden oder die Benutzeroberfläche, etwa für
 * Listenansichten, Detailansichten oder Validierungsergebnisse. Die Datei ist
 * damit ein gemeinsamer Bezugspunkt zwischen Persistenz, Fachlogik und UI, weil
 * hier festgelegt wird, welche Daten in welcher Form innerhalb des Systems
 * existieren und weitergereicht werden.
 */

// Deklariert den Namespace für alle Modell- und DTO-Klassen der Anwendung.
namespace Partitio.Models;

// Repräsentiert den vollständigen fachlichen Zustand eines Projekts.
// Ein Projekt besteht hier aus einer Sammlung von Nutzern und einer Sammlung von Objekten.
public sealed class ProjektDaten
{
    // Enthält alle Nutzer, die zu diesem Projekt gehören.
    // Die Liste ist schreibbar über ihren Inhalt, aber die Property selbst bleibt unveränderlich
    referenziert.
    public List<NutzerDaten> Nutzer { get; } = new();

    // Enthält alle Objekte, die innerhalb dieses Projekts verwaltet werden.
    public List<ObjektDaten> Objekte { get; } = new();
}

// Beschreibt die gespeicherten Stammdaten eines einzelnen Nutzers.
```

```

public sealed class NutzerDaten
{
    // Eindeutige technische ID des Nutzers, typischerweise als GUID erzeugt.
    public string NutzerId { get; set; } = string.Empty;

    // Anzeigename des Nutzers, wie er in Oberfläche und Listen erscheinen soll.
    public string NutzerName { get; set; } = string.Empty;

    // Kennzeichnet, ob der Nutzer im aktuellen fachlichen Zustand als zufrieden markiert ist.
    public bool Zufrieden { get; set; }
}

// Beschreibt ein einzelnes Objekt innerhalb eines Projekts.
public sealed class ObjektDaten
{
    // Eindeutige ID des Objekts, die zugleich häufig als technischer Bezug für Bilddateien dient.
    public string ObjektId { get; set; } = string.Empty;

    // Frei bearbeitbarer Name bzw. Titel des Objekts.
    // "..." dient als Platzhalter, solange noch kein sinnvoller Name vergeben wurde.
    public string ObjektName { get; set; } = "...";

    // Nutzer-ID des aktuellen Besitzers dieses Objekts.
    public string BesitzerId { get; set; } = string.Empty;

    // Aktuelles Gebot bzw. der aktuell gebundene Wert dieses Objekts.
    public int ObjektGebot { get; set; }
}

// Liefert eine auf die Listenansicht zugeschnittene Darstellung eines Objekts.
// Diese Klasse ist bewusst flacher und UI-näher als das zugrunde liegende ObjektDaten-Modell.
public sealed class ObjektListeEintrag
{
    // Technische Objekt-ID für Navigation, Aktionen oder DOM-/Routing-Bezüge.
    public string Id { get; set; } = string.Empty;

    // Relativer Bildpfad, unter dem die Oberfläche das Objektbild laden kann.
    public string Bild { get; set; } = string.Empty;

    // Anzeigename des Objekts in der Liste.
    public string Objekt { get; set; } = string.Empty;

    // Sichtbarer Gebotswert des Objekts.
    public int Gebot { get; set; }

    // Aufgelöster Besitzername für die Anzeige in der Objektliste.
    public string Besitzer { get; set; } = string.Empty;
}

// Enthält die Daten für eine Detailansicht oder Detailbearbeitung eines Objekts.
// Neben den eigentlichen Objektdaten werden hier auch berechnete Gebotsgrenzen mitgegeben.
public sealed class ObjektDetailDaten
{
    // Eindeutige ID des betrachteten Objekts.
    public string ObjektId { get; set; } = string.Empty;

    // Aktueller Anzeigename des Objekts.
    public string ObjektName { get; set; } = string.Empty;

    // Technische ID des aktuellen Besitzers.
    public string BesitzerId { get; set; } = string.Empty;

    // Für die Oberfläche bereits aufgelöster Name des aktuellen Besitzers.
    public string BesitzerName { get; set; } = string.Empty;

    // Aktuelles Gebot des Objekts.
    public int ObjektGebot { get; set; }

    // Kleinster fachlich zulässiger Gebotswert für den anfragenden Nutzer.
    public int MinGebot { get; set; }
}

```

```

// Größter fachlich zulässiger Gebotswert für den anfragenden Nutzer.
public int MaxGebot { get; set; }

// Gibt an, ob unter den aktuellen Bedingungen überhaupt ein gültiges Gebot möglich ist.
public bool GebotMoeglich { get; set; }

// Textlicher Hinweis für die Oberfläche, z. B. ein erlaubter Bereich oder eine Fehlersituation.
public string GebotHinweis { get; set; } = string.Empty;
}

// Stellt einen einfachen Anzeigeeintrag für den Zufriedenheitsstatus eines Nutzers dar.
public sealed class ZufriedenheitEintrag
{
    // Name des Nutzers, so wie er in der Zufriedenheitsübersicht angezeigt werden soll.
    public string NutzerName { get; set; } = string.Empty;

    // Zugehöriger boolescher Zufriedenheitswert.
    public bool Zufrieden { get; set; }
}

// Kapselt das Ergebnis einer Nutzererstellung.
// Statt nur true/false zurückzugeben, kann die Anwendung hier auch Zusatzinformationen liefern.
public sealed class NutzerErstellungErgebnis
{
    // Zeigt an, ob die Nutzererstellung erfolgreich war.
    public bool Erfolg { get; set; }

    // Enthält im Erfolgsfall die neu erzeugte Nutzer-ID.
    public string? NutzerId { get; set; }

    // Enthält im Fehlerfall eine textliche Beschreibung des Problems.
    public string? Fehlermeldung { get; set; }

    // Kann zur Unterstützung der Oberfläche bereits vergebene Namen mitliefern,
    // etwa wenn der gewünschte Name schon existiert.
    public IReadOnlyList<string> VorhandeneNamen { get; set; } = Array.Empty<string>();
}

// Kapselt das Ergebnis eines Objekt-Speichervorgangs.
// Diese Struktur transportiert sowohl Erfolg als auch Validierungsinformationen.
public sealed class ObjektSpeichernErgebnis
{
    // Kennzeichnet, ob das Objekt erfolgreich gespeichert wurde.
    public bool Erfolg { get; set; }

    // Optionaler Fehler- oder Validierungshinweis bei nicht erfolgreichem Speichern.
    public string? Fehlermeldung { get; set; }

    // Optionaler minimal zulässiger Gebotswert, falls die Oberfläche einen Bereich anzeigen soll.
    public int? MinGebot { get; set; }

    // Optionaler maximal zulässiger Gebotswert, passend zum gleichen Validierungskontext.
    public int? MaxGebot { get; set; }
}

```

4.3 IProjektRepository.cs

```

/*
 * Dieses Interface beschreibt den datenhaltenden Zugriff auf Projekte in der Anwendung.
 * Während der fachliche Service Regeln und Abläufe kapselt, stellt das Repository die
 * elementaren Operationen bereit, mit denen ein Projekt geprüft, angelegt, gelesen und
 * aktualisiert werden kann. Die Fachlogik bleibt dadurch unabhängig von der konkreten
 * Speicherform. In diesem Projekt übernimmt eine XML-basierte Implementierung diese
 * Aufgaben, aber durch das Interface könnte später ebenso eine andere Persistenzstrategie
 * eingesetzt werden, ohne die übrige Anwendung grundlegend zu verändern.
 */

// Importiert die Modellklasse ProjektDaten, die vom Repository gelesen und aktualisiert wird.
using Partitio.Models;

```

```
// Ordnet das Repository dem Service-Namespace der Anwendung zu.
namespace Partitio.Services;

// Definiert die minimale Persistenzschnittstelle für Projektzugriffe.
public interface IProjektRepository
{
    // Prüft, ob für eine bestimmte Projekt-ID bereits eine persistierte Datenbasis existiert.
    Task<bool> ExistsAsync(string projektId, CancellationToken cancellationToken = default);

    // Legt ein neues, leeres Projekt an, falls es noch nicht vorhanden ist.
    Task CreateAsync(string projektId, CancellationToken cancellationToken = default);

    // Liest den vollständigen Zustand eines Projekts aus der Persistenz und liefert ihn als Modell zurück.
    Task<ProjektDaten> ReadAsync(string projektId, CancellationToken cancellationToken = default);

    // Führt eine Änderungsfunktion auf einem Projekt aus, speichert anschließend den neuen Zustand
    // und gibt ein beliebiges Ergebnis der Änderungsfunktion an den Aufrufer zurück.
    Task<T> UpdateAsync<T>(string projektId, Func<ProjektDaten, T> update, CancellationToken
cancellationToken = default);
}
```

4.4 XmlProjektRepository.cs

```
/*
 * Diese Klasse ist die konkrete XML-basierte Implementierung des Projekt-Repositorys.
 * Sie übernimmt die technische Persistenz der Projektdaten im Dateisystem und bildet
 * die Brücke zwischen den in C# verwendeten Modellobjekten und den XML-Dateien auf der
 * Festplatte. Zusätzlich schützt sie projektbezogene Lese- und Schreibzugriffe mit
 * Sperren vor konkurrierenden Zugriffen innerhalb des Prozesses. Im Kontext dieser
 * Anwendung bedeutet das: Für jede Projekt-ID existiert eine XML-Datei, die Nutzer und
 * Objekte enthält. Die Klasse kann diese Dateien anlegen, lesen, in Modellobjekte
 * umwandeln, Änderungen zurück in XML serialisieren und dabei pro Projekt geordnete
 * Schreib-/Leseoperationen sicherstellen.
 */

// Importiert Thread-sichere ConcurrentDictionary-Strukturen für projektbezogene Locks.
using System.Collections.Concurrent;

// Importiert LINQ-to-XML-Typen zum Laden, Lesen und Schreiben von XML-Dokumenten.
using System.Xml.Linq;

// Importiert die Modellklassen, in die XML-Daten eingelesen bzw. aus denen sie geschrieben werden.
using Partitio.Models;

// Ordnet die Repository-Implementierung dem Service-Namespace der Anwendung zu.
namespace Partitio.Services;

// Konkrete, nicht weiter vererbzbare XML-Repository-Implementierung.
public sealed class XmlProjektRepository : IProjektRepository
{
    // Hält pro Projekt-ID genau ein SemaphoreSlim, um gleichzeitige Zugriffe zu serialisieren.
    private static readonly ConcurrentDictionary<string, SemaphoreSlim> ProjektLocks = new();

    // Speichert das Basisverzeichnis, in dem die XML-Projektdateien liegen.
    private readonly string _projektePfad;

    // Konstruktor, der über das Hosting-Environment den Projektpfad bestimmt.
    public XmlProjektRepository(IWebHostEnvironment environment)
    {
        // Standardverzeichnis innerhalb des Content-Roots der Anwendung.
        var projektePfad = Path.Combine(environment.ContentRootPath, "projekte");

        // Fallback für ein älteres Verzeichnislayout außerhalb des aktuellen Content-Roots.
        var legacyProjektePfad = Path.Combine(Directory.GetParent(environment.ContentRootPath)?.FullName ??
string.Empty, "projekte");

        // Nutzt den Legacy-Pfad weiter, falls nur dieser existiert.
        if (!Directory.Exists(projektePfad) && Directory.Exists(legacyProjektePfad))

```

```

    {
        projektePfad = legacyProjektePfad;
    }

    // Speichert den ermittelten Pfad in einem Feld für alle späteren Zugriffe.
    _projektePfad = projektePfad;

    // Stellt sicher, dass das Verzeichnis tatsächlich vorhanden ist.
    Directory.CreateDirectory(_projektePfad);
}

// Prüft, ob bereits eine XML-Datei für die angegebene Projekt-ID existiert.
public Task<bool> ExistsAsync(string projektId, CancellationToken cancellationToken = default)
{
    // Die Existenzprüfung ist synchron möglich und wird deshalb direkt in ein Task-Ergebnis verpackt.
    return Task.FromResult(File.Exists(HoleProjektDatei(projektId)));
}

// Legt eine neue Projektdatei mit leerer Grundstruktur an.
public async Task CreateAsync(string projektId, CancellationToken cancellationToken = default)
{
    // Holt den projektspezifischen Lock.
    var sem = HoleLock(projektId);

    // Wartet asynchron, bis exklusiver Zugriff auf dieses Projekt möglich ist.
    await sem.WaitAsync(cancellationToken);
    try
    {
        // Ermittelt den vollständigen Dateipfad für das Projekt.
        var datei = HoleProjektDatei(projektId);

        // Wenn die Datei bereits existiert, muss nichts mehr angelegt werden.
        if (File.Exists(datei))
        {
            return;
        }

        // Erstellt ein minimales XML-Dokument mit Wurzelement und leeren Sammlungen.
        var dokument = new XDocument(
            new XElement("alles",
                new XElement("nutzers"),
                new XElement("objekte")));

        // Schreibt das neue XML-Dokument auf die Festplatte.
        dokument.Save(datei);
    }
    finally
    {
        // Gibt den Lock auch bei Fehlern zuverlässig wieder frei.
        sem.Release();
    }
}

// Liest die XML-Datei eines Projekts und wandelt sie in ein ProjektDaten-Objekt um.
public async Task<ProjektDaten> ReadAsync(string projektId, CancellationToken cancellationToken = default)
{
    // Holt den projektspezifischen Lock.
    var sem = HoleLock(projektId);

    // Wartet auf exklusiven Zugriff, damit Lesen und Schreiben nicht kollidieren.
    await sem.WaitAsync(cancellationToken);
    try
    {
        // Lädt das XML-Dokument von der Festplatte.
        var dokument = XDocument.Load(HoleProjektDatei(projektId));

        // Parsiert das XML in die interne Modellstruktur der Anwendung.
        return ParseProjekt(dokument);
    }
    finally

```

```

    {
        // Gibt den Lock wieder frei.
        sem.Release();
    }
}

// Lädt ein Projekt, führt eine Änderungsfunktion aus, speichert das Ergebnis und liefert einen
Rückgabewert zurück.
public async Task<T> UpdateAsync<T>(
    string projektId,
    Func<ProjektDaten, T> update,
    CancellationToken cancellationToken = default)
{
    // Holt den projektspezifischen Lock.
    var sem = HoleLock(projektId);

    // Wartet auf exklusiven Zugriff für die komplette Read-Modify-Write-Sequenz.
    await sem.WaitAsync(cancellationToken);
    try
    {
        // Bestimmt die Projektdatei.
        var datei = HoleProjektDatei(projektId);

