

# Prototyping Projektdokumentation

Name: Julia Peric

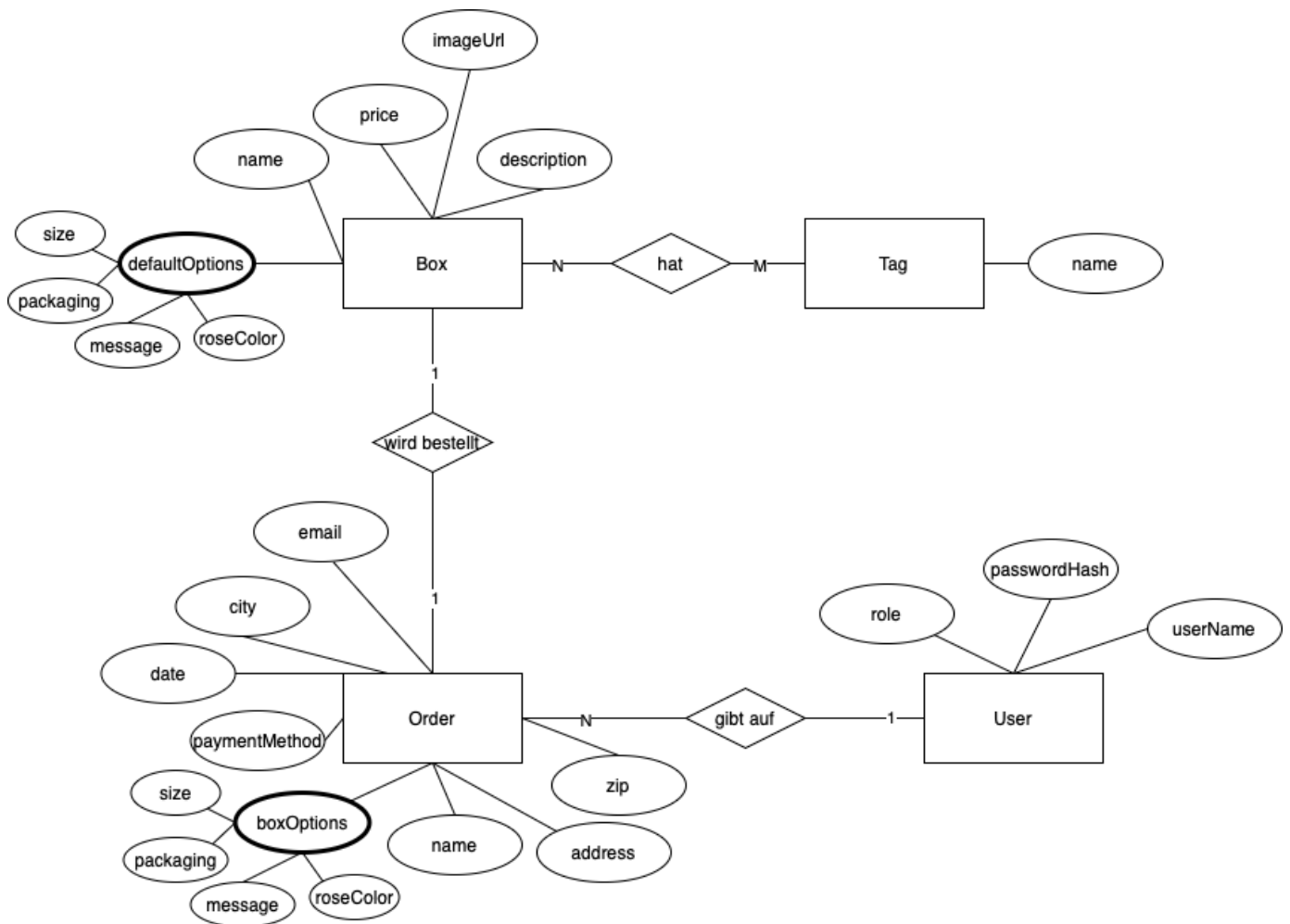
E-Mail: pericjul@students.zhaw.ch

URL der deployten Anwendung: <https://poshnrosy.netlify.app>

## 1. Einleitung

Posh & Rosy ist eine benutzerfreundlich gestaltete Web-App, die handgemachte Geschenkboxen digital erlebbar macht. Entstanden aus der kreativen Leidenschaft dreier Schwestern verbindet das Projekt florale Ästhetik mit moderner Technologie. Ziel der Anwendung ist es, persönliche Geschenke einfach, stilvoll und online bestellbar zu machen. Die App richtet sich an Kundinnen und Kunden, die das Besondere suchen. Hochwertige Geschenkboxen, individuell konfigurierbar und mit viel Liebe zum Detail zusammengestellt, stehen im Mittelpunkt. Neben der ansprechenden Darstellung der Produkte bietet die Plattform einen intuitiven Bestellprozess sowie eine flexible Anpassungsfunktion für Farben, Inhalte und Verpackungsstile. Für uns als Gründerinnen ist Posh & Rosy nicht nur eine Geschäftsidee, sondern ein Herzensprojekt. Es zeigt, wie sich familiäre Kreativität mit digitaler Benutzerfreundlichkeit vereinen lässt. Ein integrierter Adminbereich ermöglicht zudem die effiziente Verwaltung aller Inhalte wie Produkte, Bestellungen und Benutzerkonten.

