Dokumentation - Hermes

Projekttagebuch

Aufgabe	Zuständiger	Datum
Erster Entwurf von ERM/RM	Jannik & Noah	30. April
ERM/RM überarbeitet	Jannik & Noah	2. Mai
Klassendiagramm erstellt	Jannik & Noah	2. Mai
Entwurf von REST-API (Endpunkte)	Jannik & Noah	5. Mai
REST-API Endpunkte, Klassendiagramm und ERM/RM überarbeitet	Jannik & Noah	6. Mai
Logo erstellt	Jannik & Noah	6. Mai
Klassendiagramm und ERM/RM überarbeitet	Jannik & Noah	7. Mai
Klassen hinzugefügt zu Flutter	Jannik	8. Mai
BottomNavBar designed und zum Projekt hinzugefügt	Jannik	8. Mai
Map zur HomePage hinzugefügt (OpenStreetMap)	Jannik	9. Mai
Collection Page (Sammelkarten-Seite) hinzugefügt	Noah	9. Mai
Location Permition hinzugefügt	Jannik	9. Mai
Funktion zum Zoomen zur aktuellen Position hinzugefügt	Jannik	17. Mai
Erste Version von Tracking hinzugefügt	Jannik	17. Mai
Login- und Settingspage hinzugefügt	Jannik	17. Mai
Schriftart geändert	Noah	19. Mai
ERM/RM überarbeitet	Jannik & Noah	21. Mai

Aufgabe	Zuständiger	Datum
BottomNavBar responsive gemacht	Jannik	21. Mai
Problem beim Wechseln der Seiten behoben, da noch auf Widgets in der alten Page zugegriffen wurde	Jannik	21. Mai
BottomNavBar verändert	Jannik	21. Mai
Appbar zur Einstellungsseite hinzugefügt	Jannik	21. Mai
Backend und Yaml hinzugefügt	Jannik	21. Mai
Erfolgcircle hinzugefügt (kleinen Erfolge in der Einstellungsseite)	Jannik	21. Mai
Endpunkte ausprogrammiert - Bestenliste und UpdateUser	Noah	21. Mai
BenutzerController fertiggestellt	Noah	23. Mai
Background-Tracking hinzugefügt	Jannik	28. Mai
Tracking wird beim Wechseln der Seiten fortgesetzt	Jannik	28. Mai
Login Page responsive gemacht	Jannik	28. Mai
BackgroundTracking gefixed	Jannik	29. Mai
Login- und Regestrierungsfunktion hinzugefügt	Jannik	29. Mai
Chaching hinzugefügt bei Login und Register (shared_preferences)	Jannik	29. Mai
SideBar in Einstellungsseite hinzugefügt	Jannik	29. Mai
Alle Endpunkte prototypmäßig ausprogrammiert	Noah	30. Mai
Daten in Einstellungsseite laden und anzeigen	Jannik	30. Mai
Endpunkte umbenannt und DB verbessert	Noah	1. Juni

Aufgabe	Zuständiger	Datum
add_erfolg Endpunkt aktualisiert	Noah	1. Juni
Freigeschaltete und nicht Freigeschaltete Erfolge zur Einstellungsseite hinzugefügt	Jannik	1. Juni
Gelaufene Distanz und Berge/Ziele in DB speichern (also wenn man seine Wanderung trackt und dann beendet wird die Distanz in die DB geschrieben)	Jannik	1. Juni
Erfolge in der Einstellungsseite responsive gemacht	Jannik	2. Juni
Passwort aus dem Cache entfernt	Jannik	2. Juni
Bestenliste_Controller fertig gemacht (10 Plätze und den eingeloggten User, wenn er nicht unter den Top 10 ist)	Noah	4. Juni
NFC-Reader (Klasse) hinzugefügt	Jannik	4. Juni
Hinzufügen Button zur Sammelkarten-Seite hinzugefügt	Noah	4. Juni
NFC-Reader in Sammelkarten-Seite integriert	Jannik	4. Juni
Hinzufügen Button in Sammelkarten-Seite geändert	Noah	4. Juni
Login- und Regestrierungsfunktion in UserManager integriert	Jannik	4. Juni
LoadUserData, Erfolge, ... in UserManager integriert	Jannik	4. Juni
System erkennt wenn neuer Erfolg freigeschaltet wurde und zeigt diesen an (Settingsseite)	Jannik	4. Juni
Globale Variable ServerIP hinzugefügt (Connection via Tailscale)	Jannik	5. Juni
Beim Hinzufügen von Sammelkarten wird nun geprüft ob man sich in der Nähe vom Ziel befindet (500 m)	Jannik	5. Juni
NFC-Reader Funktion von Klasse in Sammelkarten-Seite integriert	Noah	5. Juni
ValidierungsManager in Sammelkarten-Seite integriert (Hinzufügen von Zielen/Sammelkarten)	Noah	5. Juni
Erfolg-Abschnitt in Settings-Seite responsive gemacht	Jannik	5. Juni
Login Page responsive gemacht	Jannik	6. Juni
App prüft bevor dem Starten ob der Server erreichbar ist	Jannik	6. Juni
Fixed Bug (CircularProgressIndicator ging nicht weg)	Jannik	6. Juni
Leaderboard-Seite hinzugefügt	Jannik	6. Juni
Logging zu UserManger und ValidierungsManager hinzugefügt	Jannik	6. Juni
Logging zu TrackingService hinzugefügt	Jannik	6. Juni
Logging zu Main, ServerOffline und Settings hinzugefügt	Jannik	6. Juni
Scrollbar zur Sammelkarten-Seite hinzugefügt	Noah	6. Juni

