# IPA-Detailbeschreibung

## Ausgangslage

Adobe zieht Anfang 2023 in ein neues Gebäude in Basel um. Am neuen Standort stehen den Mitarbeitenden eine begrenzte Anzahl Parkplätze zur Verfügung. Die Parkplätze müssen vor dem Gebrauch halbtagesweise reserviert werden. Die Reservation soll durch eine Web-Applikation erfolgen. Während dem Reservationsprozess sollen Nutzer/innen ein Datum, eine Uhrzeit und ein gespeichertes Fahrzeug wählen können (um jeder Reservation ein Kontrollschild zuweisen zu können).

Einige der Parkplätze verfügen zudem über eine Ladestation für Elektrofahrzeuge. Solange nicht alle Parkplätze ohne Ladestation belegt sind, sollte das System Fahrzeugen mit Verbrennungsmotor keine Reservationen von Parkplätzen mit Ladestation erlauben.

Nebst dem eigentlichen Reservationsprozess bietet das System auch ein Übersichts-Dashboard. In dieser Ansicht können Nutzer/innen sehen, wann sie Parkplätze reserviert haben, und existierende Reservationen wieder stornieren.

Zudem bietet das System für die Administratorin einen Überblick über den Status aller Parkplätze und Nutzer/innen. Im Admin-View gibt es auch weitere Aktionen, etwa kann ein Parkplatz für ein Datum als «disabled» markiert werden, wenn beispielsweise Bauarbeiten stattfinden.

Letztlich werden im Parkhaus über jedem Parkplatz Displays installiert, welche den Status des jeweiligen Parkplatzes im Reservationssystem anzeigt.

## Detaillierte Aufgabenstellung

In dieser Arbeit wird die Benutzeroberfläche des Parkplatzreservationssystems umgesetzt. Die Benutzeroberfläche wird in Javascript und dem React Framework implementiert. Die React-Applikation kommuniziert via HTTP API mit einem Backend. Da die Benutzeroberfläche relativ umfangreich ist, wird die Implementierung auf die wichtigsten Features begrenzt. Die Features, die umgesetzt werden müssen, sind in den folgenden Abschnitten beschrieben. Das Backend, die Datenbank und die Displays über dem Parkplatz sind nicht Teil dieser Arbeit.

Eine OpenAPI Spezifikation ist vorhanden, welche die API des Backends beschreibt. Für die Entwicklung der Benutzeroberfläche steht eine Mock-API, welche die OpenAPI Spezifikation umsetzt, zur Verfügung. Diese ist hier einsehbar: https://github.com/berufsbildung-basel/parkit-spec

Ebenfalls besteht bereits ein Adobe Experience Designer (XD) Mockup des Designs und Layouts der Benutzeroberfläche (<https://xd.adobe.com/view/c98247cd-32b5-4205-8ec2-65633f00ef90-9cd1/> - um den File im Browser anzusehen, ist ein Login mit einem Adobe-Konto erforderlich. Ein solches kann kostenlos erstellt werden, und um den File anzusehen werden keine speziellen Rechte benötigt. Alternativ ist der selbe File auch im Anhang auf PkOrg, und kann mit der Adobe XD Destop-Applikation geöffnet werden).

Die Requirements sind wie folgt:

Login:

Als erstes wird ein Login-Screen angezeigt, wenn ein/e neu/e Nutzer/in auf die Applikation zugreift. Die User-Authentication wird später mit Adobe's SSO integriert. SSO-Integration ist aber nicht im Umfang dieser Arbeit enthalten. Daher darf das Login mocked werden. Die Credentials können client-seitig überprüft werden, ob sie mit einem Test-User übereinstimmen, zum Beispiel Username [test@adobe.com](mailto:test@adobe.com) und Password testPassword. Wenn die Credentials nicht stimmen, soll eine entsprechende Fehlermeldung angezeigt werden. Wenn die Credentials stimmen, können sie client-seitig gespeichert werden, zB im Browser LocalStorage.

* Wenn die client-seitige Validierung erfolgreich war, muss der Username und das Passwort gespeichert werden (zum Beispiel im Browser Local Storage). Bei allen API-Requests muss der Username und das Passwort als http Basic Auth Header mitgeschickt werden.

