POLITECNICO
MILANO 1863

DATA BASES 2
Project

Fontana Nicolò (10581197)
Gerosa Andrea (10583298)

Academic Year: 2021/2022
Prof. Piero Fraternali

Index

Specifications

TELCO SERVICE APPLICATIONS
A telco company offers pre-paid online services to web users. Two client applications using the same database need to be developed.

CONSUMER APPLICATION
The consumer application has a public Landing page with a form for login and a form for registration. Registration requires a username (which can be assumed as the unique identification parameter), a password and an email. Login leads to the Home page of the consumer application. Registration leads back to the landing page where the user can log in.
The user can log in before browsing the application or browse it without logging in. If the user has logged in, his/her username appears in the top right corner of all the application pages.
The Home page of the consumer application displays the service packages offered by the telco company.
A service package has an ID and a name (e.g., “Basic”, “Family”, “Business”, “All Inclusive”, etc). It comprises one or more services. Services are of four types: fixed phone, mobile phone, fixed internet, and mobile internet. The mobile phone service specifies the number of minutes and SMSs included in the package plus the fee for extra minutes and the fee for extra SMSs. The fixed phone service has no specific configuration parameters. The mobile and fixed internet services specify the number of Gigabytes included in the package and the fee for extra Gigabytes.
A service package must be associated with one validity period. A validity period specifies the number of months (12, 24, or 36). Each validity period has a different monthly fee (e.g., 20€/month for 12 months, 18€/month for 24 months, and 15€ /month for 36 months). A package may be associated with one or more optional products (e.g., an SMS news feed, an internet TV channel, etc.). The validity period of an optional product is the same as the validity period that the user has chosen for the service package. An optional product has a name and a monthly fee independent of the validity period duration. The same optional product can be offered in different service packages.
From the Home page, the user can access a Buy Service page for purchasing a service package and thus creating a service subscription. The Buy Service page contains a form for purchasing a service package. The form allows the user to select one package from the list of available ones and choose the validity period duration and the optional products to buy together with the chosen service. The form also allows the user to select the start date of his/her subscription. After choosing the service packages, the validity period and (0 or more) optional products, the user can press a CONFIRM button. The application displays a CONFIRMATION page that summarizes the details of the chosen service package, the validity period, the optional products and the total price to be pre-paid: (monthly fee of service package * number of months) + (sum of monthly fees of options * number of months).
If the user has already logged in, the CONFIRMATION page displays a BUY button. If the user has not logged in, the CONFIRMATION page displays a link to the login page and a link to the REGISTRATION page. After either logging in or registering and immediately logging in, the CONFIRMATION page is redisplayed with all the confirmed details and the BUY button.
When the user presses the BUY button, an order is created. The order has an ID and a date and hour of creation. It is associated with the user and with the service package, its validity period and the chosen optional products. It also contains the total value (as in the CONFIRMATION page) and the start date of the subscription. After creating the order, the application bills the customer by calling an external service. If the external service accepts the billing, the order is marked as valid and a service activation schedule is created for the user. A service activation schedule is a record of the services and optional products to activate for the user with their date of activation and date of deactivation.
If the external service rejects the billing, the order is put in the rejected status and the user is flagged as insolvent. When an insolvent user logs in, the home page also contains the list of rejected orders. The user can select one of such orders, access the CONFIRMATION page, press the BUY button and attempt the payment again. When the same user causes three failed payments, an alert is created in a dedicated auditing table, with the user Id, username, email, and the amount, date and time of the last rejection.

EMPLOYEE APPLICATION

The employee application allows the authorized employees of the telco company to log in. In the Home page, a form allows the creation of service packages, with all the needed data and the possible optional products associated with them. The same page lets the employee create optional products as well.

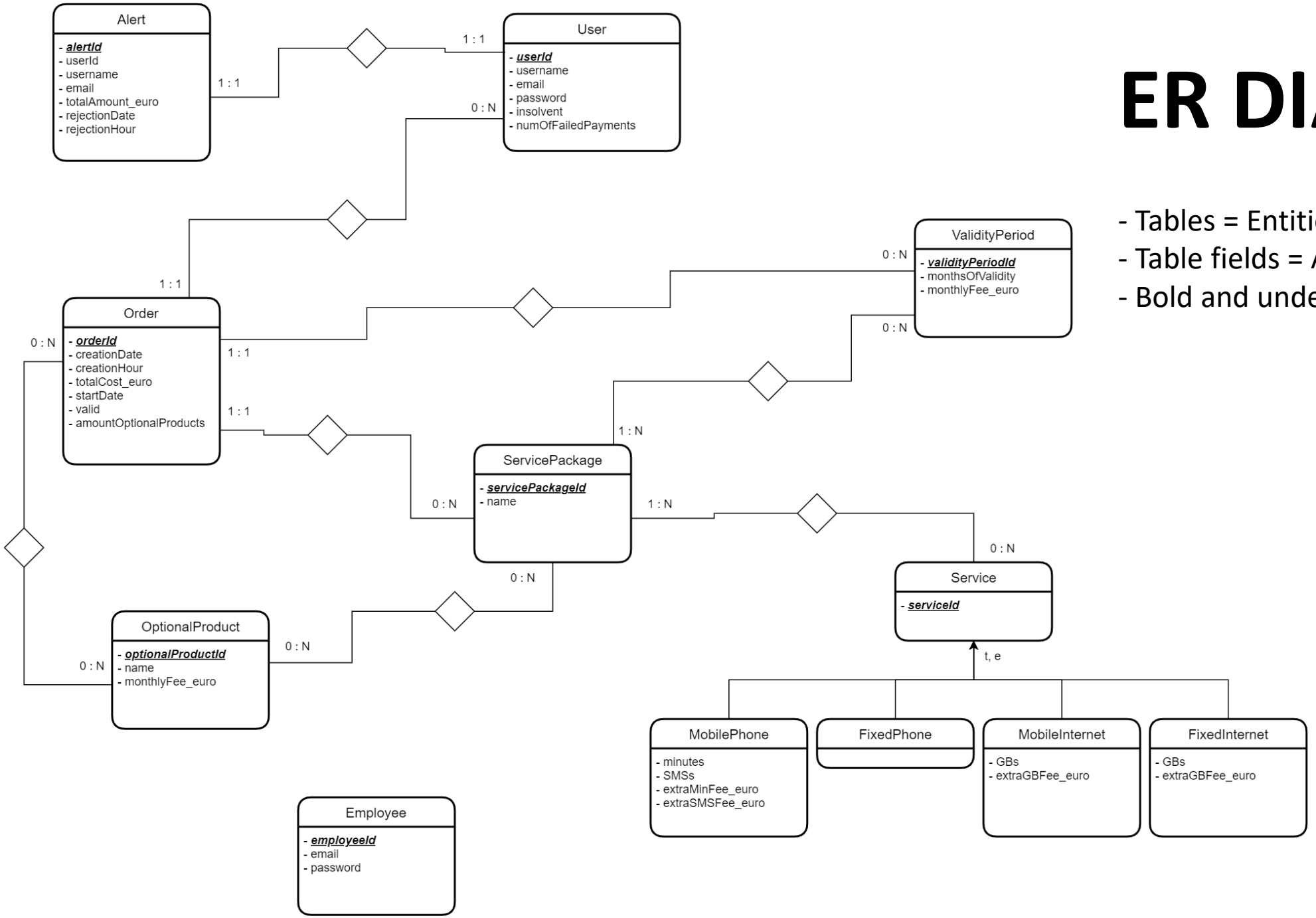
A Sales Report page allows the employee to inspect the essential data about the sales and about the users over the entire lifespan of the application:

- Number of total purchases per package.
- Number of total purchases per package and validity period.
- Total value of sales per package with and without the optional products.
- Average number of optional products sold together with each service package.
- List of insolvent users, suspended orders and alerts.
- Best seller optional product, i.e. the optional product with the greatest value of sales across all the sold service packages.

DESIGN DOCUMENTATION AND IMPLEMENTATION NOTES

- The call to the external service must be simulated with a function that returns true or false pseudo-randomly. For testing purposes, the demonstration should be able to show at least one case in which the service call fails and one case in which the service call succeeds.
- The aggregate data of the sales report must be computed by triggers that populate materialized view tables. The documentation must describe the SQL code of the view that would compute the aggregate data, the logical schema of the materialized view table(s) that store the aggregate data and the triggers that populate the content of the materialized view table(s).
- The project documentation should be realized in PDF or PPT format with the following parts:
    - o Description of any extra hypothesis on the project specifications
    - o Entity Relationship diagram
    - o Relation model of the database in SQL or graphical format
    - o Description of the views, materialized view tables and code of the materialization triggers
    - o Description of the ORM
    - o List of the application components
    - o Examples of UML sequence diagrams of particularly significant interactions
- A PPT template of the documentation is provided in the project folder in Webeep
- Examples of the documentation are provided in the JPA exercise folder

ER DIAGRAM



- Tables = Entities
- Table fields = Attributes
- Bold and underlined = Primary key

Logic model (SQL code)

```
CREATE TABLE `user` (
  `userId` int NOT NULL AUTO_INCREMENT,
  `username` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  `insolvent` int NOT NULL DEFAULT '0',
  `numOfFailedPayments` int NOT NULL DEFAULT '0',
  PRIMARY KEY (`userId`),
  UNIQUE KEY `username_UNIQUE` (`username`),
  UNIQUE KEY `email_UNIQUE` (`email`)
)
```

Attribute ‘insolvent’ is 1 if the user has a number of failed payments greater than 1, 0 otherwise

```
CREATE TABLE `employee` (
  `employeeId` int NOT NULL AUTO_INCREMENT,
  `email` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  PRIMARY KEY (`employeeId`),
  UNIQUE KEY `email_UNIQUE` (`email`)
)
```

