

Database 2 Project

2021-2022

Raffaello Fornasiere - 10790353
Elizaveta Lapiga - 10853126

Index

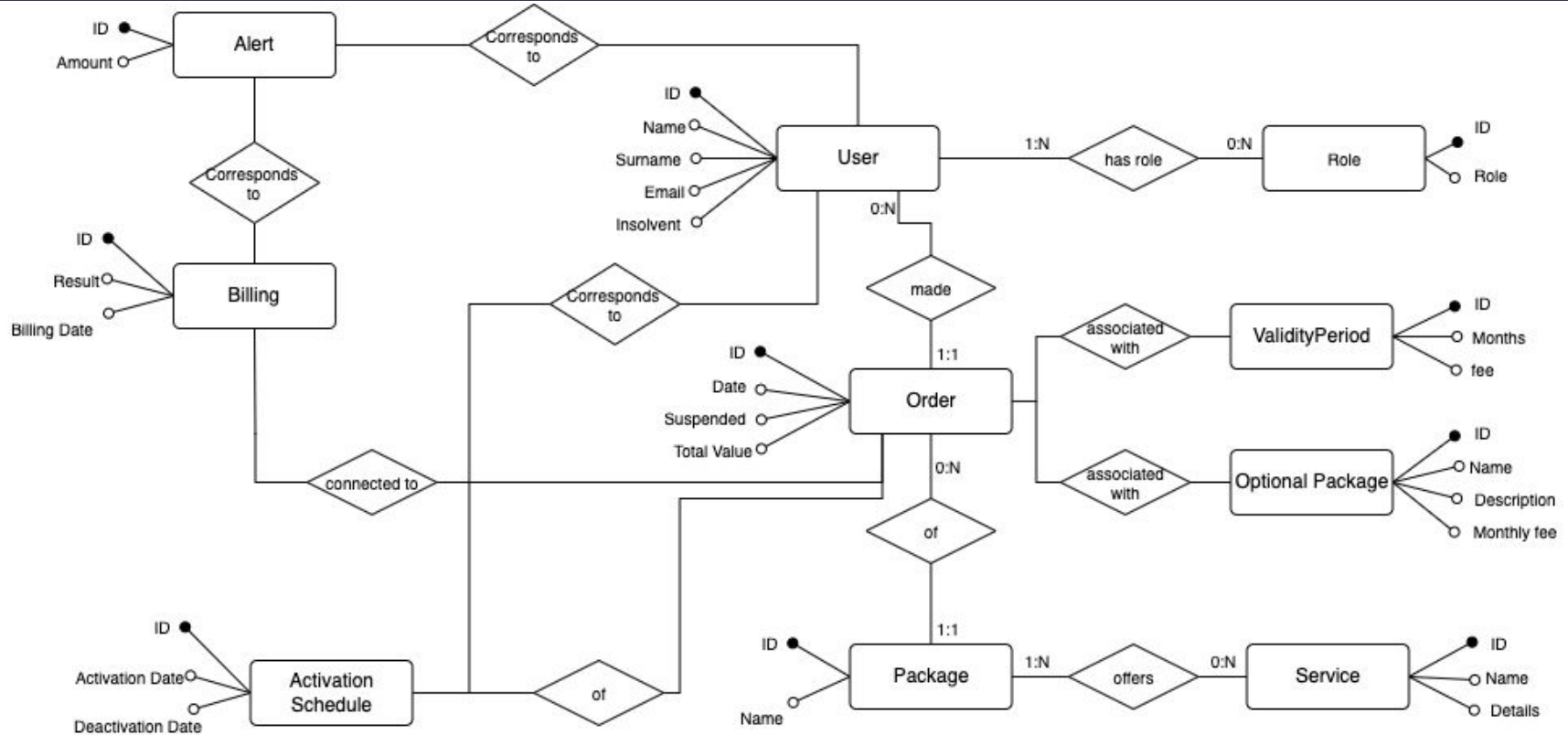
- Specification
 - Assumptions
- Conceptual and logical data models
 - ER diagram
 - Logical model
- Trigger design and code
- ORM relationship design with explanations
- Entities code
- Functional analysis
- List of components
- Sequence diagrams

Specifications – Further assumptions

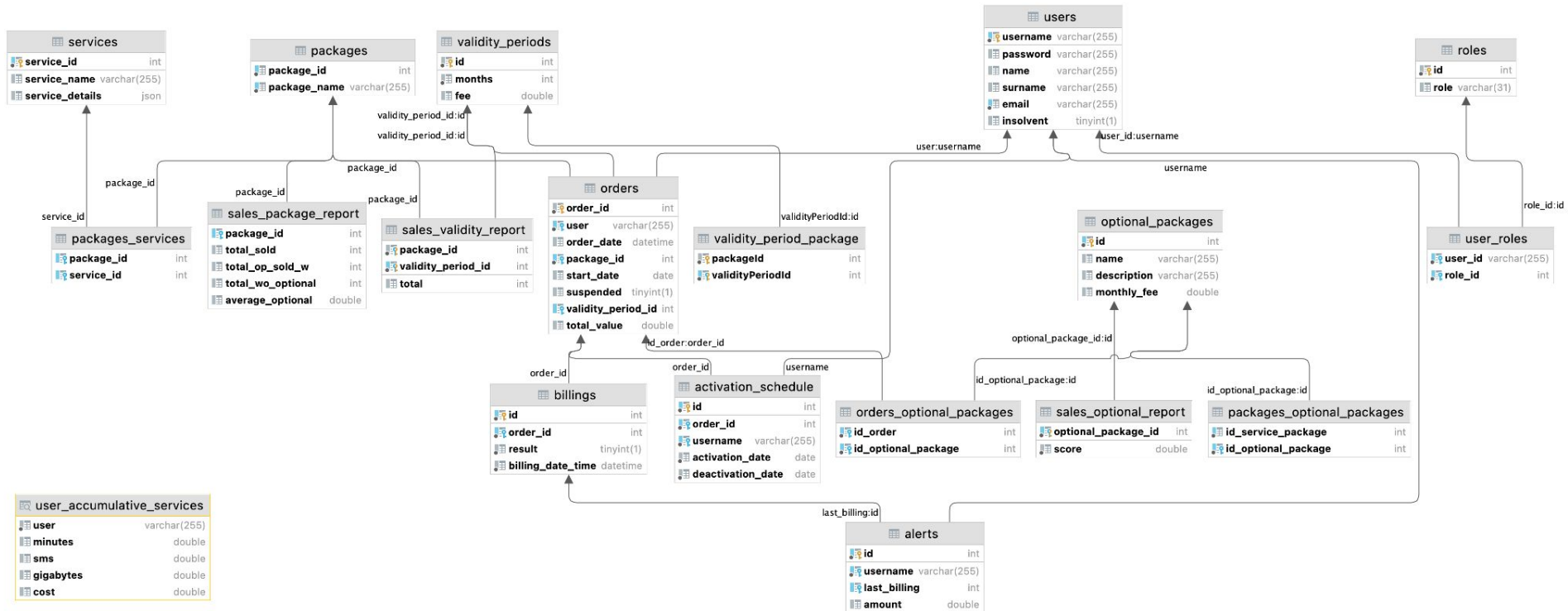
- The administrator, for testing purposes, have the ability to perform also all user actions and see all user pages.
- Products are not versioned for simplicity (this can cause some inconsistent data if current products are edited)

