# Data base 2

## TELCO SERVICE APPLICATION

Group 34:

| | |
|---|---|
| Andrea Bosia | Person code: 10591419 |
| | Mat: 978262 |
| Erika Buoninfante | Person code: 10636425 |
| | Mat: 975781 |

Professor:

Comai Sara

2021/2022

# Index

# Specifications

**CONSUMER APPLICATION**

The consumer application has a public Landing page with a form for login and a form for registration. Registration requires a username, 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.

# Specification interpretation

## AUDITING TABLE

Once a user fails 3 payments, an alert is created in the auditing table: it keeps track of the cumulative amount of the rejected orders. When the number of failed payments increases/decreases, the alert is updated by computing the new amount corresponding to the rejected orders. If the number of failed payments goes under 3, the alert is removed.

## EXTERNAL SERVICE

An additional field is used (makePaymentFail): when it is set to TRUE, the user’s payment will fail, otherwise it will end successfully (useful for DEMO).

## COMPLETE PAYMENT PAGE

An additional page showing the output of the payment is used.

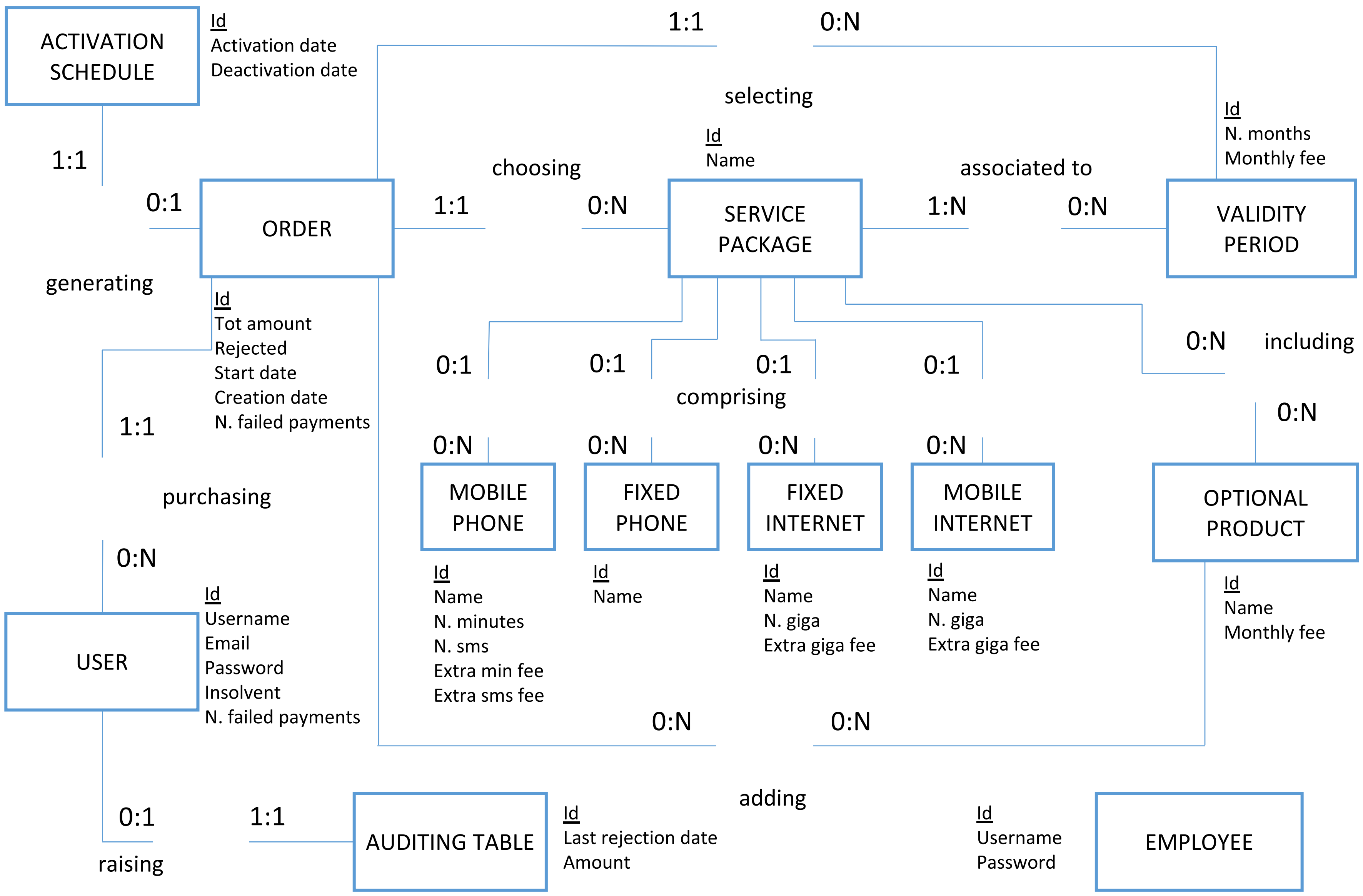
## SERVICE PACKAGE

A service package contains at least one service (fixed/mobile phone/internet).

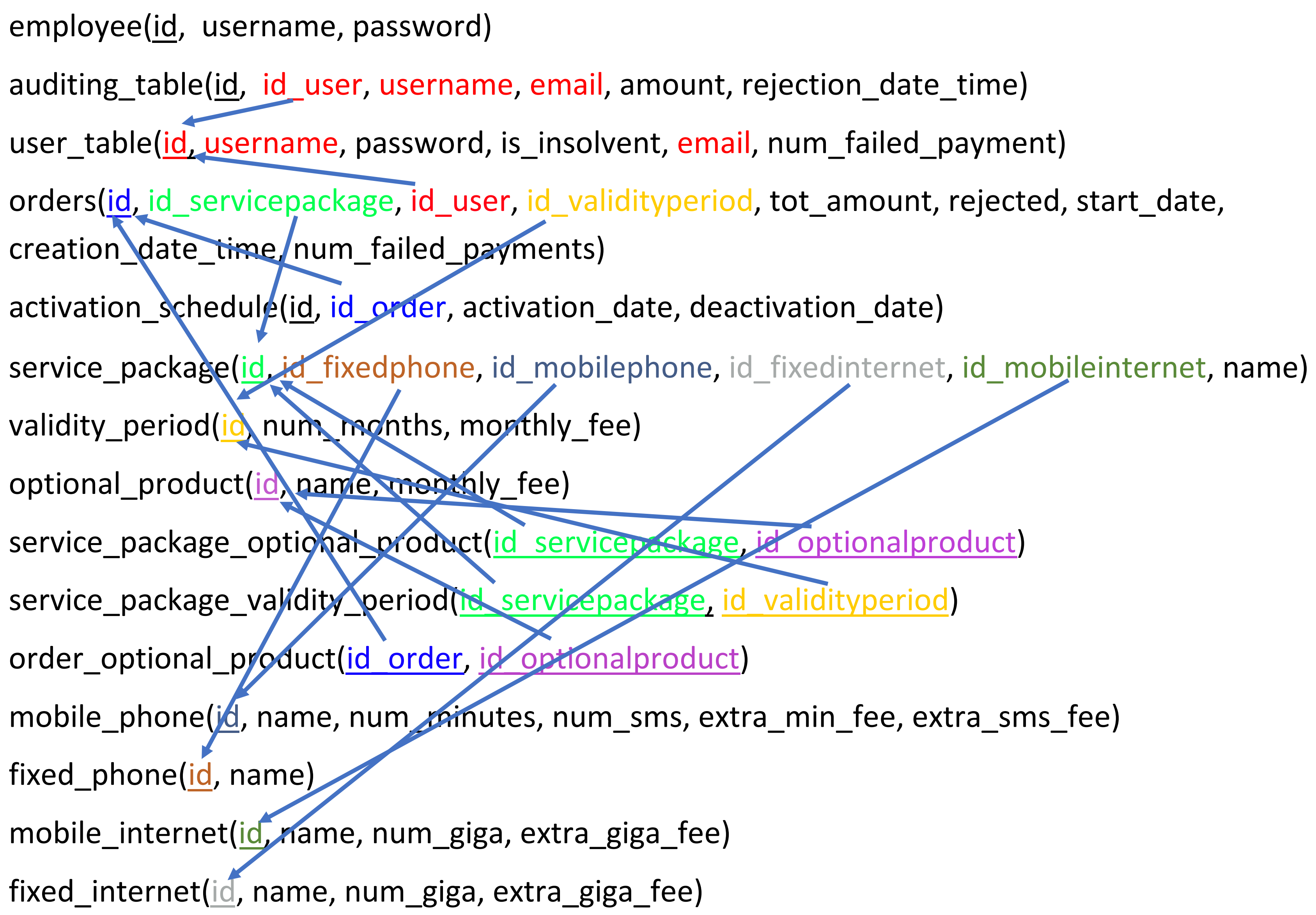
## VALIDITY PERIOD

Each service package can be associated with a subset of available validity periods. Then the consumer must choose one of them in order to complete the subscription.

# Entity Relationship diagram



# Relational model



employee(id, username, password)

auditing_table(id, id_user, username, email, amount, rejection_date_time)

user_table(id, username, password, is_insolvent, email, num_failed_payment)

orders(id, id_servicepackage, id_user, id_validityperiod, tot_amount, rejected, start_date, creation_date_time, num_failed_payments)

activation_schedule(id, id_order, activation_date, deactivation_date)

service_package(id, id_fixedphone, id_mobilephone, id_fixedinternet, id_mobileinternet, name)

validity_period(id, num_months, monthly_fee)

optional_product(id, name, monthly_fee)

service_package_optional_product(id_servicepackage, id_optionalproduct)

service_package_validity_period(id_servicepackage, id_validityperiod)

order_optional_product(id_order, id_optionalproduct)

mobile_phone(id, name, num_minutes, num_sms, extra_min_fee, extra_sms_fee)

fixed_phone(id, name)

mobile_internet(id, name, num_giga, extra_giga_fee)

fixed_internet(id, name, num_giga, extra_giga_fee)

## SQL DDL

```
create table employee (
        id int not null auto_increment,
        username varchar(45) not null unique,
        password varchar(45) not null,
        primary key (id)
);

create table user_table (
        id int not null auto_increment,
        username varchar(45) not null unique,
        password varchar(45) not null,
        is_insolvent boolean not null default false,
        email varchar(45) not null unique,
        num_failed_payments int not null default 0,
        unique (id,username,email),
        primary key (id)
);

create table auditing_table (
        id int not null auto_increment,
        id_user int not null,
        username varchar(45) not null,
        email varchar(45) not null,
        amount decimal(5,2) not null,
        rejection_date_time datetime not null,
        foreign key (id_user,username,email) references user_table(id,username,email)
                                         on update cascade on delete cascade,
        primary key (id)
);
```

## SQL DDL

```
create table fixed_phone (
        id int not null auto_increment,
        name varchar(45) not null unique,
        primary key (id)
);

create table mobile_phone (
        id int not null auto_increment,
        name varchar(45) not null unique,
        num_minutes int not null,
        num_sms int not null,
        extra_min_fee decimal(5,2) not null,
        extra_sms_fee decimal(5,2) not null,
        primary key (id)
);

create table fixed_internet (
        id int not null auto_increment,
        name varchar(45) not null unique,
        num_giga int not null,
        extra_giga_fee decimal(5,2) not null,
        primary key (id)
);

create table mobile_internet (
        id int not null auto_increment,
        name varchar(45) not null unique,
        num_giga int not null,
        extra_giga_fee decimal(5,2) not null,
        primary key (id)
);
```

