Bern University
of Applied Sciences

Software Engineering and Design

Software Architecture

Mental Health Care Patient Management System (MHC-PMS)

Team White:
Dellsperger Jan
Ellenberger Roger
Sheppard David
Sidler Matthias
Spring Mathias
Thöni Stefan

May 13, 2016

Version 1.0

1 Übersicht

1.1 Systemrequirements

Dieser Abschnitt beschreibt, welche Requirements im Projekt umgesetzt werden und wie die Umsetzung aussieht (siehe Dokument Requirements-Specification aus Task 4). Wir konzentrieren uns auf die Functional Requirements.

1.1.1 Darstellung Dashboard

Die Darstellung des Dashboards wird im Presentation Layer realisiert.

1.1.2 Konfiguration Dashboard

Der veränderbare Teil der Daten beschränkt sich auf die benutzerspezifischen Konfigurationen des Dashboards. Damit die **Datenpersistenz** gewährleistet wird, wird ein Container benötigt, welcher diese Daten abspeichert. Die Benutzerkonfiguration legen wir daher in einer Datenbank ab. Es ist angedacht, die Datenbank auf dem gleichen Server zu betreiben, der auch die Datenquelle fürs Reporting enthält.

1.1.3 Reporting anzeigen

Die Anzeige der Reports wird im Presentation Layer realisiert.

1.1.4 Schnittstelle DBS

Die Daten, welche fürs Reporting verwendet werden, sind nicht in Verantwortung der Applikation und es wird bloss lesend darauf zugegriffen. Die Anbindung wird im Data Access Layer umgesetzt.

1.1.5 Kennzahlen

Die Quelldaten fürs die Berechnung der Kennzahlen, werden aus einer externen Datenquelle geholt (Siehe Schnittstelle DBS).

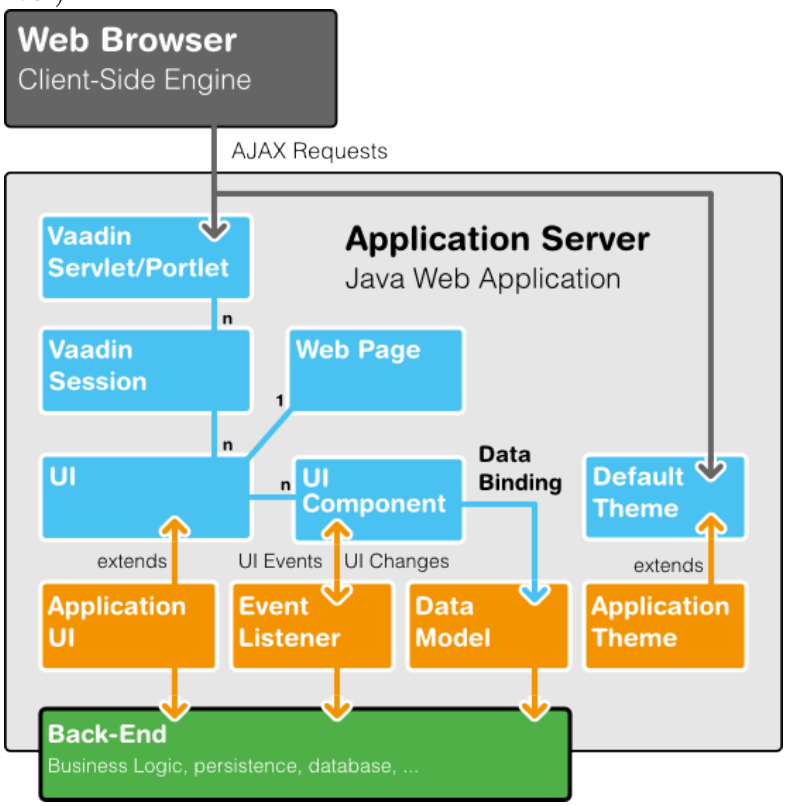
2 Architektur

Layering Pattern Für die Gesamtarchitektur verwenden wir das Layering Pattern. Die Applikation wird aufgeteilt in die Layer *Presentation*, *Business Logic* und *Data Access*. Zudem existiert der Layer *Model*. Dieser zieht sich über alle Layers hinweg und enthält die Model-Logik aus dem MVC-Pattern.

MVP Pattern Die Grafischen Komponenten strukturieren wir nach dem MVP-Pattern.

Client Server Pattern Die Applikation basiert zudem auf dem Client-Server Pattern. Clientseitige Zugriff geschieht via Web-Browser. Die dazu benötigte Logik stellt das Framework Vaadin bereit.

Als DBS verwenden wir einen MySQL-Server (bzw. den kompatiblen MariaDB-Server).