Attribute ‘valid’ is 1 if the order has been succesfully completed, 0 otherwise

```
CREATE TABLE `order` (
  `orderId` int NOT NULL AUTO_INCREMENT,
  `creationDate` date NOT NULL,
  `creationHour` time NOT NULL,
  `totalCost_euro` float NOT NULL,
  `startDate` date NOT NULL,
  `userId` int NOT NULL,
  `servicePackageId` int NOT NULL,
  `valid` int NOT NULL DEFAULT '0',
  `validityPeriodId` int NOT NULL,
  `amountOptionalProducts` int NOT NULL,
  PRIMARY KEY (`orderId`),
  KEY `userId_idx` (`userId`),
  KEY `ORDER_servicePackageId_idx` (`servicePackageId`),
  KEY `ORDER_validityPeriodId_idx` (`validityPeriodId`),
  CONSTRAINT `ORDER_servicePackageId` FOREIGN KEY (`servicePackageId`) REFERENCES `service_package` (`servicePackageId`),
  CONSTRAINT `ORDER_userId` FOREIGN KEY (`userId`) REFERENCES `user` (`userId`),
  CONSTRAINT `ORDER_validityPeriodId` FOREIGN KEY (`validityPeriodId`) REFERENCES `validity_period` (`validityPeriodId`)
)
```

```
CREATE TABLE `service_package` (
  `servicePackageId` int NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  PRIMARY KEY (`servicePackageId`)
)
```

```
CREATE TABLE `service` (
  `serviceId` int NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`serviceId`)
)
```

```
CREATE TABLE `fixed_internet` (
  `serviceId` int NOT NULL,
  `GBs` int NOT NULL,
  `extraGBFee_euro` float DEFAULT NULL,
  PRIMARY KEY (`serviceId`),
  CONSTRAINT `FIXED_INTERNET_serviceId` FOREIGN KEY (`serviceId`)
REFERENCES `service` (`serviceId`)
)
```

```
CREATE TABLE `fixed_phone` (
  `serviceId` int NOT NULL,
  PRIMARY KEY (`serviceId`),
  CONSTRAINT `FIXED_PHONE_serviceId` FOREIGN KEY (`serviceId`)
REFERENCES `service` (`serviceId`)
)
```

```
CREATE TABLE `mobile_internet` (
  `serviceId` int NOT NULL,
  `GBs` int NOT NULL,
  `extraGBFee_euro` float DEFAULT NULL,
  PRIMARY KEY (`serviceId`),
  CONSTRAINT `MOBILE_INTERNET_serviceId` FOREIGN KEY (`serviceId`)
REFERENCES `service` (`serviceId`)
)
```

```
CREATE TABLE `mobile_phone` (
  `serviceId` int NOT NULL,
  `minutes` int NOT NULL,
  `SMSs` int NOT NULL,
  `extraMinFee_euro` float DEFAULT NULL,
  `extraSMSFee_euro` float DEFAULT NULL,
  PRIMARY KEY (`serviceId`),
  CONSTRAINT `MOBILE_PHONE_serviceId` FOREIGN KEY (`serviceId`)
REFERENCES `service` (`serviceId`)
)
```

```
CREATE TABLE `optional_product` (
  `optionalProductId` int NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `monthlyFee_euro` float NOT NULL,
  PRIMARY KEY (`optionalProductId`)
)
```

```
CREATE TABLE `validity_period` (
  `validityPeriodId` int NOT NULL AUTO_INCREMENT,
  `monthsOfValidity` int NOT NULL,
  `monthlyFee_euro` float NOT NULL,
  PRIMARY KEY (`validityPeriodId`)
)
```

```
CREATE TABLE `alert` (
  `alertId` int NOT NULL AUTO_INCREMENT,
  `userId` int NOT NULL,
  `username` varchar(45) NOT NULL,
  `email` varchar(45) NOT NULL,
  `totalAmount_euro` float NOT NULL,
  `rejectionDate` date NOT NULL,
  `rejectionHour` time NOT NULL,
  PRIMARY KEY (`alertId`),
  KEY `userId_idx` (`userId`),
  CONSTRAINT `ALERT_userId` FOREIGN KEY (`userId`) REFERENCES `user` (`userId`)
)
```

Triggers

- First triggers activated on modification of table ORDER (some on update other on insert) to update materialized views about almost every aggregated data
- Last trigger activated on insert into table ORDER__OPTIONAL_PRODUCT to update the materialized view keeping track of the number of purchases made for each optional product (to extract then the best seller of them)

Update of total purchases per service package

```
-- 1.update total purchases per service package
CREATE DEFINER=`root`@`localhost` TRIGGER `Order.AfterInsert.UpdateTotalPurchasesPerSp` AFTER INSERT ON `order` FOR EACH ROW BEGIN
  IF new.valid = 1 THEN
      IF EXISTS (SELECT * FROM `mv_total_purchases_per_sp` AS tp
          WHERE new.servicePackageId = tp.servicePackageId) THEN
      UPDATE `mv_total_purchases_per_sp` AS tp
        SET tp.totalPurchases = (SELECT count(*) FROM `order` AS o
                  WHERE o.servicePackageId = new.servicePackageId AND o.valid = 1
                      GROUP BY o.servicePackageId)
      WHERE new.servicePackageId = tp.servicePackageId;
    ELSE
      INSERT INTO `mv_total_purchases_per_sp` (servicePackageId, totalPurchases)
      VALUE (new.servicePackageId, (SELECT count(*) FROM `order` AS o
                  WHERE o.servicePackageId = new.servicePackageId AND o.valid = 1
                      GROUP BY o.servicePackageId));
    END IF;
  END IF;
END$$
```

```
-- 1.update total purchases per service package
CREATE DEFINER=`root`@`localhost` TRIGGER `Order.AfterUpdate.UpdateTotalPurchasesPerSp` AFTER UPDATE ON `order` FOR EACH ROW BEGIN
  IF new.valid = 1 THEN
    IF EXISTS (SELECT * FROM `mv_total_purchases_per_sp`
          WHERE new.servicePackageId = servicePackageId) THEN
      UPDATE `mv_total_purchases_per_sp`
        SET totalPurchases = (SELECT count(*) FROM `order`
                  WHERE servicePackageId = new.servicePackageId AND valid = 1
                      GROUP BY servicePackageId)
      WHERE new.servicePackageId = servicePackageId;
    ELSE
      INSERT INTO `mv_total_purchases_per_sp` (servicePackageId, totalPurchases)
      VALUE (new.servicePackageId, (SELECT count(*) FROM `order`
                  WHERE servicePackageId = new.servicePackageId AND valid = 1
                      GROUP BY servicePackageId));
    END IF;
  END IF;
END$$
```

Update of total purchases per service package and validity period

```
-- 2.update total purchases per service package and validity period
CREATE DEFINER=`root`@`localhost` TRIGGER `Order.AfterInsert.UpdateTotalPurchasesPerSpAndVp` AFTER INSERT ON `order` FOR EACH ROW BEGIN
  IF new.valid = 1 THEN
    IF EXISTS (SELECT * FROM `mv_total_purchases_per_sp_and_vp` AS tpsv
          WHERE new.servicePackageId = tpsv.servicePackageId AND new.validityPeriodId = tpsv.validityPeriodId) THEN
      UPDATE `mv_total_purchases_per_sp_and_vp` AS tpsv
        SET tpsv.totalPurchases = (SELECT count(*) FROM `order` AS o
                  WHERE o.servicePackageId = new.servicePackageId AND
                        o.validityPeriodId = new.validityPeriodId AND o.valid = 1
                        GROUP BY o.servicePackageId, o.validityPeriodId)
      WHERE new.servicePackageId = tpsv.servicePackageId AND new.validityPeriodId = tpsv.validityPeriodId;
    ELSE
      INSERT INTO `mv_total_purchases_per_sp_and_vp` (servicePackageId, validityPeriodId, totalPurchases)
      VALUES (new.servicePackageId, new.validityPeriodId, (SELECT count(*) FROM `order` AS o
                  WHERE o.servicePackageId = new.servicePackageId AND o.validityPeriodId = new.validityPeriodId AND o.valid = 1
                        GROUP BY o.servicePackageId, o.validityPeriodId));
    END IF;
  END IF;
END$$
```

```
-- 2.update total purchases per service package and validity period
CREATE DEFINER=`root`@`localhost` TRIGGER `Order.AfterUpdate.UpdateTotalPurchasesPerSpAndVp` AFTER UPDATE ON `order` FOR EACH ROW BEGIN
  IF new.valid = 1 THEN
    IF EXISTS (SELECT * FROM `mv_total_purchases_per_sp_and_vp`
          WHERE new.servicePackageId = servicePackageId AND new.validityPeriodId = validityPeriodId) THEN
      UPDATE `mv_total_purchases_per_sp_and_vp`
        SET totalPurchases = (SELECT count(*) FROM `order`
                  WHERE servicePackageId = new.servicePackageId AND
                        validityPeriodId = new.validityPeriodId AND valid = 1
                        GROUP BY servicePackageId, validityPeriodId)
      WHERE new.servicePackageId = servicePackageId AND new.validityPeriodId = validityPeriodId;
    ELSE
      INSERT INTO `mv_total_purchases_per_sp_and_vp` (servicePackageId, validityPeriodId, totalPurchases)
      VALUES (new.servicePackageId, new.validityPeriodId, (SELECT count(*) FROM `order`
                  WHERE servicePackageId = new.servicePackageId AND validityPeriodId = new.validityPeriodId AND valid = 1
                        GROUP BY servicePackageId, validityPeriodId));
    END IF;
  END IF;
END$$
```

Update of total value per service package

```
-- 3.update total value per service package
CREATE DEFINER=`root`@`localhost` TRIGGER `Order.AfterInsert.UpdateTotalValuePerSp` AFTER INSERT ON `order` FOR EACH ROW BEGIN
  IF new.valid = 1 THEN
    IF EXISTS (SELECT * FROM `mv_total_value_per_sp` AS tv
      WHERE new.servicePackageId = tv.servicePackageId) THEN
     UPDATE `mv_total_value_per_sp` AS tv
      SET tv.totalValue_euro = (SELECT COALESCE(sum(vp.monthlyFee_euro),0) FROM `validity_period` AS vp, `order` AS o
            WHERE o.servicePackageId = new.servicePackageId AND o.valid = 1 AND vp.validityPeriodId = o.validityPeriodId)
     WHERE new.servicePackageId = tv.servicePackageId;
   ELSE
     INSERT INTO `mv_total_value_per_sp` (servicePackageId, totalValue_euro)
      VALUES (new.servicePackageId, (SELECT COALESCE(sum(vp.monthlyFee_euro),0) FROM `validity_period` AS vp, `order` AS o
            WHERE o.servicePackageId = new.servicePackageId AND o.valid = 1 AND vp.validityPeriodId = o.validityPeriodId));
   END IF;
  END IF;
END$$
```

```
-- 3.update total value per service package
CREATE DEFINER=`root`@`localhost` TRIGGER `Order.AfterUpdate.UpdateTotalValuePerSp` AFTER UPDATE ON `order` FOR EACH ROW BEGIN
  IF new.valid = 1 THEN
    IF EXISTS (SELECT * FROM `mv_total_value_per_sp` AS tv
      WHERE new.servicePackageId = tv.servicePackageId) THEN
     UPDATE `mv_total_value_per_sp` AS tv
      SET tv.totalValue_euro = (SELECT COALESCE(sum(vp.monthlyFee_euro),0) FROM `validity_period` AS vp, `order` AS o
            WHERE o.servicePackageId = new.servicePackageId AND o.valid = 1 AND vp.validityPeriodId = o.validityPeriodId)
     WHERE new.servicePackageId = tv.servicePackageId;
   ELSE
     INSERT INTO `mv_total_value_per_sp` (servicePackageId, totalValue_euro)
      VALUES (new.servicePackageId, (SELECT COALESCE(sum(vp.monthlyFee_euro),0) FROM `validity_period` AS vp, `order` AS o
            WHERE o.servicePackageId = new.servicePackageId AND o.valid = 1 AND vp.validityPeriodId = o.validityPeriodId));
   END IF;
  END IF;
END$$
```

