Task 09

TEAM ORANGE

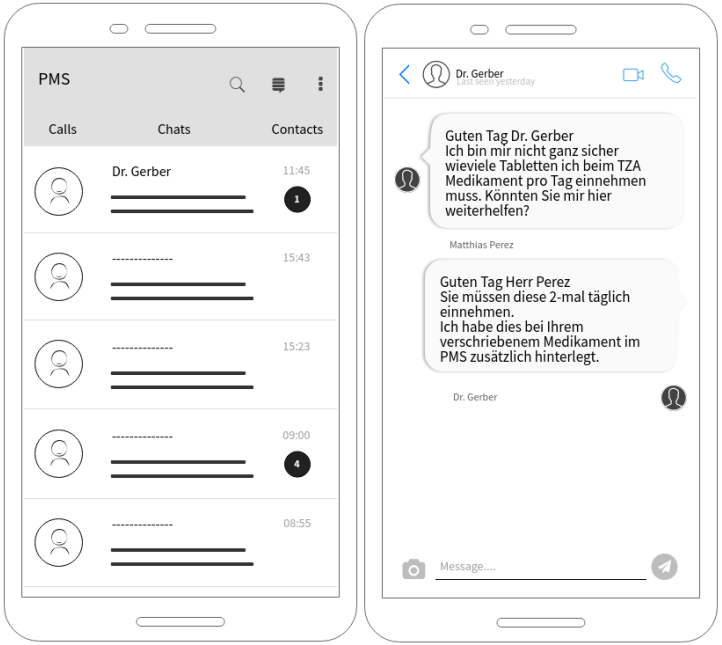
Adrian Berger, Oliver Kunz, Fabian Küng, Lorenz Sieber, Jonas Herzog

INHALT

FEATURES VS. DESIGN THINKING

DESIGN THINKING

- Informationsseite ✓
- Neuen Patienten registrieren ✓
- Neuen Termin erfassen ✓
- Stimmungstagebuch & Aktivitätentagebuch ✓
  - Eintrag erfassen ✓
  - Einträge analysieren ✓
- Medikament für Patienten hinzufügen ✓
- Erinnerung Medikamenteneinnahme ✓
- Chat ✓
- Kalender ✗
- Notfallbutton ✓
- Browser Benachrichtigungen ✗