Diagrams

Conceptual and logical data models – ER Diagram



Relational Model Diagram



Entities code

Entities code – Entities

Activation schedule

```
create table activation_schedule
(
    id int auto_increment
        primary key,
    order_id int not null,
    username varchar(255) not null,
    activation_date date not null,
    deactivation_date date not null,
    constraint activation_schedule_id_uindex
        unique (id),
    constraint
        activation_schedule_orders_order_id_fk
        foreign key (order_id) references
        orders (order_id)
        on delete cascade,
    constraint
        activation_schedule_users_username_fk
        foreign key (username) references
        users (username)
);
```

Alerts

```
create table alerts
(
    id int auto_increment
        primary key,
    username varchar(255) not null,
    last_billing int null,
    amount double null,
    constraint alerts_pk
        unique (id, username),
    constraint alerts_billings_id_fk
        foreign key (last_billing) references
        billings (id),
    constraint alerts_users_username_fk
        foreign key (username) references
        users (username)
);
```

Billings

```
create table billings
(
    id int auto_increment
        primary key,
    order_id int not null,
    result tinyint(1) not null,
    billing_date_time datetime not null,
    constraint billings_id_uindex
        unique (id),
    constraint billings_orders_order_id_fk
        foreign key (order_id) references
        orders (order_id)
        on update cascade on delete cascade
);
```


Entities code –Entities

Optional Packages

```
create table optional_packages
(
    id            int auto_increment
        primary key,
    name          varchar(255) null,
    description    varchar(255) null,
    monthly_fee   double       null,
    constraint optional_packages__id_uindex
        unique (id)
);
```

Orders

```
create table orders
(
    order_id int auto_increment primary key,
    user      varchar(255) not null,
    order_date datetime     null,
    package_id int          not null,
    start_date date         null,
    validity_period_id int  null,
    suspended tinyint(1)   null,
    total_value double      null,
    constraint orders_packages_id_fk
        foreign key (package_id) references
packages (package_id)
        on update cascade on delete
cascade,
    constraint orders_validity_periods_id_fk
        foreign key (validity_period_id)
references validity_periods (id)
        on update cascade on delete set
null,
    constraint purchases_users_username_fk
        foreign key (user) references users
(username) on delete cascade
);
```

Packages

```
create table packages
(
    package_id int auto_increment,
    package_name varchar(255) not null,
    constraint packages_id_uindex
        unique (package_id),
    constraint packages_package_name_uindex
        unique (package_name)
);
```

Entities code

Roles

```
create table roles
(
    id    int          not null
        primary key,
    role  varchar(31)  null
);
```

Users

```
create table users
(
    username varchar(255) not null
        primary key,
    password  varchar(255) null,
    name      varchar(255) null,
    surname   varchar(255) null,
    email     varchar(255) not null,
    insolvent tinyint(1)   null,
    constraint users_email_uindex
        unique (email),
    constraint users_username_uindex
        unique (username)
);
```

Services

```
create table services
(
    service_id    int auto_increment
        primary key,
    service_name  varchar(255) null,
    service_details json       null,
    constraint services__service_id_uindex
        unique (service_id)
);
```

Validity Periods

```
create table validity_periods
(
    id    int auto_increment
        primary key,
    months int    not null,
    fee   double null
);
```

Entities code – Join Tables

Orders - Optional Packages

```
create table orders_optional_packages
(
    id_order          int null,
    id_optional_package int null,
    constraint
orders_optional_packages_optional_packages_id_fk
foreign key (id_optional_package)
references optional_packages (id) on delete
cascade,
constraint
orders_optional_packages_orders_order_id_fk
foreign key (id_order) references
orders (order_id)
on update cascade on delete
cascade
);
```

Packages - Optional Packages

```
create table packages_optional_packages
(
    id_service_package int not null,
    id_optional_package int not null,
    constraint packages_optional_packages_pk
unique (id_service_package,
id_optional_package),
constraint
packages_optional_packages_optional_packages_id_fk_2
foreign key (id_optional_package)
references optional_packages (id)
on delete cascade,
constraint
packages_optional_packages_optional_packages_id_fk
foreign key (id_optional_package)
references optional_packages (id)
on delete cascade
);
```

Packages - Services

```
create table packages_services
(
    package_id int null,
    service_id int null,
    constraint packages_services_pk
unique (package_id, service_id),
constraint
packages_services_packages_package_id_fk
foreign key (package_id) references
packages (package_id)
on delete cascade,
constraint
packages_services_services_service_id_fk
foreign key (service_id) references
services (service_id)
on delete cascade
);
```

Entities code – Join Tables

User - Roles

```
create table user_roles
(
  user_id varchar(255) not null,
  role_id int          not null,
  constraint user_roles_pk
    unique (user_id, role_id),
  constraint user_roles_roles_id_fk
    foreign key (role_id) references roles
    (id)
    on delete cascade,
  constraint user_roles_users_username_fk
    foreign key (user_id) references users
    (username)
    on update cascade on delete cascade
);
```

Validity Period - Package

```
create table validity_period_package
(
  packageId          int not null,
  validityPeriodId int not null,
  primary key (packageId, validityPeriodId),
  constraint
    validity_period_package_packages_package_id_fk
      foreign key (packageId) references packages
      (package_id)
      on delete cascade,
  constraint
    validity_period_package_validity_periods_id_fk
      foreign key (validityPeriodId) references
      validity_periods (id)
      on update cascade on delete cascade
);
```

Entities code – Materialized Views

Sales optional report

```
create table sales_optional_report
(
    optional_package_id int    not null
        primary key,
    score                double not null,
    constraint
SalesOptionalReport_optional_package_id_uinde
x
        unique (optional_package_id),
    constraint
sales_optional_report_optional_packages_id_fk
        foreign key (optional_package_id)
references optional_packages (id) on delete
cascade
);
```

Sales Package Report

```
create table sales_package_report
(
    package_id          int    null,
    total_sold          int    null,
    total_w_optional    int    null,
    total_wo_optional   int    null,
    average_optional    double null,
    constraint
package_statistics_packages_package_id_fk
        foreign key (package_id) references
packages (package_id)
        on update cascade on delete cascade
);
```

Sales Validity Report

```
create table sales_validity_report
(
    package_id          int not null,
    validity_period_id  int not null,
    total               int null,
    primary key (package_id,
validity_period_id),
    constraint
SalesValidityReport_packages_package_id_fk
        foreign key (package_id) references
packages (package_id)
        on update cascade on delete
cascade,
    constraint
sales_validity_report_validity_periods_id_fk
        foreign key (validity_period_id)
references validity_periods (id)
        on update cascade on delete cascade
);
```

Entities code – Views

User Accumulative Services

```
select partial.*, (select sum(o2.total_value) from orders o2 where o2.user = o.user group by o2.user) as cost
from (select o.user,
      sum(json_extract(REPLACE(s.service_details, '@', ''), '$.minutes')) as minutes,
      sum(json_extract(REPLACE(s.service_details, '@', ''), '$.sms')) as sms,
      sum(json_extract(REPLACE(s.service_details, '@', ''), '$.gigabytes')) as gigabytes
from packages p join orders o on p.package_id = o.package_id join packages_services ps on o.package_id = ps.package_id join services s
on ps.service_id = s.service_id
      join activation_schedule activation on o.order_id = activation.order_id
where activation.activation_date <= CURDATE() and curdate() < activation.deactivation_date
group by user) partial join orders o on o.user = partial.user group by partial.user;
```

Triggers

Triggers design

Triggers have been used to maintain materialized views, to maintain alert table, to update activation_schedule table and to mark/unmark insolvent users

There are 12 triggers in total:

