Open Education Platform for Management Schools

Publikationstyp: Lehrmaterialien

Case Study Timesoft

Version Nr. 1, 9. Juni 2022

Tremp, Hansruedi
OST - Ostschweizer Fachhochschule

Publiziert auf: www.oepms.org
Unter doi: 10.25938/oepms.297

Open Education Platform
for Management Schools

Case Study Timesoft

Version Nr. 1, 9. Juni 2022

Tremp, Hansruedi
OST - Ostschweizer Fachhochschule

Publikationsform: Fallstudie
Institution: OST - Ostschweizer Fachhochschule
Schlüsselbegriffe: Softwarearchitektur; UML; Mehrschichtenarchitektur; Integrationsarchitektur
Einsatzbereich: Bachelorstudierende

Zitierweise nach APA:
Tremp, H. (2022). Case Study Timesoft. *Open Education Platform*. DOI: 10.25938/oepms.297

Open Education Platform
for Management Schools

Didaktischer Reflexionsbericht:

Case Study Timesoft

Hansruedi Tremp

OST – Ostschweizer Fachhochschule, Rosenbergstrasse 59, 9001 St. Gallen,
hansruedi.tremp@ost.ch

Abstract. Die Lehre über Architekturen verteilter Softwaresysteme im Rahmen des B.Sc. Wirtschaftsinformatik ist in vielerlei Hinsicht eine Herausforderung. Einerseits darf der Aufgabenstellende den Anwendungskontext nicht zu umfangreich fassen, um Überforderungen bzw. ein zeitliches Ausufern zu verhindern. Andererseits darf er aber die Aufgaben auch nicht zu trivial bzw. zu eng vorgegeben, um einen optimalen Lerneffekt nicht zu verhindern. «Timesoft» orientiert sich an der Praxisumgebung einer Schweizer Softwareentwicklungsfirma. Der Autor steht mit dessen CTO immer wieder im Austausch zwischen Theorie und Praxis. Die aktuelle Fallstudie ist seit ca. 5 Jahren im Modul zum Thema Software-/Applikationsarchitekturen im Hauptstudium (4./5. Semester) des B.Sc. Wirtschaftsinformatik im Einsatz und wurde dabei fortlaufend erweitert bzw. aktualisiert. Die Studierenden schätzen den Praxisbezug und die Möglichkeit, abstrakte Referenzarchitekturen an einem konkreten Fall anzuwenden und so den Nutzen der Referenzarchitektur direkt zu erleben.

Inhaltsverzeichnis

Abkürzungen

AD	Active Directory
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
async	asynchron
BSc	Bachelor of Science
Buha	Buchhaltung
CTO	Chief Technology Officer
DAC	Data Access Component
DBMS	Data Base Management System
DevOps	Development and Operations
EAI	Enterprise Application Integration
EDI	Electronic Data Interchange
EJB	Enterprise Java Beans
ERP	Enterprise Resource Planning
IDoc	Intermediate Documents
ICT	Information and Communication Technology
JMS	Java Message Service
JPA	Java Persistence API
JSF	Java Server Faces
IT	Information Technology
LDS	Lightweight Directory Service
MA	Mitarbeiter
MAUI	Multi-Platform App UI
MQ	Message Broker
MSc	Master of Science
OData	Open Data Protocol
ORM	Object Relational Mapping
REST	REpresentational State Transfer
SMTP	Simple Mail Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPA	Single Page App
sync	synchron
UBL	Universal Business Language
UI	User Interface
UML	Unified Modeling Language
UWP	Universal Windows Platform
WPF	Windows Presentation Foundation

1 Didaktischer Reflexionsbericht

1.1 Einführung

Die Lehre über Architekturen verteilter Softwaresysteme im Rahmen des B.Sc. Wirtschaftsinformatik ist in verschiedener Hinsicht eine Herausforderung. Einerseits darf ein Anwendungskontext nicht zu umfangreich sein, um Überforderungen zu verhindern. Andererseits darf es aber auch nicht zu trivial bzw. zu eng vorgegeben sein, um einen Lerneffekt nicht zu verhindern. «Timesoft» orientiert sich an der Praxisumgebung der ALL CONSULTING AG von St. Gallen. Mit dessen CTO (Chief Technology Officer) steht der Autor immer wieder im Austausch zwischen Theorie und Praxis. Die aktuelle Fallstudie ist seit ca. 5 Jahren im Modul zum Thema der Software-/Applikationsarchitektur im Hauptstudium (4./5. Semester) des B.Sc. Wirtschaftsinformatik im Einsatz. Die Studierenden schätzen den Praxisbezug und die Möglichkeit, abstrakte Referenzarchitekturen in einem konkreten Fall anzuwenden und so den Nutzen der Referenzarchitektur direkt zu erleben. Der Aufgabenstellende kann rasch erkennen, wo noch Lücken bzw. Missverständnisse im Stoff sind.

1.2 Lernziele

Die Zielgruppe bilden Studierende der Informatik bzw. Wirtschaftsinformatik im B.Sc. Hauptstudium (2./3. Studienjahr) oder auch im M.Sc. (1. Studienjahr). Die Einführung in die Programmierung mit einer konkreten Programmiersprache sowie die Grundlagen von Datenbanktechnologien bilden die Voraussetzung.

Die Studierenden können

- die Grundprinzipien von SOA in einer konkreten Architektur anwenden.
- die Makroarchitektur von einer auf Microservice basierten Applikation gestützt auf die erhobenen Anforderungen in einer vorgegebenen Systemumgebung mittels UML modellieren.
- die Schichtenarchitektur von verteilten IT-Systemen mit Web- und Mobilekomponenten problemadäquat mittels UML (Unified Modeling Language) modellieren.
- die Integrationsarchitektur von EAI (Enterprise Application Integration) und B2x-Integration erarbeiten und mittels UML abbilden.

1.3 Lehrplan

Die Fallstudie wird im Rahmen des Lernblocks «Verteilte Softwarearchitekturen» eingesetzt. Sie begleitet den Unterricht über 6 Unterrichtswochen. Dabei sind folgende Zeiten vorzusehen:

- Vorstellung und Einführung der Fallstudie im Unterricht ca. 30 Min.
- Die Aufgabenstellung wird im Rahmen einer Gruppe von 3 - 5 Studierenden gelöst.
- Für die Lösung der 6 Aufgaben ist im Durchschnitt mit ca. 1-2 Std. Aufwand je Aufgabe in der Gruppe zu rechnen.
- Für die Besprechung im Unterricht und einem primären Feedback sind ca. 15 min. je Gruppe einzuplanen.
- Zur Förderung der Beurteilungskompetenz sind Peer-Reviews vorgesehen, wobei jede Gruppe jeweils die Arbeiten von 2 anderen Gruppen gemäss den gegebenen Kriterien bewertet.

- Für die Korrektur und Bewertung der Gruppenarbeit hat der Dozierende mit ca. 30 min. je Gruppe zu rechnen.

1.4 Bewertungskriterien für die Diskussion

Bei den Lösungen der einzelnen Aufgaben sind folgende Kriterien zu beurteilen:

- Vollständigkeit:
    - Alle relevanten Aspekte der Ausgangslage sind adäquat berücksichtigt.
- Referenzarchitektur:
    - Der Bezug zur passenden Referenzarchitektur ist dargelegt.
- Zweckmässigkeit:
    - Die vorgeschlagene Lösung ist praktikabel und zielführend.
- UML-Komponentendiagramm:
    - Die Notation ist korrekt angewendet.
- Argumentation:
    - Die Begründungen sind in sich schlüssig und zeigen einen klaren Bezug sowohl zur Aufgabenstellung als auch zur Theorie.

Zu jeder Aufgabenstellung gibt es im Anhang einen Lösungsvorschlag mit weiteren Erläuterungen.

Open Education Platform
for Management Schools

Fallstudie:

Case Study Timesoft

Hansruedi Tremp

OST – Ostschweizer Fachhochschule, Rosenbergstrasse 59, 9001 St. Gallen,
hansruedi.tremp@ost.ch

Abstract. Die Lehre über Architekturen verteilter Softwaresysteme im Rahmen des B.Sc. Wirtschaftsinformatik ist in vielerlei Hinsicht eine Herausforderung. Einerseits darf der Aufgabenstellende den Anwendungskontext nicht zu umfangreich fassen, um Überforderungen bzw. ein zeitliches Ausufern zu verhindern. Andererseits darf er aber die Aufgaben auch nicht zu trivial bzw. zu eng vorgegeben, um einen optimalen Lerneffekt nicht zu verhindern. «Timesoft» orientiert sich an der Praxisumgebung einer Schweizer Softwareentwicklungsfirma. Der Autor steht mit dessen CTO immer wieder im Austausch zwischen Theorie und Praxis. Die aktuelle Fallstudie ist seit ca. 5 Jahren im Modul zum Thema Software-/Applikationsarchitekturen im Hauptstudium (4./5. Semester) des B.Sc. Wirtschaftsinformatik im Einsatz und wurde dabei fortlaufend erweitert bzw. aktualisiert. Die Studierenden schätzen den Praxisbezug und die Möglichkeit, abstrakte Referenzarchitekturen an einem konkreten Fall anzuwenden und so den Nutzen der Referenzarchitektur direkt zu erleben.

2 Fallstudie

2.1 Einführung

Die AllCons AG hat sich im deutschsprachigen Markt etabliert und betreut ihre Kundschaft von vier Standorten aus. Das bestehende Zeit- und Spesenrapportierungs- sowie Projektabrechnungssystem genügt schon seit geraumer Zeit nicht mehr. Die neu zu entwickelnde verteilte Softwarelösung soll die internen Bedürfnisse sowie die Anforderungen einer potenziellen Kundschaft aus verschiedenen Branchen abdecken. Dabei möchte AllCons die Applikation selbst verwenden und im SaaS-Cloudmodell auf dem Markt anbieten.

2.2 Unternehmung

Seit mehr als 20 Jahren implementiert die AllCons erfolgreich ERP-Gesamtlösungen mit einer Standard Business Software sowie unterschiedlichste Business und Management Services mit eigenen Software-Plattformen. Dabei unterstützt sie ihre Kundschaft mit On Premises Installationen und einem SaaS-Cloudangebot. Weiter bieten sie eine breite Palette an Dienstleistungen und Services zur Digitalisierung von Geschäftsprozessen an. Mit den verschiedenen Geschäftsstellen legen sie Wert auf eine lokale Verankerung.

2.3 ICT-Strategie

AllCons hat sich ihre IT-Strategie vor Jahren erarbeitet und überprüft diese in einem jährlichen Strategie-Workshop. Nachfolgend sind einige für die Software- bzw. Applikationsarchitektur relevanten Eckpunkte erwähnt. Die konsequente Umsetzung von SOA (Service Oriented Architecture) steht ganz zuvorderst. Zu entwickelnde Applikationen sollen konsequent auf die Bedürfnisse der Kundschaft und gleichzeitig für den Eigenbedarf ausgerichtet sein. Für die optimale Integration mit den Umsystemen sind offene standardisierte Services vorzusehen.

