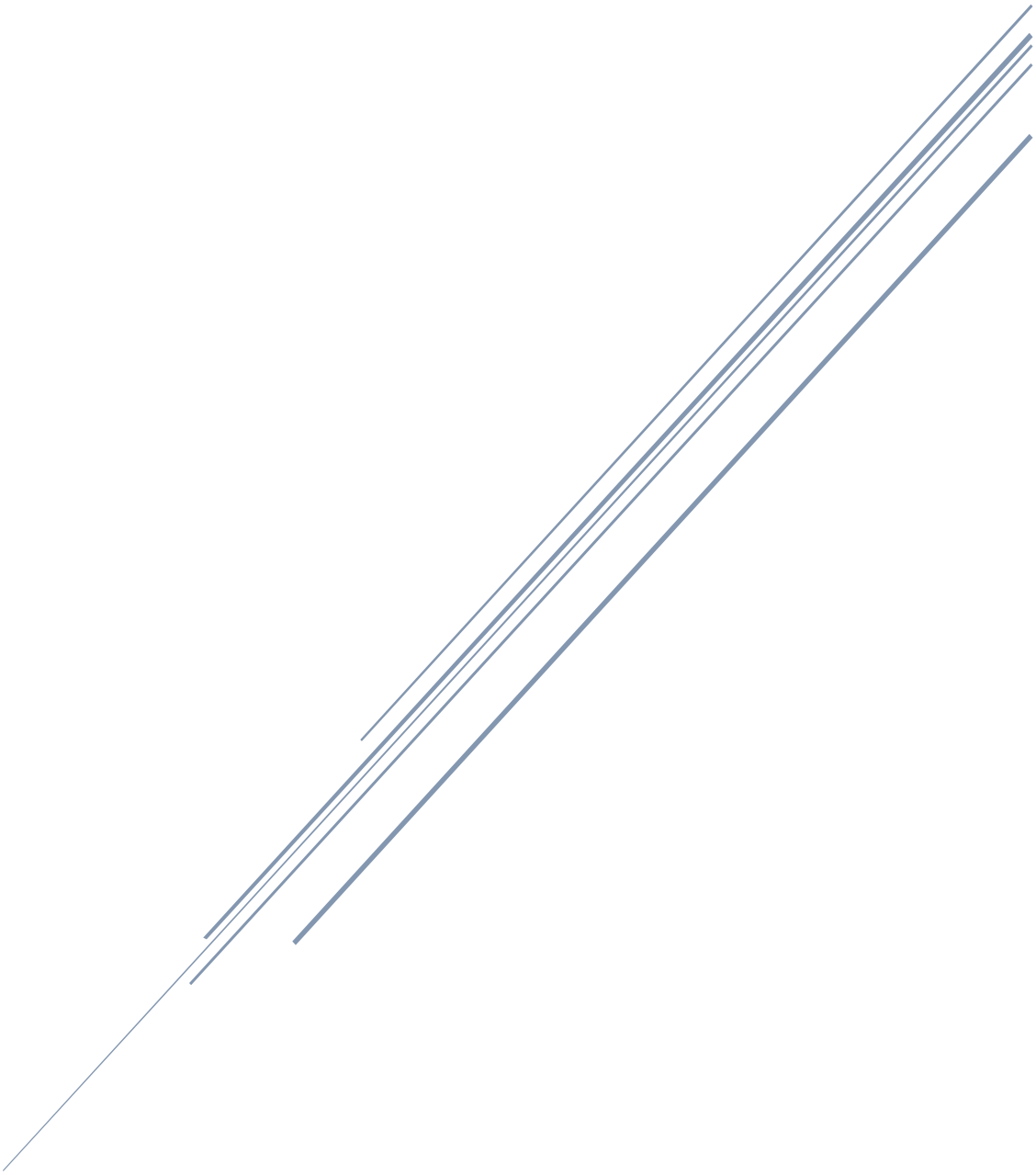
SKI-SERVICE .NET MAUI APP

Projektdokumentation



Fokko Vos & Robin Ruf
IBZ 2023

Inhaltsverzeichnis

1 Versionsverlauf

Version	Datum	Autor	Bemerkung
1.0	28.09.2023	Fokko Vos & Robin Ruf	Initial Version
1.1	03.12.2023	Fokko Vos & Robin Ruf	Vollenden der Planung
1.2	10.12.2023	Fokko Vos & Robin Ruf	Vollenden der Entscheidung
1.3	30.12.2023	Robin Ruf	Initiale Dokumentation der Realisierung
1.4	05.01.2023	Fokko Vos	Fortsetzen des Realisierungs-Teils der Dokumentation
1.5	06.01.2023	Fokko Vos & Robin Ruf	Einpflegen der Anhänge & Auswertung, Verfassung der Letzen abschnitte.

1 - Versionsverlauf

2 Einleitung

Willkommen zur Dokumentation des Projekts „Ski-Service.NET App“ der Firma Jetstream-Service. In einer Welt, in der digitale Technologien zunehmend an Bedeutung gewinnen, ist die Anpassung und Optimierung von Geschäftsprozessen durch innovative Lösungen ein Schlüsselfaktor für den Unternehmenserfolg. Die Ski-Service.NET App ist eine Antwort auf die Notwendigkeit, den Workflow und die Datenverwaltung im Bereich der Ski-Service-Aufträge effizienter zu gestalten.

Diese Dokumentation bietet einen umfassenden Einblick in das Projekt, das darauf abzielt, eine benutzerfreundliche und für unterschiedliche Endgeräte optimierte Anwendung zu entwickeln. Unser Ziel ist es, eine intuitive Benutzeroberfläche zu schaffen, die den spezifischen Anforderungen und Arbeitsbedingungen einer Skiservice-Werkstatt gerecht wird.

In dieser Dokumentation werden alle Phasen des Projekts abgedeckt, von der Konzeption bis zur technischen Umsetzung, einschließlich der Überlegungen zu den verwendeten Technologien und der Integration bestehender Systeme. Sie dient als Leitfaden und Informationsquelle für alle Projektbeteiligten und Stakeholder und gibt einen klaren Überblick über die Ziele, Strategien und erwarteten Ergebnisse des Projekts.

3 Informieren

3.1 Ausgangssituation

Unternehmen
Jetstream-Service, ein Unternehmen, das sich auf Skiservicearbeiten spezialisiert hat.

Aktueller Stand
Das Unternehmen hat neue Touchscreen-fähige Hardware angeschafft (Tablets, Surface) und stellt diese den Mitarbeitern für die Datenpflege der Ski-Service-Aufträge zur Verfügung.

Ziel
Die einfache, intuitive und aufgabenangemessene Bedienung der Benutzeroberfläche, unter Berücksichtigung der Arbeit mit Handschuhen.

Integration
Erweiterung der existierenden Basis zur Online-Anmeldung und der Service-App mit der neuen GUI-Lösung für Tablets und Handys.

3.2 Anforderungsanalyse

3.2.1 Funktionale Anforderungen

- Sicheres Login-System für Mitarbeiter mit Benutzername und Passwort.
- Verwaltung von Serviceaufträgen
    - Anzeige aller anstehenden Ski-Serviceaufträge in einer Liste.
    - Möglichkeit zur Filterung und Suche innerhalb der Serviceauftragsliste.
    - Fähigkeit zur Mutation (Statusänderung, Notizen hinzufügen, Löschen) bestehender Serviceaufträge.
    - Unterschiedliche Statusoptionen für Aufträge: Offen, In-Arbeit, Abgeschlossen.
    - Zuordnung und Verwaltung von Prioritäten bei Serviceaufträgen.
    - Ergänzung von Kundeninformationen (Name, E-Mail, Telefon) bei Bedarf.
    - Eindeutige Zuordnung einer Dienstleistung pro Serviceauftrag aus einer vorgegebenen Liste (z.B. Kleiner Service, Großer Service, Rennski-Service).

3.2.2 Nicht-Funktionale Anforderungen

- Starke Authentifizierung und Autorisierung, um den Zugang zu sensiblen Daten zu sichern.
- Schnelle Verarbeitung von Anfragen, um eine effiziente Abwicklung der Serviceaufträge zu gewährleisten.
- Einfache, intuitive und aufgabenangemessene Benutzeroberfläche, die auch mit Handschuhen bedienbar ist.
- Stabile und fehlerresistente Anwendungsleistung, insbesondere während der Hochsaison.

3.3 Technische Anforderungen

UX und UI Design
Erstellung von Mockups, die eine einfache und aufgabenangemessene Bedienung ermöglichen, unter besonderer Berücksichtigung der Bedienbarkeit mit Handschuhen.

Integration und Kompatibilität
Sicherstellung, dass das neue User Interface reibungslos mit dem bereits existierenden Backend-System integriert wird. Dies beinhaltet die Anbindung an die bestehenden Web-APIs und Datenstrukturen. Überprüfung und ggf. Anpassung der Schnittstellen, um eine effiziente Kommunikation zwischen dem neuen Frontend und dem existierenden Backend zu gewährleisten.