- 1 for alert maintenance
- 1 for insolvent user maintenance
- 1 for activation schedule table maintenance
- 3 for sales package report table maintenance
- 3 for sales validity report table maintenance
- 3 for sales optional report table maintenance

Triggers Code – Alert Management

Alert Management

```
create trigger alert_management
after insert
on billings
for each row
BEGIN
declare failed_payments integer;
declare _username varchar(255);
SET _username = (select user from orders o where o.order_id = new.order_id);

SET failed_payments := (select count(*) from billings b where b.order_id in (select o.order_id from orders o where o.suspended =1
and b.order_id = o.order_id and o.user = _username));

IF (NEW.result = 0 and failed_payments >= 3
and (select count(*) from alerts a where a.username = _username) = 0) THEN
insert into alerts (username, last_billing, amount) values (_username, new.id, (select orders.total_value from orders where
orders.order_id = new.order_id));
end if;

# if a successful payment is added and there are previous failed payments
# it deletes the old alert
if (new.result = true and failed_payments > 0) THEN
delete from alerts a where a.username = _username;
end if;
end;
```

Triggers Code – Activation Schedule Insert

Activation Schedule Insert

```
create trigger activation_schedule_insert
after insert
on billings
for each row
BEGIN
declare _username varchar(255);
declare _activation_date date;
declare _deactivation_date date;

if (new.result = 1) then
set _username = (SELECT o.user from orders o where o.order_id = new.order_id);
set _activation_date = (SELECT start_date from orders o where o.order_id = new.order_id);
set _deactivation_date = DATE_ADD(_activation_date,
                                INTERVAL (
                                    select months
                                    from validity_periods
                                    join orders o2 on validity_periods.id = o2.validity_period_id
                                    where o2.order_id = NEW.order_id) MONTH);

insert into activation_schedule (order_id, username, activation_date, deactivation_date)
value (new.order_id, _username, _activation_date, _deactivation_date);
end if;

end;
```

Triggers Code – Mark Insolvent Users

Mark Insolvent Users

```
create trigger mark_insolvent_users
after insert
on billings
for each row
BEGIN
  IF (!new.result)
  THEN
    update users set insolvent = 1
    where username in (select distinct user
                       from orders
                       where order_id = NEW.order_id
                       );
  ELSE
    update users set insolvent = 0
    where username in (select distinct user
                       from orders
                       where order_id = NEW.order_id
                       );
  end if;
end;
```

Triggers Code – Sales Package Insert

Sales Package Insert

```
create trigger SalesPackage_insert
after insert
on orders
for each row
BEGIN
    declare _package_id int;
    declare _total_sold int;
    declare _total_op_sold_with int;
    declare _total_sold_without int;
    declare _average_optional_sold double;

    set _package_id := (select package_id from orders where order_id = new.order_id);
    set _total_sold := (select count(*) from orders where package_id = _package_id);
    set _total_op_sold_with := (select count(*) from orders join orders_optional_packages oop on orders.order_id = oop.id_order
                                where package_id = _package_id);

    set _total_sold_without := (select count(*) from (select o.order_id from orders o where o.package_id = _package_id) as orders
                                where order_id not in (select oop.id_order from orders_optional_packages oop));
    set _average_optional_sold := CAST(_total_op_sold_with as DOUBLE) / CAST(_total_sold as DOUBLE);

    if (_package_id in (select package_id from sales_package_report)) THEN
        update sales_package_report set total_sold = _total_sold, total_op_sold_w = _total_op_sold_with, total_wo_optional = _total_sold_without,
        average_optional = _average_optional_sold where package_id = _package_id;
    else
        insert into sales_package_report (package_id, total_sold, total_op_sold_w, total_wo_optional, average_optional)
        values (_package_id, _total_sold, _total_op_sold_with, _total_sold_without, _average_optional_sold);
    end if;
end;
```

Triggers Code – Sales Package Update

Sales Package Update

```
create trigger SalesPackage_update
  after update
  on orders
  for each row
BEGIN
  declare _total_sold int;
  declare _total_op_sold_with int;
  declare _total_sold_without int;
  declare _average_optional_sold double;
  declare old [...];

  set _total_sold := (select count(*) from orders where package_id = new.package_id);
  set _total_op_sold_with := (select count(*) from (select o.order_id from orders o where o.package_id = new.package_id) as orders
                             where order_id in (select oop.id_order from orders_optional_packages oop));
  set _total_sold_without := (select count(*) from (select o.order_id from orders o where o.package_id = new.package_id) as orders
                              where order_id not in (select oop.id_order from orders_optional_packages oop));

  if (_total_sold != 0) then set _average_optional_sold := CAST(_total_op_sold_with as DOUBLE) / CAST(_total_sold as DOUBLE);
  else set _average_optional_sold := 0; end if;

  if (new.package_id in (select package_id from sales_package_report)) THEN
    update sales_package_report
    set total_sold = _total_sold, total_op_sold_w = _total_op_sold_with, total_wo_optional = _total_sold_without, average_optional = _average_optional_sold
    where package_id = new.package_id;
  else
    insert into sales_package_report (package_id, total_sold, total_op_sold_w, total_wo_optional, average_optional)
    values (old.package_id, _total_sold, _total_op_sold_with, _total_sold_without, _average_optional_sold);
  end if;
  *** same for old orders ***
```

Triggers Code – Sales Package Delete

Sales Package Delete

```
create trigger SalesPackage_delete
after delete
on orders
for each row
BEGIN
    declare _total_sold int;
    declare _total_op_sold_with int;
    declare _total_sold_without int;
    declare _average_optional_sold double;

    set _total_sold := (select count(*) from orders where package_id = old.package_id);
    set _total_op_sold_with := (select count(*) from (select o.order_id from orders o where o.package_id = old.package_id) as orders
                                where order_id in (select oop.id_order from orders_optional_packages oop));
    set _total_sold_without := (select count(*) from (select o.order_id from orders o where o.package_id = old.package_id) as orders
                                where order_id not in (select oop.id_order from orders_optional_packages oop));
    if (_total_sold != 0) then set _average_optional_sold := CAST(_total_op_sold_with as DOUBLE) / CAST(_total_sold as DOUBLE);
    else set _average_optional_sold := 0; end if;

    if (old.package_id in (select package_id from sales_package_report)) THEN
        update sales_package_report
        set total_sold = _total_sold, total_op_sold_w = _total_op_sold_with, total_wo_optional = _total_sold_without, average_optional =
        _average_optional_sold where package_id = old.package_id;
    else
        insert into sales_package_report (package_id, total_sold, total_op_sold_w, total_wo_optional, average_optional)
        values (old.package_id, _total_sold, _total_op_sold_with, _total_sold_without, _average_optional_sold);
    end if;
end;
```

Triggers Code – Sales Validity Manager

Sales Validity Manager

```
create trigger sales_validity_manager
  after insert
  on orders
  for each row
BEGIN
  declare _package_id int;
  declare _validity_id int;
  declare _total int;

  set _package_id := new.package_id;
  set _validity_id := new.validity_period_id;
  set _total := (select count(*) from orders where _package_id = package_id and _validity_id = validity_period_id);

  if ((_package_id, _validity_id) in
      (select svr.package_id, svr.validity_period_id from sales_validity_report svr))
  then
    update sales_validity_report svr
    set svr.package_id = _package_id, svr.total = _total
    where svr.package_id = _package_id and svr.validity_period_id = _validity_id;
  else
    insert into sales_validity_report (package_id, validity_period_id, total) values (_package_id, _validity_id, _total);
  end if;
end;
```