Clientseitig strebt AllCons die Entwicklung von Mobile-First Web Apps an. Je nach Bedürfnislage sind installierbare Apps zu entwickeln. Die Verbindung zum Web API (Application Programming Interface) erfolgt über GraphQL (siehe https://graphql.org/).

Die Softwareentwicklung erfolgt primär mit eigenen Mitarbeitern und mit einem Netzwerk von Subunternehmen, mit welchen die AllCons eine langfristige Zusammenarbeit verfolgt. Bei den nicht selbst entwickelten Applikationen strebt die AllCons, wenn immer möglich Cloudlösungen mit dem Best-of-Breed-Ansatz an.

2.4 Anforderungen

Die Mitarbeitenden des Requirements Engineering haben folgendes Anwendungsfallmodell vorgelegt:

Anwendungsfallmodell

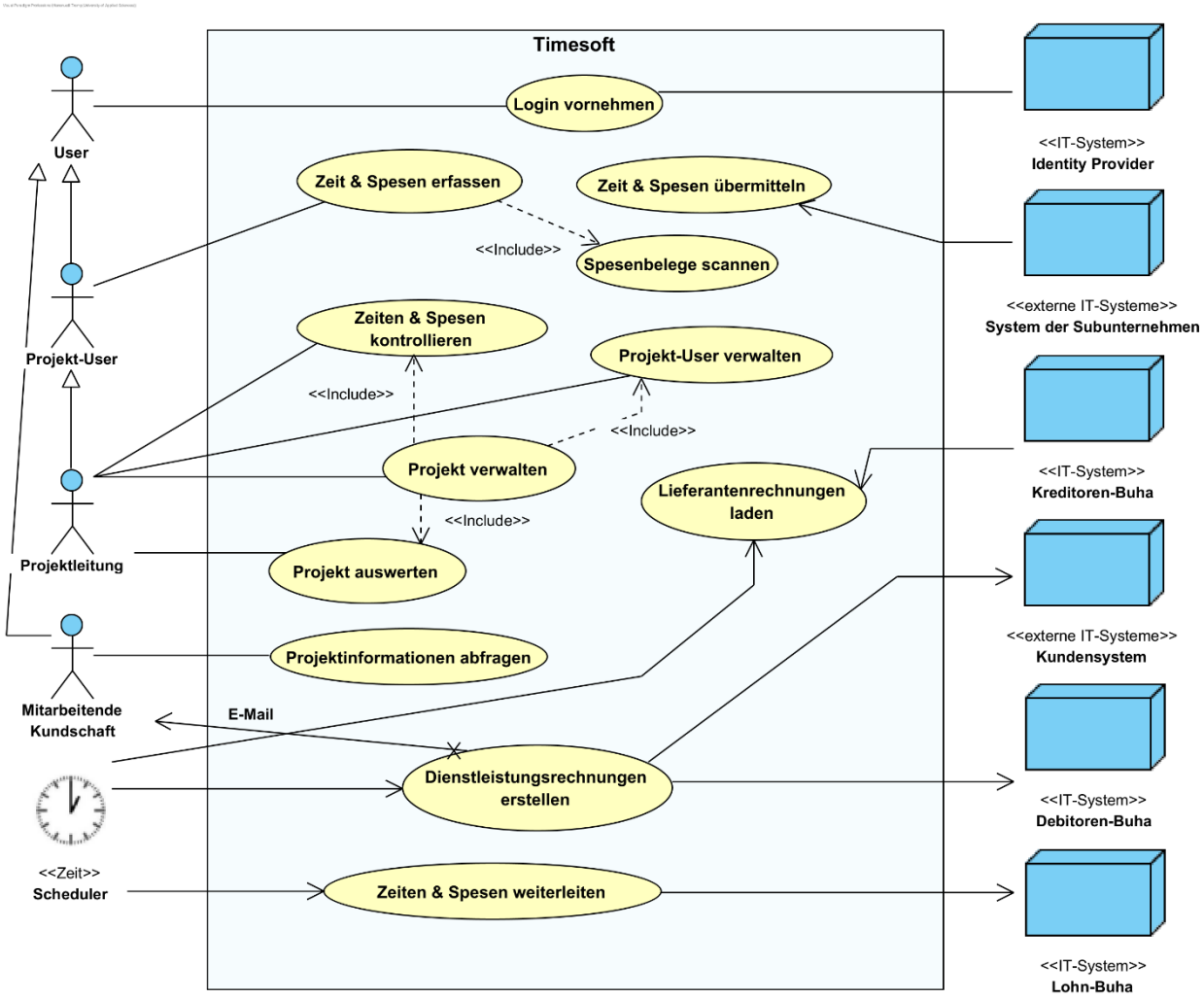


Abb. 1: Anwendungsfalldiagramm von «Timesoft»

Nachfolgend sind die einzelnen Anwendungsfälle kurz erläutert:

Login vornehmen

Die Authentisierung und Autorisierung der User erfolgt über den intern verfügbaren Identity Provider. Es ist eine 2-Phase-Authentisierung vorzusehen.

Projekt verwalten

Die Projektleitung[1] eröffnet Projekte und pflegt seine Projektdaten. Sie hat direkten Zugriff auf die Anwendungsfälle Projekt-User verwalten, Zeiten & Spesen kontrollieren und Projekt auswerten.

[1] Die Projektleitung benötigt eine App, welche die optimale Usability mit allen für sie relevanten Funktionen bietet. Weiter soll die App lokal Daten speichern und diese periodisch synchronisieren. Aktuell setzen die Projektleitung nebst Windows auch macOS-Geräte ein.

Projekt-User verwalten

Die Projektleitung verwaltet die Mitarbeitenden seines Projektes. Diese können sowohl von AllCons als auch von den Subunternehmen sein.

Zeit & Spesen erfassen

Die internen und externen Projekt-Mitarbeitenden erfassen die Zeiten und Spesen auf ihren Geräten (Betriebssysteme bei Externen meist unbekannt) sowie auf mobilen Devices. Mit den mobilen Devices müssen die Projektmitarbeitenden auch offline arbeiten können, d.h. die relevanten Daten sind zu synchronisieren. Eine optimale Usability ist besonders wichtig, damit möglichst wenige Service-Tickets entstehen.

Spesenbelege scannen

Die Projekt-Mitarbeitenden scannen die Spesenbelege über die Kamera im Mobile-Phone ein. Das System hinterlegt die Belege auf dem Server als PDF.

Zeit & Spesen übermitteln

Für die Subunternehmen ist zusätzlich ein SOAP (Simple Object Access Protocol) Webservice für die Übermittlung ihrer Zeiten und Spesen anzubieten. Die Systeme der Subunternehmen senden dabei die Daten periodisch in dem von uns vorgegebenen XML-Format.

Zeit & Spesen kontrollieren

Die Projektleitung überprüft periodisch die Zeiten und Spesen ihres Projektes und gibt diese für die weitere Verarbeitung frei.

Lieferantenrechnungen laden

Rechnungen werden in der Kreditoren-Buha (Buchhaltung) erfasst und dem entsprechenden Projekt zugeordnet. Täglich holt sich «Timesoft» die neuen Daten von der REST (REpresentational State Transfer) API der Kreditoren-Buha. Die Semantik der Daten ist in dieser Schnittstelle beschrieben.

Projekt auswerten

Die Projektleitung wertet mit einer Informations-Cockpit-Seite seine Projekte nach unterschiedlichsten Kriterien aus.

Für weitere Ad-Hoc-Auswertungen mit Excel verwendet die Projektleitung eine von «Timesoft» zur Verfügung gestellte OData (Open Data Protocol) Schnittstelle.

Projektinformationen abfragen

Die autorisierten Mitarbeitenden der Kunden fragen bei ihren Projekten mittels einer klassischen Extranet Web App verschiedene Informationen ab, laden Projektdaten herunter oder senden Projektmitarbeitenden eine Meldung zu. Die Anforderungen an den Web-Browser sollen möglichst gering sein, damit die Tätigkeit auf allen Plattformen und mit möglichst allen aktuell verfügbaren Browser getätigt werden kann.

Dienstleistungsrechnungen erstellen

Monatlich erstellt «Timesoft» die Rechnungen für die freigegebenen Stunden und Spesen. Je nach hinterlegter Präferenz im Projekt stellt das System der Kundschaft die Rechnung via E-Mail oder als UBL-Message (Universal Business Language, siehe https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ubl) zu.

«Timesoft» liefert die Rechnungsdaten unmittelbar an die Debitoren-Buha, welche diesbezüglich ein gRPC-API (siehe https://grpc.io/) anbietet. Die Debitoren-Buha gibt die Semantik der Nachricht vor.

Zeiten & Spesen weiterleiten

Wöchentlich übergibt «Timesoft» die Spesen asynchron im IDoc-Format (Intermediate Documents, siehe https://support.sap.com/en/alm/sap-focused-run/expert-portal/integration-cloud-monitoring/Idoc.html) an die Lohn-Buha zur Auszahlung in der nächsten Lohnverarbeitung. Für Mitarbeitenden im Stundenlohn meldet «Timesoft» die geleisteten Stunden.

2.5 Systemumfeld

Die nachfolgend aufgelisteten Systeme, Services bzw. Service-Provider stehen in der AllCons zur Verfügung. Die IT-Leitung möchte, dass ausschliesslich diese verwendet werden.

- SMTP-Server (Simple Mail Transfer Protocol): Alle E-Mails sind über diesen Service zu versenden.
- Azure AD-Service (Active Directory): Die IT verwaltet alle Anwender zentral in diesem System, der AD-Service bietet ein AD LDS-API (Active Directory Lightweight Directory Service API) an.
- Azure Logic Apps: Die IT bildet sämtliche Integration-Workflows hier ab.
- RabbitMQ: Dieser MQ (Message Broker) bietet ein AMQP (Advanced Message Queuing Protocol) und JMS (Java Message Service) API mit den notwendigen Message-Queues an.
- EDICenter: Dieser externe EDI (Electronic Data Interchange) Provider bietet über eine SOAP-Schnittstelle alle EDI-Dienste an.

2.6 Aufgaben

Alle nachfolgenden Aufgaben zu dieser Case Study müssen die oben beschriebene Ausgangslage berücksichtigen und die im Anhang befindlichen Referenzarchitekturen korrekt im Kontext anwenden und referenzieren. Die Architekturen sind als UML Komponentendiagramm (siehe Rupp & Queins, 2012, S. 216-225) zu modellieren.

