Gibellini Federico - 10669617
Montemurro Elena - 10627677

Documentation related to DB2 project, a.y. 2021-22

1. Specification

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.

2. Database models

2.1 ER model

The Entity-Relationship model is shown in the next page

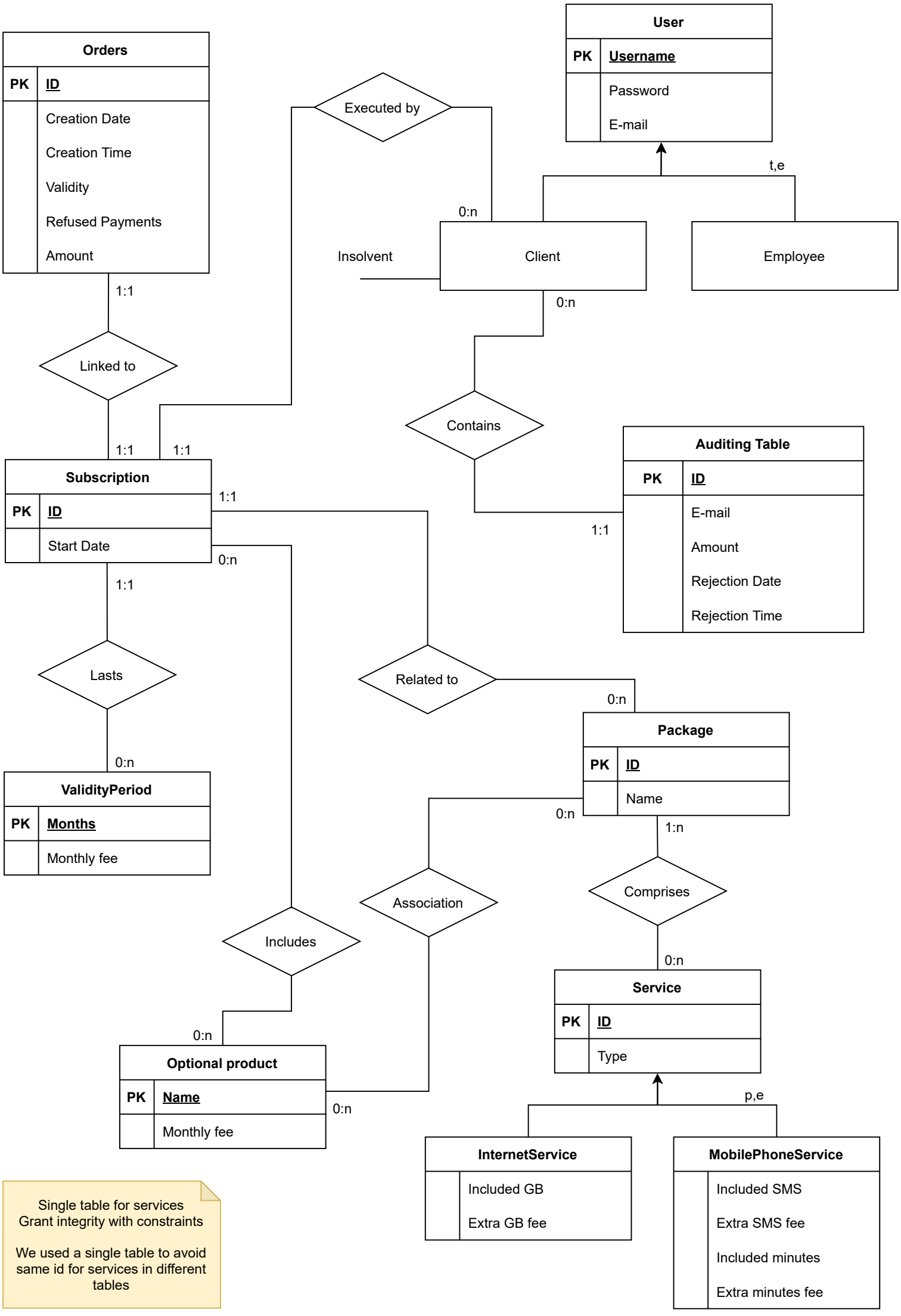
2.2 Logical model

Client (USERNAME, Password, Email, Insolvent)
Employee (USERNAME, Password, Email)
Service (ID, Type, IncludedGB, ExtraGBFee, IncludedSMS, ExtraSMSFee, IncludedMinutes, ExtraMinutesFee)
Package (ID, Name)
ServPack (PACKAGE, SERVICE)
OptionalProduct (NAME, MonthlyFee)
OptionalPack (PACKAGE, PRODUCT)
ValidityPeriod (MONTHS, MonthlyFee)
Subscription (ID, StartDate, User, ValidityPeriod, Package)
OptionalSub (SUBSCRIPTION, PRODUCT)
Orders (ID, CreationDate, CreationTime, Validity, RefusedPayments, Subscription, Amount)
AuditingTable (ID, Username, Email, Amount, RejectionDate, RejectionTime)

Mv_optprod (ID_OPTPROD, TotRevenue)
Mv_package (ID_PACKAGE, Months, PackName, TotPurchase, TotRevenueWoOpt, TotRevenueWOpt, TotOptProd)

Foreign keys:

- AuditingTable(Username) → Client(Username)
- Orders(Subscription) → Subscription(Id)
- Subscription(User) → Client(Username)
- Subscription(ValidityPeriod) → ValidityPeriod(Months)
- Subscription(Package) → Package(Id)
- OptionalPack(Package) → Package(Id)
- OptionalPack(Product) → OptionalProduct(Name)
- OptionalSub(Subscription) → Subscription(Id)
- OptionalSub(Product) → OptionalProduct(Name)
- ServPack(Package) → Package(Id)
- ServPack(Service) → Service(Id)
- Mv_optprod(Id_optprod) → OptionalProduct(Name)
- Mv_package(Id_package) → Package(Id)
- Mv_package(Months) → ValidityPeriod(Months)


User
PK
Username
Password
E-mail
Orders
PK
ID
Creation Date
Creation Time
Validity
Refused Payments
Amount
Executed by
t,e
0:n
Insolvent
Client
Employee
0:n
1:1
Linked to
Contains
1:1
1:1
Auditing Table
PK
ID
E-mail
Amount
Rejection Date
Rejection Time
Subscription
PK
ID
Start Date
1:1
1:1
0:n
1:1
Lasts
Related to
0:n
Package
PK
ID
Name
0:n
1:n
0:n
ValidityPeriod
PK
Months
Monthly fee
Association
Comprises
Includes
0:n
Service
PK
ID
Type
0:n
Optional product
PK
Name
Monthly fee
0:n
p,e
InternetService
Included GB
Extra GB fee
MobilePhoneService
Included SMS
Extra SMS fee
Included minutes
Extra minutes fee
Single table for services
Grant integrity with constraints
We used a single table to avoid same id for services in different tables

3. Additional hypotheses

- E-mails can't be changed
- No pair of users can have the same username: no client can have the username of an employee
- Service packages cannot change name
- No pair of service packages can have the same name
- All service packages can be purchased with all validity periods
- Registration can happen only for customers
- As all packages have the same fees for the same periods, monthly fees are related to periods and they are independent from the package
- Fees don't vary in time neither for the packages nor for the optional products
- Each subscription leads to the creation of a single order; payment failures do not create other orders related to the same subscription but increase the number of refused payments for a specific order
- It is not necessary to keep track of the history of failed payments but only of orders which haven't been correctly paid yet; all orders should eventually get paid
- Orders are always created as not valid and with no failed payments; upon payment one of these two values gets updated
- Values in materialized views are affected only by orders which have been correctly paid
- Auditing table alerts are raised if a user has at least 3 failed payments on the orders that are still not valid
- Each payment failure after the third triggers an alert in the auditing table
- For the auditing table and package materialized view write operations are less frequent than read ones; thence it's more convenient to introduce replicated fields than letting the DBMS join with other tables at each read request
- Service activation schedule is useless in the database as all its fields are already saved in other tables (subscription, orders); deactivation date can be easily obtained at runtime on the server knowing the activation date and the validity period. Service activation schedule can simply be shown on the web browser

4. Database schema (DDL)

4.1 General tables

Client

```
CREATE TABLE `client` (
  `Username` varchar(45) NOT NULL,
  `Password` varchar(45) NOT NULL,
  `Email` varchar(100) NOT NULL,
  `Insolvent` tinyint NOT NULL DEFAULT '0',
  PRIMARY KEY (`Username`),
  CONSTRAINT `insolventBoolean` CHECK (((`Insolvent` = 1) or (`Insolvent` = 0)))
)
```

Employee

```
CREATE TABLE `employee` (
  `Username` varchar(45) NOT NULL,
  `Password` varchar(45) NOT NULL,
  `Email` varchar(100) NOT NULL,
  PRIMARY KEY (`Username`))
```

Service

```
CREATE TABLE `service` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Type` varchar(45) NOT NULL,
  `IncludedGB` int DEFAULT NULL,
  `ExtraGBFee` double DEFAULT NULL,
  `IncludedMinutes` int DEFAULT NULL,
  `ExtraMinutesFee` double DEFAULT NULL,
  `IncludedSMS` int DEFAULT NULL,
  `ExtraSMSFee` double DEFAULT NULL,
  PRIMARY KEY (`ID`),
  CONSTRAINT `fixed_phone_features` CHECK ((((`Type` = _utf8mb4'Fixed Phone') and (`IncludedGB` is null) and (`ExtraGBFee` is null) and (`IncludedMinutes` is null) and (`ExtraMinutesFee` is null) and (`IncludedSMS` is null) and (`ExtraSMSFee` is null)) or (`Type` <> _utf8mb4'Fixed Phone'))),
  CONSTRAINT `internet_features` CHECK (((((`Type` = _utf8mb4'Fixed Internet') or (`Type` = _utf8mb4'Mobile Internet')) and (`IncludedGB` is not null) and (`ExtraGBFee` is not null) and (`IncludedMinutes` is null) and (`ExtraMinutesFee` is null) and (`IncludedSMS` is null) and (`ExtraSMSFee` is null)) or ((`Type` <> _utf8mb4'Fixed Internet') and (`Type` <> _utf8mb4'Mobile Internet')))),
  CONSTRAINT `mobile_phone_features` CHECK ((((`Type` = _utf8mb4'Mobile Phone') and (`IncludedGB` is null) and (`ExtraGBFee` is null) and (`IncludedMinutes` is not null) and (`ExtraMinutesFee` is not null) and (`IncludedSMS` is not null) and (`ExtraSMSFee` is not null)) or (`Type` <> _utf8mb4'Mobile Phone'))),
  CONSTRAINT `positive_GB_minutes_SMS` CHECK (((`IncludedGB` >= 0) and (`ExtraGBFee` >= 0) and (`IncludedMinutes` >= 0) and (`ExtraMinutesFee` >= 0) and (`IncludedSMS` >= 0) and (`ExtraSMSFee` >= 0))),
  CONSTRAINT `service_type` CHECK ((`Type` in (_utf8mb4'Fixed Phone',_utf8mb4'Mobile Phone',_utf8mb4'Fixed Internet',_utf8mb4'Mobile Internet')))
)
```