        // Liest das aktuelle XML ein.
        var dokument = XDocument.Load(datei);

        // Wandelt die XML-Daten in das veränderbare Projektmodell um.
        var projekt = ParseProjekt(dokument);

        // Führt die vom Aufrufer gelieferte Änderungsfunktion aus.
        var ergebnis = update(projekt);

        // Persistiert den geänderten Projektzustand zurück in dieselbe Datei.
        SaveProjekt(datei, projekt);

        // Gibt das Ergebnis der Änderungsfunktion an den Aufrufer weiter.
        return ergebnis;
    }
    finally
    {
        // Gibt den Lock in jedem Fall wieder frei.
        sem.Release();
    }
}

// Ermittelt den Dateinamen eines Projekts aus Basisverzeichnis und Projekt-ID.
private string HoleProjektDatei(string projektId)
{
    // Erwartet das Format "<projektId>.xml" innerhalb des Projektverzeichnisses.
    return Path.Combine(_projektePfad, $"{projektId}.xml");
}

// Liefert das Semaphore für eine Projekt-ID und legt es bei Bedarf neu an.
private static SemaphoreSlim HoleLock(string projektId)
{
    // GetOrAdd stellt sicher, dass für dieselbe ID stets dieselbe Lock-Instanz verwendet wird.
    return ProjektLocks.GetOrAdd(projektId, _ => new SemaphoreSlim(1, 1));
}

// Wandelt ein XML-Dokument in die in-memory-Repräsentation ProjektDaten um.
private static ProjektDaten ParseProjekt(XDocument dokument)
{
    // Erstellt ein leeres Projektmodell, das im Verlauf mit XML-Daten gefüllt wird.
    var projekt = new ProjektDaten();

    // Liest das Wurzelement des Dokuments.
    var wurzel = dokument.Root;

    // Fehlt die Wurzel, bleibt ein leeres Projekt zurück.
    if (wurzel is null)
    {

```

```

        return projekt;
    }

    // Liest alle Nutzerelemente; fehlt der XML-Knoten, wird eine leere Sequenz verwendet.
    var nutzers = wurzel.Element("nutzers")?.Elements("nutzer") ?? Enumerable.Empty<XElement>();
    foreach (var nutzer in nutzers)
    {
        // Erzeugt aus jedem XML-Nutzereintrag ein NutzerDaten-Objekt und hängt es an das Projekt an.
        projekt.Nutzers.Add(new NutzerDaten
        {
            NutzerId = nutzer.Element("nutzer_id")?.Value ?? string.Empty,
            NutzerName = nutzer.Element("nutzer_name")?.Value ?? string.Empty,
            // Interpretiert nur den Text "true" als wahr, unabhängig von Groß-/Kleinschreibung.
            Zufrieden = string.Equals(nutzer.Element("zufrieden")?.Value, "true",
StringComparison.OrdinalIgnoreCase)
        });
    }

    // Liest alle Objektelemente; auch hier wird bei fehlendem XML-Knoten auf eine leere Sequenz
    zurückgefallen.
    var objekte = wurzel.Element("objekte")?.Elements("objekt") ?? Enumerable.Empty<XElement>();
    foreach (var objekt in objekte)
    {
        // Versucht, das Gebot als Integer zu lesen; bei Fehlschlag bleibt der Wert 0.
        _ = int.TryParse(objekt.Element("objekt_gebot")?.Value, out var gebot);

        // Überführt jeden XML-Objekteintrag in ein ObjektDaten-Modell.
        projekt.Objekte.Add(new ObjektDaten
        {
            ObjektId = objekt.Element("objekt_id")?.Value ?? string.Empty,
            ObjektName = objekt.Element("objekt_name")?.Value ?? "...",
            BesitzerId = objekt.Element("besitzer_id")?.Value ?? string.Empty,
            ObjektGebot = gebot
        });
    }

    return projekt;
}

// Serialisiert ein Projektmodell zurück in das XML-Format der Anwendung und speichert es.
private static void SaveProjekt(string datei, ProjektDaten projekt)
{
    // Baut das vollständige XML-Dokument aus den aktuellen Nutzer- und Objektdaten neu auf.
    var dokument = new XDocument(
        new XElement("alles",
            new XElement("nutzers",
                projekt.Nutzers.Select(n =>
                    new XElement("nutzer",
                        new XElement("nutzer_id", n.NutzerId),
                        new XElement("nutzer_name", n.NutzerName),
                        new XElement("zufrieden", n.Zufrieden ? "true" : "false")))),
            new XElement("objekte",
                projekt.Objekte.Select(o =>
                    new XElement("objekt",
                        new XElement("objekt_id", o.ObjektId),
                        new XElement("objekt_name", o.ObjektName),
                        new XElement("besitzer_id", o.BesitzerId),
                        new XElement("objekt_gebot", o.ObjektGebot))))));

    // Schreibt das rekonstruierte XML vollständig in die Zielfeile.
    dokument.Save(datei);
}
}

```

4.5 IPartitioService.cs

```

/*
 * Dieses Interface definiert die fachliche Service-Oberfläche der Anwendung Partitio.
 * Es beschreibt also nicht die technische Speicherung der Daten, sondern die

```

```

* Geschäftsoperationen, die für die Benutzungslogik benötigt werden: Projekte finden
* oder anlegen, Nutzer verwalten, Guthaben berechnen, Objektlisten und Objektdetails
* laden, neue Objekte erzeugen, Gebote speichern, Objekte löschen und die Zufriedenheit
* eines Nutzers setzen. Dadurch können aufrufende Teile der Anwendung, etwa Razor Pages
* oder API-Endpunkte, gegen eine stabile Abstraktion programmieren, ohne die konkrete
* Implementierung kennen zu müssen. Die tatsächliche Ausführung dieser Logik erfolgt in
* der Klasse PartitioService, die dieses Interface implementiert.
*/

// Importiert die Modelltypen, die als Rückgabewerte und Datentransferobjekte
// in der Service-Schnittstelle verwendet werden.
using Partitio.Models;

// Deklariert den Namespace, in dem die Service-Schnittstellen und Service-Klassen
// der Anwendung organisatorisch zusammengefasst sind.
namespace Partitio.Services;

// Definiert den Vertrag für die zentrale Fachlogik rund um Projekte, Nutzer und Objekte.
public interface IPartitioService
{
    // Stellt sicher, dass eine verwendbare Projekt-ID existiert:
    // Entweder wird eine vorhandene ID übernommen oder bei Bedarf ein neues Projekt angelegt.
    Task<string> EnsureProjektIdAsync(string? vorgeschlageneProjektId, CancellationToken cancellationToken
= default);

    // Prüft, ob ein Nutzer mit der angegebenen ID innerhalb des Projekts vorhanden ist.
    Task<bool> NutzerExistiertAsync(string projektId, string nutzerId, CancellationToken cancellationToken
= default);

    // Lädt die Daten eines einzelnen Nutzers oder liefert null, wenn kein passender Nutzer existiert.
    Task<NutzerDaten?> HoleNutzerAsync(string projektId, string nutzerId, CancellationToken
cancellationToken = default);

    // Erzeugt einen neuen Nutzer innerhalb eines Projekts und liefert ein Ergebnisobjekt
    // mit Erfolgsstatus, möglicher Fehlerbeschreibung und ggf. der neuen Nutzer-ID zurück.
    Task<NutzerErstellungErgebnis> ErstelleNutzerAsync(string projektId, string name, CancellationToken
cancellationToken = default);

    // Berechnet das noch verfügbare Guthaben eines Nutzers auf Basis seiner aktuellen Gebote.
    Task<int> BerechneGuthabenAsync(string projektId, string nutzerId, CancellationToken cancellationToken
= default);

    // Lädt die Objektliste eines Projekts für die Anzeige und berücksichtigt optionale Sortierparameter.
    Task<IReadOnlyList<ObjektListeEintrag>> LadeObjektlisteAsync(
        // ID des Projekts, aus dem die Objekte gelesen werden sollen.
        string projektId,

        // ID des aktuell betrachtenden Nutzers, z. B. für Standardsortierung oder Besitz-Markierung.
        string nutzerId,

        // Name der Spalte bzw. des Felds, nach dem sortiert werden soll.
        string? sortExpression,

        // Gewünschte Sortierrichtung, typischerweise ASC oder DESC.
        string? sortDirection,

        // Optionales Abbruchsignal für asynchrone Aufrufe.
        CancellationToken cancellationToken = default);

    // Lädt Detailinformationen zu genau einem Objekt, inklusive Gebotsgrenzen für den aktuellen Nutzer.
    Task<ObjektDetailDaten?> HoleObjektDetailAsync(
        // Projektkontext des gesuchten Objekts.
        string projektId,

        // Nutzer, für den das Detail geladen wird, damit Guthaben und Gebotslogik berechnet werden können.
        string nutzerId,

        // Eindeutige ID des gewünschten Objekts.
        string objektId,

        // Optionales Cancellation-Token für Request-Abbruch oder Timeout-Steuerung.

```

```

        CancellationToken cancellationToken = default);

// Lädt die Zufriedenheitsinformationen aller Nutzer eines Projekts.
Task<IReadOnlyList<ZufriedenheitEintrag>> LadeZufriedenheitenAsync(
    // Projekt, dessen Zufriedenheitszustände abgefragt werden sollen.
    string projektId,

    // Optionales Abbruchsignal.
    CancellationToken cancellationToken = default);

// Fügt einem Projekt ein neues Objekt hinzu oder bestätigt ein bereits vorhandenes Objekt.
Task<bool> FuegeObjektHinzuAsync(string projektId, string nutzerId, string objektId, CancellationToken
cancellationToken = default);

// Speichert Name, Besitzer und Gebot eines Objekts und validiert dabei die fachlichen Gebotsregeln.
Task<ObjektSpeichernErgebnis> SpeichereObjektAsync(
    // Zielprojekt, in dem das Objekt geändert werden soll.
    string projektId,

    // Nutzer, der das Objekt gerade beansprucht oder bearbeitet.
    string nutzerId,

    // ID des zu speichernden Objekts.
    string objektId,

    // Vom Client übergebener Objektname.
    string objektName,

    // Neues Gebot, das für dieses Objekt gesetzt werden soll.
    int objektGebot,

    // Optionales Abbruchsignal für den asynchronen Speichervorgang.
    CancellationToken cancellationToken = default);

// Entfernt ein Objekt aus einem Projekt.
Task<bool> LoescheObjektAsync(string projektId, string objektId, CancellationToken cancellationToken =
default);

// Setzt den Zufriedenheitsstatus eines Nutzers innerhalb eines Projekts.
Task<bool> SetzeZufriedenheitAsync(string projektId, string nutzerId, bool zufrieden, CancellationToken
cancellationToken = default);
}

```

4.6 PartitioService.cs

```

/*
 * Diese Klasse implementiert die zentrale Fachlogik der Anwendung Partitio.
 * Sie vermittelt zwischen den aufrufenden Web-Komponenten und dem Repository,
 * das die eigentlichen Projektdaten speichert. Hier werden alle Regeln umgesetzt,
 * die das Verhalten der Anwendung bestimmen: neue Projekte anlegen, Nutzer erzeugen,
 * Budget berechnen, Objektlisten zusammenstellen, zulässige Gebote bestimmen,
 * Objektänderungen speichern, Eigentümerwechsel behandeln und Zufriedenheitszustände
 * aktualisieren. Die Klasse arbeitet bewusst nicht direkt mit HTTP oder UI-Details,
 * sondern rein mit Domänenwerten und Modellobjekten. Genau dadurch bleibt die
 * Geschäftslogik testbar, wiederverwendbar und klar von Transport- oder Anzeigeaspekten getrennt.
 */

// Importiert die Modelltypen, die in Rückgabewerten und Datenmanipulationen verwendet werden.
using Partitio.Models;

// Legt den Namespace für die Service-Schicht fest.
namespace Partitio.Services;

// "sealed" verhindert weitere Vererbung und macht klar, dass diese Implementierung
// als konkrete, abgeschlossene Service-Klasse gedacht ist.
public sealed class PartitioService : IPartitioService
{
    // Definiert das feste Startbudget, das jedem Nutzer rechnerisch zur Verfügung steht.
    private const int StartGuthaben = 1000;
}

```

```

// Referenz auf das Repository, über das Projekte geladen und gespeichert werden.
private readonly IProjektRepository _projektRepository;

// Konstruktor für Dependency Injection: Das Repository wird von außen bereitgestellt.
public PartitioService(IProjektRepository projektRepository)
{
    // Speichert die Repository-Instanz für die spätere Verwendung in allen Methoden.
    _projektRepository = projektRepository;
}

// Sorgt dafür, dass eine gültige Projekt-ID vorhanden ist, die verwendet werden kann.
public async Task<string> EnsureProjektIdAsync(string? vorgeschlageneProjektId, CancellationToken
cancellationTokens = default)
{
    // Wenn bereits eine nicht-leere Projekt-ID vorgeschlagen wurde und dieses Projekt existiert,
    // wird sie unverändert weiterverwendet.
    if (!string.IsNullOrEmpty(vorgeschlageneProjektId)
        && await _projektRepository.ExistsAsync(vorgeschlageneProjektId, cancellationTokens))
    {
        return vorgeschlageneProjektId;
    }

    // Erzeugt andernfalls eine neue GUID als eindeutige Projekt-ID.
    var neueProjektId = Guid.NewGuid().ToString();

    // Legt zu dieser ID sofort ein neues leeres Projekt in der Persistenz an.
    await _projektRepository.CreateAsync(neueProjektId, cancellationTokens);

    // Gibt die neue, garantiert vorhandene Projekt-ID zurück.
    return neueProjektId;
}

// Prüft, ob ein Nutzer im Projekt existiert.
public async Task<bool> NutzerExistiertAsync(string projektId, string nutzerId, CancellationToken
cancellationTokens = default)
{
    // Verwendet die detailliertere Nutzer-Lademethode, um doppelte Logik zu vermeiden.
    var nutzer = await HoleNutzerAsync(projektId, nutzerId, cancellationTokens);

    // Existenz ist genau dann gegeben, wenn ein Nutzerobjekt gefunden wurde.
    return nutzer is not null;
}

// Lädt einen einzelnen Nutzer anhand seiner ID.
public async Task<NutzerDaten?> HoleNutzerAsync(string projektId, string nutzerId, CancellationToken
cancellationTokens = default)
{
    // Liest zunächst den kompletten Projektzustand.
    var projekt = await _projektRepository.ReadAsync(projektId, cancellationTokens);