# SQL DDL

```
create table service_package(
        id int not null auto_increment,
        id_fixedphone int,
        id_mobilephone int,
        id_fixedinternet int,
        id_mobileinternet int,
        name varchar(45) not null unique,
        foreign key (id_fixedphone) references fixed_phone(id)
                                on update cascade on delete restrict,
        foreign key (id_mobilephone) references mobile_phone(id)
                                on update cascade on delete restrict,
        foreign key (id_fixedinternet) references fixed_internet(id)
                                on update cascade on delete restrict,
        foreign key (id_mobileinternet) references mobile_internet(id)
                                on update cascade on delete restrict,
        primary key (id)
);

create table validity_period (
        id int not null auto_increment,
        num_months int not null,
        monthly_fee decimal(5,2) not null,
        unique (num_months, monthly_fee),
        check(num_months > 0),
        check(monthly_fee >= 0),
        primary key (id)
);
```

# SQL DDL

```
create table service_package_validity_period (
        id_servicepackage int not null,
        id_validityperiod int not null,
        foreign key (id_servicepackage) references service_package(id)
                                on update cascade on delete cascade,
        foreign key (id_validityperiod) references validity_period(id)
                                on update cascade on delete cascade,
        primary key (id_servicepackage, id_validityperiod)
);

create table orders (
        id int not null auto_increment,
        id_servicepackage int not null,
        id_user int not null,
        id_validityperiod int not null,
        tot_amount decimal(5,2) not null,
        rejected boolean default null,
        start_date date not null,
        creation_date_time timestamp not null,
        num_failed_payments int not null default 0,
        foreign key (id_servicepackage) references service_package(id)
                                on update cascade on delete restrict,
        foreign key (id_user) references user_table(id)
                                on update cascade on delete restrict,
        foreign key (id_validityperiod) references validity_period(id)
                                on update cascade on delete restrict,
        primary key (id)
);
```

# SQL DDL

```sql
create table activation_schedule (
        id int not null auto_increment,
        activation_date date not null,
        deactivation_date date not null,
        id_order int not null,
        foreign key (id_order) references orders(id)
                          on update cascade on delete cascade,
        primary key (id)
);

create table optional_product (
        id int not null auto_increment,
        name varchar(45) not null unique,
        monthly_fee decimal(5,2) not null,
        check(monthly_fee >= 0),
        primary key (id)
);

create table service_package_optional_product (
        id_servicepackage int not null,
        id_optionalproduct int not null,
        foreign key (id_servicepackage) references service_package(id)
                                on update cascade on delete cascade,
        foreign key (id_optionalproduct) references optional_product(id)
                                 on update cascade on delete cascade,
        primary key (id_servicepackage, id_optionalproduct)
);

create table order_optional_product (
        id_order int not null,
        id_optionalproduct int not null,
        foreign key (id_order) references orders (id)
                          on update cascade on delete cascade,
        foreign key (id_optionalproduct) references optional_product(id)
                                 on update cascade on delete cascade,
        primary key (id_order, id_optionalproduct)
);
```

## Trigger: design and code

# Materialized views

Since materialized views are not supported in Mysql, the aggregate data required in the Sales Report page (Employee application) are computed by triggers that populate ordinary tables. The additional tables are the following:

**Number of total purchases per package**

```
create table tot_purchases_package(
        id int not null auto_increment,
        service_package int not null,
        tot_purchases int not null default 0,
        foreign key (service_package) references service_package(id)
                                on delete cascade on update cascade,
        primary key (id)
);
```

**Number of total purchases per package and validity period**

```
create table tot_purchases_package_vperiod (
        id int not null auto_increment,
        service_package int not null,
        validity_period int not null,
        tot_purchases int not null default 0,
        foreign key (service_package) references service_package(id)
                                on delete cascade on update cascade,
        foreign key (validity_period) references validity_period(id)
                                on delete cascade on update cascade,
        primary key (id)
);
```

# Materialized views

### Total value of sales per package with and without the optional products

```
create table tot_sales_package (
        id int not null auto_increment,
        service_package int not null,
        tot_sales decimal(7,2) not null default 0.00,
        sales_with_op_products decimal (7,2) not null default 0.00,
        foreign key (service_package) references service_package(id)
                                 on delete cascade on update cascade,
        primary key(id)
);
```

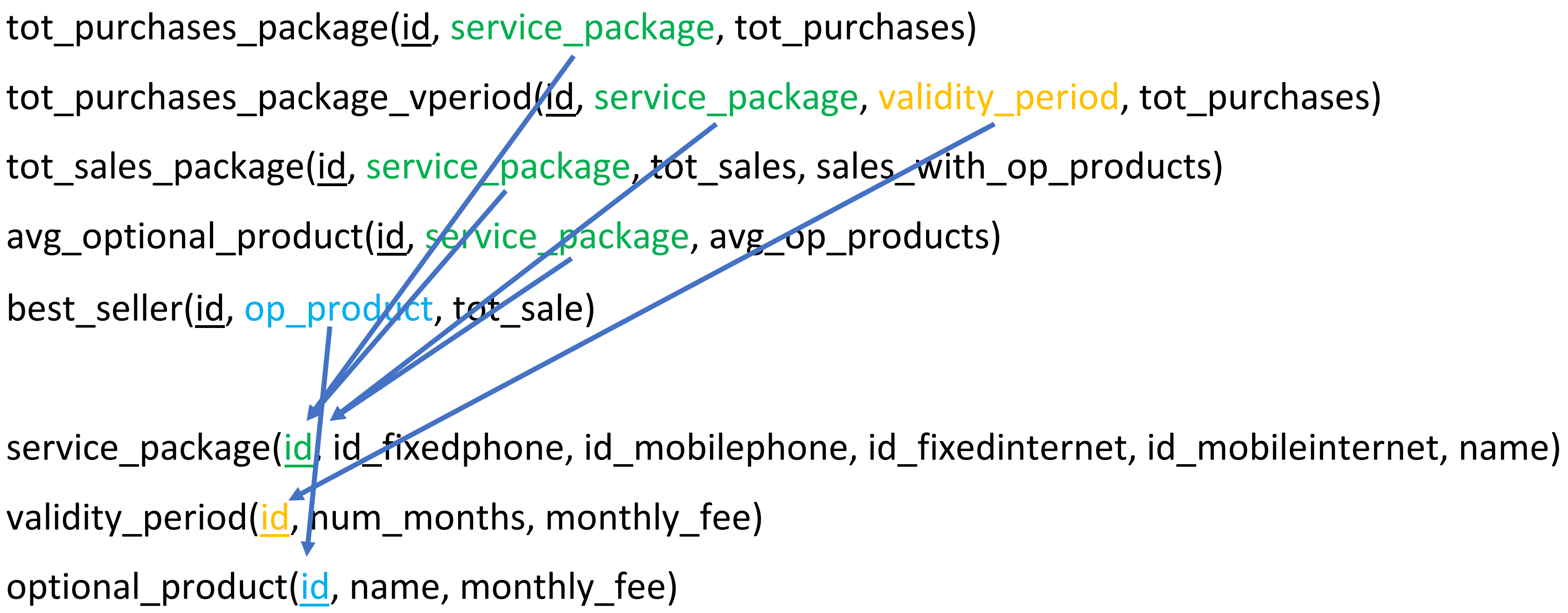
### Average number of optional products sold together with each service package

```
create table avg_optional_product (
        id int not null auto_increment,
        service_package int not null,
        avg_op_products int default 0,
        foreign key(service_package) references service_package(id)
                                 on delete cascade on update cascade,
        primary key(id)
);
```

### Best seller optional product

```
create table best_seller (
        id int default 1,
        op_product int,
        tot_sale decimal(5,2),
        foreign key(op_product) references optional_product(id)
                            on delete cascade on update cascade,
        primary key(id)
);
```

# Relational model (materialized views)



tot_purchases_package(id, service_package, tot_purchases)

tot_purchases_package_vperiod(id, service_package, validity_period, tot_purchases)

tot_sales_package(id, service_package, tot_sales, sales_with_op_products)

avg_optional_product(id, service_package, avg_op_products)

best_seller(id, op_product, tot_sale)

service_package(id, id_fixedphone, id_mobilephone, id_fixedinternet, id_mobileinternet, name)

validity_period(id, num_months, monthly_fee)

optional_product(id, name, monthly_fee)

# Trigger insert_service_package

**Event** —> After insert into service_package

**Condition** —> None

**Action** —> Insert a new tuple in tot_purchases_package with new service package and default value
Insert a new tuple in tot_sales_package with new service package and default values
Insert a new tuple in avg_optional_product with new service package and default value

```
create trigger insert_service_package
after insert on service_package
for each row
begin
 insert into tot_purchases_package(id,service_package,tot_purchases)
     values(default, new.id, default);
 insert into tot_sales_package(id,service_package,tot_sales,sales_with_op_products)
     values(default, new.id, default, default);
 insert into avg_optional_product(id,service_package,avg_op_products)
     values(default, new.id, default);
end$$
```

## Trigger insert_tot_purchases_package_vperiod

**Event** —> After insert into service_package_validity_period

**Condition** —> None

**Action** —> Insert a new tuple in tot_purchases_package_vperiod with new service package, the associated validity period and default value

```
create trigger insert_tot_purchases_package_vperiod
after insert on service_package_validity_period
for each row
insert into tot_purchases_package_vperiod(id, service_package, validity_period,
                                          tot_purchases)
        values(default, new.id_servicepackage, new.id_validityperiod, default);
```

# Trigger update_tot_purchases_package

**Event** —> After insert into activation_schedule (i.e. a new order correctly paid)

**Condition** —> None

**Action** —> Update value tot_purchases of table tot_purchases_package for involved service package

```
create trigger update_tot_purchases_package
after insert on activation_schedule
for each row
update tot_purchases_package
      set tot_purchases = tot_purchases + 1
      where service_package = (select sp.id from service_package as sp, orders as
                              ord where new.id_order = ord.id and
                              ord.id_servicepackage = sp.id);
```

# Trigger update_tot_purchases_package_vperiod

**Event** —> After insert into activation_schedule (i.e. a new order correctly paid)

**Condition** —> None

**Action** —> Update value tot_purchases of table tot_purchases_package_vperiod for involved service package and validity period

```
create trigger update_tot_purchases_package_vperiod
after insert on activation_schedule
for each row
update tot_purchases_package_vperiod
    set tot_purchases = tot_purchases + 1
    where service_package = (select sp.id from service_package as sp, orders as
                             ord where new.id_order = ord.id and
                             ord.id_servicepackage = sp.id)
    and validity_period = (select vp.id from validity_period as vp, orders as
                             ord where new.id_order = ord.id and
                             ord.id_validityperiod = vp.id);
```

# Trigger update_tot_sales_package

**Event** —> After insert into activation_schedule (i.e. a new order correctly paid)

**Condition** —> None

**Action** —> Update values tot_sales and sales_with_op_products of table tot_sales_package for involved service package

```
create trigger update_tot_sales_package
after insert on activation_schedule
for each row
update tot_sales_package
     set sales_with_op_products = sales_with_op_products + (select tot_amount from
                             orders where id = new.id_order),
         tot_sales = tot_sales + (select vp.num_months*vp.monthly_fee from
              validity_period as vp, orders as ord
          where ord.id = new.id_order and ord.id_validityperiod = vp.id)
     where service_package = (select sp.id from service_package as sp, orders as
                             ord where new.id_order = ord.id and
                        ord.id_servicepackage = sp.id);
```