Update of total value per service package (including optional products sold with it)

```
-- 4.update total value per service package with optional product
CREATE DEFINER=`root`@`localhost` TRIGGER `Order.AfterInsert.UpdateTotalValuePerSpWithOp` AFTER INSERT ON `order` FOR EACH ROW BEGIN
  IF new.valid = 1 THEN
    IF EXISTS (SELECT * FROM `mv_total_value_per_sp_with_op` AS tvso
      WHERE new.servicePackageId = tvso.servicePackageId) THEN
     UPDATE `mv_total_value_per_sp_with_op` AS tvso
      SET tvso.totalValue_euro = (SELECT COALESCE(sum(o.totalCost_euro),0) FROM `order` AS o
                  WHERE o.servicePackageId = new.servicePackageId AND o.valid = 1)
     WHERE new.servicePackageId = tvso.servicePackageId;
   ELSE
     INSERT INTO `mv_total_value_per_sp_with_op` (servicePackageId, totalValue_euro)
      VALUES (new.servicePackageId, (SELECT COALESCE(sum(o.totalCost_euro),0) FROM `order` AS o
                  WHERE o.servicePackageId = new.servicePackageId AND o.valid = 1));
   END IF;
  END IF;
END$$
```

```
-- 4.update total value per service package with optional product
CREATE DEFINER=`root`@`localhost` TRIGGER `Order.AfterUpdate.UpdateTotalValuePerSpWithOp` AFTER UPDATE ON `order` FOR EACH ROW BEGIN
  IF new.valid = 1 THEN
    IF EXISTS (SELECT * FROM `mv_total_value_per_sp_with_op` AS tvso
      WHERE new.servicePackageId = tvso.servicePackageId) THEN
     UPDATE `mv_total_value_per_sp_with_op` AS tvso
      SET tvso.totalValue_euro = (SELECT COALESCE(sum(o.totalCost_euro),0) FROM `order` AS o
                  WHERE o.servicePackageId = new.servicePackageId AND o.valid = 1)
     WHERE new.servicePackageId = tvso.servicePackageId;
   ELSE
     INSERT INTO `mv_total_value_per_sp_with_op` (servicePackageId, totalValue_euro)
      VALUES (new.servicePackageId, (SELECT COALESCE(sum(o.totalCost_euro),0) FROM `order` AS o
                  WHERE o.servicePackageId = new.servicePackageId AND o.valid = 1));
   END IF;
  END IF;
END$$
```

Update of average amount of sold optional products per service package

```
-- 5.update average amount of optional products sold with each package
CREATE DEFINER=`root`@`localhost` TRIGGER `Order.AfterInsert.UpdateAvgAmountOpPerSp` AFTER INSERT ON `order` FOR EACH ROW BEGIN
  IF new.valid = 1 THEN
    IF EXISTS (SELECT * FROM `mv_avg_amount_op_per_sp` AS aaos
      WHERE new.servicePackageId = aaos.servicePackageId) THEN
     UPDATE `mv_avg_amount_op_per_sp` AS aaos
      SET aaos.avgAmountOptionalProducts = (SELECT COALESCE(avg(o.amountOptionalProducts),0) FROM `order` AS o
                    WHERE new.servicePackageId = o.servicePackageId AND o.valid = 1
                        GROUP BY o.servicePackageId)
     WHERE new.servicePackageId = aaos.servicePackageId;
   ELSE
     INSERT INTO `mv_avg_amount_op_per_sp` (servicePackageId, avgAmountOptionalProducts)
      VALUES (new.servicePackageId, (SELECT COALESCE(avg(o.amountOptionalProducts),0) FROM `order` AS o
                    WHERE new.servicePackageId = o.servicePackageId AND o.valid = 1
                        GROUP BY o.servicePackageId));
   END IF;
 END IF;
END$$
```

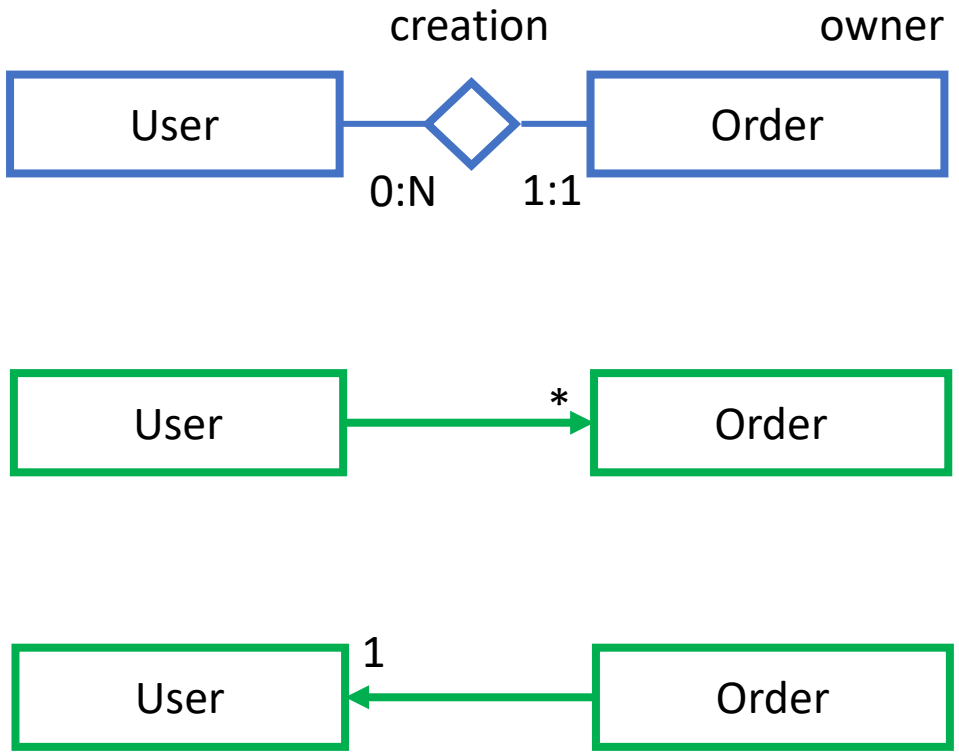
```
-- 5.update average amount of optional products sold with each package
CREATE DEFINER=`root`@`localhost` TRIGGER `Order.AfterUpdate.UpdateAvgAmountOpPerSp` AFTER UPDATE ON `order` FOR EACH ROW BEGIN
  IF new.valid = 1 THEN
    IF EXISTS (SELECT * FROM `mv_avg_amount_op_per_sp` AS aaos
      WHERE new.servicePackageId = aaos.servicePackageId) THEN
     UPDATE `mv_avg_amount_op_per_sp` AS aaos
      SET aaos.avgAmountOptionalProducts = (SELECT COALESCE(avg(o.amountOptionalProducts),0) FROM `order` AS o
                    WHERE new.servicePackageId = o.servicePackageId AND o.valid = 1
                        GROUP BY o.servicePackageId)
     WHERE new.servicePackageId = aaos.servicePackageId;
   ELSE
     INSERT INTO `mv_avg_amount_op_per_sp` (servicePackageId, avgAmountOptionalProducts)
      VALUES (new.servicePackageId, (SELECT COALESCE(avg(o.amountOptionalProducts),0) FROM `order` AS o
                    WHERE new.servicePackageId = o.servicePackageId AND o.valid = 1
                        GROUP BY o.servicePackageId));
   END IF;
 END IF;
END$$
```

Update of total purchases per optional product (the one with higher number will be the best seller)

```
-- update total purchases per optional product
CREATE DEFINER=`root`@`localhost` TRIGGER `Order_OptionalProduct.AfterInsert.UpdateTotalPurchasesPerOp` AFTER INSERT ON `order__optional_product` FOR EACH ROW BEGIN
   IF EXISTS (SELECT * FROM `order` AS o
      WHERE o.orderId = new.orderId AND o.valid = 1) THEN
    IF EXISTS (SELECT * FROM `mv_total_purchases_per_op` AS tpop
       WHERE tpop.optionalProductId = new.optionalProductId) THEN
      UPDATE `mv_total_purchases_per_op` AS tpop
       SET tpop.totalPurchases = (SELECT count(*) FROM `order__optional_product` AS oop, `order` AS o
                  WHERE oop.optionalProductId = new.optionalProductId AND oop.orderId = o.orderId AND o.valid = 1
                        GROUP BY oop.optionalProductId)
      WHERE tpop.optionalProductId = new.optionalProductId;
    ELSE
      INSERT INTO `mv_total_purchases_per_op` (optionalProductId, totalPurchases)
       VALUES (new.optionalProductId, (SELECT count(*) FROM `order__optional_product` AS oop, `order` AS o
                  WHERE oop.optionalProductId = new.optionalProductId AND oop.orderId = o.orderId AND o.valid = 1
                        GROUP BY oop.optionalProductId));
     END IF;
  END IF;
END$$
```