    // Sucht anschließend den ersten passenden Nutzer in der Nutzersammlung.
    return projekt.Nutzer.FirstOrDefault(n => n.NutzerId == nutzerId);
}

// Erzeugt einen neuen Nutzer mit validiertem Namen.
public async Task<NutzerErstellungErgebnis> ErstelleNutzerAsync(string projektId, string name,
CancellationTokens cancellationTokens = default)
{
    // Entfernt führende und nachgestellte Leerzeichen, damit die Validierung konsistent ist.
    name = name.Trim();

    // Leere Namen sind fachlich unzulässig.
    if (string.IsNullOrEmpty(name))
    {
        return new NutzerErstellungErgebnis
        {
            // Kennzeichnet den Vorgang als fehlgeschlagen.
            Erfolg = false,

            // Liefert eine direkte, UI-taugliche Fehlermeldung zurück.
        };
    }
}

```

```

        Fehlermeldung = "Der Name darf nicht leer sein."
    };
}

// Begrenzt den Namen auf maximal 10 Zeichen.
if (name.Length > 10)
{
    name = name[..10];
}

// Führt die Änderung transaktional innerhalb des Repository-Updates aus.
return await _projektRepository.UpdateAsync(
    projektId,
    projekt =>
    {
        // Erstellt eine alphabetisch sortierte Liste der vorhandenen Namen,
        // die im Fehlerfall an die Oberfläche zurückgegeben werden kann.
        var vorhandeneNamen = projekt.Nutzers.Select(n => n.NutzerName).OrderBy(n => n).ToArray();

        // Prüft, ob der gewünschte Name bereits vergeben ist, ohne auf Groß-/Kleinschreibung zu
achten.
        if (projekt.Nutzers.Any(n => string.Equals(n.NutzerName, name,
StringComparison.OrdinalIgnoreCase)))
        {
            return new NutzerErstellungErgebnis
            {
                Erfolg = false,
                Fehlermeldung = "Der Name wird schon genutzt.",
                // Übergibt zusätzlich die bereits existierenden Namen als Hilfestellung.
                VorhandeneNamen = vorhandeneNamen
            };
        }

        // Erzeugt eine eindeutige ID für den neuen Nutzer.
        var neueNutzerId = Guid.NewGuid().ToString();

        // Fügt den neuen Nutzer in die Projektliste ein.
        projekt.Nutzers.Add(new NutzerDaten
        {
            NutzerId = neueNutzerId,
            NutzerName = name,
            // Neue Nutzer starten als "nicht zufrieden".
            Zufrieden = false
        });

        // Meldet die erfolgreiche Anlage samt neuer ID zurück.
        return new NutzerErstellungErgebnis
        {
            Erfolg = true,
            NutzerId = neueNutzerId
        };
    },
    cancellationTokens);
}

// Berechnet das verbleibende Budget eines Nutzers.
public async Task<int> BerechneGuthabenAsync(string projektId, string nutzerId, CancellationToken
cancellationTokens = default)
{
    // Liest den aktuellen Projektstand aus der Persistenz.
    var projekt = await _projektRepository.ReadAsync(projektId, cancellationTokens);

    // Summiert alle Gebote der Objekte, die aktuell diesem Nutzer gehören.
    var summeGebote = projekt.Objekte
        .Where(o => o.BesitzerId == nutzerId)
        .Sum(o => o.ObjektGebot);

    // Verbleibendes Guthaben ist Startbudget minus gebundene Gebotssumme.
    return StartGuthaben - summeGebote;
}

```

```

// Stellt die Objektliste für die Oberfläche zusammen und sortiert sie bei Bedarf.
public async Task<IReadOnlyList<ObjektListeEintrag>> LadeObjektlisteAsync(
    string projektId,
    string nutzerId,
    string? sortExpression,
    string? sortDirection,
    CancellationToken cancellationToken = default)
{
    // Lädt alle notwendigen Projektdaten in einem Schritt.
    var projekt = await _projektRepository.ReadAsync(projektId, cancellationToken);

    // Baut ein Dictionary von Nutzer-ID auf Nutzernamen auf,
    // damit Besitzernamen später effizient nachgeschlagen werden können.
    var nutzerNamen = projekt.Nutzer
        .GroupBy(n => n.NutzerId)
        .ToDictionary(g => g.Key, g => g.First().NutzerName);

    // Transformiert die internen Objektdaten in Anzeigeeinträge für die UI.
    var basisListe = projekt.Objekte
        .Select(o => new ObjektListeEintrag
        {
            Id = o.ObjektId,
            // Leitet den Bildpfad aus der Objekt-ID ab.
            Bild = $"bilder/{o.ObjektId}.jpg",
            Objekt = o.ObjektName,
            Gebot = o.ObjektGebot,
            // Verwendet "?" als Fallback, falls kein Besitzernamen aufgelöst werden kann.
            Besitzer = nutzerNamen.TryGetValue(o.BesitzerId, out var besitzerName) ? besitzerName : "?"
        })
        .ToList();

    // Die fertige Liste wird immer in eine geordnete Form überführt.
    IOrderedEnumerable<ObjektListeEintrag> sortiert;

    // Interpretiert die gewünschte Sortierrichtung robust gegenüber Groß-/Kleinschreibung.
    var richtungAbsteigend = string.Equals(sortDirection, "DESC", StringComparison.OrdinalIgnoreCase);

    // Sortiert explizit nach Objektname.
    if (string.Equals(sortExpression, "Objekt", StringComparison.OrdinalIgnoreCase))
    {
        sortiert = richtungAbsteigend
            ? basisListe.OrderByDescending(x => x.Objekt)
            : basisListe.OrderBy(x => x.Objekt);
    }
    // Sortiert explizit nach Gebotswert.
    else if (string.Equals(sortExpression, "Gebot", StringComparison.OrdinalIgnoreCase))
    {
        sortiert = richtungAbsteigend
            ? basisListe.OrderByDescending(x => x.Gebot)
            : basisListe.OrderBy(x => x.Gebot);
    }
    // Sortiert explizit nach Besitzernamen.
    else if (string.Equals(sortExpression, "Besitzer", StringComparison.OrdinalIgnoreCase))
    {
        sortiert = richtungAbsteigend
            ? basisListe.OrderByDescending(x => x.Besitzer)
            : basisListe.OrderBy(x => x.Besitzer);
    }
    else
    {
        // Standardsortierung:
        // Eigene Objekte zuerst, danach nach Besitzernamen und Objektname.
        sortiert = basisListe
            .OrderBy(x => x.Besitzer == HoleNutzerName(nutzerNamen, nutzerId) ? 0 : 1)
            .ThenBy(x => x.Besitzer)
            .ThenBy(x => x.Objekt);
    }

    // Materialisiert die sortierte Folge als Liste und gibt sie schreibgeschützt zurück.
    return sortiert.ToList();
}

```

```

// Liefert Detaildaten für genau ein Objekt und berechnet dabei mögliche Gebotsgrenzen.
public async Task<ObjektDetailDaten?> HoleObjektDetailAsync(
    string projektId,
    string nutzerId,
    string objektId,
    CancellationToken cancellationToken = default)
{
    // Liest das Projekt, in dem nach dem Objekt gesucht werden soll.
    var projekt = await _projektRepository.ReadAsync(projektId, cancellationToken);

    // Sucht das gewünschte Objekt über seine eindeutige Objekt-ID.
    var objekt = projekt.Objekte.FirstOrDefault(o => o.ObjektId == objektId);

    // Wenn das Objekt nicht existiert, gibt die Methode null zurück.
    if (objekt is null)
    {
        return null;
    }

    // Löst den Namen des aktuellen Besitzers auf; bei fehlender Zuordnung wird "?" verwendet.
    var besitzerName = projekt.Nutzer.FirstOrDefault(n => n.NutzerId ==
objekt.BesitzerId)?.NutzerName ?? "?";

    // Berechnet das aktuelle freie Budget des anfragenden Nutzers.
    var guthaben = StartGuthaben - projekt.Objekte
        .Where(o => o.BesitzerId == nutzerId)
        .Sum(o => o.ObjektGebot);

    // Diese Werte werden abhängig von Besitzverhältnis und Guthaben bestimmt.
    int minGebot;
    int maxGebot;
    bool gebotMoeglich;
    string gebotHinweis;

    // Wer bereits Besitzer ist, darf das Gebot bis auf 0 senken und bekommt sein bisher gebundenes
Gebot zurückgerechnet.
    if (objekt.BesitzerId == nutzerId)
    {
        minGebot = 0;
        maxGebot = guthaben + objekt.ObjektGebot;
        gebotMoeglich = true;
        gebotHinweis = $"{minGebot} ... {maxGebot}";
    }
    else
    {
        // Wer das Objekt übernehmen will, muss mindestens 1 mehr als das aktuelle Gebot bieten.
        minGebot = objekt.ObjektGebot + 1;
        maxGebot = guthaben;
        gebotMoeglich = maxGebot >= minGebot;
        gebotHinweis = gebotMoeglich ? $"{minGebot} ... {maxGebot}" : "Nicht genug Guthaben";
    }

    // Baut das Antwortobjekt für die Detailansicht zusammen.
    return new ObjektDetailDaten
    {
        ObjektId = objekt.ObjektId,
        ObjektName = objekt.ObjektName,
        BesitzerId = objekt.BesitzerId,
        BesitzerName = besitzerName,
        ObjektGebot = objekt.ObjektGebot,
        MinGebot = minGebot,
        MaxGebot = maxGebot,
        GebotMoeglich = gebotMoeglich,
        GebotHinweis = gebotHinweis
    };
}

// Lädt alle Nutzer eines Projekts mitsamt Zufriedenheitsstatus für eine Übersicht.
public async Task<IReadOnlyList<ZufriedenheitEintrag>> LadeZufriedenheitenAsync(
    string projektId,

```

```

        CancellationToken cancellationToken = default)
    {
        // Liest das Projekt vollständig ein.
        var projekt = await _projektRepository.ReadAsync(projektId, cancellationToken);

        // Sortiert die Nutzer alphabetisch und projiziert sie auf das leichtere Anzeigeformat.
        return projekt.Nutzer
            .OrderBy(n => n.NutzerName)
            .Select(n => new ZufriedenheitEintrag
            {
                NutzerName = n.NutzerName,
                Zufrieden = n.Zufrieden
            })
            .ToList();
    }

    // Fügt ein neues Objekt in ein Projekt ein, sofern Nutzer und Objekt-ID gültig sind.
    public async Task<bool> FuegeObjektHinzuAsync(
        string projektId,
        string nutzerId,
        string objektId,
        CancellationToken cancellationToken = default)
    {
        // Ohne verwertbare Objekt-ID wird kein Objekt angelegt.
        if (string.IsNullOrEmpty(objektId))
        {
            return false;
        }

        // Führt die Objektanlage innerhalb einer Repository-Aktualisierung aus.
        return await _projektRepository.UpdateAsync(
            projektId,
            projekt =>
            {
                // Ein Objekt kann nur angelegt werden, wenn der referenzierte Nutzer existiert.
                if (!projekt.Nutzer.Any(n => n.NutzerId == nutzerId))
                {
                    return false;
                }

                // Wenn das Objekt schon vorhanden ist, wird dies als erfolgreicher Zustand behandelt.
                if (projekt.Objekte.Any(o => o.ObjektId == objektId))
                {
                    return true;
                }

                // Erstellt ein Platzhalterobjekt mit Standardname, Besitzer und Gebot 0.
                projekt.Objekte.Add(new ObjektDaten
                {
                    ObjektId = objektId,
                    ObjektName = "...",
                    BesitzerId = nutzerId,
                    ObjektGebot = 0
                });

                return true;
            },
            cancellationToken);
    }

    // Speichert Änderungen an einem Objekt und validiert dabei die Gebotsgrenzen.
    public async Task<ObjektSpeichernErgebnis> SpeichereObjektAsync(
        string projektId,
        string nutzerId,
        string objektId,
        string objektName,
        int objektGebot,
        CancellationToken cancellationToken = default)
    {
        // Delegiert die komplette Lese-/Änderungs-/Speicherlogik an das Repository.
        return await _projektRepository.UpdateAsync(

```

```

projektId,
projekt =>
{
    // Sucht das zu speichernde Objekt.
    var objekt = projekt.Objekte.FirstOrDefault(o => o.ObjektId == objektId);
    if (objekt is null)
    {
        return new ObjektSpeichernErgebnis { Erfolg = false, Fehlermeldung = "Objekt nicht
gefunden." };
    }

    // Merkt sich den bisherigen Zustand, um Gebotsregeln und Besitzerwechsel zu berechnen.
    var aktuellesGebot = objekt.ObjektGebot;
    var aktuellerBesitzer = objekt.BesitzerId;

    // Berechnet das aktuell verfügbare Guthaben des speichernden Nutzers.
    var guthaben = StartGuthaben - projekt.Objekte
        .Where(o => o.BesitzerId == nutzerId)
        .Sum(o => o.ObjektGebot);

    // Untergrenze und Obergrenze des erlaubten Gebots werden im Anschluss gesetzt.
    int minGebot;
    int maxGebot;

    // Der aktuelle Besitzer darf sein Gebot auch absenken.
    if (aktuellerBesitzer == nutzerId)
    {
        minGebot = 0;
        maxGebot = guthaben + aktuellesGebot;
    }
    else
    {
        // Ein neuer Besitzer muss mindestens das aktuelle Gebot überbieten.
        minGebot = aktuellesGebot + 1;
        maxGebot = guthaben;
    }

    // Lehnt unzulässige Gebote mit konkreter Rückmeldung ab.
    if (objektGebot < minGebot || objektGebot > maxGebot)
    {
        return new ObjektSpeichernErgebnis
        {
            Erfolg = false,
            MinGebot = minGebot,
            MaxGebot = maxGebot,
            Fehlermeldung = minGebot > maxGebot ? "Nicht genug Guthaben" : $"{minGebot} ...
{maxGebot}"
        };
    }

    // Wenn das Objekt den Besitzer wechselt, wird der bisherige Besitzer wieder auf "nicht
zufrieden" gesetzt.
    if (aktuellerBesitzer != nutzerId)
    {
        var alterBesitzer = projekt.Nutzer.FirstOrDefault(n => n.NutzerId ==
aktuellerBesitzer);
        if (alterBesitzer is not null)
        {
            alterBesitzer.Zufrieden = false;
        }
    }

    // Übernimmt den neuen Besitzer in das Objekt.
    objekt.BesitzerId = nutzerId;

    // Speichert einen bereinigten Namen; leere Namen werden durch "..." ersetzt.
    objekt.ObjektName = string.IsNullOrEmpty(objektName) ? "..." : objektName.Trim();