Test und Qualitätssicherung
Durchführung umfassender Tests, um die Funktionalität und Leistung der neuen UI-Komponenten sowie deren Integration mit dem Backend zu überprüfen.

4 Planen

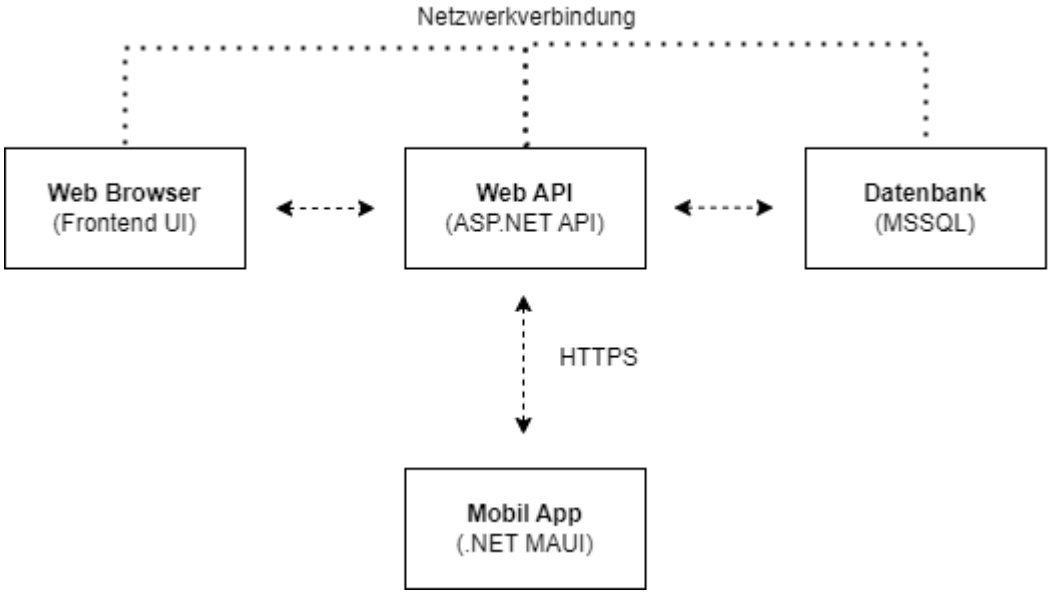
4.1 Zeitplan

Der Aufwand in Stunden bezieht sich auf die von allen Projektmitgliedern gemeinsam aufgewendete Zeit.

Nr.	Beschreibung	SOLL-Zeit (h)	IST-Zeit (h)
1	**Informieren**	**6**	**5**
1.1	Situationsanalyse	2	2
1.2	Erfassung der Nutzerbedürfnisse	3	2
1.3	Erfassung der Schulischen anforderungen	1	1
2	**Planen**	**12.5**	**14.5**
2.1	Erstellung eines Zeitplans	1.5	1.5
2.2	Entwurf der Systemarchitektur	1	1
2.3	Mockups für die Benutzeroberfläche	10	12
3	**Entscheiden**	**2.5**	**3**
3.1	Technologienwahl	0.5	1
3.2	Test-Strategie	2	2
4	**Realisieren**	**24.5**	**35.5**
4.1	Git-Repository Einrichten	0.5	0.5
4.2	Verbindung zum Backend aufbauen	1	5
4.3	Entwicklung der Benutzeroberfläche	15	20
4.4	Entwicklung der nötigen Tests	8	10
5	**Kontrollieren**	**3**	**2**
5.1	Test-Strategie ausführen	1	0.5
5.2	Anforderungen mit dem Produkt abgleichen	2	1.5
6	**Auswerten**	**6**	**8**
6.1	Finalisierung der Dokumentation	2	5
6.2	Lessons-Learned Identifizieren	2	2
6.3	Präsentation vorbereiten	2	1
Gesamt		**54.5**	**68**

2 - Zeitplan

4.2 Systemarchitekturentwurf



1 - Systemarchitekturentwurf

Das System, das wir für die "Ski-Service.NET App" entwerfen, ist eine Weiterentwicklung der bestehenden Drei-Schichten-Architektur und integriert neue Komponenten für erweiterte Funktionalität und Benutzerfreundlichkeit. Es folgt eine Beschreibung der aktualisierten Systemarchitektur basierend auf dem visuellen Entwurf.

Web Browser (Frontend UI)
Die Präsentationsschicht, zugänglich über einen Web Browser, bleibt die primäre Schnittstelle für Benutzerinteraktionen

Web API (ASP.NET API)
Die Geschäftslogikschicht wird durch eine ASP.NET API repräsentiert, die das Rückgrat unserer Anwendung bildet. Sie verarbeitet Anfragen, führt Geschäftsregeln aus und handhabt die Datenkommunikation mit der Datenbank.

Datenbank (MSSQL)
Die Datenpersistenzschicht basiert weiterhin auf Microsoft SQL Server, welcher die notwendigen Daten speichert und bereitstellt. Diese Schicht ist für hohe Leistung und Transaktionsintegrität optimiert und ermöglicht schnelle Datenabfragen.

Mobile App (.NET MAUI)
Als neue Komponente führen wir eine mobile Anwendung ein, die mit .NET MAUI entwickelt wird. Diese App erweitert die Zugänglichkeit der Systemfunktionen und bietet eine optimierte Benutzererfahrung für mobile Geräte.

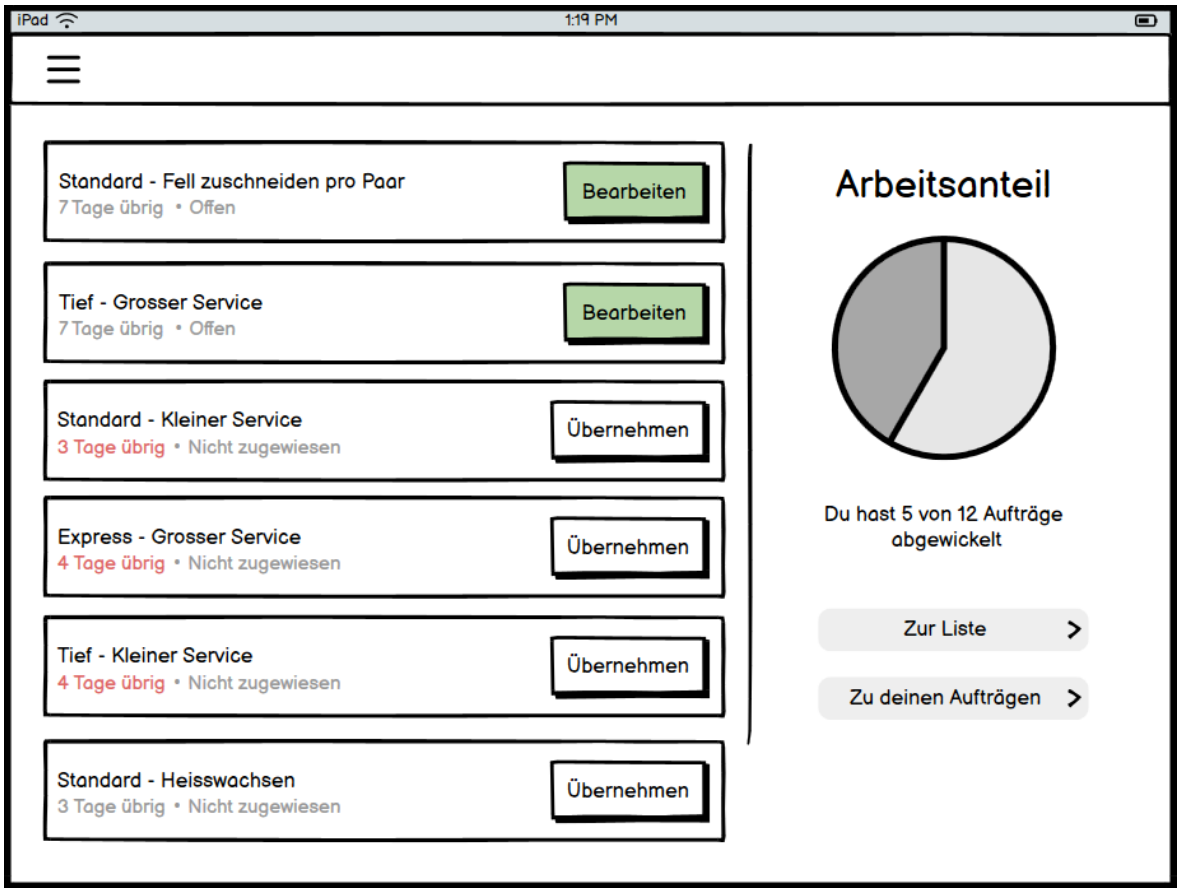
Die Netzwerkverbindung zwischen allen Schichten gewährleistet sichere und effiziente Datenübertragungen über HTTPS, wobei JSON als Datenaustauschformat zum Einsatz kommt. Dieses Format unterstützt eine hohe Interoperabilität und erleichtert das Parsen der Daten.