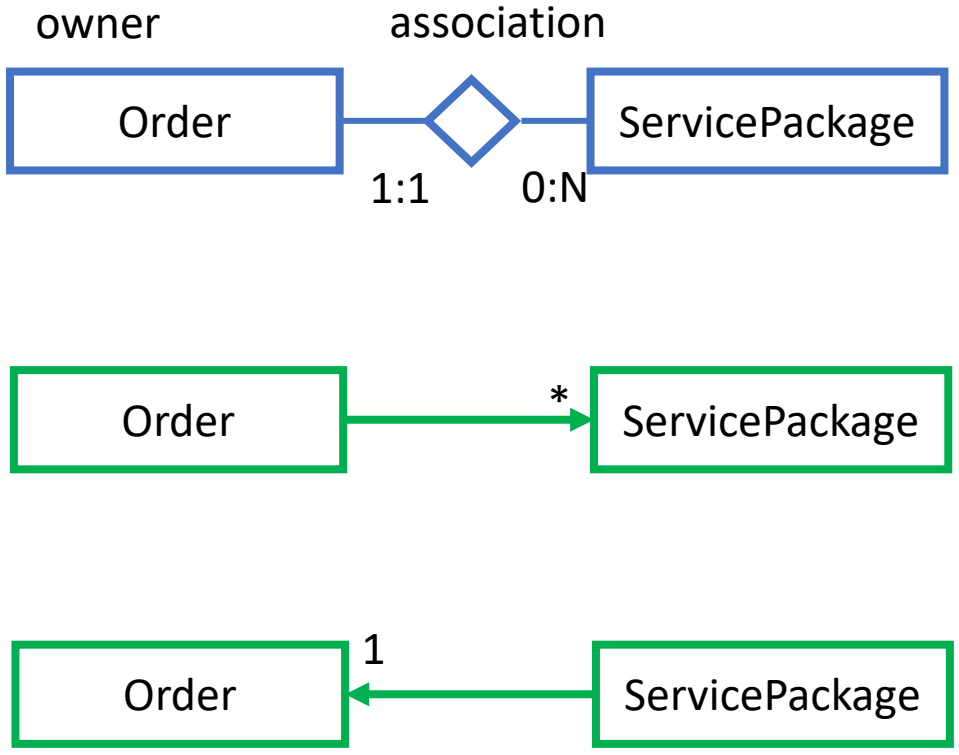
ORM mapping

Relationship “creation”



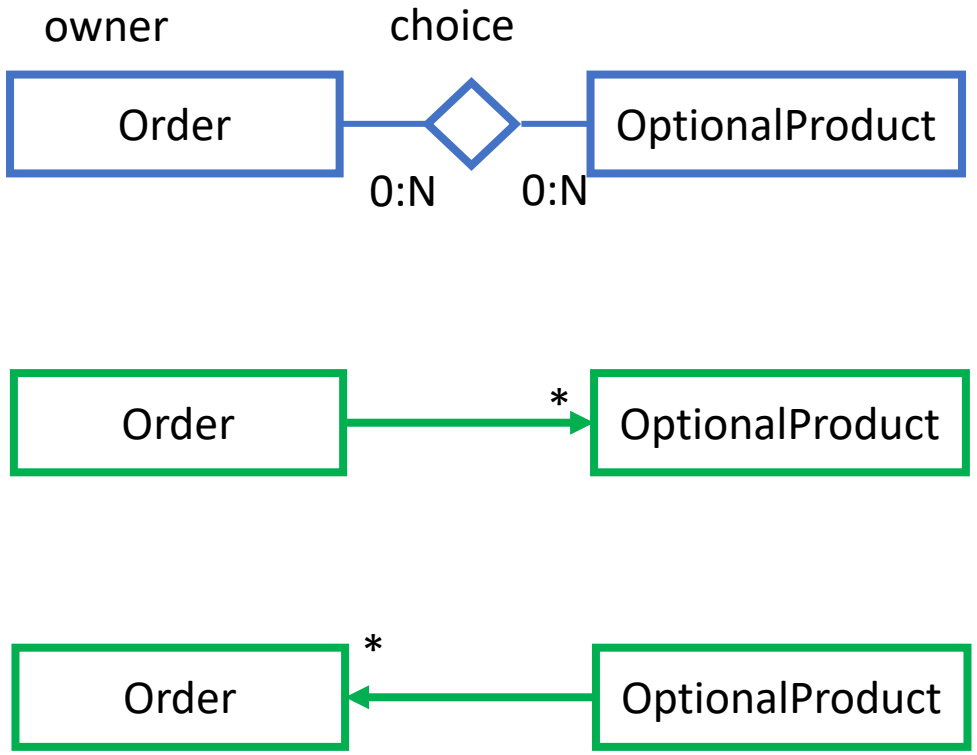
- **User → Order**
  - @OneToMany, useful to retrieve the Orders created by a User
- **Order → User**
  - @ManyToOne, useful to retrieve the User who made an Order
  - Owner because the mapped table contains the foreign key

Relationship “association”



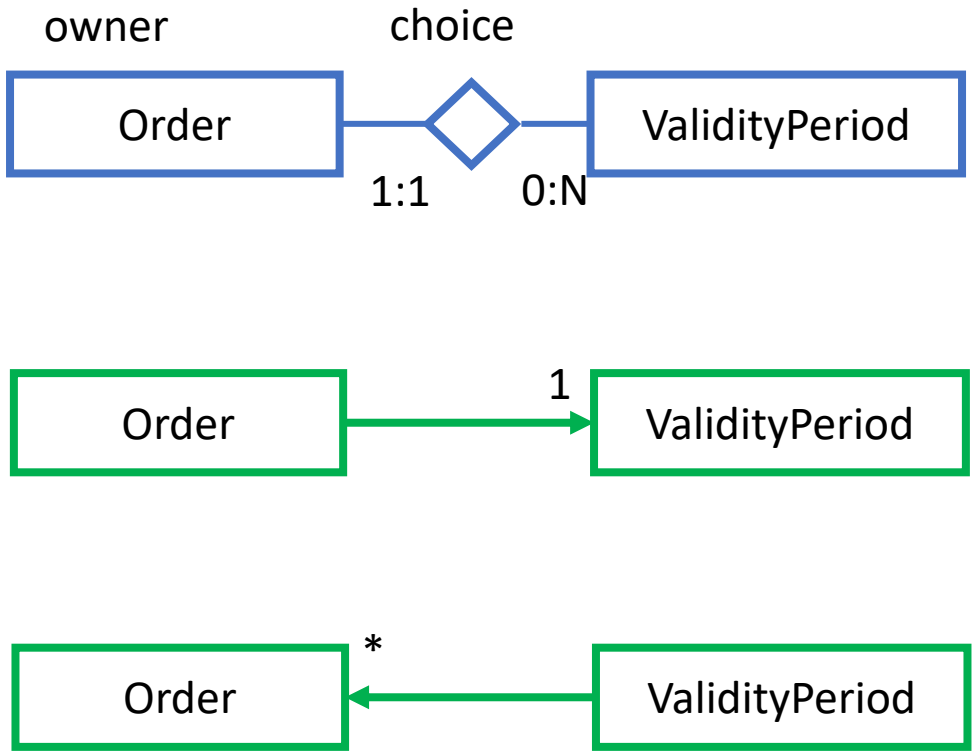
- **Order → ServicePackage**
  - @ManyToOne, useful to retrieve the ServicePackage associated to a Order
  - Owner because the mapped table contains the foreign key
  - fetchType.EAGER to pass the ServicePackage associated to an Order
- **ServicePackage → Order**
  - @OneToMany , not necessary, but implemented for semplicity

Relationship “choice”



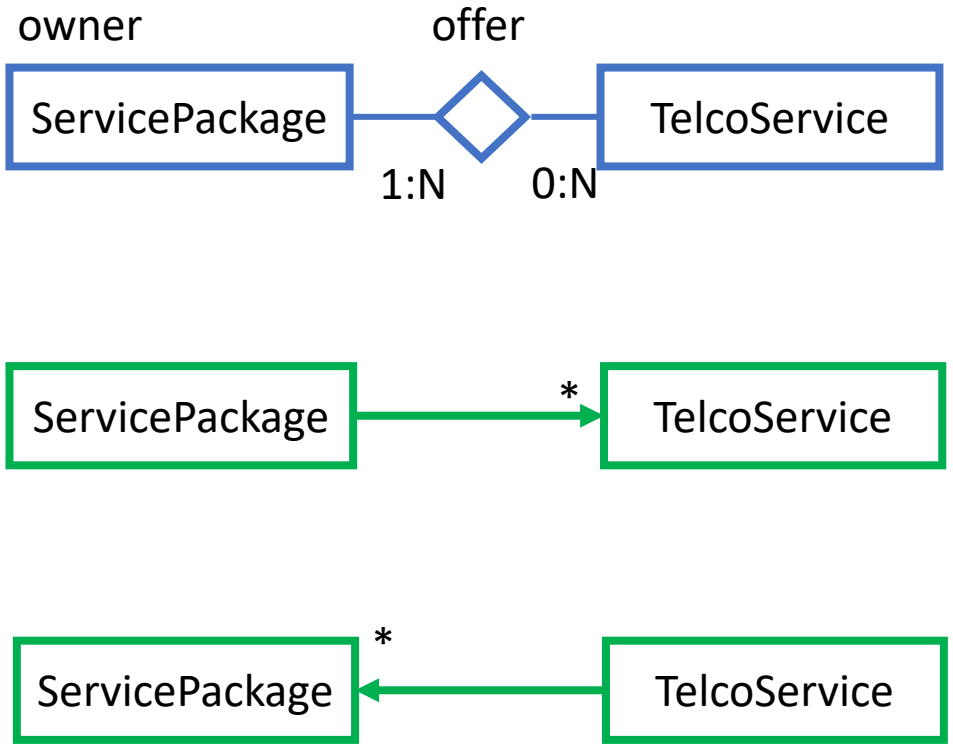
- **Order → OptionalProduct**
  - @ManyToMany, useful to retrieve the OptionalProducts chosen within an Order
  - Owner for conceptual consistency (any of them could have been the owner)
  - fetchType.EAGER to pass all the chosen OptionalProducts when passing the Order
- **OptionalProduct → Order**
  - @ManyToMany, not necessary, but implemented for semplicity

Relationship “choice”



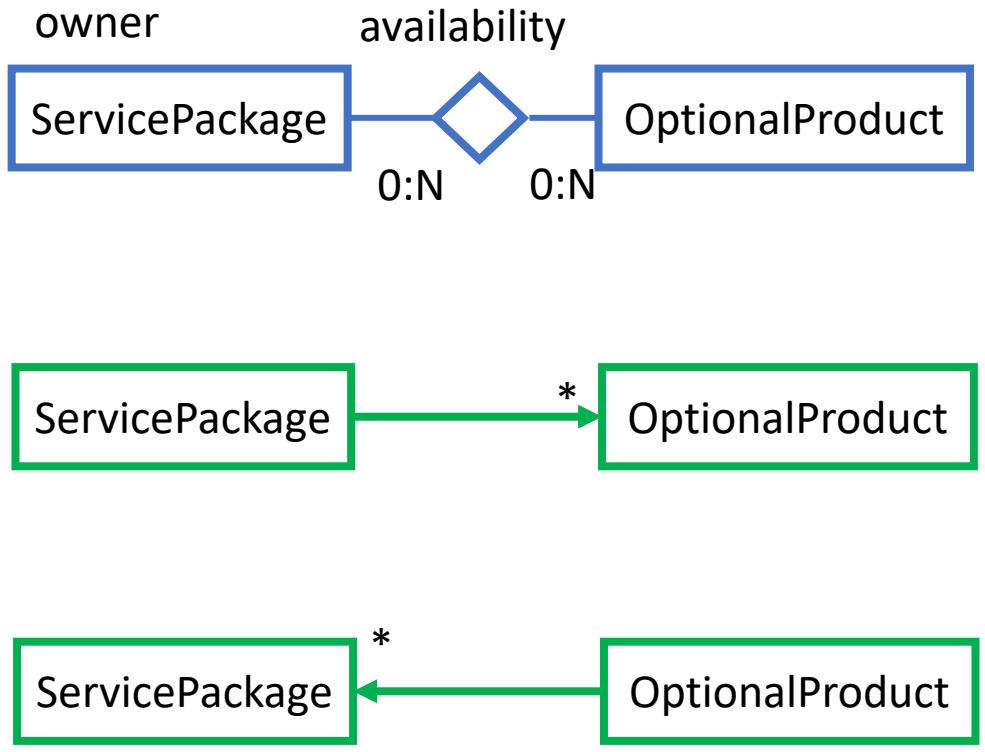
- **Order → ValidityPeriod**
  - @ManyToOne, useful to retrieve the ValidityPeriod chosen within an Order
  - Owner for conceptual consistency (any of them could have been the owner)
  - fetchType.EAGER to pass the chosen ValidityPeriod when passing the Order
- **ValidityPeriod → Order**
  - @OneToMany, not necessary, but implemented for semplicity