    // Speichert das validierte Gebot.
    objekt.ObjektGebot = objektGebot;

```

```

        // Meldet den erfolgreichen Speichervorgang zurück.
        return new ObjektSpeichernErgebnis { Erfolg = true };
    },
    cancellationToken);
}

// Entfernt ein Objekt aus dem Projekt, wenn es existiert.
public async Task<bool> LoescheObjektAsync(string projektId, string objektId, CancellationToken
cancellationToken = default)
{
    // Führt die Löschung innerhalb der Repository-Aktualisierung aus.
    return await _projektRepository.UpdateAsync(
        projektId,
        projekt =>
        {
            // Sucht das zu löschende Objekt.
            var objekt = projekt.Objekte.FirstOrDefault(o => o.ObjektId == objektId);
            if (objekt is null)
            {
                return false;
            }

            // Entfernt das Objekt aus der Sammlung.
            projekt.Objekte.Remove(objekt);

            // Signalisiert erfolgreiche Löschung.
            return true;
        },
        cancellationToken);
}

// Setzt den Zufriedenheitsstatus eines Nutzers.
public async Task<bool> SetzeZufriedenheitAsync(
    string projektId,
    string nutzerId,
    bool zufrieden,
    CancellationToken cancellationToken = default)
{
    // Führt die Statusänderung atomar im Repository aus.
    return await _projektRepository.UpdateAsync(
        projektId,
        projekt =>
        {
            // Sucht den betroffenen Nutzer.
            var nutzer = projekt.Nutzer.FirstOrDefault(n => n.NutzerId == nutzerId);
            if (nutzer is null)
            {
                return false;
            }

            // Übernimmt den neuen booleschen Zufriedenheitswert direkt.
            nutzer.Zufrieden = zufrieden;

            // Meldet, dass der Status erfolgreich aktualisiert wurde.
            return true;
        },
        cancellationToken);
}

// Hilfsmethode zum sicheren Nachschlagen eines Nutzernamens per ID.
private static string HoleNutzerName(IReadOnlyDictionary<string, string> nutzerNamen, string nutzerId)
{
    // Gibt den Namen zurück, wenn die ID im Dictionary vorhanden ist; sonst einen leeren String.
    return nutzerNamen.TryGetValue(nutzerId, out var name) ? name : string.Empty;
}
}

```

4.7 Program.cs

```
/*
 * Diese Datei ist der Einstiegspunkt der gesamten ASP.NET-Core-Anwendung "Partitio".
 * Hier wird festgelegt, welche Infrastruktur die Anwendung beim Start benötigt, wie
 * ihre internen Dienste miteinander verbunden werden, auf welche Weise HTTP-Anfragen
 * durch die Middleware-Pipeline laufen und welche URLs schließlich welche Logik
 * ausführen. Im konkreten Kontext dieser Anwendung bedeutet das: Razor Pages werden
 * aktiviert, projektspezifische Services für den Zugriff auf XML-Daten und fachliche
 * Operationen werden registriert, ein physisches Bilderverzeichnis wird für Uploads
 * und spätere Auslieferung vorbereitet, ein eigener API-Endpunkt für Bild-Uploads
 * wird bereitgestellt und ältere oder alternative Einstiegspfade werden auf die
 * zentrale Partitio-Route umgeleitet. Die Datei enthält damit nicht nur "Startcode",
 * sondern die zentrale Verdrahtung zwischen Webserver, Fachlogik, Dateisystem und
 * öffentlicher HTTP-Oberfläche der Anwendung.
 */

// Importiert Typen für FileProvider, damit ein physisches Verzeichnis
// später als Quelle für statische Dateien unter einer URL eingebunden werden kann.
using Microsoft.Extensions.FileProviders;

// Importiert die projektspezifischen Service- und Repository-Schnittstellen
// sowie ihre Implementierungen aus dem Namespace der Anwendung.
using Partitio.Services;

// Erstellt den zentralen WebApplicationBuilder.
// Er sammelt Konfiguration, Logging und alle Services, die später per Dependency Injection
// in Razor Pages, Endpunkte und andere Komponenten injiziert werden.
var builder = WebApplication.CreateBuilder(args);

// Registriert die Razor-Pages-Infrastruktur, damit .cshtml-Seiten und ihre PageModels
// über das Routing erreichbar sind.
builder.Services.AddRazorPages();

// Das Repository kapselt den Zugriff auf die Projektdaten.
// Als Singleton wird genau eine Instanz für die gesamte Lebensdauer der Anwendung verwendet,
// was sinnvoll ist, wenn die Implementierung selbst keinen Request-spezifischen Zustand hält.
builder.Services.AddSingleton<IProjektRepository, XmlProjektRepository>();

// Der Partitio-Service enthält die Anwendungslogik rund um Projekte/Objekte.
// Scoped bedeutet: pro HTTP-Request wird eine eigene Instanz erzeugt.
builder.Services.AddScoped<IPartitioService, PartitioService>();

// Baut die finale HTTP-Anwendung aus der vorher konfigurierten Builder-Struktur.
var app = builder.Build();

// In Nicht-Entwicklungsumgebungen werden produktionsnahe Fehlerbehandlung und HSTS aktiviert.
// So werden interne Fehler nicht direkt ausgegeben und Browser merken sich die HTTPS-Pflicht.
if (!app.Environment.IsDevelopment())
{
    // Leitet unbehandelte Fehler auf die Standard-Fehlerseite um.
    app.UseExceptionHandler("/Error");

    // Erzwingt per HTTP-Header, dass Clients die Seite für einen Zeitraum nur noch per HTTPS aufrufen.
    app.UseHsts();
}

// Leitet eingehende HTTP-Anfragen automatisch auf HTTPS um.
app.UseHttpsRedirection();

// Aktiviert das Routing-System. Erst danach können Endpunkte wie Razor Pages oder MapGet/MapPost
// sinnvoll aufgelöst werden.
app.UseRouting();

// Aktiviert die Autorisierungsmiddleware.
// Auch wenn hier aktuell keine expliziten Policies oder [Authorize]-Attribute sichtbar sind,
// ist die Middleware damit grundsätzlich verfügbar.
app.UseAuthorization();

// Standardpfad für hochgeladene Bilder innerhalb des Projektverzeichnisses.
var bilderPfad = Path.Combine(app.Environment.ContentRootPath, "bilder");
```

```

// Fallback auf einen älteren Ablageort: Falls das Projekt z. B. umstrukturiert wurde,
// können vorhandene Bilder weiterhin aus einem Verzeichnis neben dem Content Root geladen werden.
var legacyBilderPfad = Path.Combine(Directory.GetParent(app.Environment.ContentRootPath)?.FullName ??
string.Empty, "bilder");

// Wenn der neue Pfad noch nicht existiert, aber der alte vorhanden ist, wird der Altpfad weiterverwendet.
if (!Directory.Exists(bilderPfad) && Directory.Exists(legacyBilderPfad))
{
    // Überschreibt den Standardpfad mit dem Legacy-Verzeichnis,
    // damit bestehende Bilddaten nicht migriert werden müssen.
    bilderPfad = legacyBilderPfad;
}

// Stellt sicher, dass das Zielverzeichnis für Bilddateien tatsächlich existiert,
// bevor Uploads gespeichert oder statische Dateien daraus ausgeliefert werden.
Directory.CreateDirectory(bilderPfad);

// Aktiviert das Ausliefern statischer Dateien aus wwwroot bzw. der Standardkonfiguration.
app.UseStaticFiles();

// Bindet zusätzlich das Bilderverzeichnis unter der URL /bilder ein.
// Dadurch können hochgeladene Dateien direkt per HTTP abgerufen werden.
app.UseStaticFiles(new StaticFileOptions
{
    // Verweist auf das physische Verzeichnis im Dateisystem,
    // aus dem die Bilder tatsächlich gelesen werden.
    FileProvider = new PhysicalFileProvider(bilderPfad),

    // Legt fest, unter welchem URL-Präfix die Dateien öffentlich erreichbar sind.
    RequestPath = "/bilder"
});

// Minimal-API-Endpunkt zum Empfangen eines Bild-Uploads.
// Die benötigten Dienste werden direkt als Parameter injiziert.
async Task<IResult> BildEmpfangen(
    // Repräsentiert die eingehende HTTP-Anfrage mit Headern, Formular und Dateiinhalt.
    HttpRequest request,

    // Fachservice für das Anlegen bzw. Verknüpfen eines Bild-Objekts im Anwendungskontext.
    IPartitioService partitioService,

    // Repository zur Prüfung, ob das referenzierte Projekt überhaupt existiert.
    IProjektRepository projektRepository)
{
    // Akzeptiert nur Formularanfragen, typischerweise multipart/form-data.
    // Andere Inhaltstypen würden keine Datei im erwarteten Format liefern.
    if (!request.HasFormContentType)
    {
        return Results.Json(new { status = "Fehler", message = "Ungültiger Inhaltstyp." }, statusCode:
400);
    }

    // Liest das komplette Formular inklusive hochgeladener Dateien ein.
    var form = await request.ReadFormAsync();

    // Erwartet die Bilddatei im Feld "bild".
    var file = form.Files["bild"];

    // Lehnt die Anfrage ab, wenn keine Datei vorhanden ist oder sie leer ist.
    if (file is null || file.Length == 0)
    {
        return Results.Json(new { status = "Fehler", message = "Keine Datei hochgeladen." }, statusCode:
400);
    }

    // Diese beiden Formularfelder ordnen das Bild einem Projekt und einem Nutzer zu.
    var projektId = form["projekt_id"].ToString();

    // Liest die Nutzer-ID aus dem Formular aus, um das Bild fachlich einem Benutzer zuzuordnen.
    var nutzerId = form["nutzer_id"].ToString();

```

```

// Beide IDs sind Pflichtfelder, weil ohne sie keine fachliche Zuordnung möglich ist.
if (string.IsNullOrEmpty(projektId) || string.IsNullOrEmpty(nutzerId))
{
    return Results.Json(new { status = "Fehler", message = "projekt_id oder nutzer_id fehlt." },
statusCode: 400);
}

// Verhindert Uploads für nicht existierende Projekte.
if (!await projektRepository.ExistsAsync(projektId))
{
    return Results.Json(new { status = "Fehler", message = "Projekt nicht gefunden." }, statusCode:
404);
}

// Optional kann eine explizite Bild-ID mitgegeben werden.
// Fehlt sie, wird der Dateiname ohne Endung als ID verwendet.
var bildId = form["bild_id"].ToString();
if (string.IsNullOrEmpty(bildId))
{
    // Entfernt die Dateiendung des Uploadnamens und verwendet den verbleibenden Namen als interne
Bild-ID.
    bildId = Path.GetFileNameWithoutExtension(file.FileName);
}

// Legt zuerst das fachliche Objekt an.
// Schlägt dieser Schritt fehl, wird die Datei gar nicht erst gespeichert.
if (!await partitioService.FuegeObjektHinzuAsync(projektId, nutzerId, bildId))
{
    return Results.Json(new { status = "Fehler", message = "Objekt konnte nicht angelegt werden." },
statusCode: 400);
}

// Die Datei wird unabhängig vom ursprünglichen Uploadnamen immer als .jpg gespeichert.
// Das vereinfacht spätere Referenzen, setzt aber voraus, dass das System dieses Format erwartet.
var dateiname = $"{bildId}.jpg";

// Kombiniert Verzeichnis und Dateinamen zu einem vollständigen Zielpfad im Dateisystem.
var zielpfad = Path.Combine(bilderPfad, dateiname);

// Erstellt die Zieldatei und kopiert den Upload-Stream hinein.
// "await using" sorgt dafür, dass der Stream auch bei Fehlern asynchron sauber freigegeben wird.
await using var stream = File.Create(zielpfad);

// Kopiert den Inhalt der hochgeladenen Datei vollständig in die neu erstellte Zieldatei.
await file.CopyToAsync(stream);

// Einheitliches JSON-Ergebnisformat für den aufrufenden Client.
return Results.Json(new { status = "Erfolg" });
}

// Registriert den Upload-Endpunkt.
app.MapPost("/api/bild-upload", BildEmpfangen);

// Registriert alle Razor Pages aus dem Projekt.
app.MapRazorPages();

// Leitet Anfragen an die Root-URL auf /partitio weiter und übernimmt vorhandene Query-Parameter.
// Bestehende Links auf "/" bleiben dadurch funktionsfähig.
app.MapGet("/", (HttpContext context) =>
{
    // Extrahiert die Query-String-Komponente inklusive führendem "?", falls vorhanden.
    var query = context.Request.QueryString.HasValue ? context.Request.QueryString.Value : string.Empty;

    // Führt eine HTTP-Weiterleitung zur eigentlichen Partitio-Route aus.
    return Results.Redirect($" /partitio{query}");
});

// Kompatibilitätsroute für ältere Links auf die frühere ASPX-Adresse.
// Auch hier werden Query-Parameter unverändert an die neue Zielroute angehängt.
app.MapGet("/partitio.aspx", (HttpContext context) =>

```

```
{
    // Übernimmt auch bei Altlinks vorhandene Parameter unverändert in die neue Zieladresse.
    var query = context.Request.QueryString.HasValue ? context.Request.QueryString.Value : string.Empty;

    // Leitet den alten Pfad transparent auf die aktuelle Route um.
    return Results.Redirect($"{"/partitio{query}"}");
});

// Startet die Anwendung und beginnt damit, HTTP-Anfragen entgegenzunehmen.
app.Run();
```

4.8 _ViewImports.cshtml

```
<!--
Diese Datei definiert gemeinsame Razor-Importe für alle Views und Pages im
aktuellen Verzeichnisbaum. Alles, was hier eingetragen wird, steht den
untergeordneten .cshtml-Dateien automatisch zur Verfügung, ohne dass es
jeweils erneut pro Datei geschrieben werden muss. Dadurch wird Razor-Markup
kompakter und konsistenter. In diesem Projekt werden hier insbesondere der
Anwendungs-Namespace, der Razor-Namespace der Pages sowie die MVC-TagHelper
global eingebunden.
-->
<!-- Importiert den Basis-Namespace der Anwendung, damit Typen daraus in Razor leichter referenzierbar
sind. -->
@using Partitio
<!-- Legt den Standard-Namespace für Razor Pages in diesem Ordner fest. -->
@namespace Partitio.Pages
<!-- Aktiviert die eingebauten ASP.NET-Core-TagHelper, etwa für Formulare, Links und Labels. -->
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

