|  |
| --- |
|  |
| Bericht-Task03  **Gruppe White**  **Autoren: Viktor Velkov, Marko Miletic, David Gaupp, Alain Nippel, Janahan Sellathurai, Sugeelan Selvasingham**  **Version 1.0, 14.11.2019** |
| **Berner Fachhochschule**  Technik und Informatik  Medizininformatik |

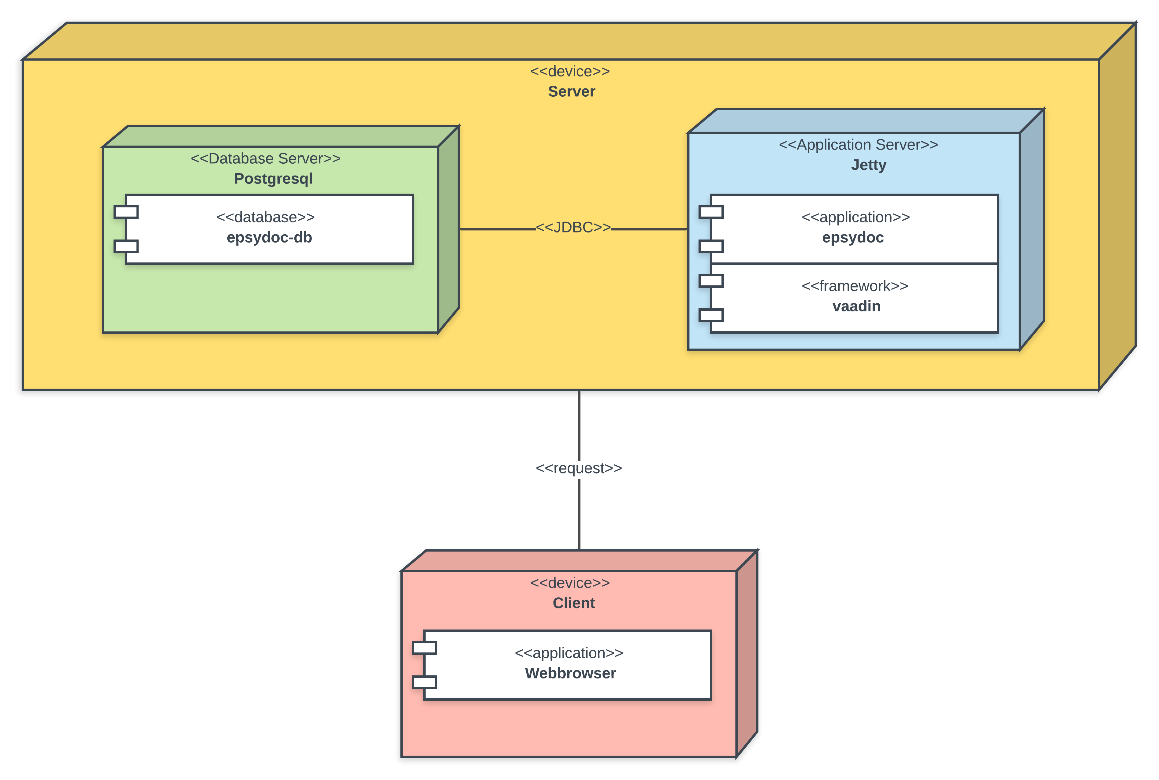
Inhaltsverzeichnis

[1 UML Deployment Diagram 3](#_Toc25231229)

[2 UML Component Diagram 4](#_Toc25231230)