Aufgabe	Zuständiger	Datum
Änderungen am Backend	Noah	9. Juni
Sammelkarten-Seite fertig gemacht (coole Effekte, Schwierigkeit und Bilder)	Noah	9. Juni
Erster Entwurf von UnitTests	Jannik	9. Juni
Filter-Funktion in Sammelkarten-Seite hinzugefügt	Noah	9. Juni
Fixed Bug (Platzierung im Leaderboard)	Jannik	9. Juni
Logging zu Sammelkarten-Seite hinzugefügt	Jannik	13. Juni
Logging zu ErfolgController hinzugefügt	Jannik	13. Juni
Logging zu BestenlisteController hinzugefügt	Jannik	13. Juni
Logging zu BenutzerController hinzugefügt	Jannik	13. Juni
UnitTests für UserManager, ValidierungsManager (KI)	Jannik	13. Juni
Datenbank Rolle (app) hinzugefügt	Jannik	14. Juni
An Doku gearbeitet	Jannik	14. Juni
An Doku gearbeitet	Jannik	15. Juni
An UnitTests gearbeitet	Noah	15. Juni
An Doku gearbeitet	Noah	16. Juni
Ordner organisiert	Jannik	16. Juni
Benutzername und Passwort verändern Funktion hinzugefügt	Jannik	17. Juni
Benutzer_Controller dokumentiert mit Doxygen	Noah	17. Juni
Bestenliste_Controller dokumentiert mit Doxygen	Noah	17. Juni
Erfolge_Controller und alle wichtigen Python Dateien dokumentiert mit Doxygen	Noah	17. Juni

Aufgabe	Zuständiger	Datum
Password Hashing hinzugefügt	Jannik	17. Juni
Doxygen fertiggestellt	Noah	17. Juni
Bei Autlogin wird nun geprüft, ob der User noch exisitert	Jannik	17. Juni
Info Seite hinzugefügt	Jannik	17. Juni
UnitTests für Backend	Noah	18. Juni
Frontend API Dokumentation	Jannik	18. Juni

Projektplanung

Die Phase der Projektplanung war für uns eine besonders wichtige, da hier die ganzen Ideen zu einem fast schon fertigen Projekt zusammengeflossen sind. Durch sorgfältige Planung von Komponenten ist uns einiges leichter gefallen, besonders in der Datenbank-Verwaltung und dem Teil der Rest-API. Den ersten Grob-Vorschlag für die App findet man auch in der Datei `ProjektIdee.md`. Die ersten Entwürfe vom ERM bzw. RM sind im Ordner `ERM_RM` zu finden, jedoch sind diese nicht mehr aktuell. Das Klassendiagramm ist dementsprechend auch im Ordner `Klassendiagramm` zu finden. Im Ordern `Bilder` sind Designs von der UI und den Sammelkarten, sowie das Logo. In der Datei `REST-API.md` ist auch der erste Entwurf der Endpunkte zu finden.