Package

```
CREATE TABLE `package` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Name` varchar(45) NOT NULL,
  PRIMARY KEY (`ID`)
)
```

ServPack

```
CREATE TABLE `servpack` (
  `Package` int NOT NULL,
  `Service` int NOT NULL,
  PRIMARY KEY (`Package`,`Service`),
  KEY `ServicePack_idx` (`Service`),
  CONSTRAINT `PackService` FOREIGN KEY (`Package`) REFERENCES `package` (`ID`) ON UPDATE CASCADE,
  CONSTRAINT `ServicePack` FOREIGN KEY (`Service`) REFERENCES `service` (`ID`) ON UPDATE CASCADE
)
```

ValidityPeriod

```
CREATE TABLE `validityperiod` (
  `Months` int NOT NULL,
  `MonthlyFee` int NOT NULL,
  PRIMARY KEY (`Months`),
  CONSTRAINT `positive_monthly_fee` CHECK ((`MonthlyFee` > 0)),
  CONSTRAINT `positive_months` CHECK ((`Months` > 0))
)
```

OptionalProduct

```
CREATE TABLE `optionalproduct` (
  `Name` varchar(45) NOT NULL,
  `MonthlyFee` int NOT NULL,
  PRIMARY KEY (`Name`),
  CONSTRAINT `monthlyFeePositiveOrNull` CHECK ((`MonthlyFee` >= 0))
)
```

OptionalPack

```
CREATE TABLE `optionalpack` (
  `Package` int NOT NULL,
  `Product` varchar(45) NOT NULL,
  PRIMARY KEY (`Package`,`Product`),
  KEY `ProductPack_idx` (`Product`),
  CONSTRAINT `PackageOpt` FOREIGN KEY (`Package`) REFERENCES `package` (`ID`) ON UPDATE CASCADE,
  CONSTRAINT `ProductPack` FOREIGN KEY (`Product`) REFERENCES `optionalproduct` (`Name`) ON UPDATE
CASCADE
)
```

Subscription

```
CREATE TABLE `subscription` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `StartDate` date NOT NULL,
  `User` varchar(45) NOT NULL,
  `ValidityPeriod` int NOT NULL,
  `Package` int NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `User_idx` (`User`),
  KEY `ValidityPeriod_idx` (`ValidityPeriod`),
  KEY `Package_idx` (`Package`),
  CONSTRAINT `Package` FOREIGN KEY (`Package`) REFERENCES `package` (`ID`),
  CONSTRAINT `User` FOREIGN KEY (`User`) REFERENCES `client` (`Username`) ON UPDATE CASCADE,
  CONSTRAINT `ValidityPeriod` FOREIGN KEY (`ValidityPeriod`) REFERENCES `validityperiod` (`Months`)
)
```

OptionalSub

```
CREATE TABLE `optionalsub` (
  `Subscription` int NOT NULL,
  `Product` varchar(45) NOT NULL,
  PRIMARY KEY (`Subscription`,`Product`),
  KEY `Product_idx` (`Product`),
  CONSTRAINT `Product` FOREIGN KEY (`Product`) REFERENCES `optionalproduct` (`Name`) ON UPDATE
CASCADE,
  CONSTRAINT `SubscriptionOpt` FOREIGN KEY (`Subscription`) REFERENCES `subscription` (`ID`) ON UPDATE
CASCADE
)
```

Orders

```
CREATE TABLE `orders` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `CreationDate` date NOT NULL,
  `CreationTime` time NOT NULL,
  `Validity` tinyint NOT NULL DEFAULT '0',
  `RefusedPayments` int NOT NULL DEFAULT '0',
  `Subscription` int NOT NULL,
  `Amount` int NOT NULL DEFAULT '0',
  PRIMARY KEY (`ID`),
  KEY `Subscription_idx` (`Subscription`),
  CONSTRAINT `Subscription` FOREIGN KEY (`Subscription`) REFERENCES `subscription` (`ID`) ON UPDATE
CASCADE
)
```

AuditingTable

```
CREATE TABLE `auditingtable` (
  `ID` int NOT NULL AUTO_INCREMENT,
  `Username` varchar(45) NOT NULL,
  `Email` varchar(45) NOT NULL,
  `Amount` int NOT NULL DEFAULT '0',
  `RejectionDate` date NOT NULL,
  `RejectionTime` time NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `audit:username_idx` (`Username`),
  CONSTRAINT `audit:username` FOREIGN KEY (`Username`) REFERENCES `client` (`Username`) ON UPDATE
CASCADE
)
```

4.2 Materialized views

Mv_package

```
CREATE TABLE `mv_package` (
  `id_package` int NOT NULL,
  `months` int NOT NULL,
  `packName` varchar(45) NOT NULL,
  `totPurchase` int NOT NULL DEFAULT '0',
  `totRevenueWoOpt` int NOT NULL DEFAULT '0',
  `totRevenueWOpt` int NOT NULL DEFAULT '0',
  `totOptProd` int NOT NULL DEFAULT '0',
  PRIMARY KEY (`id_package`,`months`),
  KEY `months_idx` (`months`),
  CONSTRAINT `id_package` FOREIGN KEY (`id_package`) REFERENCES `package` (`ID`) ON UPDATE
CASCADE,
  CONSTRAINT `months` FOREIGN KEY (`months`) REFERENCES `validityperiod` (`Months`) ON UPDATE
CASCADE
)
```

Mv_optprod

```
CREATE TABLE `mv_optprod` (
  `id_optprod` varchar(45) NOT NULL,
  `tot_revenue` int NOT NULL DEFAULT '0',
  PRIMARY KEY (`id_optprod`),
  CONSTRAINT `id_optprod` FOREIGN KEY (`id_optprod`) REFERENCES `optionalproduct` (`Name`) ON
UPDATE CASCADE
)
```

4.3 Materialized views description

Mv_package is a materialized view that keeps track of relevant information about package purchases. Each row is related to a specific package and validity period. Here is a brief description of the fields:

- packName is the name of the package
- totPurchase stands for the number of sales that have been done.
- totRevenueWoOpt represents the total revenue of sales without optional products.
- totRevenueWOpt represents the total revenue of sales including optional products.
- totOptProd stands for the number of optional products that have been sold together with that package and for that validity period.

Mv_optprod is a materialized view that registers the revenue related to an optional product. In this table there is the name of the optional product and the associated total income.

5. Triggers

5.1 General triggers

Triggers to disallow registering users with the same identifier

Event: insertion of a new client or a new employee
Condition: a user with the same username as the new one already exists
Action: throw error message to rollback insertion

```
create trigger client_not_employee_tr before insert on client
for each row
begin
        if exists (select *
                from employee
                where Username = NEW.Username)
        then SIGNAL sqlstate '45000' set message_text = "Error: Existing employee with same name";
        end if;
end;

create trigger employee_not_client_tr before insert on employee
for each row
begin
        if exists (select *
                from client
                where Username = NEW.Username)
        then SIGNAL sqlstate '45000' set message_text = "Error: Existing client with same name";
        end if;
end;
```

Trigger to disallow creation of packages with the same name

Event: insertion of a new package
Condition: a package with the same name as the new one already exists
Action: throw error message to rollback insertion

```
create trigger unique_package_name_tr before insert on package
for each row
begin
        if exists (select *
                from package
                where Name = NEW.Name)
        then SIGNAL sqlstate '45000' set message_text = "Error: Existing package with same name";
        end if;
end;
```

Trigger to set user insolvent upon payment failure

Event: update on orders
Condition: new order is not valid and number of refused payments is greater than or equal to 1
Action: set related client as insolvent

```
create trigger insolvent_user_tr after update on orders
for each row
begin
        if NEW.Validity = 0 and NEW.refusedPayments >= 1
        then update client set Insolvent = 1
        where Username = (select subscription.User
                                from subscription, orders
                                where subscription.ID = orders.Subscription and orders.ID = NEW.ID);
        end if;
end;
```

Trigger to set user uninsolvent upon completion of all payments

Event: update on orders
Condition: new order is valid and after update no invalid orders exist for related client
Action: set related user as uninsolvent (insolvent = 0)

```
create trigger uninsolvent_user_tr after update on orders
for each row
begin
        if NEW.Validity = 1 and
        not exists (select *
                    from subscription, orders, client
                    where subscription.ID = orders.Subscription and orders.ID != NEW.ID and
                          client.Username = subscription.User and client.Username =
                         (select subscription.User
                          from subscription, orders
                          where subscription.ID = orders.Subscription and orders.ID = NEW.ID)
                          and orders.Validity = 0)
        then update client set Insolvent = 0
        where Username = (select subscription.User
                                from subscription, orders
                                where subscription.ID = orders.Subscription and orders.ID = NEW.ID);
        end if;
end;
```

Trigger to disallow assignment of multiple same-type services to a single package

Event: insertion of a tuple on servPack (linking of a service to a package)
Condition: a different service of the same type as the new one is already linked to the same package
Action: throw error message to rollback insertion

```
create trigger single_type_services_tr before insert on servpack
for each row
begin
        if exists (
                select *
                from servpack AS sp, service AS s
                where sp.Service = s.ID and sp.Package = NEW.Package
                            and s.type = (select Type
                                          from service
                                          where ID=NEW.Service))
        then SIGNAL sqlstate '45000' set message_text = "Error: More than a service with the same type";
        end if;
end;
```

Trigger to create order upon creation of a subscription

Event: insertion of a new subscription
Condition: true
Action: insert in orders a new tuple related to newly-inserted subscription

```
create trigger new_order_tr after insert on subscription
for each row
begin
        set @amount = NEW.ValidityPeriod *(select MonthlyFee
                                           from validityperiod
                                           where Months = NEW.ValidityPeriod);
        insert into orders (CreationDate, CreationTime, Validity, RefusedPayments, Subscription, Amount)
        values (current_date(), current_time(), 0, 0, NEW.id, @amount);
end;
```