Diese erweiterte Architektur erhält die bewährten Prinzipien der klaren Schichtentrennung und fügt gleichzeitig neue Elemente hinzu, um die Anwendung modern, flexibel und zukunftssicher zu machen. Sie berücksichtigt sowohl die aktuellen Anforderungen als auch die Skalierbarkeit für zukünftige Erweiterungen und ist somit ein solides Fundament für die fortlaufende Entwicklung unserer Projektziele.

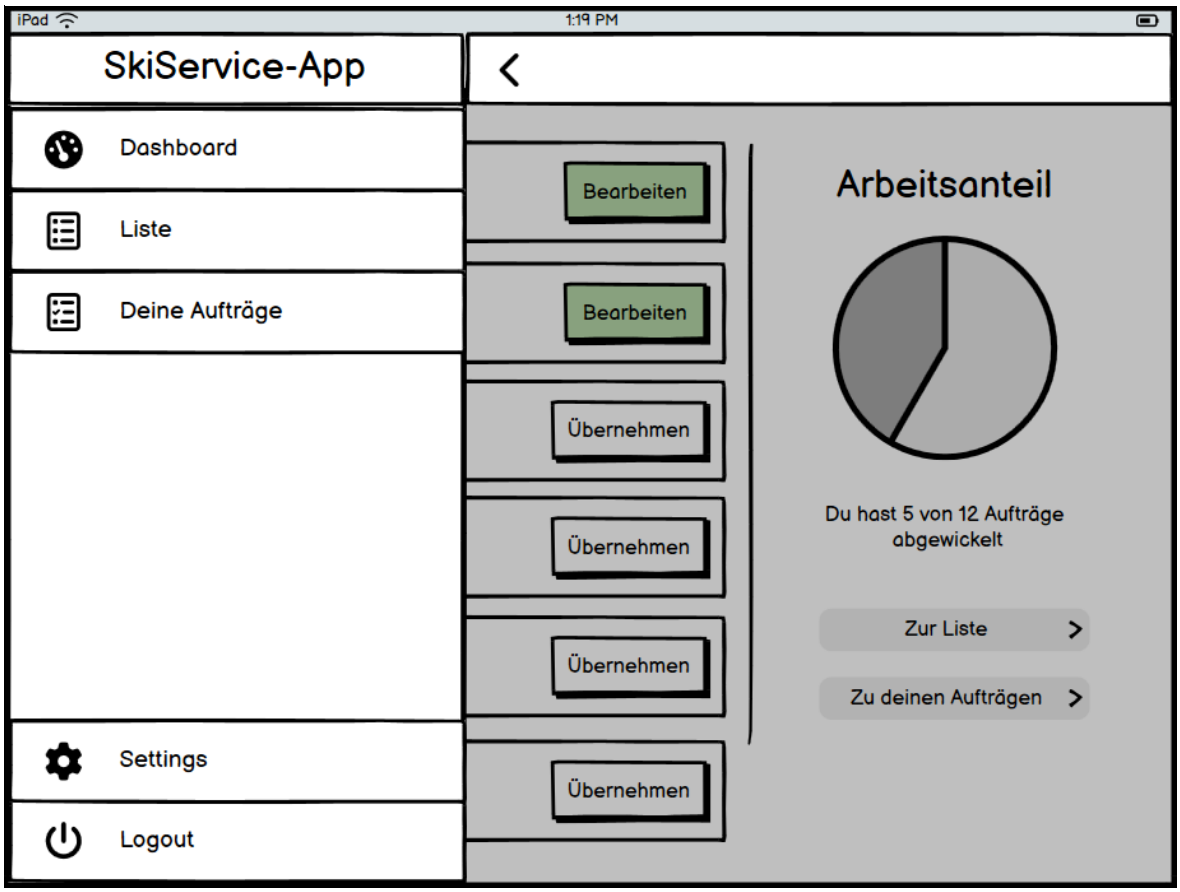
4.3 Mockups

Die Texte welche in den Mockups verwendet werden, dienen nur zur Orientierung und stehen noch nicht endgültig fest.