Übersicht:

Wenn der User eingeloggt ist, wird eine Übersicht angezeigt. Diese Ansicht entspricht dem Adobe XD mockup. Auf der linken Seite wird eine Navigationsleiste angezeigt. Hier sind die Optionen «Dashboard», «Reservations», «Parking Overview» und «Vehicles» verfügbar

Auf der Page «Dashboard» wird folgendes angezeigt:

* Eine Übersichtstabelle aller zukünftigen Reservationen des Nutzers/der Nutzerin, in der jede Zeile Datum, Uhrzeit, Fahrzeugname und Kontrollschild einer Reservation zeigt
* Ein UI-Element zeigt die aktuelle Anzahl von verfügbaren Parkplätzen an
* Ein UI-Element zeigt die aktuelle Anzahl von kommenden Reservationen an
* Diese Informationen werden vom Endpunkt listReservations entnommen. Dieser Endpunkt liefert alle Reservationen, auch vergangene. Die Resultate müssen daher gefiltert werden.

Auf der Page «Reservations» wird folgendes angezeigt:

* Eine Tabelle zeigt alle Reservationen des Nutzers/der Nutzerin an. Es wird Datum, Uhrzeit, Fahrzeugname und Kontrollschild angezeigt.
* Diese Informationen kommen vom selben listReservations Endpunkt. Eine Spalte der Tabelle soll den Status der Reservation (ob vergangen, kommend oder storniert) anzeigen.
* Ein Button lässt den Nutzer die Tabelle nach Status filtern. In dem Fall werden nur vergangene, kommende oder stornierte Reservationen respektive angezeigt.
* Ein Button «Create Reservation» führt zur «Parking Overview» page.
* Stornierung: Für jede Reservation sollte ein Button zur Stornierung existieren. Dieser soll den API-Endpunkt cancelReservation aufrufen.

Auf der Page «Parking Overview» wird folgendes angezeigt:

* Eine grafische Übersicht aller Parkplätze, und deren Status (frei, besetzt, vom Administrator als disabled markiert, Elektrofahrzeug-Ladestation vorhanden oder nicht). Das Datum für die Status-Anzeige kann durch ein Date-Picker geändert werden (max. 2 Wochen in die Zukunft). Diese Informationen stammen von den Endpunkten /parking-spots/availability und /parking-sports
* Durch einen Klick auf ein freies Parkfeld öffnet sich rechts ein Panel zur Erstellung einer Reservation. Hier kann der/die Nutzer/in die Dauer der Reservation wählen (Vormittag, Nachmittag, ganzer Tag), sowie das entsprechende Fahrzeug (Endpunkt /vehicles)
* Wenn der/die Nutzer/in bereits vergangene Reservationen hat, sollte das Interface das Panel zur Reservation für das zuletzt verwendete Parkfeld automatisch öffnen.

Der Link zu «Vehicles» öffnet rechts ein Panel, wie bei «Profile».

* Hier werden alle Fahrzeuge, die dem Nutzer/der Nutzerin gehören, angezeigt. Dazu wird der Endpunkt /vehicles aufgerufen. Die Fahrzeuge haben einen Namen und ein Kontrollschild.
* Durch einen Button neben jedem Fahrzeug kann das Fahrzeug gelöscht werden. Dazu wird die Operation /removeVehicle aufgerufen.
* Durch einen Button kann ein weiteres Fahrzeug hinzugefügt werden. Dazu wird die Operation createVehicle aufgerufen.

Oben rechts ist ein Profile-Image des Users/der Userin. Dieses Bild darf für die Arbeit ein Placeholder sein (hard-coded). Durch den Klick auf dieses Bild wird folgendes angezeigt:

* Es öffnet sich rechts ein Panel, der Informationen zum aktuellen Nutzer anzeigt: E-Mail-Adresse, Name und Benutzername. Dies wird dem Endpunkt /users/{id} entnommen. Als User-ID Parameter für den API-Request darf in der IPA der Benutzername verwendet werden, da diese ID später aus dem SSO-System stammt.

### Testing

Wichtig ist, dass unter unvorhergesehenen Applikationszuständen eine gute User Experience gewährleistet ist. Der/die Nutzer/in sinnvoll weitergeführt werden, wenn die API eine Fehlermeldung oder keine Daten zurückgibt. Die möglichen Fehlermeldungen der API sind im OpenAPI File dokumentiert. Davon sollten folgende Fälle abgedeckt werden:

* Bei allen Requests soll der/die Nutzer/in zuück zum Login-Screen geführt werden, wenn eine 401 Response erhalten wird.
* Wenn keine kommenden Reservationen im System sind, soll auf der Übersichtsseite nicht eine leere Tabelle angezeigt werden, sondern eine entsprechende Mitteilung, dass keine Reservationen gefunden wurden.
* Beim Endpunkt «createReservation» muss der Status-Code 409 – Conflict abgedeckt sein, und in der Benutzeroberfläche eine entsprechende Fehlermeldung angezeigt werden.
* Zwei zusätzliche Error-Responses der API, welche während der Entwicklung als sinnvoll empfunden werden, sollten abgedeckt sein.