Trigger to create order upon creation of a subscription

Event: insertion of a tuple in optionalsub (linking of an optional product to a subscription)
Condition: true
Action: update related order's amount with cost of optional product in newly-inserted tuple

```
create trigger amount_opt_tr after insert on optionalsub
for each row
begin
        set @vp = (select ValidityPeriod from subscription where ID = NEW.Subscription);
        update orders set Amount = Amount + @vp *(select MonthlyFee
                                                  from optionalproduct
                                                  where Name = NEW.Product)
        where subscription = NEW.Subscription;
 end;
```

Trigger to create alerts in auditing table

Event: update on orders
Condition: order is invalid, new number of refused payments is greater than the old one and the number of cumulative refused payments for related user is greater than or equal to 3
Action: create row for auditing table

```
create trigger audit_UPDATE_tr after update on orders
for each row
begin
        set @us = (select subscription.User
                   from subscription, orders
                   where subscription.ID = orders.Subscription and orders.ID = NEW.ID);
        set @email = (select email from client where username = @us);
        if (NEW.Validity = 0 and NEW.RefusedPayments > old.RefusedPayments and
             (select sum(orders.RefusedPayments)
              from subscription, orders
              where subscription.ID = orders.Subscription and subscription.User = @us
                    and orders.validity = 0) >= 3)
        then set @amount = (select sum(orders.Amount)
                            from subscription, orders
                            where subscription.ID=orders.Subscription and subscription.User = @us
                                  and orders.validity = 0);
        insert into auditingtable (Username, Email, Amount, RejectionDate, RejectionTime)
        values (@us, @email, @amount, curdate(), curtime());
        end if;
end;
```

5.2 Materialized views triggers

Trigger to update content of mv_package

Event: update on orders
Condition: new order is valid while the old one wasn't
Action: update related row in mv_package after creating new one if not present

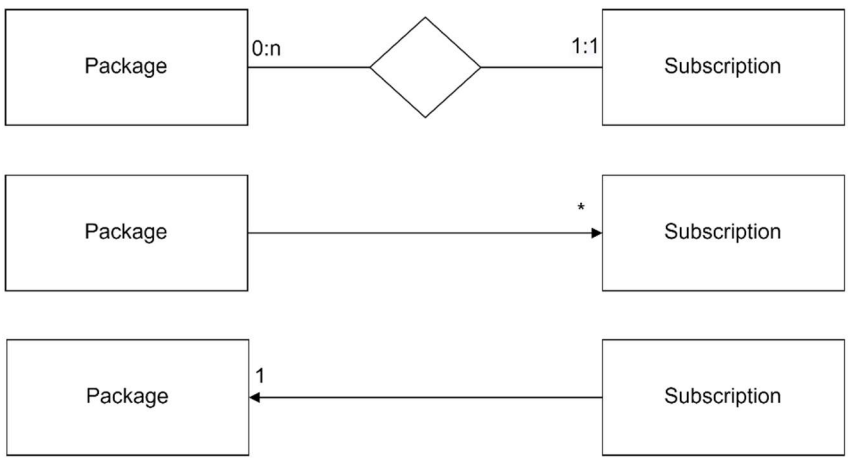
```
create trigger mv_package_tr after update on orders
for each row
begin
        #declare cursors
        declare done int default false;
        declare product varchar(45);
        declare cur_prod cursor for (select optionalsub.product from optionalsub
                                    where NEW.subscription=optionalsub.subscription);
        declare continue handler for not found set done = true;



        #update on change of validity
        if (NEW.validity = 1 and old.validity = 0) then
                set @pack = (select package from subscription where id = NEW.subscription);
                set @vp = (select validityPeriod from subscription where id=NEW.subscription);
                set @amount = @vp* (select monthlyFee from validityperiod where months =  @vp);

        #insert NEW row if not present
        if not exists (select * from mv_package where id_package = @pack and months = @vp)
        then insert into mv_package
                 values (@pack, @vp, (select name from package where id = @pack), 0, 0, 0, 0);
        end if;

        #Update totRevenueWoOpt
        update mv_package
        set totRevenueWoOpt=totRevenueWoOpt+@amount
        where id_package=@pack and months =@vp;

        #Update totRevenueWOpt
        update mv_package
        set totRevenueWOpt = totRevenueWOpt+@amount
        where id_package = @pack and months = @vp;

        #Update totPurchase
        update mv_package
        set totPurchase=totPurchase+1
        where id_package=@pack and months=@vp;

        #open cursor
        open cur_prod;

        #execute cycle
        prod_loop: loop
                fetch cur_prod into product;

                #exit condition
                if done then
                        leave prod_loop;
                end if;

                #Update totRevenueWOpt
                update mv_package set totRevenueWOpt = (totRevenueWOpt +
                        (select monthlyFee
                         from optionalproduct
                         where Name = product) * @vp)
                where id_package = @pack and months = @vp;

                #Update totOptProd
                update mv_package set totOptProd = totOptProd + 1
                        where id_package = @pack and months = @vp;
                end loop;
        end if;
end;
```

Trigger to update content of mv_optprod

Event: update on orders
Condition: new order is valid while the old one wasn't
Action: update related row in mv_optprod after creating new one if not present

```
create trigger mv_opt_prod_revenue_tr after update on orders
for each row
begin
        #declare cursors
        declare done int default false;
        declare product varchar(45);
        declare cur_prod cursor for (select optionalsub.product from optionalsub
                                        where NEW.subscription = optionalsub.subscription);
        declare continue handler for not found set done = true;

        #update on change of validity
        if (NEW.validity = 1 and old.validity = 0) then

                #open cursor
                open cur_prod;

                #execute cycle
                prod_loop: loop
                        fetch cur_prod into product;

                        #exit condition
                        if done then
                                leave prod_loop;
                        end if;

                        #insert NEW row if not present
                        if product not in (select id_optprod from mv_optprod)
                        then insert into mv_optprod values (product, 0);
                        end if;

                        update mv_optprod set tot_revenue = (tot_revenue +
                                (select monthlyFee
                                 from optionalproduct
                                 where Name = product) *
                                (select ValidityPeriod
                                 from subscription
                                 where subscription.id=NEW.Subscription))
                        where id_optprod = product;
                end loop;
        end if;
end;
```

6 ORM mapping

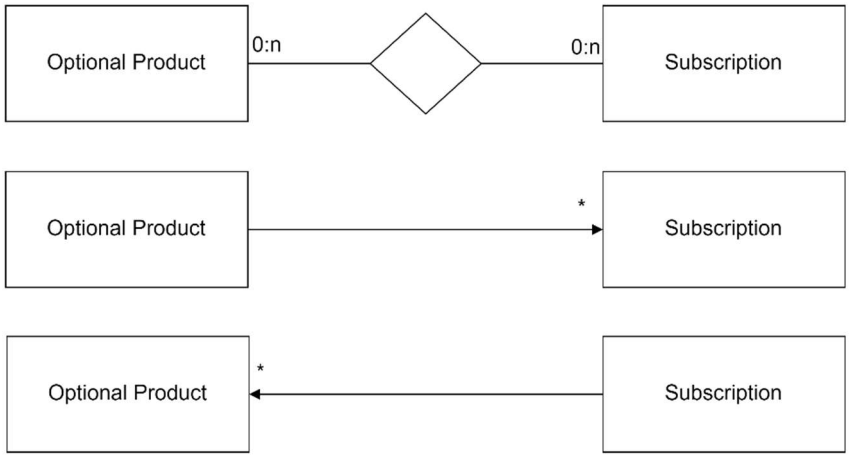


Package->Subscription

@OneToMany. It's not necessary because we don't need to get all the subscriptions containing the package.

Subscription->Package

@ManyToOne. It's necessary because we need to retrieve the package related to the subscription and to fetch it eagerly. When we show the user the subscription details, we also display the package included in it.

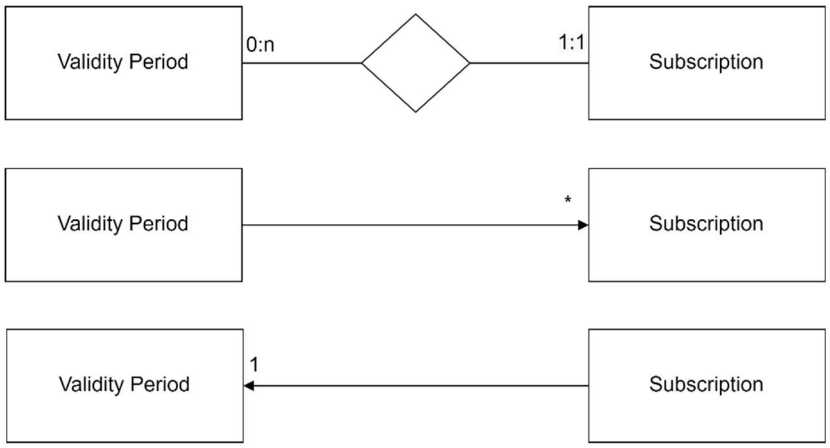


Optional product -> Subscription

@ManyToMany. It's not necessary because we don't require all the subscriptions associated to the optional product.

Subscription-> Optional product

@ManyToMany. It's necessary because we want all the optional products related to the subscription. We fetch it eagerly because, when we show the subscription details, we have to include also the optional products.

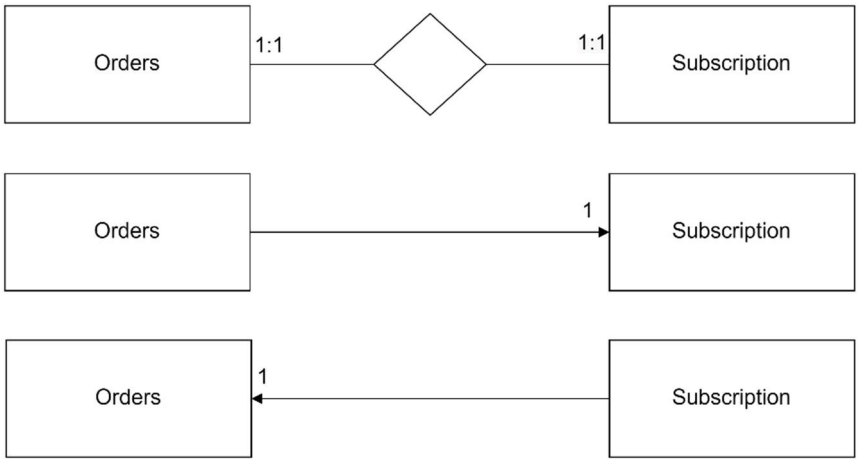


Validity period->Subscription

@OneToMany. It's not necessary because we don't need to get all the subscriptions with the same validity period.

