Data bases 2

TELCO SERVICE APPLICATIONS

Massimo Valle (10620441)

Specifications

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

Specifications

CONSUMER APPLICATION (Part 1)

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.

Specifications

CONSUMER APPLICATION (Part 2)

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.

Specification interpretation

CONSUMER APPLICATION

- Extra fee cost information is not used in the consumer or employee application, so I assume that this data are used for informational purposes only in the database.

Specifications

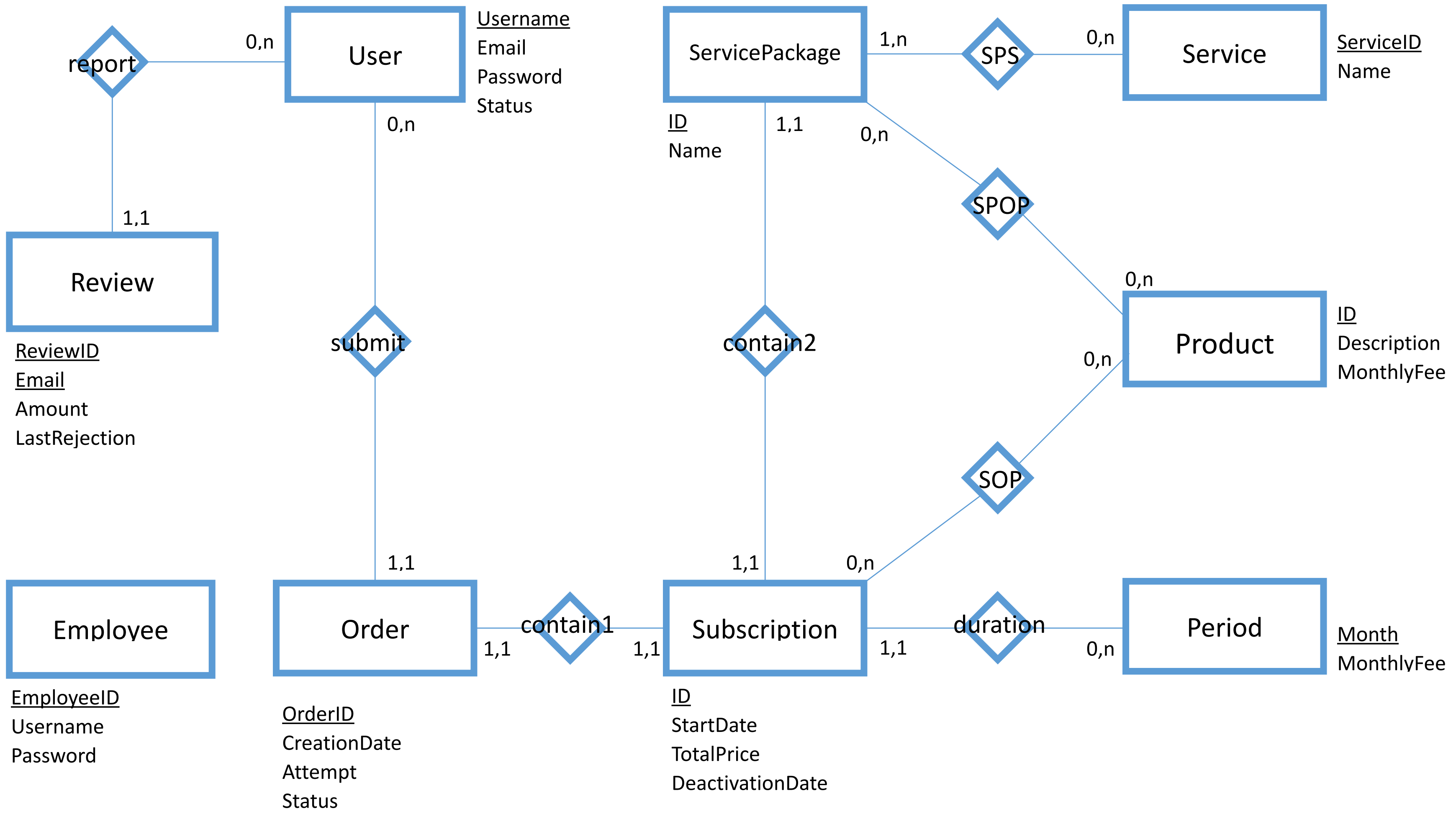
EMPLOYEE APPLICATION

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.