CODING HIGHLIGHTS

Spring Stack

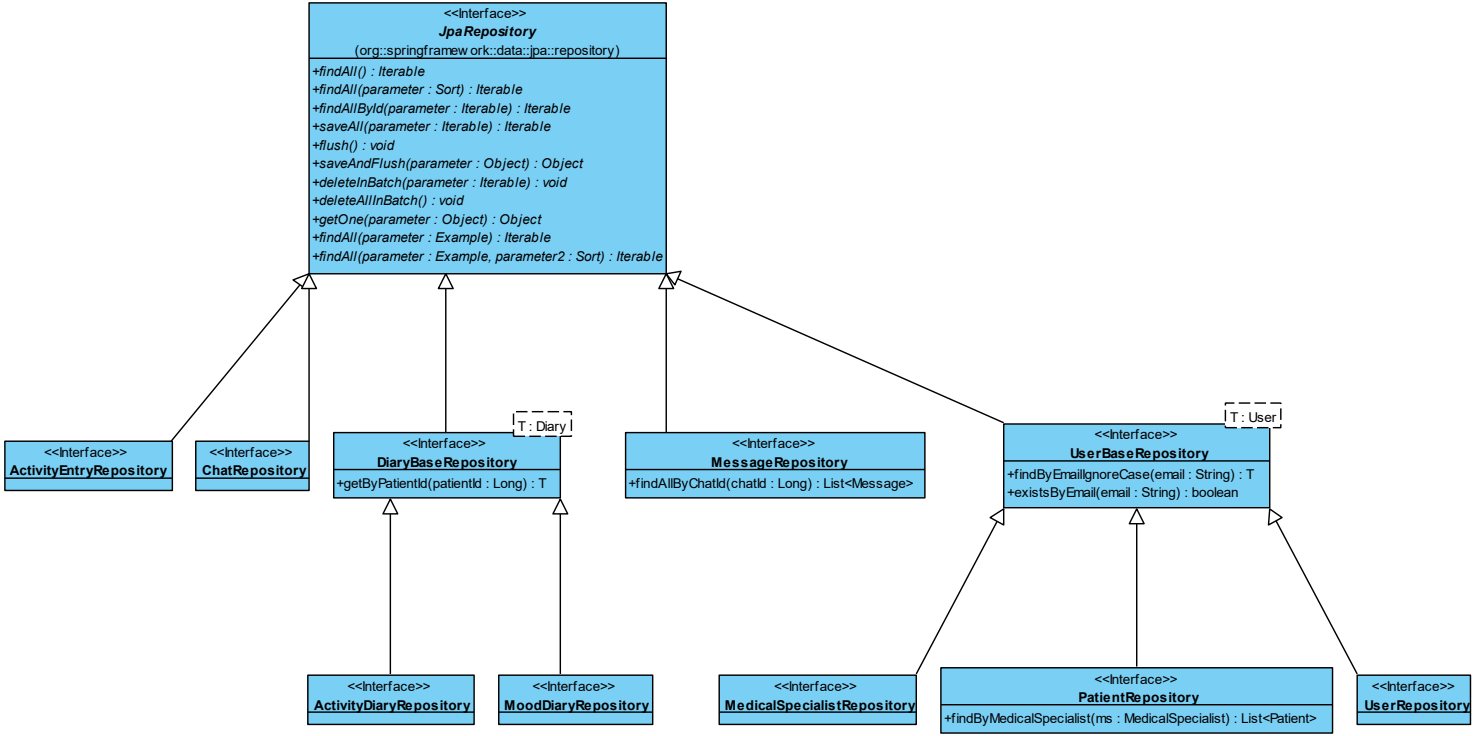
- Verwendung von **Spring Boot** (und Allgemein **Spring Stack**)
    - **Dependency Injection**
    - **Embedded Tomcat**
    - **Spring Data** für Datenbankzugriff
    - **Spring Security** für Authentication & Authorization
    - **DevTools** mit Livereload
- Verwendung von **Reactor** für **Reactive Programming** beim Chat
- Verwendung von **Lombok**

Patterns / Paradigmen

- Model-View-Presenter (MVP)
- Observer
- Dependency Injection
- Reactive, functional & object-oriented Programming
- …

CODING HIGHLIGHTS

Spring Data



CODING HIGHLIGHTS

Dependency Injection

```java
@Service
@RequiredArgsConstructor
@SessionScope
public class MessageService {

  private final UnicastProcessor<Message> publisher;
  private final MessageRepository messageRepository;
  private final Flux<Message> messages;
  private final ChatService chatService;

  ...

}
```

CODING HIGHLIGHTS

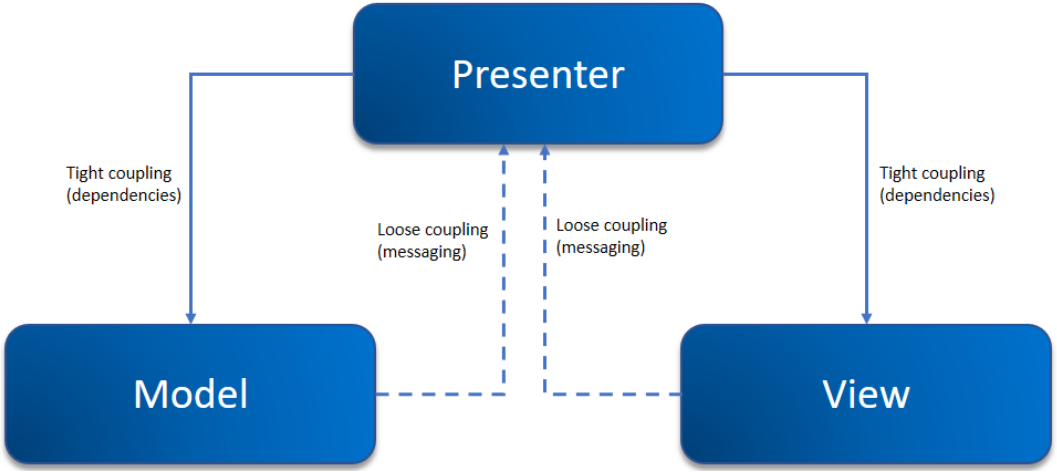
Lombok

```java
@EqualsAndHashCode(callSuper = true)
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@SuperBuilder
public class Message extends AbstractEntity {

  private String content;
  private LocalDateTime creationDate;
  private MessageState state;

  @ManyToOne
  private User sender;

  @ManyToOne
  private Chat chat;
}
```

