Dokumentation

Patterns and Frameworks: Verleihplattform

Daniel Jocher, Matrikelnummer 952593
Sophie Haack, Matrikelnummer 944011

Berliner Hochschule für Technik
Bachelor Medieninformatik Online

March 19, 2025

Contents

1 Einführung

1.1 Problemstellung

Die Entwicklung der Applikation basiert auf der Herausforderung, eine effiziente und benutzerfreundliche Plattform für die Verwaltung und Buchung von mietbaren Artikeln bereitzustellen. Traditionelle Mietprozesse sind oft mit manuellen Abstimmungen, unklaren Verfügbarkeiten und ineffizienten Kommunikationswegen verbunden, was sowohl für Anbieter:innen als auch für Mieter:innen zu Verzögerungen und Mehraufwand führt. Die Applikation soll diese Probleme durch eine digitale Lösung überwinden, indem sie eine strukturierte Verwaltung von Artikeln, eine transparente Buchungsabwicklung und eine automatisierte Benachrichtigung über wichtige Ereignisse ermöglicht. Durch eine intuitive Benutzeroberfläche und eine strukturierte Backend-Architektur wird sichergestellt, dass sowohl Anbietende als auch Mieter:innen jederzeit einen aktuellen Überblick über Verfügbarkeiten, Preise und Buchungen haben.

2 Domainmodell und User-Stories

2.1 Domainmodell

Das Domainmodell bietet die Grundlage für die Entwicklung unserer Applikation. Dennoch mussten wir das Modell in kleinen Punkten während der Entwicklung ab und zu anpassen, um auf Probleme oder Dinge die zu Beginn nicht komplett durchdacht wurden, zu reagieren.1

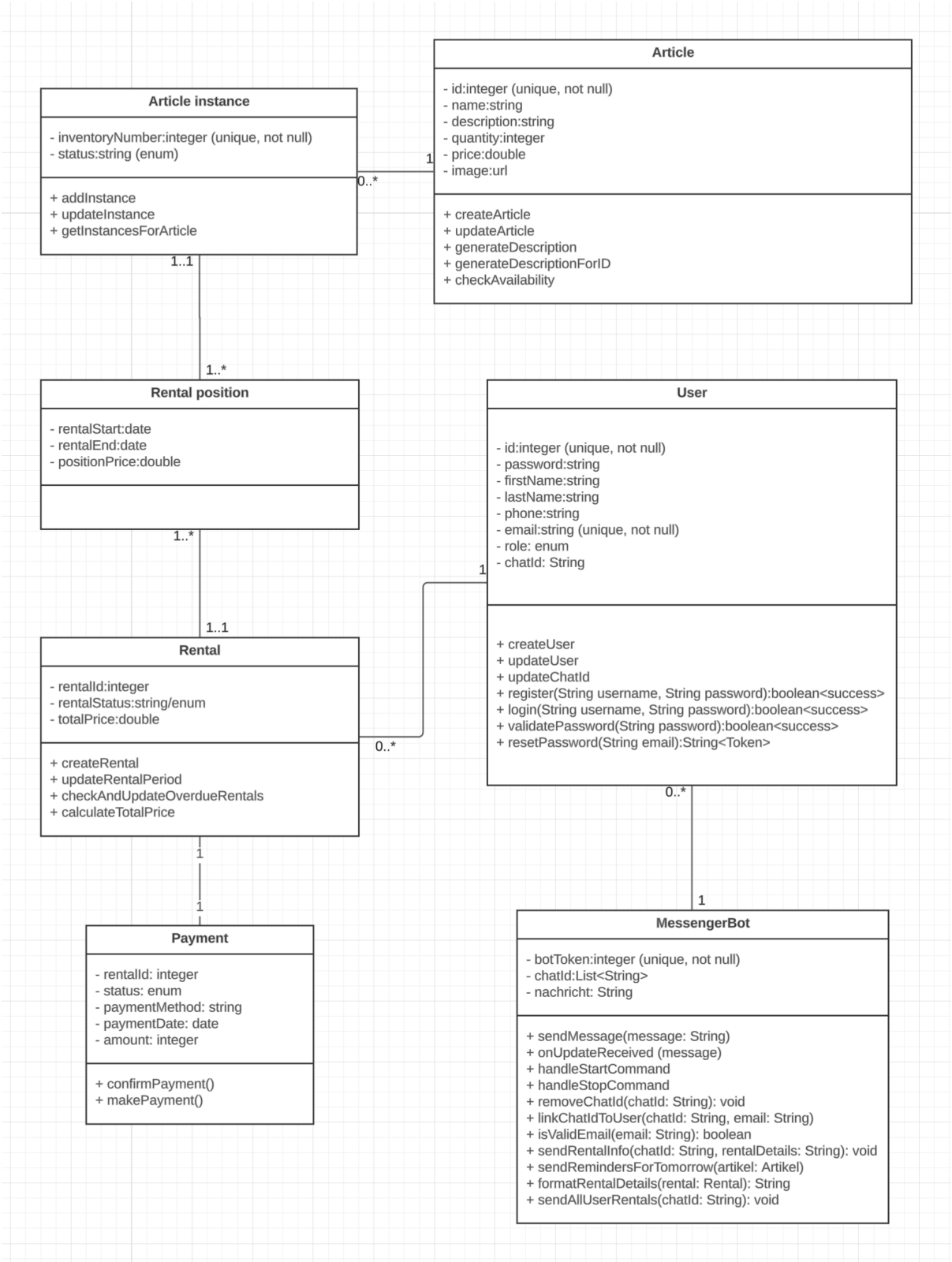


Figure 1: Domainmodell

2.2 User-Stories

2.2.1 Artikel anlegen und verwalten (Daniel)

Admins verwalten Artikel, indem sie diese mit den Attributen Name, Beschreibung, Preis, Bildern und Verfügbarkeiten (Kalender) erstellen, bearbeiten oder löschen. Dadurch wird sichergestellt, dass Mieter:innen stets aktuelle Informationen zu den verfügbaren Artikeln erhalten.

Akzeptanzkriterien Admins legen neue Artikel an, indem sie einen Namen, eine Beschreibung, einen Preis, Bilder und die Verfügbarkeiten im Kalender definieren. Sie können bestehende Artikel bearbeiten und dabei alle Attribute ändern oder den Artikel vollständig aus dem System löschen. Änderungen werden in Echtzeit gespeichert und unmittelbar für Mieter:innen sichtbar.

Ablauf Um einen neuen Artikel anzulegen, öffnen Admins die Verwaltungsoberfläche und starten den Erstellungsprozess. Sie geben den Namen, eine Beschreibung und den Preis ein und laden optional Bilder hoch. Anschließend definieren sie die Verfügbarkeiten des Artikels im Kalender. Nach dem Speichern wird der Artikel automatisch in der Datenbank hinterlegt und für Mieter:innen sichtbar.

Bestehende Artikel werden bearbeitet, indem sie aus der Liste der bereits erstellten Artikel ausgewählt werden. Änderungen an den gewünschten Attributen werden vorgenommen und gespeichert. Die Aktualisierungen werden sofort übernommen, sodass Mieter:innen stets aktuelle Informationen sehen.

Falls ein Artikel nicht mehr benötigt wird, kann er über die Verwaltungsoberfläche gelöscht werden. Nach einer Bestätigung wird der Artikel aus der Datenbank entfernt und ist nicht mehr sichtbar.

Frequenz und Multiplizität Die Verwaltung der Artikel erfolgt regelmäßig, abhängig von neuen Angeboten oder Änderungen an bestehenden Produkten. Mehrere Admins können gleichzeitig Artikel verwalten. Ein Artikel kann mehrere Bilder enthalten und über verschiedene Zeiträume hinweg verfügbar sein. Während Name, Preis und Verfügbarkeiten verpflichtend sind, bleiben Beschreibung und Bilder optional.

2.2.2 Buchung durchführen (Sophie)

Als Mieter:in wählt man einen Artikel aus und gibt ein Start- sowie Enddatum an, um eine Buchung mit berechnetem Mietpreis abzuschließen. Dadurch sichert er:sie sich den gewünschten Artikel für den angegebenen Zeitraum.

Akzeptanzkriterien Ein:e Mieter:in bestimmt das Start- und Enddatum der Buchung und sieht unmittelbar den berechneten Mietpreis, der sich aus der Mietdauer und dem hinterlegten Preis des Artikels ergibt. Anschließend kann er:sie die Buchung bestätigen und abschließen.

Ablauf Mieter:innen rufen die Detailseite eines verfügbaren Artikels auf. Dort wird das gewünschte Start- und Enddatum über die Kalenderfunktion eingegeben. Das System überprüft die Verfügbarkeit und berechnet automatisch den Gesamtmietpreis anhand der

angegebenen Dauer. Ist der Artikel im gewünschten Zeitraum verfügbar, kann der:die Mieter:in die Buchung abschließen.

Nach der Bestätigung wird die Buchung in der Datenbank gespeichert und der Artikel für den gebuchten Zeitraum als nicht verfügbar markiert. Der:die Mieter:in erhält eine Bestätigung über die erfolgreiche Buchung. Sollte der gewählte Zeitraum nicht verfügbar sein, wird der:die Mieter:in entsprechend informiert und kann ein alternatives Datum wählen.

Frequenz und Multiplizität Mieter:innen buchen Artikel abhängig von ihrem individuellen Bedarf. Ein Artikel kann mehrfach hintereinander gebucht werden, jedoch niemals gleichzeitig von verschiedenen Mieter:innen. Die Verfügbarkeiten werden dynamisch verwaltet, sodass neue Buchungen nur in freien Zeiträumen möglich sind.

Datenbezug Die Buchungsdaten werden in der Datenbank hinterlegt und mit den Verfügbarkeiten des Artikels abgeglichen. Der Mietpreis ergibt sich automatisch aus der Differenz zwischen Start- und Enddatum multipliziert mit dem täglichen Mietpreis. Änderungen oder Stornierungen von Buchungen aktualisieren die Verfügbarkeiten in Echtzeit, sodass andere Mieter:innen stets aktuelle Informationen erhalten.

2.2.3 Benachrichtigungen erhalten (Daniel)

Mieter:innen werden über relevante Ereignisse wie Buchungsbestätigungen, Rückgabefristen und Zahlungserinnerungen informiert. Sie erhalten diese Benachrichtigungen über den von ihnen gewählten Kommunikationskanal, beispielsweise per E-Mail oder SMS, um stets auf dem aktuellen Stand zu bleiben.

Akzeptanzkriterien Mieter:innen erhalten Benachrichtigungen zu Buchungsbestätigungen, Rückgabefristen und Zahlungserinnerungen entsprechend ihrer gespeicherten Präferenzen. Die Benachrichtigungen werden rechtzeitig und zuverlässig über den gewählten Kanal zugestellt.

Ablauf Sobald ein buchungsrelevantes Ereignis eintritt, generiert das System eine Benachrichtigung und überprüft, welche Mieter:innen für diese Art von Nachricht registriert sind. Die hinterlegten Präferenzen bestimmen, ob die Benachrichtigung per E-Mail oder SMS versendet wird.