4.9 _ViewStart.cshtml

```
<!--
Diese Datei wird vor dem Rendern der Razor-Seiten in diesem Verzeichnisbaum
ausgeführt und dient dazu, gemeinsame Anzeigeeinstellungen zentral zu setzen.
Hier wird insbesondere festgelegt, welches Layout standardmäßig verwendet
werden soll. Dadurch müssen einzelne Seiten das Layout nicht jeweils selbst
definieren, solange sie den Standard verwenden.
-->
@{
    <!-- Weist allen untergeordneten Seiten standardmäßig das Layout "_Layout" zu. -->
    Layout = "_Layout";
}
```

4.10 Partitio.cshtml.cs

```
/*
 * Diese Datei enthält das zentrale PageModel der eigentlichen Partitio-Anwendung.
 * Hier laufen die wesentlichen Interaktionen der Benutzeroberfläche zusammen:
 * Projekt und Nutzer werden aus Query-Parametern oder Cookies aufgelöst, Objekte
 * und Zufriedenheitszustände werden geladen, Detailansichten vorbereitet, neue
 * Nutzer angelegt, Objekte gespeichert oder gelöscht und Statusänderungen des
 * Nutzers verarbeitet. Das PageModel bildet damit die Brücke zwischen HTTP/Razor
 * Pages auf der einen Seite und der eigentlichen Fachlogik im PartitioService auf
 * der anderen Seite. Gleichzeitig koordiniert es Redirects, URL-Konsistenz und
 * die Persistenz bestimmter Kontextdaten in Cookies, damit die Anwendung aus Sicht
 * des Nutzers stabil, bookmarkbar und zustandsbewusst funktioniert.
 */

// Importiert MVC-Typen wie IActionResult, BindProperty und Redirect-Helfer.
using Microsoft.AspNetCore.Mvc;

// Importiert die PageModel-Basisklasse für Razor Pages.
using Microsoft.AspNetCore.Mvc.RazorPages;

// Importiert die Modelltypen, die in Properties und Rückgaben für die View verwendet werden.
using Partitio.Models;
```

```

// Importiert die Service-Schnittstellen mit der zentralen Fachlogik.
using Partitio.Services;

// Namespace für die Razor-Pages-Modelklassen der Anwendung.
namespace Partitio.Pages;

// PageModel der Hauptanwendungsseite "Partitio".
public class PartitioModel : PageModel
{
    // Fachservice für Projekt-, Nutzer- und Objektlogik.
    private readonly IPartitioService _partitioService;

    // Hosting-Umgebung, hier vor allem für Dateisystemzugriffe auf Bilder relevant.
    private readonly IWebHostEnvironment _environment;

    // Konstruktor für Dependency Injection.
    public PartitioModel(IPartitioService partitioService, IWebHostEnvironment environment)
    {
        // Speichert den injizierten Fachservice für spätere Handler-Aufrufe.
        _partitioService = partitioService;

        // Speichert die Hosting-Umgebung, um Pfade relativ zum Content Root bilden zu können.
        _environment = environment;
    }

    // Bindet den Query-Parameter "projekt" an diese Property und erlaubt das auch bei GET-Anfragen.
    [BindProperty(SupportsGet = true, Name = "projekt")]
    public string? ProjektId { get; set; }

    // Bindet den Query-Parameter "nutzer" für den aktuellen Nutzerkontext.
    [BindProperty(SupportsGet = true, Name = "nutzer")]
    public string? NutzerId { get; set; }

    // Bindet die gewünschte Sortierspalte aus dem Query-String.
    [BindProperty(SupportsGet = true, Name = "sort")]
    public string? SortExpression { get; set; }

    // Bindet die gewünschte Sortierrichtung aus dem Query-String.
    [BindProperty(SupportsGet = true, Name = "dir")]
    public string? SortDirection { get; set; }

    // Bindet die ID des aktuell in der Detailansicht geöffneten Objekts.
    [BindProperty(SupportsGet = true, Name = "detail")]
    public string? DetailObjektId { get; set; }

    // Bindet die ID des Objekts, für das gerade eine Löschbestätigung aktiv ist.
    [BindProperty(SupportsGet = true, Name = "loeschen")]
    public string? LoeschObjektId { get; set; }

    // Bindet den vom Benutzer eingegebenen Namen für die Erstellung eines neuen Nutzers.
    [BindProperty]
    public string NeuerName { get; set; } = string.Empty;

    // Bindet den Namen aus dem Objekt-Detailformular.
    [BindProperty]
    public string DetailName { get; set; } = string.Empty;

    // Bindet das Gebot aus dem Objekt-Detailformular.
    [BindProperty]
    public int DetailGebot { get; set; }

    // Enthält den Anzeigenamen des aktuell angemeldeten Nutzers.
    public string NutzerName { get; private set; } = string.Empty;

    // Enthält das freie Guthaben des aktuellen Nutzers.
    public int Guthaben { get; private set; } = 1000;

    // Steuert, ob die Seite einen Anmeldungs-/Nutzeranlagezustand anzeigen soll.
    public bool AnmeldungErforderlich { get; private set; } = true;
}

```

```

// Optional anzuzeigende Fehlermeldung für die Oberfläche.
public string? FehlerMeldung { get; private set; }

// Optional anzuzeigende Informationsmeldung für die Oberfläche.
public string? InfoMeldung { get; private set; }

// Liste bereits vorhandener Namen, z. B. als Rückmeldung bei doppelter Nutzeranlage.
public IReadOnlyList<string> VorhandeneNamen { get; private set; } = Array.Empty<string>();

// Die aktuell für die Seite geladene Objektliste.
public IReadOnlyList<ObjektListeEintrag> Objekte { get; private set; } =
Array.Empty<ObjektListeEintrag>();

// Übersicht aller Zufriedenheitszustände im Projekt.
public IReadOnlyList<ZufriedenheitEintrag> Zufriedenheiten { get; private set; } =
Array.Empty<ZufriedenheitEintrag>();

// Detaildaten des aktuell ausgewählten Objekts, falls eine Detailansicht offen ist.
public ObjektDetailDaten? Detail { get; private set; }

// Anzeigename des Objekts, das gerade zur Löschung bestätigt werden soll.
public string? LoeschObjektName { get; private set; }

// Baut einen absoluten Lesezeichen-Link für denselben Nutzer- und Projektkontext.
// Nur wenn Projekt und Nutzer vollständig bekannt sind und keine Anmeldung erforderlich ist,
// ergibt dieser Link fachlich Sinn.
public string? LesezeichenLink =>
    AnmeldungErforderlich || string.IsNullOrEmpty(ProjektId) ||
string.IsNullOrEmpty(NutzerId)
    ? null
    : AbsoluterLink($" /partitio?projekt={Uri.EscapeDataString(ProjektId)}
&nutzer={Uri.EscapeDataString(NutzerId)}");

// Baut einen absoluten Einladungslink für dasselbe Projekt.
// Der Nutzerparameter wird hier bewusst auf einen Platzhalterwert gesetzt,
// damit ein eingeladener Nutzer einen eigenen Nutzerkontext erzeugen kann.
public string? EinladungLink =>
    string.IsNullOrEmpty(ProjektId)
    ? null
    : AbsoluterLink($" /partitio?projekt={Uri.EscapeDataString(ProjektId)}&nutzer=42");

// Hilfseigenschaft für die View: nächste Sortierrichtung für die Objektnamen-Spalte.
public string NextSortDirectionObjekt => NextSortDirection("Objekt");

// Hilfseigenschaft für die View: nächste Sortierrichtung für die Gebots-Spalte.
public string NextSortDirectionGebot => NextSortDirection("Gebot");

// Hilfseigenschaft für die View: nächste Sortierrichtung für die Besitzer-Spalte.
public string NextSortDirectionBesitzer => NextSortDirection("Besitzer");

// GET-Handler der Hauptseite.
public async Task<IActionResult> OnGetAsync(CancellationTokens cancellationTokens)
{
    // Lädt den Seitentext bzw. führt falls nötig eine kanonisierende Weiterleitung aus.
    return await LadeOderRedirectAsync(cancellationTokens);
}

// POST-Handler zum Anlegen eines neuen Nutzers.
public async Task<IActionResult> OnPostNutzerAnlegenAsync(CancellationTokens cancellationTokens)
{
    // Stellt sicher, dass ein gültiges Projekt existiert, in dem der Nutzer angelegt werden kann.
    ProjektId = await _partitioService.EnsureProjektIdAsync(ProjektId, cancellationTokens);

    // Delegiert die eigentliche Nutzeranlage an den Fachservice.
    var ergebnis = await _partitioService.ErstelleNutzerAsync(ProjektId, NeuerName, cancellationTokens);

    // Wenn keine erfolgreiche Anlage stattgefunden hat, werden Fehlermeldungen aufbereitet
    // und die Seite im Registrierungsstatus neu geladen.
    if (!ergebnis.Erfolg || string.IsNullOrEmpty(ergebnis.NutzerId))
    {
        FehlerMeldung = ergebnis.Fehlermeldung;
    }
}

```

```

        VorhandeneNamen = ergebnis.VorhandeneNamen;
        AnmeldungErforderlich = true;
        return await LadeOderRedirectAsync(cancellationToken, ladeObjekte: false);
    }

    // Übernimmt bei Erfolg die neue Nutzer-ID in den aktuellen Seitenkontext.
    NutzerId = ergebnis.NutzerId;

    // Persistiert Projekt- und Nutzerkontext in Cookies.
    SetzeCookies(ProjektId, NutzerId);

    // Leitet auf die kanonische Seiten-URL um.
    return RedirectToSelf();
}

// POST-Handler zum Öffnen einer Objekt-Detailansicht.
public async Task<IActionResult> OnPostDetailAnzeigenAsync(string objektId, CancellationTokens cancellationToken)
{
    // Merkt sich, welches Objekt in der Detailansicht geöffnet werden soll.
    DetailObjektId = objektId;

    // Lädt die Seite erneut, damit die Detaildaten in die Anzeige einfließen.
    return await LadeOderRedirectAsync(cancellationToken);
}

// POST-Handler zum Speichern eines bearbeiteten Objekts.
public async Task<IActionResult> OnPostObjektSpeichernAsync(string objektId, CancellationTokens cancellationToken)
{
    // Bleibt nach dem Speichern zunächst im Kontext desselben Detailobjekts.
    DetailObjektId = objektId;

    // Eine evtl. offene Löschbestätigung wird beim Speichern verworfen.
    LoeschObjektId = null;

    // Lädt zunächst den Seitenkontext, ohne dabei Detailwerte wieder in das Formular zurückzuschreiben.
    var load = await LadeSeiteAsync(cancellationToken, uebernehmeDetailInForm: false);
    if (load is not null)
    {
        return load;
    }

    // Ohne gültigen Nutzer- und Projektkontext ist ein Speichern nicht zulässig.
    if (AnmeldungErforderlich || string.IsNullOrEmpty(ProjektId) ||
        string.IsNullOrEmpty(NutzerId))
    {
        return RedirectToSelf();
    }

    // Übergibt die Formulardaten an den Fachservice, der Gebotsregeln und Besitzwechsel validiert.
    var ergebnis = await _partitioService.SpeichereObjektAsync(
        ProjektId,
        NutzerId,
        objektId,
        DetailName,
        DetailGebot,
        cancellationToken);

    // Bei fachlichem Fehler wird die Meldung angezeigt und die Seite erneut geladen.
    if (!ergebnis.Erfolg)
    {
        FehlerMeldung = ergebnis.Fehlermeldung;
        return await LadeOderRedirectAsync(cancellationToken);
    }

    // Schließt die Detailansicht nach erfolgreichem Speichern.
    DetailObjektId = null;
    Detail = null;
}

```

```

        // Setzt eine positive Rückmeldung für die Oberfläche.
        InfoMeldung = "Objekt gespeichert.";
        return await LadeOderRedirectAsync(cancellationTokens);
    }

    // POST-Handler, der zunächst nur die Löschbestätigung für ein Objekt anzeigt.
    public async Task<IActionResult> OnPostObjektLoeschenAnfragenAsync(string objektId, CancellationToken
cancellationTokens)
    {
        // Öffnet das Objekt in der Detailansicht.
        DetailObjektId = objektId;

        // Markiert dasselbe Objekt zusätzlich als aktuell zu löschendes Objekt.
        LoeschObjektId = objektId;
        return await LadeOderRedirectAsync(cancellationTokens);
    }

    // POST-Handler zum Abbrechen einer offenen Löschbestätigung.
    public async Task<IActionResult> OnPostObjektLoeschenAbbrechenAsync(CancellationToken
cancellationTokens)
    {
        // Entfernt nur den Löschdialog-Zustand, ohne notwendigerweise die Detailansicht zu schließen.
        LoeschObjektId = null;
        return await LadeOderRedirectAsync(cancellationTokens);
    }

    // POST-Handler, der die Löschung eines Objekts endgültig bestätigt.
    public async Task<IActionResult> OnPostObjektLoeschenBestaetigenAsync(string objektId,
CancellationTokens cancellationTokens)
    {
        // Nach Bestätigung werden Detail- und Löschzustand zurückgesetzt.
        DetailObjektId = null;
        LoeschObjektId = null;

        // Lädt den restlichen Seitenkontext vor der eigentlichen Löschung.
        var load = await LadeSeiteAsync(cancellationTokens);
        if (load is not null)
        {
            return load;
        }

        // Ohne gültiges Projekt oder mit Anmeldebedarf wird zur Seite zurückgeleitet.
        if (AnmeldungErforderlich || string.IsNullOrEmpty(ProjektId))
        {
            return RedirectToSelf();
        }

        // Führt die fachliche Löschung des Objekts aus.
        var geloescht = await _partitioService.LoescheObjektAsync(ProjektId, objektId, cancellationTokens);
        if (geloescht)
        {
            // Ermittelt den zugehörigen Bildpfad des gelöschten Objekts im Standard-Bilderordner.
            var bildPfad = Path.Combine(_environment.ContentRootPath, "bilder", $"{objektId}.jpg");
            if (System.IO.File.Exists(bildPfad))
            {
                // Entfernt die Bilddatei physisch, damit kein verwaistes Bild zurückbleibt.
                System.IO.File.Delete(bildPfad);
            }

            InfoMeldung = "Objekt gelöscht.";
        }

        return await LadeOderRedirectAsync(cancellationTokens);
    }

    // POST-Handler zum Setzen des Status "zufrieden".
    public async Task<IActionResult> OnPostZufriedenAsync(CancellationToken cancellationTokens)
    {
        return await SetzeZufriedenheitAsync(true, cancellationTokens);
    }