## 2. Datenmodell



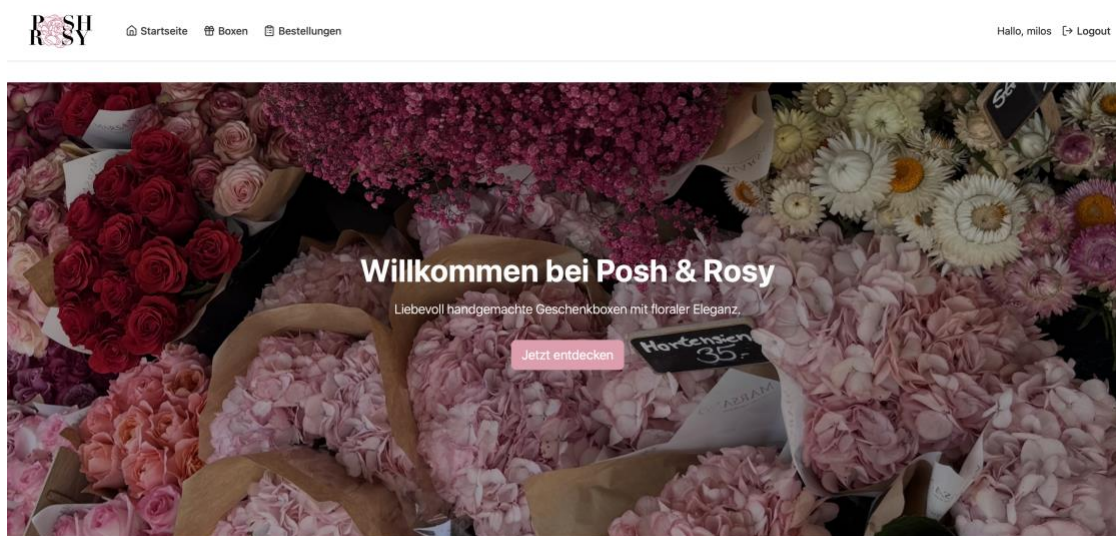
Das obenstehende ER-Diagramm beschreibt die Datenstruktur der Webanwendung Posh & Rosy. Es enthält die vier zentralen Entitäten Box, Order, User und Tag, ergänzt durch strukturierte Untergruppen wie `defaultOptions` und `boxOptions`, die zur Gruppierung der Produktkonfigurationen dienen.

- Box enthält allgemeine Angaben zu einer Geschenkbox wie Name, Beschreibung, Preis und Bild sowie standardmässige Optionen (defaultOptions) zur Konfiguration.
- Über eine N:M-Beziehung ist jede Box mit beliebig vielen Tags (z. B. „Elegant“, „Rosa“) verknüpft.
- Orders (Bestellungen) sind direkt einer Box zugeordnet und beinhalten Lieferinformationen sowie individuell angepasste Optionen (boxOptions).
- Ein User kann mehrere Bestellungen aufgeben, muss aber nicht zwingend existieren – Bestellungen durch Gäste sind möglich.
- Die Entität defaultOptions sowie boxOptions sind im Diagramm als eigene Objekte dargestellt, dienen aber rein der Strukturierung und existieren in der MongoDB als verschachtelte Objekte in box bzw. order.

### 3. Beschreibung der Anwendung

#### 3.1. Startseite

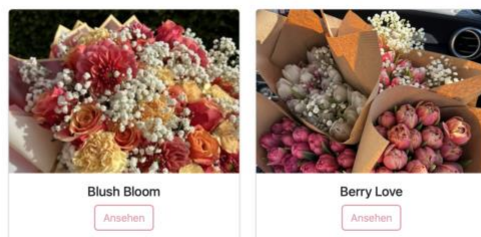
Route: /



#### Unsere beliebtesten Boxen

Im oberen Bereich der Startseite befindet sich ein Hero-Element mit einem vollflächigen Hintergrundbild. Der Titel «Willkommen bei Posh & Rosy» sowie ein Untertitel und ein Call-to-Action-Button («Jetzt entdecken») stehen zentral im Fokus. Durch Klick auf den Button wird die Benutzerin zur Übersicht aller Geschenkboxen weitergeleitet. Die Navigation im oberen Bereich passt sich je nach Login-Status und Benutzerrolle an.

#### Unsere beliebtesten Boxen



#### Warum Posh & Rosy?

- 👑 Mit Liebe handgemacht
- 🎨 Individuell anpassbar
- 📦 Schneller Versand in der Schweiz

#### Das sagen unsere Kund:innen

"So viel Liebe zum Detail – meine Schwester hat sich riesig gefreut!"

"Wunderschön verpackt und perfekt als Geschenk – ich bestelle wieder!"

