CleverCash

Gruppe Stratmann

Agenda

Heutiger Fahrplan

1. **Vorstellung**
2. **Erfolge**
3. **Probleme**
4. **Lessons learned**
5. **Noch offen**
6. **Show-Case**

Vorstellung

Gruppe Stratmann mit Projekt „CleverCash“

Erfolge
Probleme
Lessons learned
Noch offen
Show-Case

Vorstellung Gruppe Stratmann

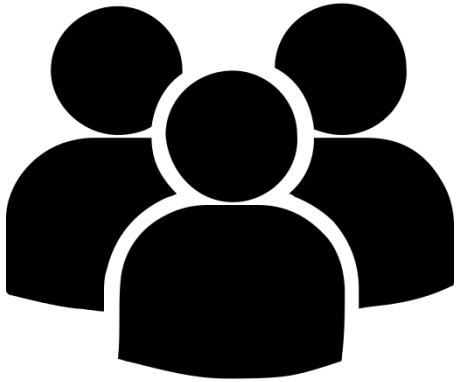
Wir Wirtschaftler brauchen das fürs Ego

Richard Prax
4. Semester

Technical Lead
Developer

Lilou Steffen
6. Semester

Developer



Anton Götz
4. Semester

Tester

Jakob Roch
4. Semester

Developer

Vorstellung Projekt „CleverCash“

Was haben Wir Uns nur dabei gedacht?.. Die Idee dahinter

- Teams mit mehreren Beteiligten einen leichteren Zugang bieten
  - Über ihre Finanzen insgesamt, aber auch einzeln
  - Übersichtliche Darstellungen von Einnahmen und Ausgaben
  - Den Beteiligten flexibler Geld zur Verfügung stellen können

- Beteiligten die Chance geben, mehr Mitbestimmung über Finanzen zu bekommen
  - Leichter Pizza für die Kollegschaft bestellen können
  - Mietwagen einfacher bestellen können
  - Auf einen teuren, aber geilen Kaffee-Vollautomaten sparen können
  - Crowdfunding für frisch Verheiratete oder die, welche mit Nachwuchs gesegnet wurden

Vorstellung
Erfolge

Was wir erreicht haben

Probleme
Lessons learned
Noch offen
Show-Case

Codebeispiele

Was wir erreicht haben

- Implementierung der Modelle

```java
@Entity (name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int userID;

    @Column(nullable = false)
    private String firstName;

    @Column(nullable = false)
    private String lastName;

    @Column(nullable = false, unique = true)
    private String email;

    @Column(nullable = false)
    private String password;

    @Column(nullable = false)
    private LocalDate birthDate;

    @Cascade(CascadeType.ALL)
    @ManyToOne
    @JoinColumn(name = "addressID")
    private Address address;

    @JsonManagedReference
    @Cascade(CascadeType.ALL)
    @OneToMany(mappedBy = "user", orphanRemoval = true)
    private List<BankAccount> bankAccounts = new ArrayList<>();

    @Temporal(TemporalType.TIMESTAMP)
    private LocalDateTime lastOnline;

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof User user)) return false;
        return userID == user.userID;
    }

    @Override
    public int hashCode() {
        return Objects.hash(userID);
    }
}
```

```java
public class UserDTO {
    private int id;
    private String firstName;
    private String lastName;
    private String birthDate;
    private String email;
    private String address;

    public UserDTO(int id, String firstName, String lastName, String birthDate, String email){
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.birthDate = birthDate;
        this.email = email;
        this.address = null;
    }
}
```