Triggers Code – Sales Validity Manager

Sales Validity Manager

```
create trigger sales_validity_manager
  after insert
  on orders
  for each row
BEGIN
  declare _package_id int;
  declare _validity_id int;
  declare _total int;

  set _package_id := new.package_id;
  set _validity_id := new.validity_period_id;
  set _total := (select count(*) from orders where _package_id = package_id and _validity_id = validity_period_id);

  if ((_package_id, _validity_id) in
      (select svr.package_id, svr.validity_period_id from sales_validity_report svr))
  then
    update sales_validity_report svr
    set svr.package_id = _package_id, svr.total = _total
    where svr.package_id = _package_id and svr.validity_period_id = _validity_id;
  else
    insert into sales_validity_report (package_id, validity_period_id, total) values (_package_id, _validity_id, _total);
  end if;
end;
```


Triggers Code – Sales Validity Delete

Sales Validity Delete

```
create trigger sales_validity_update
after update
on orders
for each row
BEGIN
    declare _package_id int;
    declare _validity_id int;
    declare _total int;
    declare old_[...]

    if (new.validity_period_id is not null) then
        set _package_id := new.package_id;
        set _validity_id := new.validity_period_id;
        set _total := (select count(*) from orders o where _package_id = o.package_id and _validity_id = o.validity_period_id );

        if ((_package_id, _validity_id) in (select svr.package_id, svr.validity_period_id from sales_validity_report svr))
        then
            update sales_validity_report svr set svr.package_id = _package_id, svr.total = _total
            where svr.package_id = _package_id and svr.validity_period_id = _validity_id;
        else
            insert into sales_validity_report (package_id, validity_period_id, total)
            values (_package_id, _validity_id, _total);
        end if;
    end if;
    ***same for old_ ***
end;
```

Triggers Code – Sales Optional Delete

Sales Optional Delete

```
create trigger sales_validity_insert
after insert
on orders
for each row
BEGIN
    declare _package_id int;
    declare _validity_id int;
    declare _total int;

    set _package_id := new.package_id;
    set _validity_id := new.validity_period_id;

    set _total := (
        select count(*) from orders where _package_id = package_id and _validity_id = validity_period_id
    );

    if ((_package_id, _validity_id) in
        (select svr.package_id, svr.validity_period_id from sales_validity_report svr))
    then
        update sales_validity_report svr
        set svr.package_id = _package_id, svr.total = _total
        where svr.package_id = _package_id and svr.validity_period_id = _validity_id;
    else
        insert into sales_validity_report (package_id, validity_period_id, total)
        values (_package_id, _validity_id, _total);
    end if;
end;
```

Triggers Code – Sales Optional Insert

Sales Optional Insert

```
create trigger sales_optional_insert
  after insert
  on orders_optional_packages
  for each row
BEGIN

  if (new.id_optional_package not in (select s.optional_package_id from sales_optional_report s)) then
    insert into sales_optional_report (optional_package_id, score) value (new.id_optional_package, 0);
  end if;

  update sales_optional_report s
  set s.score = (CAST((select count(*)
                        from orders_optional_packages oop
                        where oop.id_optional_package = s.optional_package_id) as DOUBLE) /
                CAST((select count(*) from orders_optional_packages) as DOUBLE));

end;
```

Triggers Code – Sales Optional Update

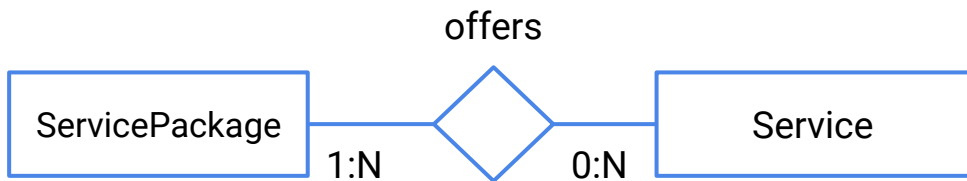
Sales Optional Update

```
create trigger sales_optional_update
  after update
  on orders_optional_packages
  for each row
BEGIN
  if (new.id_optional_package not in (select s.optional_package_id from sales_optional_report s)) then
    insert into sales_optional_report (optional_package_id, score) value (new.id_optional_package, 0);
  end if;

  update sales_optional_report s
  set s.score = (CAST((select count(*)
                      from orders_optional_packages oop
                      where oop.id_optional_package = s.optional_package_id) as DOUBLE) /
                CAST((select count(*) from orders_optional_packages) as DOUBLE));
end;
```

ORM

Relationship



Package → Service

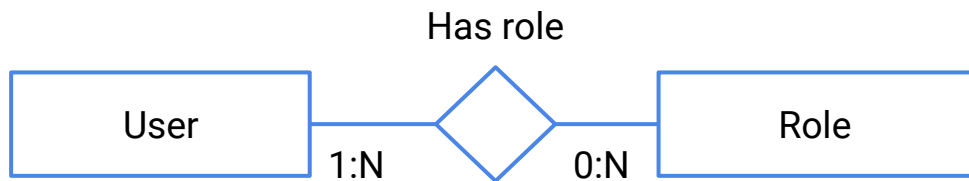
@ManyToMany

- Eager Fetch

Service → Package

Not mapped

Relationship



User → Role

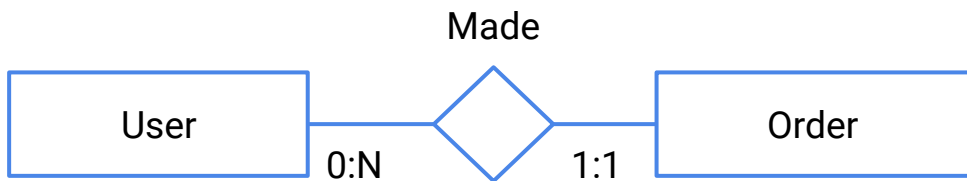
@ManyToMany

- Eager Fetch

Role → User

Not mapped

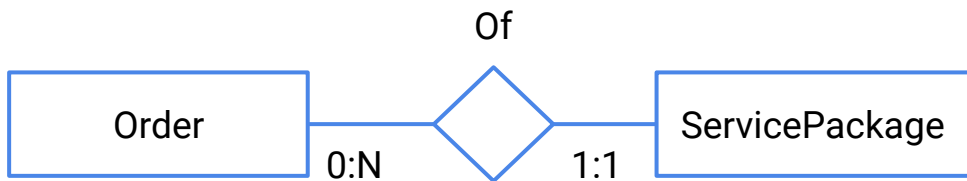
Relationship



User → Order
Not mapped

Order → User
@ManyToOne
• Eager Fetch (default)

Relationship



Order → Package

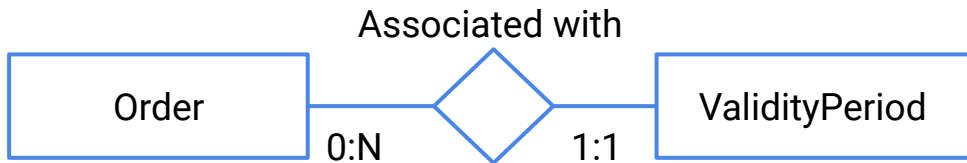
@ManyToOne

- Eager Fetch (default)

