Aufgabenblatt 6 - Prüfungsleistung

Präambel Die folgenden Aufgaben beschreiben die Anforderungen an zwei Apps, welche Clients des FruitShop-Servers sind. Laden Sie diesen aus Moodle herunter und starten Sie den Server per `swift run`. Auf `http://127.0.0.1:8080` lässt sich die API-Dokumentation betrachten. Machen Sie sich mit der API und insbesondere dem Paginationsystem vertraut.

Listen werden mit den Query-Parametern `page` und `per` abgerufen. Beispiel: `http://127.0.0.1:8080/api/products?page=1&per=10`. "Page" ist die Seitennummer, welche mit 1 beginnt. "Per" gibt an, wieviele Elemente pro Seite abgerufen werden sollen. Listeneinträge sind immer alphabetisch sortiert.

Der Fruit Shop Server implementiert kein Rollen- oder Rechtesystem. Außerdem gibt es keine Nutzer oder Sessions. Das Bezahlen einer Bestellung wird simuliert, indem der Client eine beliebige UUID als Transaktions-ID verschickt. Der Server sendet keine Emails und ruft auch keine andere HTTP-Services ab. Die Daten werden im Projektverzeichnis unter `database.sqlite` abgespeichert.

Aufgabe 1 Darstellung von Produkten, Händlern und Kategorien

(a) Stellen Sie die Produkte, Händler und Kategorien dar. Nutzen Sie dafür eine TabView und drei entsprechende Seiten mit Listen. Die Produktliste muss mindestens Produktname, Produktbild und Preis darstellen. Ignorieren Sie das Pagination-System: Zunächst soll also die erste Seite mit 10 Elementen angezeigt werden.

(b) Erstellen Sie jeweils Detailseiten, welche erreicht werden, wenn auf einen Eintrag geklickt wird.

(c) Auf der Produktdetailseite müssen mindestens Name des Händlers und Name der Kategorie dargestellt werden.

(d) Von der Produktdetailseite kann der Nutzer auf die entsprechende Händlerdetailseite und Kategoriedetailseite gelangen.

Aufgabe 2 Pagination

Die folgenden Aufgaben beziehen sich auf die Produkt-, Händler- und Kategorieliste.

(a) Wenn der Nutzer in einer Liste komplett heruntergescrollt ist, sollen 5 weitere Elemente geladen werden.

(b) Konfigurieren Sie den Request der Listen, sodass immer 5 Elemente pro Seite abgerufen werden.

Aufgabe 3 Warenkorb

(a) Um einen Warenkorb umzusetzen, erzeugen und entwerfen Sie eine GRDB-Datenbank. Es soll eine Tabelle mit dem Namen "CartEntry" geben.

(b) Fügen Sie einen Button auf der Produktdetailseite hinzu, welcher das Produkt in den Warenkorb einfügt.

(c) Erstellen Sie eine weitere Seite, welche über die TabBar erreichbar ist und den Inhalt des Warenkorbs darstellt. Beim Tippen auf einen Eintrag soll die Produktdetailseite erreicht werden.

(d) Geben Sie im Warenkorb dem Kunden die Möglichkeit, die Anzahl pro Produkt zu setzen.

(e) Per Swipe-To-Delete (also per Swipe von rechts nach links) soll ein Warenkorbeintrag gelöscht werden können.

Aufgabe 4 Image Cache

(a) Speichern Sie die Produktbilder im temporären Verzeichnis ab, indem Sie sich den Pfad vom System geben lassen. Nutzen Sie `UIImage.pngData()` und die UUID des Produktes als Dateinamen. Beispiel: `64563A13-D874-4C0A-926C-10D931A1E55D.png`.

(b) Ändern Sie das Abrufen der Produktbilder aus Aufgabe 1 ab, sodass das abgespeicherte Bild benutzt wird. Erst wenn kein Bild vorhanden ist, soll dieses per API abgerufen und auch im Cache gespeichert werden.

Aufgabe 5 Bestellung

(a) Fügen Sie einen Bestellbutton auf der Warenkorbseite hinzu. Der Titel des Bestellknopfes soll den Begriff "Kostenpflichtig Bestellen" und den Gesamtpreis der Bestellung beinhalten.

(b) Beim Tippen auf den Button soll eine Bestellung erzeugt und an `/api/orders` gesendet werden.

(c) Zeigen Sie eine View an, welche einen großen Bezahlknopf enthält und die Route `/api/orders/:id/pay` abruft. Für den Parameter `paypalTransactionId` erzeugen Sie eine neue, beliebige UUID.

(d) Zeigen Sie daraufhin eine Erfolgsmeldung an. Die Bestellung muss nicht weiter gespeichert oder dargestellt werden.

Aufgabe 6 Händler App

(a) Erstellen Sie eine weitere iPhone-App zur Verwaltung der Bestellungen. Zeigen Sie alle Bestellungen in einer Liste an.

Erzeugen Sie Views und Funktionalitäten um die folgenden Routen zu bedienen:

(b) `PUT /api/orders/:id/complete` (Nur möglich für Bestellungen mit Zustand "processing")

(c) `PUT /api/orders/:id/cancel`

(d) `DELETE /api/orders/:id` (Nur möglich für Bestellungen mit Zustand "cancelled")