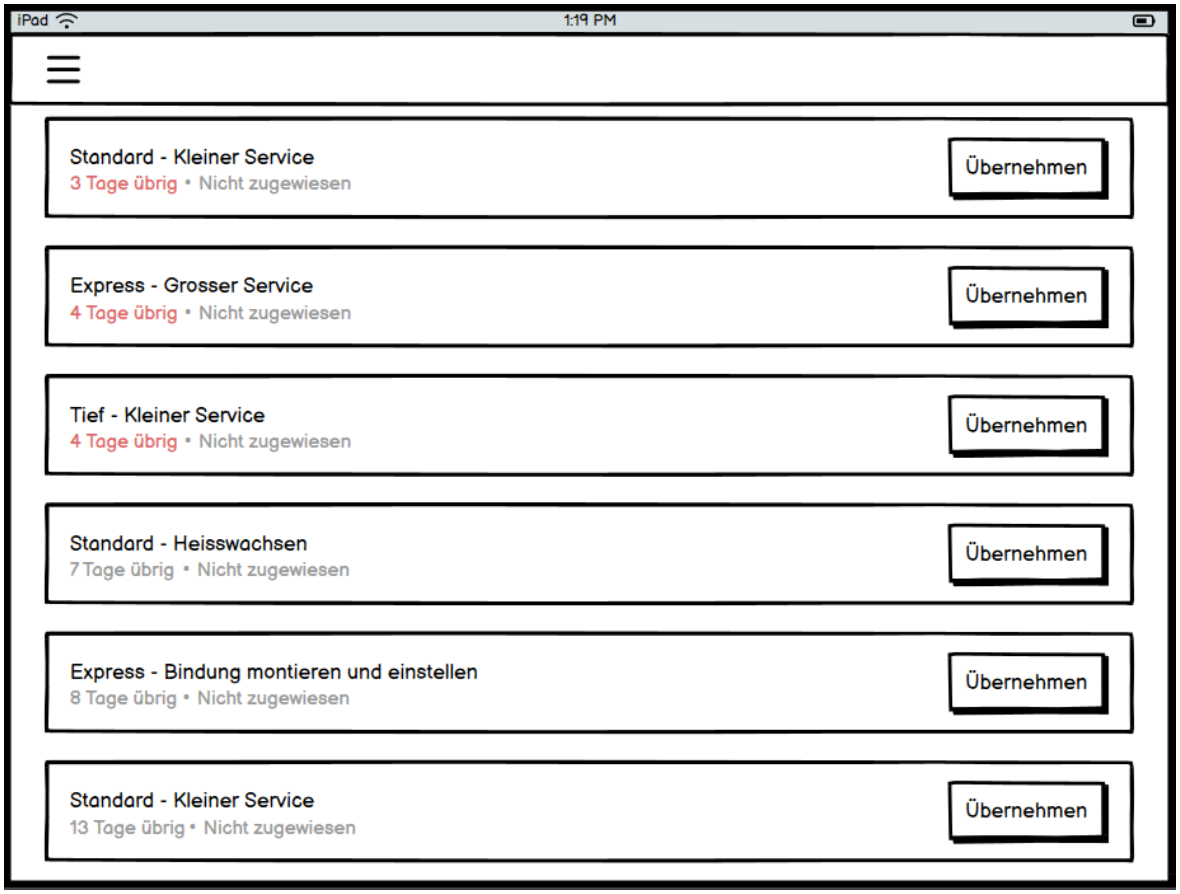
2 - Mockup - Anmeldung



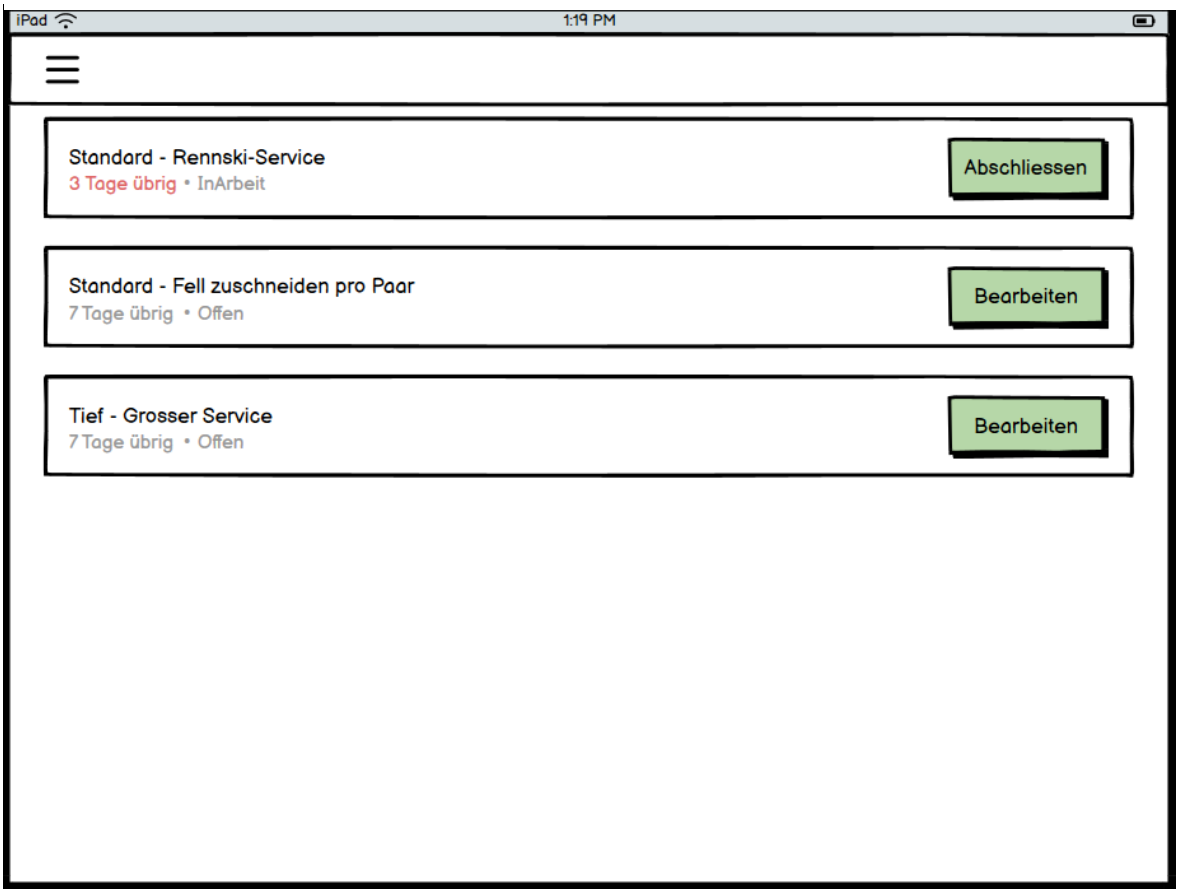
3 - Mockup - Dashboard



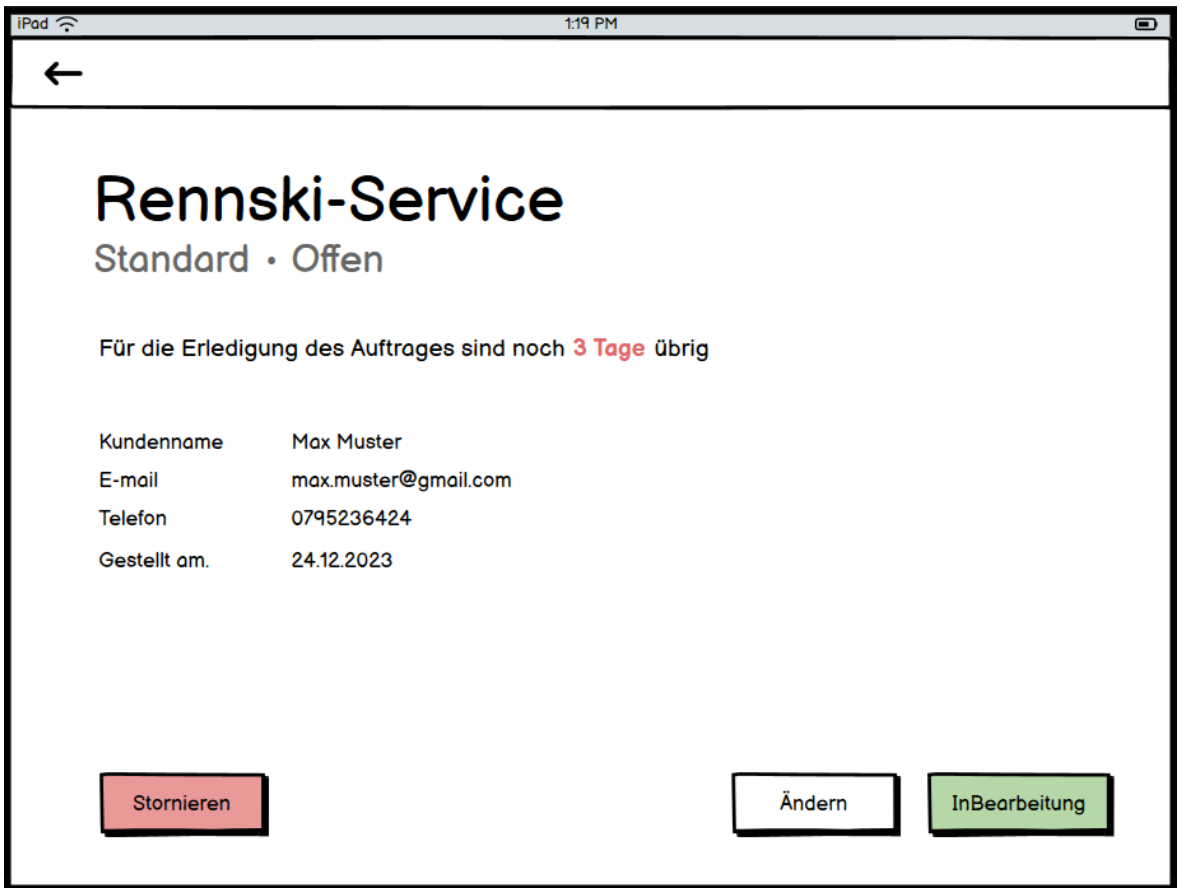
4 - Mockup - Menu Open



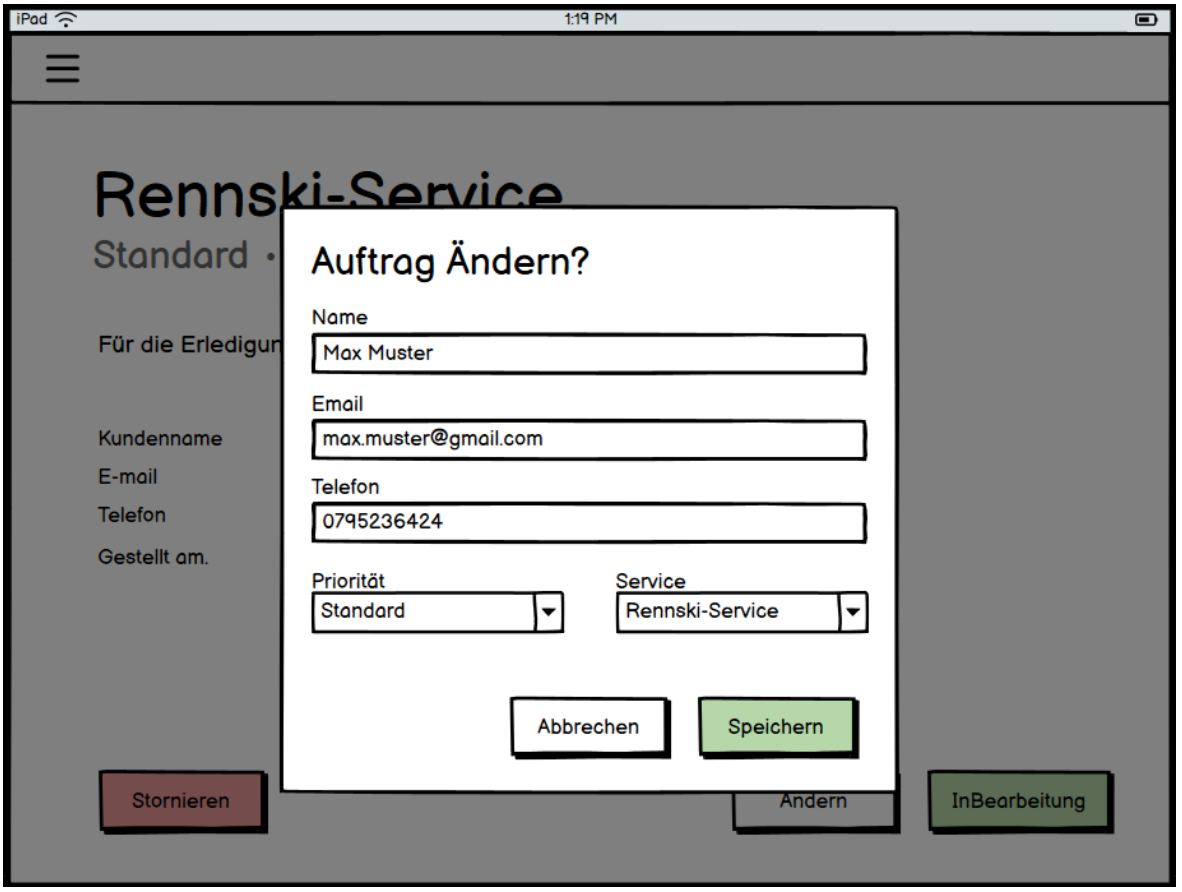
5 - Mockup - Liste



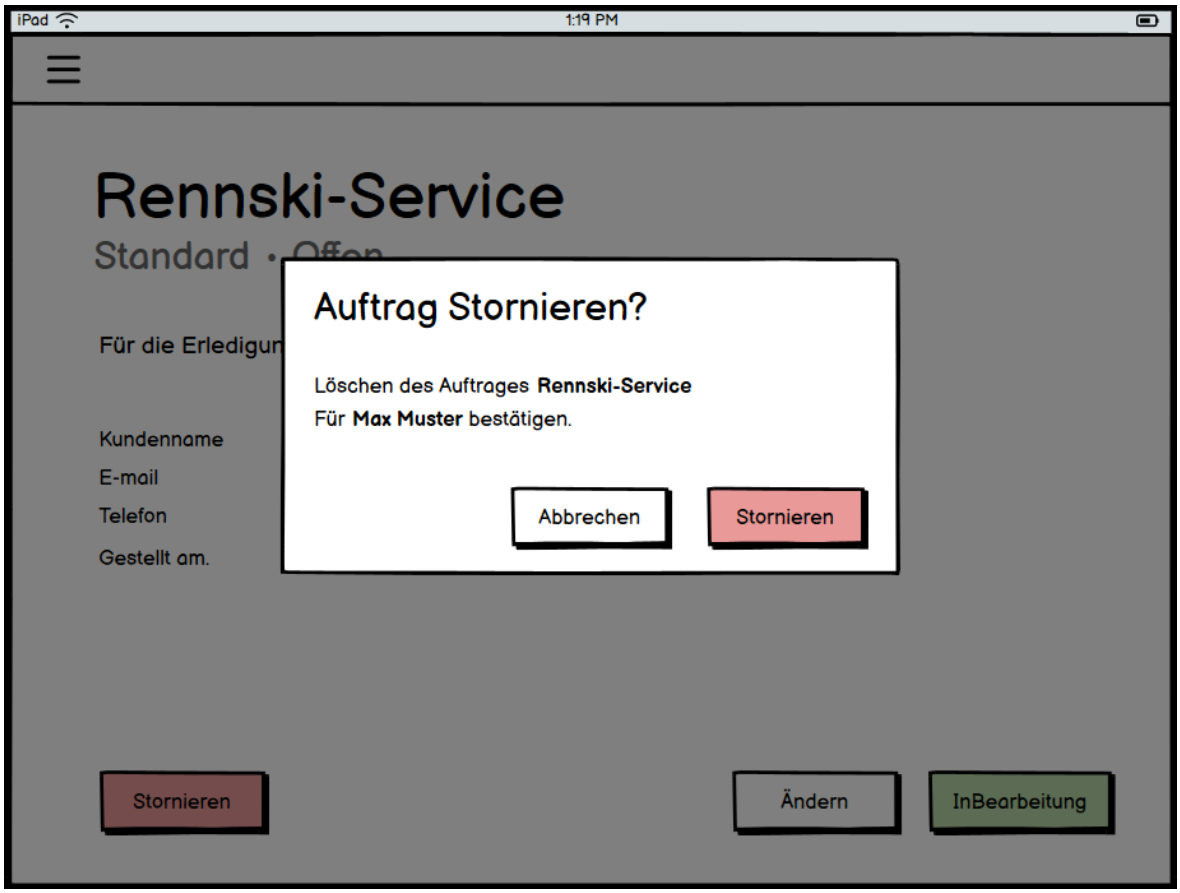
6 - Mockup - Nutzer liste



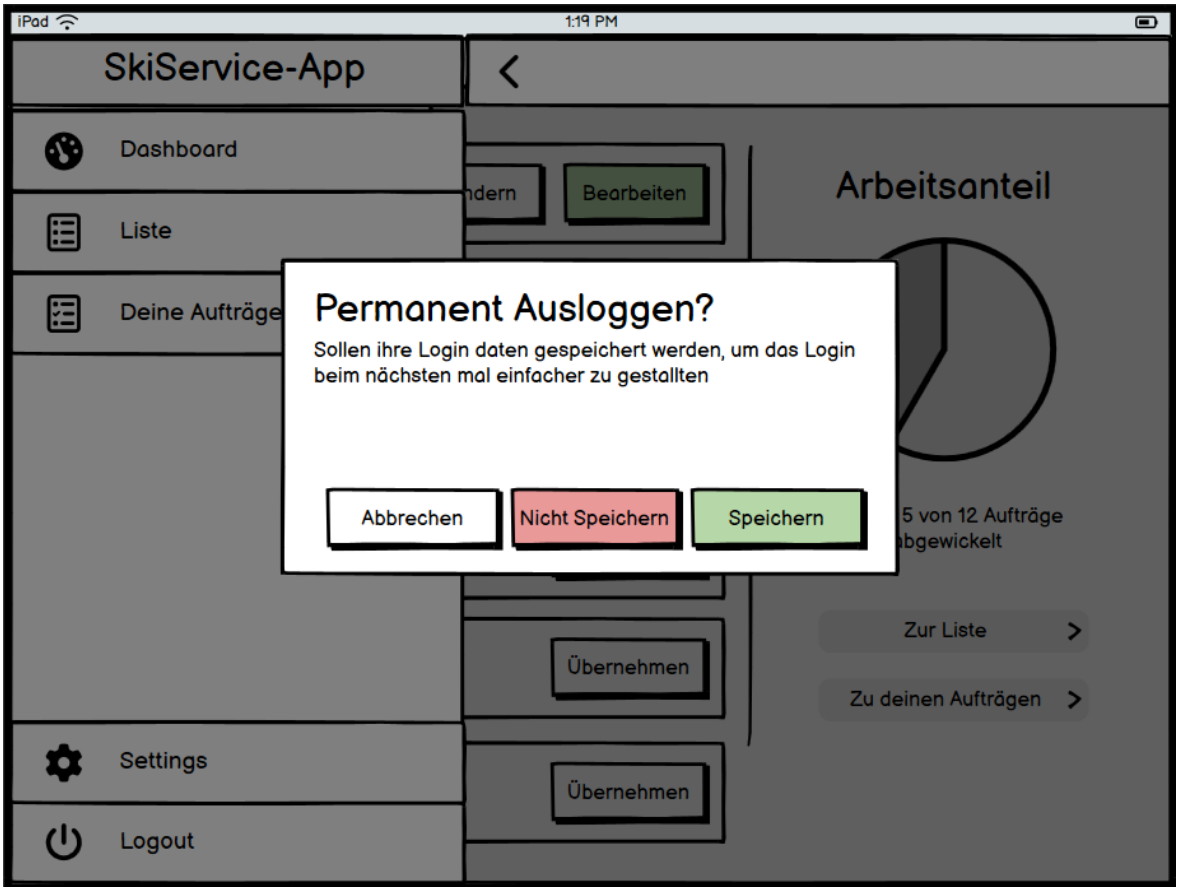
7 - Mockup - Auftrag Informationen



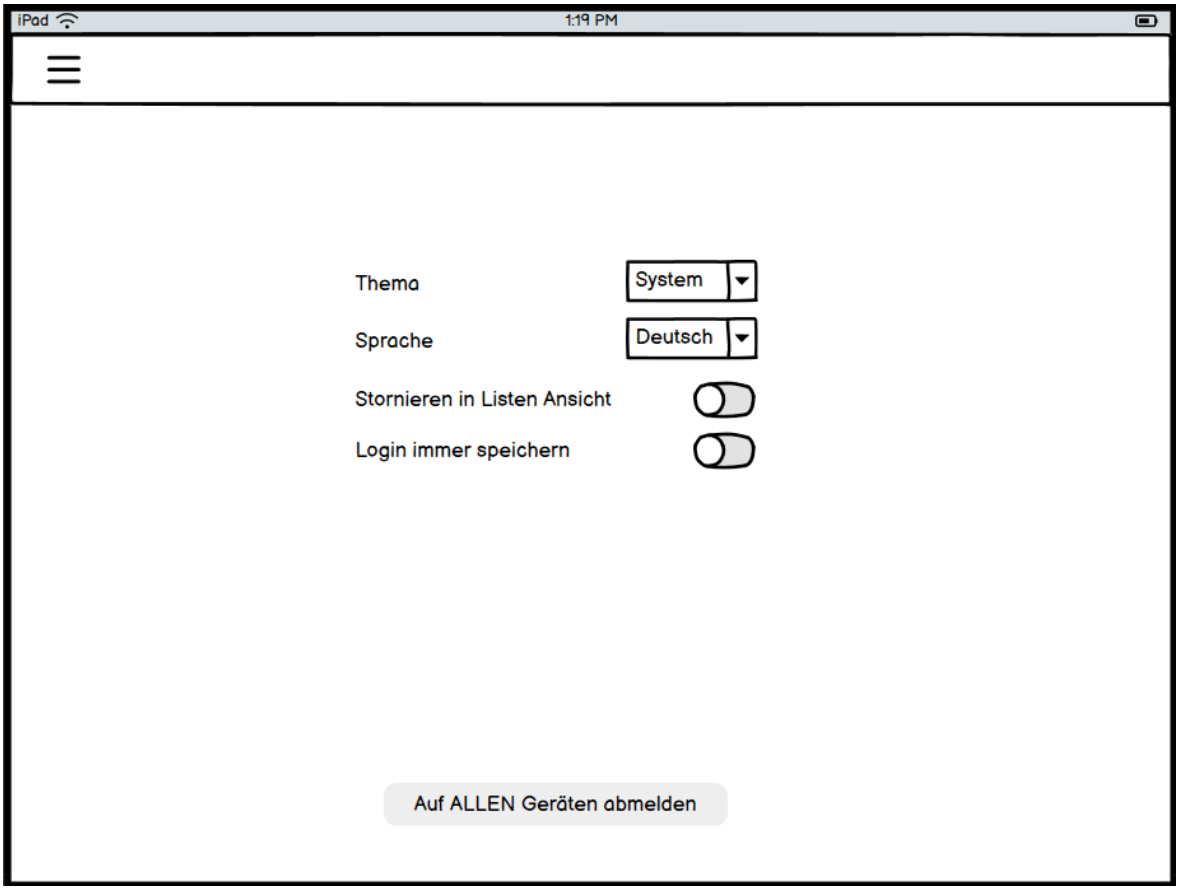
8 - Mockup - Auftrag bearbeiten



9 - Mockup - Stornieren Dialog



10 - Mockup – Logout Dialog



11 - Mockup - Einstellungen

5 Entscheiden

5.1 Technologie und Testgeräte

Die technologische Basis für das Projekt "Ski-Service.NET App" bildet die Entscheidung für die Nutzung von .NET MAUI und .NET 8, wodurch wir die Vorteile der neuesten Entwicklungen in der Softwaretechnologie voll ausschöpfen. .NET MAUI ermöglicht uns eine plattformübergreifende Entwicklung, während .NET 8 uns Zugriff auf die neuesten Sprachfeatures, Sicherheitsupdates und Performance-Optimierungen bietet.

Für die Entwicklungsumgebung haben wir uns auf Visual Studio festgelegt. Diese IDE stellt uns alle notwendigen Werkzeuge und Emulatoren zur Verfügung, um eine effiziente Entwicklung und ein akkurates Testing zu ermöglichen. Die Emulatoren in Visual Studio erlauben es uns, ein breites Spektrum an Geräten zu simulieren, um so die Kompatibilität und Benutzerfreundlichkeit unserer App sicherzustellen.

Speziell für die Testzwecke haben wir uns entschieden, das "Tablet 420 DPI 8in" als Standard-Testgerät zu verwenden. Dieses Gerät, mit einem Arbeitsspeicher von 4 GB und einer Auflösung von 2200 x 2480 bei 420 DPI, gewährleistet, dass wir die App unter realistischen Bedingungen entwickeln und testen können. Diese Spezifikationen stellen sicher, dass die App performant läuft und die Benutzererfahrung optimiert ist, insbesondere in Bezug auf die Darstellung und Handhabung der Benutzeroberfläche auf Geräten mit hoher Pixel- und Punktdichte (DPI).

5.2 Test-Strategie

Für die "Ski-Service.NET App" wird eine mehrschichtige Teststrategie verfolgt, die sicherstellt, dass sowohl die einzelnen Komponenten als auch die Anwendung als Ganzes die festgelegten Anforderungen und Qualitätsstandards erfüllen. Die Entscheidung für unsere Teststrategie beinhaltet folgende Kernpunkte

Unit Tests
Werden verwendet, um die grundlegenden Bausteine der Anwendung zu prüfen.

Integrationstests
Stellen das korrekte Zusammenspiel der einzelnen Module und Dienste sicher.