Umsetzungsdetails

Softwarevoraussetzungen (Versionen)

- flutter_map: ^6.0.0
- flutter_svg: ^2.0.0
- latlong2: ^0.9.0
- pdf : ^3.10.6
- path_provider: ^2.1.2
- flutter_map: ^6.0.0
- flutter_svg: ^2.0.0
- latlong2: ^0.9.0
- pdf: ^3.10.6
- path_provider: ^2.1.2
- cupertino_icons: ^1.0.8
- google_nav_bar: ^5.0.6
- geolocator: ^10.0.0
- flutter_map_location_marker: any
- location: ^5.0.0

- http: ^1.2.0
- shared_preferences: ^2.2.2
- flutter_nfc_kit: ^3.3.1
- logger: ^2.0.2
- bcrypt: ^1.1.3
- url_launcher: ^6.2.5

Umsetzung

Tracking

Das Tracking bzw. die Karte wurde mit OpenStreetMap (`flutter_map`) umgesetzt. Zudem wurde die `flutter_map_location_marker`-Bibliothek verwendet, um den aktuellen Standort des Benutzers anzuzeigen. Die Karte kann gezoomt und verschoben werden, und es gibt eine Funktion, die den Benutzer auf seine aktuelle Position zentriert. Das `location`-Paket wird verwendet, um die Standortberechtigungen zu verwalten und den aktuellen Standort des Benutzers zu erhalten. Die Strecke die man gelaufen ist, wird dann mithilfe einer Polyline auf der Karte angezeigt.

Das wär z.B. die Funktion um das Tracking zu starten:

```
void startTracking() {
    logger.i('Tracking gestartet');
    trackedRoute.clear();
    totalDistance = 0.0;
    accumulatedDuration = Duration.zero;
    startTime = DateTime.now();
    isTracking = true;

    _locationSubscription = location.onLocationChanged.listen((loc) {
      final LatLng newPoint = LatLng(loc.latitude!, loc.longitude!);
      if (trackedRoute.isNotEmpty) {
        totalDistance += _distanceCalculator(trackedRoute.last, newPoint);
      }
      trackedRoute.add(newPoint);
      onLocationUpdated?.call();
    });
  }
```

Dies ist die Funktion, die verwendet wird um die aktuelle Position zu zoomen:

```
Future<void> _zoomToCurrentLocation() async {
    final hasPermission = await _location.hasPermission();
    if (hasPermission == PermissionStatus.denied) {
      await _location.requestPermission();
      logger.i('Berechtigung zum Orten nicht erteilt... Erlaubnis wird
angefragt');
    }

    final serviceEnabled = await _location.serviceEnabled();
```

```
    if (!serviceEnabled) {
      await _location.requestService();
    }

    final currentLocation = await _location.getLocation();
    if (mounted) {
      _mapController.move(
        LatLng(currentLocation.latitude!, currentLocation.longitude!),
        20.0,
      );
    }
    logger.i('Zum aktuellen Standort gezoomt');
  }
```

Das ist die Funktion, die aufgerufen wird wenn der Button auf der HomePage gedrückt wird und je nach Status des Trackings wird es dann entweder gestartet, fortsetzt oder gestoppt:

```
void _toggleTracking() {
    if (trackingService.isTracking) {
      logger.i('Tracking Stop wurde gedrückt');
      trackingService.stopTracking();
    } else {
      if (trackingService.startTime != null) {
        logger.i('Tracking Fortsetzung wurde gedrückt');
        trackingService.resumeTracking();
      } else {
        logger.i('Tracking Start wurde gedrückt');
        trackingService.startTracking();
      }
    }
    setState(() {});
  }
```

Sammelkarten

Die Seite Sammelkarten wurde erstellt mithilfe eines Stateless Widgets. Dieses ermöglicht, Änderungen sichtbar zu machen, wenn der NFC-Chip gescannt wird. Die Funktionen in dieser Seite beinhalten das Scannen von NFC-Chips, die Anzeige von den Sammelkarten des jeweiligen Users, die Sortierung der Sammelkarten nach bestimmten Kriterien und somit einen der Kerninhalte der App Hermes. Das ganze wurde aufgeteilt in mehrere Dateien.