Nach der Zustellung wird die Nachricht als gesendet markiert und optional in der Benutzeroberfläche oder im Posteingang der Mieter:innen angezeigt. Falls eine Nachricht aufgrund technischer Probleme nicht zugestellt werden kann, erfolgt ein erneuter Sendeversuch oder eine alternative Zustellung. Mieter:innen können ihre erhaltenen Benachrichtigungen jederzeit in ihrem Benutzerkonto einsehen.

Frequenz und Multiplizität Die Benachrichtigungen werden je nach Buchungsaktivität der Mieter:innen versendet. Ein:e Nutzer:in kann mehrere Benachrichtigungen für verschiedene Buchungen erhalten. Der Versand erfolgt automatisch und richtet sich nach den gewählten Einstellungen sowie den zeitlichen Ereignissen im Buchungsprozess.

Datenbezug Die Benachrichtigungsdaten werden in der Datenbank gespeichert und basieren auf den in den Buchungen hinterlegten Informationen. Die Zustellung erfolgt in Echtzeit oder zu vordefinierten Zeitpunkten, um sicherzustellen, dass Mieter:innen ihre Buchungen und Fristen rechtzeitig im Blick haben. Zustellprotokolle ermöglichen eine Nachverfolgung und stellen sicher, dass keine wichtigen Benachrichtigungen verloren gehen.

2.2.4 Suchen und Filtern (Sophie)

Mieter:innen durchsuchen das Angebot an Artikeln, indem sie Filter nach Kategorien, Verfügbarkeit und Preis anwenden. Dadurch finden sie schnell passende Artikel, die ihren Anforderungen entsprechen.

Akzeptanzkriterien Mieter:innen können Artikel nach verschiedenen Kriterien filtern, darunter Kategorie, Verfügbarkeit und Preis. Die Such- und Filteroptionen liefern stets korrekte und relevante Ergebnisse, die den ausgewählten Parametern entsprechen.

Ablauf Mieter:innen öffnen die Such- oder Filterfunktion innerhalb der Artikelübersicht. Sie wählen eine oder mehrere Kategorien aus, legen einen Preisspannenfilter fest und prüfen die Verfügbarkeit der Artikel für den gewünschten Zeitraum. Die Ergebnisse aktualisieren sich dynamisch basierend auf den gewählten Kriterien.

Zusätzlich können Mieter:innen ein Suchfeld nutzen, um gezielt nach bestimmten Artikeln oder Schlüsselwörtern zu suchen. Die Suchergebnisse passen sich an die eingegebenen Begriffe an und zeigen relevante Treffer. Sobald die gewünschten Artikel gefunden wurden, können Mieter:innen die Detailansicht aufrufen und eine Buchung vornehmen.

Frequenz und Multiplizität Die Suche und Filterung erfolgen je nach Bedarf der Mieter:innen. Mehrere Filteroptionen können gleichzeitig angewendet werden, um präzisere Ergebnisse zu erhalten. Die Verfügbarkeitsprüfung erfolgt dynamisch, sodass nur buchbare Artikel für den ausgewählten Zeitraum angezeigt werden.

Datenbezug Such- und Filteranfragen greifen auf die Artikeldatenbank zu, um passende Ergebnisse basierend auf den gewählten Kriterien zu liefern. Änderungen an Verfügbarkeiten oder Preisen werden in Echtzeit berücksichtigt, sodass die Ergebnisse stets aktuell sind. Die Filterlogik stellt sicher, dass nur Artikel angezeigt werden, die den Anforderungen der Mieter:innen entsprechen.

3 Backend

3.1 Architektur der Backend-Komponenten (Sophie/Daniel)

Die Anwendung nutzt Java und das Spring Framework als Backend. Die Architektur ist modular aufgebaut und folgt dem Prinzip der klaren Trennung von Verantwortlichkeiten. Die einzelnen Module der Anwendung sind in getrennte Verzeichnisse unterteilt. Wir haben uns dabei möglichst an Standards für die Verwendung von Spring gehalten.

- **API-Schicht:** Das Verzeichnis `api` enthält sämtliche REST-Controller (z.B. `ArticleController`, `RentalController`, `UserController` `etc.`), die als Schnittstelle

zur Außenwelt fungieren. Die Controller empfangen HTTP-Anfragen, leiten diese an die entsprechende Logik weiter und liefern die Ergebnisse im JSON-Format an den Client zurück (Domingues, 2024). Das Angular-Frontend kommuniziert mit dieser API-Schicht über HTTP-Requests (GET, POST, PUT, DELETE), welche die entsprechenden Service-Methoden im Backend ansprechen.

- **Datenübertragung und Mapping:** Das Verzeichnis `dto` enthält Data Transfer Objects, die als reine Informationsträger dienen und keine Geschäftslogik enthalten. Sie ermöglichen den Austausch von Daten zwischen den verschieden Schichten, ohne dabei geschützte Informationen direkt preiszugeben (Tacu & Williams, 2024). Die Mapper-Klassen (z.B. `ArticleMapper`) übernehmen die Umwandlung von Entitäten in DTOs und umgekehrt. Sie sorgen dafür, dass nur die für den Anwendungsfall relevanten Daten an den Client übermittelt werden, während interne Details – insbesondere sicherheitskritische Informationen – ausgeblendet bleiben. So enthält zum Beispiel das User DTO weder das Passwort noch eine Zugriffsmethode darauf, obwohl die zugehörige Entität diese Daten intern verwaltet.

- **Entitäten:** Im `entity`-Verzeichnis befinden sich zentrale Klassen wie `Rental`, `RentalPosition`, `Article` und `User`. Diese Klassen repräsentieren die Datenstruktur der Objekte und und modellieren die Beziehungen zueinander (z.B. ein Nutzer hat mehrere Ausleihen). Sie sind als sogenannte Entities gemäß der JPA-Spezifikation gestaltet und mit Annotationen wie `@Entity`, `@Id` oder `@ManyToOne` versehen, welche die Zuordnung zu einer relationalen Datenbank ermöglichen. Jede Entität repräsentiert dabei eine Tabelle in der Datenbank, und jede Instanz dieser Klasse entspricht einer Zeile innerhalb dieser Tabelle (Balasubramaniam & Vilao, 2024).

- **Datenzugriff:** Die Repositories im Verzeichnis `repo` (z.B. `RentalRepo`, `ArticleRepo` oder `UserRepo`) kapseln den Datenzugriff. Sie basieren auf Spring Data JPA und erfordern keine eigene Implementierung: Durch das Erweitern von Interfaces wie `CrudRepository` oder `JpaRepository` stellt Spring zur Laufzeit automatisch eine vollständige Implementierung bereit, inklusive gängiger CRUD-Operationen (Create, Read, Update, Delete) (Paraschiv & Avramović, 2025). Die Auswahl des Repository-Typs hängt vom jeweiligen Anwendungsfall ab:

  - `CrudRepository` wird bei einfachen Zugriffsmustern verwendet, z. B. im `UserRepo`, das primär Basisoperationen wie `findByEmail` bereitstellt.
  - `JpaRepository` kommt zum Einsatz, wenn zusätzliche Funktionalitäten wie Paging, Sortierung oder komplexe Abfragen benötigt werden, z.B. im `RentalPositionRepo` mit eigenen SQL-Abfragen via `@Query`.

- **Logik:** Die im Verzeichnis `service` implementierten Klassen beinhalten die Kernlogik der Anwendung. Services für Artikel, Ausleihen, Authentifizierung und Kategorien stellen die Methoden Datenzugriffe bereit. Sie dienen als zentrale Schnittstelle zwischen den Controllern und den Repositories.

  Dabei wird zwischen Service-Interface und -Implementierung unterschieden: Jedes Service-Interface (z.B. `ArticleService`) beschreibt die verfügbaren Methoden, während die Implementierungsklasse (z.B. `ArticleServiceImpl`) die konkrete Logik enthält. Das soll eine lose Kopplung ermöglichen und die Testbarkeit verbessern. Außerdem

kennt der Controller so lediglich das Interface und muss keine Details der konkreten Implementierung kennen. Dadurch lassen sich alternative Implementierungen leichter einführen oder austauschen (Datta, 2020).

- **Ereignisverwaltung und Benachrichtigung:** Mithilfe des `events`-Pakets (z.B. `RentalCreatedEvent`, `UserUpdatedEvent`) und des `listener`-Pakets (z.B. `NotificationListener`) wird die asynchrone Verarbeitung von Ereignissen in der Anwendung realisiert. Änderungen führen zum Auslösen von Ereignissen, auf die andere Komponenten reagieren und beispielsweise Benachrichtigungen versenden.

- **Scheduler:** Im Verzeichnis `Scheduler` befinden sich Klassen, die für sich wiederholende Aufgaben genutzt werden. Mithilfe von cron ist dort zum Beispiel eine Klasse implementiert, welche täglich auf Ausleihen für den nächsten Tag prüft und Benachrichtungen für alle gefundenen auslöst. Ebenso wird um 0 Uhr geprüft, ob es Ausleihen gibt, die noch den Status `Active` und deren Enddatum abgelaufen ist. Diesen werden dann auf Overdue gesetzt.

- **Utils** Das Verzeichnis `utils` enthält Hilfsklassen, die spezifische Aufgaben übernehmen und die Hauptlogik entlasten sollen.

  - `JWTUtil` bietet Funktionen zur Erstellung, Validierung und Verarbeitung von JSON Web Tokens (JWT). Damit können Benutzerinformationen (wie die Benutzer E-Mail) aus Tokens extrahiert, neue Tokens generiert und bestehende Tokens auf ihre Gültigkeit überprüft werden.
  - `ResponseHandler` standardisiert den Umgang mit HTTP-Antworten innerhalb der Controller. Mittels generischer Methoden wird die Verarbeitung von erfolgreichen Ergebnissen und Exceptions vereinheitlicht. Dadurch reduziert sich der wiederholte Einstaz von try-catch Blöcken in den Controllern.
  - `ArticleSpecification` stellt dynamische Filterkriterien für Datenbankabfragen bereit, basierend auf der JPA Specification-API. Diese API erlaubt es, Bedingungen (z.B. Preisbereich, Verfügbarkeit oder Kategorien) zu definieren, ohne für jede Variante eine eigene Repository-Methode zu schreiben. Die Spezifikationen werden dabei als separate Objekte definiert und können flexibel kombiniert werden (Spring, n.d.).

    So wird z.B. mittels `isAvailableBetween()` geprüft, ob ein Artikel in einem Zeitraum nicht durch eine Buchung blockiert ist. `hasCategory()` filtert Artikel nach bestimmten Kategorien während `hasPriceBetween()` eine Preisfilterung ermöglicht.

- **Configuration:** Im Verzeichnis configuration befinden sich zentrale Klassen, die sicherheitsrelevante Aspekte der Anwendung sowie die Initialisierung der Datenbank steuern.