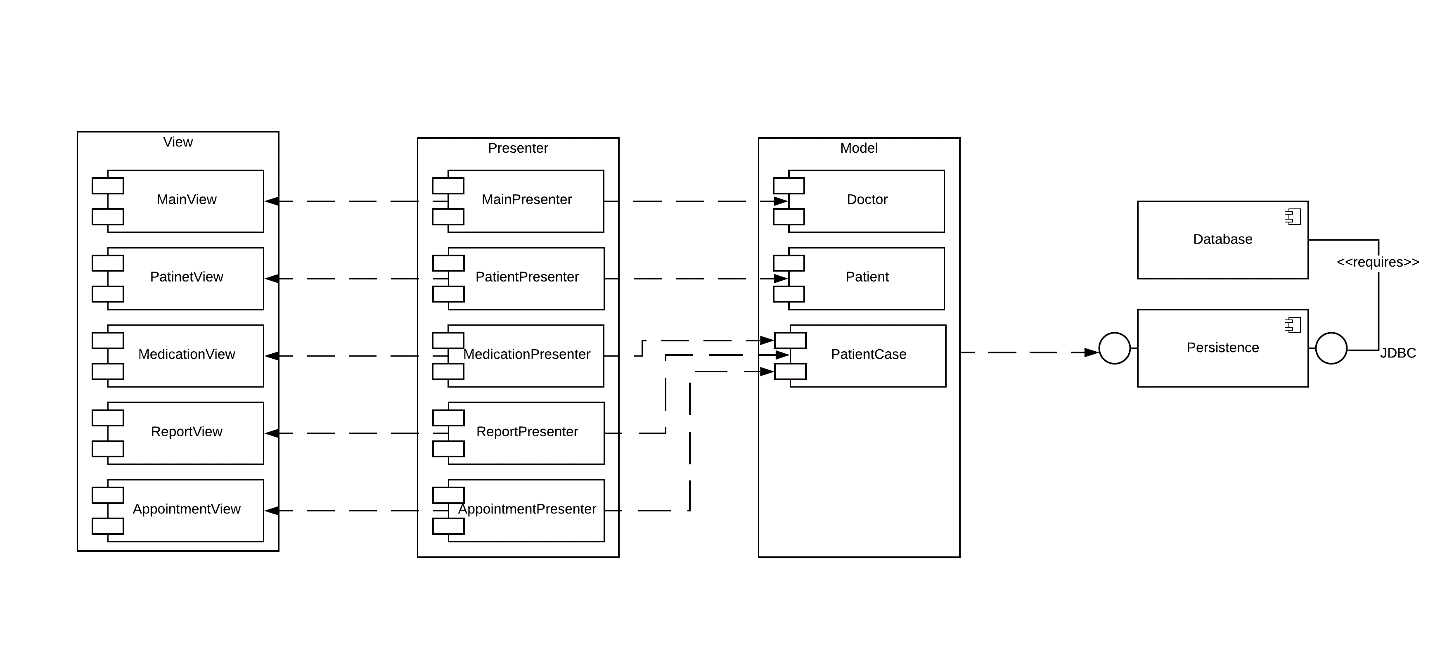
[3 UML Class Diagram of main view 5](#_Toc25231231)