Subscription->Validity period

@ManyToOne. It's necessary because we need to retrieve the validity period related to the subscription. Like in the previous relationship we retrieve it eagerly because it's needed to calculate the amount of subscription for the confirmation page.

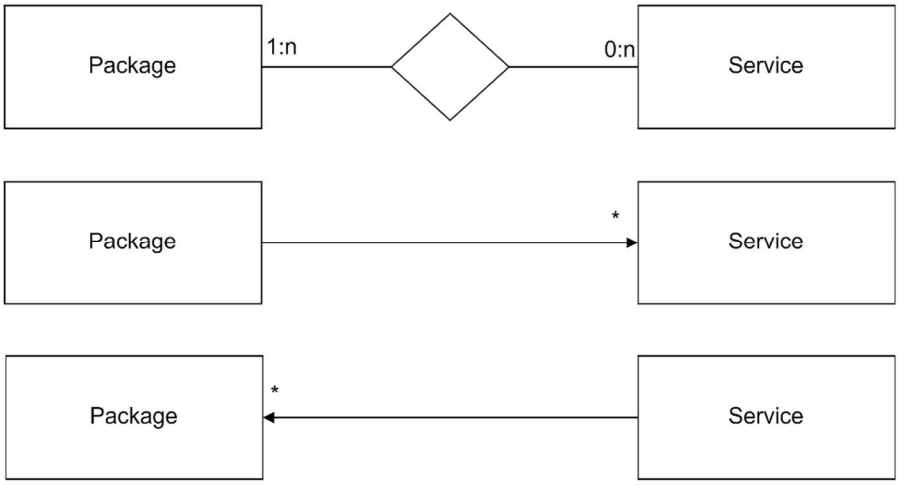


Order->Subscription

@OneToOne it's necessary because we want to get the Subscription associated to the Order. Fetch type: eager; when we show the order, we also display data contained in the subscription.

Subscription->Order

@OneToOne it's not necessary because we don't have to get the Order from the Subscription.

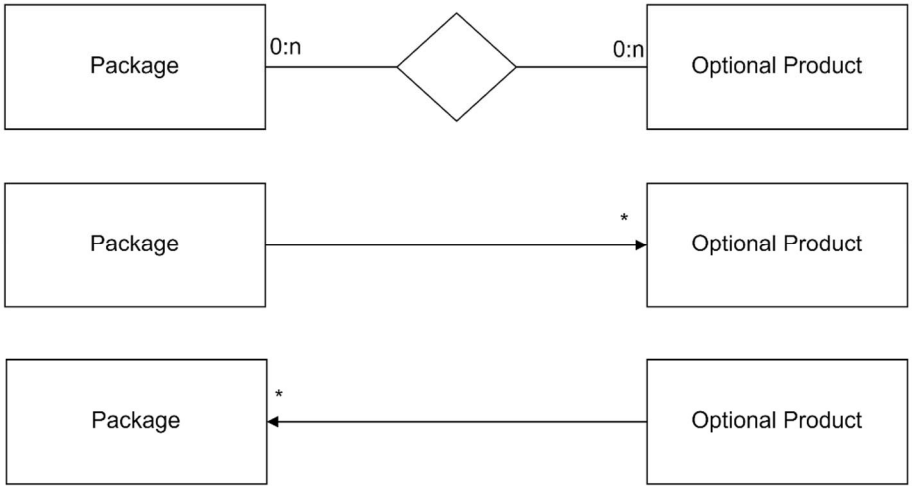


Package -> Service

@ManyToMany. It's necessary because we must get all the services linked to the package. Fetch type: Eager. When we want to show the package, we must also get the services related to it and display them.

Service-> Package

@ManyToMany. It's not necessary because we don't need to retrieve all packages containing the same service.



Package-> Optional Product

@ManyToMany. It's necessary because we must get all the optional products related to the package. Fetch type: Eager. When we want to select the package, we have to retrieve the optional products associated to it and display them.

Optional Product-> Package

@ManyToMany. It's not necessary because we don't require all the packages containing the same optional product.

7 Entities

@Data annotation is provided by Lombok library and automatically generates getters and setters.

Auditing table

```
@Entity
@Table(name = "auditingtable")
@NamedQuery(name="AuditingTable.findAlerts", query="SELECT a FROM AuditingTable a")
public @Data class AuditingTable implements Serializable{
	private static final long serialVersionUID = 1L;

	@Id
	private int id;
	private String username;
	private String email;
	private int amount;
	private Date rejectionDate;
	private Time rejectionTime;
}
```

Client

```
@Entity
@Table(name = "client")
@NamedQueries({
@NamedQuery(name="Client.clientFromCredentials", query="SELECT c FROM Client c WHERE
c.username = ?1 and c.password = ?2"),
@NamedQuery(name="Client.getInsolventClients", query="SELECT c FROM Client c WHERE
c.insolvent=true"),
})
public @Data class Client implements Serializable, User{

	private static final long serialVersionUID = 1L;

	@Id
	@Column(name="Username")
	private String username;

	private String password;
	private String email;
	private boolean insolvent;

}
```

Employee

```
@Entity
@Table(name = "employee")
@NamedQuery(name="Employee.employeeFromCredentials", query="SELECT e FROM Employee e
WHERE e.username = ?1 and e.password = ?2")
public @Data class Employee implements Serializable, User {

	private static final long serialVersionUID = 1L;

	@Id
	@Column(name="Username")
	private String username;

	private String password;
	private String email;

}
```

Materialed view, optional products

```
@Entity
@Table(name = "mv_optprod")
@NamedQuery(name="MvOptProd.findBestSeller", query="SELECT mv FROM MvOptProd mv WHERE
mv.tot_revenue= (SELECT max(mv.tot_revenue) FROM MvOptProd mv)")
public @Data class MvOptProd implements Serializable{

	private static final long serialVersionUID = 1L;

	@Id
	String id_optprod;

	int tot_revenue;

}
```

Materialed view, service packages

```
@Entity
@IdClass(MvPackageId.class)
@Table(name = "mv_package")
@NamedQueries({
	@NamedQuery(name="MvPackage.findPurchasesPerPackageAndValidityPeriod",
query="SELECT mv.packName, mv.months, mv.totPurchase FROM MvPackage mv"),
	@NamedQuery(name="MvPackage.findTotPurchase", query="SELECT mv.packName,
sum(mv.totPurchase) FROM MvPackage mv GROUP BY mv.id_package"),
	@NamedQuery(name="MvPackage.revWoOpt", query="SELECT mv.packName,
sum(mv.totRevenueWoOpt) FROM MvPackage mv GROUP BY mv.id_package"),
	@NamedQuery(name="MvPackage.revWOpt", query="SELECT mv.packName,
sum(mv.totRevenueWOpt) FROM MvPackage mv GROUP BY mv.id_package"),
	@NamedQuery(name="MvPackage.avgOpt", query="SELECT mv.packName,
sum(mv.totPurchase), sum(mv.totOptProd) FROM MvPackage mv GROUP BY mv.id_package")
})
public @Data class MvPackage implements Serializable{
	private static final long serialVersionUID = 1L;

	@Id
	int id_package;
	@Id
	int months;
	String packName;
	int totPurchase;
	int totRevenueWoOpt;
	int totRevenueWOpt;
	int totOptProd;

}

public class MvPackageId implements Serializable {

	private static final long serialVersionUID = 1L;

	private int id_package;
	private int months;

	/*This class also has methods to work as IdClass for MvPackage*/
}
```

Optional product

```
@Entity
@Table(name = "optionalproduct")
@NamedQuery(name="OptionalProduct.findAllOptionalProducts", query="SELECT opt FROM
OptionalProduct opt")
public @Data class OptionalProduct implements Serializable {
	private static final long serialVersionUID = 1L;

	@Id
	private String name;

	private int monthlyFee;

}
```

Order

```
@Entity
@Table(name="orders")
@NamedQueries({
	@NamedQuery(name="Order.findAllByUser", query="SELECT o FROM Order o WHERE
o.subscription.user = ?1"),
	@NamedQuery(name="Order.findAllInvalidByUser", query="SELECT o FROM Order o WHERE
o.subscription.user = ?1 and o.validity = false"),
	@NamedQuery(name="Order.findBySubscription", query="SELECT o FROM Order o WHERE
o.subscription = :sub"),
	@NamedQuery(name="Order.suspendedOrders",query="SELECT o FROM Order o WHERE
o.validity=false"),
	@NamedQuery(name="Order.findById", query="SELECT o FROM Order o WHERE o.id =
:id")
})
public @Data class Order implements Serializable {
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;

	private Date creationDate;
	private Time creationTime;
	private boolean validity;
	private int refusedPayments;

	@OneToOne(fetch=FetchType.EAGER)
	@JoinColumn(name="subscription")
	private Subscription subscription;

	private int amount;

}
```

Service

```
@Entity
@Table(name = "service")
@NamedQuery(name="Service.findAllServices", query="SELECT s FROM Service s")
public @Data class Service implements Serializable {

	private static final long serialVersionUID = 1L;

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

	private String type;
	private Integer includedGB;
	private Double extraGBFee;
	private Integer includedMinutes;
	private Double extraMinutesFee;
	private Integer includedSMS;
	private Double extraSMSFee;

}
```

Service package

```
@Entity
@Table(name = "package")
@NamedQuery(name="ServicePackage.findAll", query="SELECT sp FROM ServicePackage sp")
public @Data class ServicePackage implements Serializable {

	private static final long serialVersionUID = 1L;

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

	private String name;

	@ManyToMany(fetch=FetchType.EAGER)
	@JoinTable(name="servpack",
			joinColumns=@JoinColumn(name="Package"),
			inverseJoinColumns=@JoinColumn(name="Service"))
	private List<Service> services;

	@ManyToMany(fetch=FetchType.EAGER)
	@JoinTable(name="optionalpack",
			joinColumns=@JoinColumn(name="Package"),
		inverseJoinColumns=@JoinColumn(name="Product"))
	private List<OptionalProduct> optionalProducts;
}
```