Package → Order

Not mapped

Relationship



Order \longrightarrow ValidityPeriod

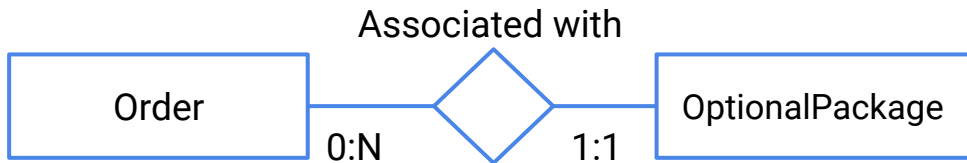
@ManyToOne

- Lazy Fetch

ValidityPeriod \longrightarrow Order

Not mapped

Relationship



Order \longrightarrow OptionalPackage

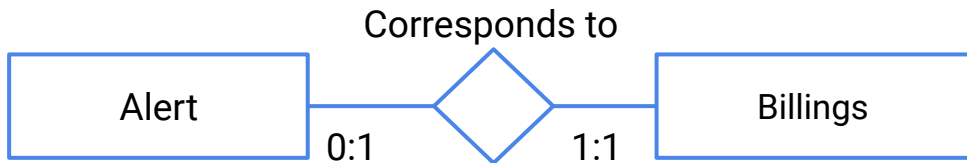
@ManyToOne

- Eager Fetch

OptionalPackage \longrightarrow Order

Not mapped

Relationship



Alert → Billings

@OneToOne

- Eager Fetch (default)

Billings → Alert

Not mapped

Entities Code

Service Entity

Service Entity

```
@Entity
@Table(name = "services")
public class ServiceEntity implements Serializable {
    @Id
    @Column(name = "service_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long serviceId;

    @Column(name = "service_name")
    private String serviceName;

    @Column(name = "service_details")
    private String serviceDetails;
}
```

Alerts Entity

Alerts Entity

```
@Entity
@EntityListeners(ReadOnlyEntity.class)
@Table(name = "alerts")
public class AlertEntity {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

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

    @OneToOne
    @JoinColumn(name = "last_billing")
    BillingEntity billing;
}
```

Billings Entity

Billings Entity

```
@Entity
@Table(name = "billings")
public class BillingEntity {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Column(name = "order_id")
    private Integer orderId;

    @Column(name = "result")
    private Boolean result;

    @Column(name = "billing_date_time")
    private LocalDateTime billingDateTime;
}
```


OptionalPackages Entity

OptionalPackages Entity

```
@Entity
@Table(name = "optional_packages")
public class OptionalPackageEntity {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "name")
    private String name;

    @Column(name = "description")
    private String description;

    @Column(name = "monthly_fee")
    private Double monthlyFee;
}
```

Order Entity

Order Entity

```
@Entity
@Table(name = "orders")
public class OrderEntity {
    @Id
    @Column(name = "order_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "order_date")
    private LocalDateTime orderDate;

    @ManyToOne
    @JoinColumn(name = "user")
    private UserEntity user;

    @ManyToOne
    @JoinColumn(name = "package_id")
    private ServicePackageEntity servicePackageEntity;

    @Column(name = "start_date")
    private LocalDate startDate;
```

```
@ManyToOne
@JoinColumn(name = "validity_period_id")
private ValidityPeriodEntity validityPeriod;

@Column(name = "suspended")
private Boolean suspended;

@Column(name = "total_value")
private Double totalValue;

@ManyToMany(fetch = FetchType.EAGER)
@ToString.Exclude
@JoinTable(
    name = "orders_optional_packages",
    joinColumns = {@JoinColumn(name = "id_order")},
    inverseJoinColumns = {@JoinColumn(name =
        "id_optional_package")})
private List<OptionalPackageEntity> optionalPackages;
}
```

Role Entity

Role Entity

```
@Entity
@Table(name = "roles")
public class RoleEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Long id;

    @Enumerated(EnumType.STRING)
    @Column(name = "role")
    private Role role;
}
```

SalesOptionalReport Entity

SalesOptionalReport Entity

```
@Entity
@EntityListeners(ReadOnlyEntity.class)
@Table(name = "sales_optional_report")
public class SalesOptionalReportEntity {
    @Id
    @Column(name = "optional_package_id")
    private Long id;

    @OneToOne
    @JoinColumn(name = "optional_package_id", insertable = false, updatable = false)
    private OptionalPackageEntity optionalPackage;

    @Column(name = "score")
    private Double score;
}
```

SalesPackageReport Entity

SalesPackageReport Entity

```
@Entity
@EntityListeners(ReadOnlyEntity.class)
@Table(name = "sales_package_report")
public class SalesPackageReportEntity {
    @Id
    @Column(name = "package_id")
    private Long id;

    @ManyToOne
    @JoinColumn(name = "package_id", insertable = false,
        updatable = false)
    private ServicePackageEntity servicePackage;

    @Column(name = "total_sold")
    private Integer totalSold;
```

```
    @Column(name = "total_w_optional")
    private Integer totalWOptional;

    @Column(name = "total_wo_optional")
    private Integer totalWoOptional;

    @Column(name = "average_optional")
    private Double averageOptional;
}
```

SalesValidityReport Entity

SalesValidityReport Entity

```
@Entity
@EntityListeners(ReadOnlyEntity.class)
@Table(name = "sales_validity_report")
public class SalesValidityReportEntity {
    @EmbeddedId
    private SalesValidityId salesValidityId;

    @Column(name = "total")
    private Integer total;
}
```

```
@Embeddable
public class SalesValidityId implements Serializable {
    @ManyToOne
    @JoinColumn(name = "validity_period_id", nullable =
false)
    private ValidityPeriodEntity validityPeriod;

    @ManyToOne
    @JoinColumn(name = "package_id", nullable = false)
    private ServicePackageEntity servicePackage;
}
```

ServicePackage Entity

ServicePackage Entity

```
@Entity
@Table(name = "packages")
public class ServicePackageEntity implements Serializable {
    @Id
    @Column(name = "package_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "package_name")
    private String name;

    @ManyToMany(fetch = FetchType.EAGER)
    @ToString.Exclude
    @JoinTable(
        name = "packages_services", schema = "db2_pdb",
        joinColumns = {@JoinColumn(name = "package_id", referencedColumnName = "package_id")},
        inverseJoinColumns = {@JoinColumn(name = "service_id", referencedColumnName = "service_id")})
    private Set<ServiceEntity> services;
}
```

User Entity

User Entity

```
@Entity
@Table(name = "users", uniqueConstraints = {
    @UniqueConstraint(name = "users_username_uindex",
        columnNames = {"username"}),
    @UniqueConstraint(name = "users_email_uindex",
        columnNames = {"email"})
})
public class UserEntity {
    @Id
    @Column(name = "username", nullable = false)
    private String username;

    @Column(name = "name")
    private String name;

    @Column(name = "password")
    private String password;

    @Column(name = "surname")
    private String surname;
```

```
    @Column(name = "email")
    private String email;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name = "user_roles",
        joinColumns = @JoinColumn(name = "user_id"),
        inverseJoinColumns = @JoinColumn(name =
            "role_id"))

    @ToString.Exclude
    private Set<RoleEntity> roles;

    @Column(name = "insolvent")
    private Boolean insolvent;

}
```


