

Is this the Krusty Krab?



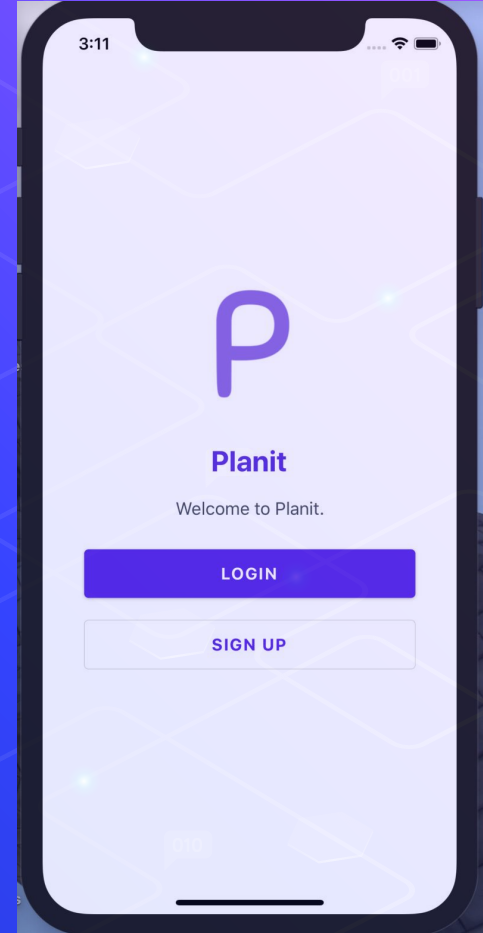
THIS IS THE KRUSTY KRAB!

THE TEAM

- ⬡ Hemant Bhanot
- ⬡ Logan Chester
- ⬡ Jason Conte
- ⬡ XiangQian Hong
- ⬡ Jason Hu
- ⬡ Gnanaswarup Srinivasan
- ⬡ Jackie Tran

OUR PURPOSE

As a team, we used scrum tactics and a REST-based MVC design to build a working implementation of Planit - a traveller-friendly mobile application envisioned by Flora Ng and Lora Haskul.



Planit

Key Features

1. Registration/Login
2. Itinerary Creation
3. Adding/Deleting Events
4. Google Maps API

Feature #1: Registration/Login

- ⬡ Registration and Login with data from database
- ⬡ Every user has a unique itinerary
- ⬡ Itineraries are saved for each user

Feature #2: Itinerary Creation

- Itineraries are created based on filters passed by user
- Events chosen based on the filters and location
- Transportation between events chosen based on filters

Feature #3: Adding/Deleting Events

- ⬡ More events available on Explore Page
- ⬡ User can add events for anytime in the itinerary
 - Event conflicts are handled
- ⬡ User can delete any events in the itinerary

Feature #4: Google Maps API

- ⬡ From filters passed by user, we grab events using Google Maps API
- ⬡ Gets REAL events in your proximity
- ⬡ Gets REAL transportation times between events
- ⬡ Show path to events on a map

Process

The background features a complex network of white lines forming a grid-like pattern. Several hexagonal shapes are scattered throughout, some containing binary code (011, 001, 100). Small blue dots are also visible along the lines.

Process

Highlights

- Scheduling Team Meetings
- Communication

Difficulties

- Github Branching
- Learning New Technology
- API Documentation

Techniques

- Splitting into Frontend and Backend teams
- Frontend Modular Design Pattern

Logan Chester: Contributions

- Itinerary Creation
 - Time/Date Filters
 - Budget/Distance Filters
 - Location Selection
- Itinerary Event Mapping
 - Front-End Map Routing Implementation



XiangQian Hong: Contributions

- Rendering of Itinerary
 - Events
 - Transportation
- Front-End Architecture
- Add/Delete Events