  - `WebSecConfig`: Die Klasse definiert die Sicherheitskonfiguration für den Webzugriff. Sie legt fest, welche Endpunkte öffentlich zugänglich sind und welche eine Authentifizierung erfordern. Zudem wird der JWT-Filter in die Filterkette eingebunden, um Token-basierte Authentifizierung zu ermöglichen.
  - `JwtAuthFilter`: Dieser Filter wird bei jedem HTTP-Request ausgeführt und prüft, ob ein gültiger JWT im Authorization-Header enthalten ist. Ist dies der

Fall, wird der Benutzer authentifiziert und seine Identität im SecurityContext der Anwendung hinterlegt. Dadurch können geschützte Ressourcen nur mit gültigem Token aufgerufen werden.

- `CorsFilter`: Die Komponente ermöglicht die Cross-Origin Resource Sharing Konfiguration. Sie erlaubt Anfragen von beliebigen Ursprüngen (dynamisch basierend auf dem Origin-Header der Anfrage) und gibt an, welche HTTP-Methoden und Header zugelassen sind. Damit wird der Zugriff auf die API auch von externen Frontends ermöglicht.
- `DatabaseSeeder`: Diese Komponente wird beim Start der Anwendung ausgeführt und füllt die Datenbank mit Testdaten. Dazu gehören unter anderem vordefinierte Benutzer, Kategorien, Artikel und Mietvorgänge.

Der Vollständigkeit halber sei erwähnt, dass die im Verzeichnis configuration enthaltenen Komponenten in ihrer aktuellen Form vor allem für Test- und Entwicklungszwecke vorgesehen sind und nicht für eine produktive Umgebung.

3.2 Design-Pattern

3.2.1 Observer Pattern (Daniel)

Wir haben das Observer Pattern eingesetzt, um die Benachrichtigungssysteme – sowohl den E-Mail-Versand als auch die Telegram-Nachrichten – von der restlichen Logik zu entkoppeln. Konkret nutzen wir hierfür den Spring Event-Mechanismus, mit dem Prozesse, wie das Aktualisieren eines Nutzers oder das Erstellen einer Ausleihe, ihre Aktionen über Events kommunizieren. Ohne dieses Pattern müssten die Services den Versand von Benachrichtigungen direkt steuern, was zu einer starken Verknüpfung zwischen der Kernlogik und den Benachrichtigungssystemen führen würde. Dies erschwert Änderungen, da Anpassungen am Benachrichtigungssystem auch den restlichen Code betreffen würden, was wiederum die Wartbarkeit und Testbarkeit einschränkt.

Durch das Observer Pattern wird dieses Problem vermieden. Wenn beispielsweise ein Nutzer aktualisiert oder eine Ausleihe erstellt wird, veröffentlicht der jeweilige Service (wie `UserService` oder `RentalService`) ein entsprechendes Event, etwa `UserUpdatedEvent` oder `RentalCreatedEvent`. Diese Events werden von einem separaten Listener, dem `NotificationListener`, registriert und verarbeitet. Der Listener entscheidet dann, ob und wie der Benutzer benachrichtigt wird – entweder per E-Mail oder über den `MessengerBot` für Telegram.

Dies bietet mehrere Vorteile: Änderungen an der Benachrichtigungslogik wirken sich nicht direkt auf den Kerncode aus, und es ist einfach, weitere Benachrichtigungskanäle – etwa für SMS oder Push-Mitteilungen – zu integrieren, da einfach weitere Listener hinzugefügt werden können oder ein listenere mehrere Events auslösen kann. Außerdem verbessert sich die Testbarkeit, da die Ausleihprozesse isoliert von der Benachrichtigung geprüft werden kann, und durch die Möglichkeit zur asynchronen Verarbeitung über Spring Events wird auch die Performance der Anwendung gesteigert und die Anwendung nicht blockiert.2

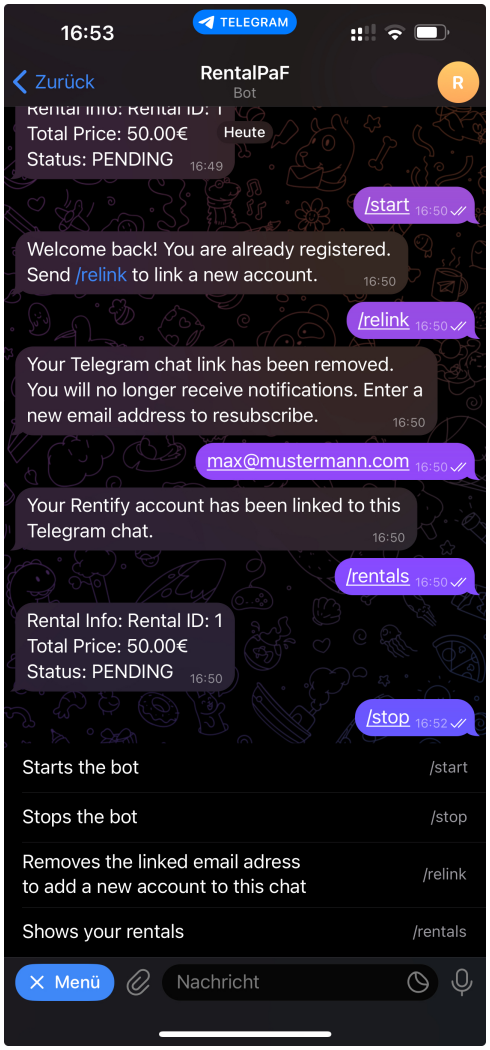


Figure 2: Testphase des Telegram Bots

3.2.2 Composite Pattern (Daniel)

Desweiteren setzen wir das Composite Pattern ein, um die Struktur von Ausleihen übersichtlich abzubilden. Ein `Rental` besteht aus mehreren `RentalPosition`-Objekten, was die Teil-Ganzes-Beziehung darstellt. So können wir die gesamte Ausleihe als eine Einheit behandeln, während wir gleichzeitig die Details der einzelnen Positionen berücksichtigen.

Der Hauptvorteil dieses Ansatzes liegt darin, dass Operationen wie das Berechnen des Gesamtpreises auf einfache Weise umgesetzt werden können – indem man die Preise aller enthaltenen `RentalPositionen` zusammenrechnet. Dadurch entfällt die Notwendigkeit, jede Position separat zu bearbeiten, was den Code übersichtlicher macht.

3.2.3 Stratgey Pattern (Sophie)

Im Rahmen der Implementierung unseres Authentifizierungs- und Registrierungsprozesses bestand die Herausforderung darin, eine flexible und wartbare Architektur zu schaffen, die verschiedene Authentifizierungs- und Validierungsstrategien austauschbar macht. Ziel war es, den Authentifizierungsprozess nicht nur auf aktuelle Anforderungen auszulegen, sondern ihn auch erweiterbar für zukünftige Szenarien zu gestalten – beispielsweise Passwort-Zurücksetzen via Link.

Das Strategy Pattern ermöglicht es, unterschiedliche Prozesse in jeweils eigenen Klassen zu kapseln und zur Laufzeit dynamisch auszutauschen (Shvets, n.d.). Dadurch kann flexibel zwischen verschiedenen Arten der Authentifizierung oder Token-Validierung gewechselt werden, ohne die aufrufende Logik anpassen zu müssen.

In unserer Anwendung wurden folgende Strategien implementiert:

- `AuthenticationStrategy`: Definiert unterschiedliche Wege zur Authentifizierung. `MagicLinkAuthenticationStrategy` implementiert das Interface und versendet einen Magic Link zur Authentifizierung
- `MagicLinkStrategy`: Ermöglicht unterschiedliche Versandlogiken für Magic Links, abhängig vom Nutzungskontext (z.B. Registrierung, Einladung).
  `RegistrationMagicLinkStrategy` implementiert das Interface und sendet einen Bestätigungslink zur Aktivierung eines neuen Benutzerkontos.
- `TokenValidationStrategy`: Definiert flexible Validierungsstrategien für Tokens, z.B. für Kontoaktivierung oder einmaligen Login.
  `RegistrationTokenValidationStrategy` implementiert das Interface und validiert einen Registrierungstoken, aktiviert das Benutzerkonto und leitet auf das Frontend weiter.

Alle Strategien implementieren jeweils ein gemeinsames Interface, welches als Strategy fungiert. Die konkreten Klassen (sog. Concrete Strategies) enthalten die jeweilige Logik und können beliebig erweitert oder ausgetauscht werden. Diese Entkopplung isoliert die Implementierungsdetails vom restlichen Code und erhöht somit die Wartbarkeit und Flexibilität. Neue Strategien können eingeführt werden, ohne die bestehende Geschäftslogik anpassen zu müssen (Open/Closed Principle) (Shvets, n.d.).

3.2.4 Facade Pattern (Sophie)

Innerhalb der Authentifizierungslogik mussten mehrere miteinander verknüpfte Schritte umgesetzt werden – darunter das Setzen des SecurityContext, die Validierung von Tokens sowie die Generierung von JWTs. Jeder dieser Schritte war auf verschiedene Services und Sicherheitskomponenten angewiesen die der Controller direkt ansprechen musste, was den Code schwerer wartbar gemacht hat.

Da es in diesem Fall um das koordinierte Zusammenspiel mehrerer Services und Sicherheitsmechanismen ging, haben wir das Facade-Pattern angwendet, um eine einheitliche Schnittstelle für den gesamten Authentifizierungsprozess zu schaffen, ohne dass der Controller die dahinterliegende Komplexität kennen oder steuern muss. In unserer Anwendung kapselt `AuthenticationFacade` die Authentifizierungslogik und stellt dem Controller eine zentrale Schnittstelle zur Verfügung – z.B. für den Start eines Authentifizierungsprozesses oder die Validierung eines Tokens.

3.3 Spezielle Themen

3.3.1 Database Seeder (Sophie/Daniel)

Der DatabaseSeeder ist eine CommandLineRunner-Komponente, die initial beim Start der Anwendung ausgeführt wird, um eine Beispiel-Datenbank mit Benutzern, Artikeln und Ausleihen zu befüllen. Das hilft uns bei der Entwicklung und beim Testen der Applikation indem folgende Schritte ausgeführt werden:

Benutzererstellung Die Methode ‘`seedUsers()`‘ prüft, ob die vordefinierten Benutzer bereits existieren. Falls nicht, werden sie mit einer Rolle und einem verschlüsselten Passwort gespeichert. Bereits bestehende Benutzer werden übersprungen, um Duplikate zu vermeiden.

Artikel- und Kategoriegenerierung Die Methode ‘`seedArticles()`‘ erstellt eine hierarchische Struktur von Kategorien, in der jede Hauptkategorie mehrere Unterkategorien besitzt. Für jede Unterkategorie werden Artikel generiert, die dann mit Artikelinstanzen versehen werden. Diese Instanzen sind konkrete Ausleihobjekte mit einem eindeutigen Inventarnummern-System.

Erstellung von Beispiel-Ausleihen Die Methode ‘`seedRentals()`‘ generiert Test-Ausleihen für bestehende Benutzer. Dabei werden Artikelinstanzen bestimmten Benutzern zugeordnet, mit einem Mietzeitraum versehen und als RentalPositionen gespeichert.

3.3.2 Externe API (Daniel)

Wir haben uns dazu entschieden, Gemini als Large language model einzubinden, um die Möglichkeit zu schaffen, den Beschreibungstext auf Knopfdruck generieren zu lassen.