# UML Deployment Diagram



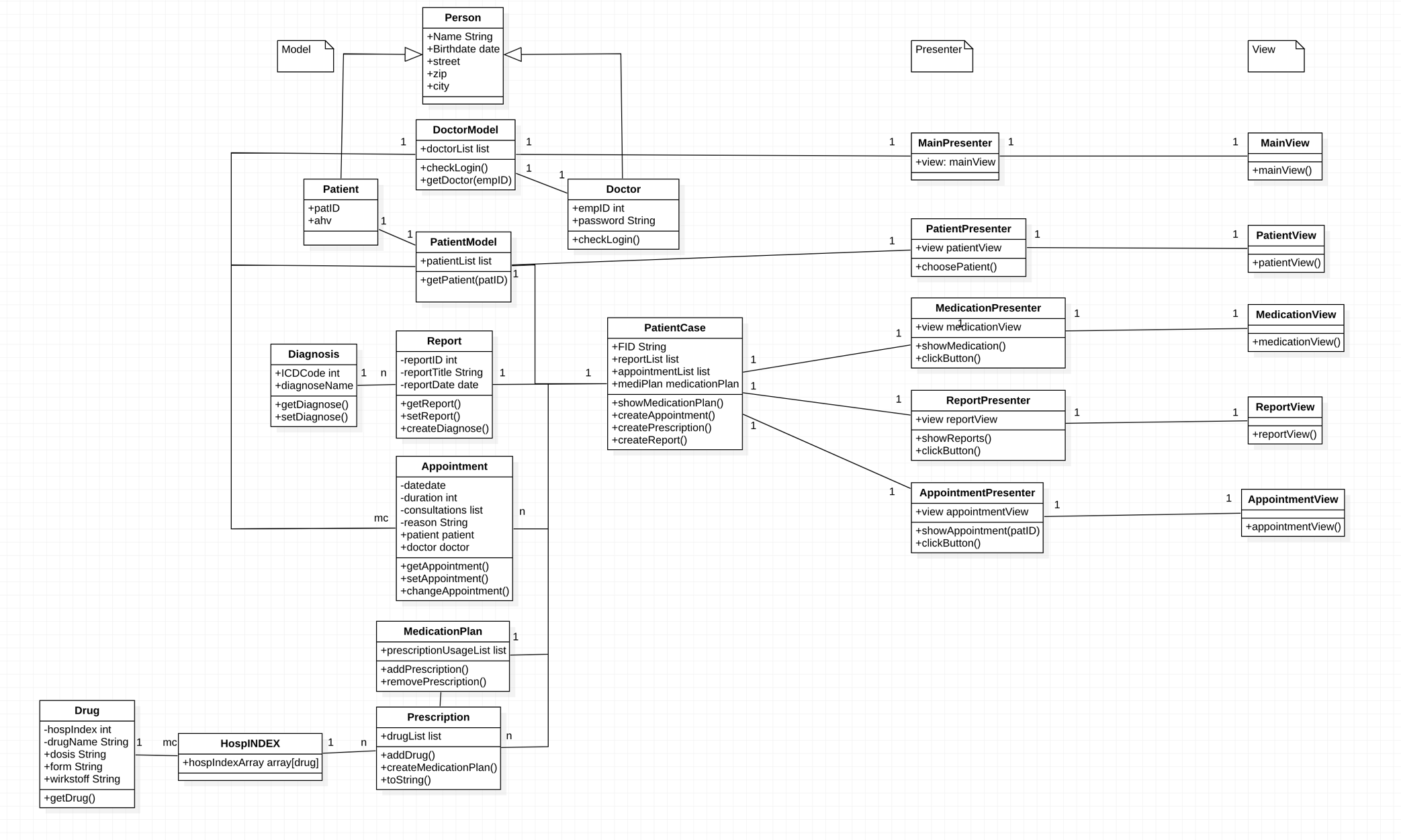
Deployment Diagramme werden verwendet, um die Hardware Knoten/Geräte eines Systems sowie die Verbindungen der Kommunikation zwischen ihnen darzustellen. Anhand des obigen Deployment Diagram erkennt man, dass wir Knoten in Form von Devices haben. Diese sind bei uns in gelb die Server allgemein. Verschachtelt in den Serverknoten ist der Datenbank-Server (Postgresql) sowie der Applikations-Server (Jetty). Die Applikation welche verschachtelt im Applikkations-Server ist, stellt unsere Applikation dar die wir jetzt epsydoc genannt haben. Das Framework wird durch Vaadin repräsentiert und ist vom Auftraggeber festgelegt. Zwischen dem Datenbank-Server und dem Applikations-Server ist ein Kommunikationspfad über JDBC. Ein Kommunikationspfad verbindet auch das Device Client über die Applikation Webbrowser mit den Servern der Datenbank und der Applikation.

# UML Component Diagram



Das Component Diagramm dient zur Darstellung der verschiedenen Komponenten unseres Systems. Wenn sich die Klasse PatientCase ändert, dann werden die folgenden Presenter informiert (MedicationPresenter, ReportPresenter und AppointmentPresenter). Auf Grund der Änderung dieser Presenter wird die UI der drei Views MedicationView, ReportView, AppointmentView bei Bedarf geändert. Nach Änderungen durch Benutzeraktionen über die Views, wird das Model über die Presenter angepasst. Der Presenter stellt eine Schnittstelle zwischen Model und View dar. Dasselbe gilt für die beiden Model Doctor und Patient. Alle Daten der Modelle werden in einer Datenbank langfristig gespeichert.

# UML Class Diagram of main view



Das oben abgebildete UML Diagramm zeigt die Klassen mit Attributen und einigen vorläufigen Methoden, die für unsere Software benötigt werden. Verwendet wurde dabei das MVP Design Pattern, wobei wir für jede Maske unserer Software eine eigene View- und eine eigene Presenter-Klasse verwenden, die sich gegenseitig aufrufen. In der Main-View wird der Arzt sich zunächst mit ID und Passwort einloggen können, weswegen dort das DoctorModel mit einer Liste aller verfügbaren Ärzte als Model an den MainPresenter übergeben. Das DoctorModel übernimmt dann die Funktion des Login-Checks. Danach wählt man im PatientView einen Patienten, mit dem man arbeiten möchte aus einer Liste von Patienten. Dies wird über das PatientModel verarbeitet. Für die restlichen Funktionen bedient der Arzt ein PatientCase, welches bei uns als Model fungiert. Der PatientCase enthält alle Reports, Appointments und den Medikamentenplan zu einem Patientenfall und kann solche auch erstellen. Diese Funktionen werden über die entsprechenden Presenter angesteuert.