# Trigger update_avg_optional_product

**Event** —> After insert into activation_schedule (i.e. a new order correctly paid)

**Condition** —> None

**Action** —> Update value avg_op_products of table avg_op_product for involved service package

A view has been created in order to support the computation of the average value:
it shows the service package and the number of optional products sold for each completed order.

```
create view op_products_sold_per_order as (
       select act.id_order as id_order, sp.id as id_servicepackage,
              count(oop.id_order) as op_products_sold
       from activation_schedule as act left join order_optional_product as oop
              on act.id_order = oop.id_order, service_package as sp, orders as ord
       where ord.id = act.id_order and sp.id = ord.id_servicepackage
       group by act.id_order
);
```

# Trigger update_avg_optional_product

The trigger that computes the average value uses the view previously showed.

```
create trigger update_avg_optional_product
after insert on activation_schedule
for each row
update avg_optional_product
      set avg_op_products = (select avg(temp.op_products_sold)
                             from op_products_sold_per_order as temp, orders as ord
                             where new.id_order = ord.id and
                                   temp.id_servicepackage = ord.id_servicepackage)
      where service_package = (select sp.id
                               from service_package as sp, orders as ord
                               where new.id_order = ord.id and
                                ord.id_servicepackage = sp.id);
```

# Trigger compute_best_seller

**Event** —> After insert into activation_schedule (i.e. a new order correctly paid)

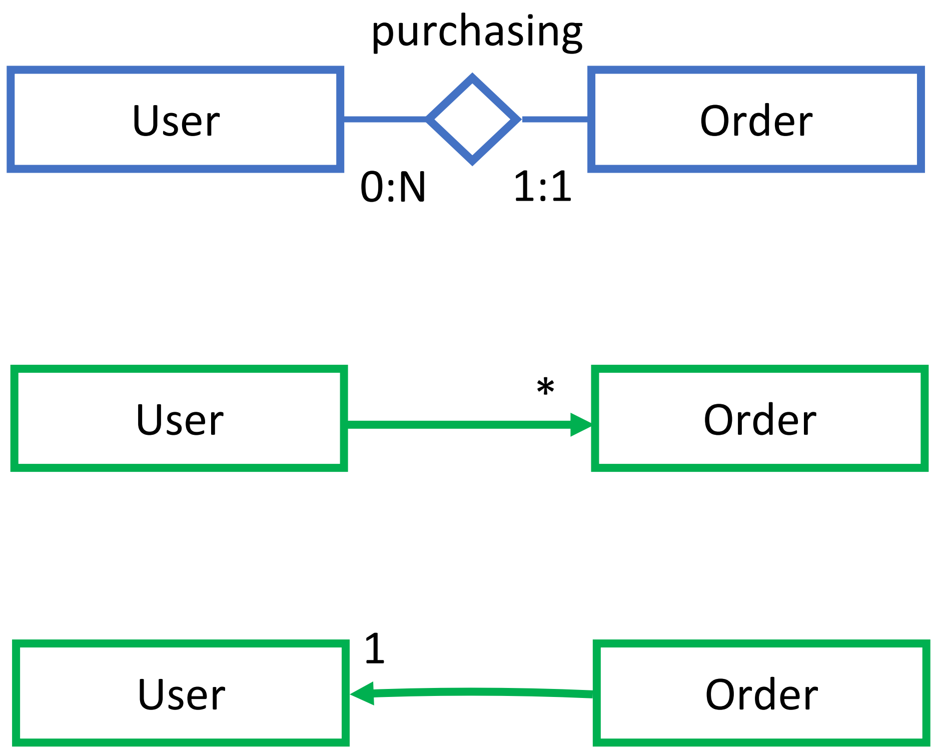
**Condition** —> If a new order comprises optional products

**Action** —> Delete the old best seller, compute and insert a new best seller into table best_seller

```
create trigger compute_best_seller
after insert on activation_schedule
for each row
begin
if(new.id_order in (select id_order from order_optional_product)) then
    delete from best_seller where id = 1;
    insert into best_seller(op_product,tot_sale)
        (select op.id as op_product, sum(op.monthly_fee*vp.num_months) as tot_sale
         from optional_product as op, activation_schedule as act,
          order_optional_product as oop, orders as ord, validity_period as vp
         where  ord.id = act.id_order and oop.id_order = ord.id and
                op.id = oop.id_optionalproduct and vp.id = ord.id_validityperiod
         group by op.id
         order by tot_sale desc limit 1);
end if;
end$$
```

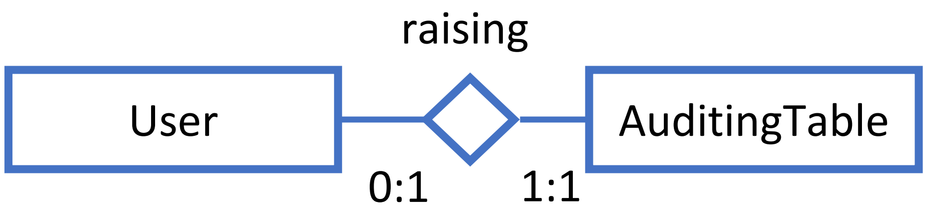
## ORM design

# Relationship “purchasing”

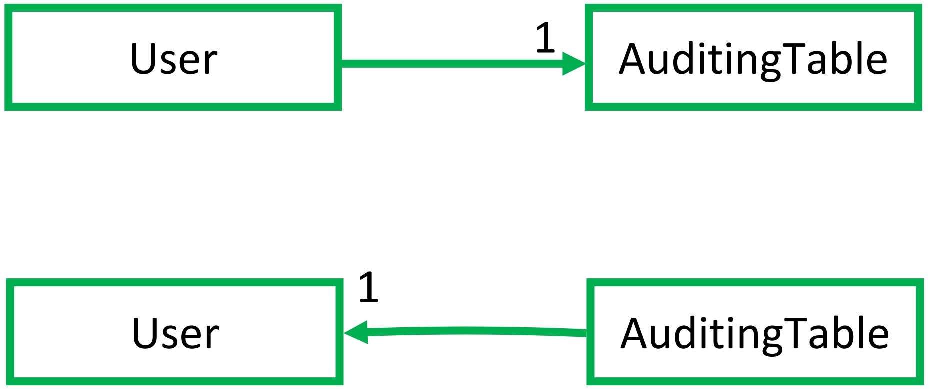


- User → Order
  @OneToMany
  Not necessary, but mapped for simplicity
- Order → User @ManyToOne
  (owner, FK: id_user)
  Used to display the user who has created the order

# Relationship "raising"

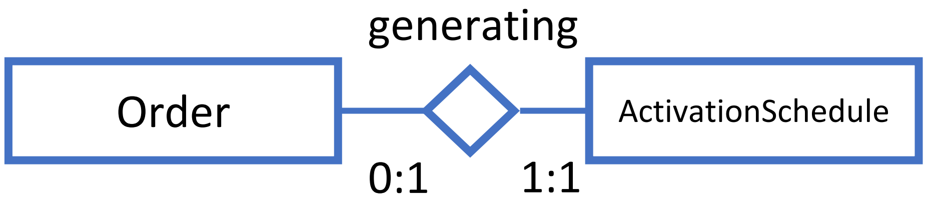


- User → AuditingTable
  @OneToOne Not necessary, but mapped for simplicity

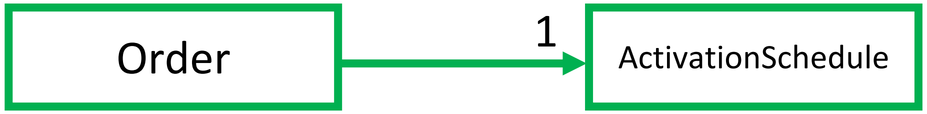


- AuditingTable →User
  @OneToOne (owner,
  FK: id_user, username, email)
  Used to display the user who raised an alert

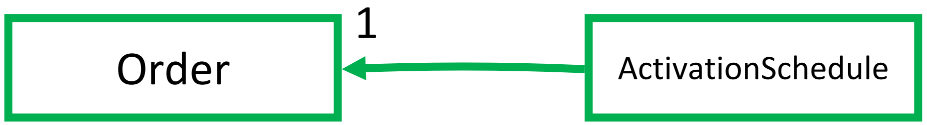
# Relationship “generating”



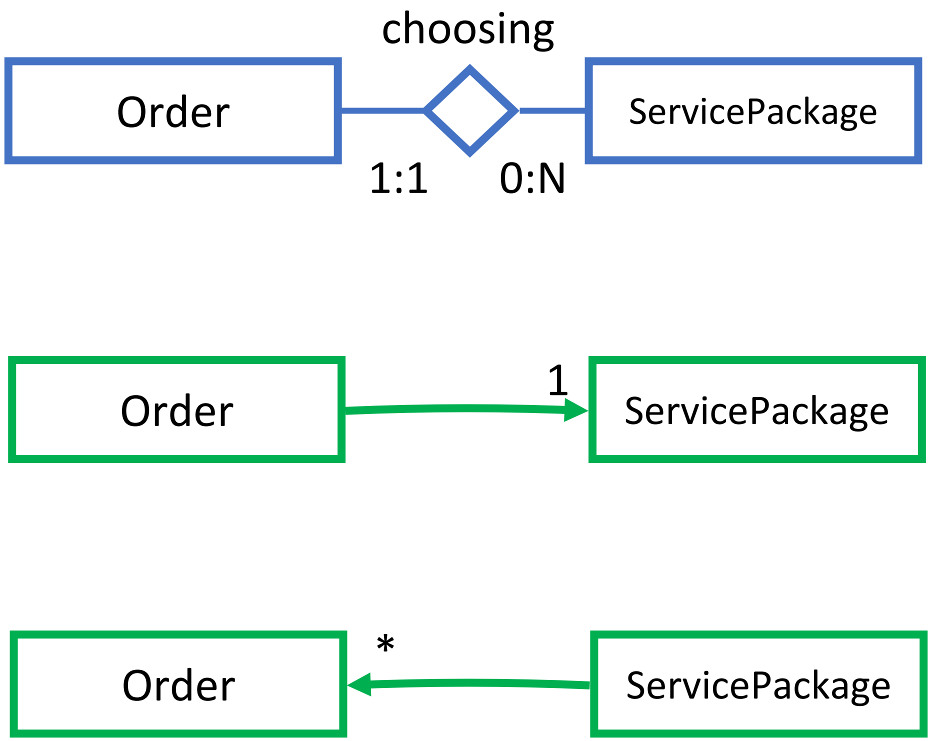
- Order → ActivationSchedule @OneToOne Not necessary, but mapped for simplicity



- ActivationSchedule → Order @OneToOne (owner, FK: id_order)