```

```

// POST-Handler zum Setzen des Status "unzufrieden".
public async Task<IActionResult> OnPostUnzufriedenAsync(CancellationTokencancellationToken)
{
    return await SetzeZufriedenheitAsync(false, cancellationToken);
}

// Ermittelt für eine Sortierspalte die jeweils nächste Sortierrichtung.
public string NextSortDirection(string column)
{
    // Wenn eine andere Spalte neu gewählt wird, startet die Sortierung mit aufsteigender Richtung.
    if (!string.Equals(SortExpression, column, StringComparison.OrdinalIgnoreCase))
    {
        return "ASC";
    }

    // Bei wiederholtem Klick auf dieselbe Spalte wird zwischen ASC und DESC umgeschaltet.
    return string.Equals(SortDirection, "ASC", StringComparison.OrdinalIgnoreCase)
        ? "DESC"
        : "ASC";
}

// Interne Hilfsmethode zum Setzen des Zufriedenheitsstatus.
private async Task<IActionResult> SetzeZufriedenheitAsync(bool zufrieden, CancellationTokencancellationToken)
{
    // Lädt zunächst den Seitenkontext und prüft dabei ggf. Redirect-Bedingungen.
    var load = await LadeSeiteAsync(cancellationToken);
    if (load is not null)
    {
        return load;
    }

    // Ohne gültige Anmeldung oder IDs wird auf die Seite selbst zurückgeleitet.
    if (AnmeldungErforderlich || string.IsNullOrEmpty(ProjektId) || string.IsNullOrEmpty(NutzerId))
    {
        return RedirectToSelf();
    }

    // Persistiert den neuen Status über den Fachservice.
    await _partitioService.SetzeZufriedenheitAsync(ProjektId, NutzerId, zufrieden, cancellationToken);

    // Erzeugt eine kurze Statusmeldung für die Oberfläche.
    InfoMeldung = zufrieden ? "Status: zufrieden" : "Status: unzufrieden";
    return await LadeOderRedirectAsync(cancellationToken);
}

// Lädt die Seite und gibt entweder einen Redirect oder die aktuelle Page zurück.
private async Task<IActionResult> LadeOderRedirectAsync(CancellationTokencancellationToken, bool ladeObjekte = true)
{
    // Wenn LadeSeiteAsync einen Redirect benötigt, wird dieser zurückgegeben;
    // andernfalls wird dieselbe Razor Page gerendert.
    var redirect = await LadeSeiteAsync(cancellationToken, ladeObjekte);
    return redirect ?? Page();
}

// Zentrale Lademethode für den Seitentext.
// Sie löst Projekt/Nutzer auf, lädt ggf. Daten für die View und erkennt,
// ob zur URL-Kanonisierung ein Redirect erforderlich ist.
private async Task<IActionResult?> LadeSeiteAsync(
    CancellationTokencancellationToken,
    bool ladeObjekte = true,
    bool uebernehmeDetailInForm = true)
{
    // Wenn keine Projekt-ID im Request vorhanden ist, wird versucht, sie aus dem Cookie zu übernehmen.
    if (string.IsNullOrEmpty(ProjektId))
    {
        ProjektId = Request.Cookies["partitio_projekt"];
    }
}

```

```

// Analog dazu wird die Nutzer-ID aus dem Cookie gelesen, falls sie nicht explizit übergeben wurde.
if (string.IsNullOrEmpty(NutzerId))
{
    NutzerId = Request.Cookies["partitio_nutzer"];
}

// Stellt sicher, dass der aktuelle Projektkontext auf eine existierende Projekt-ID zeigt.
var aufgeloeesteProjektId = await _partitioService.EnsureProjektIdAsync(ProjektId,
cancellationToken);

// Merkt sich, ob das Projekt während dieses Aufrufs implizit gewechselt bzw. neu erzeugt wurde.
var projektGewechselt = !string.Equals(ProjektId, aufgeloeesteProjektId, StringComparison.Ordinal);
ProjektId = aufgeloeesteProjektId;

// Persistiert die aufgelöste Projekt-ID in einem Cookie.
SetzeProjektCookie(ProjektId);

// Falls eine Nutzer-ID bekannt ist, wird versucht, den zugehörigen Nutzer im Projekt zu laden.
if (!string.IsNullOrEmpty(NutzerId))
{
    var nutzer = await _partitioService.HoleNutzerAsync(ProjektId, NutzerId, cancellationToken);
    if (nutzer is not null)
    {
        // Ein gefundener Nutzer bedeutet, dass keine erneute Anmeldung erforderlich ist.
        AnmeldungErforderlich = false;
        NutzerName = nutzer.NutzerName;

        // Aktualisiert die Cookies für den aktuellen Nutzerkontext.
        SetzeCookies(ProjektId, NutzerId);

        if (ladeObjekte)
        {
            // Setzt die Detaildaten zunächst zurück, um alte Zustände zu vermeiden.
            Detail = null;

            // Lädt das freie Guthaben des aktuellen Nutzers.
            Guthaben = await _partitioService.BerechneGuthabenAsync(ProjektId, NutzerId,
cancellationToken);

            // Lädt die sortierte Objektliste für die Tabellen- bzw. Listenansicht.
            Objekte = await _partitioService.LadeObjektlisteAsync(
                ProjektId,
                NutzerId,
                SortExpression,
                SortDirection,
                cancellationToken);

            // Lädt die Zufriedenheitsübersicht aller Nutzer des Projekts.
            Zufriedenheiten = await _partitioService.LadeZufriedenheitenAsync(ProjektId,
cancellationToken);

            // Wenn eine Detail-ID angegeben ist, werden die zugehörigen Detaildaten geladen.
            if (!string.IsNullOrEmpty(DetailObjektId))
            {
                Detail = await _partitioService.HoleObjektDetailAsync(ProjektId, NutzerId,
DetailObjektId, cancellationToken);
                if (Detail is not null && uebernehmeDetailInForm)
                {
                    // Übernimmt geladene Detailwerte in die gebundenen Formularfelder,
                    // sofern das Verhalten in diesem Aufruf gewünscht ist.
                    DetailName = Detail.ObjektName;
                    DetailGebot = Detail.ObjektGebot;
                }
            }

            // Falls ein Löschdialog offen ist, wird zusätzlich der Objektname für die Bestätigung
geladen.
            if (!string.IsNullOrEmpty(LoeschObjektId))
            {
                var detailZumLoeschen = await _partitioService.HoleObjektDetailAsync(ProjektId,
NutzerId, LoeschObjektId, cancellationToken);

```

```

        LoeschObjektName = detailZumLoeschen?.ObjektName;
    }
}

// Liest die aktuell tatsächlich in der URL vorhandenen Query-Werte aus.
var queryProjekt = Request.Query["projekt"].ToString();
var queryNutzer = Request.Query["nutzer"].ToString();

// Prüft, ob die aktuelle Anfrage eine GET-Anfrage ist.
var istGet = HttpMethods.IsGet(Request.Method);

// Bei GET-Anfragen wird die URL kanonisiert, falls Projekt/Nutzer-Kontext nicht zum geladenen
Zustand passt.
if (istGet && (
    projektGewechselt
    || !string.Equals(queryProjekt, ProjektId, StringComparison.Ordinal)
    || (!AnmeldungErforderlich && !string.Equals(queryNutzer, NutzerId,
StringComparison.Ordinal))))
{
    return RedirectToSelf();
}

// Kein Redirect erforderlich: Die aktuelle Seite kann normal gerendert werden.
return null;
}

// Baut eine Redirect-Antwort zurück auf dieselbe Razor Page mit dem aktuellen Seitenzustand in der
URL.
private IActionResult RedirectToSelf()
{
    // Übergibt den aktuellen Kontext explizit als Route-/Query-Werte an RedirectToPage.
    return RedirectToPage("/Partitio", null, new
    {
        projekt = ProjektId,
        // Wenn eine Anmeldung erforderlich ist, wird keine Nutzer-ID in die URL geschrieben.
        nutzer = AnmeldungErforderlich ? null : NutzerId,
        sort = SortExpression,
        dir = SortDirection,
        detail = DetailObjektId,
        loeschen = LoeschObjektId
    });
}

// Baut aus einem relativen Pfad plus Query einen absoluten Link inklusive Schema und Host.
private string AbsoluterLink(string pathAndQuery)
{
    return $"{Request.Scheme}://{Request.Host}{pathAndQuery}";
}

// Schreibt den aktuellen Projektkontext in ein langlebiges Cookie.
private void SetzeProjektCookie(string projektId)
{
    Response.Cookies.Append("partitio_projekt", projektId, new CookieOptions
    {
        // Das Cookie bleibt für ein Jahr gültig.
        Expires = DateTimeOffset.UtcNow.AddYears(1),

        // HttpOnly = false, damit der Wert bei Bedarf auch clientseitig erreichbar bleibt.
        HttpOnly = false,

        // Markiert das Cookie als für die Anwendung wesentlich.
        IsEssential = true
    });
}

// Schreibt sowohl Projekt- als auch Nutzerkontext in Cookies.
private void SetzeCookies(string projektId, string nutzerId)
{
    // Projekt-Cookie wird immer gemeinsam mit dem Nutzerkontext aktualisiert.

```

```

        SetzeProjektCookie(projektId);

        Response.Cookies.Append("partitio_nutzer", nutzerId, new CookieOptions
        {
            // Auch dieses Cookie bleibt ein Jahr gültig.
            Expires = DateTimeOffset.UtcNow.AddYears(1),

            // Nicht HttpOnly, damit der Kontext nicht ausschließlich serverseitig verborgen ist.
            HttpOnly = false,

            // Für den Betrieb der Anwendung wesentlich.
            IsEssential = true
        });
    }
}

```

4.11 Partitio.cshtml

```

<!--
Diese Datei ist die zentrale Benutzeroberfläche der Anwendung Partitio.
Sie bildet den eigentlichen Arbeitsbereich der Nutzerinnen ab: Anmeldung
über einen Namen, Anzeige des aktuellen Projekts, Überblick über Guthaben,
Objektliste, Detailbearbeitung einzelner Objekte, Löschdialoge, Status
"zufrieden / unzufrieden", Einladungs- und Lesezeichenfunktionen sowie die
kamerabasierte Aufnahme neuer Objekte mit anschließendem Upload. Die View
ist eng mit dem PartitioModel verbunden und reagiert stark auf dessen Zustand:
Je nachdem, ob ein Nutzerkontext vorhanden ist, rendert sie entweder die
Registrierungsoberfläche oder die eigentliche Anwendungsansicht. Neben Razor-
Markup enthält die Datei auch clientseitiges JavaScript für Sidebar, optionale
Panels, Kamera, Upload und kleinere Komfortfunktionen. Sie ist damit die
zentrale Schnittstelle zwischen Benutzerinteraktion, serverseitiger Logik und
clientseitigem Verhalten.
-->
<!-- Definiert eine feste Route "/partitio" für diese Razor Page. -->
@page "/partitio"
<!-- Verbindet die Ansicht explizit mit dem voll qualifizierten PartitioModel-Typ. -->
@model Partitio.Pages.PartitioModel
@{
    <!-- Setzt den Seitentitel dynamisch: ohne Nutzernamen neutral, mit Nutzernamen personalisiert. -->
    ViewData["Title"] = string.IsNullOrEmpty(Model.NutzerName) ? "Partitio : $"Partitio -
{Model.NutzerName}";
}

<!-- Definiert einen optionalen Head-Abschnitt, den das Layout an geeigneter Stelle rendert. -->
@section Head {
    <!-- Bindet W3CSS als zentrales UI-Framework für viele Layout- und Komponentenklassen ein. -->
    <link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css" />
    <!-- Bindet Font Awesome für Icons wie das Hamburger-Menü ein. -->
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/font-awesome/4.7.0/css/font-awesome.min.css" />
    <!-- Bindet Material Icons für Symbolschaltflächen wie "close" ein. -->
    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons" />
}

<!-- Solange kein gültiger Nutzerkontext vorhanden ist, wird die Anmelde-/Registrierungsoberfläche
angezeigt. -->
@if (Model.AnmeldungErforderlich)
{
    <div class="w3-content w3-margin-top">
        <!-- Zeigt eine serverseitig gesetzte Fehlermeldung oberhalb des Formulars an. -->
        @if (!string.IsNullOrEmpty(Model.FehlerMeldung))
        {
            <div class="w3-panel w3-pale-red w3-leftbar w3-border-red">@Model.FehlerMeldung</div>
        }

        <!-- Karte mit dem Formular zum Anlegen einer neuen Teilnehmerin. -->
        <div class="w3-card w3-round w3-margin">
            <div class="w3-container">
                <h3>Neue Teilnehmerin</h3>
            </div>
        </div>
    </div>
}

```

```

<!-- Formular ruft den Razor-Page-Handler "NutzerAnlegen" auf. -->
<form method="post" asp-page-handler="NutzerAnlegen">
  <!-- Versteckte Felder halten Kontextparameter über den Postback hinweg stabil. -->
  <input type="hidden" name="projekt" value="@Model.ProjektId" />
  <input type="hidden" name="sort" value="@Model.SortExpression" />
  <input type="hidden" name="dir" value="@Model.SortDirection" />
  <!-- asp-for verbindet Label und Eingabefeld mit der Property NeuerName des PageModels. -->

  <label asp-for="NeuerName">Name</label>
  <input asp-for="NeuerName" maxlength="10" class="w3-input w3-border" />
  <!-- Sendet das Formular ab und stößt damit die Nutzeranlage an. -->
  <button type="submit" class="w3-button w3-round w3-light-gray w3-border w3-border-gray
w3-margin-top">OK</button>
</form>

<!-- Falls der Service vorhandene Namen zurückliefert, werden diese als Hilfestellung
angezeigt. -->
@if (Model.VorhandeneNamen.Any())
{
  <h4>Bisherige Teilnehmerinnen</h4>
  <ul>
    <!-- Rendert jeden bereits existierenden Namen als Listenelement. -->
    @foreach (var name in Model.VorhandeneNamen)
    {
      <li>@name</li>
    }
  </ul>
}
</div>
</div>
</div>
}
else
{
  <!-- Die eigentliche App-Ansicht für bereits bekannte/angemeldete Nutzerinnen. -->
  <div id="Panel_seitenleiste" class="halbtransparent w3-sidebar w3-bar-block w3-card-4 w3-animate-left"
style="display:none;">
    <div id="seitenleiste_inhalt" class="w3-card w3-margin w3-round w3-light-grey automatische_breite"
onclick="event.stopPropagation()">
      <!-- Schaltfläche zum Schließen der Seitenleiste. -->
      <button type="button" class="w3-button w3-block text-left"
onclick="seitenleiste_schliessen()"><i class='material-icons'>close</i></button>
      <p></p>
      <!-- Öffnet das Panel für den Bookmark-Link. -->
      <button type="button" class="w3-button w3-block text-left"
onclick="oeffne_panel('panel lesezeichen')">Lesezeichen erstellen</button>
      <p></p>
      <!-- Öffnet das Einladungs-Panel. -->
      <button type="button" class="w3-button w3-block text-left"
onclick="oeffne_panel('panel einladung')">Neue Teilnehmerinnen</button>
      <p></p>
      <!-- Öffnet das Panel zur Anzeige und Änderung von Zufriedenheitszuständen. -->
      <button type="button" class="w3-button w3-block text-left"
onclick="oeffne_panel('panel zufriedenheit')">Alle zufrieden?</button>
      <p></p>
      <!-- Statischer Link auf eine Hilfe-PDF. -->
      <a class="w3-button w3-block text-left" href="/partitio.pdf">Hilfe</a>
    </div>
  </div>

  <!-- Obere feste Leiste mit Menü und aktuellem Nutzernamen. -->
  <div class="w3-top">
    <div class="w3-content w3-teal w3-padding">
      <button id="button_hamburger" type="button" class="w3-button"
onclick="seitenleiste_oeffnen(event)"><i class="fa fa-bars"></i></button>
      &nbsp; &nbsp; &nbsp; &nbsp; Partitio - @Model.NutzerName
    </div>
  </div>

  <br />
  <br />