Subscription

```
@Entity
public @Data class Subscription implements Serializable {
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;

	@ManyToOne(fetch = FetchType.EAGER)
	@JoinColumn(name = "Package")
	private ServicePackage package_;

	@Temporal(TemporalType.DATE)
	private Date startDate;

	private String user;

	// bi-directional many-to-one association to ValidityPeriod
	@ManyToOne(fetch = FetchType.EAGER)
	@JoinColumn(name = "ValidityPeriod")
	private ValidityPeriod validityPeriod;

	@ManyToMany(fetch = FetchType.EAGER)
	@JoinTable(name = "optionalsub", joinColumns = @JoinColumn(name =
"Subscription"), inverseJoinColumns = @JoinColumn(name = "Product"))
	private List<OptionalProduct> optionalProductsSub;
}
```

Validity period

```
@Entity
@NamedQuery(name="ValidityPeriod.findAll", query="SELECT v FROM ValidityPeriod v")
public @Data class ValidityPeriod implements Serializable {
	private static final long serialVersionUID = 1L;

	@Id
	private int months;

	private int monthlyFee;

}
```

8 List of components

Entities:

- AuditingTable
- Client
- Employee
- MvOptProd
- MvPackage
- MvPackageId
- OptionalProduct
- Order
- Service
- ServicePackage
- Subscription
- User
- ValidityPeriod

Services:

- AuditingService
  - findAlerts()
- MvOptProdService
  - findBestSeller()
- MvPackageService
  - findAllPurchasesPerPackageAndValidityPeriod()
  - findTotPurchase()
  - findTotRevWoOpt()
  - findTotRevWOpt()
  - findAvgOpt()
- OptService
  - findAllOptProducts()
  - addOptionalProduct(name, monthlyFee)
  - findProductsSelected(optProdList)
- OrderService
  - getInvalidOrdersByClient(client)
  - getSuspendedOrders()
  - getOrderBySubscription(sub)
  - persistOrder(order)
  - refreshOrder(order)
  - mergeOrder(order)
  - findOrderById(id)
- ServService
  - findAllServices()
  - findServicesSelected(serviceList)
  - persistService(service)
- SpService
  - getAllPackages()
  - getServicePackageById(id)
  - addServicePackage(name, services, optionalProducts)
- SubService
  - persistSubscription (sub)
  - refreshSubscription (sub)
- UserService
  - getClient(username, password)
  - getEmployee(username, password)
  - addClient(username, password, email)
  - findInsolvents()
- VPService
  - getAllValidityPeriod()
  - getValidityPeriod(months)

Servlet:

- CheckLogin
- ConfirmSub
- CreateOptionalProduct
- CreateService
- CreateServicePackage
- CreateSubscription
- GetActivationSchedule
- GetBuyPage
- GetClientHomePage
- GetConfirmationPage
- GetEmployeeHomePage
- GetLogin
- GetSalesPage
- JSONPackage
- JSONTable

- Logout
- PayOrder
- RefetchOrder
- Register

Utils:

- Error
- JSONConverter
- TemplateEngineHandler

Filters:

- ClientFilter
- ClientOrNullFilter
- EmployeeFilter

Views:

- buy.html
- clientHome.html
- confirmation.html
- employeeHome.html
- index.html
- sales.html

Scripts:

- optprod.js
- sales.js
- service.js
- utils.js

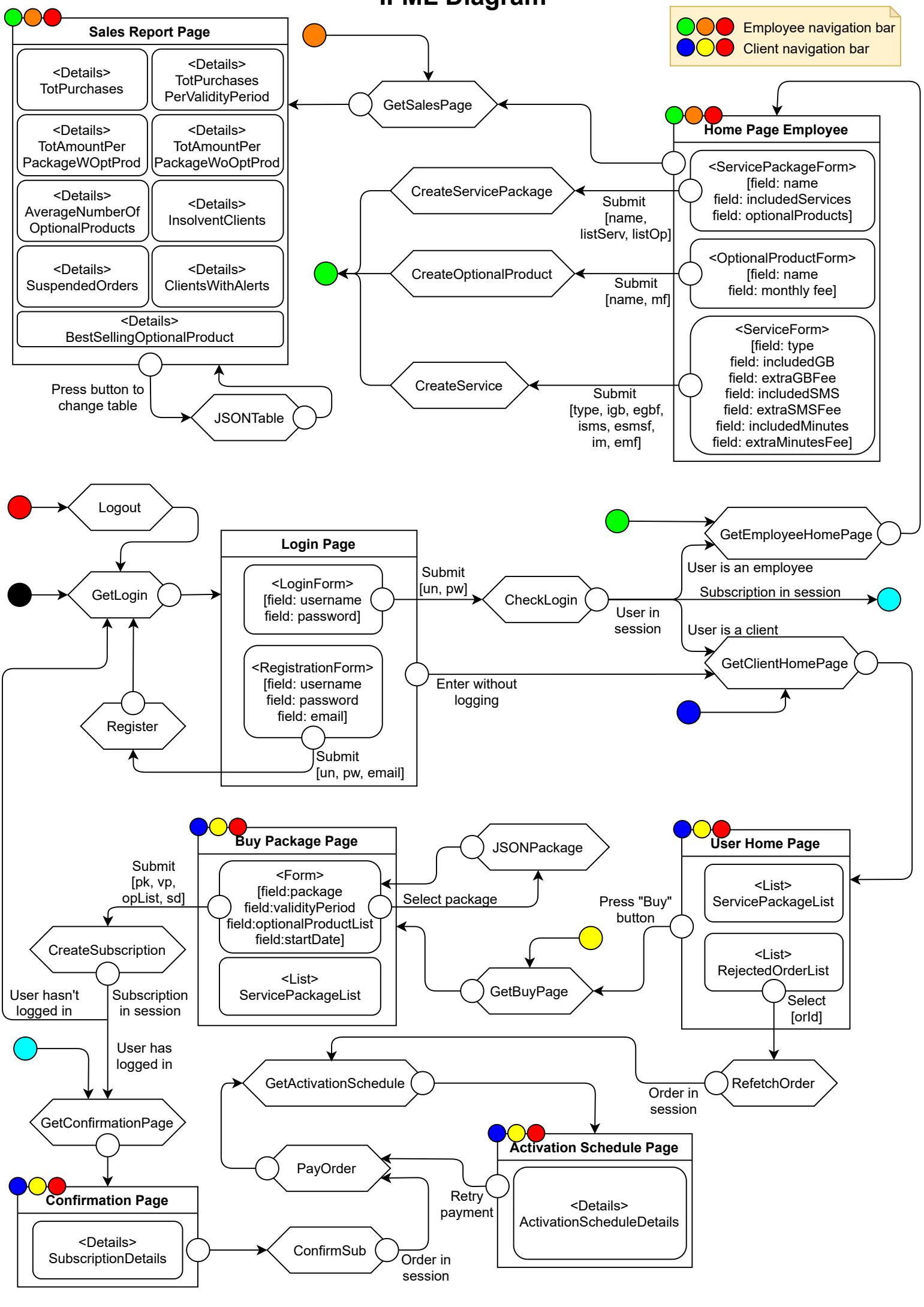
9 IFML

IFML diagram is shown in the next page.