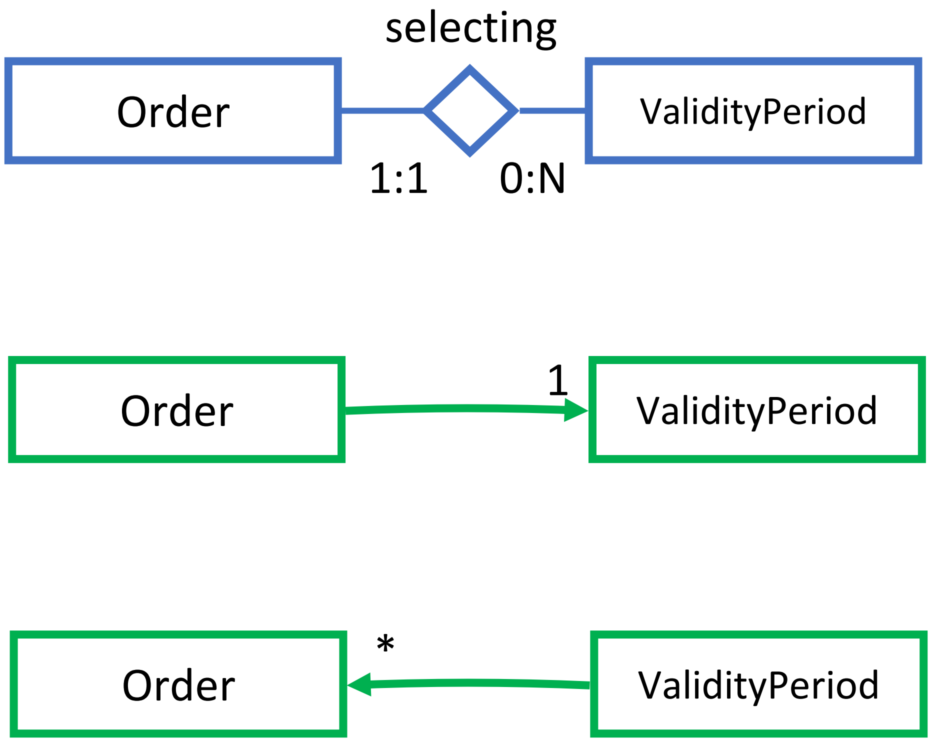
# Relationship “choosing”



- Order → ServicePackage
  @ManyToOne (owner, FK: id_servicepackage) Used to retrieve the service package associated with an order

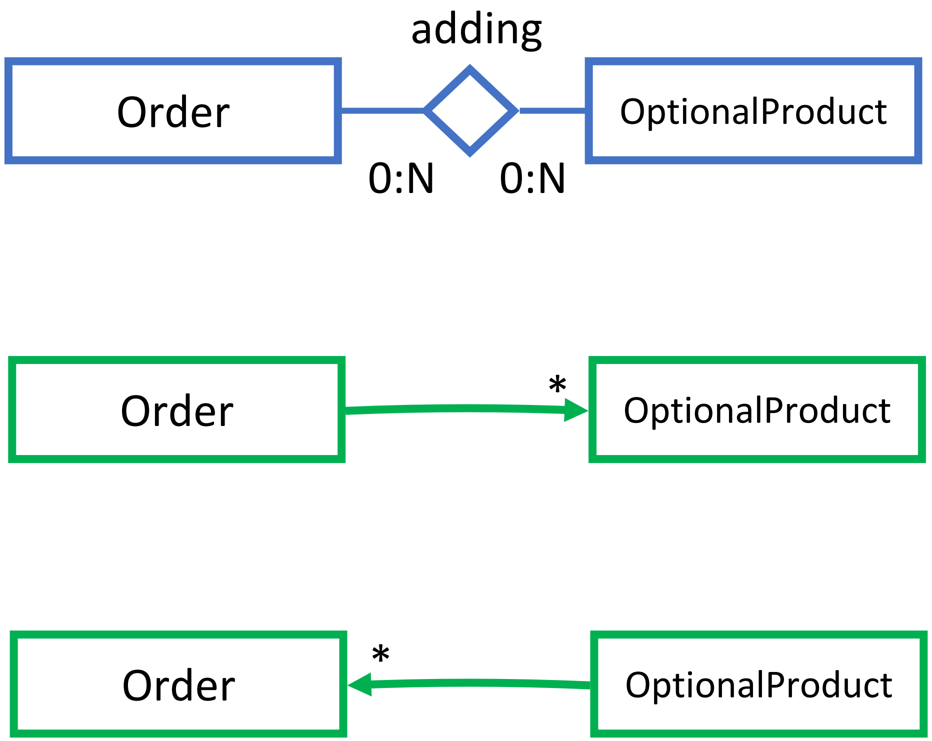
- ServicePackage → Order
  @OneToMany
  Not necessary, but mapped for simplicity

# Relationship "selecting"



- Order → ValidityPeriod @ManyToOne (owner, FK: id_validityperiod) Used to retrieve the validity period associated with an order
- ValidityPeriod → Order @OneToMany Not necessary, but mapped for simplicity

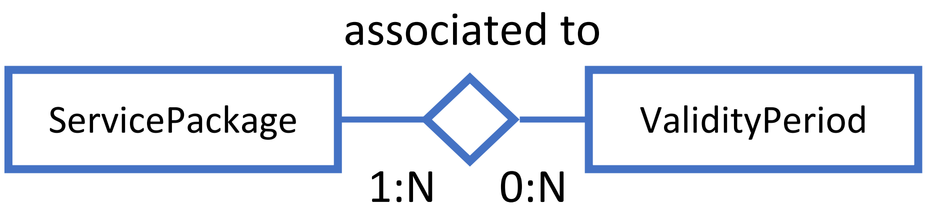
# Relationship “adding”

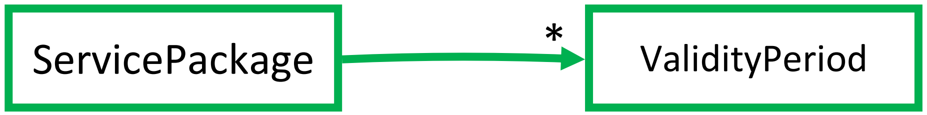


- Order → OptionalProduct
  @ManyToMany (owner, joinTable: order_optional_product) Used to retrieve the optional products associated with an order

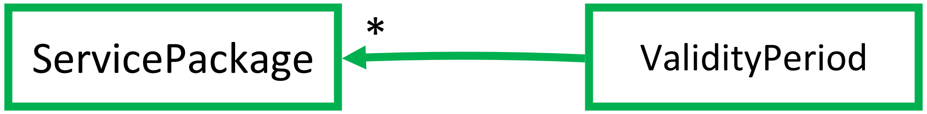
- OptionalProduct → Order
  @ManyToMany
  Not necessary, but mapped for simplicity

# Relationship “associated to”



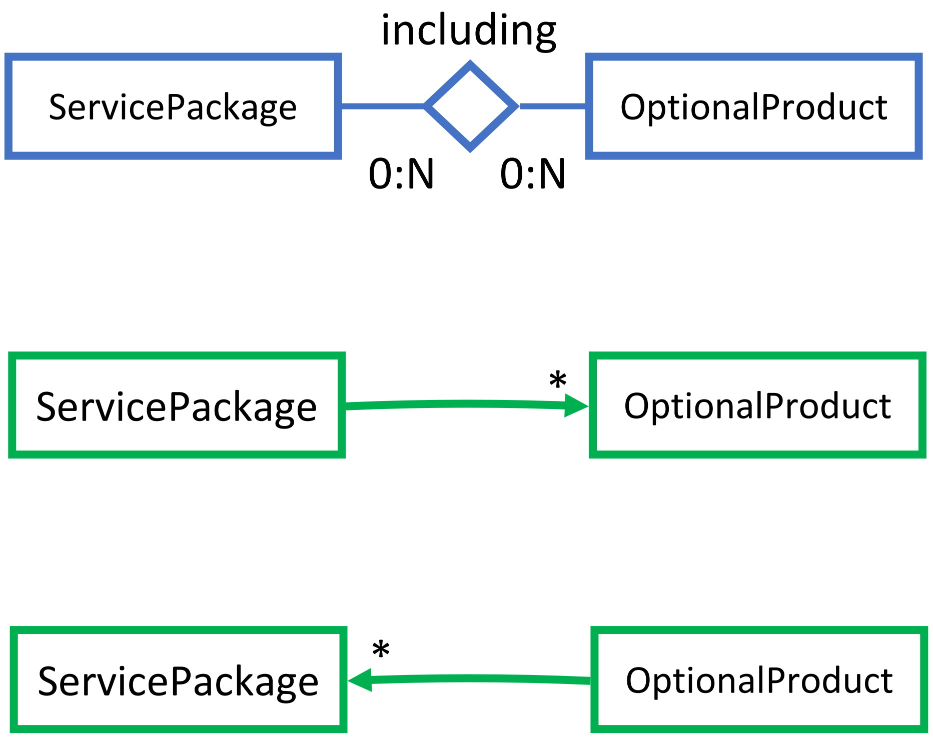


- ServicePackage → ValidityPeriod
  @ManyToMany (owner, joinTable: service_package_validity_period)
  *fetch type*: EAGER (necessary to navigate the set of validity periods associated to a service package in the Home page)



- ValidityPeriod → ServicePackage
  @ManyToMany
  Not necessary, but mapped for simplicity

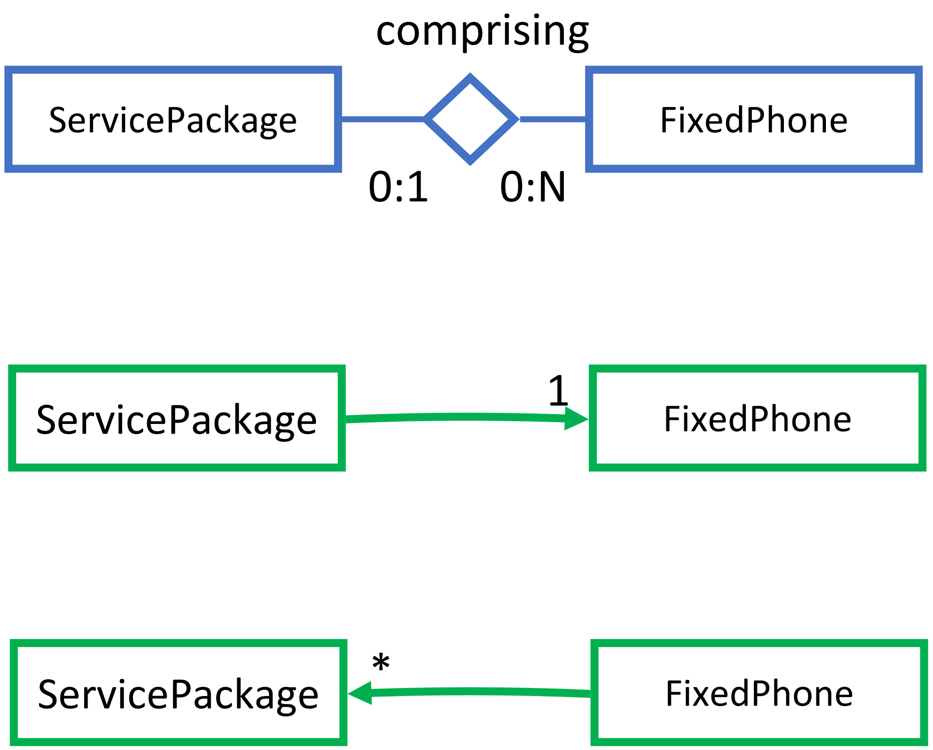
# Relationship "including"



- ServicePackage → OptionalProduct
  @ManyToMany (owner, joinTable: service_package_validity_period)
  *fetch type*: EAGER (necessary to navigate the set of optional products associated to a service package in the Home page)