Zusätzlich zu diesen automatisierten Testverfahren werden manuelle **Usability Tests** während der gesamten Entwicklung durchgeführt. Diese Tests sind entscheidend, um ein benutzerzentriertes Design zu gewährleisten und um sicherzustellen, dass die Anwendung nicht nur funktioniert, sondern auch effizient und angenehm zu bedienen ist. Die Usability Tests werden in verschiedenen Phasen der Entwicklung durchgeführt, um kontinuierliches Feedback in den Entwicklungsprozess einfließen zu lassen und eine hohe Gebrauchstauglichkeit zu sichern.

6 Realisieren

6.1 Git-Repository aufsetzen

Das Git-Repository für das MAUI-Projekt, bekannt als "ict-322-ski-service-app", dient als zentrale Stelle für den Quellcode, die Ressourcen und die Dokumentation der Anwendung. Es wurde auf GitHub eingerichtet, um die Zusammenarbeit und Versionskontrolle zu erleichtern. Das Repository beinhaltet die MAUI-Anwendung und ist so strukturiert, dass es die verschiedenen Aspekte der App, wie Benutzeroberfläche, Backend-Verbindung und Tests, klar trennt.

Die Hauptkomponenten des Repositories sind im 'SkiServiceApp'-Verzeichnis organisiert, das alle notwendigen Dateien für die Anwendung enthält. Dies umfasst XAML-Dateien für die Benutzeroberfläche, C#-Dateien für die Logik und verschiedene Ressourcen für die Lokalisierung und das Styling. Zusätzlich gibt es Konfigurationsdateien für verschiedene Plattformen wie Android und iOS, die für die Ausführung und Anpassung der App auf unterschiedlichen Geräten notwendig sind.

6.2 Verbindung zum Backend aufbauen

Die Verbindung zum Backend ist ein wesentlicher Bestandteil der MAUI-App, um eine dynamische und interaktive Benutzererfahrung zu ermöglichen. Die App nutzt eine Reihe von API-Services, um mit dem Backend zu kommunizieren und Daten zu verwalten. Diese Services sind so konzipiert, dass sie die API-Schnittstellen implementieren und über Funktionen zugänglich machen.

Eine der Herausforderungen war der Umgang mit der speziellen Localhost-Adresse des Emulators. Dies führte zu anfänglichen Verbindungsproblemen, die durch gezieltes Debugging und Recherche behoben wurden. Das Verständnis und die Anpassung an die Netzwerkumgebung des Emulators waren entscheidend für den erfolgreichen Aufbau der Verbindung.

API-Services Implementierung

BaseAPIService
Dies ist die Grundlage für alle API-Services. Es handhabt die grundlegenden CRUD-Operationen und stellt eine generische Implementierung zur Verfügung, die von spezifischen Services erweitert wird.

Spezifische API-Services