Die Beschreibungserstellung erfolgt über die Methoden `generateDescription` und `generateDescriptionForName` durch eine Anfrage an das Gemini-Modell. Dabei wird der Name des Artikels als Eingabe verwendet und in einem passenden Kontext formuliert. Die generierte Beschreibung kann entweder direkt in der Datenbank gespeichert oder als Antwort zurückgegeben werden.

Im Frontent ist dies über einen Button bei Anlegen oder Bearbeiten eines Artikels verfügbar.

3.4 Tests (Daniel)

Wir haben einige Tests geschrieben, um verschiedene Teilsysteme im Back- und Frontend unabhängig testen zu können. Einige Dinge, wie zum Beispiel die Telegram-Integration lässt sich mit unseren Möglichkeiten nicht End-to-End automatisiert testen. Daher gibt es dafür zum Beispiel im ReminderScheduler eine auskommentierte Zeile, die man für Tests nutzen kann.

3.4.1 SpringBootTests

- `EmailServiceTest` In diesem Test wird die grundlegende Funktionalität des EmailService überprüft. Dabei wird der Spring Application Context geladen und der EmailService in den Test injiziert, um sicherzustellen, dass dieser korrekt konfiguriert ist. Im Test wird die Methode `sendEmail` aufgerufen, um eine E-Mail an eine Testadresse mit einem definierten Betreff und Inhalt zu versenden. Ziel des Tests ist es, zu verifizieren, dass der E-Mail-Versand fehlerfrei abläuft und keine Fehler geworfen werden. Da die Methode keinen Rückgabewert liefert, gilt der Test als bestanden, wenn der Versandprozess ohne Fehler durchgeführt wird.

- `PaymentServiceTest` In diesem Test wird überprüft, ob der `PaymentService` Zahlungen korrekt erstellt, bestätigt und validiert. Dabei wird ein Mock des `PaymentRepo` verwendet, um das Verhalten der Datenbank zu simulieren.

  Es wird getestet, ob eine neue Zahlung mit dem Status `PENDING` gespeichert wird und ob eine bestehende Zahlung korrekt auf `COMPLETED` gesetzt werden kann. Zudem werden Fehlerfälle geprüft, etwa wenn eine Zahlung nicht existiert oder bereits storniert wurde. Durch Assertions und die Verifizierung von Methodenaufrufen wird sichergestellt, dass der Service erwartungsgemäß reagiert.

- `ReminderTest` In diesem Test wird überprüft, ob das System rechtzeitig Erinnerungen für anstehende Ausleihen versendet. Zunächst wird die Datenbank mit Testdaten befüllt, wobei sichergestellt wird, dass mindestens eine Ausleihe für den nächsten Tag existiert.

  Anschließend wird ein Mock des `MessengerBot` verwendet, um zu testen, ob die Methode `sendRemindersForTomorrow` aufgerufen wird. Durch die Verifikation der Methodenaufrufe wird bestätigt, dass die Erinnerungsfunktion korrekt ausgelöst wird.

- `RentalNotificationListenerTest` In diesem Test wird überprüft, ob der `NotificationListener` nach der Erstellung einer Ausleihe eine Benachrichtigung über den `MessengerBot` versendet. Dazu wird ein `RentalCreatedEvent` ausgelöst, das vom Event-Publisher verarbeitet wird.

  Ein Mock des `MessengerBot` stellt sicher, dass die Methode `sendRentalInfo` mit den korrekten Parametern aufgerufen wird. Durch das Erfassen der Chat-ID und der Nachricht wird überprüft, ob die Benachrichtigung den richtigen Benutzer erreicht und die erwarteten Inhalte enthält.

- `RentalServiceTest` In diesem Test wird überprüft, ob der `RentalService` Mietvorgänge mit mehreren Artikeln korrekt erstellt und speichert. Dazu werden Testbenutzer und Artikel angelegt, aus denen anschließend `RentalPositionen` mit verschiedenen Mietzeiträumen gebildet werden.

  Beim Erstellen einer Ausleihe wird sichergestellt, dass diese mit dem richtigen Benutzer verknüpft ist, die Mietpositionen korrekt gespeichert werden und der Gesamtpreis stimmt. Durch Assertions wird überprüft, ob alle Objekte persistiert wurden und die Datenstruktur beibehalten wird.

3.4.2 Postman

Wir haben Postman für Tests unserer REST-Schnittstelle genutzt. Dabei wurden alle CRUD-Operationen mithilfe von POST, PUT, GET und DELETE durchgeführt. Eine Postman-Collection mit Beispielen befindet sich auch im Repository.3

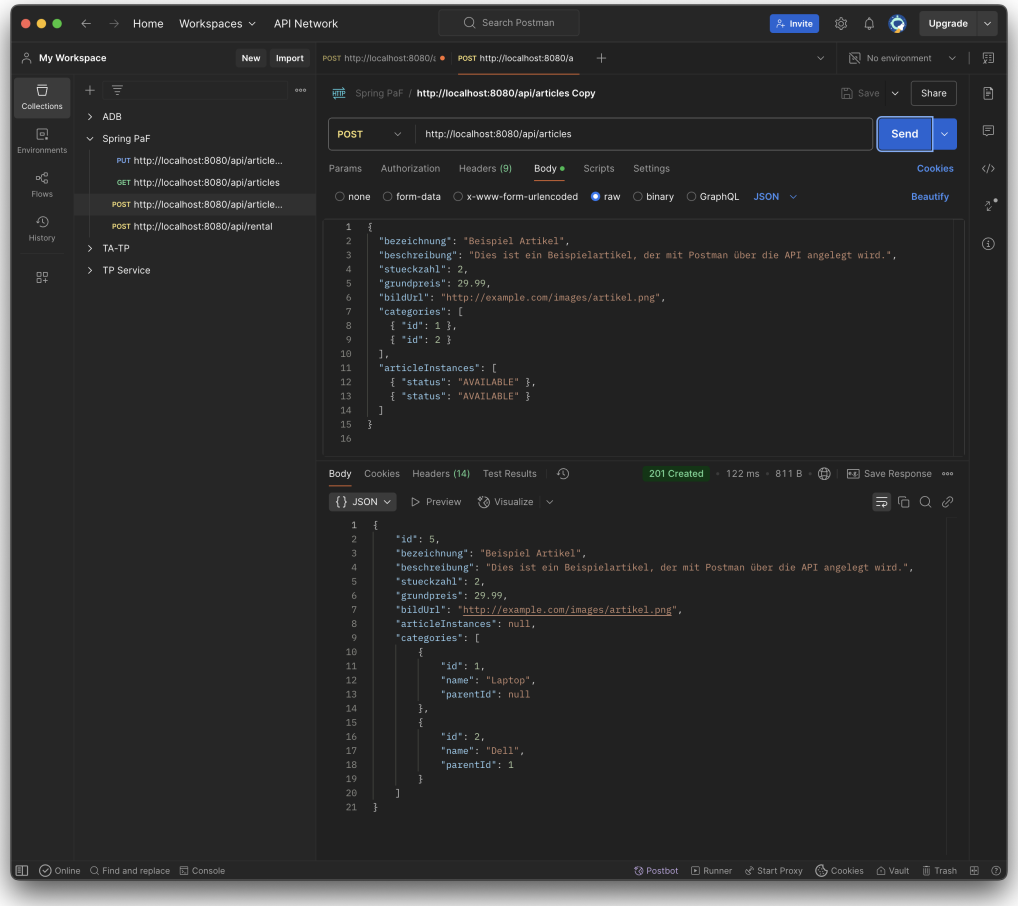


Figure 3: Exemplarisch - HTTP POST, um das Anlegen eines Artikels zu testen

4 Frontend (Sophie)

Das Frontend wurde mit Angular in Typescript realisiert. Um es zu starten muss über das terminal in PaF/frontend gewechselt werden und es mit `ng serve` gestartet werden. Anschließend kann die Weboberfläche über `localhost:4200` erreicht werden.

4.1 Layout

Die Benutzeroberfläche unserer Anwendung gliedert sich in mehrere Seiten mit klar definierten Funktionen. Wiederkehrende Layout-Elemente wie Header, Menü und Footer wurden zentral implementiert.

Die Startseite informiert über den Zweck des Services und bietet einen ersten Überblick über das Angebot.

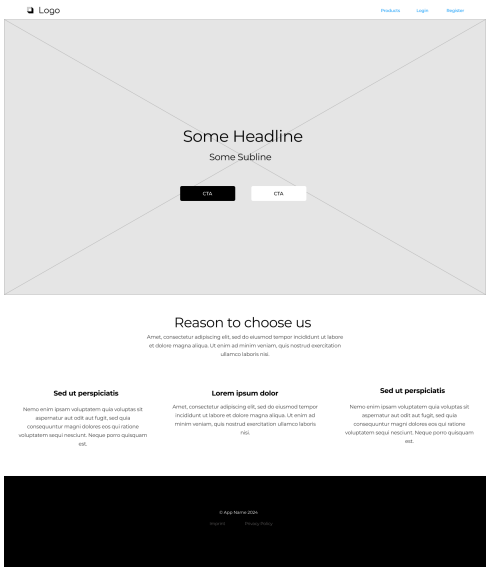


Figure 4: Mockup der Startseite

Die Produktübersicht zeigt alle verfügbaren Artikel; eine Filterfunktion ermöglicht die gezielte Eingrenzung nach Kategorien.

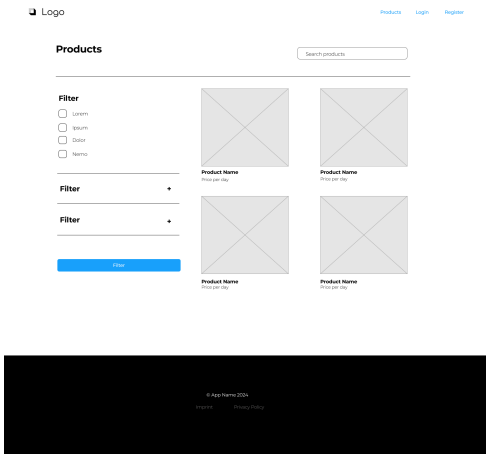


Figure 5: Mockup der Produktübersicht mit Filterfunktion

Auf der Product-Detail-Page (PDP) können Nutzer:innen Artikelinformationen einsehen sowie Verfügbarkeiten prüfen und Artikel dem Warenkorb hinzufügen.