Gnanaswarup Srinivasan: Contributions

- User Registration/Login
 - App entry screen
 - Login/register screens
 - Navigation logic between the screens
 - RegEx pattern matching for email validity
- User Location Selection

Hemant Bhanot: Contributions

- Google Maps Places API
 - Nearby place search queries/requests
 - Max Distance
 - Types of activities
 - Budget (Price level)
- Google Maps Routes API
 - Transportation between events
 - Types of transportation
 - Cost and location

Jason Conte: Contributions

- Itinerary
 - Creation Algorithm
 - Event Scoring Algorithm
- Users
 - Objects
 - Database
- Event Database

Jason Hu: Contributions

- Itinerary Manipulation
 - Adding Events
 - Deleting Events
 - Event Time Evaluation filter
- REST Endpoint Mappings for storing and providing itinerary information
- Software architecture diagram w/ Jackie

Jackie Tran: Contributions

- Itinerary Manipulation
 - Adding Events (Handling Conflicts)
- Explore Page Data
 - Choosing Events to Send to FrontEnd
- Useful REST endpoints for Frontend Team



Software Architecture

View

<<React Native>>
iOS/Android User Interface

Controller

<<Java, Maven, Spring>>
UserController

+ register user
+ login user
+ check password

<<Java, Maven, Spring>>
ItineraryController

+ create itinerary
+ view itinerary
+ check itinerary
+ add event
+ delete event

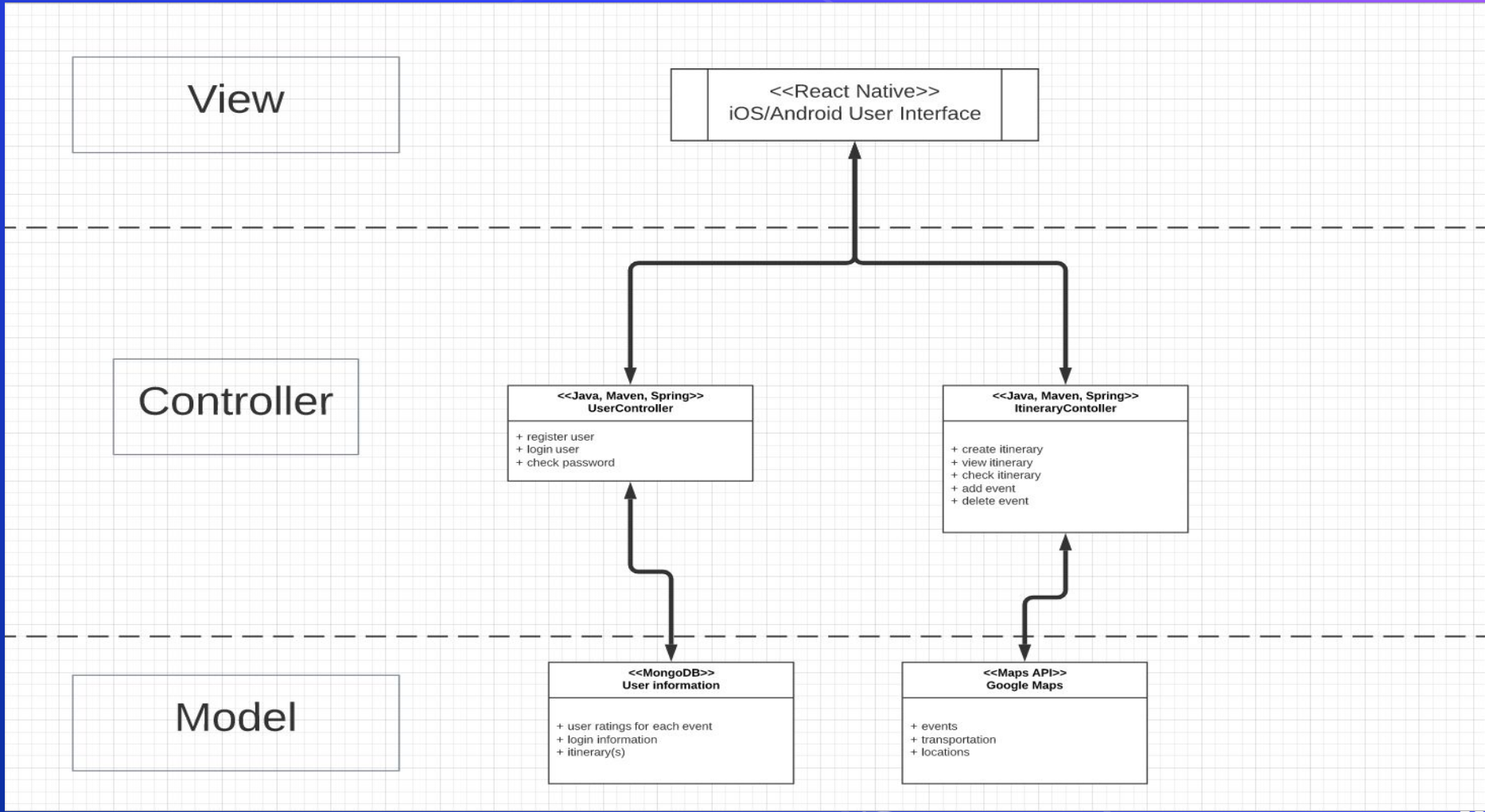
Model

<<MongoDB>>
User information

+ user ratings for each event