2.6.1 Festlegen der fachlichen Komponenten

Legen Sie gemäss der dargelegten Ausgangslage ca. 5 fachliche Komponenten fest. Eine mögliche Methode ist das Eruieren der Bounded Contexts im Rahmen des strategischen Designs der Methode DDD gemäss Vernon (2017, S. 11-42). Geben Sie als Resultat eine Tabelle mit Komponentenname, abgedeckte Anwendungsfälle sowie persistente Objekte. Achten Sie darauf, dass alle Anwendungsfälle durch mindestens eine Komponente abgedeckt sind. Die persistenten Objekte müssen sich aus den Anwendungsfällen ableiten lassen.

2.6.2 Festlegen der Client-Apps

Legen Sie ausgehend von der Ausgangslage die notwendigen unterschiedlichen Client Apps fest. Geben Sie das Resultat in einer Tabelle mit folgenden Spalten:

- Technologie: Art der Client App gemäss den Ausführungen in Tremp (2021, S. 84-104), wie z.B. Desktop App, Mobile App usw.
- Begründung: Geben Sie hier stichwortartig die Hauptfunktionen sowie nachvollziehbare Gründe für die Wahl dieses Client-Typs, wie z.B. optimale Usability, Portabilität usw.
- User, welche die jeweilige App nutzen.

Berücksichtigen Sie dabei auch allenfalls erwähnte Third-Party Apps.

2.6.3 Identifizieren der Applikationsschnittstellen

Geben Sie ausgehend von der Ausgangslage eine Tabelle von den erwähnten Applikationsschnittstellen der «Timesoft». Sehen Sie dabei folgende Spalten vor:

- Applikation: Umsystem, welches eine Schnittstelle zu «Timesoft» aufweist
- sync/async: Art der Schnittstelle, d.h. synchron oder asynchron
- Funktion: Funktion der Applikation in Bezug auf die Schnittstelle:
    - bei sync: Service Consumer oder Provider
    - bei async: Sender oder Receiver
- Protokoll: Transportprotokoll und sofern bekannt in Klammer die semantische Norm der Nachricht
- Anwendungsfall: Hinweis auf den Anwendungsfall, in welchem die Schnittstelle erwähnt ist

2.6.4 Vorschlag für Makroarchitektur mit Microservices

Die Projektleitung diskutiert mit dem Entwicklungsteam die Lösungsmöglichkeiten für die neue Software. Dabei melden sich viele Stimmen und zwei Szenarien zeichnen sich ab.

Eine Gruppe möchte auf eine Reihe von unabhängigen Microservices setzen. Jedes DevOps (Development and Operations) Team übernimmt dabei die Verantwortung für die Wahl des Technologie-Stacks sowie die weitere Pflege der Software. Jeder Microservice muss so gebaut sein, dass er für sich allein voll skalierbar ist.

Eine zweite Gruppe möchte auf eine bewährte monolithische Mehrschichtenarchitektur setzen, welcher eine Skalierung auf dem Web-, Business- und Data-Tier ermöglicht.

Zur Vertiefung von unterschiedlichen Architekturstile sei auf Richards & Ford (2020, S. 125-278) hingewiesen.

Erarbeiten Sie nun in einem ersten Schritt die Makroarchitektur (inkl. Client Apps gemäss Lösung der Aufgabe 2.6.2) für die Variante mit Microservices auf Basis der Referenzarchitektur gemäss Anhang. Die Grundlagen zum Thema Makroarchitektur im Rahmen von Microservices bietet Tremp (2021, S. 61-81). Eine erweiterte Vertiefung und Darlegung zum Thema Mircoservices gibt Wolff (2018) in seinem Grundlagenwerk.

Legen Sie alle notwendigen Microservices fest. Zeichnen Sie als Middleware nur einen allfällig notwendigen Message Broker ein. Erwähnen Sie im korrekten Kontext die notwendige Middleware zur Containerisierung. Modellieren Sie keine Umsysteme.

Geben Sie jeweils eine Begründung mit Bezug zur Ausgangslage bzw. Lösungen in den vorangegangenen 3 Aufgaben in wenigen ganzen Sätzen:

- Variante des User Interface
- Variante der Persistenz
- Variante der Kommunikation unter den Microservices
- angebotene Interfaces beim Web API
- jedes Paket und Komponente

2.6.5 Vorschlag für monolithische Mehrschichtenarchitektur

Wie oben erwähnt, möchte eine zweite Gruppe auf eine bewährte monolithische Mehrschichtenarchitektur setzen, welcher eine Skalierung auf dem Web-, Business- und Data-Tier ermöglicht. Die theoretischen Grundlagen zur serverseitigen Mehrschichtenarchitektur vermittelt Tremp (2021, S. 115-143).

Für diese Aufgabenstellung stehen Ihnen drei Varianten von Technologie-Stacks zur Verfügung:

- JavaScript-Basiert
- .NET-basiert
- Jakarta EE[2]-basiert

Wählen Sie für Ihre Lösung einer der drei erwähnten Technologien aus. Begründen Sie Ihren Entscheid, indem Sie Vor- und Nachteile allgemein sowie in Bezug auf die Aufgabenstellung reflektieren. Es werden ca. 5 Argumente in ganzen Sätzen erwartet.

Zeichnen Sie die Mehrschichtenarchitektur mit der gewählten Technologie mittels eines Komponentendiagramms auf. Basieren Sie dabei auf der entsprechenden im Anhang 1 ausgeführten Referenzarchitektur. Wenden Sie diese auf den spezifischen Fall korrekt an. Komponenten im Business Logic sowie Service Access Layer sind einzeln auszuführen.

Wählen Sie für die unterschiedlichen Client-Apps ein für die gewählte Technologie adäquates Framework.

Wählen Sie für die Persistenz eine für die Architektur passende Variante. Modellieren Sie diese korrekt und begründen Sie Ihre Wahl mit wenigen ganzen Sätzen.

2.6.6 Aufzeigen der Integrationsarchitektur

Identifizieren Sie alle in der Ausgangslage erwähnten Applikationen und ordnen Sie diese aus Sicht der Organisation in zwei Pakete zu.

Die theoretischen Grundlagen zur Anwendungsintegration vermittelt Tremp (2021, S. 145-165). Auf die Integration als Dienstleitung im Unternehmen gehen Pero, Kühne und Fähnrich (2014) vertieft ein.

[2] früher Java EE

Modellieren Sie die Integrationsarchitektur als UML-Komponentendiagramm gemäss der Referenzarchitektur im Anhang 1. Berücksichtigen Sie alle erarbeiteten Applikationsschnittstellen gemäss der Aufgabenstellung 2.6.3. Verwenden Sie die im Systemumfeld beschriebene Middleware bzw. Service-Provider als Komponenten im Modell.

Begründen Sie jede Komponente und Assoziation stichwortartig.

Literaturverzeichnis

Pero, M., Kühne, S., & Fähnrich, K.-P. (2014). Integration – eine Dienstleistung mit Zukunft. In G. Schuh & V. Stich (Hrsg.), *Enterprise -Integration: Auf dem Weg zum kollaborativen Unternehmen* (S. 125–137). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-41891-4_10

Richards, M., & Ford, N. (2021). *Handbuch moderner Softwarearchitektur: Architekturstile, Patterns und Best Practices* (J. W. Lang, Übers.). Heidelberg: O'Reilly.

Rupp, C., & Queins, S. (2012). *UML 2 glasklar: Praxiswissen für die UML-Modellierung* (4., aktualisierte und erw. Aufl). München: Hanser.

Schuh, G., & Stich, V. (Hrsg.). (2014). *Enterprise -Integration*. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-41891-4

Tremp, H. (2021). *Architekturen Verteilter Softwaresysteme: SOA & Microservices - Mehrschichtenarchitekturen - Anwendungsintegration*. Wiesbaden: Springer Fachmedien. https://doi.org/10.1007/978-3-658-33179-5

Vernon, V. (2017). *Domain-Driven Design kompakt* (1. Auflage; C. Lilienthal & H. Schwentner, Übers.). Heidelberg: dpunkt.verlag.

Wolff, E. (2018). *Microservices: Grundlagen flexibler Softwarearchitekturen* (2., aktualisierte Auflage). Heidelberg: dpunkt.verlag.

Anhang 1 – Referenzarchitekturen

In den Lösungsvorschlägen sind die zugrundeliegenden Referenzarchitekturen anzugeben.

3.1 Makroarchitektur für eine Microservice-basierte Applikation

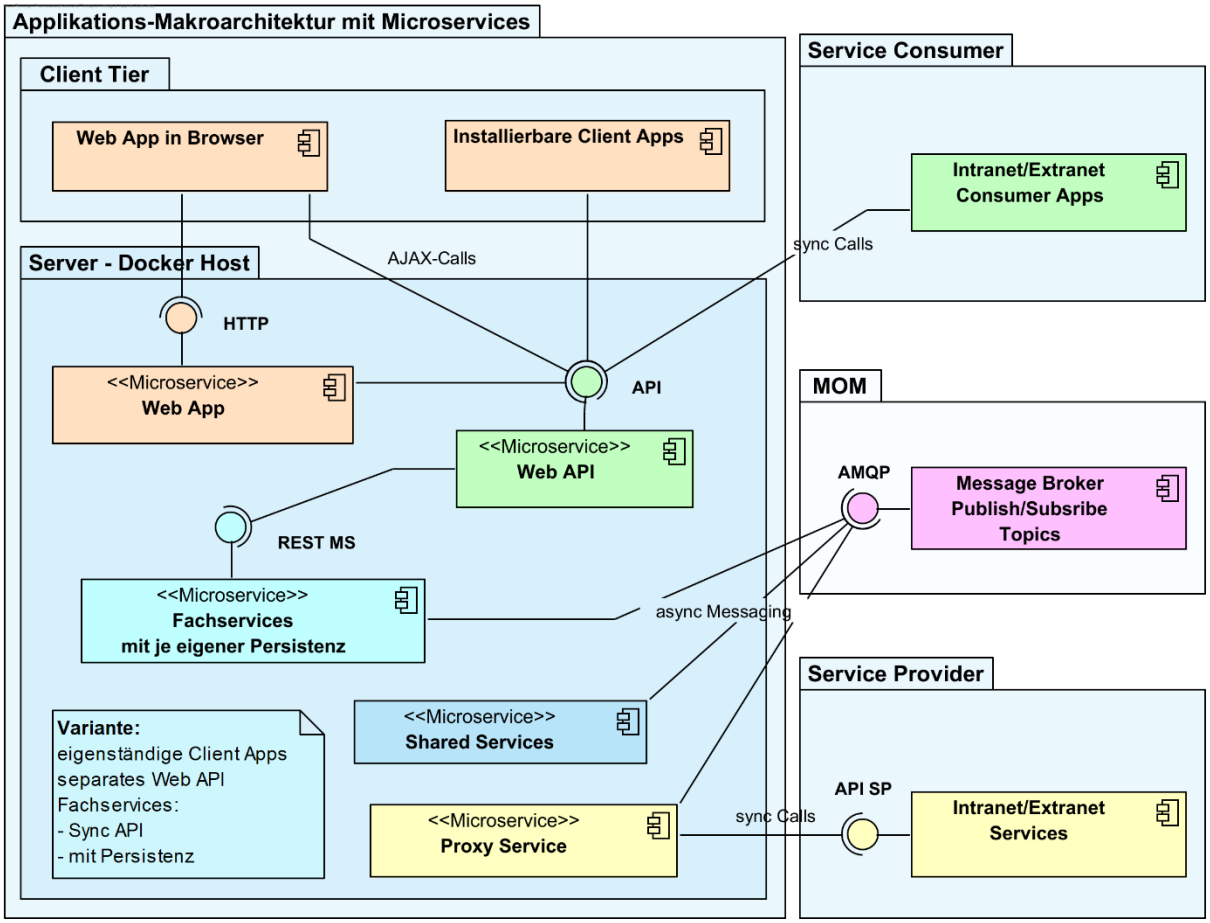


Abb. 2: Referenzarchitektur für eine Makroarchitektur

Weitere Erläuterungen ersehen Sie in Tremp (2021, S. 69-75).