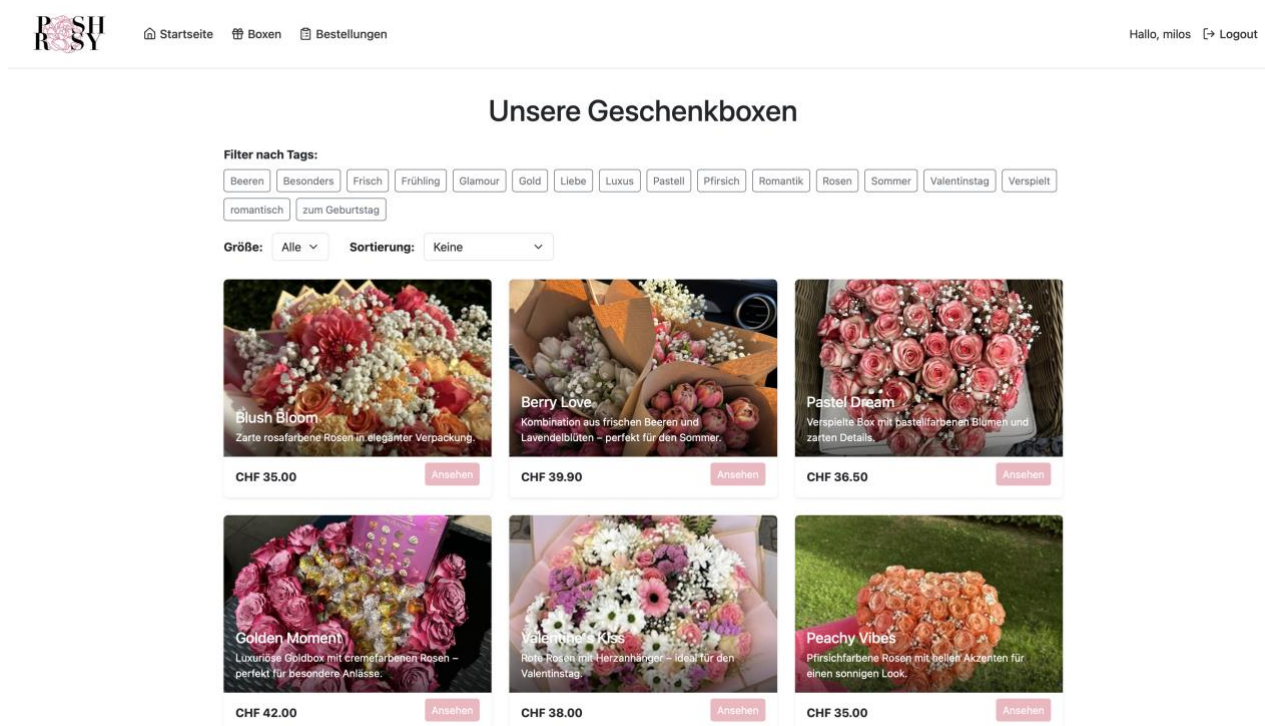
Im unteren Teil der Startseite werden zwei besonders beliebte Geschenkboxen dargestellt, inklusive Bild, Name und einem Link zur Detailansicht. Es folgt ein Abschnitt mit den Alleinstellungsmerkmalen des Angebots (Handarbeit, Individualisierung, schneller Versand). Abschliessend werden zwei Kundenmeinungen angezeigt, die die Qualität und Präsentation der Produkte betonen.

Dateien:

- `src/routes/+page.svelte`
- `src/lib/components/layout/Navbar.svelte`
- `src/lib/components/layout/Footer.svelte`

### 3.2. Boxübersicht

Route: `/boxes`



Auf dieser Seite werden alle verfügbaren Geschenkboxen angezeigt. Nutzerinnen und Nutzer haben die Möglichkeit, das Angebot mithilfe von drei Filtern zu durchsuchen. Die Tag-Filter erlauben die Auswahl thematischer oder saisonaler Eigenschaften wie "Valentinstag", "frisch" oder "zum Geburtstag". Diese Tags basieren auf der Datenbank und werden dynamisch generiert. Über das Dropdown für die Grösse kann zwischen den Optionen S, M, L oder "Alle" gewählt werden. Zusätzlich steht ein Sortierfeld zur Verfügung, mit dem die Boxen nach Preis aufsteigend oder absteigend sortiert werden können. Die Filter sind kombinierbar und werden direkt im Frontend umgesetzt. Die Filterlogik basiert auf einem abgeleiteten Array namens `filteredBoxes`, welches sich automatisch aktualisiert, sobald der Nutzer eine Auswahl trifft. Die Seite wird dabei nicht neu geladen, was die Benutzererfahrung verbessert. Jede Box wird in Form einer Karte im Grid dargestellt. Die Darstellung erfolgt mit der Komponente `BoxCard.svelte`, welche das Bild, den Preis und eine Kurzbeschreibung anzeigt. Mit dem Button "Ansehen" wird die Detailseite der jeweiligen Box aufgerufen. Die Seite bietet einen übersichtlichen Einstieg in das Angebot und macht es durch die Filter einfach, gezielt nach passenden Geschenkboxen zu suchen.

