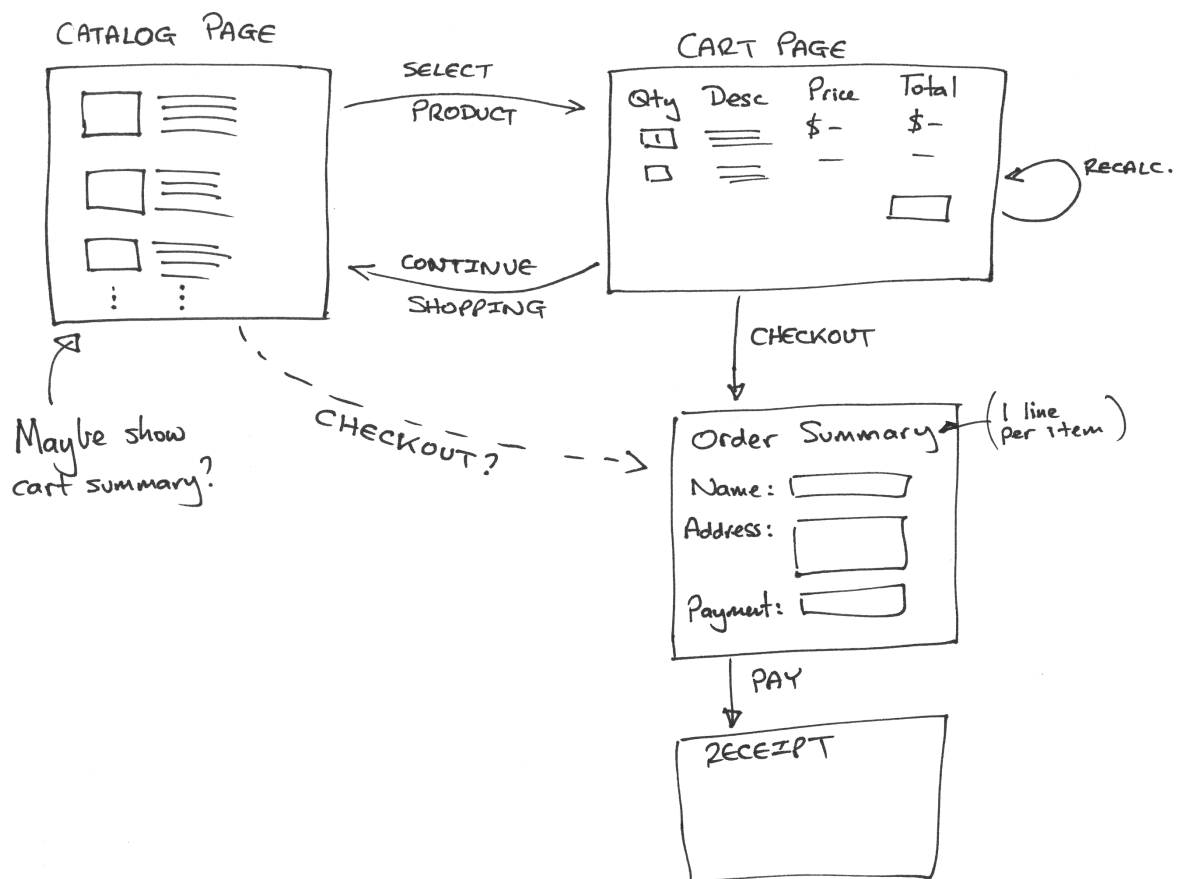


Einstieg in das Shop-Projekt

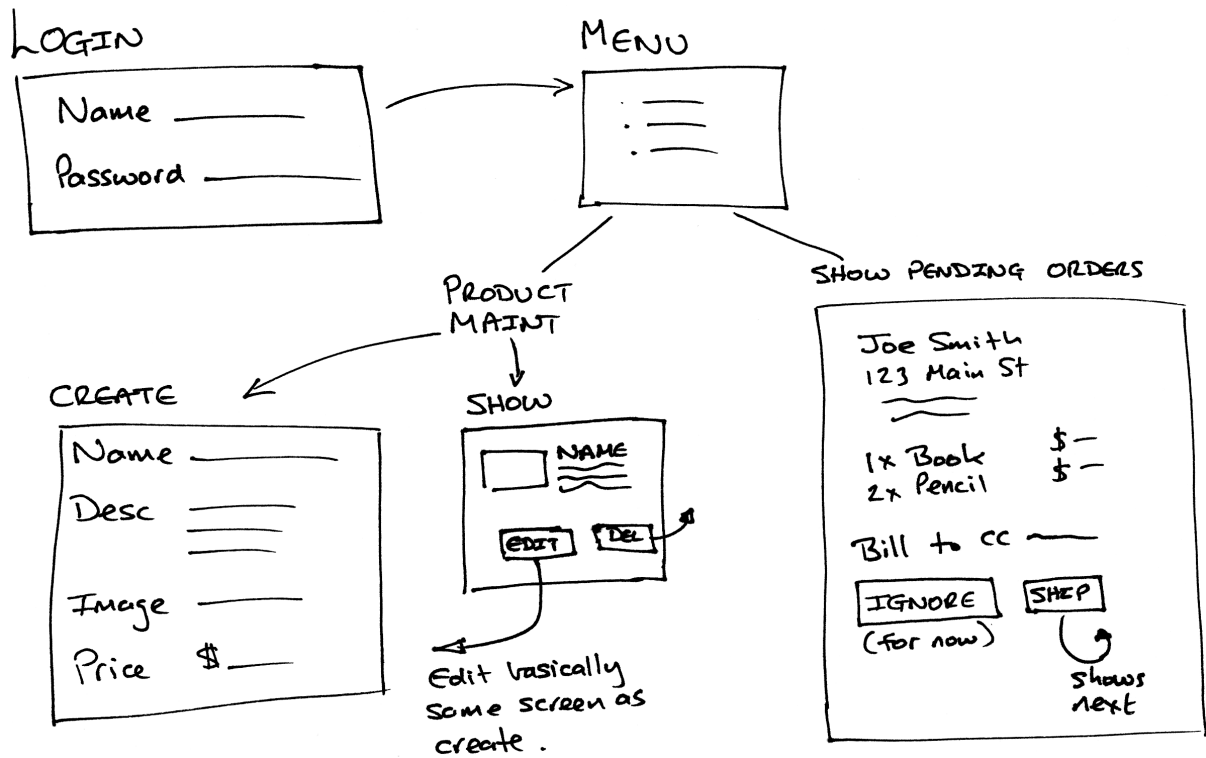
Es soll ein Buch-Shop programmiert werden. Der Shop heisst 'depot'.

Als CSS_Framework wird Foundation verwendet.