3.2 Referenzarchitektur für eine skriptsprachenbasierte Mehrschichtenarchitektur

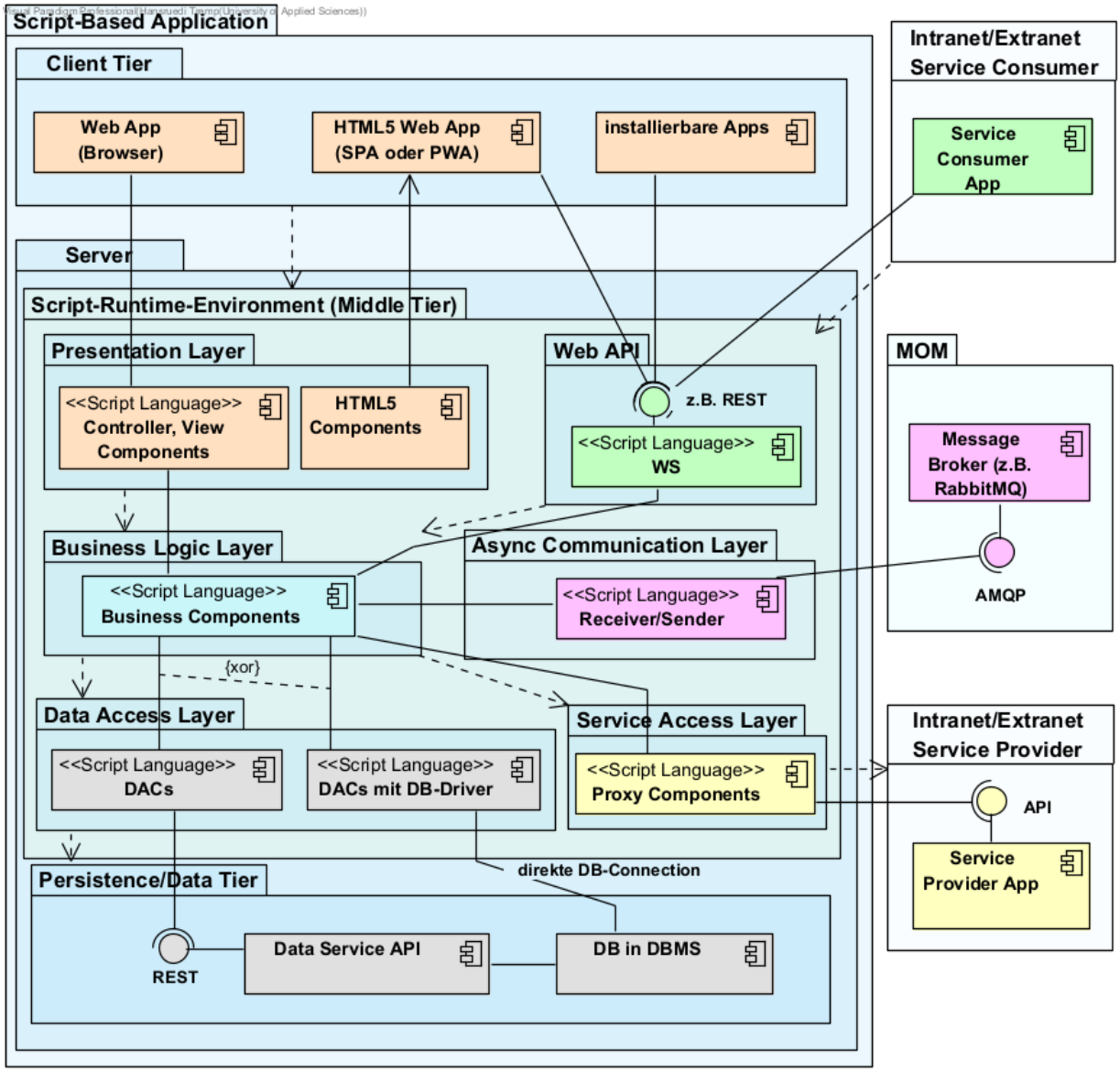


Abb. 3: Referenzarchitektur mit skriptbasierter Sprache

Weitere Erläuterungen ersehen Sie in Tremp (2021, S. 125-129).

3.3 Referenzarchitektur für eine .NET-basierte Mehrschichtenarchitektur

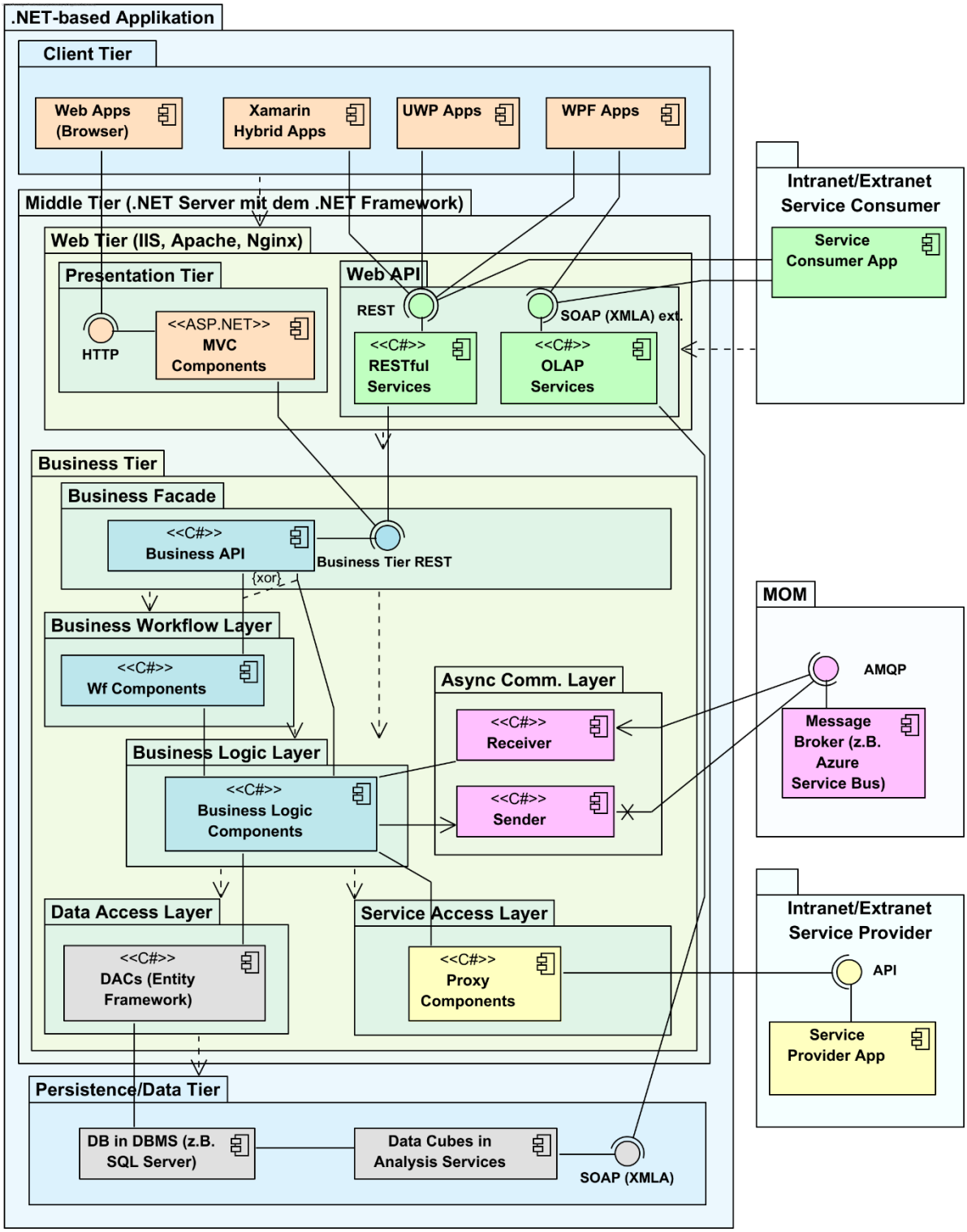


Abb. 4: Referenzarchitektur .NET

Weitere Erläuterungen ersehen Sie in Tremp (2021, S. 129-133).