```

```

<!-- Untere feste Leiste mit Guthaben und manuellem Aktualisierungslink. -->
<div class="w3-bottom">
  <div class="w3-content w3-teal w3-padding">
    Guthaben: @Model.Guthaben
    <a class="w3-right" style="text-decoration:none" asp-page="/Partitio" asp-route-
projekt="@Model.ProjektId" asp-route-nutzer="@Model.NutzerId" asp-route-sort="@Model.SortExpression" asp-
route-dir="@Model.SortDirection">Aktualisieren</a>
  </div>
</div>

<!-- Floating Action Button zum Starten der Kamerafunktion. -->
<button id="button_kamera_starten" onclick="kamera_starten(); return false;" class="w3-button w3-round-
xlarge w3-large w3-card w3-teal fixed-button">+</button>

<div class="w3-content">
  <!-- Fehler- und Informationsmeldungen werden im aktiven App-Modus oberhalb des Inhalts
eingebildet. -->
  @if (!string.IsNullOrEmpty(Model.FehlerMeldung))
  {
    <div class="w3-panel w3-pale-red w3-leftbar w3-border-red w3-margin">@Model.FehlerMeldung</div>
  }

  @if (!string.IsNullOrEmpty(Model.InfoMeldung))
  {
    <div class="w3-panel w3-pale-green w3-leftbar w3-border-green w3-margin">@Model.InfoMeldung</
div>
  }

  <!-- Panel zum Anzeigen und Kopieren eines persönlichen Bookmark-Links. -->
  <div id="panel lesezeichen" class="w3-card w3-round w3-margin optionale-panels"
style="display:none;">
    <div class="w3-container">
      <h3>Lesezeichen erstellen</h3>
      <p>Partitio hat einen Cookie gesetzt. Trotzdem ist es sinnvoll, diesen Link zu speichern:</
p>
      <!-- Zeigt den vom PageModel vorberechneten absoluten Lesezeichen-Link an. -->
      <div id="label_link lesezeichen" class="w3-container w3-text-blue monospace-
font">@Model.LesezeichenLink</div>
      <button type="button" class="w3-button w3-round w3-light-gray w3-border w3-border-gray w3-
margin" onclick="kopiere_in_die_zwischenablage('label_link lesezeichen')">Link in Zwischenablage kopieren</
button>
    </div>
  </div>

  <!-- Panel zum Anzeigen und Kopieren eines Einladungslinks für weitere Teilnehmerinnen. -->
  <div id="panel einladung" class="w3-card w3-round w3-margin optionale-panels"
style="display:none;">
    <div class="w3-container">
      <h3>Neue Teilnehmerinnen</h3>
      <p>Diesen Link an neue Teilnehmerinnen senden:</p>
      <div id="label_link neue teilnehmer" class="w3-container w3-text-blue monospace-
font">@Model.EinladungLink</div>
      <button type="button" class="w3-button w3-round w3-light-gray w3-border w3-border-gray w3-
margin" onclick="kopiere_in_die_zwischenablage('label_link neue teilnehmer')">Link in Zwischenablage
kopieren</button>
    </div>
  </div>

  <!-- Panel für die Übersicht aller Zufriedenheitszustände inklusive eigener Statusänderung. -->
  <div id="panel zufriedenheit" class="w3-card w3-round w3-margin optionale-panels"
style="display:none;">
    <div class="w3-container">
      <h3>Alle zufrieden?</h3>
      <table class="w3-table w3-bordered">
        <tbody>
          <!-- Jede Zeile zeigt einen Nutzer und den aktuellen Zufriedenheitsstatus an. -->
          @foreach (var eintrag in Model.Zufriedenheiten)
          {
            <tr>
              <td>@eintrag.NutzerName</td>

```

```

        <td>@(eintrag.Zufrieden ? "ist zufrieden" : "ist unzufrieden")</td>
    </tr>
}
</tbody>
</table>
<div class="w3-margin-bottom">
    <!-- Formular zum Setzen des eigenen Status auf zufrieden. -->
    <form method="post" asp-page-handler="Zufrieden" class="w3-show-inline-block">
        <input type="hidden" name="projekt" value="@Model.ProjektId" />
        <input type="hidden" name="nutzer" value="@Model.NutzerId" />
        <input type="hidden" name="sort" value="@Model.SortExpression" />
        <input type="hidden" name="dir" value="@Model.SortDirection" />
        <input type="hidden" name="detail" value="@Model.DetailObjektId" />
        <button type="submit" class="w3-button w3-round w3-light-gray w3-border w3-border-
gray w3-margin-top">Zufrieden</button>
    </form>
    <!-- Formular zum Setzen des eigenen Status auf unzufrieden. -->
    <form method="post" asp-page-handler="Unzufrieden" class="w3-show-inline-block">
        <input type="hidden" name="projekt" value="@Model.ProjektId" />
        <input type="hidden" name="nutzer" value="@Model.NutzerId" />
        <input type="hidden" name="sort" value="@Model.SortExpression" />
        <input type="hidden" name="dir" value="@Model.SortDirection" />
        <input type="hidden" name="detail" value="@Model.DetailObjektId" />
        <button type="submit" class="w3-button w3-round w3-light-gray w3-border w3-border-
gray w3-margin-top">Unzufrieden</button>
    </form>
</div>
</div>
</div>

<!-- Versteckter Bereich für die Kamerasitzung, der erst nach Start eingeblendet wird. -->
<div id="foto_session" style="display:none;">
    <br />
    <br />
    <div class="w3-card w3-round w3-margin">
        <div id="videoContainer">
            <!-- Videoelement für den Live-Kamerastream. -->
            <video id="videoElement" class="w3-round w3-margin-top" autoplay="autoplay"
playsinline="playsinline"></video>
        </div>
        <!-- Canvas bleibt unsichtbar und dient nur zum Aufnehmen/Skalieren des Bildes vor dem
Upload. -->
        <canvas id="canvas" height="300" width="300" style="display:none;"></canvas>
        <button type="button" class="w3-button w3-round w3-light-gray w3-border w3-border-gray w3-
margin" onclick="kamera_stoppen()">Abbrechen</button>
        <button type="button" class="w3-button w3-round w3-light-gray w3-border w3-border-gray w3-
right w3-margin" onclick="bild_aufnehmen()">Bild aufnehmen</button>
    </div>
</div>

<!-- Löschbestätigungsdialog, der nur erscheint, wenn ein Objekt zur Löschung markiert wurde. -->
@if (!string.IsNullOrEmpty(Model.LoeschObjektId))
{
    <div class="w3-card w3-round w3-margin">
        <div class="w3-container">
            <h3>Löschbestätigung</h3>
            <p>@(Model.LoeschObjektName ?? "Objekt") wird unwiderbringlich gelöscht.</p>
            <div class="w3-margin-top w3-margin-bottom">
                <div class="w3-left">
                    <!-- Formular zum Abbrechen des Löschvorgangs. -->
                    <form method="post" asp-page-handler="ObjektLoeschenAbbrechen" class="w3-show-
inline-block">
                        <input type="hidden" name="projekt" value="@Model.ProjektId" />
                        <input type="hidden" name="nutzer" value="@Model.NutzerId" />
                        <input type="hidden" name="sort" value="@Model.SortExpression" />
                        <input type="hidden" name="dir" value="@Model.SortDirection" />
                        <input type="hidden" name="detail" value="@Model.DetailObjektId" />
                        <button type="submit" class="w3-button w3-round w3-light-gray w3-border w3-
border-gray">Abbrechen</button>
                    </form>
                </div>
            </div>
        </div>
    </div>
}

```

```

        <div class="w3-right">
            <!-- Formular zur endgültigen Bestätigung der Löschung. -->
            <form method="post" asp-page-handler="ObjektLoeschenBestaetigen" asp-route-
objektId="@Model.LoeschObjektId" class="w3-show-inline-block">
                <input type="hidden" name="projekt" value="@Model.ProjektId" />
                <input type="hidden" name="nutzer" value="@Model.NutzerId" />
                <input type="hidden" name="sort" value="@Model.SortExpression" />
                <input type="hidden" name="dir" value="@Model.SortDirection" />
                <button type="submit" class="w3-button w3-round w3-pale-red w3-border w3-
border-gray">Löschen</button>
            </form>
        </div>
        <div class="w3-clear"></div>
    </div>
</div>
</div>
}

<!-- Detailkarte für das aktuell ausgewählte Objekt. -->
@if (Model.Detail is not null)
{
    <div class="w3-card w3-round w3-margin">
        <!-- Zeigt das Bild des ausgewählten Objekts an. -->
        
        <!-- Link zum Schließen der Detailansicht durch Rückkehr auf dieselbe Seite ohne detail-
Parameter. -->
        <a class="w3-button w3-right" asp-page="/Partitio" asp-route-projekt="@Model.ProjektId"
asp-route-nutzer="@Model.NutzerId" asp-route-sort="@Model.SortExpression" asp-route-
dir="@Model.SortDirection"><i class='material-icons'>close</i></a>
        <br />
        <!-- Formular zum Speichern von Namen und Gebot des Objekts. -->
        <form id="detail-save-form" method="post" asp-page-handler="ObjektSpeichern" asp-route-
objektId="@Model.Detail.ObjektId">
            <input type="hidden" name="projekt" value="@Model.ProjektId" />
            <input type="hidden" name="nutzer" value="@Model.NutzerId" />
            <input type="hidden" name="sort" value="@Model.SortExpression" />
            <input type="hidden" name="dir" value="@Model.SortDirection" />
            <input type="hidden" name="detail" value="@Model.DetailObjektId" />
            <!-- Eingabefeld für den editierbaren Objektname. -->
            <input asp-for="DetailName"
                class="w3-margin-left"
                style="width:200px"
                onfocus="objekt_name_fokus(this)" />
            <!-- Reine Anzeigeinformation zum aktuellen Besitzer. -->
            <div class="w3-margin-left w3-margin-top">Besitzer: @Model.Detail.BesitzerName</div>
            <div class="w3-margin-left w3-margin-top">
                Gebot:
                <!-- Nur wenn ein Gebot möglich ist, wird das Feld aktiv und mit Min/Max versehen
gerendert. -->
                @if (Model.Detail.GebotMoeglich)
                {
                    <input asp-for="DetailGebot" type="number" min="@Model.Detail.MinGebot"
max="@Model.Detail.MaxGebot" style="width:70px" onfocus="detail_gebot_fokus(this)" />
                }
                else
                {
                    <input asp-for="DetailGebot" type="number" readonly style="width:70px"
onfocus="detail_gebot_fokus(this)" />
                }
                <span class="w3-text-red">@Model.Detail.GebotHinweis</span>
            </div>
        </form>
        <div class="w3-container w3-margin-top w3-margin-bottom">
            <div class="w3-left">
                <!-- Aktiviert den Bestätigen-Button nur bei zulässigem Gebot. -->
                @if (Model.Detail.GebotMoeglich)
                {
                    <button type="submit"
                        form="detail-save-form"
                        class="w3-button w3-round w3-light-gray w3-border w3-border-gray w3-

```

```

margin-bottom">Bestätigen</button>
    }
    else
    {
        <button type="button" disabled class="w3-button w3-round w3-light-gray w3-
border w3-border-gray w3-margin-bottom">Bestätigen</button>
    }
</div>
<div class="w3-right">
    <!-- Separates Formular zum Starten des Löschmods für dieses Objekt. -->
    <form method="post" asp-page-handler="ObjektLoeschenAnfragen" asp-route-
objektId="@Model.Detail.ObjektId" class="w3-show-inline-block">
        <input type="hidden" name="projekt" value="@Model.ProjektId" />
        <input type="hidden" name="nutzer" value="@Model.NutzerId" />
        <input type="hidden" name="sort" value="@Model.SortExpression" />
        <input type="hidden" name="dir" value="@Model.SortDirection" />
        <button type="submit" class="w3-button w3-pale-red w3-round w3-border w3-
border-gray w3-margin-bottom">Löschen</button>
    </form>
</div>
<div class="w3-clear"></div>
</div>
</div>
}

<!-- Haupttabelle mit allen Objekten des Projekts. -->
<div class="w3-card w3-round w3-margin">
    <table class="w3-table w3-bordered">
        <thead>
            <tr>
                <!-- Erste Spalte für das Vorschaubild ohne eigene Sortierung. -->
                <th></th>
                <th>
                    <!-- Sortierbarer Link für die Objektnamen-Spalte. -->
                    <a asp-page="/Partitio" asp-route-projekt="@Model.ProjektId" asp-route-
nutzer="@Model.NutzerId" asp-route-sort="Objekt" asp-route-dir="@Model.NextSortDirectionObjekt"
style="text-decoration:none">Objekt</a>
                </th>
                <th class="text-right">
                    <!-- Sortierbarer Link für die Gebots-Spalte. -->
                    <a asp-page="/Partitio" asp-route-projekt="@Model.ProjektId" asp-route-
nutzer="@Model.NutzerId" asp-route-sort="Gebot" asp-route-dir="@Model.NextSortDirectionGebot" style="text-
decoration:none">Gebot</a>
                </th>
                <th>
                    <!-- Sortierbarer Link für die Besitzer-Spalte. -->
                    <a asp-page="/Partitio" asp-route-projekt="@Model.ProjektId" asp-route-
nutzer="@Model.NutzerId" asp-route-sort="Besitzer" asp-route-dir="@Model.NextSortDirectionBesitzer"
style="text-decoration:none">Besitzer</a>
                </th>
            </tr>
        </thead>
        <tbody>
            <!-- Jede Objektzeile erhält einen Detail-Link, damit die ganze Zeile klickbar gemacht
werden kann. -->
            @foreach (var objekt in Model.Objekte)
            {
                var detailLink = Url.Page("/Partitio", null, new
                {
                    projekt = Model.ProjektId,
                    nutzer = Model.NutzerId,
                    sort = Model.SortExpression,
                    dir = Model.SortDirection,
                    detail = objekt.Id
                }, null);
                <tr class="w3-hover-light-grey"
onclick="window.location.href='@detailLink'">
                    <!-- Zeigt Vorschaubild, Namen, Gebot und Besitzer. -->
                    <td></td>
                    <td>@objekt.Objekt</td>