Codebeispiele

Was wir erreicht haben

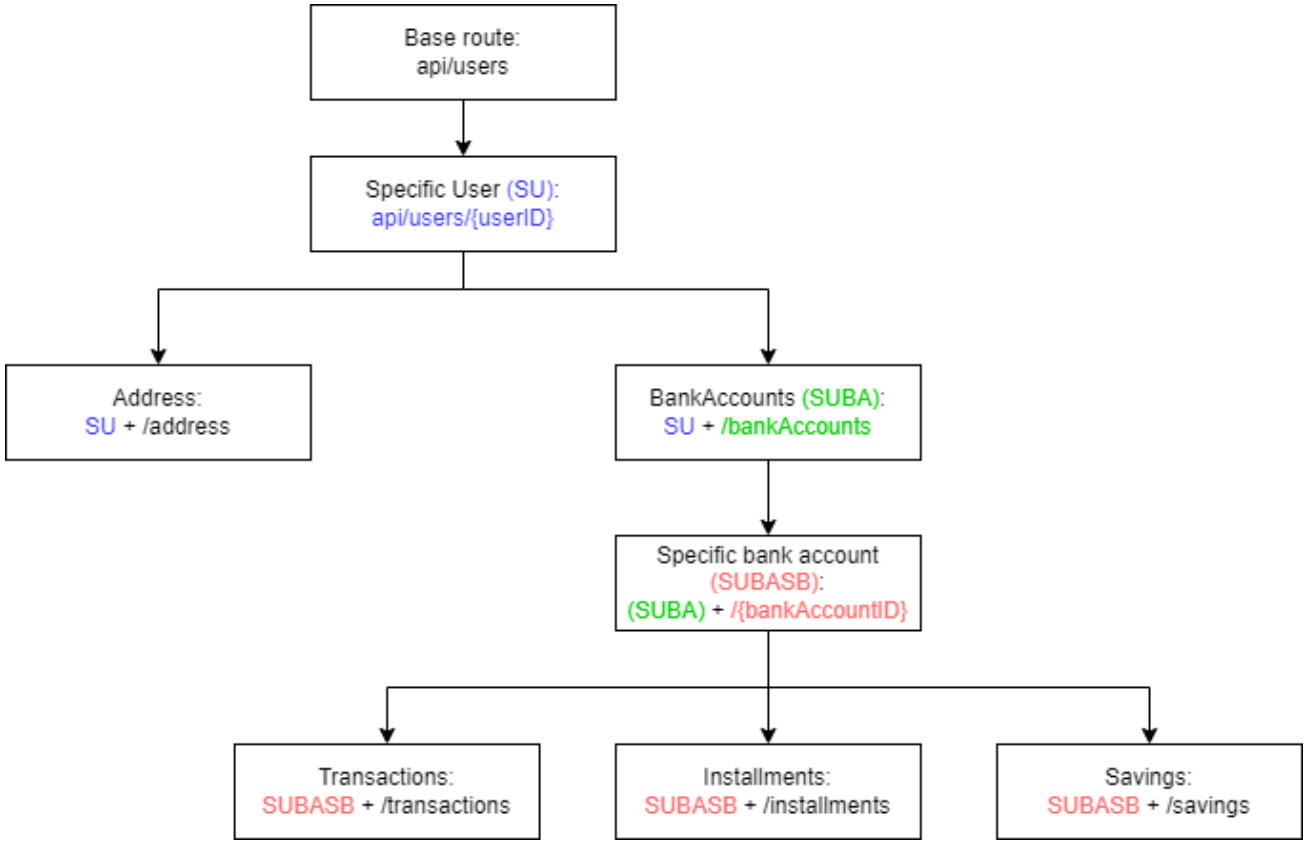
- Implementierung der Modelle
- Implementierung der Mapper

```java
public static UserDTO userToUserDTO(User user) {
    UserDTO userDTO = new UserDTO(user.getUserID(),
            user.getFirstName(),
            user.getLastName(),
            user.getBirthDate().format(defaultDateTimeFormatter),
            user.getEmail());

    if (user.getAddress() != null) {
        userDTO.setAddress(addressToString(user.getAddress()));
    }

    return userDTO;
}

public static User newUserDTOToUser(NewUserDTO newUserDTO) throws MappingException {
    // Parse Birthday
    LocalDate parsedBirthday = null;
    try {
        parsedBirthday = LocalDate
                .parse(newUserDTO.getBirthDate(), defaultDateTimeFormatter);
    } catch (DateTimeParseException dtpe) {
        throw new MappingException("Birthday could not be parsed", dtpe);
    }

    // Create and return User
    return new User(
            newUserDTO.getFirstName(),
            newUserDTO.getLastName(),
            newUserDTO.getEmail(),
            newUserDTO.getPassword(),
            parsedBirthday
    );
}
```

Codebeispiele

Was wir erreicht haben

- Implementierung der Modelle
- Implementierung der Mapper
- Festlegen der Routen für REST Anfragen



Bsp.: User1 möchte sich Details zu BankAccount1 anzeigen lassen
=> {GET, localhost:8080/api/users/1/bankAccounts/1}

Codebeispiele

Was wir erreicht haben

- Implementierung der Modells
- Implementierung der Mapper
- Festlegen der Routen für REST Anfragen
- Implementierung der REST-Controller zur Verarbeitung von HTTP Anfragen

```java
@RequestMapping(path = "/api/users/{userID}/bankAccounts/{bankAccountID}/transactions")
public class TransactionController {
    private final TransactionService transactionService;
    @GetMapping
    public ResponseEntity<List<TransactionDTO>> findTransactions(
            @PathVariable int userID,
            @PathVariable int bankAccountID,
            @RequestParam(required = false) String startDate,
            @RequestParam(required = false) String endDate,
            @RequestParam(required = false) String transactionType,
            @RequestParam(required = false) String description) {

        try {
            List<Transaction> transactions = transactionService.findAllTransactionsForUserWithFilters(userID, bankAccountID, startDate, endDate, transactionType, description);
            return ResponseEntity.ok(transactions
                    .stream()
                    .map(Mapper::transactionToTransactionDTO)
                    .collect(Collectors.toList()));
        } catch (NoSuchElementException e) {
            return ResponseEntity.notFound().build();
        } catch (DateTimeParseException | IllegalArgumentException e) {
            return ResponseEntity.badRequest().build();
        }
    }
    @GetMapping(path = "/{transactionID}")
    public ResponseEntity<TransactionDTO> findTransactionByID(
            @PathVariable int userID,
            @PathVariable int bankAccountID,
            @PathVariable int transactionID) {
        try {
            Transaction transaction = transactionService.findTransactionByID(userID, bankAccountID, transactionID);
            return ResponseEntity.ok(Mapper.transactionToTransactionDTO(transaction));
        } catch (NoSuchElementException e) {
            return ResponseEntity.notFound().build();
        }
    }
    @PostMapping
    public ResponseEntity<BankAccountDTO> addTransactionToBankAccount(
            @PathVariable int userID,
            @PathVariable int bankAccountID,
            @RequestBody NewTransactionDTO newTransactionDTO) {
        try {
            BankAccount bankAccount = transactionService.addTransactionToBankAccount(userID, bankAccountID, newTransactionDTO);
            return ResponseEntity.ok(Mapper.bankAccountToBankAccountDTO(bankAccount));
        } catch (IllegalArgumentException e) {
            return ResponseEntity.badRequest().build();
        } catch (NoSuchElementException e){
            return ResponseEntity.notFound().build();
        }
    }
}
```