ValidityPeriod Entity

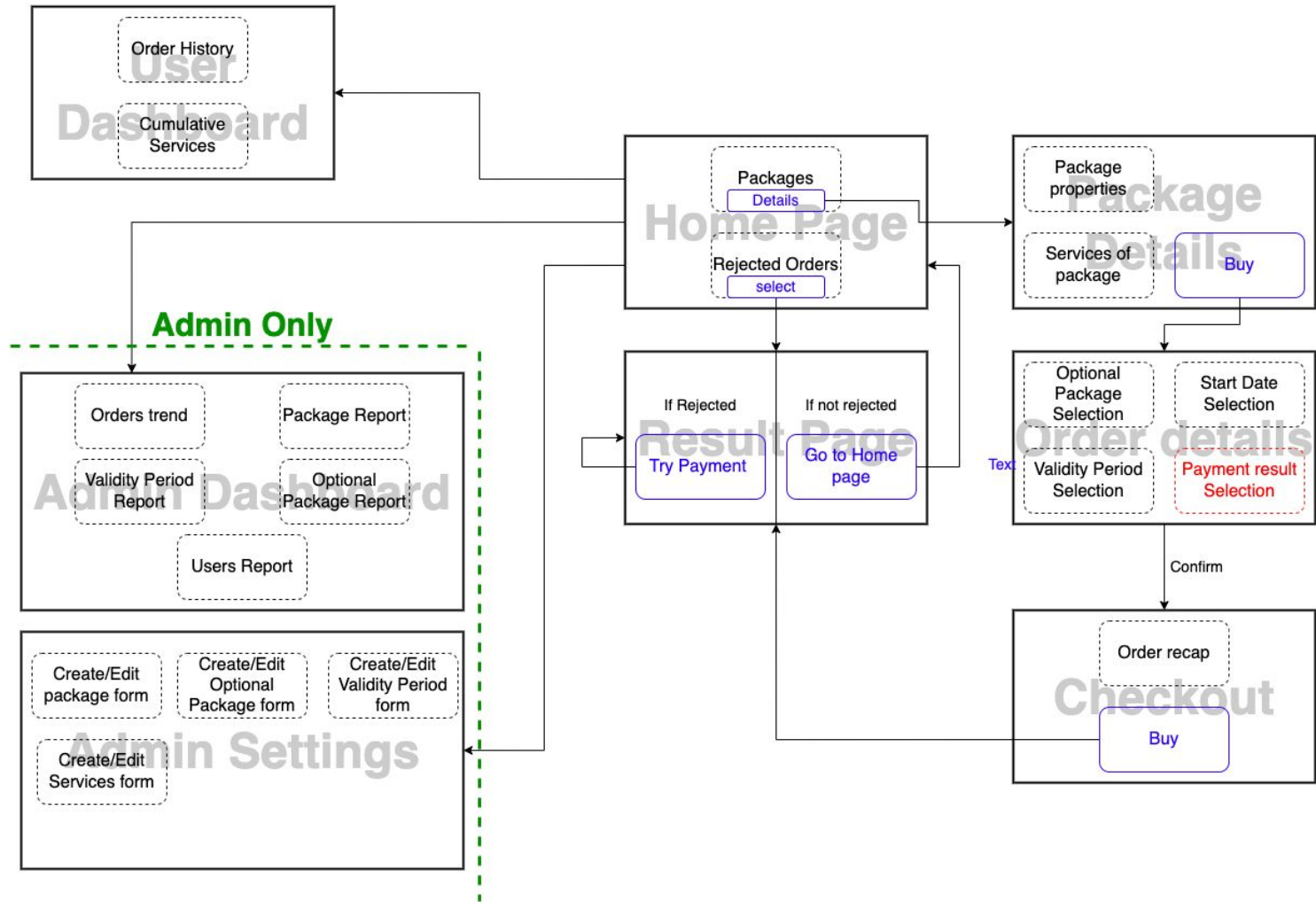
ValidityPeriod Entity

```
@Entity
@Table(name = "validity_periods")
public class ValidityPeriodEntity implements Serializable {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "months")
    private Integer months;

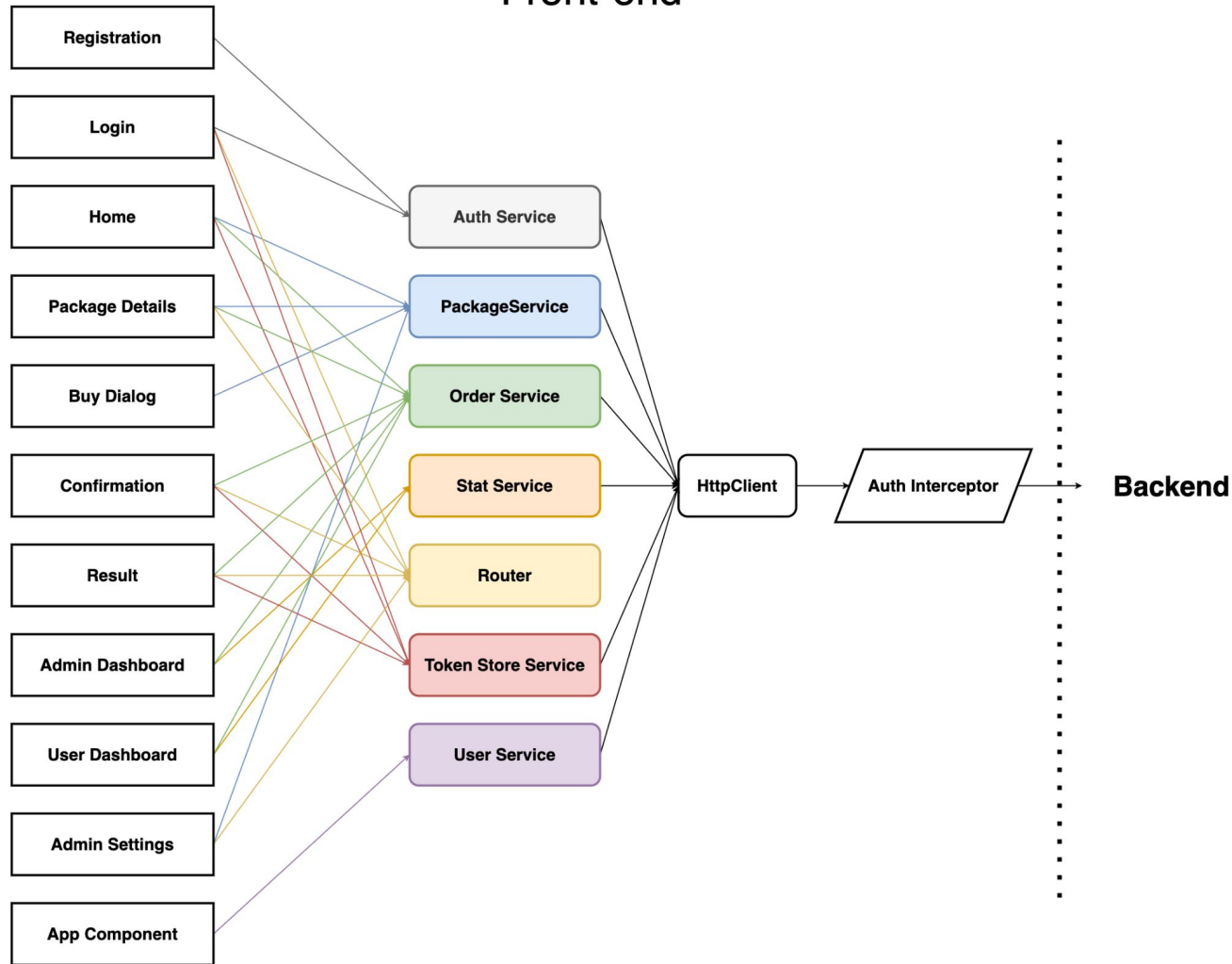
    @Column(name = "fee")
    private Double fee;
}
```