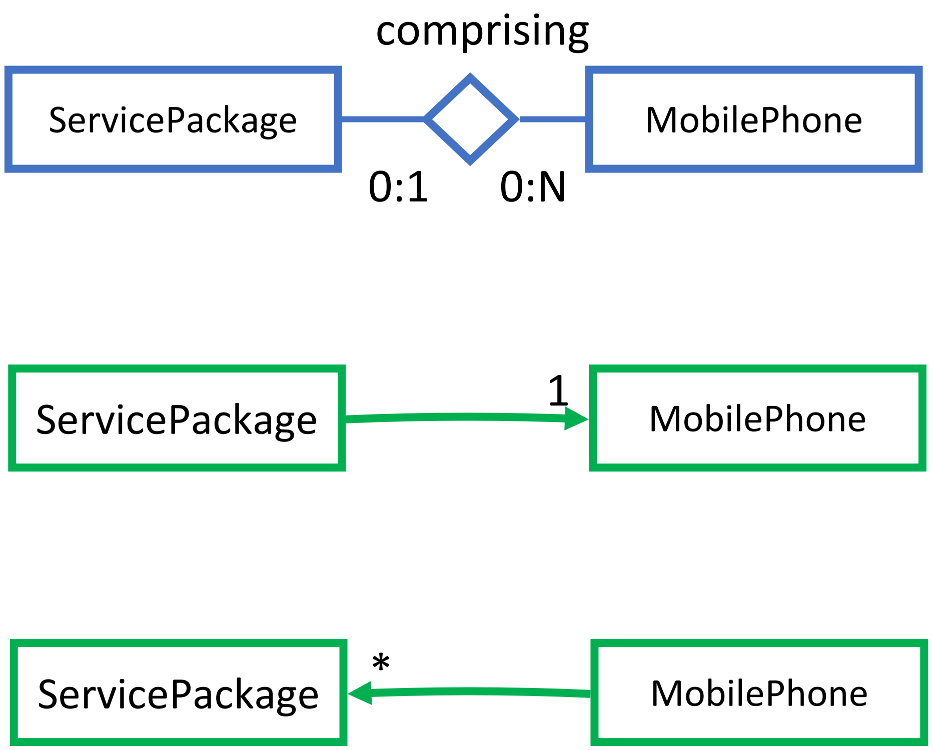
- OptionalProduct → ServicePackage
  @ManyToMany
  Not necessary, but mapped for simplicity

# Relationship “comprising” (FixedPhone)



- ServicePackage → FixedPhone
  @ManyToOne (owner, FK: id_fixedphone) Used to show the fixed phone service associated to a service package
- FixedPhone → ServicePackage
  @OneToMany
  Not necessary, but mapped for simplicity

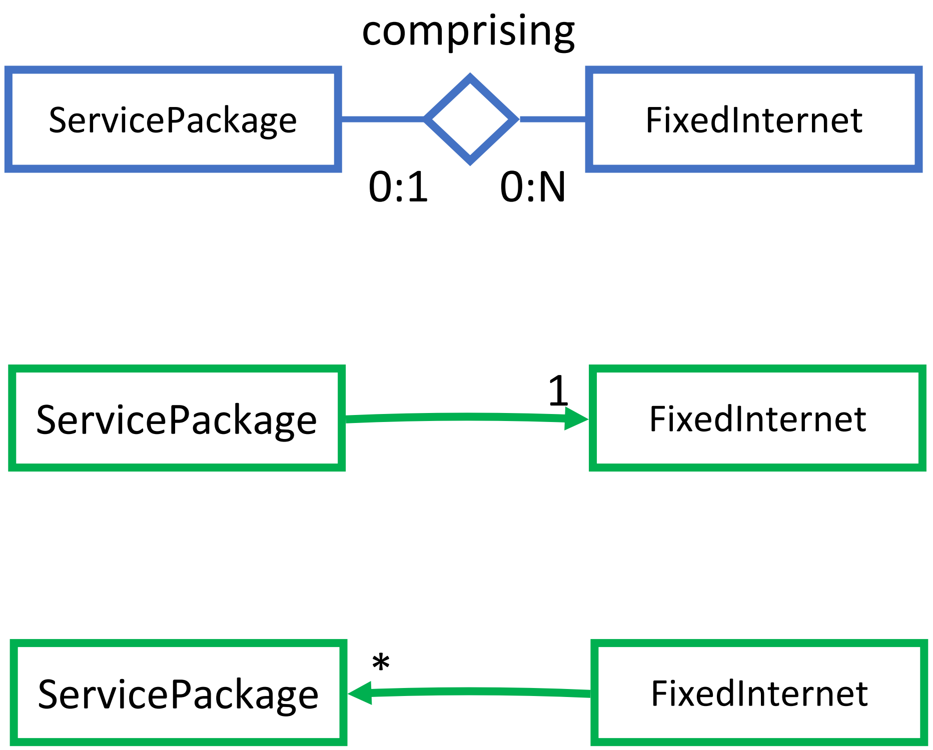
# Relationship “comprising” (MobilePhone)



- ServicePackage → MobilePhone
  @ManyToOne (owner, FK: id_mobilephone) Used to show the mobile phone service associated to a service package

- MobilePhone → ServicePackage
  @OneToMany
  Not necessary, but mapped for simplicity

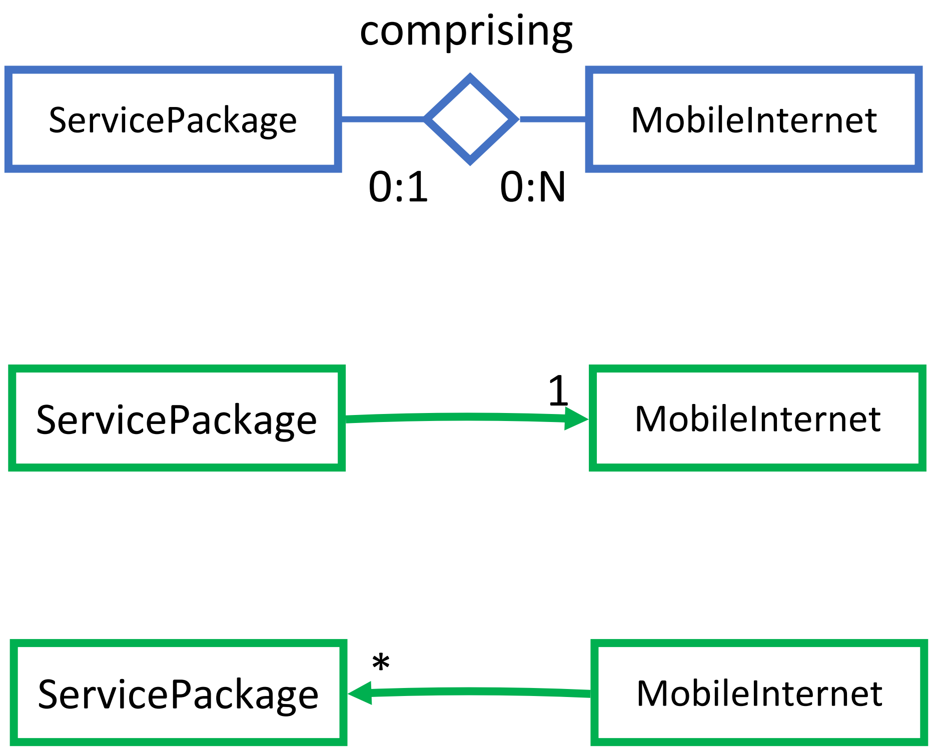
# Relationship “comprising” (FixedInternet)



- ServicePackage → FixedInternet
  @ManyToOne (owner, FK: id_fixedinternet) Used to show the fixed internet service associated to a service package

- FixedInternet → ServicePackage
  @OneToMany
  Not necessary, but mapped for simplicity

# Relationship “comprising” (MobileInternet)



- ServicePackage → MobileInternet @ManyToOne (owner, FK: id_mobileinternet) Used to show the mobile internet service associated to a service package
- MobileInternet → ServicePackage @OneToMany Not necessary, but mapped for simplicity

# Entity Employee

```java
@Entity
@Table(name = "employee", schema = "db_telco")
@NamedQuery(name = "Employee.checkCredentials", query = "SELECT e
FROM Employee e                    WHERE e.username = :username
and e.password = :password")
public class Employee implements Serializable {
    private static final long serialVersionUID = 1L;

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

    private String username;

    private String password;

}
```

# Entity User

```
@Entity
@Table(name = "user_table", schema = "db_telco")
@NamedQueries({
        @NamedQuery(name = "User.checkCredentials", query = "SELECT u FROM User u  WHERE u.username =
:username and u.password = :password"),
        @NamedQuery(name = "User.findByEmailOrUsername", query = "SELECT u FROM User u WHERE u.email = :email
or u.username = :username"),
        @NamedQuery(name = "User.findInsolventUsers", query = "SELECT u FROM User u WHERE u.insolvent =
true")})

public class User implements Serializable {
    private static final long serialVersionUID = 1L;

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

    private String username;

    private String password;

    private String email;

    @Column(name = "num_failed_payments")
    private int numFailedPayments;

    @Column(name = "is_insolvent")
    private boolean insolvent;

    @OneToOne(mappedBy = "user")
    private AuditingTable auditingTableAlert;

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

# Entity ActivationSchedule

```java
@Entity
@Table(name = "activation_schedule", schema = "db_telco")
public class ActivationSchedule implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Temporal(TemporalType.DATE)
    @Column(name = "activation_date")
    private Date activationDate;

    @Temporal(TemporalType.DATE)
    @Column(name = "deactivation_date")
    private Date deactivationDate;

    @OneToOne
    @JoinColumn(name = "id_order")
    private Order order;
```

# Entity AuditingTable

```java
@Entity
@Table(name = "auditing_table", schema = "db_telco")
@NamedQuery(name="AuditingTable.findAll", query="SELECT at FROM AuditingTable
at")
public class AuditingTable implements Serializable {
    private static final long serialVersionUID = 1L;

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

    private float amount;

    @Column(name = "rejection_date_time")
    @Temporal(TemporalType.TIMESTAMP)
    private Date rejectionDateTime;

    @OneToOne
    @JoinColumns({
        @JoinColumn(name="id_user", referencedColumnName="id"),
        @JoinColumn(name="username", referencedColumnName="username"),
        @JoinColumn(name="email", referencedColumnName="email") })
    private User user;
```

## Entity FixedInternet

```
@Entity
@Table(name = "fixed_internet", schema = "db_telco")
@NamedQuery(name="FixedInternet.findAll", query="SELECT fi FROM
FixedInternet fi")
public class FixedInternet implements Serializable{
    private static final long serialVersionUID = 1L;

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

    private String name;

    @Column(name = "num_giga")
    private int numGiga;

    @Column(name = "extra_giga_fee")
    private float extraGigaFee;

    @OneToMany(mappedBy = "fixedInternet")
    private Collection<ServicePackage> servicePackages;
```

## Entity FixedPhone

```
@Entity
@Table(name = "fixed_phone", schema = "db_telco")
@NamedQuery(name="FixedPhone.findAll", query="SELECT fp FROM
FixedPhone fp")
public class FixedPhone implements Serializable{
    private static final long serialVersionUID = 1L;

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

    private String name;

    @OneToMany(mappedBy = "fixedPhone")
    private Collection<ServicePackage> servicePackages;
```

# Entity MobilePhone

```java
@Entity
@Table(name = "mobile_phone", schema = "db_telco")
@NamedQuery(name="MobilePhone.findAll", query="SELECT mp FROM MobilePhone mp")
public class MobilePhone implements Serializable{
	private static final long serialVersionUID = 1L;

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

	private String name;

	@Column(name = "num_minutes")
	private int numMinutes;

	@Column(name = "num_sms")
	private int numSms;