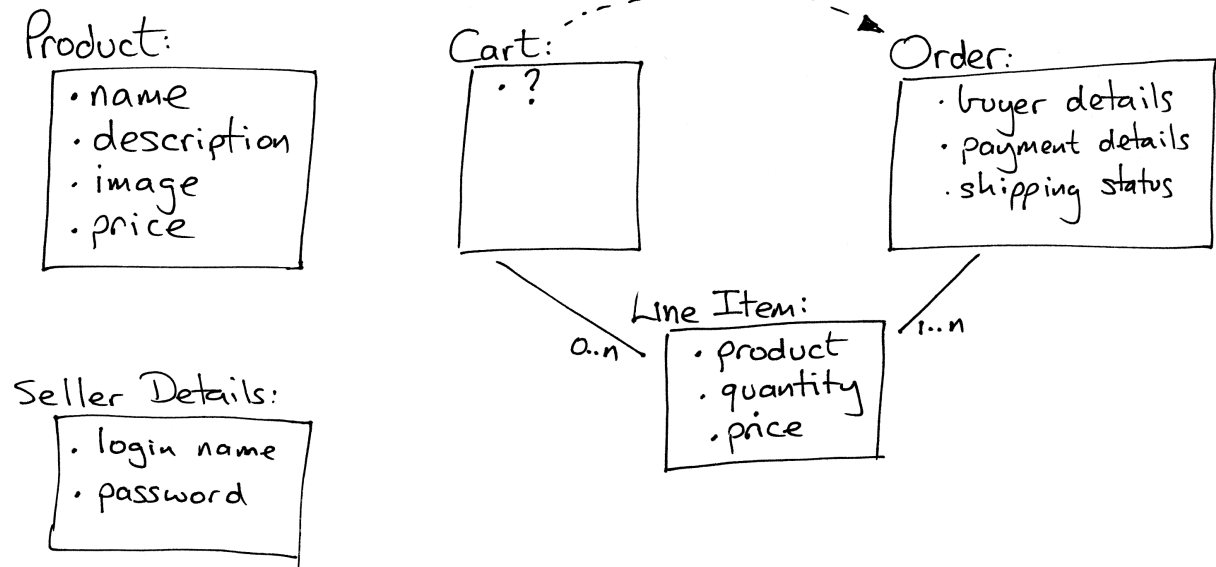
Perspektive des Käufers



Perspektive des Verkäufers



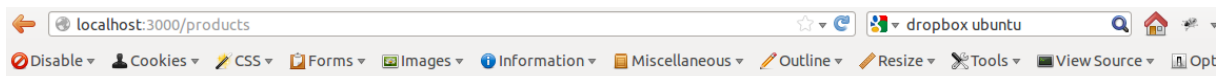
Struktur der Daten






PROJEKTSTART

1. Erstellen Sie ein neues Projekt

2. Führen Sie einen Scaffold für die Tabelle products mit den Spalten: description, image_url, title, price durch
3. Passen Sie das Layout soweit an, dass die Übersichtsseite folgende Ansicht zeigt



Listing products

	Rails. Expertenwissen Ruby on Rails ist seit vielen Jahren als modernes, agiles Webframework bekannt...	Show Edit Destroy
	Rails. Galileo Das erste und umfassendste Handbuch zu Ruby on Rails 3! Einiges hat sich geta...	Show Edit Destroy
	Rails. Openbook Mit diesem Buch beweist Stefan Wintermeyer, dass Ruby on Rails 3.2 auch für E...	Show Edit Destroy

[New Product](#)


4. Informieren Sie sich über den Befehl rake db:seed und erstellen Sie eine entsprechende Datei mit drei Produkten.
5. Sorgen Sie mit Hilfe von Validatoren, dass
 - in jeder der Spalten ein Wert stehen muss
 - der Preis nicht 0 sein darf
 - die image_url ein Bild sein muss
 - die title-Spalte nicht doppelt vorkommt

Store – Funktionalität

1. Erstellen Sie einen Controller Store mit einer Action. Die Produkte sollen für den Kunden angezeigt werden.
2. Diese Seite soll die Standard-Root-Seite der Applikation sein
3. Die Darstellung sollte in etwa folgendermaßen sein:

localhost:3000

Book Catalogue



Rails. Expertenwissen

Ruby on Rails ist seit vielen Jahren als modernes, agiles Webframework bekannt. Das Release von Version 3.1 setzt neue Maßstäbe und markiert einen wichtigen Meilenstein in der Evolution des Open-Source-Projekts. Ob die Implementierung von komplexen Anwendungsfällen oder die Erstellung von standardkonformen Schnittstellen: Mit Ruby on Rails können nahezu sämtliche Aufgaben im Webumfeld gelöst werden. Dank bewährter Vorgehensweisen und Prinzipien wird eine schnelle Entwicklung unterstützt, die zu leicht wartbaren und flexiblen Anwendungen führt. Mit der Veröffentlichung von Version 3 integriert sich Ruby on Rails besser denn je in das Ökosystem von Ruby und erlaubt, einzelne Framework-Komponenten gegen alternative Technologien auszutauschen. Als Beispiel ist die Verwendung von NoSQL-Datenbanken zu nennen. Die Autoren führen praxisnah in die Entwicklung mit Ruby on Rails 3.1 ein und geben ihre langjährigen Erfahrungen weiter.