Relationship “offer”



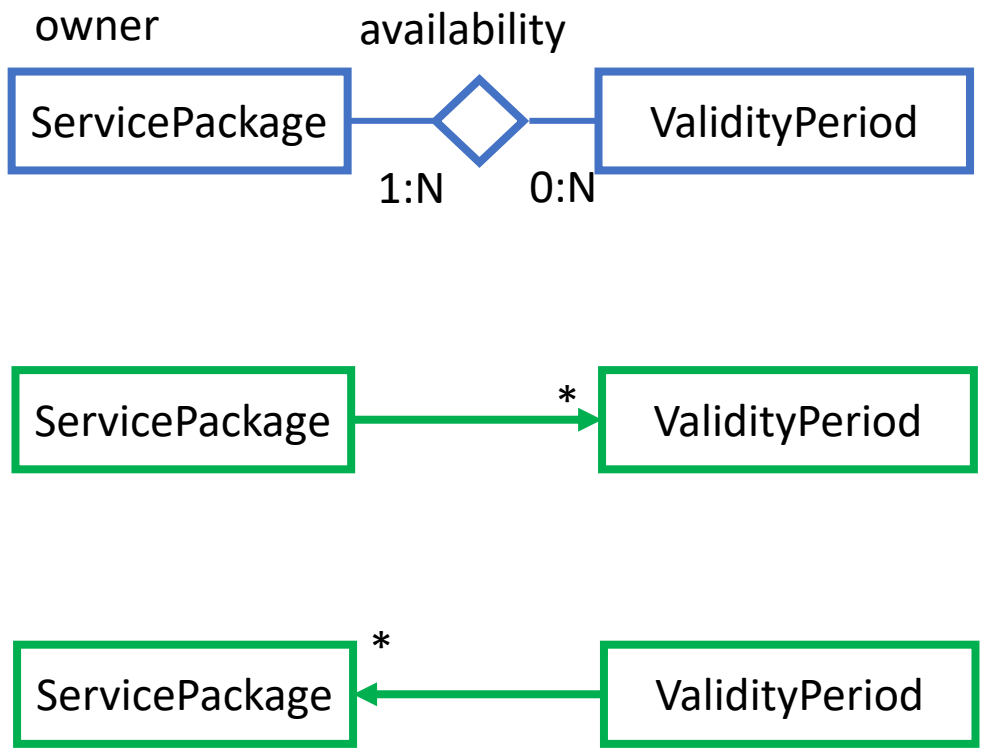
- **ServicePackage → TelcoService**
  - @ManyToMany, useful to retrieve the TelcoServices offered by a ServicePackage
  - Owner for conceptual consistency (any of them could have been owner)
  - fetchType.EAGER to pass all the offered TelcoServices when passing the ServicePackage
- **TelcoService → ServicePackage**
  - @ManyToMany, not necessary, but implemented for simplicity

Relationship “availability”



- **ServicePackage → OptionalProduct**
  - @ManyToMany, useful to retrieve the OptionalProducts available to be bought in pair with a ServicePackage
  - Owner for conceptual consistency (any of them could have been owner)
  - fetchType.EAGER to pass all the available OptionalProducts when passing the ServicePackage
- **OptionalProduct → ServicePackage**
  - @ManyToMany, not necessary, but implemented for semplicity

Relationship “availability”



- **ServicePackage → ValidityPeriod**
  - @ManyToMany, useful to retrieve the ValidityPeriods available to be bought in pair with a ServicePackage
  - Owner for conceptual consistency (any of them could have been owner)
  - fetchType.EAGER to pass all the available ValidityPeriods when passing the ServicePackage
- **ValidityPeriod → ServicePackage**
  - @ManyToMany, not necessary, but implemented for semplicity

Java entities

Entity Order

```
@Entity
@Table(name = "order", schema = "db2_project")
@NamedQuery(name = "Order.getAllRejectedOrders", query = "SELECT o FROM Order o WHERE o.valid = 0")
@NamedQuery(name = "Order.getRejectedOrders", query = "SELECT o FROM Order o WHERE o.user.userId = ?1 AND o.valid = 0")
@NamedQuery(name = "Order.getRejectedOrderById", query = "SELECT o FROM Order o WHERE o.orderId = ?1 AND o.valid = 0")
@NamedQuery(name = "Order.getOrdersByUserId", query = "SELECT o from Order o WHERE o.user.userId = ?1")
@NamedQuery(name = "Order.getAllRejectedOrders", query = "SELECT o FROM Order o WHERE o.valid = 0")
public class Order {
```

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int orderId;
@Column(name = "creationDate", nullable = false)
@Temporal(TemporalType.DATE)
private Date creationDate;
@Column(name = "creationHour", nullable = false)
@Temporal(TemporalType.TIME)
private Time creationHour;
@Column(name = "totalCost_euro", nullable = false)
private float totalCost_euro;
@Column(name = "startDate", nullable = false)
@Temporal(TemporalType.DATE)
private Date startDate;
@Column(name = "valid", nullable = false)
private int valid;
@Column(name = "amountOptionalProducts", nullable = false)
private int amountOptionalProducts;
```

```
@ManyToOne
@JoinColumn(name = "userId")
private User user;
@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "servicePackageId")
private ServicePackage servicePackage;
@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "validityPeriodId")
private ValidityPeriod chosenValidityPeriod;
@ManyToMany(fetch = FetchType.EAGER)
@JoinTable(name="order__optional_product",
      joinColumns = @JoinColumn(name = "orderId", referencedColumnName
= "orderId"),
      inverseJoinColumns = @JoinColumn(name = "optionalProductId",
referencedColumnName = "optionalProductId"))
private Collection<OptionalProduct> chosenOptionalProducts;

//...getter and setter...
```

Entity User

```
@Entity
@Table(name = "user", schema = "db2_project")
@NamedQuery(name = "User.checkCredentials", query = "SELECT u from User u where u.email = ?1 and u.password = ?2")
@NamedQuery(name = "User.findByEmail", query = "SELECT u from User u where  u.email = ?1")
@NamedQuery(name = "User.findByUsername", query = "SELECT u from User u where  u.username = ?1")
@NamedQuery(name = "User.getAllInsolventUsers", query = "SELECT u FROM User u WHERE u.insolvent = 1")
@NamedQuery(name = "User.getUserByOrderId", query = "SELECT u FROM User u, Order o WHERE u.userId = o.user.userId AND o.orderId = ?1")
public class User {
```

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int userId;
@Column(name = "username", unique = true, nullable = false)
private String username;
@Column(name = "email", unique = true, nullable = false)
private String email;
@Column(name = "password", nullable = false)
private String password;
@Column(name = "insolvent", nullable = false)
private int insolvent = 0;
@Column(name = "numOfFailedPayments", nullable = false)
private int numOfFailedPayments = 0;

@OneToMany(mappedBy = "user")
private Collection<Order> orders;
```

```
public User(){
}
public User(String email, String username, String password){
   this.email = email;
   this.username = username;
   this.password = password;
}

public String getDescription(){
   return "User " + username + "(id:" + userId + "), with mail " + email + "
and " + numOfFailedPayments + " failed payments";

}

//...getter and setter...
```

Entity Employee

```
@Entity
@Table(name = "employee", schema = "db2_project")
@NamedQuery(name = "Employee.checkEmployeeCredentials", query = "SELECT e from Employee e where e.email = ?1
and e.password = ?2")
public class Employee {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int employeeId;
   @Column(name = "email", nullable = false)
   private String email;
   @Column(name = "password", nullable = false)
   private String password;

   public Employee() {
   }
   public Employee(String email, String password) {
      this.email = email;
      this.password = password;
   }

//...getter and setter...
```

Entity ServicePackage

```
@Entity
@Table(name = "service_package", schema = "db2_project")
@NamedQuery(name = "ServicePackage.getAllServicePackages", query = "SELECT sp from ServicePackage sp")
@NamedQuery(name = "ServicePackage.getServicePackageByOrderId", query = "SELECT s FROM ServicePackage s, Order o" +
      " WHERE s.servicePackageId = o.servicePackage.servicePackageId AND o.orderId = ?1")
public class ServicePackage {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int servicePackageId;
   @Column(name = "name")
   private String name;

   @OneToMany(mappedBy = "servicePackage")
   private Collection<Order> orders;
   @ManyToMany(fetch = FetchType.EAGER)
   @JoinTable(name="service_package__service",
      joinColumns = @JoinColumn(name = "servicePackageId", referencedColumnName = "servicePackageId"),
      inverseJoinColumns = @JoinColumn(name = "serviceId", referencedColumnName = "serviceId"))
   private Collection<TelcoService> services;
   @ManyToMany(fetch = FetchType.EAGER)
   @JoinTable(name="service_package__optional_product",
         joinColumns = @JoinColumn(name = "servicePackageId", referencedColumnName = "servicePackageId"),
         inverseJoinColumns = @JoinColumn(name = "optionalProductId", referencedColumnName = "optionalProductId"))
   private Collection<OptionalProduct> availableOptionalProducts;
   @ManyToMany(fetch = FetchType.EAGER)
   @JoinTable(name = "service_package__validity_period",
         joinColumns = @JoinColumn(name = "servicePackageId", referencedColumnName = "servicePackageId"),
         inverseJoinColumns = @JoinColumn(name = "validityPeriodId", referencedColumnName = "validityPeriodId"))
   private Collection<ValidityPeriod> availableValidityPeriods;

//...getter and setter...
```