CODING HIGHLIGHTS

MVP-PATTERN

- Vaadin Route von View getrennt
- Loose Coupling mit Observer-Pattern



CODING HIGHLIGHTS

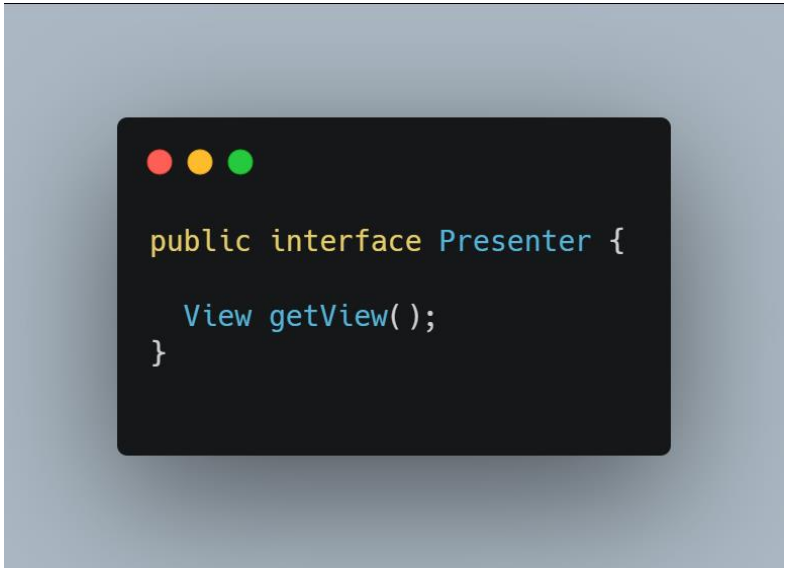
MVP-PATTERN (View)

```java
public interface View extends HasComponents {

  <C> C getComponent(Class<C> type);
}
```

```java
public interface ViewWithObserver<T> extends View {

  void setObserver(T observer);
}
```

CODING HIGHLIGHTS

MVP-PATTERN (Presenter)



CODING HIGHLIGHTS

MVP-PATTERN (Beispiel View)

```java
public interface ChatView extends ViewWithObserver<Observer> {

  void addMessage(Message message);

  void setChats(List<Chat> chats);

  interface Observer {

    void onAddMessage(String messageContent);

    void onLoadChat(long chatId);
  }
}
```

```java
@UIScope
@Component
@RequiredArgsConstructor
public class ChatViewImpl extends SplitLayout implements ChatView {
  ....
}
```