Um diese Fehler in der Mock-API zu testen, können im Request Header verwendet werden, um eine gewisse Antwort der API zu erzwingen. Die Boilerplate-Applikation und das Mock-Backend (beides Teil der Vorarbeit) sind so eingerichtet, dass alle variablen im Browser Local Storage, welche mit «x-test-» starten, als Header ans Mock-Backend geschickt werden. Das Mock-Backend versteht bereits folgende Header, aber kann bei Bedarf für die Arbeit oder als Teil der Vorarbeit auch noch ausgebaut werden (je nach ausgesuchten Testfällen):

* x-test-response-status: Das Backend antwortet mit diesem Response-Code
* x-test-response-text: Das Backend antwortet mit diesem Text als Status-Text
* x-test-empty-response: Der Endpunkt «listReservations» antwortet mit einem leeren Array als Body

### Dokumentation

Als technische Dokumentation sollte das Github-Repository einen Readme-File in Markdown Format enthalten. Dieses Readme dient dazu, dass andere Entwickler nach der IPA den Code verstehen sowie daran arbeiten können. Der Readme File sollte kurz den Zweck der Applikation beschreiben, eine grobe Übersicht über die Architektur aufzeigen und eine Anleitung zur Installation beinhalten.

Methoden und Funktionen, welche Business-Logik umsetzen, sollten mit JSDOC die Inputs und Outputs dokumentieren. Davon ausgenommen sind React-Komponenten.

Die technische Dokumentation (Kommentare, JSDoc, Readme), sowie aller Code, wird auf Englisch geschrieben.

Als Teil der schriftlichen Arbeit müssen Entscheidungen, die während der Implementation getroffen werden, dokumentiert und begründet werden.

## Out of Scope

Folgende Punkte werden bei der IPA bewusst nicht bearbeitet:

* Build System: Die Kandidatin soll verstehen, wieso ein Build-System benötigt wird, und soll die Commands, die für die Arbeit benötigt werden, selbstständig verwenden können. Allerdings ist das Build-System für die Arbeit vorgegeben, und sie muss selbst keine Konfigurationsänderungen vornehmen können.
* Das Backend, die Datenbank und die Displays über dem Parkplatz sind nicht Teil dieser Arbeit.
* Das Admin-Dashboard ist nicht Teil der Arbeit
* Integration mit Adobe’s SSO Login System ist nicht Teil der Arbeit, daher darf der Authentifizierungsprozess client-seitig mocked werden.
* Responsive Design: Es wird nur die Desktop-Version implementiert
* Deployment: Die Applikation wird nur lokal entwickelt. Sie muss nicht auf einen Web-Server deployed werden oder als Produktions-Build gebaut werden.

## Mittel und Methoden

* MacOs
* Node und NPM-Umgebung
* Chrome Browser
* VSCode IDE
* Die Benutzeroberfläche wird in React und Javascript implementiert
* Die Grafischen Elemente im User Interface wird mit dem React MUI Framework implementiert

Vorkenntnisse

* Die Kandidatin hat bereits mit React gearbeitet, sowohl an Projektarbeiten sowie auch als Teil ihres Teams
* Sie hat Erfahrung mit der MUI Component-Library, welche sie für die Arbeit verwenden wird.
* Sie hat Erfahrung in der Frontend-Entwicklung und der Benützung von HTTP RESTful APIs.

Neue Lerninhalte

* Die Kandidatin hat zuvor noch nie mit einer Mock-API gearbeitet.
* Sie hat noch nie OpenAPI-Spezifikationen verwendet.
* Sie hat wenig Erfahrung mit Testing
* Sie hat bisher mit React Class Components gearbeitet. Die React Hooks API ist ihr neu. Obwohl nicht vorgeschrieben ist, dass sie diese API verwendet, ist der Kontakt dazu möglicherweise nicht vermeidbar, zum Beispiel bei der Verwendung von gewissen Libraries.

## Vorarbeiten

* Es besteht ein Git-Repository welches den React-Boilerplate (Build-System, Linter) enthält. Ebenfalls besteht die Mock-API im selben Git-Repository.
* In der Mock-API wurden bereits für die benötigten Endpunkte Handler-Funkionen geschrieben
* Es wurden UI-Mocks in Adobe XD bereits erstellt, als Vorlage für die Benutzeroberfläche
* Ein Dependency-Injection Framework (mit React-Context API) ist bereits im React-Code enthalten, um Dependencies wie den API-Client in der Applikation global verfügbar zu machen.