1. Die Datei collection_page.dart

Hier gibt es die Funktion:

```
Future<void> readNfcTag() async {
    try {
      var availability = await FlutterNfcKit.nfcAvailability;
```

```
      if (availability != NFCAvailability.available) {
        logger.w('NFC wird auf diesem Gerät nicht unterstützt oder ist
deaktiviert!');
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(content: Text('NFC wird auf diesem Gerät nicht unterstützt oder
ist deaktiviert.')),
        );
        return;
      }

      setState(() => state = 1);
      NFCTag tag = await FlutterNfcKit.poll();

      List<List<int>> data = <List<int>>[];

      if (tag.ndefAvailable == true) {
        var ndef = await FlutterNfcKit.readNDEFRecords();
        for (var record in ndef) {
          if (record.payload != null) {
            data.add(record.payload!.toList());
            logger.i('NDEF Record: ${record.payload}');
          }
        }
      }

      List<int> cleaned = data[1].sublist(3);
      int? id = int.tryParse(String.fromCharCodes(cleaned));
      if (id == null) {
        logger.e('ID konnte nicht von NFC Chip extrahiert werden');
        throw Exception("ID konnte nicht extrahiert werden.");
      }

      print("ID: $id");
      logger.i('ID erfolgreich extrahiert: $id');
      await FlutterNfcKit.finish();
      setState(() => state = 2);
      await Future.delayed(Duration(seconds: 1));
      setState(() => state = 0);

      await Validierungsmanager.AddSammelkarteNFCGPS(context, id);
      await loadZiele();
    } catch (e) {
      print("Fehler beim Lesen: $e");
      logger.w('Fehler beim Lesen des NFC Chips: $e');
      await FlutterNfcKit.finish();
      setState(() => state = 3);
      await Future.delayed(Duration(seconds: 1));
      setState(() => state = 0);
    }
  }
```

Diese liest den NFC-Chip mithilfe des Pakets FlutterNfcKit ein und gibt die Daten, die darauf sind, dementsprechend zurück - Bei uns ist das einfach die ID des Zieles.

Die folgende Widget-Struktur ermöglicht eine übersichtliche Darstellung der bereits im Besitz befinden Sammelkarten.

```
@override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Stack(
        children: [
          Container(
            color: const Color.fromRGBO(30, 30, 30, 1),
            width: double.infinity,
            padding: const EdgeInsets.only(top: 45.0),
            child: Column(
              children: [
                Row(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    const Text(
                      "Deine Sammelkarten",
                      style: TextStyle(fontSize: 28, fontFamily: "Sans", color:
Colors.white),
                    ),
                    const SizedBox(width: 10),
                    IconButton(
                      onPressed: wechsleSortierung,
                      icon: Icon(
                        sortierIcons[aktuellerSortIndex],
                        color: Colors.white,
                        size: 24,
                      ),
                      tooltip: 'Sortieren nach:
${sortierKriterien[aktuellerSortIndex]}',
                    ),
                  ],
                ),
                Expanded(
                  child: isLoading
                      ? Center(child: CircularProgressIndicator())
                      : Scrollbar(
                          thumbVisibility: true,
                          child: SingleChildScrollView(
                            child: Wrap(
                              children: collection.sammelkarten
                                  .map((karte) => MySammelkarte(karte: karte))
                                  .toList(),
                            ),
                          ),
                        ),
                ),
```

```
                const SizedBox(height: 200),
              ],
            ),
          ),
          Positioned(
            bottom: 130.0,
            left: 0,
            right: 0,
            child: Center(
              child: FloatingActionButton(
                onPressed: readNfcTag,
                backgroundColor: state == 1
                    ? Colors.red.withOpacity(0.7)
                    : const Color.fromARGB(255, 178, 180, 16).withOpacity(0.5),
                child: Icon(
                  state == 0
                      ? Icons.add_rounded
                      : state == 1
                          ? Icons.document_scanner_rounded
                          : state == 2
                              ? Icons.check_rounded
                              : Icons.close_rounded,
                  color: Colors.white,
                ),
              ),
            ),
          ),
          const Positioned(
            bottom: 10.0,
            left: 5,
            right: 5,
            child: MyBottomNavBar(
              currentIndex: 1,
            ),
          ),
        ],
      ),
    );
  }
```

Der ansprechende Effekt der Sammelkarten wird durch ein selbst erstelltes Widget sammelkarte_GUI.dart gelöst. Dieses wird erweitert durch die Variante, bei der die Sammelkarte vergrößert ist und nur aufgerufen wird, wenn die Sammelkarte angeklickt wird.