CODING HIGHLIGHTS

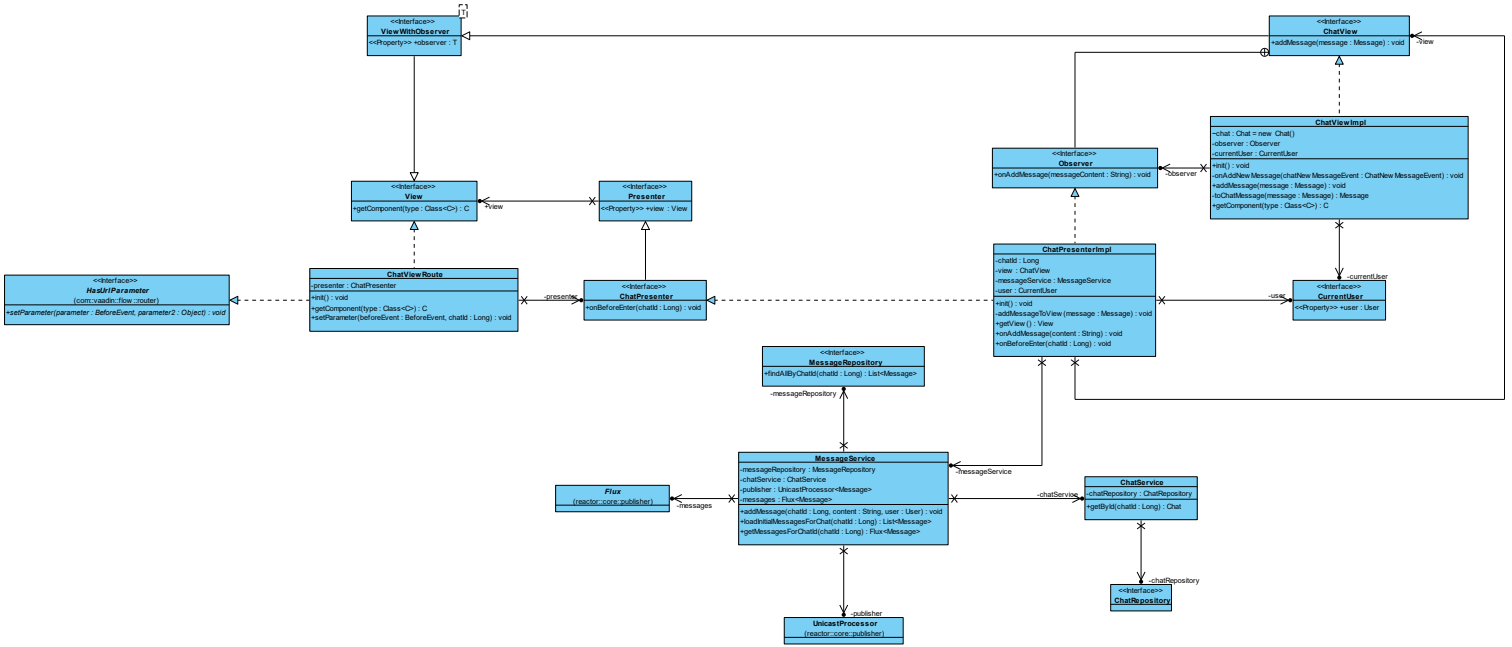
MVP-PATTERN (Beispiel Presenter)

```java
public interface ChatPresenter extends Presenter {

  void onBeforeEnter();
}
```

```java
@Component
@Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)
@RequiredArgsConstructor
public class ChatPresenterImpl implements ChatPresenter, ChatView.Observer {
  ...
}
```