+ login information
+ itinerary(s)

<<Maps API>>
Google Maps

+ events
+ transportation
+ locations



Model

MongoDB

- Store users information and itinerary(s)

Google Maps API

- Called to obtain:
 - Events
 - Transportation

Itinerary Controller

- Create Itinerary
- View Itinerary
- Add Events
- Delete Events

```
358 @PutMapping("/deleteEvent")
359 @ public ResponseEntity<?> deleteEvent(@RequestBody Map<String, String> body) {
360     String eventId = new String(body.get("eventId"));
361     user.getItinerary().deleteEvent(eventId);
362     mpd.updateUser(user);
363     return ResponseEntity.ok().build();
364 }
365
366 @PutMapping("/addRating")
367 @ public ResponseEntity<?> addRating(@RequestBody Map<String, String> body) {
368     user.getEventRatings().put(body.get("eventId"), Integer.parseInt(body.get("rating")));
369     mpd.updateUser(user);
370     return ResponseEntity.ok().build();
371 }
372
373 @PutMapping("/deleteRating")
374 @ public ResponseEntity<?> deleteRating(@RequestBody Map<String, String> body) {
375     user.getEventRatings().remove(body.get("eventId"));
376     mpd.updateUser(user);
377     return ResponseEntity.ok().build();
378 }
379
380 @PutMapping("/addVisitedEvent")
381 @ public ResponseEntity<?> addVisitedEvent(@RequestBody Map<String, String> body) {
382     user.getVisitedEvents().add(body.get("eventId"));
383     mpd.updateUser(user);
384     return ResponseEntity.ok().build();
385 }
```