Leaderboard

Das Leaderboard wurde mithilfe einer PageView erstellt, welche das Wechseln zwischen den einzelnen Kategorieren ermöglicht. Zudem gibt es eine Klasse `leaderboardsingle`, welche das jeweilige Leaderboard je nach Kategorie anzeigt. Die ersten drei Plätze sind mit Gold, Silber und Bronze gekennzeichnet und der eingeloggte User mit Grün (Funktion `getMedalColor`). Befindet er sich nicht unter den Top 10, wird noch eine Platzierung unterhalb der Top 10 angezeigt.

Hier holt man z.B. die Daten für das Leaderboard:

```
Future<List<Map<String, dynamic>>> loadleaderboard() async {
    final prefs = await SharedPreferences.getInstance();
    int? id = prefs.getInt('id');
    final url = Uri.parse('http://$serverIP:8080/user/bestenliste?
userID=$id&filterDB=$category');
    final response = await http.get(url);
    final result = json.decode(response.body);
    print(result);

    return List<Map<String, dynamic>>.from(result);
  }
```

Funktion für die Farben der Platzierungen:

```
Future<Color> getMedalColor(int place, Map<String, dynamic> entry) async {
    int? userID = await getUserId();
    switch (place) {
      case 1:
        return const Color(0xFFFFD700); // Gold
      case 2:
        return const Color(0xFFC0C0C0); // Silber
      case 3:
        return const Color(0xFFCD7F32); // Bronze
      case _ when entry['ID'] == userID:
        return const Color(0xFF4A742F); // Aktueller User hervorheben
      default:
        return Colors.white;
    }
  }
```

Login/Registrierung

Das Login bzw. die Registierung wurden hauptsächlich einfach mit HTTP-Requests umgesetzt. Um sich nicht jedes Mal neu anmelden zu müssen, wurde der Benutzername, ID und ein Bool `isLoggedIn` mithilfe von `shared_preferences` gecached. Beim Start der App wird dann geprüft, ob `isLoggedIn` true ist und je nach dem wird dann das Login übersrprungen und direkt zur HomePage weitergeleitet.

Login-Funktion in `UserManager`:

```
static Future<bool> Login(BuildContext? context, String username, String password,
{http.Client? client,}) async {
    client ??= http.Client(); // Normalfall, das andere ist nur für die Unit-Tests
    final url = Uri.parse('http://$serverIP:8080/user/login?
benutzername=$username&passwort=$password');
    final response = await client.get(url);
```

```
    final result = json.decode(response.body);
    if (result == -1){
      logger.w('Login fehlgeschlagen!');
      if (context != null){
        showDialog(
          context: context,
          builder: (context) => AlertDialog(
            title: Text('Login fehlgeschlagen'),
            content: Text('Benutzername oder Passwort ist falsch.'),
            actions: [
              TextButton(
                onPressed: () => Navigator.of(context).pop(),
                child: Text('OK'),
              ),
            ],
          ),
        );
      }
      return false;
    }

    final prefs = await SharedPreferences.getInstance();
    await prefs.setInt('id', result); // Cachen der Daten, damit man sich nicht
jedes Mal neu anmelden muss
    await prefs.setString('username', username);
    await prefs.setBool('isLoggedIn', true);

    logger.i('Login erfolgreich');
    return true;
  }
```

Einstellungen

Die Einstellungsseite ist im Prinzip die Übersicht über die Stats und Erfolge eines Users. Sie biete auch die Möglichkeit sich wieder abzumelden und eine Funktion um den User zu bearbeiten, sowie eine Information über die App wie Version, Quellen, usw. Die Seite holt sich die Userdaten über den `UserManager`, genauer gesagt über die Funktion `loadUserData` und zeigt diese an. Zudem werden die Erfolge angezeigt, die ein User freigeschaltet hat und die es noch gibt. Zuerst werden dafür die Erfolge des Users mit der Funktion `loadUserErfolge` und anschließend der Funktion `loadAllErfolge` mitgeben, welche die noch fehlenden Erfolge ermittelt. Anschließend werden sie dann angezeigt, je nach Bildschirmgröße, werden verschieden viele in einer Reihe angezeigt. Zusätzlich wird immer wenn man die Einstellungsseite öffnet, geprüft, ob neue Erfolge freigeschaltet wurden mithilfe der Funktion `checkErfolge`. Diese zeigt dann auch kurz einen Dialog, wenn man einen neuen Erfolg fregeschaltet hat.