Für jede Art von Daten oder Aktion gibt es einen spezifischen Service, wie *OrderAPIService*, *PriorityAPIService*, *ServiceAPIService*, *StateAPIService* und *UserAPIService*. Diese Services erben von *BaseAPIService* und implementieren teilweise zusätzliche spezifische Funktionen.

6.3 Entwicklung der Benutzeroberfläche

Die Entwicklung der Benutzeroberfläche war ein zentraler Aspekt des MAUI-Projekts, um eine intuitive und effiziente Nutzererfahrung zu gewährleisten. Die App zielt darauf ab, den Mitarbeitern von Jetstream eine einfache und schnelle Möglichkeit zu bieten, ihre Aufträge zu verwalten, insbesondere unter Berücksichtigung der Arbeitsumgebung, in der oft Handschuhe getragen werden.

Die entwickelte Benutzeroberfläche ermöglicht es den Mitarbeitern von Jetstream, ihre Aufträge effizient und effektiv zu verwalten. Die intuitive Gestaltung und die Berücksichtigung spezifischer Arbeitsbedingungen tragen dazu bei, dass die App reibungslos in den Arbeitsalltag integriert werden kann. Durch die Verwendung des MAUI Frameworks ist die App zudem zukunftssicher und leicht auf weitere Plattformen erweiterbar.

6.3.1 MVVM-Architektur

In der Entwicklung der Benutzeroberfläche der Ski-Service-App wurde die Model-View-ViewModel (MVVM) Architektur implementiert, um eine klare Trennung von Logik und Darstellung zu gewährleisten. Diese Entscheidung unterstützt eine modulare und wartbare Codebasis, die für die Skalierbarkeit und Wartung der App entscheidend ist.

MVVM für Pages

Die MVVM-Architektur wurde speziell für die Seiten (Pages) der App umgesetzt. Jede Seite hat ein zugehöriges ViewModel, das die Datenlogik und Interaktionen handhabt. Die Views binden sich an diese ViewModels, wodurch eine klare Trennung der Benutzeroberflächenlogik und der Geschäftslogik erreicht wird. Dies erleichtert das Testen und die Wiederverwendung von Code und ermöglicht ein reaktives Design, bei dem sich die Benutzeroberfläche dynamisch an die zugrunde liegenden Daten anpasst.

Code-Behind für Components

Für kleinere Komponenten und Dialoge haben wir uns entschieden, die traditionelle Code-Behind-Technik zu verwenden. Dieser Ansatz wurde gewählt, um die Komplexität zu reduzieren und die Entwicklungsgeschwindigkeit für kleinere, weniger komplexe Teile der Benutzeroberfläche zu erhöhen. Obwohl diese Komponenten nicht die MVVM-Architektur nutzen, sind sie dennoch so gestaltet, dass sie gut mit den MVVM-Teilen der App interagieren und eine konsistente Benutzererfahrung bieten.