3.4 Referenzarchitektur für eine Jakarta EE-basierte Mehrschichtenarchitektur

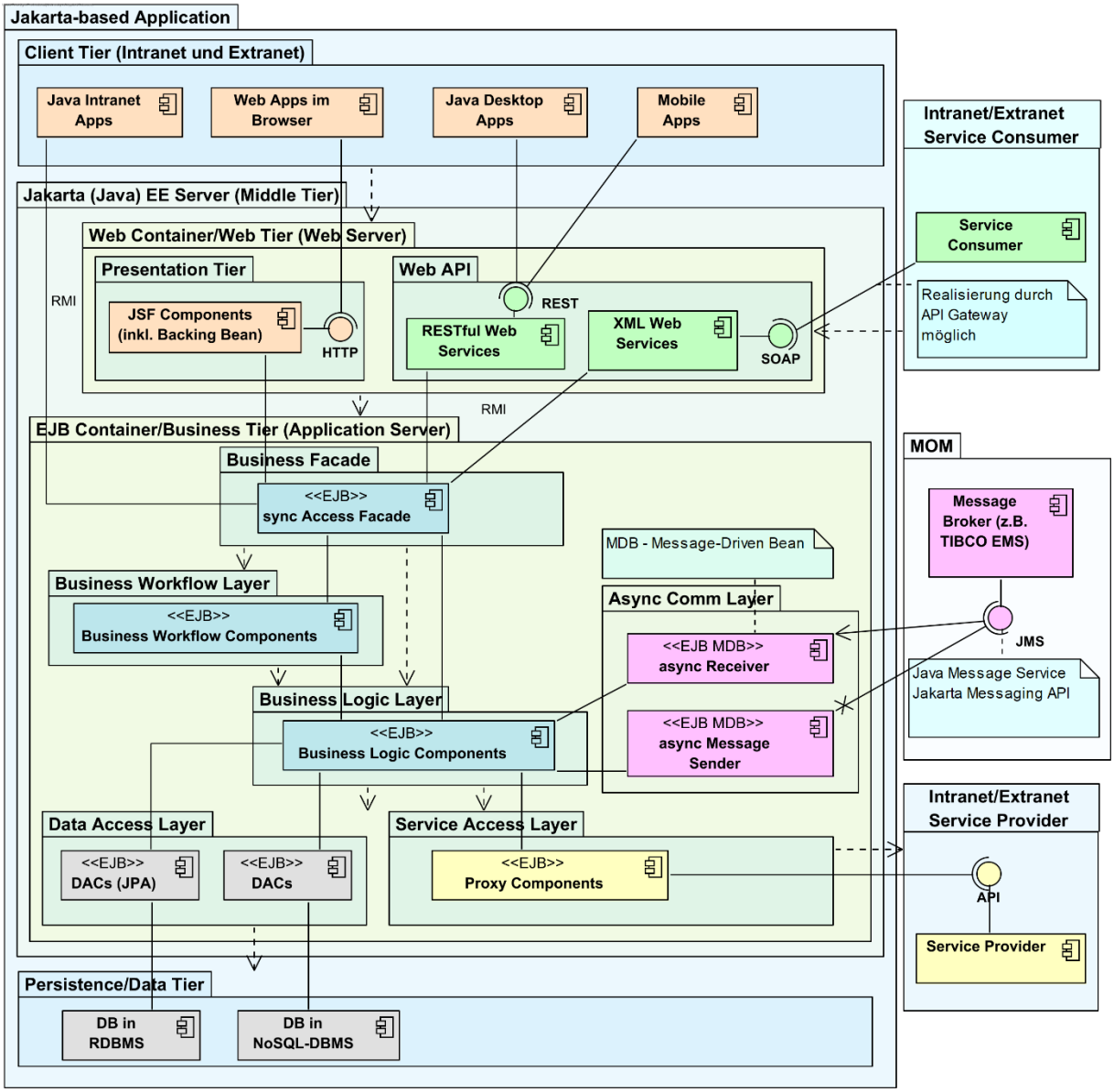


Abb. 5: Referenzarchitektur Jakarta (Java) EE

Weitere Erläuterungen ersehen Sie in Tremp (2021, S. 133-138).

3.5 Referenzarchitektur für die Integrationsarchitektur

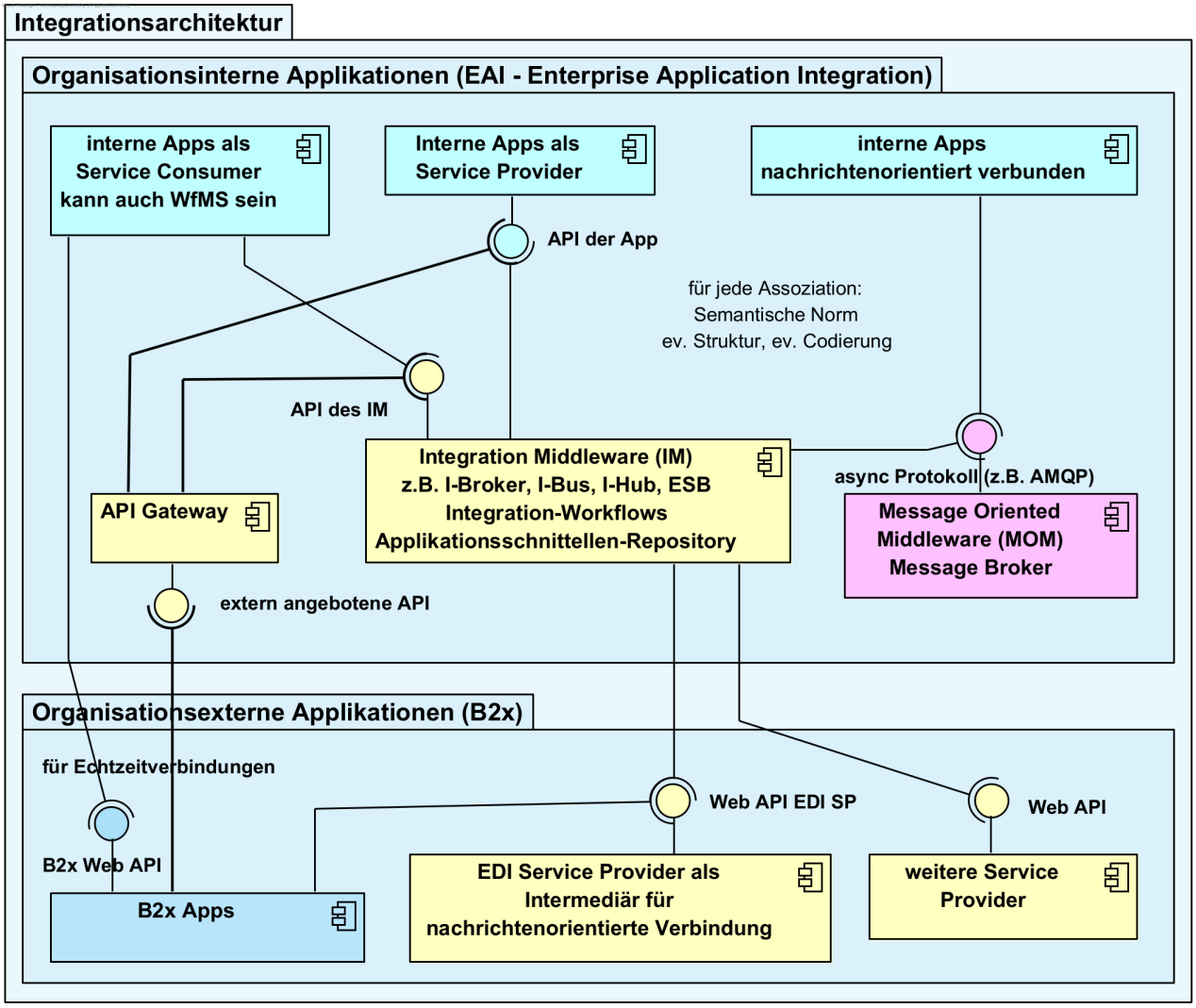


Abb. 6: Referenzarchitektur Applikationsintegration

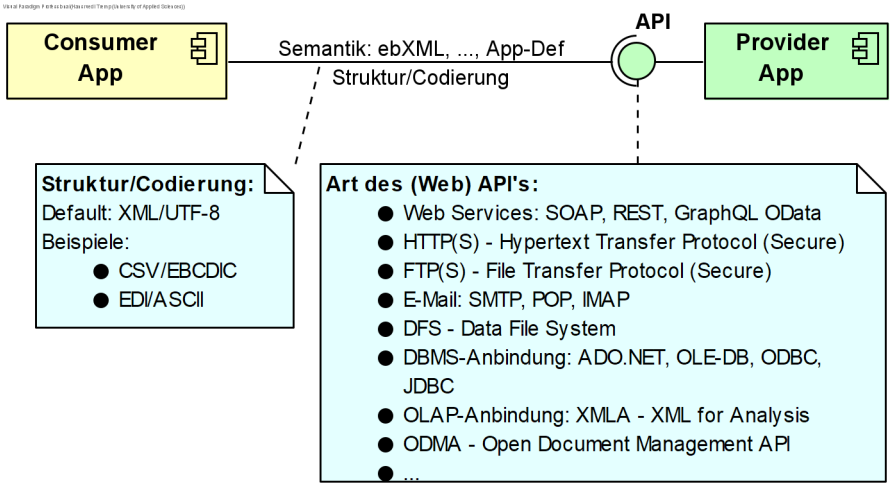


Abb. 7: Notation für die Schnittstellenbeschreibung

Weitere Erläuterungen ersehen Sie in Tremp (2021, S. 152-160).

Anhang 2 – Lösungsvorschläge

Diese Lösungsvorschläge zeigen eine gültige Lösungsvariante für die Aufgabenstellung «Case Study Timesoft» und geben Hinweise zur Bewertung. Die Natur der Aufgabenstellung lässt aber eine Vielzahl von gültigen Lösungsvarianten zu.

4.1 Fachliche Komponenten

Komponentenname	Abgedeckte Anwendungsfälle	Persistente Objekte
Login	Login vornehmen	
Zeit & Spesen	Zeit und Spesen erfassen Spesenbelege scannen Zeit & Spesen übermitteln Zeiten & Spesen kontrollieren	Zeiteinträge Speseneinträge Spesenbelege
Projekt	Projekt verwalten Lieferantenrechnungen laden Projekt auswerten Projektinformation abfragen	Projekt Stammdaten Lieferantenrechnungen Projektinfocube
Mitarbeitende	Projekt-Mitarbeitende verwalten Zeiten & Spesen weiterleiten	Projekt-Mitarbeitende Stammdaten
Abrechnung	Dienstleistungsrechnungen erstellen	Rechnungsdaten

Bewertung

Es sollten 4 - 6 sinnvoll benannte Komponenten sein.

Die Zuordnung zu den abgedeckten Anwendungsfällen muss korrekt sein, alle Anwendungsfälle müssen erwähnt werden; Doppelerwähnung sind möglich, wenn jeweils nur Teilaspekte gelöst sind.

Die persistenten Objekte müssen sachlogisch korrekt und in Bezug auf die erwähnten Objekte in den Anwendungsfällen vollständig sein.

4.2 Client Apps

Technologie	Begründung	Anwender
SPA (Single Page App) Web App	Voller Funktionsumfang für Projekt-Mitarbeitende Optimale Usability Keine Installation notwendig, Betriebssystem ist oft nicht bekannt Browserkompatibilität zu den wichtigen Internetbrowser ist sicherzustellen	Projekt-Mitarbeitende intern und extern
Mobile App	Reduzierter Funktionsumfang für mobile Geräte, erlaubt mit synchronisierten Daten offline zu arbeiten. Installierbar auf • iOS • Android	Projekt-Mitarbeitende intern und extern
Installierbare Desktop-App	Voller Funktionsumfang für die Projektleitung, optimale Usability, Geräte bzw. Betriebssysteme sind bekannt Installierbar auf • Windows • MacOS	Projektleitung
Excel-Auswertungen	Zugriff auf offene Datenschnittstelle: OData-API	Projektleitung
(klassische) Extranet Web App	Geringe Anforderungen an die Browserkompatibilität Projektauswertung	externe Projekt-Mitarbeitende

Bewertung

Es sind 2 Web-Apps zu erwähnen:

- klassische mit geringen Anforderungen an die Browser-Kompatibilität gem. Hinweis in Anwendungsfall Projektinformation abfragen
- SPA mit optimaler Usability gem. Hinweis im Anwendungsfall Zeit & Spesen erfassen

Weiter müssen 2 installierbare Apps aufgeführt sein:

- Desktop App für Windows und macOS gem. Hinweis im Anwendungsfall Projekt verwalten
- Mobile App für iOS und Android gem. Hinweis im Anwendungsfall Zeit & Spesen erfassen sowie Spesenbelege scannen

Die Excel-Auswertungen via OData-Feed ist im Anwendungsfall Projekt auswerten erwähnt.