Dateien:

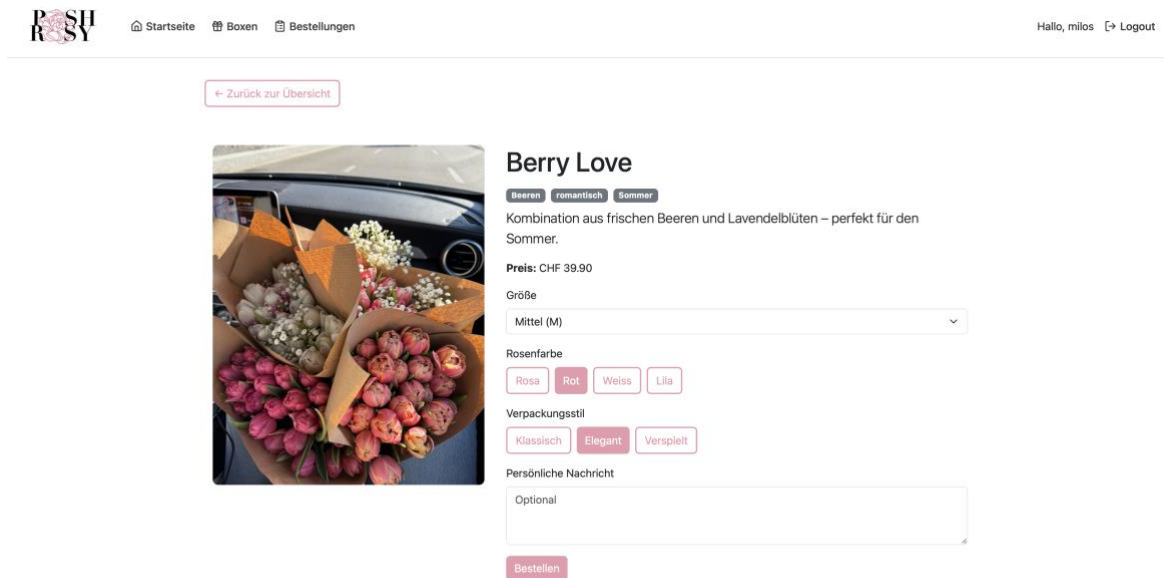
- `lib/components/boxes/BoxCard.svelte`
- `routes/boxes/+page.svelte`
- `routes/boxes/+page.server.js`

### 3.3. Box-Detailseite

Route: /boxes/[id]

Damit die Detailseite optimal genutzt wird, habe ich zwei unterschiedliche Ansichten eingebaut. Je nach Benutzerrolle wird eine passende Funktion angezeigt. In diesem Kapitel unterscheide ich deshalb zwischen den beiden Rollen.

#### 3.3.1. Für Kund:Innen




Auf der Detailseite erhalten Kund:innen einen umfassenden Überblick über eine spezifische Geschenkbox. Im linken Bereich wird das Boxenbild gross dargestellt, während auf der rechten Seite der Name der Box, eine Beschreibung, zugewiesene Tags sowie der Preis angezeigt werden. Der angezeigte Preis bezieht sich auf die Standardkonfiguration der Box. Diese Standardkonfiguration umfasst drei Merkmale: Grösse, Rosenfarbe und Verpackungsstil. Diese Werte sind für jede Box individuell vordefiniert und stellen den Grundwert dar. Gleichzeitig haben Kund:innen die Möglichkeit, die Konfiguration vor der Bestellung individuell anzupassen. Dazu stehen Dropdowns und Buttons zur Auswahl, über die Grösse, Farbe und Verpackung frei gewählt werden können. Zusätzlich kann optional eine persönliche Nachricht eingegeben werden, die der Lieferung beigelegt wird. Diese Anpassungen wirken sich ausschliesslich auf die jeweilige Bestellung aus und ändern nicht die Produktdaten der Box. Auch der Preis bleibt gleich und bezieht sich weiterhin auf die ursprünglich definierte Konfiguration. Nach Abschluss der Anpassung kann die Box über den Button „Bestellen“ zur nächsten Seite weitergeleitet werden. Dabei werden die gewählten Optionen als Parameter übergeben und auf der Folgeseite übernommen.

Dateien:

- routes/boxes/[id]/+page.svelte
- routes/boxes/[id]/+page.server.js


### 3.3.2. Für Admins



[Startseite](#) [Boxen](#) [Admin](#)

Hallo, julia [Logout](#)

[← Zurück zur Übersicht](#)



#### Box bearbeiten

Name

Berry Love

Beschreibung

Kombination aus frischen Beeren und Lavendelblüten – perfekt für den Sommer.

Preis (CHF)

39,9

Tags (kommagetrennt)

Beeren, romantisch, Sommer

Standardgröße

M

Standard-Rosenfarbe

Rot

Standard-Verpackung

Elegant

Speichern

Abbrechen

Auf der Detailseite einer Box wird eingeloggten Admins zusätzlich ein Button „Bearbeiten“ angezeigt. Nach Klick auf diesen Button wechselt die Ansicht in ein Bearbeitungsformular, in dem alle relevanten Felder der Box angepasst werden können. Dazu gehören der Name, die Beschreibung, der Preis sowie die zugewiesenen Tags. Ebenso können die Standardwerte für die Grösse, Rosenfarbe und Verpackung der Box bearbeitet werden. Beim Öffnen der Seite wird die Box anhand der ID über die Funktion `getBox(id)` aus der `boxes`-Collection gelesen. Das vorhandene Objekt wird an das Formular übergeben, sodass die Felder vorausgefüllt erscheinen.