6.3.2 Design und Implementierung

Benutzerzentriertes Design

Das Design der Benutzeroberfläche wurde mit einem Fokus auf Benutzerfreundlichkeit und Effizienz entwickelt. Besonderes Augenmerk wurde auf die Minimierung der erforderlichen Interaktionen (Tipp-Eingaben und Mausklicks) gelegt, um die Bedienung auch mit Handschuhen praktikabel zu machen.

MAUI Framework

Die App wurde unter Verwendung des .NET MAUI Frameworks entwickelt, das eine plattformübergreifende Entwicklung von Benutzeroberflächen ermöglicht. Dies erleichterte die konsistente Darstellung und Funktionalität der App über verschiedene Geräte hinweg.

6.3.3 Komponenten und Layout

Seiten und Navigation

Die App besteht aus mehreren Seiten, die verschiedene Funktionen und Informationen darstellen, wie das Dashboard, die Auftragsliste und die Detailansichten. Die Navigation zwischen diesen Seiten ist intuitiv gestaltet, mit einer klaren Struktur und (fast) fliessenden Übergängen.

Interaktive Elemente

Interaktive Elemente wie Buttons, Listen und Formulare wurden so gestaltet, dass sie leicht zu verstehen und zu bedienen sind. Die Größe und Abstände der Elemente berücksichtigen die Nutzung mit Handschuhen.

6.3.4 Anpassung und Stil

Responsive Design

Das Layout passt sich flexibel an verschiedene Bildschirmgrößen und Orientierungen an, um eine optimale Darstellung auf Tablets, Surfaces und Handys zu gewährleisten.

Stil und Thema

Die visuelle Gestaltung folgt einem konsistenten Thema, das die Markenidentität von Jetstream widerspiegelt. Farben, Schriftarten und Icons wurden sorgfältig ausgewählt, um eine angenehme und klare Benutzeroberfläche zu schaffen.

6.3.5 Multi-Lingual Support

Die Benutzeroberfläche der Ski-Service-App wurde mit umfangreichem Multi-Lingual Support entwickelt, um eine breite, weltweite Nutzerbasis anzusprechen. Durch die Integration von Sprachressourcen in .resx-Dateien bietet die App lokalisierte Texte für UI-Elemente wie Menüs, Dialoge und Anweisungen in verschiedenen Sprachen, darunter Englisch, Spanisch, Französisch und Deutsch. Dies verbessert die Zugänglichkeit und Benutzerfreundlichkeit erheblich, indem Benutzer die App in ihrer bevorzugten Sprache nutzen können.

6.3.6 Schlüsselkomponenten

In der Entwicklung der Benutzeroberfläche der Ski-Service-App wurden spezielle Schlüsselkomponenten implementiert, um die Verwaltung und Darstellung von Aufträgen zu optimieren und zu vereinfachen.

AuthManager

Der *AuthManager* ist für die Handhabung der Authentifizierung zuständig. Er ermöglicht das Einloggen und Ausloggen von Benutzern, verwaltet den Zugriff auf Benutzerdaten wie Tokens und Benutzer-IDs und signalisiert Änderungen im Anmeldestatus durch Events. Dies gewährleistet eine sichere und reibungslose Benutzererfahrung, indem er sicherstellt, dass nur autorisierte Benutzer Zugriff auf bestimmte Funktionen und Daten haben. Der *AuthManager* ist somit zentral für die Sicherheit und Benutzerfreundlichkeit der App.

SettingsManager

Der *SettingsManager* verwaltet die Benutzereinstellungen. Er ermöglicht es Benutzern, ihre Präferenzen wie Sprache und Thema festzulegen und sorgt dafür, dass diese Einstellungen konsistent angewendet werden. Durch die Speicherung und das Laden von Benutzerpräferenzen trägt die Klasse wesentlich zur Personalisierung und Benutzerfreundlichkeit der App bei.

OrderCollection als Wrapper Type
Die *OrderCollection* ist ein maßgeschneiderter Typ, der als Wrapper für die Verwaltung von Auftragslisten dient. Diese Klasse erweitert *ObservableCollection<CustomListItem>* und bietet eine vereinfachte Nutzung auf mehreren Seiten. Sie ermöglicht es, die Liste mit den neuesten Daten aus der API zu aktualisieren und diese Daten zu sortieren. Die *OrderCollection* ist zentral für die Handhabung von Auftragsdaten und trägt zur Effizienz der App bei, indem sie eine reaktive und aktualisierbare Liste von Aufträgen bereitstellt.

CustomListItem als Datenklasse
Das *CustomListItem* ist eine weitere Kernkomponente und dient als Datenklasse für die *OrderList*. Es enthält alle Logiken zur Darstellung von Aufträgen auf individueller Basis. Diese Klasse beinhaltet Informationen wie die verbleibenden Tage bis zur Fertigstellung des Auftrags, Priorität, Zustand und Dienstleistung in der aktuellen Sprache sowie Funktionen zum Aktualisieren und Verwalten des Auftragsstatus. Die Verwendung von *CustomListItem* ermöglicht eine flexible und detaillierte Darstellung von Aufträgen in der Benutzeroberfläche und trägt zur intuitiven Bedienung der App bei.

OrderList
Die *OrderList* ist eine zentrale Komponente für die Darstellung der Aufträge. Sie verwendet die *OrderCollection* und *CustomListItem* für eine dynamische und reaktive Anzeige der Auftragsdaten. Die *OrderList* ist als *CollectionView* implementiert und ermöglicht es Benutzern, durch die Aufträge zu scrollen und Details zu jedem Auftrag zu sehen. Jedes Element in der Liste ist interaktiv und ermöglicht Aktionen wie das Ändern des Status oder das Zuweisen von Aufträgen. Die OrderList ist flexibel und kann an verschiedenen Stellen in der App verwendet werden, um eine konsistente und effiziente Darstellung von Aufträgen zu gewährleisten.

Localization
Die Lokalisierung ist ein wesentlicher Aspekt der Benutzeroberfläche, um Multi-Lingual Support zu bieten. Die *Localization*-Klasse verwaltet die Sprachressourcen und ermöglicht eine dynamische Sprachumschaltung in der App. Sie bindet UI-Elemente an lokalisierte Texte, die in .resx-Dateien definiert sind, und aktualisiert die Anzeige sofort bei einem Sprachwechsel.

DialogPage
Die *DialogPage* ist eine spezielle Seite, die als Container für Dialoge dient. Sie ermöglicht es, Dialoge in einem modularen und kontrollierten Umfeld anzuzeigen. Die *DialogPage* verwendet Animationen, um das Ein- und Ausblenden von Dialogen zu steuern, und bietet eine visuell ansprechende Erfahrung. Sie ist so konzipiert, dass sie flexibel mit verschiedenen Dialoginhalten umgehen kann und eine zentrale Stelle für die Darstellung aller Dialoge in der App ist.

Dialog Manager
Anstelle eines Dienstes verwendet die App statische Funktionen, um Dialoge zu verwalten. Diese Funktionen ermöglichen es, Dialoge dynamisch zu erstellen, anzuzeigen und zu schließen. Der Dialogmanager kümmert sich um die Logik, die notwendig ist, um sicherzustellen, dass Dialoge korrekt in der Benutzeroberfläche dargestellt werden und dass Benutzeraktionen angemessen behandelt werden. Dieser Ansatz ermöglicht eine zentrale Steuerung und Verwaltung aller Dialoge in der App.

6.4 Entwicklung der nötigen Tests

In der Entwicklungsphase des MAUI-Projekts wurde ein besonderes Augenmerk auf das Testen gelegt. Ursprünglich war geplant, sowohl Unit- als auch UI-Tests zu implementieren. Im Laufe der Entwicklung stellte sich jedoch heraus, dass UI-Tests experimenteller und weniger verlässlich waren als erwartet. Daher entschieden wir, auf bewährte und offizielle Methoden zurückzugreifen und eine Kombination aus Unit- und Integrationstests zu schreiben.

Unit-Tests

Unit-Tests wurden für einzelne Komponenten und Funktionen der App geschrieben, um sicherzustellen, dass diese wie erwartet funktionieren. Diese Tests sind isoliert und fokussieren sich auf die kleinste testbare Einheit der Anwendung. Beispiele für Unit-Tests im Repository sind AuthManagerTests und *HttpResponseTests*.

Integrationstests

Integrationstests wurden implementiert, um die Zusammenarbeit zwischen verschiedenen Teilen der Anwendung zu überprüfen. Diese Tests sind besonders wertvoll, um sicherzustellen, dass die Integration von verschiedenen Services und Komponenten reibungslos funktioniert. Beispiele für Integrationstests im Repository sind *AppLoginViewModelTests*, *AppShellViewModelTests* und *OrderDetailViewModelTests.*

7 Kontrollieren

7.1 Test-Strategie ausführen

In dieser Phase des Projekts lag der Fokus auf der Durchführung und Dokumentation von Tests, um die Funktionalität und Zuverlässigkeit der entwickelten API sicherzustellen. Die Teststrategie umfasste sowohl Unit Tests als auch grundlegende Integrationstests.