Interaction Diagram

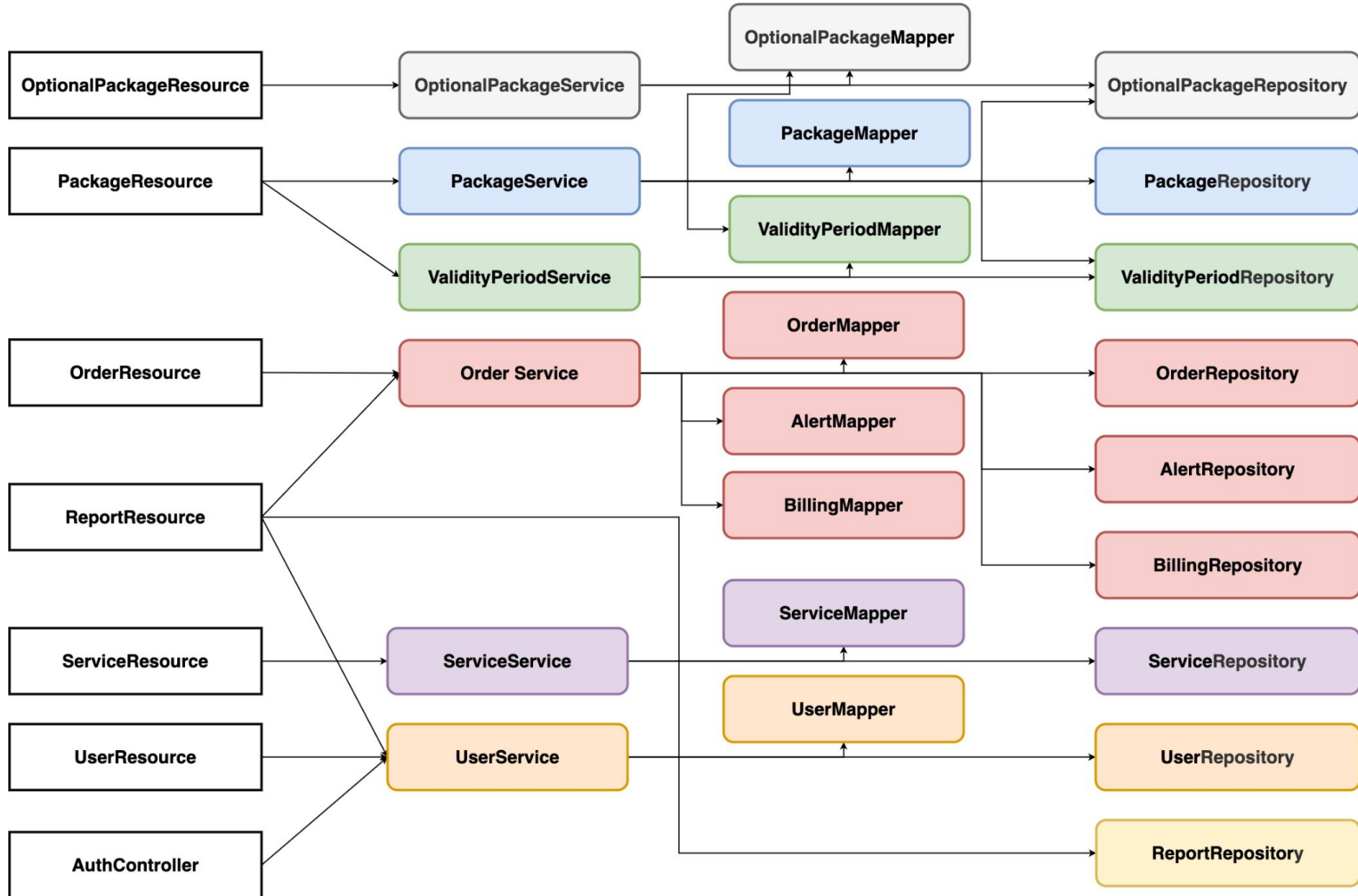


Components Diagram

Front-end

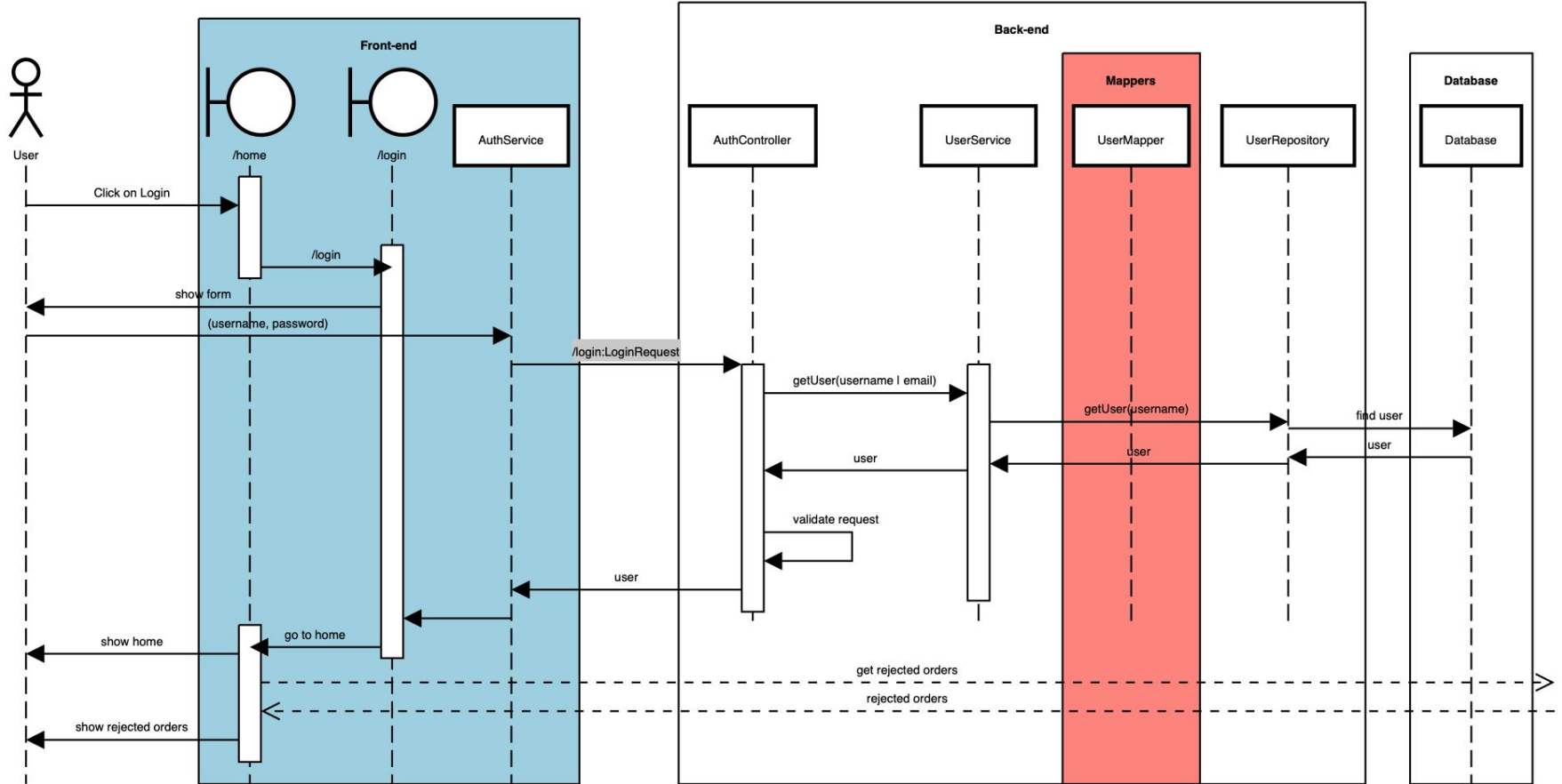


Back-end