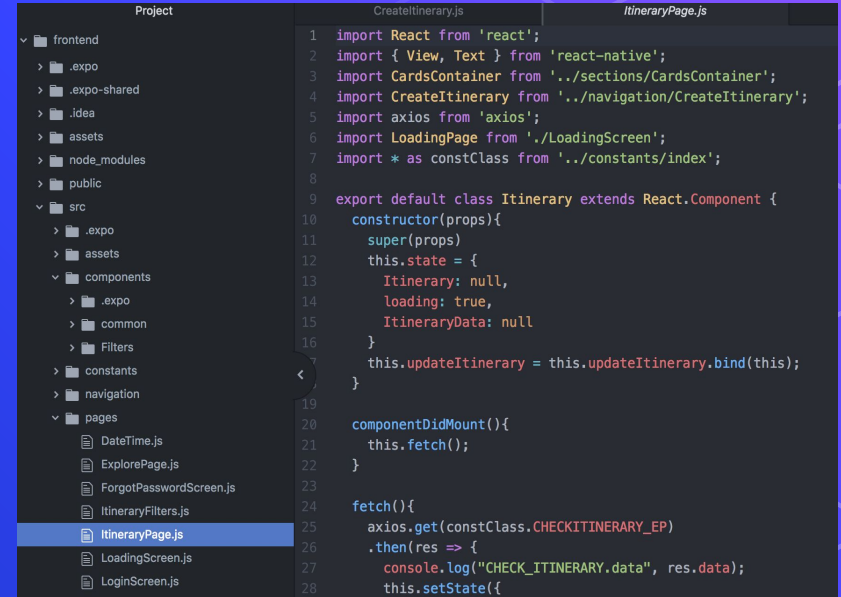
User Controller

- User Registration
- User Login
- User Credentials
- Fetch Itinerary

```
431 @PutMapping("/login")
432 @ public void login(@RequestBody User body) { this.user = mpd.readUser(body.getEmail()); }
435
436 @PostMapping("/checkPw")
437 @ public String checkPw(@RequestBody Map<String, String> body) {
438     User user = mpd.readUser(body.get("email"));
439     if(user != null && user.getPassword().equals(body.get("password"))){
440         return "valid";
441     }
442     else{
443         return "invalid";
444     }
445 }
446
447 @PostMapping("/register")
448 @ public void register(@RequestBody Map<String, String> body) {
449     System.out.println(body);
450     this.user = new User();
451     this.itin = this.user.getItinerary();
452     this.user.setUsername(body.get("username"));
453     this.user.setPassword(body.get("password"));
454     this.user.setEmail(body.get("email"));
455     mpd.createUser(this.user);
456     // userService.addUser(user);
457 }
```

View (Client)

- React Native
- Modular Design Pattern
- Handles User Data Input
- Interacts with Controllers to Render Application using REST Endpoints



The screenshot displays a code editor with a project file structure on the left and the code for `ItineraryPage.js` on the right. The file structure shows a `Project` directory containing `frontend`, `expo`, `expo-shared`, `idea`, `assets`, `node_modules`, `public`, `src`, `components`, `constants`, `navigation`, and `pages`. The `pages` directory contains `DateTime.js`, `ExplorePage.js`, `ForgotPasswordScreen.js`, `ItineraryFilters.js`, `ItineraryPage.js` (selected), `LoadingScreen.js`, and `LoginScreen.js`. The code for `ItineraryPage.js` is as follows:

```
1 import React from 'react';
2 import { View, Text } from 'react-native';
3 import CardsContainer from '../sections/CardsContainer';
4 import CreateItinerary from '../navigation/CreateItinerary';
5 import axios from 'axios';
6 import LoadingPage from './LoadingScreen';
7 import * as constClass from '../constants/index';
8
9 export default class Itinerary extends React.Component {
10   constructor(props){
11     super(props)
12     this.state = {
13       Itinerary: null,
14       loading: true,
15       ItineraryData: null
16     }
17     this.updateItinerary = this.updateItinerary.bind(this);
18   }
19
20   componentDidMount(){
21     this.fetch();
22   }
23
24   fetch(){
25     axios.get(constClass.CHECKITINERARY_EP)
26       .then(res => {
27         console.log("CHECK_ITINERARY.data", res.data);
28         this.setState({
```

Tools

- Java
- MongoDB
- Google Maps API
- React Native
- SpringBoot
- Maven
- Jira
- Postman
- Git/Github
- Expo

Technical Challenges

- React Native State Management
- React Native Dynamic Photo Rendering
- Google Maps API Documentation Outdated
- Cost of API Calls (Testing and Demo)

Thank You For Listening!

Please feel free to ask questions.