39.95

4. Nutzen Sie für den Preis einen Helper.

5. Fügen Sie der Seite ein minimales Layout hinzu. Aussehen in etwa:

localhost:3000



- [Home](#)
- [Questions](#)
- [News](#)
- [Contact](#)

RAILS BOOKS

Book Catalogue



Rails. Expertenwissen

Ruby on Rails ist seit vielen Jahren als modernes, agiles Webframework bekannt. Das Release von Version 3.1 setzt neue Maßstäbe und markiert einen wichtigen Meilenstein in der Evolution des Open-Source-Projekts. Ob die Implementierung von komplexen Anwendungsfällen oder die Erstellung von standardkonformen Schnittstellen: Mit Ruby on Rails können nahezu sämtliche Aufgaben im Webumfeld gelöst werden. Dank bewährter Vorgehensweisen und Prinzipien wird eine schnelle Entwicklung unterstützt, die zu leicht wartbaren und flexiblen Anwendungen führt. Mit der Veröffentlichung von Version 3 integriert sich Ruby on Rails besser denn je in das Ökosystem von Ruby und erlaubt, einzelne Framework-Komponenten gegen alternative Technologien auszutauschen. Als Beispiel ist die Verwendung von NoSQL-Datenbanken zu nennen. Die Autoren führen praxisnah in die Entwicklung mit Ruby on Rails 3.1 ein und geben ihre langjährigen Erfahrungen weiter.

39,95 €



Rails. Galileo

Das erste und umfassendste Handbuch zu Ruby on Rails 3! Einiges hat sich getan: Rails 3 integriert das Ruby-Framework »Merb« und erweitert damit enorm sein Potenzial. Die neue