4.3 Identifikation der Applikationsschnittstellen

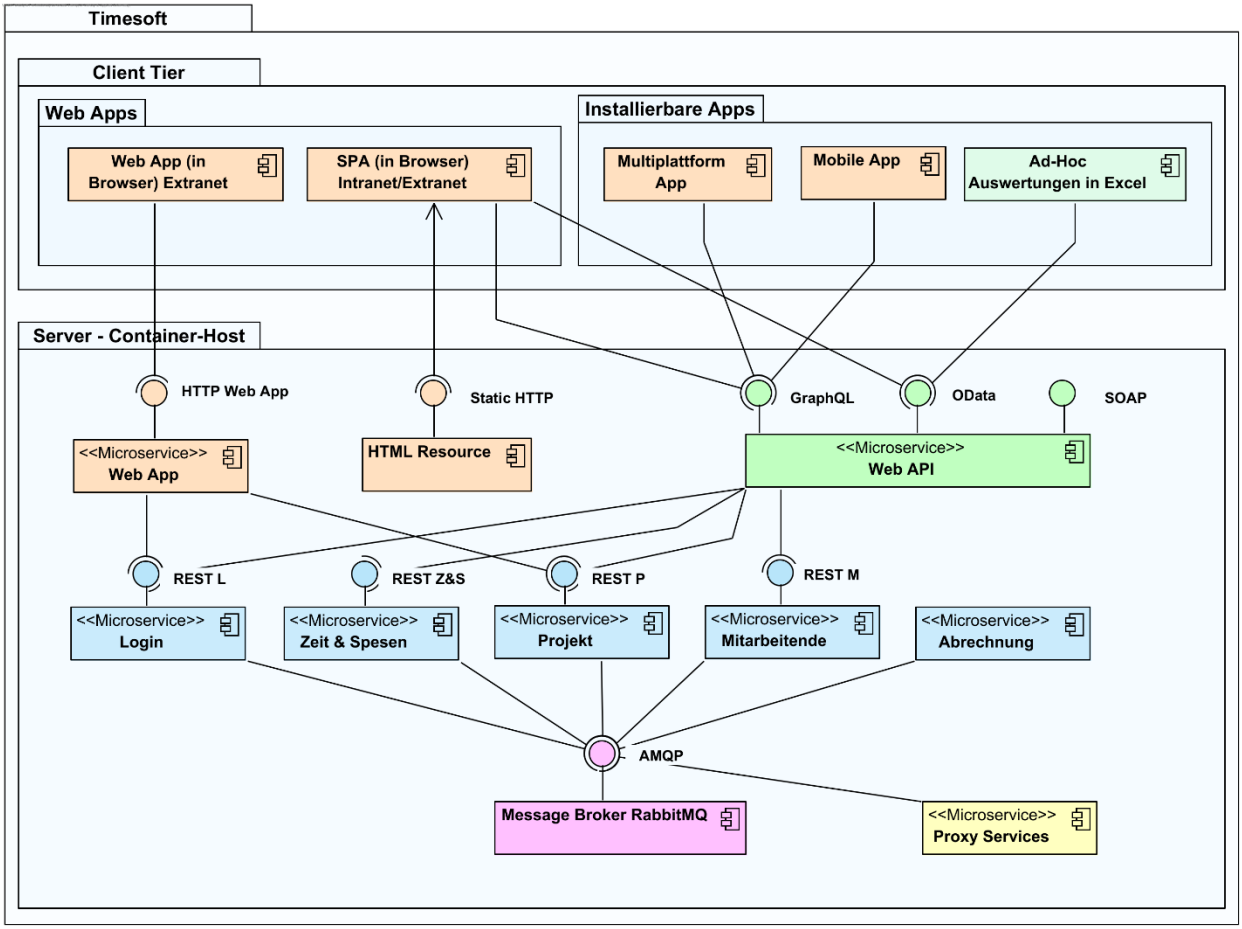
Applikation	sync/async	Funktion	Protokoll	Anwendungsfall
Identity-Provider	sync	Service Provider	AD LDS	Login vornehmen
System der Sub-unternehmen	sync	Service Consumer	SOAP	Zeit und Spesen erfassen
Kundensystem via EDI-Provider	sync	Service Provider	SOAP (UBL)	Dienstleistungsrechnungen erstellen
Debitoren-Buha	sync	Service Provider	gRPC	Dienstleistungsrechnungen erstellen
Mailsystem	sync	Service Provider	SMTP	Dienstleistungsrechnungen erstellen
Kreditoren-Buha	sync	Service Provider	REST	Lieferantenrechnungen laden
Excel	sync	Service Provider	OData	Projekt auswerten
Lohn-Buha	async	Receiver	AMQP (IDoc)	Zeiten und Spesen weiterleiten
Scheduler	async	Sender	AMQP	Dienstleistungsrechnungen erstellen; Zeiten & Spesen weiterleiten

Bewertung

Alle 10 Schnittstellen lassen sich aus den angegebenen Anwendungsfällen ableiten.

Für durch einen Zeittrigger gesteuerten Anwendungsfälle sind auch andere Lösungsansätze möglich. Sie müssen aber das vorgegebene Systemumfeld berücksichtigen.

4.4 Makroarchitektur für Variante Microservices



Bewertungsraster

Der Technologie-Stack für die einzelnen Microservices sowie die einzusetzenden Frameworks für die Client-Apps bleibt offen.

Die clientseitigen Apps gemäss 4.2 sind vollständig vorhanden und mit den korrekten Assoziationen mit dem Server verbunden.

Das Web API muss die drei erwähnten APIs aufweisen: GraphQL (siehe ICT-Strategie), OData (siehe Anwendungsfall Projekt auswerten) und SOAP (siehe Anwendungsfall Zeit & Spesen übermitteln).

Die in 4.1 eruierten fachlichen Komponenten müssen hier als je ein Microservice erscheinen.

Ausser Abrechnung soll jeder Microservice ein REST-Interface anbieten, welches aus der darüberliegenden Ebene aufgerufen wird. Alternativ wäre die async Kommunikation zwischen allen Microservices denkbar.

Die Proxy Services sind in einen eigenen Microservice gekapselt.

Der Message Broker ist mit der korrekten Schnittstelle (in diesem Falle AMQP) eingezeichnet.

Für den Server wird eine Betriebsumgebung für die Containerisierung vorgeschlagen (Docker-Host, Kubernetes).

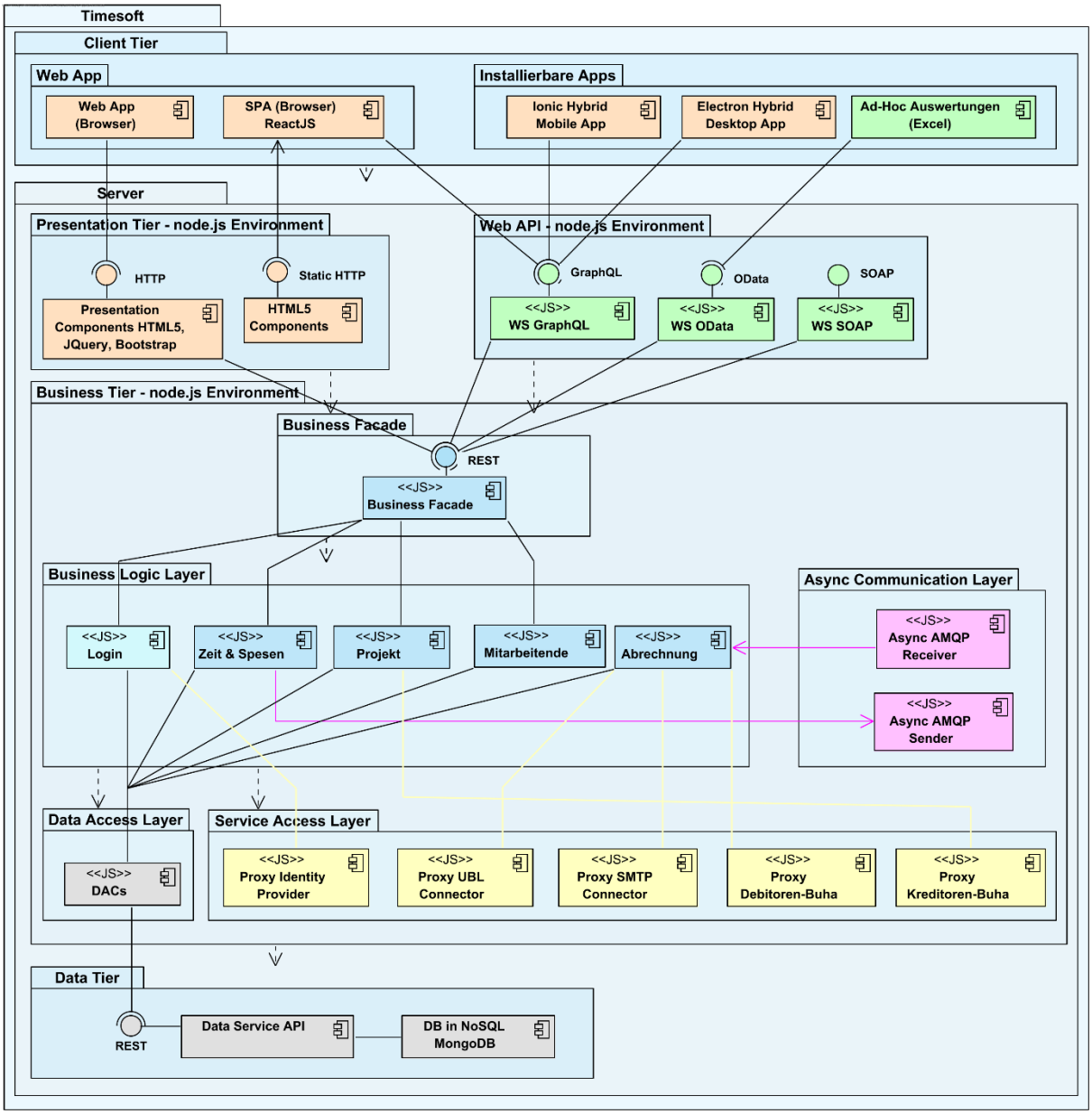
Jedes Paket und jede Komponente ist kurz begründet.