Entity OptionalProduct

```
@Entity
@Table(name = "optional_product", schema = "db2_project")
@NamedQuery(name = "OptionalProduct.getAllOptionalProducts", query = "SELECT o FROM OptionalProduct o")
@NamedQuery(name = "OptionalProduct.getOptionalProductsByPackageId",
      query = "SELECT o FROM OptionalProduct o, ServicePackage sp, ServicePackage.availableOptionalProducts spo WHERE " +
            "o.optionalProductId = spo.optionalProductId AND sp.servicePackageId = ?1")
public class OptionalProduct {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int optionalProductId;
   @Column(name = "name", nullable = false)
   private String name;
   @Column(name = "monthlyFee_euro", nullable = false)
   private float monthlyFee_euro;

   @ManyToMany(mappedBy = "availableOptionalProducts")
   private Collection<ServicePackage> servicePackages;
   @ManyToMany(mappedBy = "chosenOptionalProducts")

   private Collection<Order> orders;

//...getter and setter...
```

Entity ValidityPeriod

```java
@Entity
@Table(name = "validity_period", schema = "db2_project")
@NamedQuery(name = "ValidityPeriod.getAllValidityPeriods", query = "SELECT v FROM ValidityPeriod v")
public class ValidityPeriod {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int validityPeriodId;
   @Column(name = "monthsOfValidity", nullable = false)
   private int monthsOfValidity;
   @Column(name = "monthlyFee_euro", nullable = false)
   private float monthlyFee_euro;

   @ManyToMany(mappedBy = "availableValidityPeriods")
   private Collection<ServicePackage> servicePackages;
   @OneToMany(mappedBy = "chosenValidityPeriod")

   private Collection<Order> orders;

//...getter and setter...
```

Entity TelcoService

```
@Entity
@Table(name = "service", schema = "db2_project")
@Inheritance(strategy = InheritanceType.JOINED)
@NamedQuery(name = "TelcoService.getAllServices", query = "SELECT s FROM TelcoService s")
@NamedQuery(name = "TelcoService.getServicesByPackageId",
      query = "SELECT s FROM TelcoService s, ServicePackage sp, ServicePackage.services sps WHERE " +
            "s.serviceId = sps.serviceId AND sp.servicePackageId = ?1")
abstract public class TelcoService {

   @Id
   private int serviceId;

   @ManyToMany(mappedBy = "services")
   private Collection<ServicePackage> servicePackages;

//...getter...
```

...and subclasses

```java
@Entity
@Table(name = "mobile_phone", schema = "db2_project")
@PrimaryKeyJoinColumn(name = "serviceId", referencedColumnName = "serviceId")
public class MobilePhone extends TelcoService {

    @Column(name = "minutes", nullable = false)
    private int minutes;
    @Column(name = "SMSs", nullable = false)
    private int SMSs;
    @Column(name = "extraMinFee_euro")
    private float extraMinFee_euro;
    @Column(name = "extraSMSFee_euro")
    private float extraSMSFee_euro;


    @Override
    public String getDescription() {
        String description = "You get " + minutes + " minutes and " + SMSs + " SMSs on your
mobile device.";
        return description;
    }
```

```java
@Entity
@Table(name = "fixed_phone", schema = "db2_project")
@PrimaryKeyJoinColumn(name = "serviceId", referencedColumnName = "serviceId")
public class FixedPhone extends TelcoService {

    @Override
    public String getDescription() {
        String description = "You get unlimited calls from home.";
        return description;
    }
```

```java
@Entity
@Table(name = "mobile_internet", schema = "db2_project")
@PrimaryKeyJoinColumn(name = "serviceId", referencedColumnName = "serviceId")
public class MobileInternet extends TelcoService {
    @Column(name = "GBs", nullable = false)
    private int GBs;
    @Column(name = "extraGBFee_euro")
    private float extraGBFee_euro;


    @Override
    public String getDescription() {
        String description = "You get " + GBs + " GBs on your mobile device connection.";
        return description;
    }
```

```java
@Entity
@Table(name = "fixed_internet", schema = "db2_project")
@PrimaryKeyJoinColumn(name = "serviceId", referencedColumnName = "serviceId")
public class FixedInternet extends TelcoService {
    @Column(name = "GBs", nullable = false)
    private int GBs;
    @Column(name = "extraGBFee_euro")
    private float extraGBFee_euro;


    @Override
    public String getDescription() {
        String description = "You get " + GBs + " GBs on your home connection.";
        return description;
    }
```

Components

- Client tier
  - LandingPage.html
  - landingPageManagement.js
  - HomePage.html
  - homePageManagement.js
  - BuyPage.html
  - buyPageManagement.js
  - ConfirmationPage.html
  - confirmationPageManagement.js
  - EmployeeHomePage.html
  - employeeHomePageManagement.js
  - SalesReportPage.html
  - salesReportPageManagement.js
  - utils.js
  - my_style.css

Components

- Web tier
  - CheckLogin
  - RegisterUser
  - Logout
  - GetLoggedUserInfo
  - GetServicePackages
  - GetServicePackagesToBuy
  - GetServices
  - GetOptionalProducts
  - GetValidityPeriods
  - ManageOrder
  - GetRejectedOrders
  - GetRejectedOrdersToComplete
  - CreateServicePackage
  - CreateOptionalProduct
  - GetAggregatedData

Components

All EJBs are stateless!

- Business tier
  - AlertService
    - getAllAlertsDescription()
  - EmployeeService
    - checkEmployeeCredentials(String email, String password)
  - OptionalProductService
    - getOptionalProductById(int optionalProductId)
    - getOptionalProductsByPackage(ServicePackage servicePackage)
    - getAllOptionalProducts()
    - insertOptionalProduct(OptionalProduct optionalProduct)

Components

- Business tier
  - OrderService
    - getAllRejectedOrders()
    - getRejectedOrdersByUserId(int userId)
    - getRejectedOrderById(int orderId)
    - getOrdersByUserId(int userId)
    - createNewOrder (Order newOrder)
    - updateOrder(Order order)
    - makeUserInsolvent(User user, Order order)
    - checkAlert(User user, Order order)
    - getAllRejectedOrdersDescriptions()
    - getOrderDescription(Order order)

Components

- Business tier
  - ServicePackageService
    - getAllClientServicePackages()
    - getServicePackageById(int servicePackageId)
    - insertServicePackage(ServicePackage servicePackage)
    - getServicePackageByOrderId(int orderId)
  - TelcoServiceService
    - getServicesByPackage(ServicePackage servicePackage)
    - getAllServices()
    - getServiceById(int serviceId)
  - ValidityPeriodService
    - getValidityPeriodById(int validityPeriodId)
    - getAllValidityPeriods()

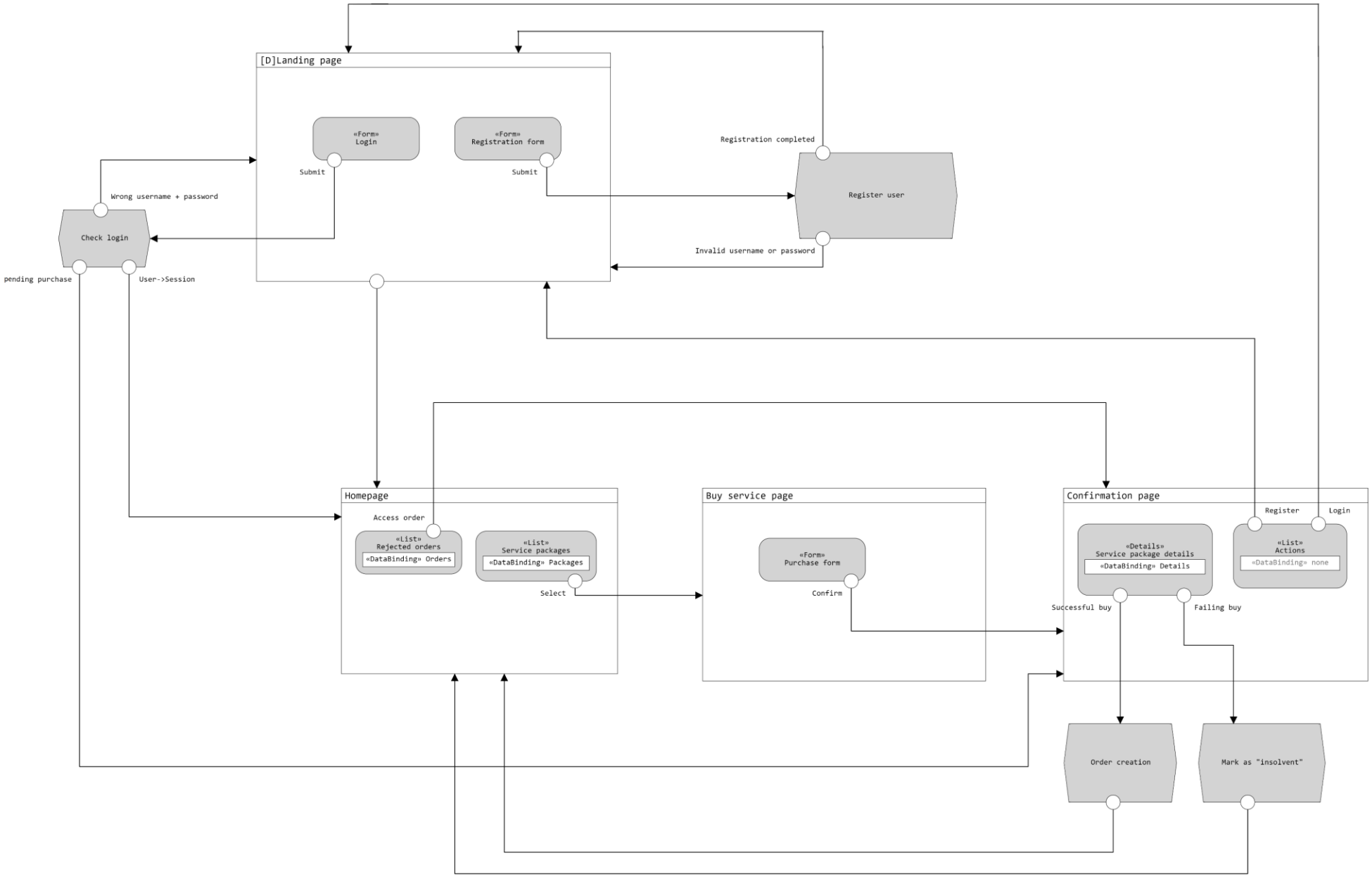
Components

- Business tier
  - UserService
    - checkCredentials(String email, String password)
    - findUserById(int userId)
    - findUserByEmail(String email)
    - findUserByUsername(String username)
    - getAllInsolventUsers()
    - registerUser(String email, String username, String password)
    - getAllInsolventUsersDescriptions()
    - getUserByOrderId(int orderId)