Die Funktion `loadUserData`:

```
static Future<Map<String, dynamic>> loadUserData({http.Client? client, }) async {
    client ??= http.Client();
    final prefs = await SharedPreferences.getInstance();
    int? id = prefs.getInt('id');
```

```
    final url = Uri.parse('http://$serverIP:8080/user/datenabfrage?user_id=$id');
    final response = await client.get(url);
    if (response.statusCode == 401){
      logger.w('User not found');
      throw Exception('User not found');
    }
    final result = json.decode(response.body);
    final urlBerge = Uri.parse('http://$serverIP:8080/erfolg/erreichteziele?
userID=$id');
    final responseBerge = await client.get(urlBerge);
    if (responseBerge.statusCode != 200){
      logger.w('Ungülte Eingabe/Fehler bei Ziel-Abfrage');
      throw Exception('Ungültige Eingabe/Fehler bei Ziel-Abfrage');
    }
    final resultBerge = json.decode(responseBerge.body);
    logger.i('Benutzerdaten erfolgreich geladen');
    return {
      'username': prefs.getString('username') ?? 'Unbekannt',
      'kmgelaufen': result['kmgelaufen'] ?? 0,
      'hoehenmeter': result['hoehenmeter'] ?? 0,
      'berge': resultBerge.length ?? 0,
    };
  }
```

Die Funktion `loadUserErfolge`:

```
static Future<List<Erfolg>> loadUserErfolge({http.Client? client, }) async {
    client ??= http.Client();
    final prefs = await SharedPreferences.getInstance();
    int? id = prefs.getInt('id');
    final url = Uri.parse('http://$serverIP:8080/erfolg/get_erfolge?userID=$id');
    final response = await client.get(url);
    if (response.statusCode != 200){
      logger.w('Fehler beim Laden der Erfolge');
      throw Exception('Fehler beim Laden der Erfolge!');
    }
    final result = json.decode(response.body);
    print(result);
    logger.i('Erfolge erfolgreich geladen');
    return (result as List)
      .map((e) => Erfolg.fromJson(e as Map<String, dynamic>))
      .toList();
  }
```

Die Funktion `checkErfolge`:

```
static Future<void> checkErfolge(BuildContext? context, {http.Client? client, })
async {
    client ??= http.Client();
```

```
    final prefs = await SharedPreferences.getInstance();
    int? id = prefs.getInt('id');
    final url = Uri.parse('http://$serverIP:8080/erfolg/check_erfolge?
userID=$id');
    final response = await client.get(url);
    if (response.statusCode != 200){
      logger.w('Fehler bei der Abfrage, ob ein neuer Erfolg freigeschaltet
wurde');
      throw Exception('Fehler bei der Abfrage, ob ein neuer Erfolg freigeschaltet
wurde');
    }
    final result = json.decode(response.body);

    if (result){
      logger.i('Erfolge erfolgreich überprüft');
      if (context != null){
        showDialog(
          context: context,
          builder: (context) => AlertDialog(
            title: Text('Neuer Erfolg!'),
            content: Text('Neuer Erfolg freischalten'),
            actions: [
              TextButton(
                onPressed: () => Navigator.of(context).pop(),
                child: Text('OK'),
              ),
            ],
          ),
        );
      }
    }
  }
```

Probleme und Lösungen

Tracking

Also das größte Problem hierbei war, dass das Tracking auch im Hintergrund weiterlaufen soll, wenn der Benutzer die App verlässt oder auf eine ander Page wechselt. Zuerst wurde das Ganze mit `background_locator_2` umgesetzt, aber dabei hat es jedoch sehr viele Probleme mit Versionen usw. gegeben, weshalb wir uns dann für das `location`-Paket entschieden haben, welches bis zum jeztigen Zeitpunkt keine Probleme gemacht hat.

Leaderboard

Das Problem hierbei war, dass wenn sich der User nicht unter den Top 10 befand, dass man nicht im Frontend nicht wusste, welche Platzierung er hat. Deshalb hat Noah noch ein zusätzliches Attribut hinzugefügt, welches die Platzierung des jeweiligen Users angibt und somit war das Problem auch wieder behoben.

Die richtigen Endpunkte machen

Es ist eine Challenge gewesen, im Backend alle Endpunkte sinnvoll zu nutzen und keine unnötigen Dinge zu implementieren. Also hat man hier vergleichsmäßig viel Zeit investiert, um ein anständiges Ergebnis zu erhalten.

Die Sammelkarten vergrößerbar machen.

Die Vergrößerung der Sammelkarten für bessere Sichtbarkeit war eine Challenge in der selben Datei umzusetzen. Für dieses Feature benötigten wir deswegen eine zusätzliche Datei, da sonst die Funktionalität nicht gegangen wäre.