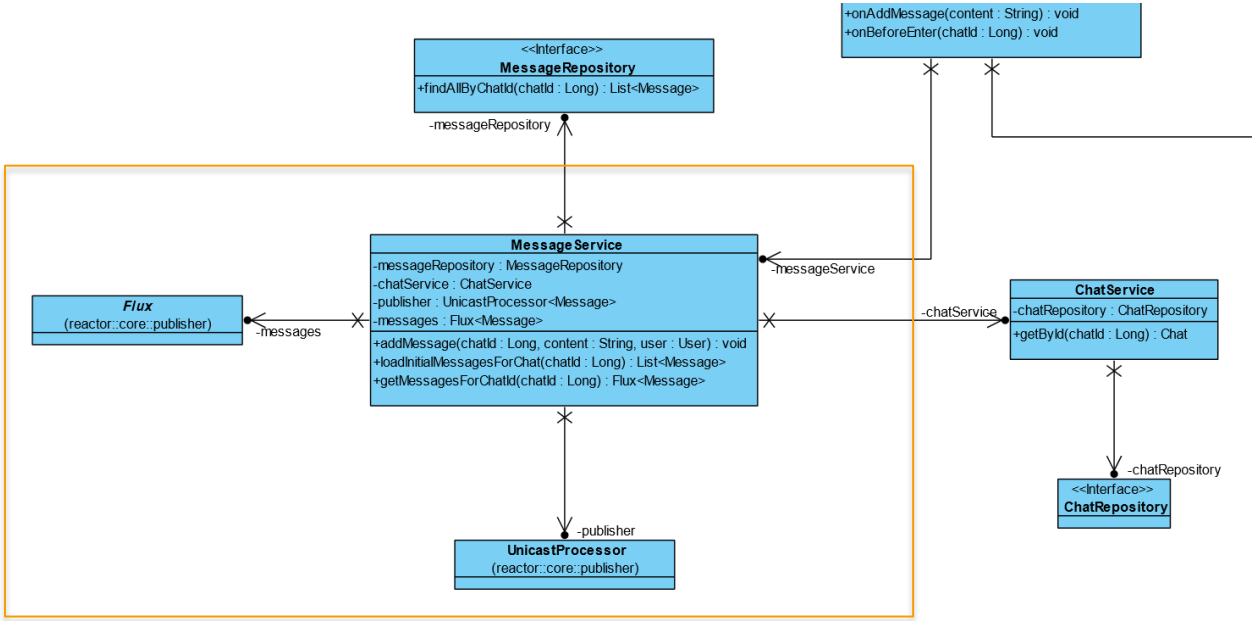
CODING HIGHLIGHTS

CHAT (Reactor)



CODING HIGHLIGHTS

CHAT (Reactor)



CODING HIGHLIGHTS

CHAT (Reactor)

Bean erstellen (Application.java)

```
@Bean
UnicastProcessor<Message> publisher() {
  return UnicastProcessor.create();
}

@Bean
Flux<Message> messages(UnicastProcessor<Message> publisher) {
  return publisher.replay(0).autoConnect();
}
```

CODING HIGHLIGHTS

CHAT (Reactor)

Neue Nachrichten publishen

```java
@Service
@RequiredArgsConstructor
@SessionScope
public class MessageService {

  private final UnicastProcessor<Message> publisher;
  private final MessageRepository messageRepository;
  private final Flux<Message> messages;
  private final ChatService chatService;

  public void addMessage(Long chatId, String content, User user) {
    Chat chat = chatService.getById(chatId);

    Message message = new Message(content, LocalDateTime.now(), MessageState.UNREAD, user,
chat);
    publisher.onNext(messageRepository.save(message));
  }

  public Flux<Message> getMessages() {
    return messages;
  }
}
```

CODING HIGHLIGHTS

CHAT (Reactor)

Subscriben (Presenter)

```
@PostConstruct
public void init() {
  messageService.getMessages().subscribe(this::addMessageToView);
}
```

CODING HIGHLIGHTS

VAADIN-VALIDIERUNG

```
binder.forField(birthDateDP)
        .asRequired("Geburtsdatum muss gesetzt sein.")
        .withValidator(date -> date.isBefore(LocalDate.now()),
            "Geburtsdatum muss in der Vergangenheit sein.")
        .bind(Patient::getBirthDate, Patient::setBirthDate);
```

LESSONS LEARNT

WAS LIEF GUT?

- Verwendung von GitHub war sehr praktisch (Backlog, Issues, Code Reviews, Pull Requests, …)
- Zusammenarbeit via Microsoft Teams, WhatsApp und GitHub funktionierte gut
- Allgemein sehr viel gelernt (vor allem dank Spring)
- Viel konnte sehr einfach implementiert warden (vor allem dank Spring)
- Viele und hilfreiche Features implementiert
- Eingeführte Retroperspektive & Pair-Programming hat sehr viel geholfen

WAS LIEF SCHLECHT?

- Zu Beginn viele Merge-Konflikte, da noch kein solide Code Basis / Abhängigkeiten
- Ab und zu vor Ort wäre trotzdem hilfreich gewesen (Corona)
- Viele neue Frameworks, Einarbeitungszeit (bspw. Vererbung mit Spring Data)
- Testing war herausforderend aufgrund von Spring (auch schwierig was genau testen)

LESSONS LEARNT

VERBESSERUNGSMÖGLICHKEITEN

- GitHub Issues in noch kleinere Tasks unterteilen (Schätzung, mehrere Sprints)
- Tasks für Testing, Visuals, Finishing erstellen
- Refinement der Tickets

Ausblick

Anpassungen

- Von In-Memory (H2) auf persistente Datenbank (MySQL, MariaDB) wechseln

→ Kleine Anpassung in Spring Konfiguration

- Temporäre Benutzer entfernen & Rolle ‘Administrator’ hinzufügen
- Security (SSL, OWASP, …)
- Erweitern der Tests
- Optimierungen der verschiedenen Features (neue Features hinzufügen)
- Bereitstellen Infrastruktur & Deployment
- Gesetzliches/Regulationen & Linzenzen
- Application Lifecycle Management
- Continuous Integration, Continuous Delivery
- …

FRAGEN?

DANKE