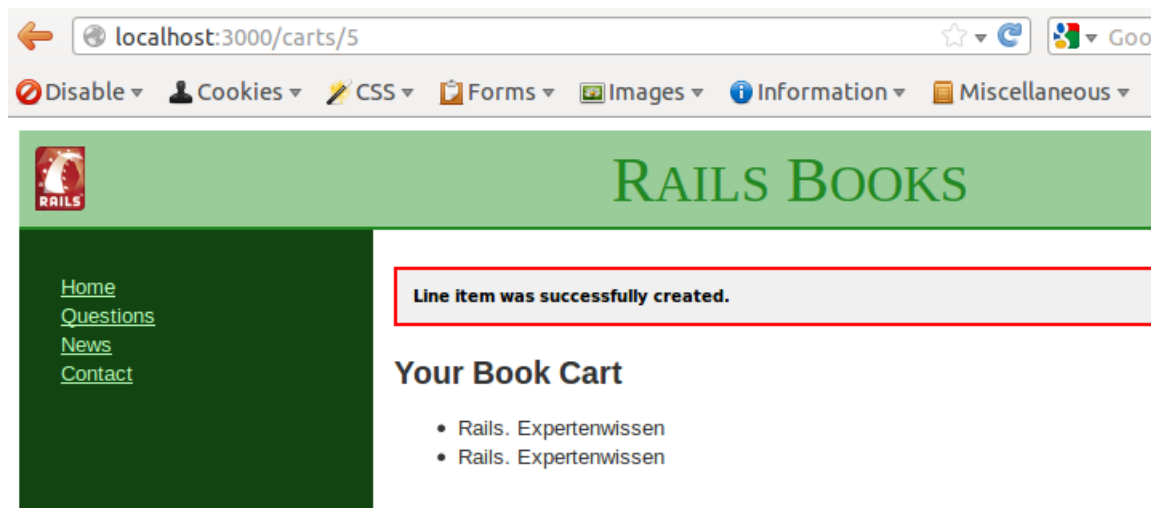
Warenkorb

Warenkorb: Erstellen Sie einen Warenkorb cart über einen Scaffold
Definieren Sie eine Funktion `current_cart` auf die alle Controller zugreifen können.
Die Funktion überprüft ob es eine Session für den Cart gibt, falls nein wird in der Tabelle `carts` ein neuer Eintrag erzeugt und die ID in der Session gespeichert.
Erstellen Sie einen Scaffold für `line_items` mit den Spalten: `product_id` `cart_id`
Anpassen der Models mit einer 1:n-Beziehung zwischen `product` und `line_items` und einer 1:n-Beziehung zwischen `cart` und `line_items`
Erstellen Sie im Model `products` eine `hook-method` für das `before_destroy` – Ereignis, das überprüft, ob es zu einem konkreten Produkt Einträge in der `line_items` Tabelle gibt
Fügen Sie Im Store-View einen Button pro Product: Add to cart hinzu
Passen Sie die Create-Methode von `line_item` an
Passen Sie die Show-View_Methode von `Cart` soweit an, dass die einzelnen Warenkorbbelemente angezeigt werden
Führen Sie die funktionalen Tests durch

Store-Ansicht mit Warenkorb-Button:



Seite nach zweimaligem Hinzufügen des Buches Rails.Expertenwissen



Warenkorb verbessert

Der bisherige Warenkorb hat mindestens einen Nachteil: Wird ein Produkt zwei Mal hinzugefügt, dann wird es auch zwei Mal angezeigt.

Verbessern Sie die Funktionalität indem Sie zum Warenkorb (LineItem) eine Anzahl hinzufügen und entsprechende Überprüfungen vornehmen.

Beim Anzeigen eines Warenkorbes sieht die URL so aus: <http://localhost:3000/carts/5>.

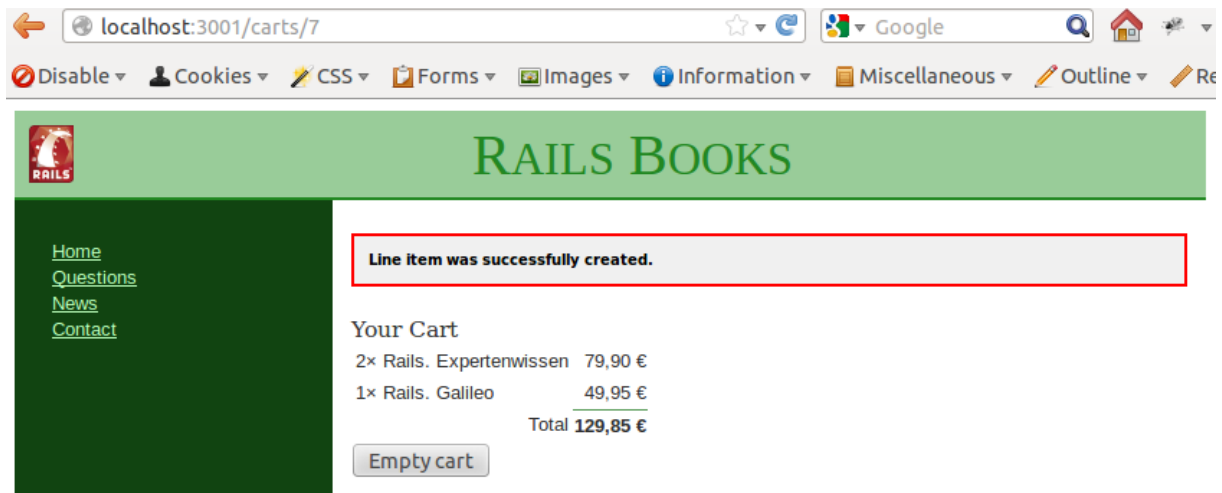
Ein Warenkorb mit der ID 5 wird angezeigt.