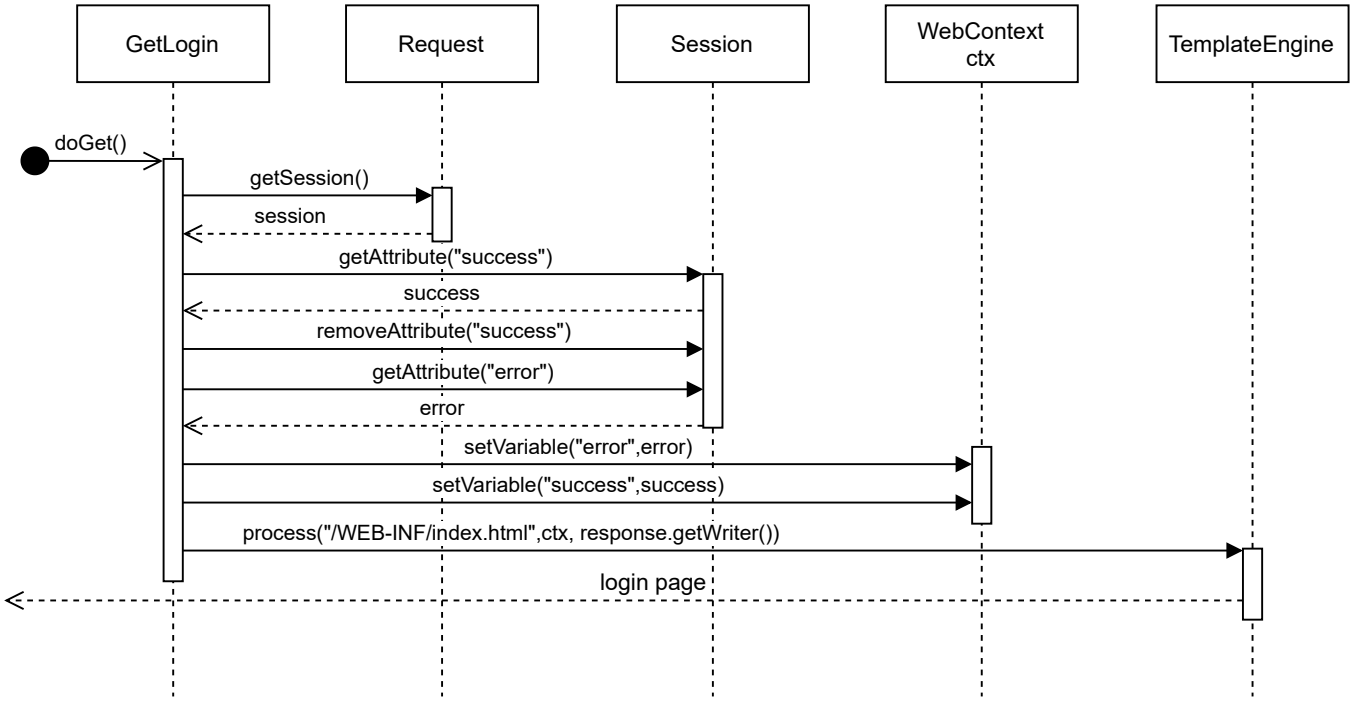
10 Sequence diagrams

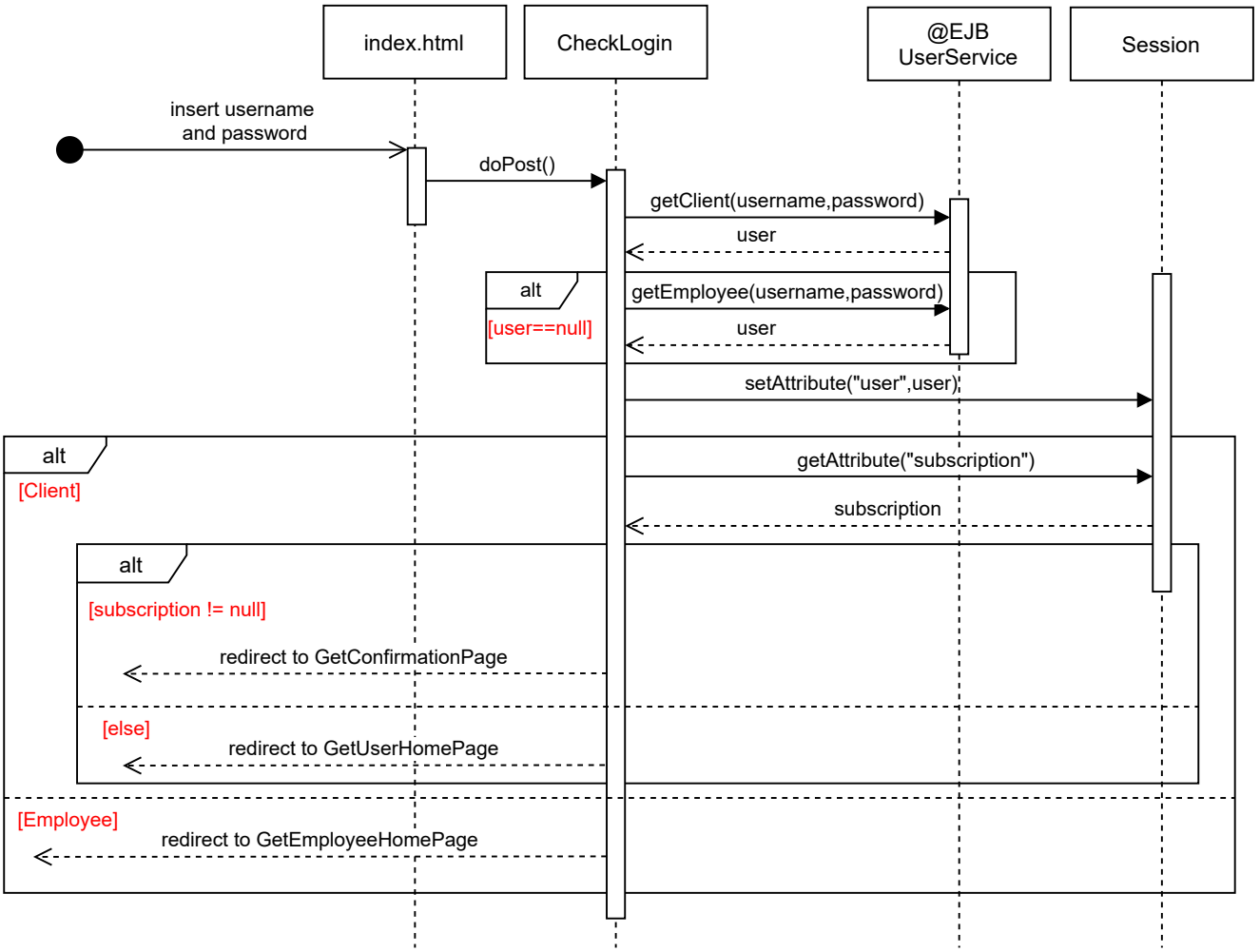
After the IFML diagram, sequence diagrams are inserted.

IFML Diagram

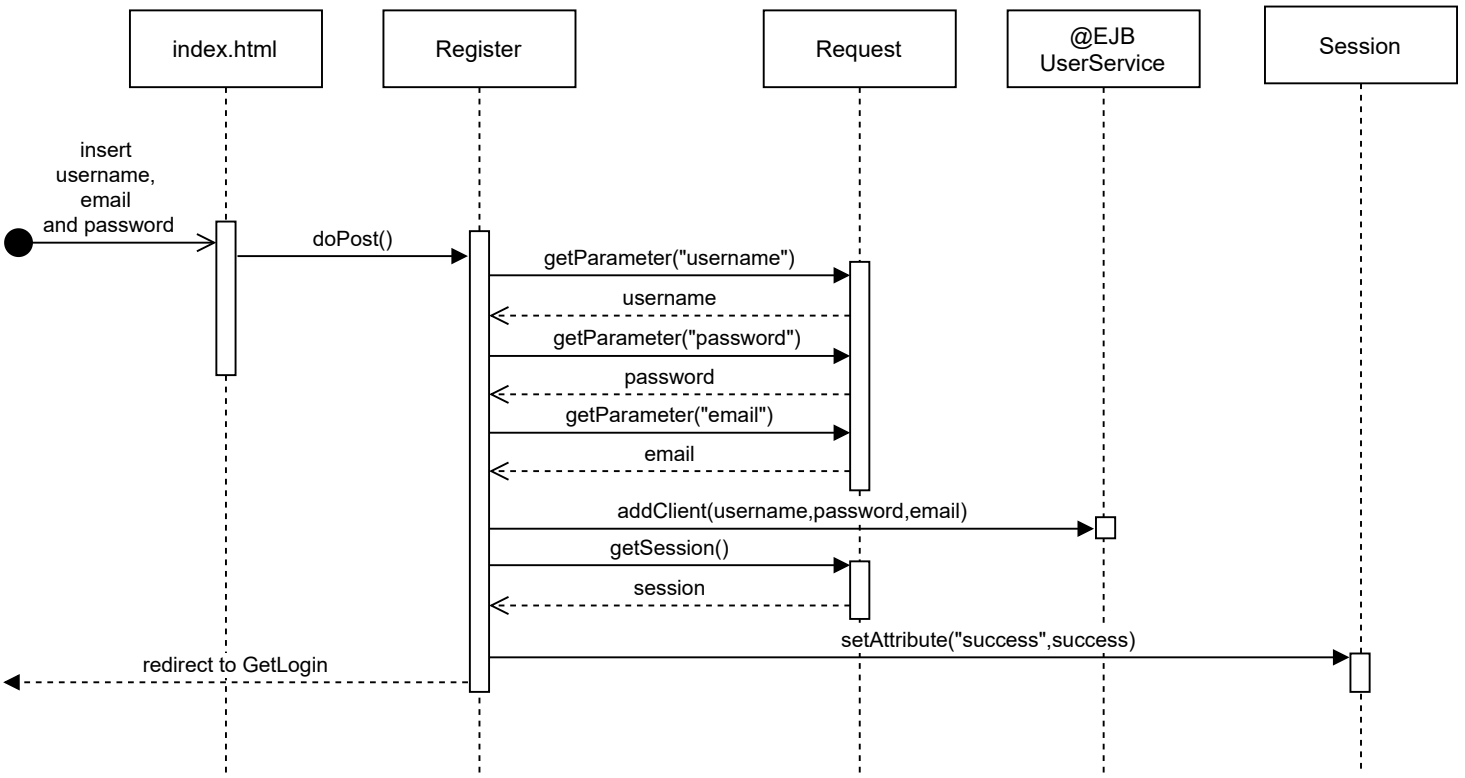


Login

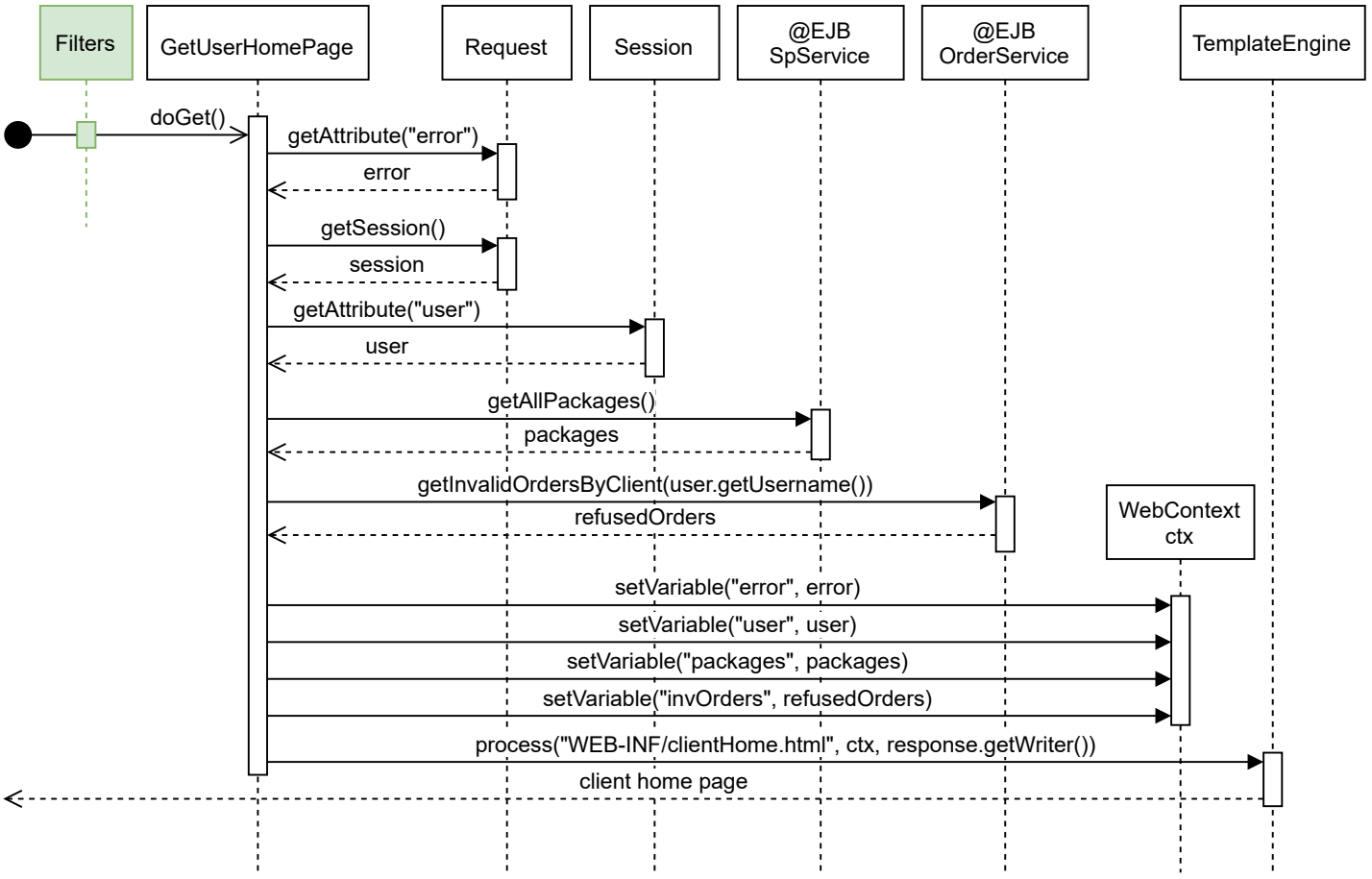

GetLogin
Request
Session
WebContext
ctx
TemplateEngine
doGet()
getSession()
session
getAttribute("success")
success
removeAttribute("success")
getAttribute("error")
error
setVariable("error",error)
setVariable("success",success)
process("/WEB-INF/index.html",ctx, response.getWriter())
login page


index.html
CheckLogin
@EJB
UserService
Session
insert username
and password
doPost()
getClient(username,password)
user
alt
[user==null]
getEmployee(username,password)
user
setAttribute("user",user);
alt
[Client]
getAttribute("subscription")
subscription
alt
[subscription != null]
redirect to GetConfirmationPage
[else]
redirect to GetUserHomePage
[Employee]
redirect to GetEmployeeHomePage