[1]

[1] https://vaadin.com

2.1 Presentation Layer

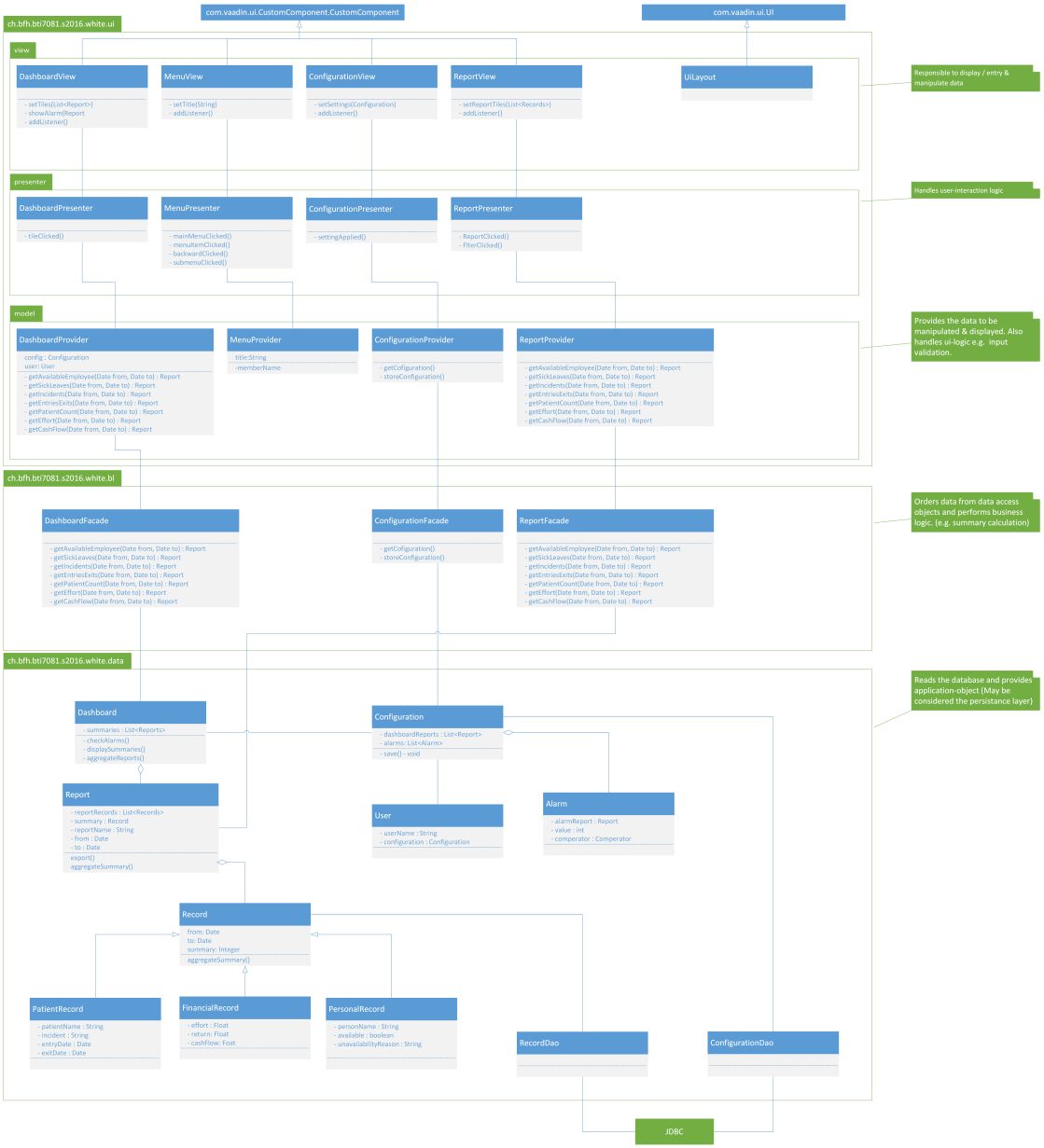
Für die Bereitstellung der grafische Oberfläche nutzen wir Vaadin. In diesem Layer verwenden wir das MVP-Pattern zur Umsetzung der Grafischen Oberfläche.

2.2 Business Logic Layer

Im Business Logic Layer werden die Daten aggregiert und fürs Reporting aufbereitet.

2.3 Data Access Layer

Dieser Layer realisiert die Anbindung an das Datenbank-Backend. Die Daten aus der Datenbank werden in entsprechenden Klassen abgebildet. Wir verwenden JDBC um die Verbindung auf das Datenbanksystem (DBS) herzustellen. Als DBS verwenden wir einen MySQL-Server (bzw. den kompatiblen MariaDB-Server). Das System kann aufgrund der Layering-Architektur auch mit relativ wenig Aufwand ausgetauscht werden.



Die Architektur ist durch das Layering sehr flexibel aufgebaut. Die Komponenten können mit relativ wenig Aufwand ausgetausch werden. Zudem erlaubt es die Implementationsarbeiten unter den Teammitgliedern aufzuteilen, ohne dass die einzelnen Arbeiten in Konflikt geraten. Das Erweitern der Funktionalitäten für zukünftige Versionen wird ebenfalls vereinfacht durch das Layering-Modell.