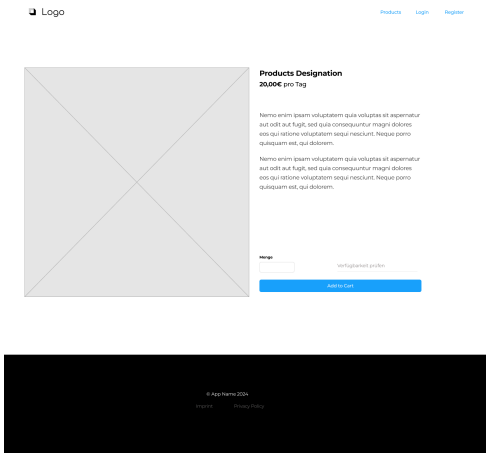
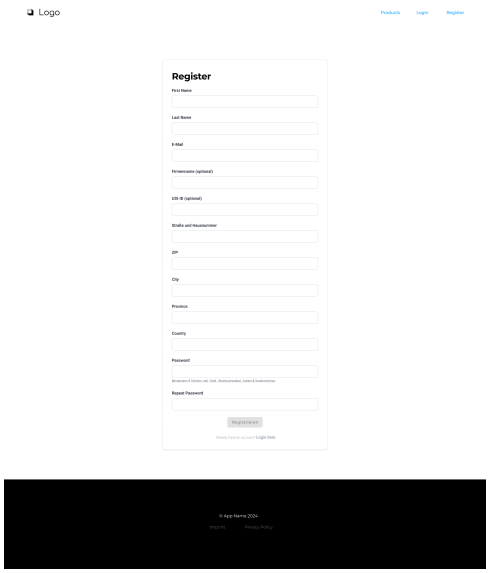
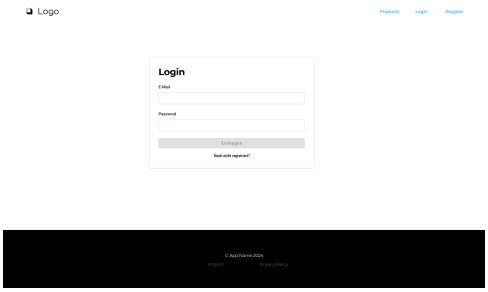


Figure 6: Mockup der PDP für Kund:innen

Um Artikel in den Warenkorb legen zu können, ist eine Registrierung bzw. Anmeldung erforderlich. Separate Seiten stehen hierfür zur Verfügung.



(a) Mockup der Registrierungsseite



(b) Mockup der Login-Seite

Figure 7: Mockups Registrierung und Login

Warenkorb und Buchung

Der Warenkorb ermöglicht es Kund:innen, ausgewählte Artikel zunächst zu sammeln und anschließend gemeinsam zu buchen. Im Warenkorb werden alle ausgewählten Artikel mit dem geplanten Mietzeitraum, Preis der einzenen Positionen und dem Gesamtpreis angezeigt. Die finale Buchung erfolgt aus dem Warenkorb heraus.

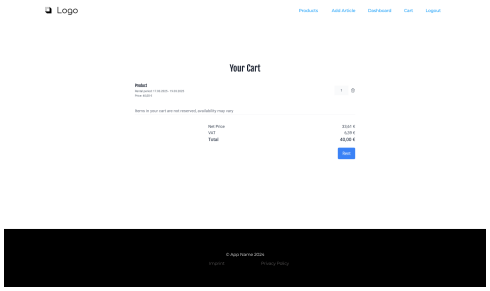
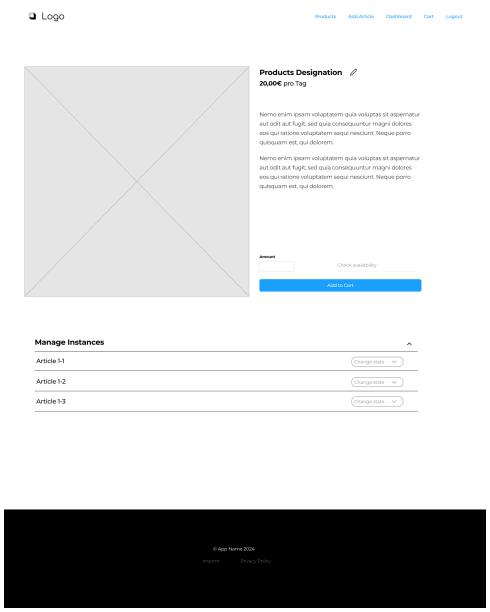


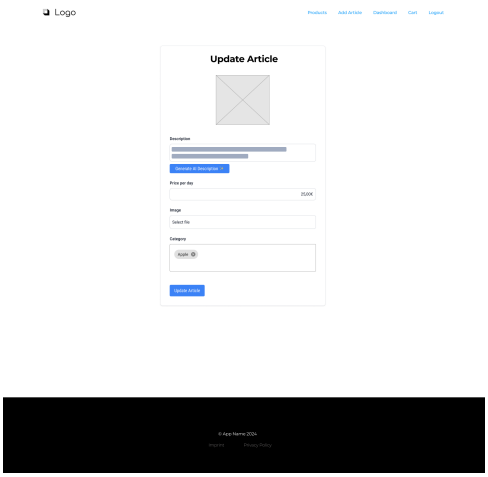
Figure 8: Mockup des Warenkrobs

Admin-Funktionen

Admins haben auf der PDP zusätzlich die Möglichkeit, den Status einzelner Artikelinstanzen zu ändern oder weitere Instanzen hinzuzufügen. Außerdem steht ihnen ein Bearbeitungsbutton zur Verfügung, über den sie zu einer separaten Seite gelangen, auf der Artikelbeschreibung, Preis, Kategorien und das Produktbild bearbeitet werden können.



(a) Mockup der PDP für Administrator:innen



(b) Mockup Artikel bearbeiten

Figure 9: Mockups der Admin-Oberflächen zum Bearbeiten von Artikeln und Artikelinstanzen

Darüber hinaus existiert eine weitere Seite zum Anlegen neuer Artikel, die über einen eigenen Menüpunkt aufgerufen werden kann.

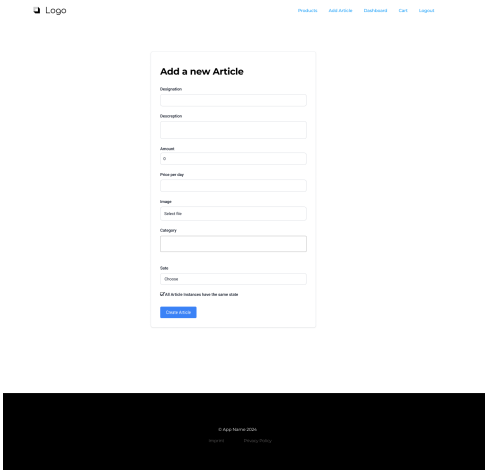


Figure 10: Mockup der Seite zum Artikel anlegen

Dashboard

Ein Dashboard soll Kund:innen eine Übersicht über laufende und vergangene Ausleihpositionen bieten, die bei Bedarf angepasst oder storniert werden können. Admins können darüber hinaus Ausleihpositionen verwalten und Änderungen an bestehenden Buchungen vornehmen.

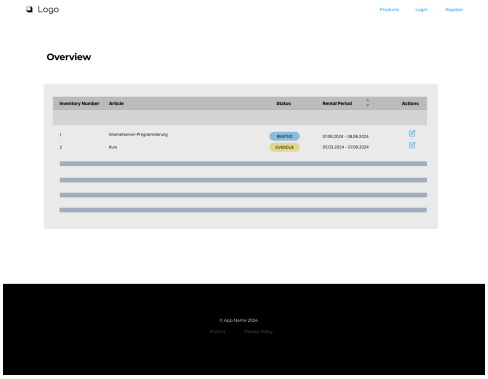


Figure 11: Mockup des Dashboards mit Ausleihübersicht

4.2 Projektstruktur

Die Angular-Anwendung ist modular aufgebaut und folgt einem Trennungskonzept zwischen rollenbasierten Modulen, Authentifizierungs-Logik und gemeinsam genutzten Funktionalitäten. Der `shared/`-Ordner enthält Services, Komponenten und Guards, die von mehreren Modulen genutzt werden. Dabei handelt es sich nicht ausschließlich um klassische Wiederverwendung im Sinne generischer Komponenten, sondern auch um zentralisierte Funktionalität (z.B. `AuthService`, `CartService`), die projektweit benötigt wird.

4.3 Frontend Logik

4.3.1 API-Zugriffe

Sensible HTTP-Anfragen (z.B. zur Artikelverwaltung) erfordern eine Authentifizierung per JWT-Token. Dieser wird über den `StorageService` im Local Storage gespeichert

und kann global per ausgelesen werden. Jeder schreibende Service nutzt eine zentrale Methode `createAuthHeader()` zur Erzeugung eines `Authorization`-Headers.

Listing 1: JWT Header-Erzeugung

```
private createAuthHeader(): HttpHeaders {
  return new HttpHeaders().set('Authorization', 'Bearer ' +
      StorageService.getToken());
}
```

Die Services binden den Header in API-Aufrufe ein. Beispiel: `ArticleService` schützt Endpunkte wie `createArticle()`, `generateDescriptionForName()` durch den Header:

```
createArticle(article: Article): Observable<Article> {
  return this.http.post<Article>(this.apiUrl, article, {
    headers: this.createAuthHeader()
  });
}
```

Lesende Endpunkte (z.B. `getArticles()`) sind öffentlich und benötigen keinen Header. Der Schutz erfolgt serverseitig, sodass unautorisierte Anfragen abgelehnt werden. Das soll sicherstellen, dass nur berechtigte Rollen wie `ADMIN` oder `STAFF` API-Funktionen nutzen können. Guards wie `AuthGuard` und `AdminGuard` verhindern zudem unbefugte UI-Zugriffe.

Zur Vermeidung unbefugter UI-Zugriffe wurden ergänzend zwei sogenannte **Guards** implementiert, welche Angulars Interface `CanActivate` verwenden. Diese Guards prüfen bereits vor dem Routing, ob ein Nutzer angemeldet ist und/oder die benötigte Rolle besitzt, und verhindern so die Sichtbarkeit von geschützten Seiten oder Funktionen (Angular, n.d.-a).

4.3.2 Datenmodelle

Im Frontend erfolgt die Modellierung der Anwendungsdaten über Interfaces, welche die Form der ausgetauschten Objekte typisieren. Diese Interfaces dienen als Data Transfer Objects (DTOs) - also als strukturierte Datentransferobjekte, die zur Übertragung zwischen Frontend und Backend genutzt werden.

Praktisches Beispiel: AdminRentalInfoDto Das `AdminRentalInfoDto` wird im Adminbereich zur Anzeige und Verwaltung von Mietpositionen verwendet und enthält relevante Miet-, Artikel- und Nutzerinformationen.

Listing 2: DTO zur Darstellung von Mietpositionen im Admin-Modul

```
export interface AdminRentalInfoDto {
       rentalPositionId: number;
       rentalStart: string;
       rentalEnd: string;
       positionPrice: number;
       userEmail: string;
       userId: number;
       userFirstName: string;
```

```
    userLastName: string;
    articleDesignation: string; articleInstanceInventoryNumber: string;
    status: string;
    newRentalStart?: string;
    newRentalEnd?: string;
}
```

Das DTO wird im `AdminService` genutzt, um Daten zwischen Backend und UI-Komponenten zu übertragen. Beispielhafte Verwendung:

Listing 3: Nutzung des DTOs zur Aktualisierung einer Mietinstanz

```
markInstanceAsRepaired(rental: AdminRentalInfoDto) {
if(rental.rentalPositionId !== 0 {
    this.adminService.updateInstanceStatus(rental.rentalPositionId,
    'AVAILABLE').subscribe(...);
    }
    ...
}
```

Weitere Datenmodelle

- **Article:** Beschreibt einen Artikel mit Feldern wie Bezeichnung, Beschreibung, Grundpreis, Bild-URL, zugewiesenen Kategorien sowie einer Liste an ArticleInstances.
- **ArticleInstanceDto:** Modelliert eine einzelne Artikelinstanz mit Status (z.,B. `VERMIETET`, `AVAILABLE`) und optionaler Inventarnummer. Wird z.,B. für Statusupdates oder Löschoperationen verwendet.
- **CartItem:** Dient der Client-seitigen Speicherung von Warenkorbeinträgen im Local Storage. Enthält Artikel-IDs, Mietzeitraum, Tagespreis und Stückzahl.
- **RentalPositionDto:** Wird für Backend-Anfragen zur Mietabwicklung verwendet. Enthält Mietzeitraum, Preis und zugeordnete Artikelinstanz-ID.