Unit-Tests
Die Unit Tests zielten darauf ab, die Funktionalität der einzelnen Komponenten isoliert zu überprüfen. Dies beinhaltete die Validierung der Kernfunktionalitäten, das korrekte Handling von Eingabeparametern und die angemessene Reaktion auf verschiedene Instanziierung-/Aufruf-möglichkeiten.

Die Ergebnisse der Unit Tests wurden in Visual Studio dokumentiert und sind im beigefügten Testbericht detailliert aufgeführt. Sie zeigen eine hohe Abdeckung und erfolgreiche Validierung der-Komponenten.

Ort: files/Berichte/test_results.trx

Integrationstests
Die Integrationstests dienten dazu die Zusammenarbeit zwischen verschiedenen Komponenten der Anwendung zu überprüfen und sicherzustellen. In diesem Fall mit speziellem Fokus auf die Integration des Backends in das Frontend.

Die Integrationstests werden mithilfe einer CollectionFixture «Gruppiert» was dafür sorgt, das Docker am Anfang gestartet und am Ende wieder beendet wird.

Ort: files/Berichte/test_results.trx

Testbericht
Die Testergebnisse wurden sorgfältig dokumentiert und sind im angehängten Testbericht einsehbar. Dieser Bericht enthält detaillierte Informationen über die durchgeführten Tests, einschließlich der Testfälle, der erzielten Ergebnisse und der festgestellten Probleme.

7.2 Anforderungen mit dem Produkt abgleichen

Bei der eingehenden Überprüfung des Ski-Service-Management-Projekts, konnten wir feststellen, dass alle vorgegebenen Anforderungen mit großer Sorgfalt und technischem Verständnis umgesetzt wurden. Die Anwendung präsentiert sich als ein robustes System, das eine Vielzahl von Funktionen und eine benutzerfreundliche Oberfläche bietet.

Login und Authentifizierung (A1, AO4)
Ein sicherer Login-Dialog wurde implementiert, der die Mitarbeiter mit einer Passwortauthentifizierung autorisiert. Besondere Merkmale wie das Sperren des Zugangs nach mehreren Fehlversuchen tragen zur Sicherheit bei.

Benutzer- und Auftragsmanagement (A5, A6, AO5)
Die App ermöglicht es den Mitarbeitern, Aufträge effektiv zu verwalten. Sie können Änderungen am Status vornehmen und Aufträge detailliert betrachten und bearbeiten. Durch die Verwendung des MVVM-Designmusters wird eine klare Trennung von Präsentation und Logik erreicht, was die Wartung und Erweiterbarkeit verbessert.

Benutzer- und Auftragsverwaltung (A2, A3, A4, A5, AO2, AO5, AO6, AO7)
Die Applikation verwaltet effektiv die Serviceaufträge und Benutzerinformationen. Mitarbeiter können Aufträge anlegen, bearbeiten, filtern und ihren Status ändern. Die Integration personalisierter Listen und Kommentarfunktionen erweitert die Interaktivität und Funktionalität der App.

Darstellung und Navigation (A8, AO1)
Mit dem Einsatz des MVVM-Design-Musters und der MAUI-Framework-Features bietet die Anwendung eine intuitive und aufgabenangemessene Benutzerführung. Dashboards und Statistiken liefern einen schnellen Überblick und fördern die Benutzererfahrung.

Erweiterte Funktionalitäten und Sicherheit (A7, AO3, AO8)
Durch die Nutzung von RESTful APIs und robusten Backend-Services ermöglicht die Anwendung einen reibungslosen Datenaustausch und eine flexible Datenverwaltung. Die Implementierung von "Soft Deletes" bei Aufträgen gewährleistet Datenintegrität und eine nachvollziehbare Historie.

Normen und Dokumentation (A9, A10)
Die Einhaltung von Design- und Entwicklungsstandards wie EN ISO 9241-110 sowie eine umfassende Dokumentation des Projekts nach dem IPERKA-Modell stellen sicher, dass die Anwendung nicht nur funktional, sondern auch methodisch und in der Dokumentation erstklassig ist.

8 Auswerten

8.1 Fazit

Im Verlauf unseres Projekts, einer Ski-Service-Management-Applikation, stellten wir uns den Herausforderungen und Chancen, die mit der Einführung von .NET Maui als neuem Framework einhergehen. Trotz der anfänglichen Einarbeitungszeit und der relativen Jugend von .NET Maui, die einige Aspekte erschwerte, erwies sich die Wahl als bereichernd. Mit Geduld, Google und Kaffee konnten wir uns in die Thematik einarbeiten und die meisten Probleme lösen. Besonders das Konzipieren der Listenansicht war für uns ein spannender Aspekt, bei dem es galt, Performance und Design geschickt zu verbinden.

Die Arbeit mit .NET Maui war nach Überwindung der anfänglichen Lernkurve nicht nur lehrreich, sondern machte auch Spaß und eröffnete neue Perspektiven in der Applikationsentwicklung. Wir haben die Übergänge zwischen verschiedenen Views optimiert und eine effiziente, nutzerfreundliche Lösung implementiert. Diese Erfahrung hat uns ein tieferes Verständnis des Frameworks sowie der Anwendungslogik gebracht.

Ein weiterer positiver Aspekt war die schnelle und reibungslose Integration des bestehenden Backends in die neue .NET Maui-Anwendung. Trotz einiger verbliebener Herausforderungen, wie leichten Lags in der UI und der Notwendigkeit verbesserter Fehlerbehandlung, hat uns das Projekt gezeigt, wie flexibel und kompatibel .NET Maui mit bestehenden Technologien ist.

Obwohl das Projekt zum aktuellen Zeitpunkt noch nicht produktionsbereit ist, sind wir stolz auf unsere Arbeit und fühlen uns durch die überwundenen Herausforderungen bereichert. Die Kombination aus technischem Fortschritt, kreativer Problemlösung und dem erfolgreichen Abschluss komplexer Aufgaben hinterlässt ein tiefes Gefühl der Zufriedenheit.

Für zukünftige Projekte würden wir .NET Maui aufgrund seiner Vorteile, des Spaßfaktors und der fortschrittlichen Technologie erneut wählen, trotz gelegentlicher, unerwarteter Fehler, die typisch für Microsoft-spezifische Produkte sein können. Unser Projekt hat nicht nur unsere technischen Fähigkeiten verbessert, sondern auch das Vertrauen in die Wahl moderner Entwicklungswerkzeuge gestärkt. Wir betrachten unser Projekt als einen Erfolg im Rahmen eines Schulprojekts und freuen uns auf zukünftige Entwicklungen mit .NET Maui.

9 Anhänge

9.1 Verwendete NuGet Pakete

Paket	Version
CommunityToolkit.Maui	7.0.1
Fody	6.8.0
Microsoft.Extensions.Logging.Debug	8.0.3
Microsoft.Maui.Controls	8.0.3
Microsoft.Maui.Controls.Compatibility	8.0.3
Newtonsoft.Json	13.0.3
PropertyChanged.Fody	4.1.0
SkiServiceModels	1.0.50
Syncfusion.Licensing	24.1.44
Syncfusion.Maui.Charts	24.1.44

3 - NuGet Paket Versionen

9.2 Glossar

Abkürzung	Ausgeschrieben	Erklärung
UI	User Interface	Beschreibt die Oberfläche, durch die der Benutzer mit einer Maschine interagiert, in diesem Fall die Anwendungsschnittstelle.
API	Application Programming Interface	Eine Reihe von Routinen, Protokollen und Werkzeugen zum Erstellen von Software und Anwendungen. API ermöglicht die Interaktion zwischen Softwareprodukten.
CRUD	Create, Read, Update, Delete	Die vier grundlegenden Funktionen, die in Datenbankanwendungen implementiert sind, um Daten zu erstellen, zu lesen, zu aktualisieren und zu löschen.
DPI	Dots Per Inch	Eine Maßeinheit für die Druck- oder Bildauflösung in einem digitalen Bild.
MVVM	Model-View-ViewModel	Ein Softwarearchitekturmuster, das zur Trennung von Geschäftslogik und Benutzeroberfläche verwendet wird.
HTTPS	Hypertext Transfer Protocol Secure	Eine Erweiterung des HTTP-Protokolls mit dem Ziel, sichere Transaktionen im Internet zu ermöglichen.
MSSQL	Microsoft SQL Server	Ein relationales Datenbankmanagementsystem von Microsoft.
MAUI	.NET Multi-platform App UI	Eine plattformübergreifende Framework-Initiative von Microsoft, die die Entwicklung von Anwendungen für Android, iOS, macOS und Windows vereinfacht.

4 – Glossar

10 Verweise

10.1 Quellen

Im aktuellen Dokument sind keine Quellen vorhanden.

10.2 Abbildungsverzeichnis

10.3 Tabellenverzeichnis