4.5 Variante monolithische Mehrschichtenarchitektur

Aus den drei möglichen Varianten (Skriptbasiert, .NET oder Jakarta [Java] EE) ist eine gewählt und die Wahl sinnvoll begründet.

Stichwortartig sind die Vor- bzw. Nachteile des gewählten Technologie-Stacks ausgeführt. Siehe mögliche Antworten bei den Lösungsvorschlägen der jeweiligen Varianten.

4.5.1 Lösungsvorschlag für Variante Skriptbasiert



Bewertungsraster

Die Architektur ist grundsätzlich an der passenden Referenzarchitektur ausgerichtet und wendet diese korrekt an.

Die clientseitigen Apps gemäss 4.2 sind vollständig vorhanden und mit den korrekten Assoziationen mit dem Server verbunden. Die gewählte Technologie soll in Übereinstimmung mit dem Grundsatz sein, dass JavaScript durchgängig als Programmiersprache verwendet wird:

- Einfache Web App: HTML5 Framework
- SPA: entsprechendes HTML5 Framework wie z.B. ReactJS, EmberJS, AngularJS usw.
- Mobile App: Hybrid-Ansatz mit Frameworks wie z.B. Ionic, React Native usw.
- Desktop App: ebenfalls einen Hybrid-Ansatz wie z.B. Electron usw.
- Excel für die Ad-Hoc Auswertungen, verbunden mit OData

Das Web API muss die drei erwähnten APIs aufweisen: GraphQL (siehe ICT-Strategie), OData (siehe Anwendungsfall Projekt auswerten) und SOAP (siehe Anwendungsfall Zeit & Spesen übermitteln).

Da die Vorgabe für die Skalierung zwingend mind. 2 Tiers für den Middle Tier vorgibt, sind die 3 Tiers vorzusehen, welche unabhängig voneinander installiert und betrieben werden können:

- Presentation Tier
- Web API (oder Web Service Tier)
- Business Tier

Der Zugriff auf den Business Tier ist zwingend mit einem Web Service zu lösen. Präferenziert wird REST, gRPC wäre auch möglich. Der Business Tier muss eine Facade aufweisen, welche den Service anbietet.

Die in 4.1 eruierten fachlichen Komponenten müssen hier als je eine Komponente erscheinen.

Alle Komponenten sind mit dem Stereotype <<JS>> auszuzeichnen, um aufzuzeigen, dass es sich um JavaScript-Programme handelt.

Für die Verbindung zum Message-Broker sind in einem eigenen Layer Receiver und Sender-Komponente darzulegen.

Für den Zugriff auf die angegebenen Service-Provider (gemäss 4.3) sind je eine Proxy-Komponente mit den passenden Assoziationen vorzusehen.

Um die durchgängige Nutzung von JavaScript zu ermöglichen, ist für die Persistenz eine NoSQL-DB vorzuschlagen, damit die JSON-Objekte direkt gespeichert werden können. Die Data Service API ist nicht unbedingt als eigene Komponente einzuzeichnen. Den Data Access Layer mit den DACs (Data Access Components) kann bei entsprechender Begründung auch weggelassen werden, da ja kein ORM (Object Relational Mapping) erfolgt.

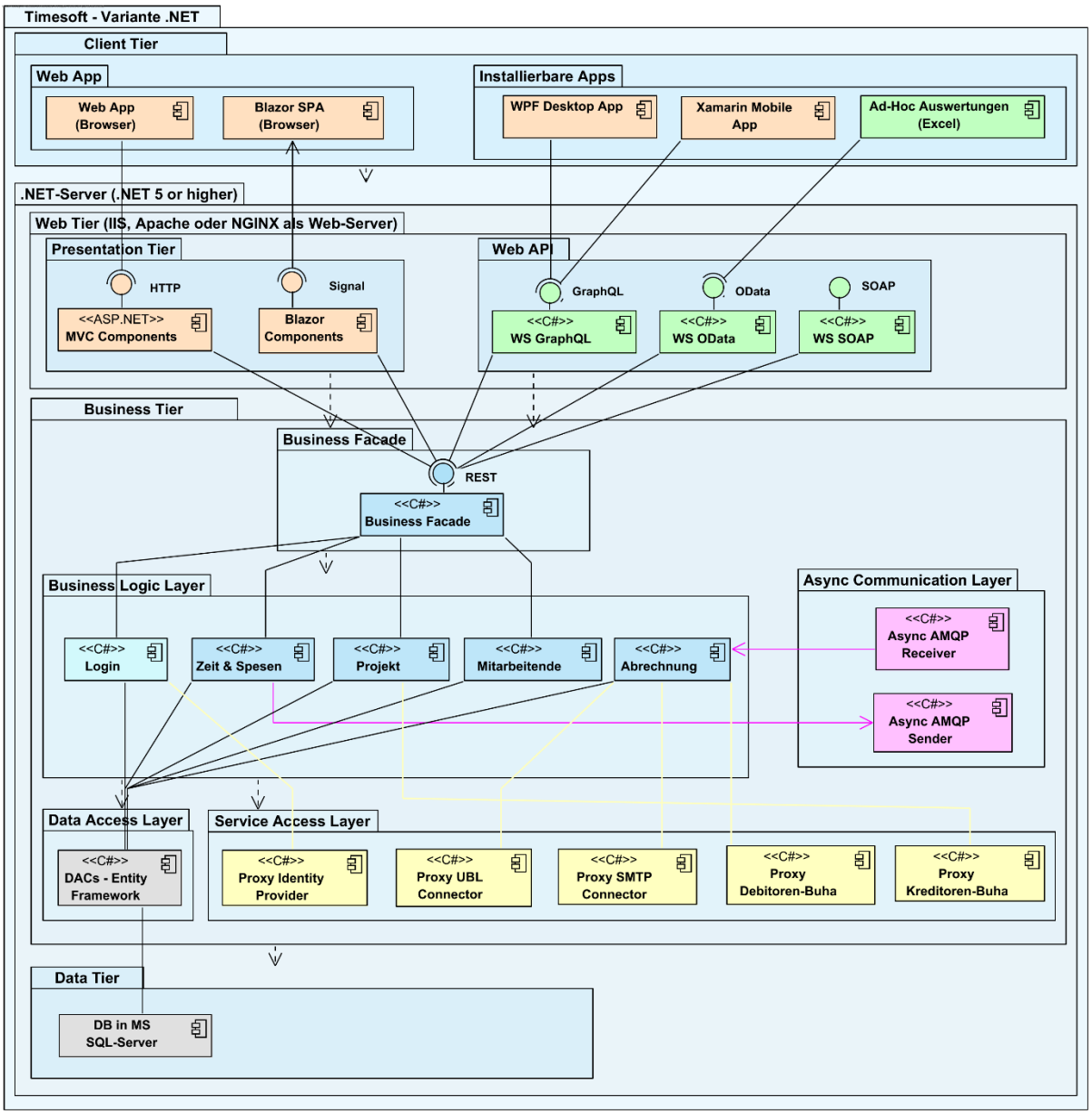
Begründung für Node.js

Node.js erlaubt die durchgängige Verwendung von JavaScript als Programmiersprache. Somit können Softwareentwickler der Front-Seite auch hier den Code implementieren, ohne eine neue Sprache zu lernen.

Node.js lässt sich einfacher erlernen und begnügt sich mit einer schlanken Server-Infrastruktur.

Node.js ist gegenüber den anderen Frameworks wie z.B. .NET bzw. Jakarta EE weniger performant und lässt sich auch schlechter skalieren.

4.5.2 Lösungsvorschlag für Variante .NET



Bewertungsraster

Die Architektur ist grundsätzlich an der passenden Referenzarchitektur ausgerichtet und wendet diese korrekt an.

Die clientseitigen Apps gemäss 4.2 sind vollständig vorhanden und mit den korrekten Assoziationen mit dem Server verbunden. Die gewählte Technologie soll in Übereinstimmung mit dem Grundsatz sein, dass .NET mit C# durchgängig als Programmiersprache verwendet wird:

- Einfache Web App: ASP.NET
- SPA: clientseitige Blazor-Technologie (HTML mit C#)
- Mobile App: Hybrid-Ansatz mit Xamarin

- Desktop App: hier sollte WPF (Windows Presentation Foundation) verwendet werden; UWP (Universal Windows Platform) geht nicht, da macOS nicht unterstützt wird, möglich wäre die neueste Entwicklung mit .NET MAUI
- Excel für die Ad-Hoc Auswertungen, verbunden mit OData

Das Web API muss die drei erwähnten APIs aufweisen: GraphQL (siehe ICT-Strategie), OData (siehe Anwendungsfall Projekt auswerten) und SOAP (siehe Anwendungsfall Zeit & Spesen übermitteln).

Da die Vorgabe für die Skalierung zwingend mind. 2 Tiers für den Middle Tier vorgibt, sind die 3 Tiers vorzusehen, welche unabhängig voneinander installiert und betrieben werden können:

- Presentation Tier
- Web API (oder Web Service Tier)
- Business Tier

Der Zugriff auf den Business Tier ist zwingend mit einem Web Service zu lösen. Präferenziert wird REST, gRPC wäre auch möglich. Der Business Tier muss eine Facade aufweisen, welche den Service anbietet.

Die in 4.1 eruierten fachlichen Komponenten müssen hier als je eine Komponente erscheinen.

Alle Komponenten sind mit dem Stereotype <<C#>> auszuzeichnen, um aufzuzeigen, dass es sich um C#-Programme handelt.

Für die Verbindung zum Message-Broker sind in einem eigenen Layer Receiver und Sender-Komponente darzulegen.

Für den Zugriff auf die angegebenen Service-Provider (gemäss 4.3) sind je eine Proxy-Komponente mit den passenden Assoziationen vorzusehen.

Die Persistenz lässt sich gut in einer relationalen DB im MS SQL-Server abbilden. Der Zugriff ist zwingend über den Data Access Layer mit DACs unter Verwendung des .NET Entity Frameworks für das ORM zu realisieren.

Begründung für .NET

.NET ist ein stabiler und performanter Technologie-Stack.