Es soll ausgeschlossen werden, dass durch eine Manipulation der URL eine Fehlermeldung erscheint. Passen Sie die show-Action des carts-controllers so an, dass eine Flash-Meldung erscheint und kein interner Fehler angezeigt wird, wenn es eine Warenkorb-ID nicht gibt.

Fügen Sie einen Button hinzu, der den Warenkorb leert.

Fügen Sie eine abschließende Addition hinzu.

Aussehen des Warenkorbs nach Fertigstellung.



Warenkorb-Funktionalität mit optionalem Ajax

Der Warenkorb wird auf der Hauptseite des Store in den linken Bereich ausgelagert. Dort soll er immer sichtbar sein. Nach der Fertigstellung sieht der Store in etwa so aus:



Aufgaben:

Binden Sie auf der Layout-Seite ein Partial ein, das den ganzen Warenkorb enthält.

- Erstellen einer Partial-Datei (carts/_cart.html.erb) und Einpflegen der Daten
- Ändern der Application-Layout-Datei
- Anpassen des Store-Controllers mit Variable: @cart
- Anpassen der Style-Infos

Für die Fortgeschrittenen, die eine Herausforderung suchen:

Ändern Sie den entsprechenden Controller, so dass nach dem Hinzufügen eines Items zum Warenkorb die Store-Seite geladen wird

Ändern Sie den Button: Ware in Warenkorb, in einen Ajax-Aufruf

Passen Sie die Create-Action des Line_Items_Controller so an, dass eine Weiterleitung zu eine js-Template mit dem gleichen Namen angesprochen wird. Erstellen Sie die Datei, die den Warenkorb anzeigt.

Lagern Sie den Aufbau der einzelnen Cart-Items in ein Partial aus.

Plus-Aufgabe: Beim Hinzufügen eines neuen Elements soll dieser neue Eintrag im Warenkorb kurz aufleuchten

Plus-Aufgabe: Schreiben Sie einen Helper, der dafür sorgt, dass der Warenkorb nur angezeigt wird, wenn etwas drin ist

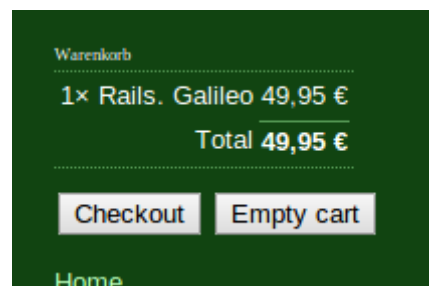
Plus-Plus-Aufgabe: der Klick auf das Bild soll ebenfalls den Warenkorb bestücken. Realisieren mit Javascript.

Check out.

Der Warenkorb ist in die Applikation integriert, nun wird der Bestellvorgang programmiert.

Aufgaben:

1. Scaffold fuer order, mit den Spalten name, address, email, pay_type
2. Migration fuer line_item, bei der eine neue Spalte order_id hinzugefuegt wird
3. CHECKOUT – Button bei Warenkorb, der das neue Order-Fomular laedt.



4. Vor dem Aufbau des Formulars pruefen, ob der der Warenkorb leer ist, falls ja, zum Store weiterleiten mit der der Notiz, dass der Warenkorb leer ist.

5. Ergaenzen Sie das Formular zur Eingabe des Bezahltyps.

Email

Pay type

Select a payment method ▼

Order erstellen

Select a payment method

- Check
- Credit card
- Purchase order

6. Passen Sie die Models `Line_items` und `orders` an. Ergänzen Sie `order` in dem Sie die Email-Adresse validieren und sicherstellen, dass nur eine der oben aufgeführten Bezahlarten akzeptiert werden.

7. Beim Speichern einer Adresse (order-Form) wird nach dem Speicher-Vorgang die order-Id zu den line-items hinzugefügt, dann wird der Warenkorb auf Null gesetzt.