Wenn der Admin auf „Speichern“ klickt, wird ein POST-Request an den `?/edit`-Endpoint gesendet. In der zugehörigen `actions.edit`-Funktion (in der Datei `+page.server.js`) werden die übermittelten Formular Daten gesammelt, validiert und mithilfe der Funktion `updateBox(id, updatedBox)` aktualisiert. Diese Funktion führt in der Datenbank ein `$set`-Update auf die entsprechende Box aus. Die Änderungen wirken sich unmittelbar auf die Anzeige der Box in der Übersicht und auf der Detailseite aus.

Die Datenänderung betrifft folgende Felder in der Collection `boxes`:

- Name
- description
- price
- tags
- defaultOptions.size
- defaultOptions.roseColor
- defaultOptions.packaging

Auf diese Weise können Administratorinnen und Administratoren bestehende Produkte flexibel und effizient verwalten. Die Rechteprüfung, ob ein Benutzer Admin ist, erfolgt auf Serverseite über `locals.user?.role`.


Dateien:

- `src/routes/boxes/[id]/+page.svelte`
- `src/routes/boxes/[id]/+page.server.js`
- `lib/server/db.js (updateBox())`



### 3.4. Bestellen

Route: /boxes/[id]/order



StartseiteBoxenBestellungen

Hallo, milos → Logout

[← Zurück](#)

## Deine Bestellung: Berry Love

Ausgewählte Optionen:

- **Grösse:** M
- **Rosenfarbe:** Rot
- **Verpackung:** Elegant
- **Nachricht:** Keine

**Lieferadresse**

Name

Adresse

PLZ

Ort

E-Mail (fuer Rechnung)

Bezahlart: Rechnung

Bestellung abschicken

Nach Auswahl und individueller Konfiguration einer Geschenkbox gelangt der Benutzer durch Klick auf den Button “Bestellen” auf die finale Bestellseite. Dort werden die gewählten Optionen wie Grösse, Rosenfarbe, Verpackung und persönliche Nachricht nochmals übersichtlich dargestellt. So kann der Benutzer sicherstellen, dass die Bestellung seinen Vorstellungen entspricht. Direkt darunter befindet sich ein Formular zur Eingabe der Lieferdaten. Es erfasst Name, Adresse, Postleitzahl, Ort und E-Mail-Adresse. Die E-Mail-Adresse ist notwendig, um die Rechnung zustellen zu können, da die Bezahlart standardmässig auf “Rechnung” gesetzt ist.

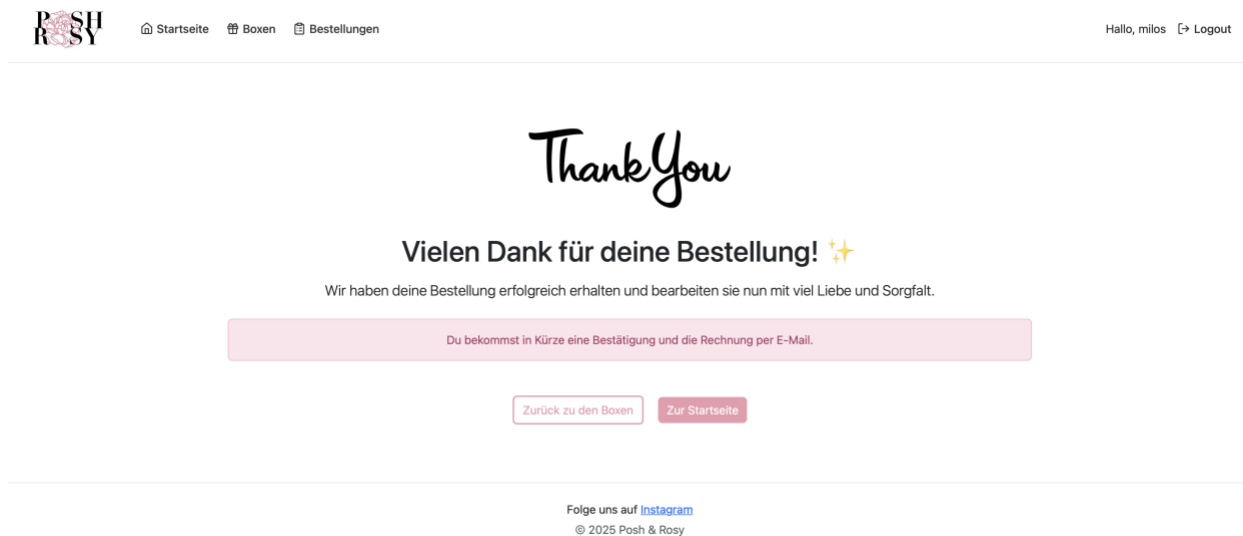
Beim Klick auf “Bestellung abschicken” wird ein neuer Eintrag in der Collection orders in der MongoDB erzeugt. Der Eintrag enthält die ID und den Namen der Box, die individuell gewählten Optionen, alle Lieferangaben sowie das Bestelldatum. Wenn der Benutzer eingeloggt ist, wird zusätzlich seine Benutzer-ID gespeichert. Dadurch können ihm die Bestellungen im persönlichen Bereich zugeordnet werden. Bestellt eine nicht eingeloggte Person, bleibt die Benutzer-ID leer, allerdings wird der Name der Person trotzdem gespeichert.