4.3.3 Hilfsfunktionen

Hilfsfunktionen sind innerhalb der jeweiligen Services deklariert. Sie erfüllen Aufgaben wie Header-Erstellung, Datumskonvertierung oder Datenmanipulation.

JWT-Header Zur Authentifizierung gegen geschützte API-Endpunkte erzeugt jede Service-Klasse über eine eigene Methode einen Authorization-Header basierend auf dem im Local Storage gespeicherten Token. Diese Methode kommt z.,B. im `ArticleService`, `AdminService`, `CategoryService` usw. zum Einsatz.

Listing 4: Funktion zur Header-Erzeugung

```
private createAuthHeader(): HttpHeaders {
    return new HttpHeaders()
        .set('Authorization',
        'Bearer ' + StorageService.getToken());
```

```
    }
```

Datumskonvertierung für API-Kompatibilität Im `AdminService` und `CartService` werden Methoden genutzt, um JavaScript-`Date`-Objekte in das ISO-Format `YYYY-MM-DD` zu konvertieren, um diese korrekt an das Backend zu übermitteln oder im Local Storage zu speichern.

Listing 5: Formatierung von Datum als ISO-String (AdminService)

```
formatToLocalDate(dateInput: string | Date): string {
    const date = (dateInput instanceof Date) ?
        dateInput : new Date(dateInput);
    const year = date.getFullYear();
    const month = String(date.getMonth() + 1).padStart(2, '0');
    const day = String(date.getDate()).padStart(2, '0'); return
        ${year}-${month}-${day};
}
```

Deserialisierung im Warenkorb Im `CartService` sorgt eine Hilfsfunktion dafür, dass gespeicherte Date-Strings beim Laden aus dem Local Storage korrekt zurück in Date-Objekte überführt werden.

Listing 6: Wiederherstellung von Date-Objekten aus JSON

```
private dateReviver(key: string, value: any): any {
    if (key === 'rentalStart' || key === 'rentalEnd') {
        return new Date(value); } return value;
    }
```

4.3.4 Authentifizierung und Token-Verarbeitung

Die Nutzer-Authentifizierung erfolgt über einen sogenannten Magic Link. Nach erfolgreicher Registrierung kann sich der Nutzer über seine E-Mail-Adresse einloggen – ein temporärer Token wird per E-Mail versendet und nach Klick auf den Link lokal gespeichert.

Registrierung Die `RegisterComponent` erlaubt die Eingabe von Nutzer-, Adress- und optionalen Unternehmensdaten. Nach Absenden des Formulars wird eine POST-Anfrage an das Backend gesendet. Bei Erfolg erfolgt eine visuelle Rückmeldung, andernfalls eine Fehlermeldung via Snackbar.

Listing 7: Registrierung eines Nutzers über den AuthService

```
    this.authService.register(payload).subscribe({ next: () => {
        this.mailSent = true;
        this._snackBar.open('Registration successful! Please confirm your
            e-mail ', 'OK'); },
        error: (err) => {
        this._snackBar.open(Error: ${err.error?.message}, '');
```

```
    }
});
```

Login In der `LoginComponent` wird lediglich die E-Mail abgefragt. Anschließend sendet der `AuthService` eine POST-Anfrage, um einen Login-Link an die E-Mail-Adresse zu übermitteln. Der Nutzer wird visuell über den Versand informiert.

Token-Verarbeitung der MagicLoginComponent Nach dem Klick auf den Magic Link wird der Nutzer über eine URL mit enthaltenem JWT-Token zur Anwendung zurückgeleitet. In der `MagicLoginComponent` wird dieser Token ausgelesen, dekodiert und zusammen mit den Nutzerinformationen (ID und Rolle) lokal gespeichert.

Listing 8: Token-Handling und Speicherung

```
const token = this.route.snapshot.queryParams['token'];
    StorageService.saveToken(token);
const payload = this.decodeToken(token);
StorageService.saveUser({ id: payload.sub, role: payload.role });
```

Hierbei unterstützt der `StorageService` durch verschiedene Methoden zur Verwaltung und Überprüfung des Tokens. Er ermöglicht:

- das Speichern und Abrufen des Tokens im Local Storage,
- das Ablegen der Nutzerrolle,
- sowie die Prüfung, ob ein `ADMIN` oder `STAFF` eingeloggt ist.

Nach erfolgreichem Login erfolgt eine Weiterleitung basierend auf der Nutzerrolle. So werden administrative Nutzer (`ADMIN` oder `STAFF`) in den Admin-Bereich geleitet, während reguläre Kunden in den Kundenbereich gelangen.

Listing 9: Rollenbasierte Navigation

```
if (user.role === 'ADMIN' || user.role === 'STAFF') {
    this.router.navigateByUrl('/admin');
} else { this.router.navigateByUrl('/customer');
}
```

Die gespeicherten Rolleninformationen werden später von den Guards verwendet, um geschützte Routen abzusichern. Diese Guards prüfen, ob ein gültiger Token vorliegt und ob der Nutzer die erforderlichen Berechtigungen besitzt. Sollte dies nicht der Fall sein, erfolgt eine automatische Weiterleitung – z.,B. zur Login- oder Startseite.

4.3.5 Verarbeitung und Anzeige von Artikeln

Ein zentrale Funktion der Anwendung ist die dynamische Anzeige und Filterung von Artikeln. Hierzu arbeitet die `ProductCategoryComponent` mit dem `ArticleService`, um Artikel aus dem Backend zu laden und auf Basis von Nutzerinteraktionen gezielt zu filtern und darzustellen.

Initiale Datenverarbeitung Beim Initialisieren der Komponente werden alle Artikel sowie die verfügbaren Kategorien geladen. Zusätzlich werden die Minimal- und Maximalpreise aller Artikel ermittelt, um diese als Vorgabewerte für Preisfilter zu verwenden.

Filterlogik und Preisbereich Zur Filterung stehen folgende Kriterien zur Verfügung:

- Auswahl von Kategorien
- Preisbereich (Min/Max)
- Verfügbarkeit im Zeitraum (Start-/Enddatum)

Der Nutzer kann diese Kriterien in einem Formular setzen, woraufhin ein API-Call mit entsprechenden Query-Parametern erfolgt.

Listing 10: Filteranwendung bei Artikelabfrage

```
applyFilters() {
    const filters = this.filterForm.value;
    filters.priceRange = [filters.minPrice, filters.maxPrice];
    this.articleService.getFilteredArticles(filters)
    .subscribe(
        filteredArticles =>
            this.articles = filteredArticles);
}
```

Die Filterung erfolgt serverseitig, wobei Preisgrenzen und verfügbare Kategorien als Query-Parameter übergeben werden.

Anzeige und Navigation Gefilterte Artikel werden visuell als Liste mit Bild, Bezeichnung und Tagespreis angezeigt. Beim Klick auf einen Artikel erfolgt eine Navigation zur Detailansicht, wobei das gewählte Datum (Verfügbarkeit) als URL-Parameter mitgegeben wird.

Listing 11: Navigation zur Detailansicht mit Datum

```
<a
    routerLink="/pdp/article/{{article.id}}"
    [queryParams]="{start: filterForm.get('startDate')?.value,
    end: filterForm.get('endDate')?.value}"
>
    {{ article.bezeichnung }} - {{ article.grundpreis }}
</a>
```

Verfügbarkeitsprüfung (ArtikelService) In der Detailansicht wird geprüft, ob ein Artikel im angegebenen Zeitraum verfügbar ist. Dies erfolgt über den `checkAvailability()`-Aufruf des `ArticleService`. Hierzu werden Datum und Artikel-ID in ein kompatibles Format konvertiert und als Query-Parameter übergeben.

Listing 12: Prüfung der Artikelverfügbarkeit per API

```
checkAvailability(articleId: number, start: Date, end: Date): Observable<{
    available: boolean }> {
```

```
        const params = new HttpParams()
            .set('articleId', articleId.toString())
            .set('startDate', this.formatDate(start))
        .   set('endDate', this.formatDate(end));
    return this.http.get<{ available: boolean }>(${this.apiUrl}/availability,
        { params }); }
```

4.3.6 Artikelverarbeitung auf der Produktdetailseite

Die `ProductDetailPageComponent` dient sowohl der Anzeige detaillierter Artikeldaten als auch der Buchungs- und Verwaltungslogik. Beim Aufruf der Seite wird die Artikel-ID aus der URL extrahiert. Anschließend werden die vollständigen Artikeldaten sowie zugehörige Artikelinstanzen geladen.

Die Nutzer können über ein Datumsauswahlfeld einen Mietzeitraum wählen. Mit jeder Änderung erfolgt ein API-Call, um zu prüfen, ob der Artikel in dem Zeitraum verfügbar ist und wie hoch der Gesamtpreis wäre.

Listing 13: API-gestützte Verfügbarkeitsprüfung

```
    checkAvailability() {
    const start = this.range.get('start')?.value;
    const end = this.range.get('end')?.value;
    if (start && end && this.articleId) {
        this.articleService.checkAvailability(this.articleId,
    start, end).subscribe({
        next: (result) => {
            this.available = result.available; this.totalPrice =
                result.totalPrice * this.quantity; this.availableInstances =
                result.availableInstances;
        }, error: () => {
            this.available = false;
        }
    });
}}
```

Ist ein Artikel verfügbar, kann dieser in einer bestimmten Menge dem Warenkorb hinzugefügt werden. Hierbei wird das Datum ebenfalls gespeichert, damit später eine korrekte Mietabwicklung erfolgen kann.

Instanzverwaltung Admin- oder Staff-Nutzer sehen zusätzlich ein Panel zur Verwaltung von Artikelinstanzen (z.B. einzelne Leihgeräte). Diese können dort:

- im Status geändert,
- gelöscht (nur wenn `RETIRED` und Admin),
- oder neu angelegt werden.

Statusänderung einer Instanz Statusänderungen werden direkt per API aktualisiert.