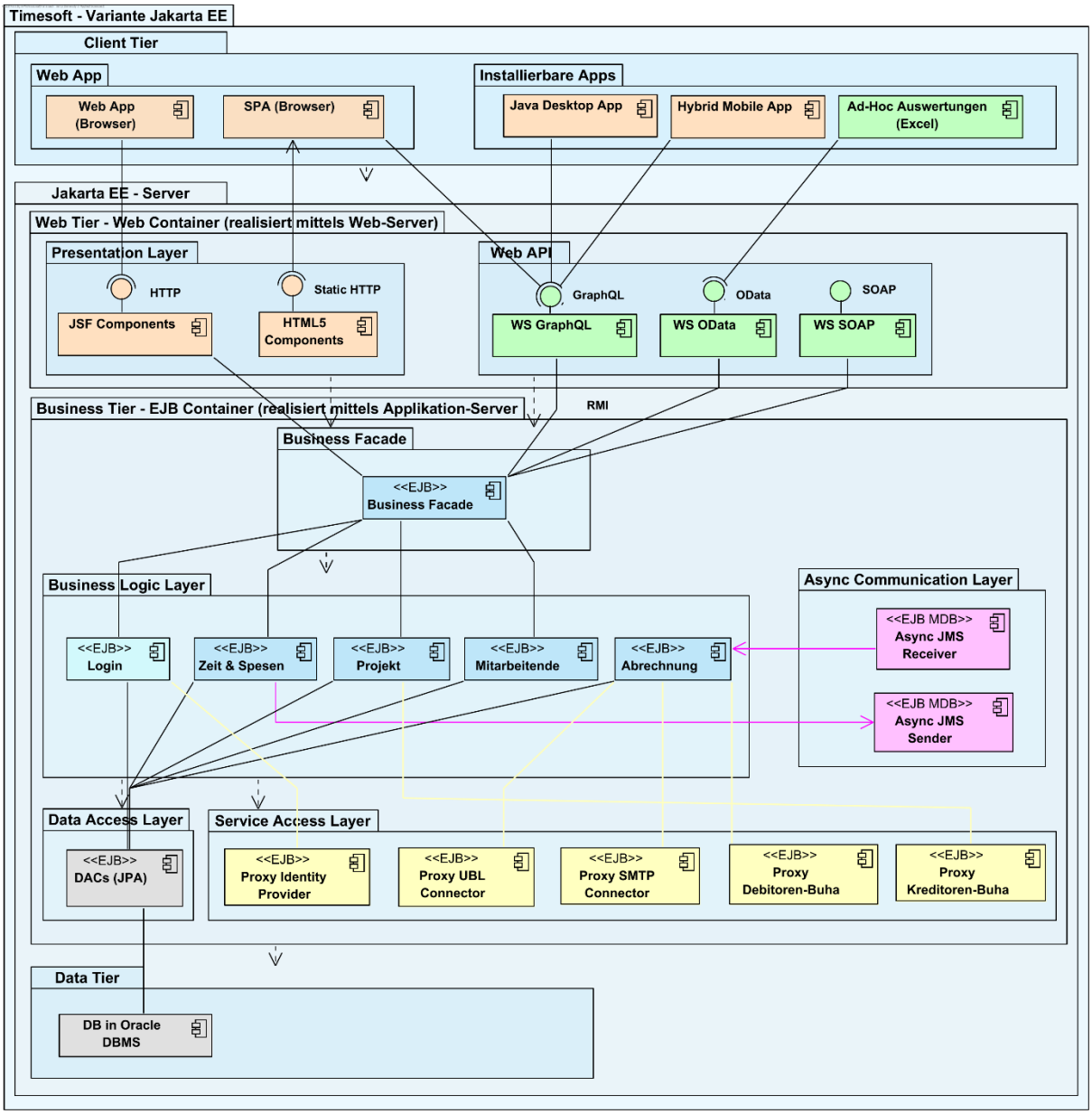
.NET weist eine hohe Verbreitung auf

C# als primäre Programmiersprache kann durchgängig verwendet werden.

Serverseitig auf Windows oder Linux limitiert

Benötigt verhältnismässig viel Ressourcen

4.5.3 Lösungsvorschlag Variante Jakarta EE-basiert



Bewertungsraster

Die Architektur ist grundsätzlich an der passenden Referenzarchitektur ausgerichtet und wendet diese korrekt an.

Anstelle von Java EE ist die aktuelle Entwicklung Jakarta EE zu verwenden.

Die clientseitigen Apps gemäss 4.2 sind vollständig vorhanden und mit den korrekten Assoziationen mit dem Server verbunden. Die gewählte Technologie soll in Übereinstimmung mit dem Grundsatz sein, dass Java durchgängig als Programmiersprache verwendet wird:

- Einfache Web App: JSF
- SPA: Java-seitig gibt es aktuell keine SPA-Frameworks, deshalb Verwendung von den besten HTML5 Frameworks wie z.B. ReactJS, EmberJS, AngularJS usw.

- Mobile App: auch hier gibt es aktuell keine Java-basierten Frameworks, deshalb Verwendung der besten Frameworks für Hybrid-Apps wie z.B. Ionic, React Native usw.
- Desktop App: hier sollte Java verwendet werden
- Excel für die Ad-Hoc Auswertungen, verbunden mit OData

Das Web API muss die drei erwähnten APIs aufweisen: GraphQL (siehe ICT-Strategie), OData (siehe Anwendungsfall Projekt auswerten) und SOAP (siehe Anwendungsfall Zeit & Spesen übermitteln).

In Jakarta EE sind die beiden Tiers vorgegeben:

- Web Tier oder Web Container - Hinweis auf Web Server für die Realisierung
    - Presentation Layer für die Web Apps
    - Web API (oder Web Service Layer) für die APIs
- Business Tier oder EJB (Enterprise Java Beans) Container, Realisierung mittels eines Application Servers

Der Zugriff auf den Business Tier ist zwingend über eine Business Facade zu lösen. Der Zugriff erfolgt mittels RMI, d.h. hier ist keine Web Service notwendig.

Die in 4.1 eruierten fachlichen Komponenten müssen hier als je eine Komponente erscheinen.

Alle Session Beans im EJB Container sind mit dem Stereotype <<EJB>> auszuzeichnen, um aufzuzeigen, dass es sich um Java-Klassen handelt.

Für die Verbindung zum Message-Broker sind in einem eigenen Layer Receiver und Sender-Komponente darzulegen, welche Message Driven Beans sind und somit den Stereotype <<EJB MDB>> aufweisen müssen.

Für den Zugriff auf die angegebenen Service-Provider (gemäss 4.3) sind je eine Proxy-Komponente mit den passenden Assoziationen vorzusehen.

Die Persistenz lässt sich gut in einer relationalen DB wie z.B. Oracle DB, MySQL usw. abbilden. Der Zugriff ist zwingend über den Data Access Layer mit DACs unter Verwendung von JPA (Java Persistence API) für das ORM zu realisieren.

Begründung für Jakarta EE

Von Java bzw. Java EE her ist es eine bewährte Technologie mit einer weltweiten Anwenderbasis.

Aktuelle Weiterentwicklung mit einer ansehnlichen Community von Jakarta EE unter der Eclipse Foundation.

Jakarta EE ist Open-Source.

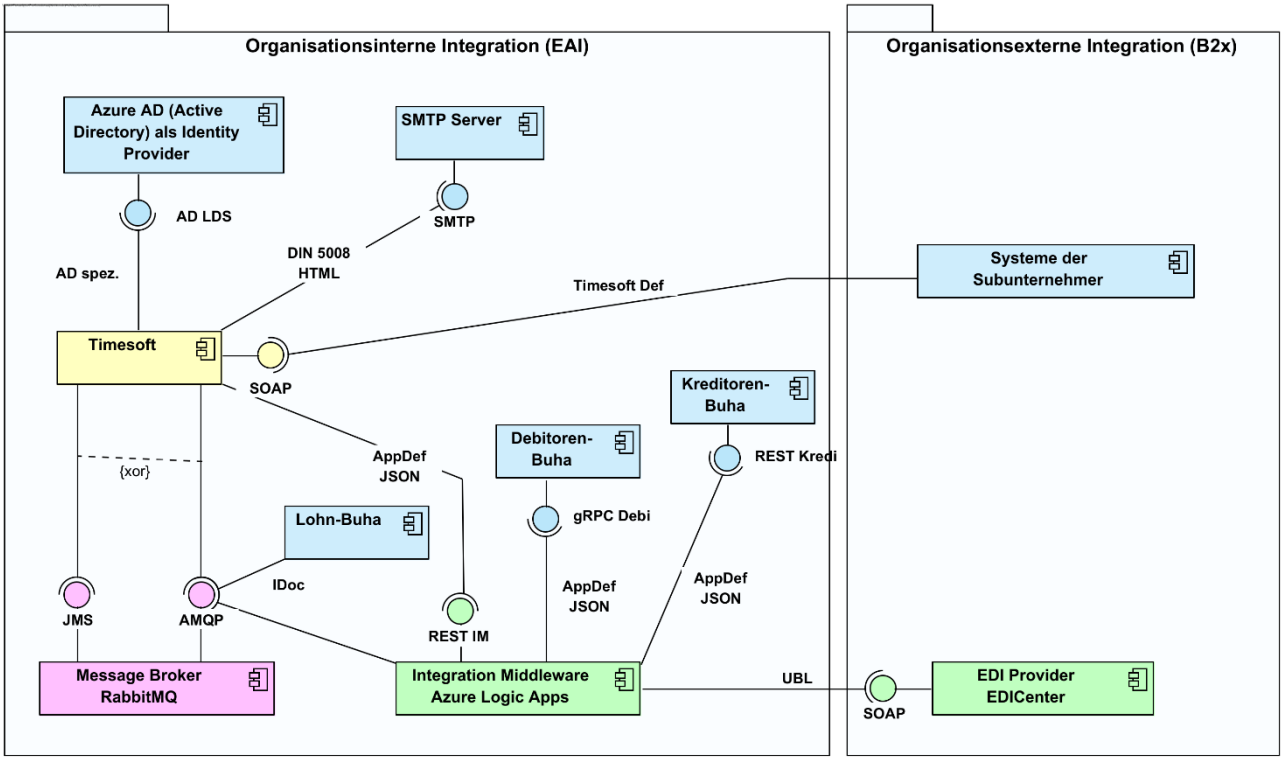
Es lässt sich gut skalieren.

Durchgehend (ausser SPA und Mobile Apps) kann mit Java programmiert werden.

Serverseitig ist die Installation mit Web- und Applicationserver komplexer.

Es ist der richtige Mix von Tools aus unterschiedlichen Quellen zu finden. Die Abhängigkeiten sind zum Teil schwierig unter Kontrolle zu halten.

4.6 Integrationsarchitektur



Bewertungsraster

Die Architektur ist grundsätzlich an der passenden Referenzarchitektur ausgerichtet und wendet diese korrekt an.

Jede Komponente und Assoziation ist auf die Aufgabenstellung hin stichwortartig zu begründen.

Timesoft bildet hier eine Komponente, welche gegen aussen nur die SOAP-Schnittstelle anbietet.

Folgende Applikationen lassen sich nebst den Anforderungen aus dem Systemumfeld (2.5) ableiten:

- Identity Provider Azure AD
- Message Broker RabbitMQ
- SMTP Mail Server
- Integration Middleware Azure Logic Apps
- EDI Provider EDICenter

Die dargelegten APIs müssen mit der Auflistung gemäss Aufgabenstellung in 2.6.3 übereinstimmen.

Die Verbindungen sind korrekt gemäss Abb. 7: Notation für die Schnittstellenbeschreibung anzugeben.