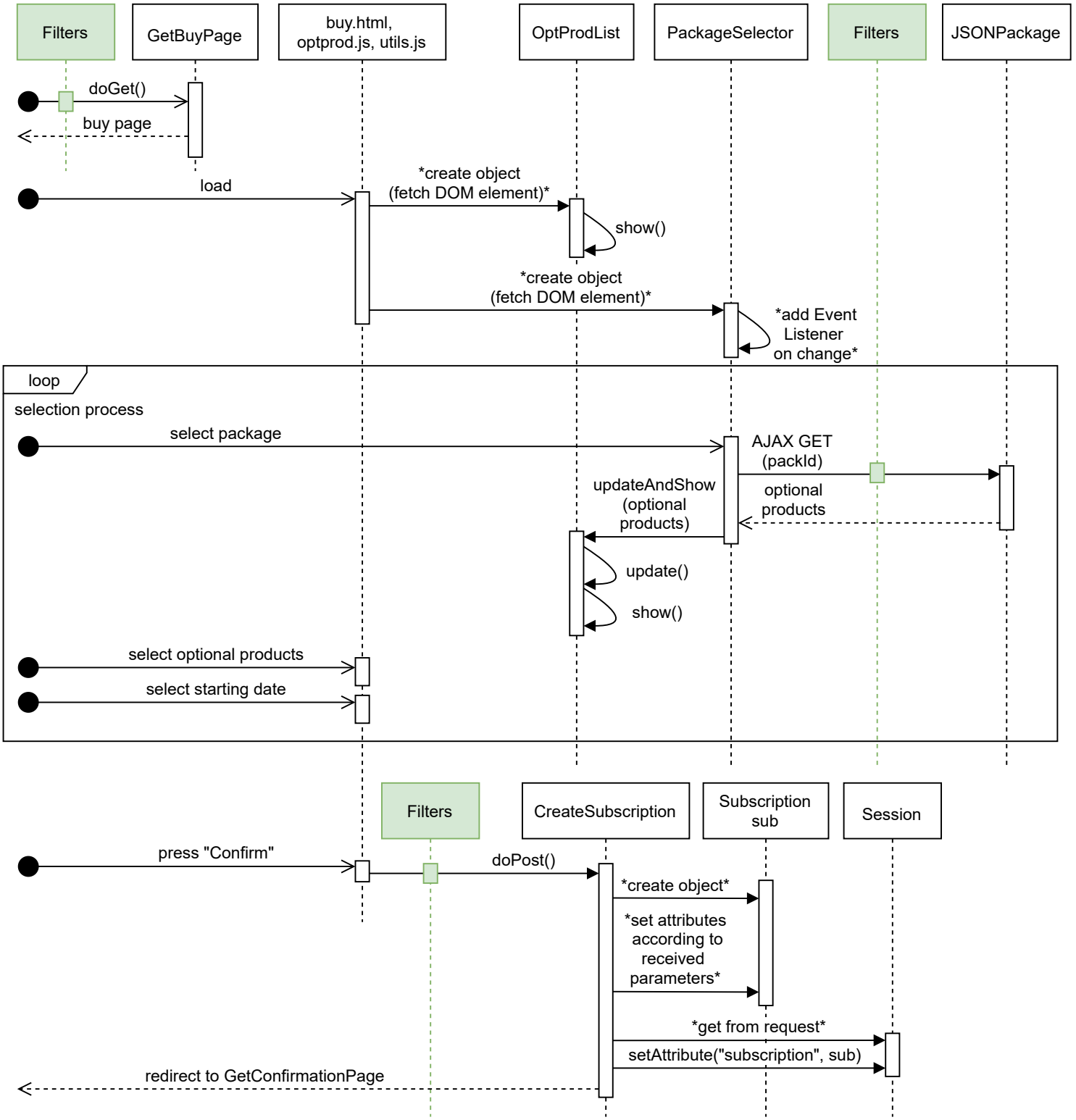
Register



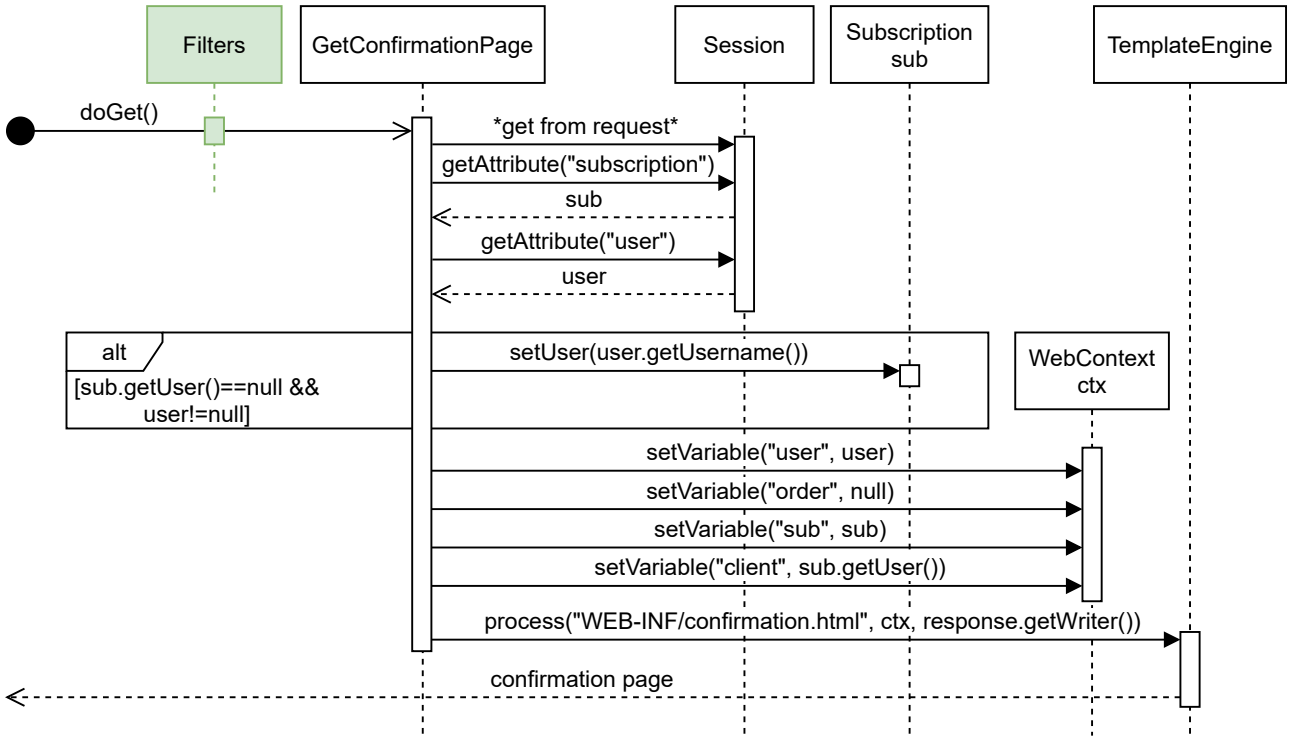
Client home page loading



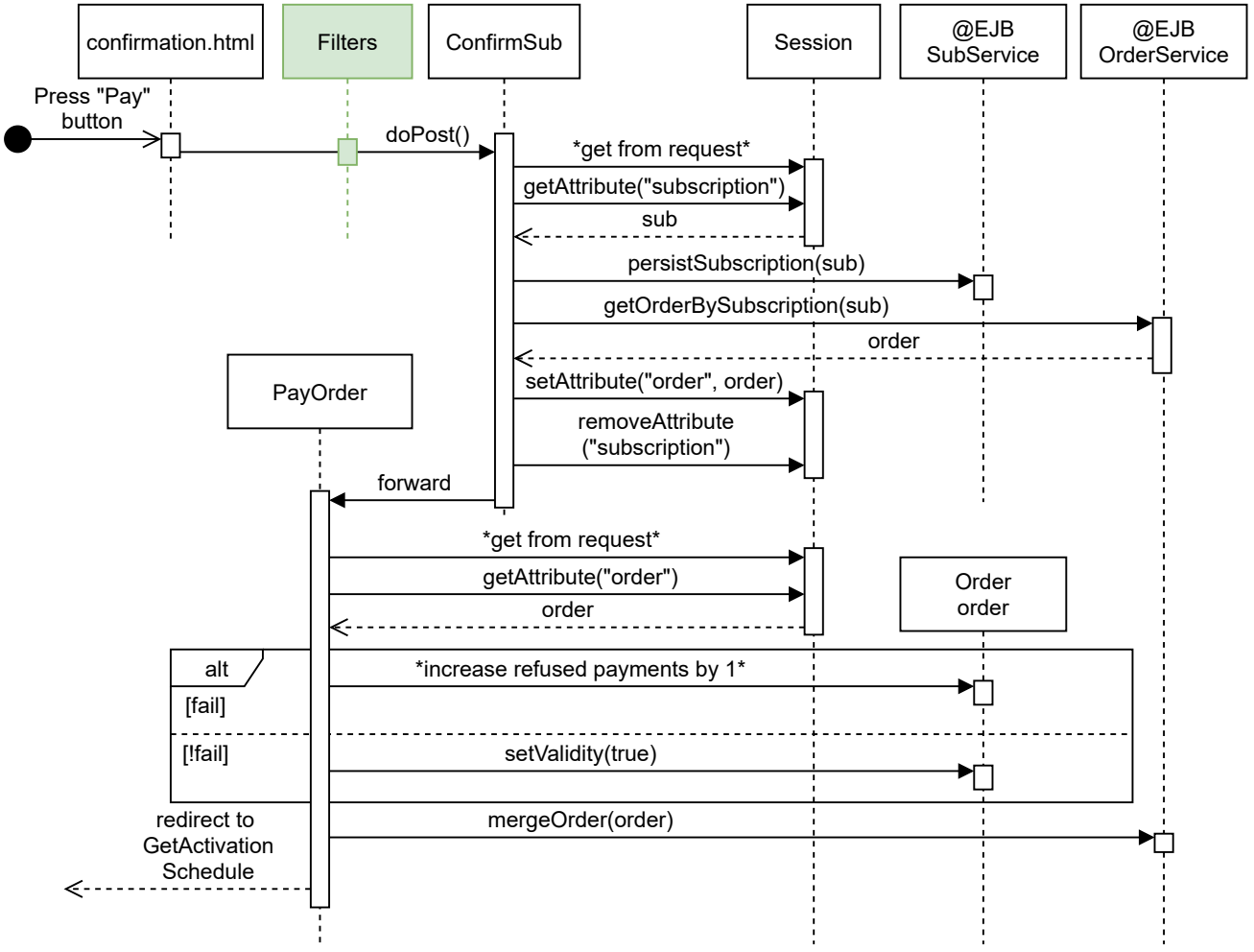
Package selection and subscription creation



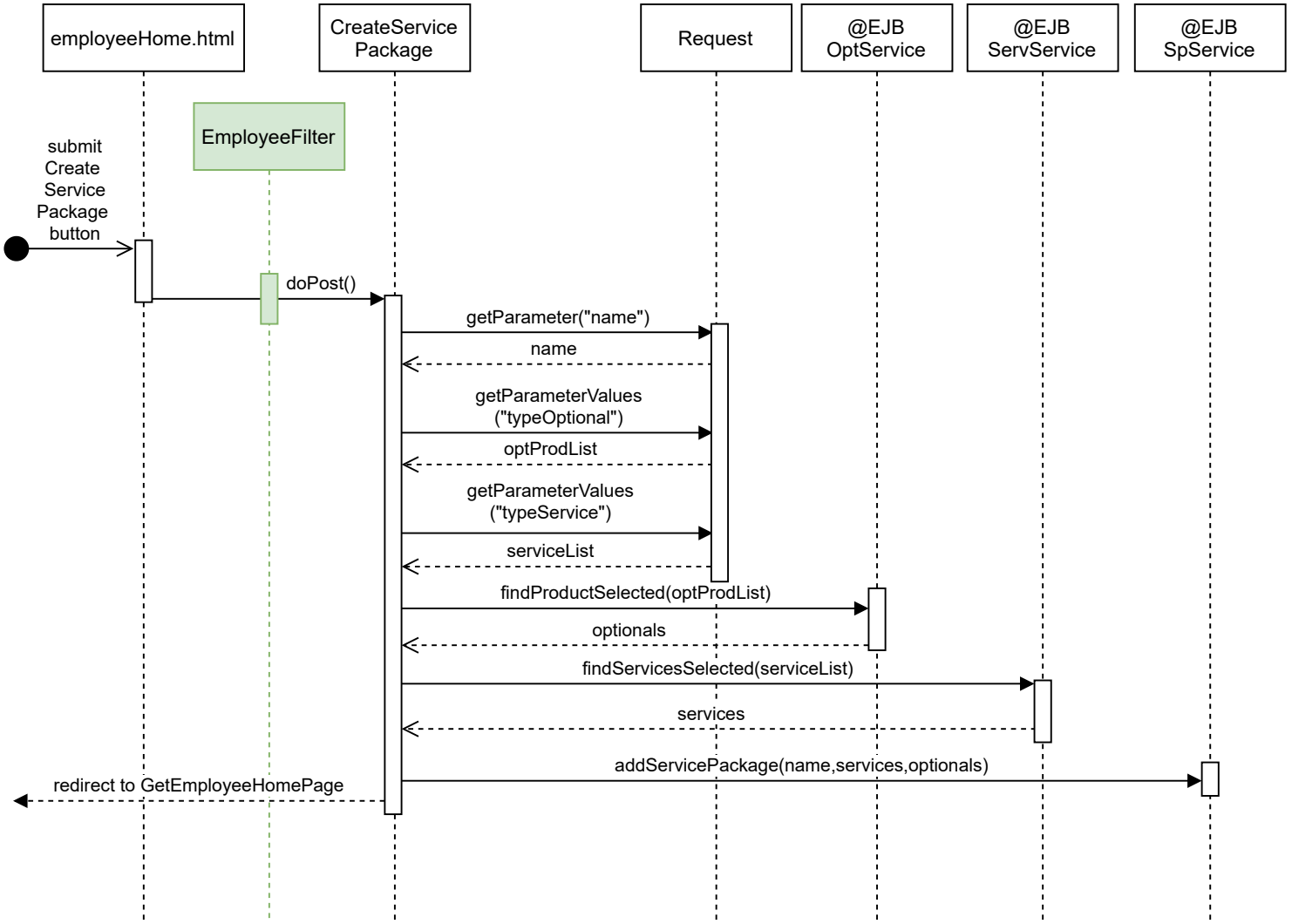
Getting subscription page



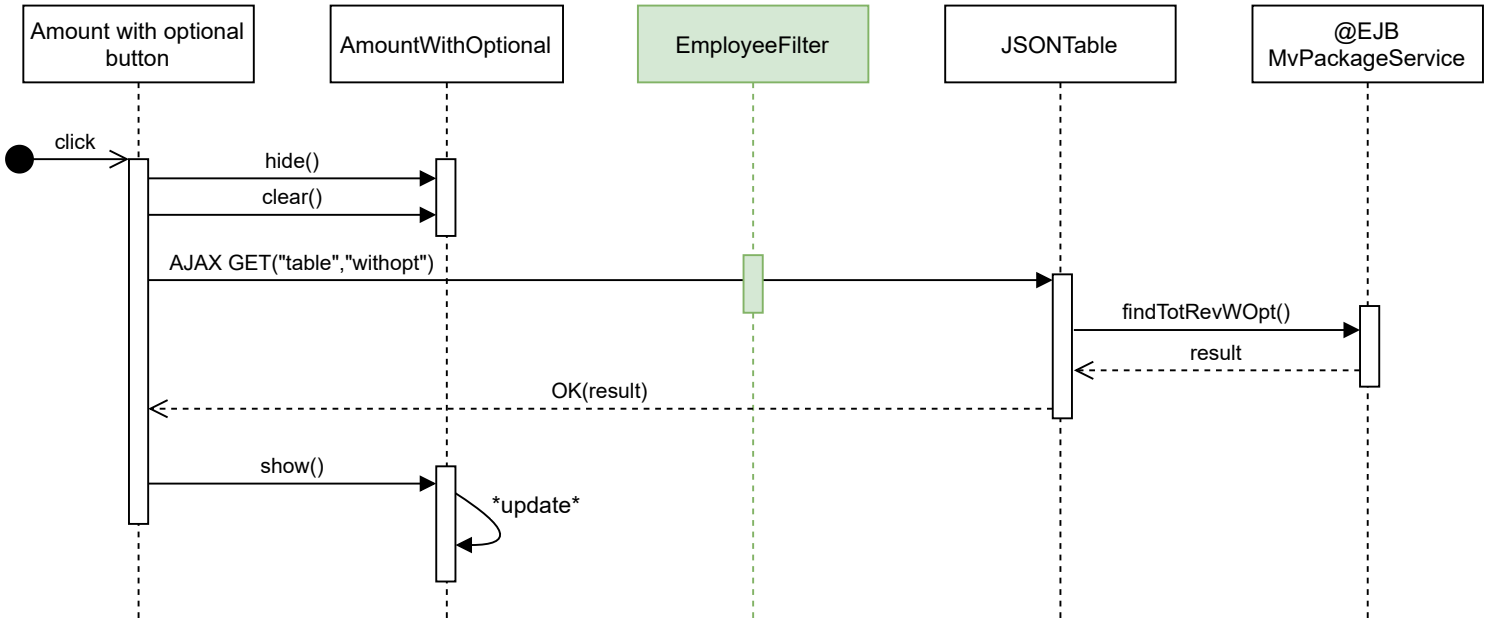
Subscription confirmation and order payment