```

```

        <td class="text-right">@objekt.Gebot</td>
        <td>@objekt.Besitzer</td>
    </tr>
}
</tbody>
</table>
</div>

<br />
<br />
<br />
</div>

<script>
    // Übernimmt Projekt- und Nutzer-ID aus dem gerenderten Model in JavaScript-Konstanten.
    const PROJEKT_ID = '@(Model.ProjektId ?? string.Empty)';
    const NUTZER_ID = '@(Model.NutzerId ?? string.Empty)';

    // Hält den aktuellen MediaStream der Kamera, solange eine Fotosession aktiv ist.
    let stream = null;

    // Lädt die Seite alle 10 Minuten neu, um den Datenstand aktuell zu halten.
    setInterval(function () {
        window.location.href = window.location.pathname + window.location.search;
    }, 600000);

    // Öffnet die Seitenleiste und verhindert dabei, dass das auslösende Klickevent weiterblubbert.
    function seitenleiste_oeffnen(event) {
        if (event) {
            event.stopPropagation();
        }
        const panel = document.getElementById('Panel_seitenleiste');
        if (panel) {
            panel.style.display = 'block';
        }
    }

    // Schließt die Seitenleiste wieder.
    function seitenleiste_schliessen() {
        const panel = document.getElementById('Panel_seitenleiste');
        if (panel) {
            panel.style.display = 'none';
        }
    }

    // Schließt die Seitenleiste, wenn außerhalb des Menüs geklickt wird.
    document.addEventListener('click', function (event) {
        const panel = document.getElementById('Panel_seitenleiste');
        const menuInhalt = document.getElementById('seitenleiste_inhalt');
        const hamburger = document.getElementById('button_hamburger');
        const klickImMenue = menuInhalt && menuInhalt.contains(event.target);
        const klickAufHamburger = hamburger && hamburger.contains(event.target);
        if (panel && panel.style.display !== 'none' && !klickImMenue && !klickAufHamburger) {
            panel.style.display = 'none';
        }
    });

    // Blendet alle optionalen Inhalts-Panels aus.
    function verstecke_optionale_panels() {
        const panels = document.querySelectorAll('.optionale-panels');
        panels.forEach(function (panel) {
            panel.style.display = 'none';
        });
    }

    // Öffnet genau ein optionales Panel, schließt die Seitenleiste und scrollt zum Panel.
    function oeffne_panel(id) {
        verstecke_optionale_panels();
        seitenleiste_schliessen();
        const element = document.getElementById(id);
        if (element) {

```

```

        element.style.display = 'block';
        element.scrollIntoView({ behavior: 'smooth' });
    }
}

// Kopiert den Textinhalt eines Elements in die Zwischenablage.
async function kopiere_in_die_zwischenablage(labelId) {
    const element = document.getElementById(labelId);
    if (!element) {
        return;
    }

    const text = (element.innerText || '').trim();
    if (!text) {
        return;
    }

    // navigator.clipboard funktioniert auf mobilen Browsern oft nur in Secure Contexts (HTTPS/
localhost).
    try {
        // Bevorzugt die moderne Clipboard API, wenn sie verfügbar ist.
        if (navigator.clipboard && window.isSecureContext) {
            await navigator.clipboard.writeText(text);
            return;
        }
    } catch (fehler) {
        console.log('Kopieren per Clipboard API fehlgeschlagen:', fehler);
    }

    try {
        // Fallback: unsichtbares Textfeld erzeugen und per execCommand kopieren.
        const textarea = document.createElement('textarea');
        textarea.value = text;
        textarea.setAttribute('readonly', '');
        textarea.style.position = 'fixed';
        textarea.style.opacity = '0';
        document.body.appendChild(textarea);
        textarea.focus();
        textarea.select();
        textarea.setSelectionRange(0, textarea.value.length);
        document.execCommand('copy');
        document.body.removeChild(textarea);
    } catch (fehler) {
        console.log('Kopieren fehlgeschlagen:', fehler);
        alert('Kopieren in die Zwischenablage ist in diesem Browser nicht möglich.');
```

```

    }
}

// Beendet einen laufenden Kamerastream und blendet die Fotosession wieder aus.
function kamera_stoppen() {
    if (stream) {
        stream.getTracks().forEach(function (track) { track.stop(); });
        stream = null;
    }

    const session = document.getElementById('foto_session');
    const button = document.getElementById('button_kamera_starten');
    if (session) {
        session.style.display = 'none';
    }
    if (button) {
        button.style.display = 'block';
    }
}

// Nimmt ein Bild aus dem Video-Stream auf, bereitet es auf und lädt es an den Upload-Endpunkt
hoch.
async function bild_aufnehmen() {
    // Ohne Projekt- und Nutzerkontext ist kein Upload möglich.
    if (!PROJEKT_ID || !NUTZER_ID) {
        alert('Projekt- oder Nutzer-ID fehlt.');
```

return;

```
    }

    // Ohne laufenden Stream kann kein Bild aufgenommen werden.
    if (!stream) {
        alert('Zugriff auf die Kamera nicht möglich oder abgelehnt.');
```

return;

```
    }

    // Erzeugt eine neue clientseitige UUID als Bild-ID.
    const bildId = crypto.randomUUID();
    const video = document.getElementById('videoElement');
    const canvas = document.getElementById('canvas');
    if (!video || !canvas) {
        return;
    }

    const context = canvas.getContext('2d');
    if (!context) {
        return;
    }

    const videoWidth = video.videoWidth;
    const videoHeight = video.videoHeight;
    const canvasSize = canvas.width;

    // Skaliert das Kamerabild so, dass das Canvas vollständig gefüllt wird.
    const scale = Math.max(canvasSize / videoWidth, canvasSize / videoHeight);
    const x = (canvasSize / 2) - (videoWidth / 2) * scale;
    const y = (canvasSize / 2) - (videoHeight / 2) * scale;

    // Zeichnet den skalierten Videoausschnitt auf das Canvas.
    context.drawImage(video, x, y, videoWidth * scale, videoHeight * scale);

    // Exportiert den Canvas-Inhalt als JPEG-Daten-URL und wandelt sie danach in ein Blob um.
    const imageData = canvas.toDataURL('image/jpeg', 0.8);
    const blob = await fetch(imageData).then(function (res) { return res.blob(); });

    // Baut die Multipart-Formular Daten für den Upload auf.
    const daten = new FormData();
    daten.append('bild', blob, bildId + '.jpg');
    daten.append('projekt_id', PROJEKT_ID);
    daten.append('nutzer_id', NUTZER_ID);
    daten.append('bild_id', bildId);

```

```

    try {
        // Sendet den Upload an den serverseitigen Minimal-API-Endpunkt.
        const response = await fetch('/api/bild-upload', {
            method: 'POST',
            body: daten
        });

        if (!response.ok) {
            throw new Error('Upload fehlgeschlagen');
        }

        // Liest die JSON-Antwort, beendet die Kameraansicht und lädt die Seite neu.
        await response.json();
        kamera_stoppen();
        window.location.href = window.location.pathname + window.location.search;
    } catch (fehler) {
        console.log('Ein Fehler ist aufgetreten beim Hochladen:', fehler);
    }
}

// Leert den Platzhalternamen "..." beim Fokussieren und markiert den Text.
function objekt_name_fokus(input) {
    if (!input) {
        return;
    }

    if (input.value === '...') {
        input.value = '';
    }

    input.select();
}

// Markiert den kompletten Gebotswert beim Fokussieren des Eingabefelds.
function detail_gebot_fokus(input) {
    if (!input) {
        return;
    }

    input.select();
}
}
</script>
}

```

4.12 Error.cshtml.cs

```

/*
 * Diese Datei enthält das PageModel für die Fehlerseite der Anwendung.
 * Sie wird typischerweise dann verwendet, wenn während der Verarbeitung einer
 * Anfrage ein unbehandelter Fehler auftritt und die ASP.NET-Core-Pipeline auf
 * die konfigurierte Error-Seite umleitet. Das Model stellt dabei vor allem eine
 * Request-ID bereit, mit der sich eine konkrete Anfrage in Logs oder Diagnosedaten
 * wiederfinden lässt. Die Seite selbst enthält also kaum Fachlogik, erfüllt aber
 * eine wichtige Rolle für Fehlersichtbarkeit, Support und technische Nachverfolgung.
 */

// Importiert Typen zur Laufzeitdiagnose, insbesondere Activity für Request-Tracing.
using System.Diagnostics;

// Importiert ASP.NET-Core-MVC-Attribute und Hilfstypen.
using Microsoft.AspNetCore.Mvc;

// Importiert die Basisklasse PageModel für Razor-Pages-PageModels.
using Microsoft.AspNetCore.Mvc.RazorPages;

// Ordnet die Fehlerseite dem Pages-Namespace der Anwendung zu.
namespace Partitio.Pages;

// Verhindert Caching dieser Fehlerseite.

```

```
// Fehlerseiten sollen nicht vom Browser oder Proxy zwischengespeichert werden,
// damit immer der aktuelle Fehlerkontext angezeigt wird.
[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]

// Deaktiviert Antiforgery-Prüfung für diese Seite.
// Da die Seite nur angezeigt und nicht für zustandsändernde Formulare verwendet wird,
// ist das hier unkritisch und vereinfacht die Fehlerbehandlung.
[IgnoreAntiforgeryToken]

// PageModel für die Razor-Page "Error".
public class ErrorModel : PageModel
{
    // Enthält die eindeutige Kennung der aktuellen Anfrage, sofern verfügbar.
    public string? RequestId { get; set; }

    // Liefert true, wenn eine Request-ID vorhanden ist und angezeigt werden kann.
    // Die View kann diese Hilfeigenschaft nutzen, um die ID nur bei Bedarf einzublenden.
    public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);

    // GET-Handler der Fehlerseite.
    public void OnGet()
    {
        // Verwendet bevorzugt die aktuelle Activity-ID aus dem verteilten Tracing.
        // Falls keine Activity aktiv ist, wird als Fallback die HTTP-TraceIdentifier-ID genutzt.
        RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier;
    }
}
```

4.13 Error.cshtml

```
<!--
Diese Razor-Page stellt die sichtbare Fehlerseite der Anwendung dar.
Sie wird typischerweise angezeigt, wenn die ASP.NET-Core-Pipeline einen
unbehandelten Fehler abfängt und auf die Error-Seite umleitet. Die Seite
ist bewusst einfach gehalten: Sie informiert allgemein darüber, dass ein
Fehler aufgetreten ist, und blendet optional eine Request-ID ein, über die
sich der konkrete Vorfall in Logs oder Diagnosewerkzeugen wiederfinden lässt.
Zusätzlich erklärt sie den Unterschied zwischen Entwicklungs- und
Produktivumgebung, damit bei lokaler Fehlersuche mehr Details sichtbar
gemacht werden können, ohne in einer produktiven Umgebung interne
Ausnahmeinformationen an Endnutzer preiszugeben.
-->
<!-- Markiert diese Datei als direkt routbare Razor Page. -->
@page
<!-- Verknüpft die View mit dem zugehörigen PageModel ErrorModel. -->
@model ErrorModel
@{
    <!-- Setzt den Seitentitel für Layout und Browser-Tab. -->
    ViewData["Title"] = "Error";
}

<!-- Hauptüberschrift der Fehlerseite. -->
<h1 class="text-danger">Error.</h1>
<!-- Kurze sekundäre Erklärung, dass die Anfrage nicht verarbeitet werden konnte. -->
<h2 class="text-danger">An error occurred while processing your request.</h2>

<!-- Zeigt die Request-ID nur dann an, wenn das Model tatsächlich eine ID bereitstellt. -->
@if (Model.ShowRequestId)
{
    <p>
        <!-- Beschriftung der Diagnoseinformation. -->
        <strong>Request ID:</strong> <code>@Model.RequestId</code>
    </p>
}

<!-- Zusatzinformation speziell für Entwicklerinnen und Entwickler. -->
<h3>Development Mode</h3>
<p>
    <!-- Erläutert, dass in der Development-Umgebung detailliertere Fehlermeldungen sichtbar werden. -->
```

Swapping to the **Development** environment displays detailed information about the error that occurred.

</p>

<p>

<!-- Warnhinweis, dass Development nicht in produktiven Umgebungen aktiv sein sollte. -->

The Development environment shouldn't be enabled for deployed applications.

It can result in displaying sensitive information from exceptions to end users.

For local debugging, enable the **Development** environment by setting the **ASPNETCORE_ENVIRONMENT** environment variable to **Development** and restarting the app.

</p>

5 Literatur

- [1] J. Buchholz, „Partitio Webseite“. Zugegriffen: 16. März 2026. [Online]. Verfügbar unter: https://webserver.kee9wtun8rfetirg.myfritz.net/partitio_webseite/
- [2] „ChatGPT“. Zugegriffen: 16. März 2026. [Online]. Verfügbar unter: <https://chatgpt.com/de-DE/>
- [3] „GitHub Copilot · Your AI pair programmer“. Zugegriffen: 16. März 2026. [Online]. Verfügbar unter: <https://github.com/features/copilot>
- [4] „Vibe Coding“. Zugegriffen: 16. März 2026. [Online]. Verfügbar unter: https://de.wikipedia.org/w/index.php?title=Vibe_Coding&oldid=264883112
- [5] „Kō“. Zugegriffen: 16. März 2026. [Online]. Verfügbar unter: <https://de.wikipedia.org/w/index.php?title=K%C5%8D&oldid=236300827>