Codebeispiele

Was wir erreicht haben

- Implementierung der Modells
- Implementierung der Mapper
- Festlegen der Routen für REST Anfragen
- Implementierung der REST-Controller zur Verarbeitung von HTTP Anfragen
- Implementierung der Services zur Bereitstellung nötiger Informationen für die Controller

```java
public Transaction findTransactionByID(int userID, int bankAccountID, int transactionID) throws NoSuchElementException {
    BankAccount bankAccount = bankAccountService.findBankAccountByID(userID, bankAccountID);
    return bankAccount.getTransactions().stream()
            .filter(transaction -> transaction.getTransactionID() == transactionID)
            .findFirst()
            .orElseThrow(() -> new NoSuchElementException("Transaction not found for bankAccount with id " + bankAccountID));
}
```

```java
public User findUserByID(int userID) {
    return userRepository.findById(userID)
            .orElseThrow(() -> new NoSuchElementException("User not found"));
}
```

Codebeispiele

Was wir erreicht haben

- Implementierung der Modells
- Implementierung der Mapper
- Festlegen der Routen für REST Anfragen
- Implementierung der REST-Controller zur Verarbeitung von HTTP Anfragen
- Implementierung der Services zur Bereitstellung nötiger Informationen für die Controller
- Implementierung der Repositories zur Kommunikation mit der DB

```java
@Repository
public interface UserRepository extends JpaRepository<User, Integer> {
    Optional<User> findByEmail(String email);
}
```

Codebeispiele

Was wir erreicht haben

- Implementierung der Modells
- Implementierung der Mapper
- Festlegen der Routen für REST Anfragen
- Implementierung der REST-Controller zur Verarbeitung von HTTP Anfragen
- Implementierung der Services zur Bereitstellung nötiger Informationen für die Controller
- Implementierung der Repositories zur Kommunikation mit der DB
- Implementierung von Tests

```java
@Test
void testGetAllUsers(){
    // arrange
    when(userRepository.findAll()).thenReturn(TestUtils.getTestUserList());

    // act
    List<User> users = userService.getAllUsers();

    // assert
    assertEquals(2, users.size());
    assertTrue(users.contains(TestUtils.getTestUser1()));
    assertTrue(users.contains(TestUtils.getTestUser2()));
}
```

Vorstellung
Erfolge
Probleme

Wo Schwierigkeiten auftraten

Lessons learned
Noch offen
Show-Case

Codebeispiele

Wo Schwierigkeiten auftraten

- Wie, wo, wann genau werden DTOs verwendet?
- ist das DTO-Pattern verpflichtend?
- ist das gängige Praxis?

- Was unterscheidet hier DTO und DBO? (Beispiel aus Addressbook)

```java
public class Person extends AbstractDatabaseEntity {
    private String firstName;
    private String lastName;
    private LocalDate birthday;
    @Column(unique = true)
    private String phoneNumber;
    @Column(unique = true)
    private String email;
    private Category category = Category.UNSPECIFIED;
    @ManyToOne(cascade = CascadeType.PERSIST)
    private Address address;
```

```java
public static PersonDto personToDto(Person person) {
    return new PersonDto(
            person.getId(),
            person.getFirstName(),
            person.getLastName(),
            person.getBirthday().format(defaultDateTimeFormatter),
            person.getPhoneNumber(),
            person.getEmail(),
            person.getCategory().name(),
            addressToString(person.getAddress())
    );
}
```

Vorstellung
Erfolge
Probleme
Lessons learned

Was wir dabei lernen konnten

Noch offen
Show-Case

Lessons Learned

Was wir dabei lernen konnten

- Implementierung von RESTful APIs
- Docker
- Lambdas und Streams

Vorstellung
Erfolge
Probleme
Lessons learned
Noch offen

Was liegen blieb

Show-Case

Codebeispiele

Was liegen blieb

- Tests fertig schreiben
- Doku aufbereiten
- Refactoring der Programmeinheiten
- JavaDoc kontrollieren und überarbeiten
- Umbau auf DTO Pattern
- Implementierung JWT
- JSON Web Token implementieren wenn noch Zeit ist

Vorstellung
Erfolge
Probleme
Lessons learned
Noch offen
Show-Case

Und funktionierts auch?

...wenn keine Fragen mehr sind...

Vielen Dank für Eure Aufmerksamkeit!