Listing 14: Statusänderung per PUT-Request

```
onInstanceStatusChange(instance: ArticleInstanceDto, index: number) {
    if (this.article?.id && instance.id) {
        this.instanceService.updateInstance(this.article.id,
    instance.id, instance)
    .subscribe(updatedInstance => {
        this.instances[index] = updatedInstance;
        });
    }
}
```

4.3.7 Warenkorb- und Buchungslogik

Die `ShoppingCartComponent` erlaubt Nutzern das Verwalten und Buchen ihrer ausgewählten Artikel. Technisch basiert der Warenkorb auf dem `CartService`, welcher Einträge lokal im Browser speichert und den Zustand über ein BehaviorSubject verwaltet. Verfügbarkeiten werden per API geprüft, der Checkout initiiert eine serverseitige Buchung.

Der Warenkorb speichert alle Mietartikel lokal im Browser Local Storage und stellt diese als Observable bereit. Beim Seitenaufruf werden diese geladen und der Gesamtpreis berechnet.

Listing 15: Laden von Warenkorbdaten

```
    ngOnInit(): void {
        this.cartService.items$.subscribe(items => {
            this.cartItems = items; this.updateTotalPrice();
        });
    }
```

Die Kosten werden auf Basis des Tagespreises, der Mietdauer sowie der Stückzahl berechnet. Zur Darstellung werden Brutto-, Netto- und Mehrwertsteuer separat ermittelt.

Da Warenkörbe keine Reservierung darstellen, wird regelmäßig die Verfügbarkeit geprüft. Dabei werden gleichartige Positionen gruppiert. Artikel, die nicht mehr verfügbar sind, werden entfernt oder in der Anzahl reduziert (zum Beispiel: Hat man sich 5 von 5 Artikelinstanzen in den Warenkorb gelegt, so wird die Anzahl auf 4 reduziert, sollte eine Instanz in der Zwischenzeit gebucht werden).

Die Verfügbarkeit wird initial, beim Ändern der Menge sowie regelmäßig alle 60 Sekunden aktualisiert.

Beim Klick auf den "Rent Button" wird die Verfügbarkeit ein letztes Mal geprüft. Danach sendet der `RentalService` eine Anfrage zur Erstellung einer Rental. Bei Erfolg wird der Warenkorb geleert und der Nutzer wird zum Dashboard weitergeleitet.

Die Buchungsfunktion ist nur eingeloggten Nutzern vorbehalten. Bereits beim Aufruf der `ShoppingCartComponent` wird geprüft, ob ein gültiger JWT-Token im Local Storage vorhanden ist. Ohne Token ist der Zugriff auf geschützte Routen per AuthGuard blockiert.

4.3.8 Admin Dashboard

Das Admin Dashboard bietet eine Übersicht über aktuelle, fällige und überfällige Mietvorgänge sowie Artikel, die sich in Reparatur befinden. Zusätzlich können Admins und Mitarbeiter die Mietdauer bearbeiten, Artikel zurücknehmen und Reparaturstatus verwalten.

In den Übersichten Due Today, Overdue und Under Repair können Artikel als zurückgegeben markiert oder als repariert bestätigt werden. Dabei wird ein Dialog geöffnet, um den neuen Status zu setzen. Je nach Status wird der Artikel aus der entsprechenden Liste entfernt, z.B. nach erfolgreicher Reparatur aus Under Repair.

Alle Mietvorgänge werden tabellarisch angezeigt. Die Tabelle unterstützt:

- **Sortierung** nach verschiedenen Feldern (z.B. Artikel, Mietende, Mieter)
- **Filterung** nach Status (z.,B. Current Active, Overdue)
- **Suchfunktion** über mehrere Felder
- **Pagination**

Listing 16: Dynamisches Filtern und Sortieren

```
getProcessedRentals(): AdminRentalInfoDto[] {
    return this.allRentals
    .filter(rental => /* Filterlogik je nach Status und Suchbegriff /)
       .sort((a, b) => / Sortierlogik */);
}
```

Admins und Mitarbeiter können die Mietdauer einzelner Mietpositionen verlängern oder verkürzen. Dabei wird die neue Mietdauer serverseitig validiert. Überschneidungen mit anderen Mietvorgängen werden vom Server erkannt und führen zu einer Fehlermeldung.

Listing 17: Mietdauer aktualisieren

```
    this.adminService.updateRentalPeriod(rentalId, newEndDate)
    .subscribe( updated => rental.rentalEnd = updated.rentalEnd );
```

4.3.9 Geplantes Customer Dashboard

Analog zum Admin Dashboard soll ein eigenes Dashboard für Kund:innen entstehen, in dem sie ihre aktiven und geplanten Ausleihen verwalten können. Dort können Buchungen, deren Mietzeitraum noch nicht begonnen hat, selbstständig storniert werden. Zusätzlich wird es die Möglichkeit geben, laufende oder zukünftige Ausleihen zu verlängern oder zu verkürzen. Die Oberfläche orientiert sich dabei der des Admin Dashboards.

4.3.10 Artikel anlegen

Admins oder Mitarbeitende können über ein detailliertes Formular neue Artikel erfassen. Dabei stehen erweiterte Optionen zur Verfügung: Statuswahl für jede Artikelinstanz, Bild-Upload, Kategoriezuweisung sowie eine automatische KI-Beschreibungsgenerierung.

- Bezeichnung, Beschreibung, Stückzahl, Grundpreis (jeweils required)
- Bild (required, validiert clientseitig)
- Kategorieauswahl via eigenem Selector
- Artikelinstanzen: Entweder alle mit gleichem Status, oder individueller Status pro Instanz

Abhängig von der Checkbox *Alle Artikel Instanzen haben denselben Status* wird entweder ein einzelnes Statusfeld oder für jede Instanz ein separates Dropdown angezeigt.

Listing 18: Status-Felder aktualisieren

```
updateInstanceStatuses(): void {
    while (this.instanceStatuses.length) {
        this.instanceStatuses.removeAt(0);
    }
    const count = this.articleForm.get('stueckzahl')?.value || 0;
    for (let i = 0; i < count; i++) {
        this.instanceStatuses.push(this.fb.control('', Validators.required));
    }
}
```

Das gewählte Bild wird direkt als Vorschau angezeigt. Der Upload erfolgt vor dem POST des Artikels. Nach erfolgreichem Upload wird die URL ins Payload übernommen. Die Erstellung eines Artikels erfolgt nur bei erfolgreichem Bild-Upload. Nach erfolgreicher Anlage erfolgt eine Weiterleitung zum Admin-Dashboard.

4.3.11 Artikel bearbeiten

Admins und Mitarbeitende können Artikel über ein Formular bearbeiten. Dabei können Beschreibung, Preis, Bild und Kategorien angepasst werden.

Optional steht auch eine automatische Beschreibungsgenerierung per KI zur Verfügung.Beim Laden der Komponente wird der bestehende Artikel über die ID geladen und in das Formular überführt. Auch vorhandene Kategorien und das aktuelle Bild werden angezeigt.

Listing 19: Artikel laden und Formular befüllen

```
    this.articleService.getArticleById(this.articleId).subscribe((res) => {
        this.originalArticle = res;
    this.articleName = res.bezeichnung;
    this.existingImage = res.bildUrl;

    this.updateForm.patchValue({ beschreibung: res.beschreibung, grundpreis:
        res.grundpreis, categories: res.categories });
    this.selectedCategories = res.categories;
    });
```

Wird ein neues Bild gewählt, wird es direkt als Vorschau angezeigt. Vor dem Patch wird das Bild über den `FileUploadService` hochgeladen, anschließend die neue URL übernommen. Es werden nur geänderte Felder per PATCH an den Server gesendet. Erfolgt keine Änderung, wird kein Feld übermittelt. Nach erfolgreichem Update erfolgt eine Navigation zur Produktdetailseite.

4.4 Spezielle Themen

4.4.1 State Management mit BehaviorSubject im Warenkorb

Damit der Warenkorb im gesamten Frontend einheitlich und aktuell bleibt – egal ob in der Produktübersicht, im Warenkorb selbst oder beim Checkout – wird der aktuelle Inhalt zentral verwaltet. Hierfür wurde in der Anwendung ein BehaviorSubject aus der RxJS Library genutzt.

Ein BehaviorSubject ist eine spezielle Datenquelle, die den aktuellen Zustand speichert und automatisch an alle interessierten Stellen (z.B. Komponenten, Services) weiterleitet. Sobald sich der Warenkorb ändert (etwa durch das Hinzufügen eines Artikels), erhalten alle Abonnenten sofort den neuen Zustand, ohne aktiv nachfragen zu müssen (RxJS, n.d., Zayed, 2023).

Für die Anwendung bedeutet das: Der Warenkorb-Zustand steht jederzeit aktuell und reaktiv zur Verfügung, selbst wenn Komponenten später initialisiert werden. Der `CartService` verwendet ein `BehaviorSubject<CartItem[]>`, um den Zustand zu verwalten. Über die Methode `next()` werden Änderungen wie Hinzufügen, Löschen oder Anpassen von Mengen an interessierte Komponenten weitergegeben.

Listing 20: Initialisierung und Nutzung von BehaviorSubject im Service

```
private itemsSubject: BehaviorSubject<CartItem[]> =
    new BehaviorSubject<CartItem[]>(this.loadInitialItems());
public items$ = this.itemsSubject.asObservable();

addItem(newItem: CartItem): void {
    const items = this.itemsSubject.getValue();
    items.push(newItem);
    this.itemsSubject.next([...items]); // nderung verteilen
}
```

Der Warenkorb ist somit über items$ (als Observable bereitgestellt) in allen Komponenten synchron nutzbar.

Listing 21: Abonnement in einer Komponente

```
ngOnInit(): void {
    this.cartService.items$.subscribe(items => {
        this.cartItems = items; // Immer der aktuellste Stand
        this.updateTotalPrice();
    });
}
```

Ein zusätzlicher Vorteil ist, dass durch das Initialisieren des BehaviorSubject mit Daten aus dem localStorage Persistenz gewährleistet wird – auch nach einem Seitenreload ist der Zustand verfügbar.

4.4.2 Pagination im Admin Dashboard

Damit auch bei vielen Ausleihvorgängen noch eine übersichtliche Darstellung möglich ist, wurde im Admin Dashboard Pagination implementiert.

Dazu wird zunächst der komplette Bestand an Ausleihpositionen vom Backend geladen und im Frontend zwischengespeichert. Die eigentliche Aufteilung übernimmt der `MatPaginator`

aus dem Angular Material – eine Komponente, die automatisch Seitenzahlen anzeigt, zwischen Seiten navigieren lässt und auch die Anzahl der Einträge pro Seite festlegt (Angular, n.d.-b).

Listing 22: Pagination mit MatPaginator im Dashboard

```
<mat-paginator
    [length]="getProcessedRentals().length"
    [pageSize]="pageSize"
    [pageSizeOptions]="[5, 10, 25, 100]" [pageIndex]="pageIndex"
        (page)="onPageChange($event)">
</mat-paginator>
```

Die Daten, die auf einer Seite angezeigt werden, berechnet eine Methode über einen einfachen `slice()`-Aufruf (GeeksforGeeks, 2024) :

Listing 23: Berechnung der aktuellen Seite

```
paginatedRentals(): AdminRentalInfoDto[] {
    const filtered = this.getProcessedRentals();
    const start = this.pageIndex * this.pageSize;
    return filtered.slice(start, start + this.pageSize);
}
```

Dabei wird zuerst der gesamte Datenbestand nach den aktuell gesetzten Filtern und Sortierungen gefiltert – das passiert alles lokal über `getProcessedRentals()`. Erst danach erfolgt die eigentliche Pagination. Die Lösung funktioniert gut bei mittleren Datenmengen. Sie ist allerdings nicht besonders effizient, weil sämtliche Einträge vom Server geladen und im Browser verarbeitet werden. Bei sehr großen Datenmengen würde man stattdessen auf serverseitige Pagination setzen, bei der nur die jeweils benötigte Seite vom Backend geliefert wird – inklusive Filterung und Sortierung.