	@Column(name = "extra_min_fee")
	private float extraMinFee;

	@Column(name = "extra_sms_fee")
	private float extraSmsFee;

	@OneToMany(mappedBy = "mobilePhone")
	private Collection<ServicePackage> servicePackages;
```

# Entity MobileInternet

```
@Entity
@Table(name = "mobile_internet", schema = "db_telco")
@NamedQuery(name="MobileInternet.findAll", query="SELECT mi FROM 
MobileInternet mi")
public class MobileInternet implements Serializable{
    private static final long serialVersionUID = 1L;

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

    private String name;

    @Column(name = "num_giga")
    private int numGiga;

    @Column(name = "extra_giga_fee")
    private float extraGigaFee;

    @OneToMany(mappedBy = "mobileInternet")
    private Collection<ServicePackage> servicePackages;
```

# Entity OptionalProduct

```java
@Entity
@Table(name = "optional_product", schema = "db_telco")
@NamedQueries({
    @NamedQuery(name = "OptionalProduct.findAll", query = "SELECT op FROM
                                OptionalProduct op"),
    @NamedQuery(name = "OptionalProduct.findByName", query = "SELECT op FROM
                            OptionalProduct op WHERE op.name = :name") })
public class OptionalProduct implements Serializable {
    private static final long serialVersionUID = 1L;

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

    private String name;

    @Column(name = "monthly_fee")
    private float monthlyFee;

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

    @ManyToMany(mappedBy = "optionalProducts")
    private Collection<Order> orders;
```

# Entity Order

```java
@Entity
@Table(name = "orders", schema = "db_telco")
@NamedQueries({
    @NamedQuery(name = "Order.findRejectedOrders", query = "SELECT o FROM Order o  WHERE o.user = :user AND o.rejected = true"),
    @NamedQuery(name = "Order.findAllRejectedOrders", query = "SELECT o FROM Order o  WHERE o.rejected = true")})
public class Order implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name="tot_amount")
    private float totAmount;

    private boolean rejected;

    @Column(name="creation_date_time")
    @Temporal(TemporalType.TIMESTAMP)
    private Date creationDateTime;

    @Column(name="start_date")
    @Temporal(TemporalType.DATE)
    private Date startDate;

    @Column(name = "num_failed_payments")
    private int numFailedPayments;

    @ManyToOne
    @JoinColumn(name = "id_user")
    private User user;

    @OneToOne(mappedBy = "order")
    private ActivationSchedule activationSchedule;

    @ManyToOne
    @JoinColumn(name = "id_servicepackage")
    private ServicePackage servicePackage;

    @ManyToOne
    @JoinColumn(name = "id_validityperiod")
    private ValidityPeriod validityPeriod;

    @ManyToMany
    @JoinTable(name = "order_optional_product",
               joinColumns = @JoinColumn(name = "id_order"),
               inverseJoinColumns = @JoinColumn(name = "id_optionalproduct"))
    private List<OptionalProduct> optionalProducts;
```

# Entity ServicePackage

```java
@Entity
@Table(name = "service_package", schema = "db_telco")
@NamedQueries({
    @NamedQuery(name = "ServicePackage.findAll", query = "SELECT sp FROM ServicePackage sp"),
    @NamedQuery(name = "ServicePackage.findByName", query = "SELECT sp FROM ServicePackage sp WHERE sp.name = :name") })
public class ServicePackage implements Serializable {
    private static final long serialVersionUID = 1L;

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

    private String name;

    @ManyToOne
    @JoinColumn(name = "id_fixedphone")
    private FixedPhone fixedPhone;

    @ManyToOne
    @JoinColumn(name = "id_mobilephone")
    private MobilePhone mobilePhone;

    @ManyToOne
    @JoinColumn(name = "id_fixedinternet")
    private FixedInternet fixedInternet;

    @ManyToOne
    @JoinColumn(name = "id_mobileinternet")
    private MobileInternet mobileInternet;

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

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name = "service_package_validity_period",
               joinColumns = @JoinColumn(name = "id_servicepackage"),
               inverseJoinColumns = @JoinColumn(name = "id_validityperiod"))
    private List<ValidityPeriod> validityPeriods;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name = "service_package_optional_product",
               joinColumns = @JoinColumn(name = "id_servicepackage"),
               inverseJoinColumns = @JoinColumn(name = "id_optionalproduct"))
    private List<OptionalProduct> optionalProducts;
```

# Entity ValidityPeriod

```java
@Entity
@Table(name = "validity_period", schema = "db_telco")
@NamedQuery(name = "ValidityPeriod.findAll", query = "SELECT vp FROM
ValidityPeriod vp")
public class ValidityPeriod implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name = "monthly_fee")
    private float monthlyFee;

    @Column(name = "num_months")
    private int numMonths;

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

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

## Materialized views mapping

The additional tables are mapped in JPA as ordinary entities.

For simplicity, only the <u>owner</u> side (@JoinColumn) of the relationships is implemented: it is used in the Sales Report page to show all the needed data of service packages, optional products and validity periods.

# Entity AvgOptionalProduct

```java
@Entity
@Table(name = "avg_optional_product", schema = "db_telco")
@NamedQuery(name = "AvgOptionalProduct.findAll", query = "SELECT i
FROM
                                                  AvgOptionalProduct i")

public class AvgOptionalProduct implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @OneToOne
    @JoinColumn(name = "service_package")
    private ServicePackage servicePackage;

    @Column(name = "avg_op_products")
    private int avgOpProducts;
```

## Entity TotPurchasesPackage

```
@Entity
@Table(name = "tot_purchases_package", schema = "db_telco")
@NamedQuery(name = "TotPurchasesPackage.findAll", query = "SELECT i
FROM
    TotPurchasesPackage i")

public class TotPurchasesPackage implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @Column(name = "tot_purchases")
    private int totPurchases;

    @OneToOne
    @JoinColumn(name = "service_package")
    private ServicePackage servicePackage;
```

# Entity TotPurchasesPackageVPeriod

```
@Entity
@Table(name = "tot_purchases_package_vperiod", schema = "db_telco")
@NamedQuery(name = "TotPurchasesPackageVPeriod.findAll", query = "SELECT i
FROM
    TotPurchasesPackageVPeriod i")

public class TotPurchasesPackageVPeriod implements Serializable{
private static final long serialVersionUID = 1L;

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

    @OneToOne
    @JoinColumn(name = "service_package")
    private ServicePackage servicePackage;

    @OneToOne
    @JoinColumn(name = "validity_period")
    private ValidityPeriod validityPeriod;

    @Column(name="tot_purchases")
    private int totPurchases;
```