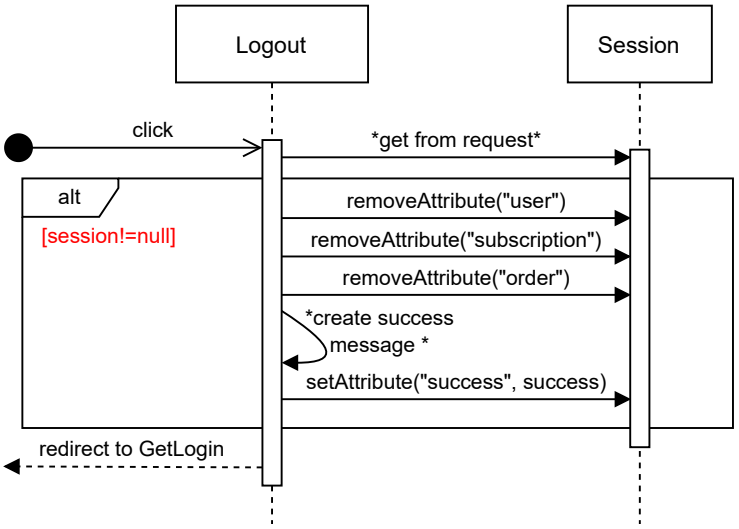
Create service package



See sales details (example: amount with optional product)



Logout


Logout
Session
click
get from request
alt
[session!=null]
removeAttribute("user")
removeAttribute("subscription")
removeAttribute("order")
*create success
message *
setAttribute("success", success)
redirect to GetLogin