4.5 End-to-End Testing mit Cypress

Mit dem End-to-End Testing haben wir überprüft, ob typische Nutzerinteraktionen im Frontend wie erwartet funktionieren. Dazu gehörten unter anderem das Aufrufen der Startseite, die Registrierung, der Login sowie das Mieten eines Produkts. Um die Tests unabhängig vom Backend ausführen zu können, haben wir cy.intercept() genutzt, um Netzwerkrequests abzufangen und gezielt Mockdaten zurückzugeben. Damit konnten wir beispielsweise sowohl erfolgreiche Registrierung und Login-Vorgänge als auch Fehlerszenarien testen. Besonders im Fokus stand dabei der Mietprozess: Vom Login über die Verfügbarkeitsprüfung eines Artikels bis hin zum Checkout haben wir den kompletten Ablauf automatisiert getestet. Es sei angemerkt, dass die Tests auf einer früheren Version der Anwendung mit klassischem Login per Passwort basieren. Inzwischen kommt ein Magic-Link-Verfahren zum Einsatz, wodurch die Tests künftig angepasst werden müssen.

Um den Test starten zu können muss via die Commandline in das frontend-Verzeichnis gewechselt werden. Dort müssen zunächst, sofern noch nicht geschehen, alle Dependencies mit npm install installiert werden. Im Anschluss kann mit ng e2e der Test gestartet werden.

5 Organisatorisches

5.1 Projektmanagement und Versionierung

Wir haben uns zu Beginn des Projekts die groben Teilbereiche aufgeteilt, die jede Person bearbeitet und uns anschließend regelmäßig gegenseitig über die Fortschritte Informiert. Discord war dafür ein Hauptkommunikationsweg und auch nützlich, um über Screenshare gemeinsam Fehler zu suchen. Eine strikte Trennung der Verantwortlichkeiten war nicht ganz möglich, daher haben wir an manchen Programmteilen und Logiken auch beide gearbeitet. Dies erforderte teils Disziplin, um sich nicht gegenseitig in der Entwicklung zu behindern. Merges haben wir daher auch teilweise gemeinsam durchgeführt.

Zur gemeinsamen Entwicklung haben wir ein GitHub-Repository erstellt, in verschiedenen Branches gearbeitet und diese regelmäßig in den Main-Branch gemerged.

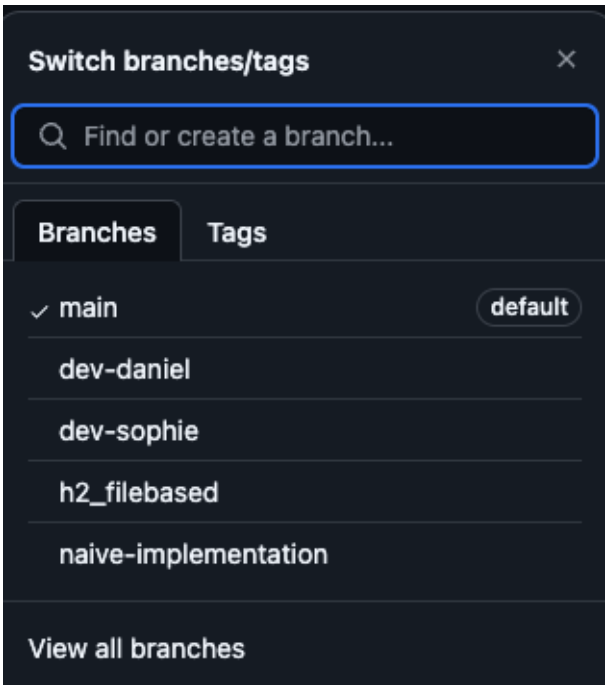


Figure 12: GitHub Branches

Für das Domainmodell, und den Austausch von Kommentaren haben wir Lucid als Arbeits-Board genutzt.

Retrospektive (Sophie) Ich habe das Projekt anfangs ehrlich gesagt ein bisschen unterschätzt – sowohl was den zeitlichen Aufwand als auch die Komplexität vieler Features angeht. Besonders das Debuggen hat oft extrem viel Zeit gefressen, teilweise habe ich stundenlang an einem Problem gesessen, ohne direkt sichtbare Fortschritte zu machen.

Auch der Einstieg fiel mir schwer, weil ich erstmal gar nicht genau wusste, wo ich eigentlich anfangen soll. Ich hatte viele Ideen und Features im Kopf, musste dann aber im Verlauf einiges davon verwerfen oder stark vereinfachen, einfach weil die Zeit knapp wurde. Im Nachhinein war das aber auch gut so, weil ich mich so besser auf die wirklich wichtigen Kernfunktionen konzentrieren konnte.

Mein größtes Learning aus dem Projekt: Lieber früh einfach anfangen, kleine Prototypen bauen und ausprobieren, statt zu lange theoretisch zu planen. Dadurch kommt

man schneller ins Arbeiten und bekommt schneller ein Gefühl für das Projekt.

Retrospektive (Daniel) Außerdem ist der zusätzliche Aufwand, ein Projekt zu zweit zu organisieren, nicht zu unterschätzen. Es ist nicht nur die höhere Anforderung an die Klassen und Funktionen, die sich durch die Teamarbeit erhöht. Auch die gemeinsame Planung, der regelmäßige Abgleich, damit beide Personen auf dem gleichen Wissensstand sind, das Verteilen von Aufgaben, das gemeinsame Reviewen von Code und Mergen sind Aufwände, die bei einem Einzelprojekt nicht entstehen. Dennoch sind das wiederum auch wertvolle Learnings und Erfahrungen, die in zukünftigen Projekten nur nützlich sein werden.

5.2 Installation, Konfiguration und Deployment

5.2.1 Installation

Die folgenden Tools sind erforderlich, um die Anwendung auszuführen:

- **Node.js und npm** für das Angular-Frontend https://nodejs.org/.
- **Angular CLI** – über terminal installieren:

  ```
  npm install -g @angular/cli
  ```

- **Java JDK 11 (oder höher)** https://www.oracle.com/java/technologies/javase-downloads.html.
- **Maven** https://maven.apache.org/download.cgi.

Genaue Informationen zur Installation finden sich in `README.md`.
Die Anwendung ist standardmäßig unter `http://localhost:4200` erreichbar. Das Backend ist standardmäßig unter `http://localhost:8080` erreichbar.

5.2.2 Konfiguration

Für einige Programmbestandteile müssen in den `application.properties` Einstellungen vorgenommen werden:

- **Datenbank** Wir nutzen für die Datenbank H2 in Memory, da sie simpel zu implementieren ist und für unsere Testumgebung ausreicht. Wir hatten zwischenzeitlich auch eine Filebased h2 Datenbank im Einsatz, diese hatte jedoch Probleme mehrere Verbindungen zu verwalten. Um die Datenbank automatisiert mit Daten zu füllen, gibt es den `DatabaseSeeder`. Folgende Parameter nutzen wir:

  ```
  spring.datasource.url=jdbc:h2:mem:rentify;
  DB_CLOSE_DELAY=-1;
  DB_CLOSE_ON_EXIT=FALSE

  spring.datasource.driver-class-name=org.h2.Driver
  spring.datasource.username=sa
  spring.datasource.password=password
  spring.h2.console.enabled=true
  spring.h2.console.path=/h2-console
  ```

Für den produktiven Einsatz würden wir auf eine persistente Datenkbank, wie zum Beispiel PostgreSQL wechseln, zu Demonstrationszwecken reicht diese Struktur aber vollkommen aus.

- **Telegram Bot Token** Der Parameter muss ausgefüllt werden, damit der Messenger Bot funktioniert:

  ```
  telegram.bot.token=
  ```

- **Gemini API** Die Google Gemini API benötigt einen Key, damit die LLM-Erzeugung des Beschreibungstextes funktioniert:

  ```
  gemini.api.key=
  ```

- **Spring Mail** Die Konfiguration für den E-Mail-Versand muss ebenso hinterlegt werden. Wir nutzen zum Testen mailtrap.

  ```
  spring.mail.host=smtp.mailtrap.io
  spring.mail.port=2525
  spring.mail.username=
  spring.mail.password=
  spring.mail.properties.mail.smtp.auth=true
  spring.mail.properties.mail.smtp.starttls.enable=true
  ```

References

Angular. (n.d.-a). *Canactivate*. Google LLC. Retrieved March 18, 2025, from https://angular.io/api/router/CanActivate

Angular. (n.d.-b). *Paginator — angular material*. Retrieved March 18, 2025, from https://material.angular.io/components/paginator/overview

Balasubramaniam, V., & Vilao, R. (2024, May). *Defining jpa entities*. Retrieved March 16, 2025, from https://www.baeldung.com/jpa-entities

Datta, D. (2020, June). *Purpose of service interface class in spring boot*. Retrieved March 16, 2025, from https://stackoverflow.com/q/62513212

Domingues, M. (2024, September). *Spring boot's @restcontroller vs @controller: A comprehensive guide*. Retrieved March 16, 2025, from https://medium.com/devdomain/spring-boots-restcontroller-vs-controller-a-comprehensive-guide-c045ab1c97a9

GeeksforGeeks. (2024, July). *Create a pagination using html css and javascript*. Retrieved March 18, 2025, from https://www.geeksforgeeks.org/create-a-pagination-using-html-css-and-javascript/

Paraschiv, E., & Avramović, S. (2025, February). *Introduction to spring data jpa*. Retrieved March 16, 2025, from https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa

RxJS. (n.d.). *Behaviorsubject*. Retrieved March 18, 2025, from https://www.learnrxjs.io/learn-rxjs/subjects/behaviorsubject

Shvets, A. (n.d.). *Strategy - design patterns*. Retrieved March 18, 2025, from https://refactoring.guru/design-patterns/strategy

Spring. (n.d.). *Specifications - spring data jpa*. Retrieved March 16, 2025, from https://docs.spring.io/spring-data/jpa/reference/jpa/specifications.html

Tacu, O. M., & Williams, L. (2024, April). *Differences between entities and dtos*. Retrieved March 16, 2025, from https://www.baeldung.com/java-entity-vs-dto

Zayed, M. (2023, October). *Subjects and behaviorsubjects in angular: A deep dive*. Retrieved March 18, 2025, from https://dev.to/mariazayed/subjects-and-behaviorsubjects-in-angular-a-deep-dive-4kf2

.0.3 Anhang A: GitHub-Link

Der Quellcode des Projekts ist auf GitHub verfügbar: https://github.com/sophie4075/PaF.