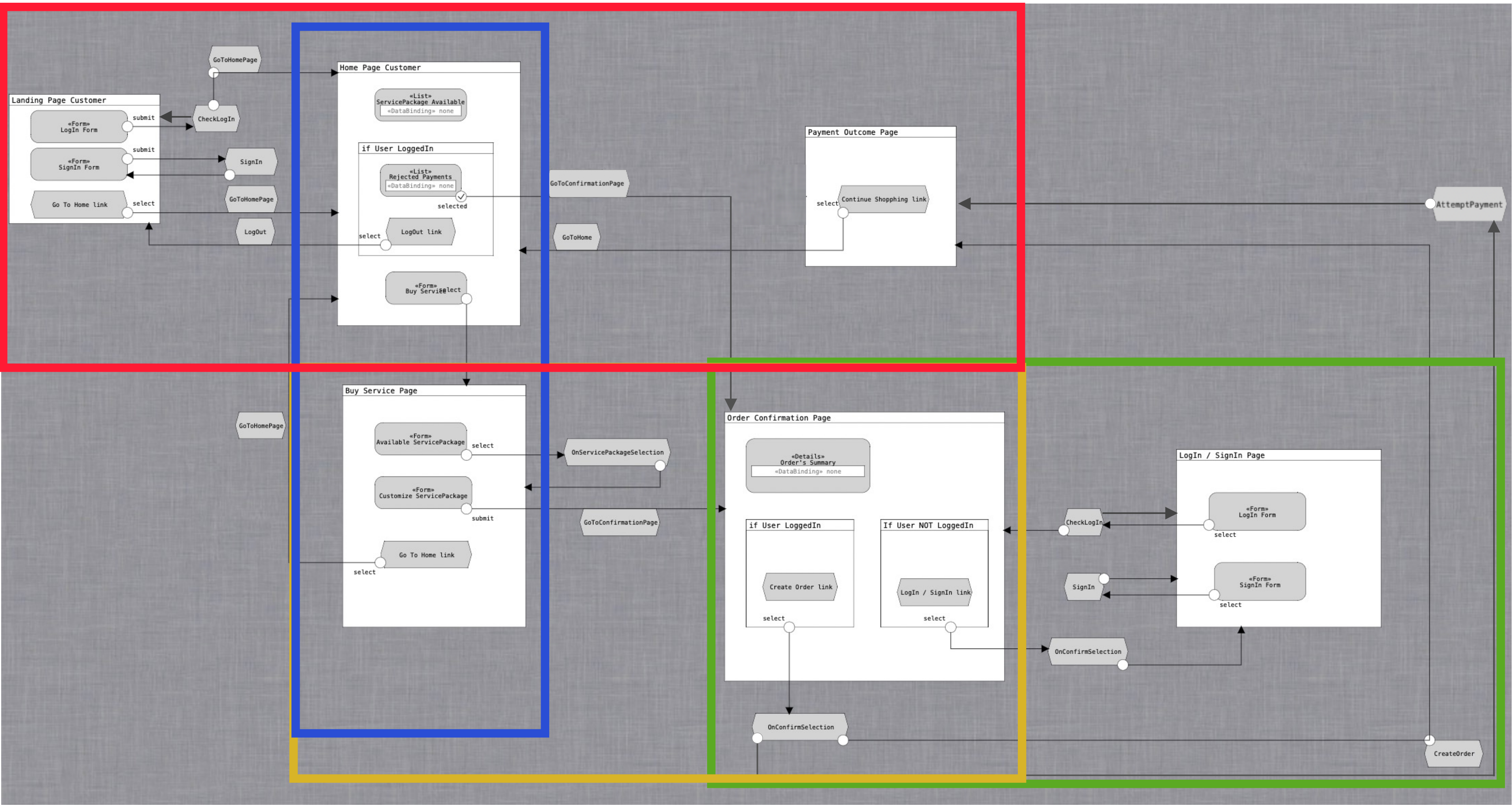
# Entity TotSalesPackage

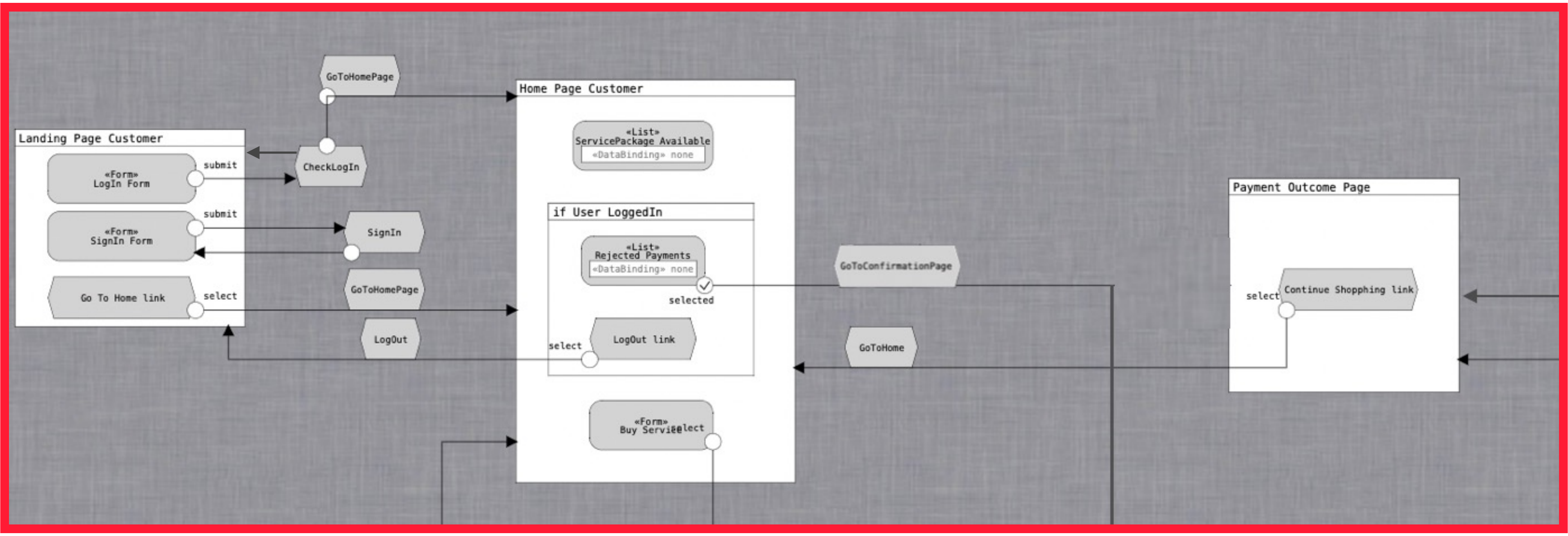
```java
@Entity
@Table(name = "tot_sales_package", schema = "db_telco")
@NamedQuery(name = "TotSalesPackage.findAll", query = "SELECT i FROM
TotSalesPackage i")

public class TotSalesPackage implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @OneToOne
    @JoinColumn(name = "service_package")
    private ServicePackage servicePackage;

    @Column(name = "tot_sales")
    private Double totSales;

    @Column(name = "sales_with_op_products")
    private Double salesWithOpProduct;
```

## Entity BestSeller

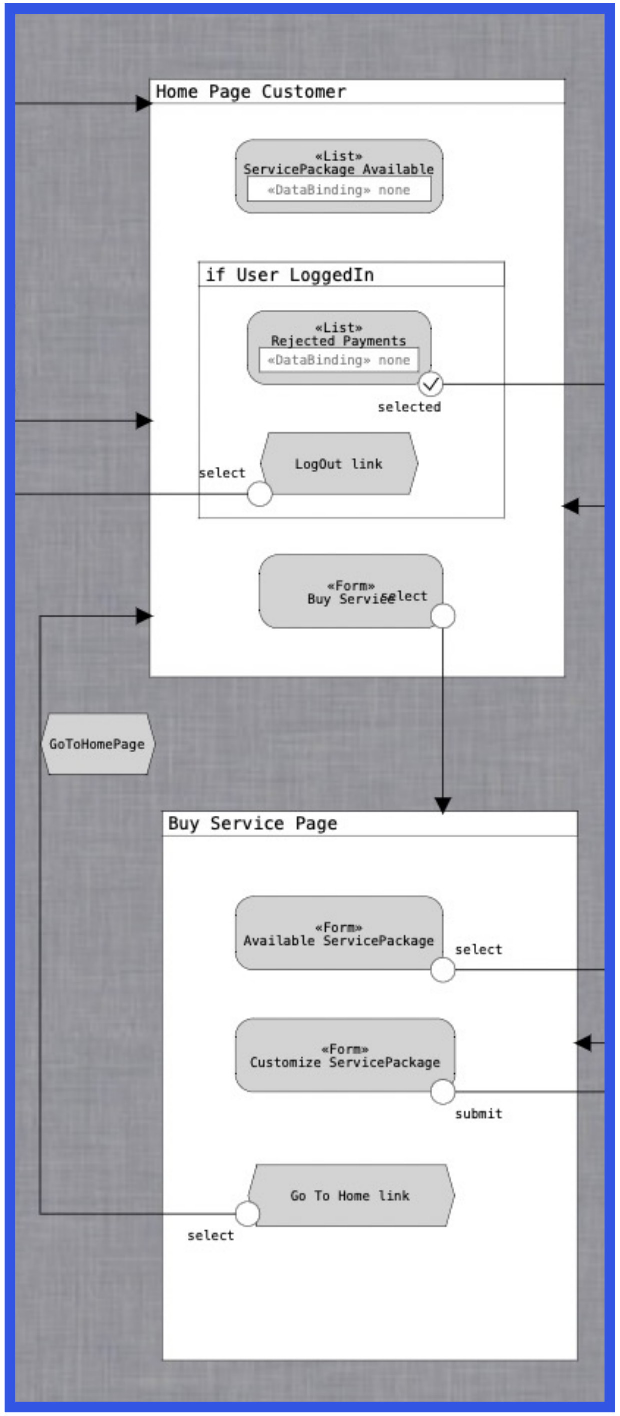
```
@Entity
@Table(name = "best_seller"), schema = "db_telco")
@NamedQuery(name = "BestSeller.findBestSeller", query = "SELECT bs
FROM BestSeller bs")

public class BestSeller implements Serializable {
    private static final long serialVersionUID = 1L;

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

    @OneToOne
    @JoinColumn(name = "op_product")
    private OptionalProduct bestSeller;

    @Column(name = "tot_sales")
    private Double totSale;
```

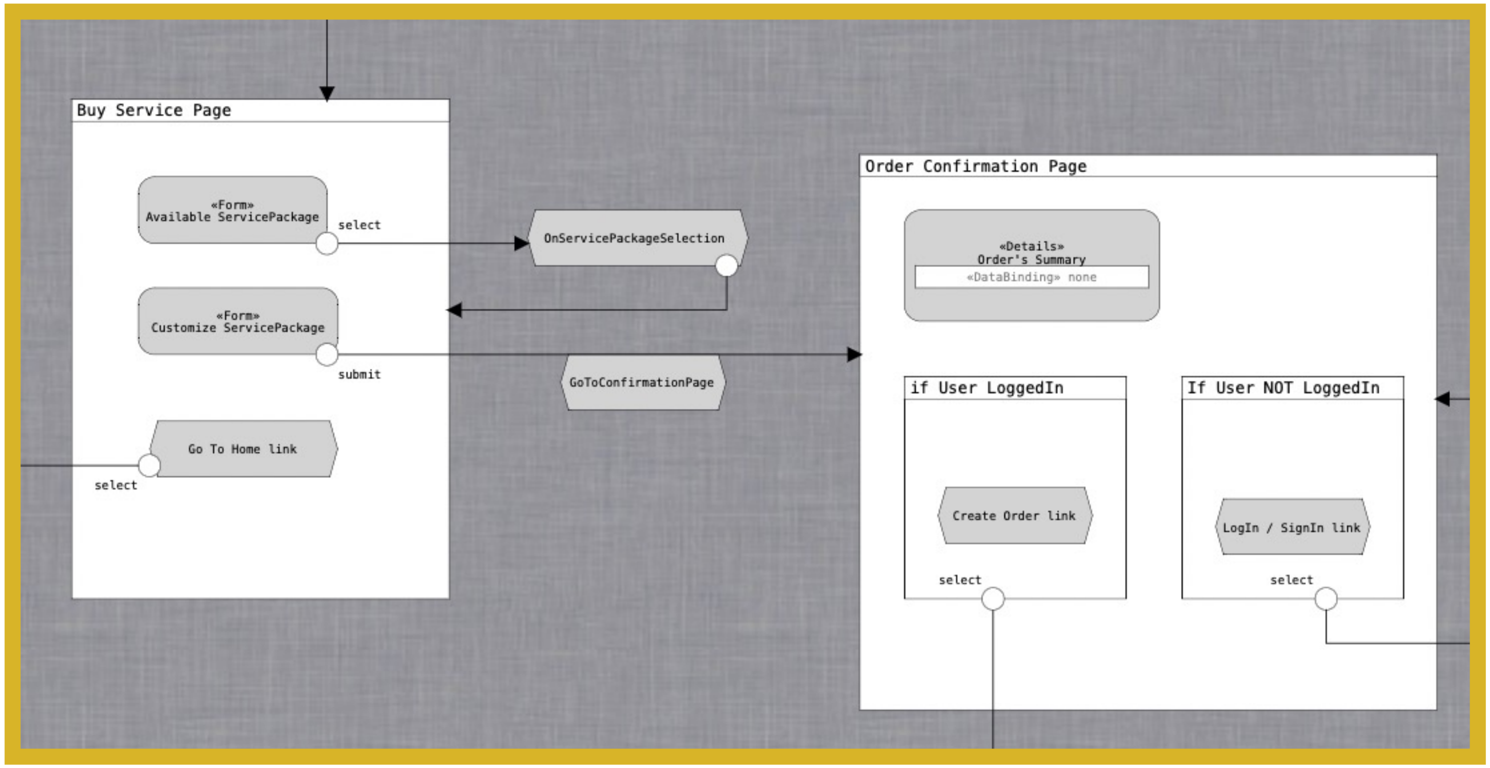
# Functional analysis of the interaction (Consumer APP)