Dateien:

- `src/routes/boxes/[id]/order/+page.svelte`
- `src/routes/boxes/[id]/order/+page.server.js`
- `lib/server/db.js` (Funktion `createOrder`)

### 3.5. Dankeseite

Route: /boxes/[id]/order/thank-you



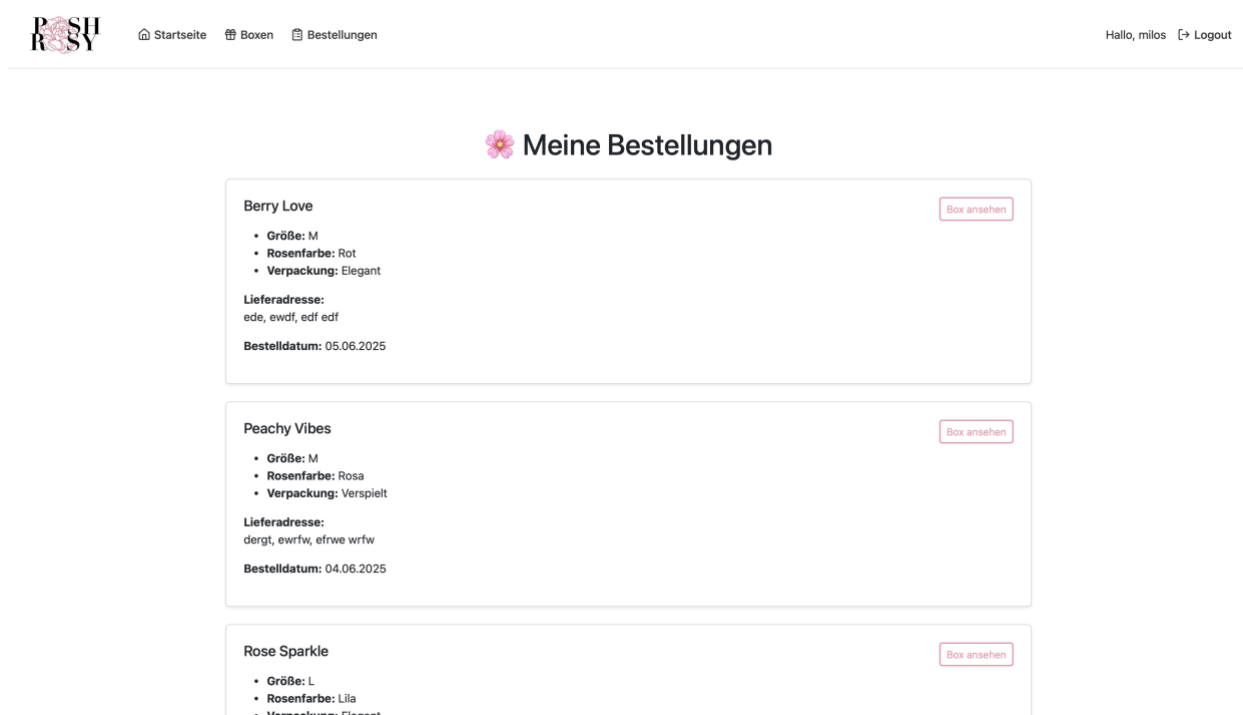
Nach erfolgreichem Absenden der Bestellung wird der Benutzer automatisch auf eine Bestätigungsseite weitergeleitet. Diese Seite zeigt eine visuell ansprechende Danksagung und informiert den Benutzer, dass seine Bestellung erfolgreich eingegangen ist. Gleichzeitig wird darauf hingewiesen, dass die Bestellung nun bearbeitet wird und in Kürze eine Bestätigung sowie die Rechnung per E-Mail versendet wird. Die Seite enthält zwei Navigationsbuttons, mit denen man entweder zurück zur Boxenübersicht oder zur Startseite gelangt.

Diese Seite dient rein der Rückmeldung für den Benutzer. Es finden an dieser Stelle keine Datenbankoperationen mehr statt, da die Bestellung bereits vorher in der Datenbank persistiert wurde. Der Zugriff erfolgt über eine eigene Route, die direkt nach dem POST-Request über einen redirect angesteuert wird.

Dateien:

### 3.6. Bestellübersicht

Route: /orders



Eingeloggte Benutzerinnen und Benutzer haben über die Navigation Zugriff auf eine persönliche Bestellübersicht. Diese Seite zeigt eine Liste aller bisher aufgegebenen Bestellungen, inklusive der jeweiligen Boxnamen, Konfigurationen (Grösse, Rosenfarbe, Verpackung, Nachricht), Lieferadresse und Bestelldatum. Zusätzlich wird bei jeder Bestellung ein Button angezeigt, mit dem die entsprechende Box erneut aufgerufen werden kann.

Die Daten werden serverseitig in der Datei `routes/orders/+page.server.js` über die Funktion `getOrdersByUser(user._id)` aus der Datenbank geladen. Dafür wird in der Collection `orders` nach allen Einträgen gesucht, die dem eingeloggten Benutzer zugeordnet sind.

Die Anzeige erfolgt in `routes/orders/+page.svelte`, wo jede Bestellung formatiert und im Layout dargestellt wird. Die Übersicht dient der Transparenz und Nachverfolgbarkeit für Kundinnen und Kunden und ermöglicht den direkten Zugriff auf einzelne Boxdetails.