Softwaretests

Die Software wurde größtenteils auf unseren Smartphones getestet aber auch auf Emulatoren, sowie auf einem Tablet. Zudem haben wir auch Unit-Tests für den `UserManager` und den `ValidierungsManager` geschrieben, welche hautpsächlich testen, ob die Funktionen richtig auf die Responses des Server reagieren. Dies wurde mithilfe von `mockito` umgesetzt, um die HTTP-Requests zu mocken.

Bedienungsanleiten

siehe `readme.md`

Quellen

Für die Bilder der Sammelkarten nahmen wir uns Bilder aus dem Internet, hier die Quellen:

Vecteezy

Diese Bilder sind unter der Fraktion "kostenlos" in der Website zu finden. Es ist lediglich von Nöten, den Urheber dieser Bilder anzugeben, welcher ebenfalls ersichtlich ist, wenn man diesen Links folgt.

- https://de.vecteezy.com/vektorkunst/48189557-himmel-sternenklar-universum-hintergrund-dunkelblau-himmel-galaxis-raum-wolke-mit-nebel-und-sterne-im-winter-nacht-natur-sternenstaub-feld-im-tief-universum-milchig-weg-galaxis
- https://de.vecteezy.com/vektorkunst/15973314-abstrakter-vektorhintergrund-mit-goldenem-farbverlauf-und-weich-leuchtender-hintergrund-luxurioses-goldhintergrunddesign-vektor-design-hintergrund
- https://de.vecteezy.com/vektorkunst/50384759-einfach-grau-gradient-hintergrund-design
- https://de.vecteezy.com/vektorkunst/46436817-gradient-bunt-hintergrund
- https://de.vecteezy.com/vektorkunst/447228-eine-mauer-aus-stein
- https://de.vecteezy.com/vektorkunst/237970-abstrakter-realistischer-holzerner-beschaffenheitshintergrund

Bilder der Berge:

Da diese Bilder nicht alle Copyright enthalten, ist es unsere Aufgabe, falls die App öffentlich wird, diese Bilder auszutauschen.

- https://www.vorarlberg.travel/wp-content/uploads/2021/04/pois/sulzfluh-1617833582.jpg
- https://img.oastatic.com/img2/6830545/2048x2048f/variant.jpg

- https://cdn.britannica.com/17/83817-050-67C814CD/Mount-Everest.jpg
- https://www.alpenverein.at/vorarlberg-bezirk-montafon_wAssets/oeavoffice_bilder/8f6b904e-5608-4dc1-81ae-cf2d4cc3f269.jpg
- https://upload.wikimedia.org/wikipedia/commons/8/8b/20150824_Watzmann%2C_Berchtesgaden_%2801982%29.jpg
- https://www.alpenklimagipfel.jetzt/media/multikarussell/26.png
- https://upload.wikimedia.org/wikipedia/commons/4/4a/Drei_tuerme_montafon.JPG
- https://upload.wikimedia.org/wikipedia/commons/a/a7/Brocken_vom_Torfhaus_neu.jpg
- https://img.oastatic.com/img2/71685665/max/variant.jpg

Info

Möchte man das Backend lokal laufen lassen muss man in dem File `lib/components/globals.dart` die IP-Adresse des Servers anpassen. Diese ist standardmäßig auf die Public IP-Adresse des Servers gesetzt.

Die **ausführbare Version** bzw. APK befindet sich im Ordner `build/app/outputs/flutter-apk/app-release.apk` oder `bin/app-release.apk`.

Es kann sein, dass man für das **Background-Tracking** der App die Berechtigung für den Standort im Hintergrund geben muss also `Immer zulassen`.

ERM/RM

Das ERM/RM Diagramm wurde kaum verändert und wurde eigentlich wie geplant umgesetzt. Es ist im Ordner `ERM_RM` zu finden.

Klassendiagramm

Beim Klassendiagramm hat es schon mehr Änderungen gegeben, da wir erkannt haben, dass wir nicht wirklich eine User Klasse brauchen und haben dann alle Funktionen, welche ein User benötigt in den `UserManager` ausgelagert. Bei den Schwierigkeiten sind zwei neue dazugekommen, aber ansonsten ist es ziemlich gleich geblieben. Das Klassendiagramm ist im Ordner `Klassendiagramm` zu finden.