Der Checkout-Prozess wird erweitert, in mehrere Teile aufgespalten und mit einem Wizard, den das gem 'wicked' zur Verfügung stellt, in einzelne Schritte unterteilt.

Der Checkout-Prozess hat folgende Einzelschritte:

- Eingabe der Adresse des Kunden
- Eingabe der Bezahlmethode
- Anzeige der Produkte und Bestätigung der Bestellung
- Fortgeschritten: Eingabe einer Lieferanschrift, soll optional sein. Welche Änderungen sind an welchem Modell nötig.
- Fortgeschritten: Professionalisieren Sie den Checkout-Vorgang mit der Anpassung des Layouts, der Anzeige des jeweils aktuellen Schrittes, der Möglichkeit, einen

Schritt vor- oder zurückzuspringen und einem Abbrechen-Button, der den Zustand vor dem Checkout wieder herstellt.

1. Warenkorb	2. Adresseingabe	3. Zahlungsart	4. Bestellung prüfen	5. Bestätigung
--------------	------------------	----------------	----------------------	----------------

2. Adresse eingeben

Anrede

Vorname

Nachname

Adresszusatz (optional)

Straße und Hausnummer

Postleitzahl und Ort DE-

Email-Adresse

Telefonnummer (optional)

[Schritt zurück](#) [Nächster Schritt](#) [Bestellung abbrechen](#)

Vorübung:

Legen Sie ein neues Projekt tasks an.

Führen Sie einen Scaffold durch. Die Tabelle tasks hat eine String-Spalte und heißt name, und eine Boolean-Spalte mit dem Namen complete und dem Default-Wert false.

Legen Sie mit einem Rake-Task 100 Datensätze an

Nutzen Sie das will_paginate gem um in der show-action jeden task einzeln anzeigen zu lassen.

Weiterentwicklung Blog:

Führen Sie die Paginierung auch für die comments durch, sodass bei der Anzeige eines Artikels folgende Sicht erscheint:

Depot: Anzeigen der Bestellungen

Die Anzeige aller Bestellungen wurde vom Scaffold durch das Erzeugen der entsprechenden index-Action vorbereitet. Nun kommt es darauf an, die Funktionalitaet zu verbessern.

Aufgaben:

1. Erzeugen von 100 Bestellungen.

Legen Sie im Verzeichnis script eine Datei load_orders.rb an. Wird diese Datei aufgerufen, dann sollen automatisiert 100 Bestellvorgaenge in die Datenbank eingetragen werden.

Alternativ können Sie auch mit einem rake-Task arbeiten.

2. Fügen Sie die 100 Datensaeetze hinzu.

Rufen Sie zur Kontrolle die index-Action von orders auf.

3. Nutzen Sie das gem: will_paginate um folgende Darstellung zu erreichen:

Listing orders

Previous Label 1 2 3 4 5 6 7 8 9 10 Next Label

Name	Address	Email	Pay type	
Customer 100	100 Main Street	customer-100@example.com	Check	Show Edit Destroy
Customer 99	99 Main Street	customer-99@example.com	Check	Show Edit Destroy
Customer 98	98 Main Street	customer-98@example.com	Check	Show Edit Destroy
Customer 97	97 Main Street	customer-97@example.com	Check	Show Edit Destroy
Customer 96	96 Main Street	customer-96@example.com	Check	Show Edit Destroy
Customer 95	95 Main Street	customer-95@example.com	Check	Show Edit Destroy
Customer 94	94 Main Street	customer-94@example.com	Check	Show Edit Destroy
Customer 93	93 Main Street	customer-93@example.com	Check	Show Edit Destroy
Customer 92	92 Main Street	customer-92@example.com	Check	Show Edit Destroy
Customer 91	91 Main Street	customer-91@example.com	Check	Show Edit Destroy

[New Order](#)