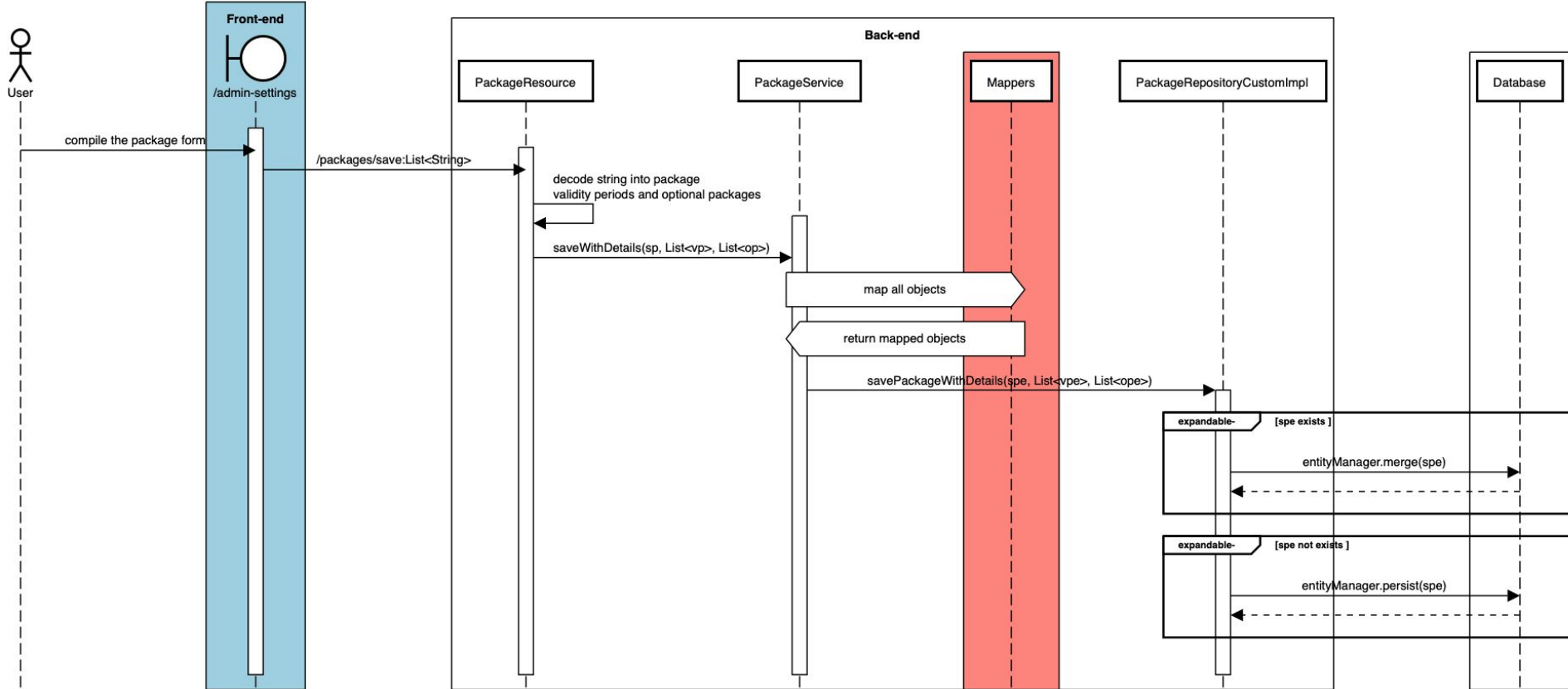


Sequence Diagram

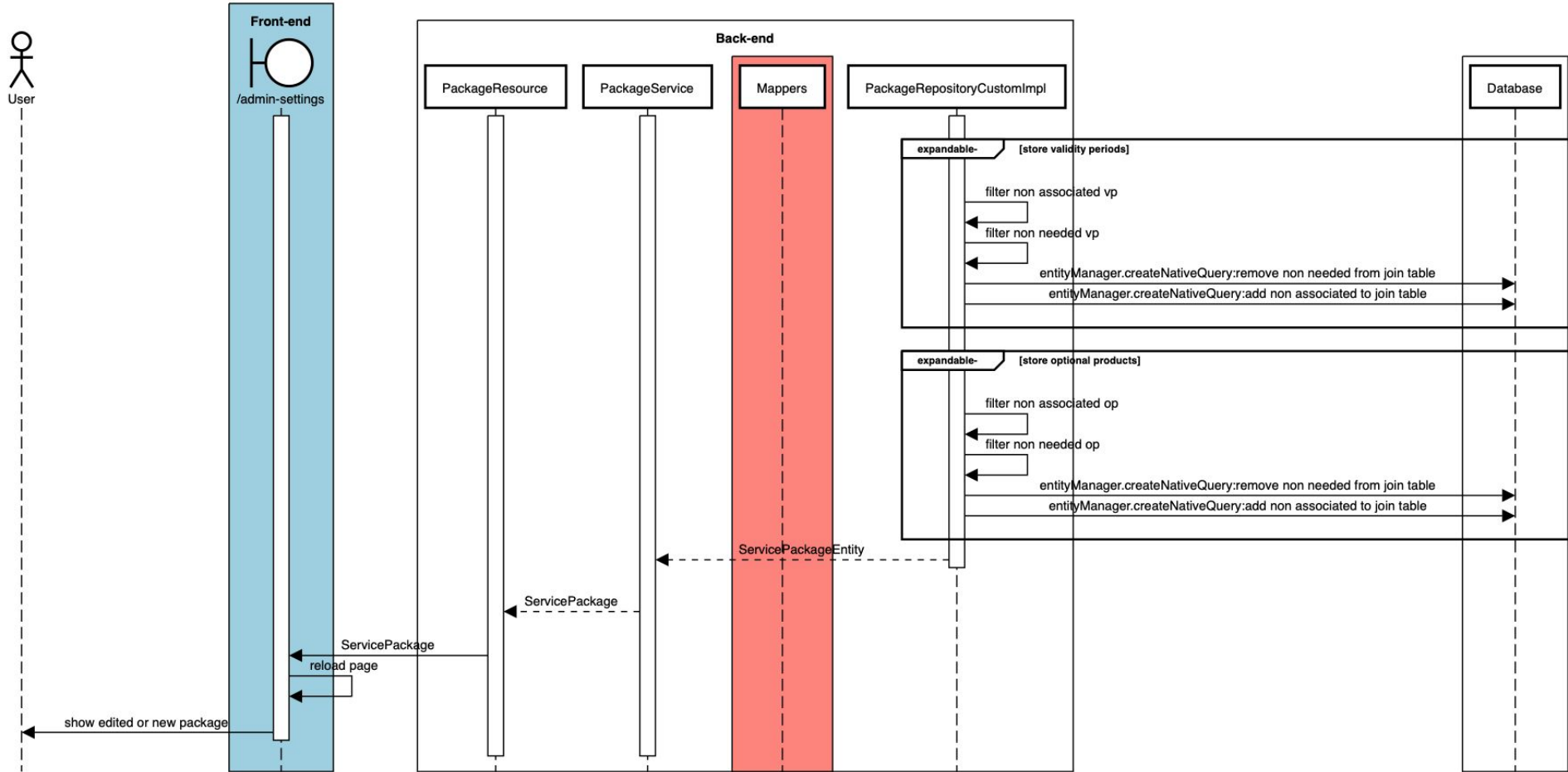
Login



Edit/Create Package



Edit/Create Package



Thank You