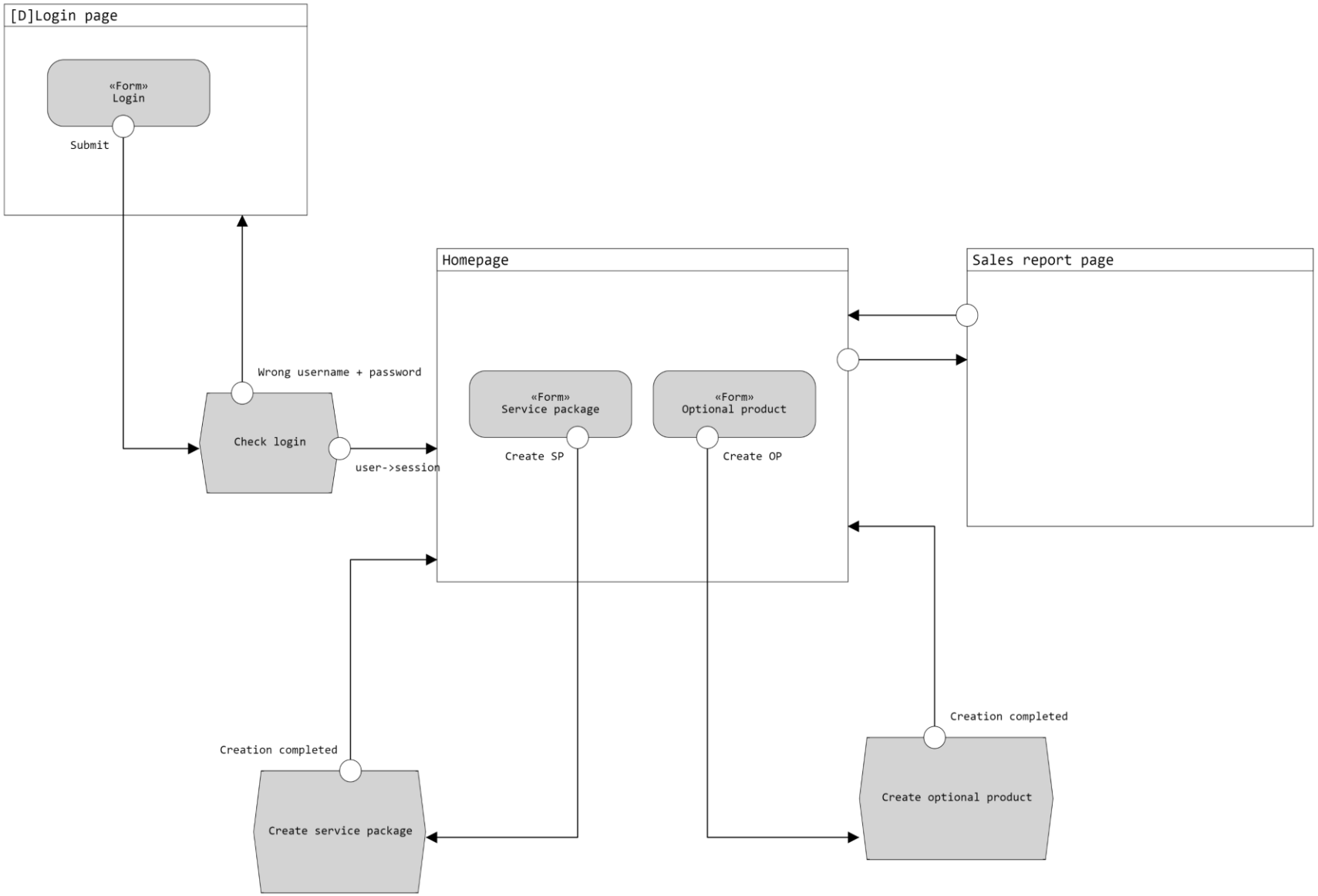
Components

- Business tier
  - MVService
    - getAllAvgAmountOpPerSpDescriptions()
    - getAvgAmountOpPerSpDescription(MVAvgAmountOpPerSp mv)
    - getAllTotalPurchasesPerSpDescriptions()
    - getTotalPurchasesPerSpDescription(MVTotalPurchasesPerSp mv)
    - getAllTotalPurchasesPerSpAndVpDescriptions()
    - getTotalPurchasesPerSpAndVpDescription(MVTotalPurchasesPerSpAndVp mv)
    - getAllTotalValuePerSpDescriptions()
    - getTotalValuePerSpDescription(MVTotalValuePerSp mv)
    - getAllTotalValuePerSpWithOpDescriptions()
    - getTotalValuePerSpWithOpDescription(MVTotalValuePerSpWithOp mv)
    - getBestSellerOpDescription()
    - getBestSellerOpDescription(MVTotalPurchasesPerOp mv)