Dateien:

- `lib/server/db.js`
- `routes/orders/+page.server.js`
- `routes/orders/+page.svelte`

### 3.7. Admin-Overview

Route: `/admin/overview`

The screenshot shows the Admin-Overview page with a navigation bar at the top containing the logo, links for 'Startseite', 'Boxen', and 'Admin', and a user greeting 'Hallo, Julia' with a 'Logout' link.

The main content area is divided into three panels:

- Boxen verwalten:** A list of gift boxes with details like name, price, and size. Each entry has 'Ansehen' and 'Löschen' buttons. Below the list is a form to 'Neue Box erstellen' with fields for Boxname, Beschreibung, Preis in CHF, Bild hochladen, Tags, and Standard-Optionen (Groesse, Rosa, Klassisch). A 'Box hinzufügen' button is at the bottom.
- Kund:innen:** A list of users with their roles (admin or customer) and 'Löschen' buttons. Below is a form to 'Neue Kundin / neuer Kunde' with fields for Username, Passwort, and a dropdown for 'Kund:in', followed by a 'Hinzufügen' button.
- Bestellungen:** A list of orders with details like item name, user, date, and time. The list shows three orders: #1 - Rose Sparkle, #2 - Peachy Vibes, and #3 - Berry Love.

Die Admin-Übersicht stellt eine zentrale Verwaltungsseite für Benutzerinnen und Benutzer mit Adminrolle dar. Sie besteht aus drei funktionalen Bereichen: Boxenverwaltung, Benutzerverwaltung und Bestellübersicht. Im Bereich Boxen verwalten werden alle aktuell gespeicherten Geschenkboxen aufgelistet. Zu jeder Box kann direkt eine Detailansicht geöffnet oder die Box vollständig gelöscht werden. Die Löschung wird mit einem Formular-Submit über die `actions.deleteBox`-Funktion in der Datei `routes/admin/overview/+page.server.js` abgewickelt. Dabei wird ein entsprechender Eintrag aus der `boxes`-Collection in der MongoDB gelöscht.

Darunter befindet sich ein Formular zum Erstellen einer neuen Box. Es erlaubt das Eingeben aller relevanten Felder inklusive Bildupload, Beschreibung, Preis, Tags und Standardoptionen. Beim Absenden wird das Bild im Verzeichnis `static/images` gespeichert und der neue Eintrag mithilfe der `createBox`-Funktion aus `lib/server/db.js` in der



Datenbank persistiert. Der mittlere Bereich zeigt alle registrierten Benutzerinnen und Benutzer. Admins können über den Button „Löschen“ ein Benutzerkonto vollständig entfernen. Zusätzlich steht ein Formular zur Verfügung, mit dem neue Benutzer erfasst werden können.

Beim Absenden wird das Passwort gehasht und in der Collection users gespeichert. Die entsprechende Logik findet sich in actions.createUser in der Datei routes/admin/overview/+page.server.js. Im dritten Bereich unten rechts wird eine einfache Liste aller Bestellungen angezeigt. Diese wird über die Funktion getOrders() aus lib/server/db.js geladen. Für jede Bestellung werden die interne ID, der Name der bestellten Box, der Name der Kundin oder des Gasts sowie Datum und Uhrzeit angezeigt. Dieser Bereich dient der schnellen Übersicht über neue Bestellungen, ist aber bewusst schlank gehalten.

Dateien:

- lib/server/db.js
- routes/admin/overview/+page.server.js
- routes/admin/overview/+page.svelte
- lib/components/admin/BoxManager.svelte
- lib/components/admin/UserManager.svelte
- lib/components/admin/OrderList.svelte

### 3.8. Login

Route: /login

The screenshot shows the login page of the 'Rosy' application. The header features the 'Rosy' logo, navigation links for 'Startseite' and 'Boxen', and a 'Login' link. The main content area is titled 'Login' and contains a form with a message 'Benutzer nicht gefunden' (User not found) in a pink box. Below this are input fields for 'Benutzername' (Username) and 'Passwort' (Password), followed by a pink 'Einloggen' (Login) button. The footer includes a link to follow on Instagram and a copyright notice for 2025 Posh & Rosy.

Die Login-Seite ermöglicht es registrierten Benutzer:innen, sich mit ihrem Benutzernamen und Passwort anzumelden. Dabei wird überprüft, ob die Zugangsdaten korrekt sind. Falls die Anmeldung erfolgreich ist, wird eine Session gestartet und man wird auf die Startseite weitergeleitet. Bei fehlerhaften Eingaben wird eine entsprechende Fehlermeldung angezeigt.

Ein eigenes Benutzerkonto kann zurzeit nicht über die Anwendung erstellt werden. Neue Accounts werden ausschliesslich durch die Administratorinnen eingerichtet. Dies soll später erweitert werden, z. B. durch eine Selbstregistrierung oder einen Adminbereich zur Benutzerverwaltung.

Dateien:

- routes/login/+page.svelte
- routes/login/+page.server.js

## 4. Erweiterungen

Im Rahmen der Anwendung wurden verschiedene Funktionen umgesetzt, die über die Grundanforderungen hinausgehen und als Erweiterungen gelten. Nachfolgend werden diese detailliert beschrieben, inklusive technischer Umsetzung und Verweise auf die entsprechenden Dateien. Die meisten Erweiterungen wurden bereits im Kapitel 3 anhand von Screenshots und Workflows veranschaulicht.