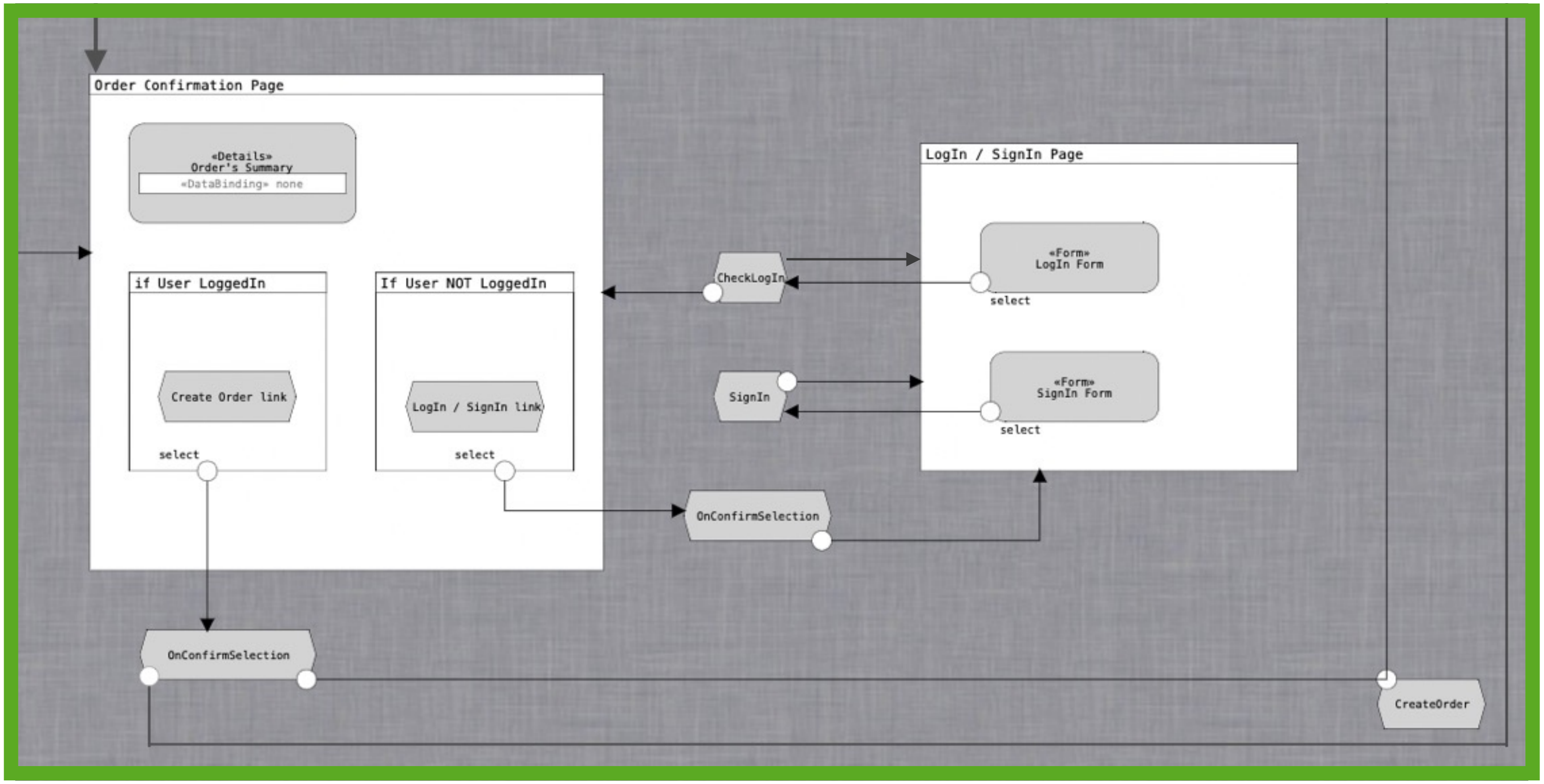




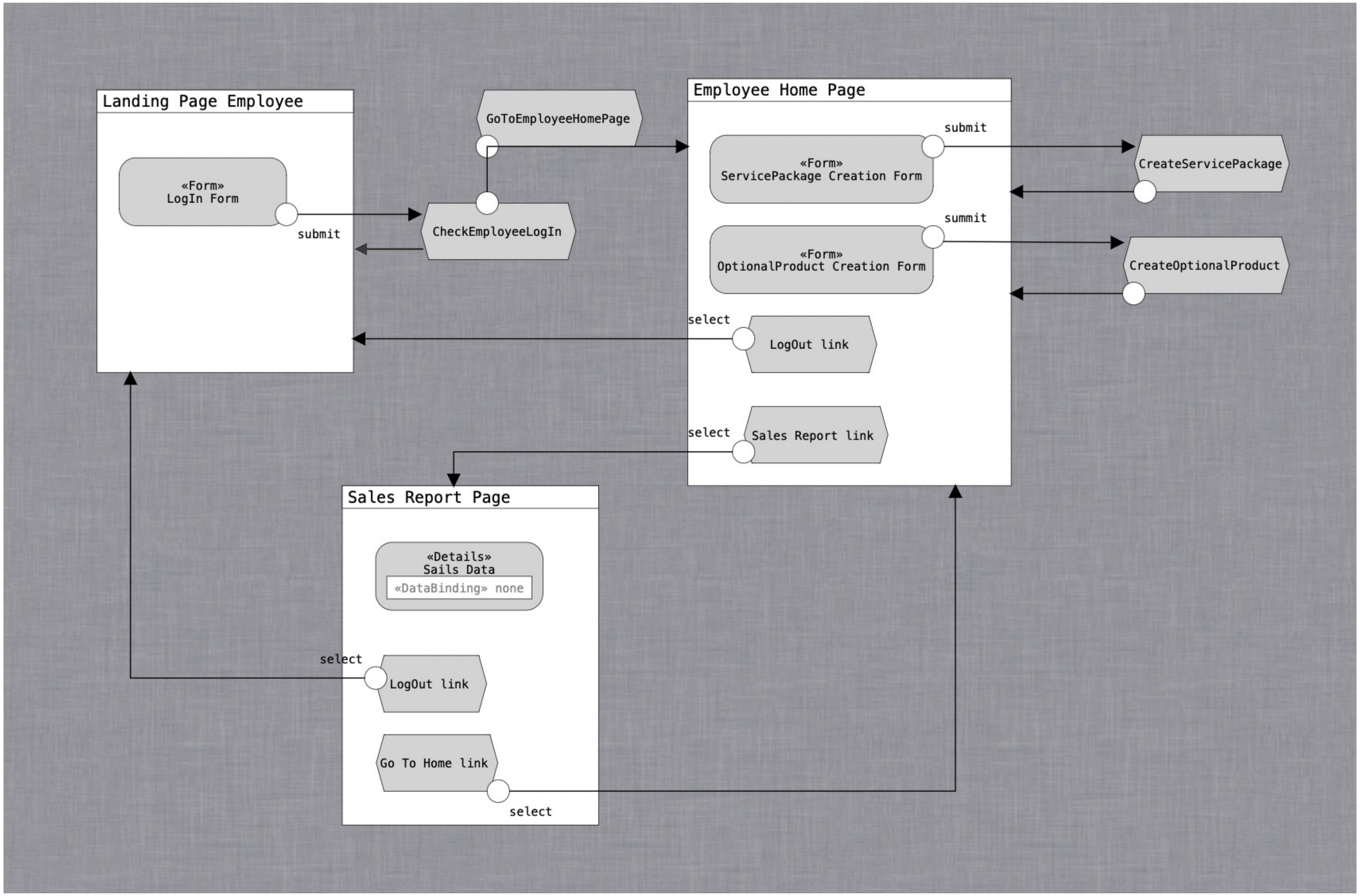
GoToHomePage
Home Page Customer
Landing Page Customer
«Form» LogIn Form
submit
CheckLogIn
«List» ServicePackage Available
«DataBinding» none
Payment Outcome Page
if User LoggedIn
«Form» SignIn Form
submit
SignIn
«List» Rejected Payments
«DataBinding» none
GoToConfirmationPage
GoToHomePage
Go To Home link
select
selected
select
Continue Shopphing link
LogOut
select
LogOut link
GoToHome
«Form» Buy Service
select


Home Page Customer
«List»
ServicePackage Available
«DataBinding» none
if User LoggedIn
«List»
Rejected Payments
«DataBinding» none
selected
select
LogOut link
«Form»
Buy Service
select
GoToHomePage
Buy Service Page
«Form»
Available ServicePackage
select
«Form»
Customize ServicePackage
submit
Go To Home link
select


Buy Service Page
«Form» Available ServicePackage
select
OnServicePackageSelection
«Form» Customize ServicePackage
submit
GoToConfirmationPage
Go To Home link
select
Order Confirmation Page
«Details» Order's Summary
«DataBinding» none
if User LoggedIn
Create Order link
select
If User NOT LoggedIn
LogIn / SignIn link
select


Order Confirmation Page
«Details»
Order's Summary
«DataBinding» none
if User LoggedIn
Create Order link
select
If User NOT LoggedIn
LogIn / SignIn link
select
CheckLogIn
SignIn
LogIn / SignIn Page
«Form»
LogIn Form
select
«Form»
SignIn Form
select
OnConfirmSelection
OnConfirmSelection
CreateOrder

# Functional analysis of the interaction (Employee APP)



## Client components

**Servlets**

- AttemptPayment
- CheckEmployeeLogin
- CheckLogin
- CreateOptionalProduct
- CreateOrder
- CreateServicePackage
- EmployeeLogout
- GoToBuyServicePage
- GoToConfirmationPage
- GoToEmployeeHomePage
- GoToHomePage
- GoToSalesReportPage
- Logout
- OnConfirmSelection
- OnServicePackageSelection
- SignIn

**Views**

- index
- employeeLogin
- BuyServicePage
- CompletePayment
- ConfirmationPage
- EmployeeHome
- Home
- LogInSignIn
- SalesReportPage

# Backend components

**Entities**

- ActivationSchedule
- AuditingTable
- Employee
- FixedInternet
- FixedPhone
- MobileInternet
- MobilePhone
- OptionalProduct
- Order
- ServicePackage
- User
- ValidityPeriod

**Materialized views**

- AvgOptionalProduct
- BestSeller
- TotPurchasesPackage
- TotPurchasesPackageVPeriod
- TotSalesPackage

# Backend components

**Business components (EJBs)**

@Stateless ActivationScheduleService

- void createActivationSchedule(Date startDate, int numMonth, Order order)

@Stateless AuditingTableService

- AuditingTable findAuditingTableByUserName(User user)
- void createAuditingEntry(User user)
- void updateAuditingEntry(User user, Order order, boolean decrease)
- void deleteAlert(User user, Order order)

@Stateless EmployeeService

- Employee checkCredentials(String usrn, String pwd)

@Stateless OptionalProductService

- OptionalProduct findOptionalProductById(int optionalProductId)
- List<OptionalProduct> createListOfSelectedOptionalProducts(List<String> optionalProductsId)
- float calculateTotAmountForOptionalProductsSelected(List<OptionalProduct> optionalProductList)
- List<OptionalProduct> findAllOptionalProducts()
- void createOptionalProduct(String name, float monthlyFee)

## Backend components

**Business components (EJBs)**

@Stateless OrderService

- List<Order> findRejectedOrders(User user)
- Order createOrder(User user, ServicePackage sp, List<OptionalProduct> opList,
  ValidityPeriod vp, Date startDate, float totAmount)
- void processOrder(boolean successfulPayment, Order order, User user, ValidityPeriod vp)
- void reProcessOrder(boolean successfulPayment, Order order, User user, ValidityPeriod vp)
- Order findOrderById(int idOrder)
- ValidityPeriod getValidityPeriodForOrder(Order order)
- ServicePackage getServicePackageForOrder(Order order)
- Date getStartDateForOrder(Order order)
- float getTotAmountForOrder(Order order)
- List<OptionalProduct> getOptionalProductsForOrder(Order order)
- float calculateTotExpenseForOrder(ValidityPeriod vp, List<OptionalProduct>
  optionalProductList)

## Backend components

**Business components (EJBs)**

@Stateless PhoneInternetService

- List<FixedInternet> findAllFixedInternetServices()
- List<FixedPhone> findAllFixedPhoneServices()
- List<MobileInternet> findAllMobileInternetServices()
- List<MobilePhone> findAllMobilePhoneServices()

@Stateless SalesReportService

- List<TotPurchasesPackage> findAllPurchasesPackage()
- List<TotPurchasesPackageVPeriod> findAllPurchasesPackageVPeriod()
- List<TotSalesPackage> findAllSalesPackage()
- List<AvgOptionalProduct> findAllAvgOpProduct()
- List<User> findAllInsolventUsers()
- List<Order> findAllRejectedOrders ()
- List<AuditingTable> findAllAlerts()
- BestSeller findBestSeller()

# Backend components

**Business components (EJBs)**

@Stateless ServicePackageService

- List<ServicePackage> findAllServicePackages()
- ServicePackage findServicePackageById(int servicePackageId)
- void createServicePackage (String name, int fixedPhoneId, int mobilePhoneId, int fixedInternetId, int mobileInternetId, List<Integer> validityPeriodIds, List<Integer> optionalProductIds)

@Stateless UserService

- User findUserById(User user)
- User checkCredentials (String usrn, String pwd)
- User registerUser(String usrn, String email, String pwd)

@Stateless ValidityPeriodService

- ValidityPeriod findValidityPeriodById(int validityPeriodId)
- List<ValidityPeriod> findAllValidityPeriods()