Application interaction diagram: user

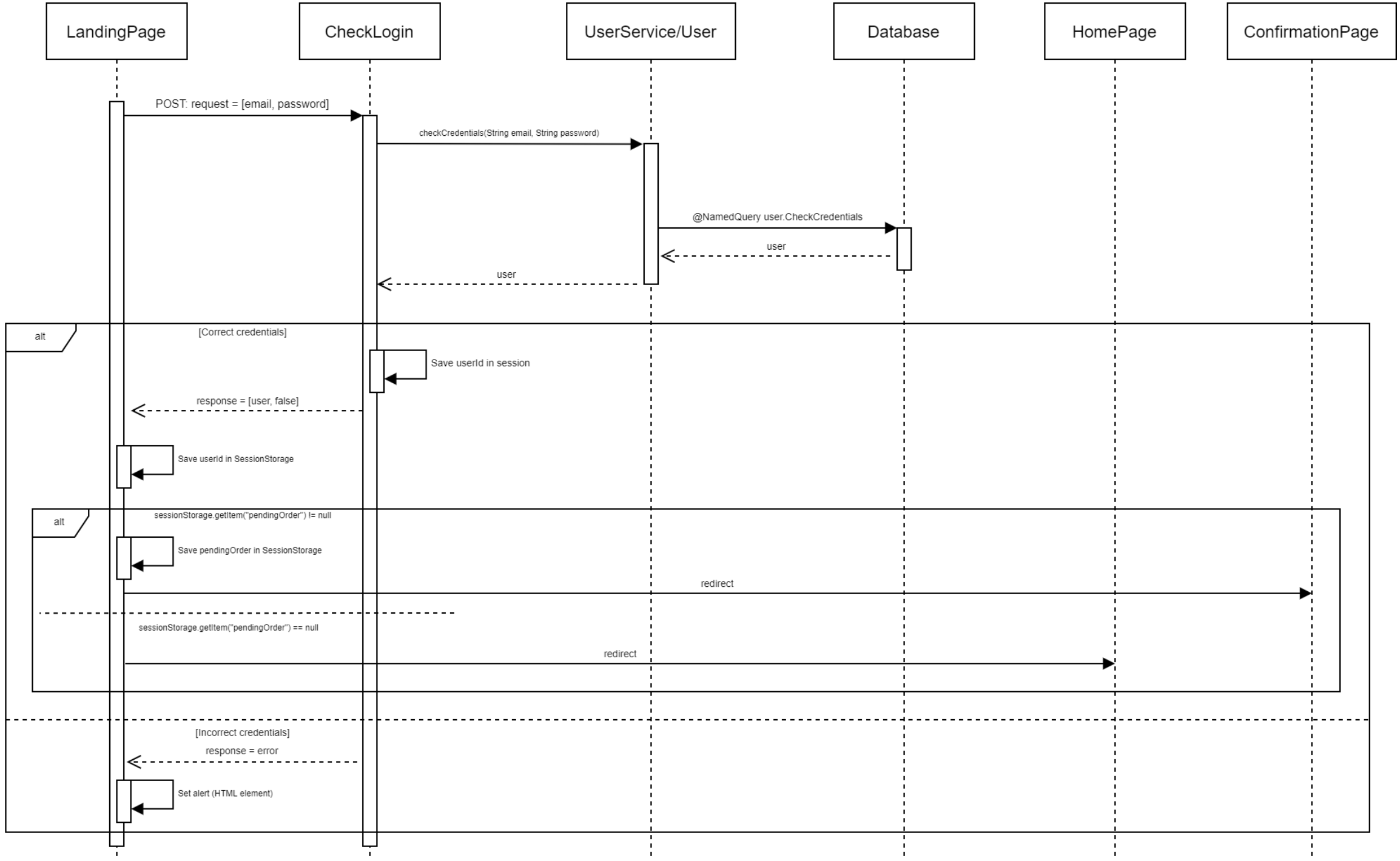


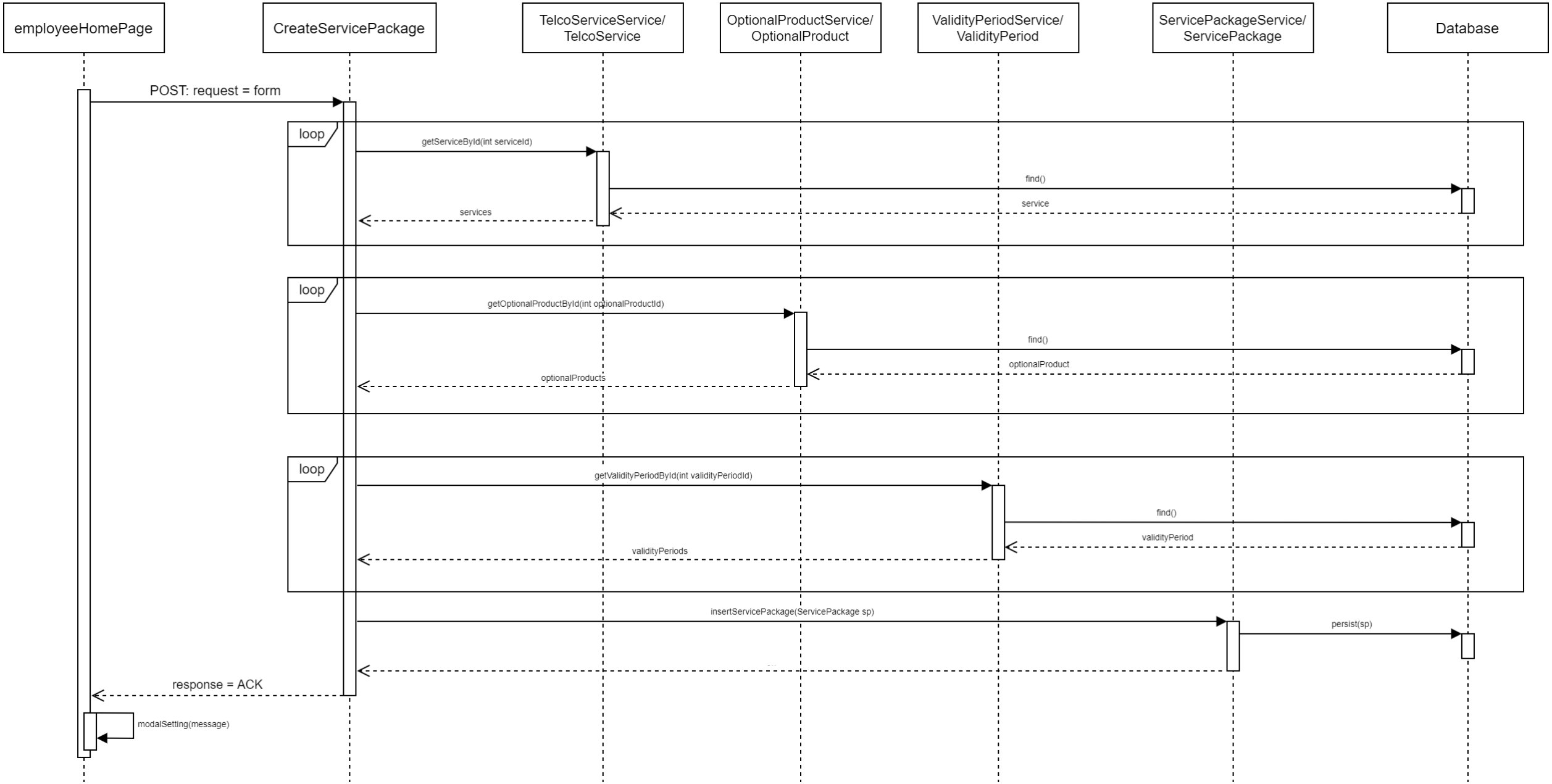
Application interaction diagram: employee

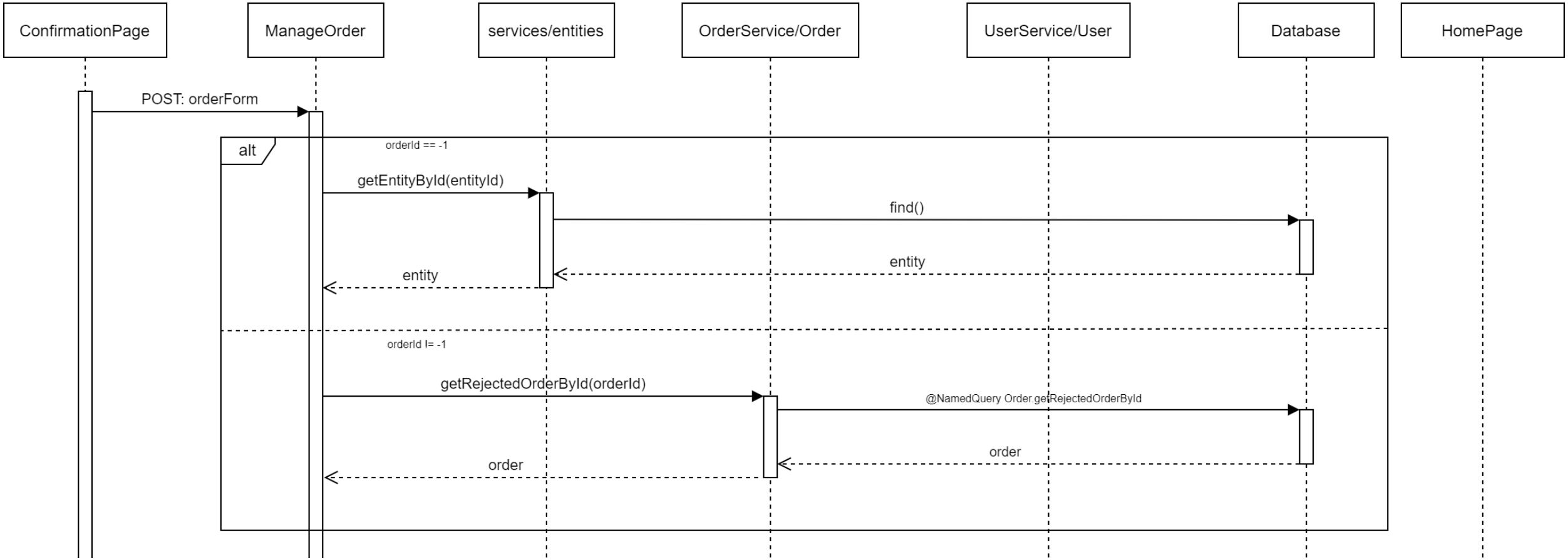


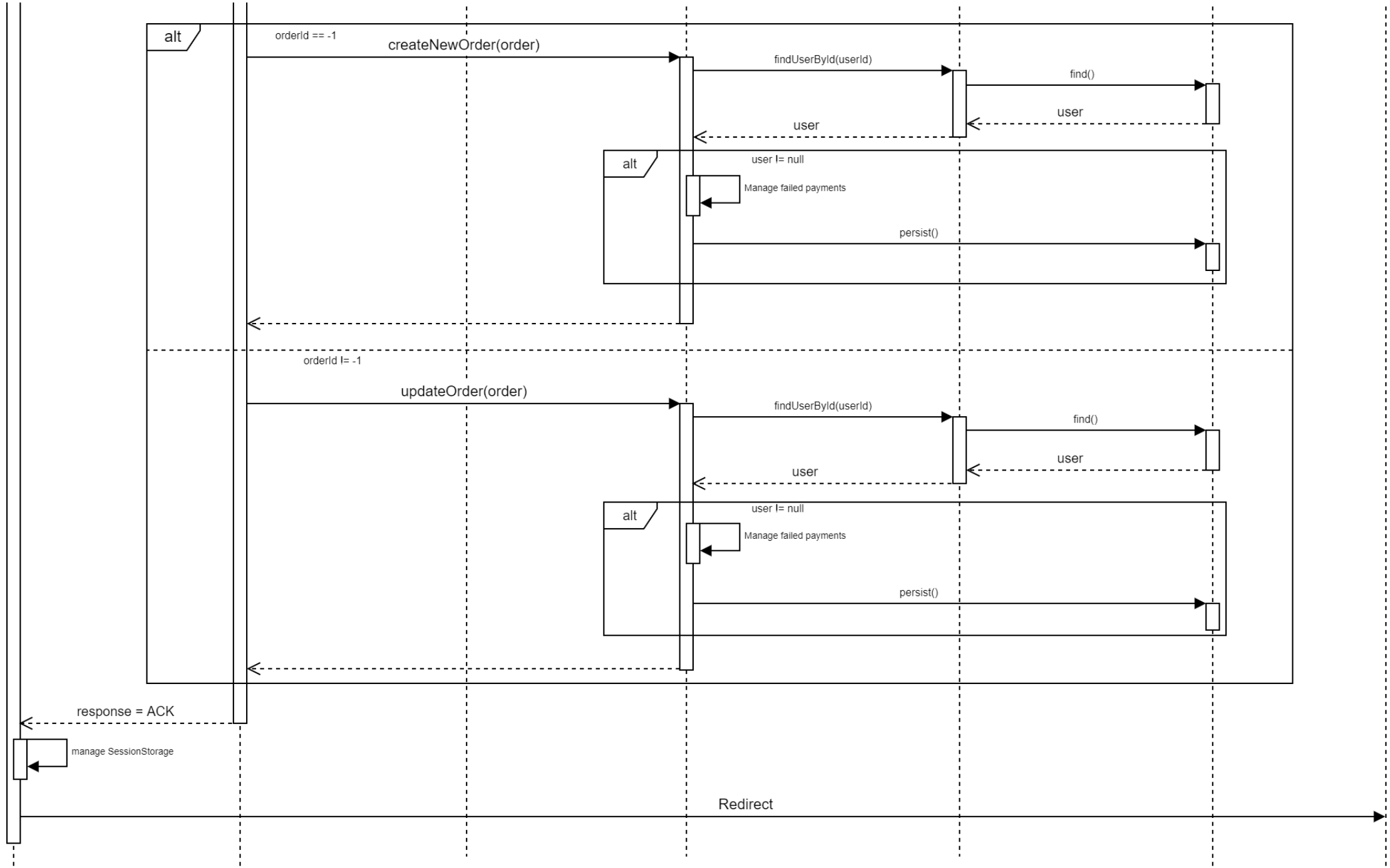
Sequence diagrams

1. Login
2. Service package creation
3. Order creation


LandingPage
CheckLogin
UserService/User
Database
HomePage
ConfirmationPage
POST: request = [email, password]
checkCredentials(String email, String password)
@NamedQuery user.CheckCredentials
user
user
alt
[Correct credentials]
Save userId in session
response = [user, false]
Save userId in SessionStorage
alt
sessionStorage.getItem("pendingOrder") != null
Save pendingOrder in SessionStorage
redirect
sessionStorage.getItem("pendingOrder") == null
redirect
[Incorrect credentials]
response = error
Set alert (HTML element)


employeeHomePage
CreateServicePackage
TelcoServiceService/
TelcoService
OptionalProductService/
OptionalProduct
ValidityPeriodService/
ValidityPeriod
ServicePackageService/
ServicePackage
Database
POST: request = form
loop
getServiceById(int serviceId)
find()
service
services
loop
getOptionalProductById(int optionalProductId)
find()
optionalProduct
optionalProducts
loop
getValidityPeriodById(int validityPeriodId)
find()
validityPeriod
validityPeriods
insertServicePackage(ServicePackage sp)
persist(sp)
response = ACK
modalSetting(message)


ConfirmationPage
ManageOrder
services/entities
OrderService/Order
UserService/User
Database
HomePage
POST: orderForm
alt
orderId == -1
getEntityById(entityId)
find()
entity
entity
orderId != -1
getRejectedOrderById(orderId)
@NamedQuery Order.getRejectedOrderById
order
order


alt
orderId == -1
createNewOrder(order)
findUserById(userId)
find()
user
user
alt
user != null
Manage failed payments
persist()
orderId != -1
updateOrder(order)
findUserById(userId)
find()
user
user
alt
user != null
Manage failed payments
persist()
response = ACK
manage SessionStorage
Redirect