### 4.1. Dynamische Navigation je nach Benutzerrolle

Admin:



Kunde:



Gast:



Die Navigation passt sich dynamisch an den Login-Status und die Benutzerrolle an. Gäste sehen nur allgemeine Seiten und den Warenkorb. Eingeloggte Benutzer:innen sehen zusätzlich ihre Bestellungen. Administrator:innen erhalten exklusiven Zugang zur Admin-Seite. Diese dynamische Steuerung ist sauber implementiert und verbessert die User Experience.

Dateien:

- lib/components/layout/Navbar.svelte
- src/routes/+layout.svelte
- hooks.server.js

### 4.2. Benutzerauthentifizierung mit Rollen

Die Web-App unterstützt eine vollständige Authentifizierung auf Basis von Cookies. Benutzer:innen können sich einloggen, erhalten je nach Rolle (admin oder user) unterschiedliche Rechte und UI-Elemente. Die Login-Daten werden serverseitig verifiziert und in der Session gespeichert. Die Navigation verändert sich dynamisch je nach Login-Status und Rolle. Die Accounts werden aktuell durch den Admin manuell in der Datenbank erstellt – perspektivisch könnte später eine eigene Registrierungsfunktion ergänzt werden.

Dateien:

- routes/login/+page.svelte
- routes/login/+page.server.js
- routes/logout/+server.js
- hooks.server.js
- lib/server/db.js

### 4.3. Personalisierung der Boxen für einzelne Bestellungen

Obwohl jede Box eine Standardkonfiguration (Grösse, Rosenfarbe, Verpackung) besitzt, kann diese bei der Bestellung individuell überschrieben werden. Diese individuellen Anpassungen gelten nur für die einzelne Bestellung

und werden separat in der orders-Collection gespeichert. Die Auswahl erfolgt interaktiv über Buttons und Dropdowns. Die Werte werden beim Klick auf "Bestellen" via Query-Parameter übergeben und auf der nächsten Seite übernommen.

Dateien:

- routes/boxes/[id]/+page.svelte
- routes/boxes/[id]/order/+page.svelte
- routes/boxes/[id]/order/+page.server.js
- lib/server/db.js

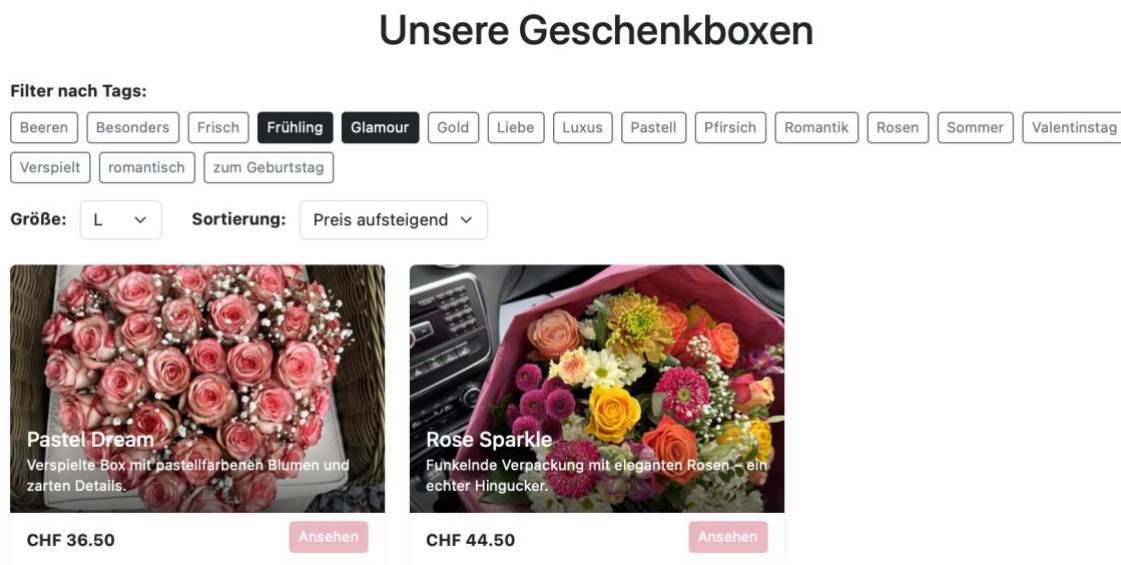
#### 4.4. Bestellübersicht für eingeloggte Benutzer:innen

Eingeloggte Benutzer:innen sehen ihre vergangenen Bestellungen auf der Seite /orders. Die Bestellungen werden nutzerspezifisch mit userId in der MongoDB gespeichert und beim Laden der Seite aus der Datenbank gefiltert. Angezeigt werden dabei auch die individuell gewählten Optionen.

Dateien:

- routes/orders/+page.svelte
- routes/orders/+page.server.js
- lib/server/db.js

#### 4.5. Filter- und Sortierfunktion auf der Boxenübersicht



Die Seite /boxes bietet eine umfangreiche Filtermöglichkeit. Benutzer:innen können nach Tags filtern, nach Boxengrösse selektieren sowie die Boxen nach Preis auf- oder absteigend sortieren. Die Filter wirken sich direkt auf die Client-seitige Darstellung aus.

Dateien:

- routes/boxes/+page.svelte