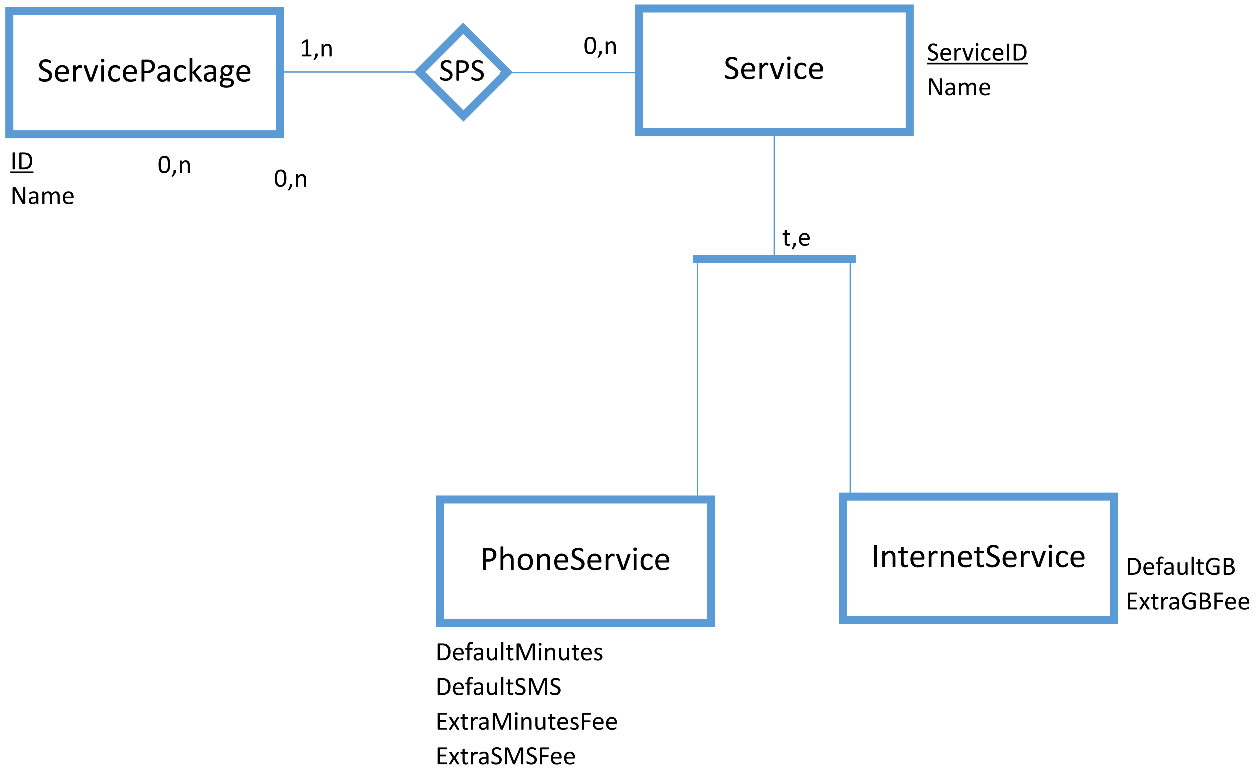
A Sales Report page allows the employee to inspect the essential data about the sales and about the users over the entire lifespan of the application:

- 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.

Entity Relationship - part 1



Entity Relationship - part 2



Motivations of the ER design

- I decided to set Order as strong entity because if a user associated with the order is removed, this one must remain valid.

Relational model - part 1

User(<u>Username</u>, Email, Password, Status)

Order(<u>OrderID</u>, UserID, SubscriptionID, CreationDate, Attempt, Status)

Subscription(<u>ID</u>, ServicePackageID, PeriodID, TotalPrice, StartDate, DeactivationDate)

ServicePackage(<u>ID</u>, Name)

Period(<u>Month</u>, MonthlyFee)

Review(<u>ReviewID</u>, UserID, Email, Amount, LastRejection)

Employee(<u>EmployeeID</u>, Username, Password)

Relational model - part 2

Subscription(ID, ServicePackageID, PeriodID, TotalPrice, StartDate, DeactivationDate)

ServicePackage(ID, Name)

SPS(ServicePackageID, ServiceID)

Service(ServiceID, Name)

SPOP(ServicePackageID, OptionalProductID)

SOP(SubscriptionID, OptionalProductID)

Product(ID, Description, MonthlyFee)

Trigger design & code

TRIGGER NAME: **TRG_TOTAL_COST_BASE**

- EVENT: New subscription added
- CONDITION: None
- ACTION: Compute base cost of subscription

```
DELIMITER //
CREATE TRIGGER TRG_TOTAL_COST_BASE
BEFORE INSERT ON db_Telco_DB2.Subscription
FOR EACH ROW
BEGIN
		SET NEW.TotalPrice = NEW.PeriodID * (SELECT MonthlyFee FROM db_Telco_DB2.Period WHERE Period.Month = NEW.PeriodID);
END;//
DELIMITER ;
```

- Trigger design motivation: trigger used to compute only the cost of the service package for the selected period

Trigger design & code

TRIGGER NAME: **TRG_TOTAL_COST_WITH_PRODUCTS**

- EVENT: New subscription with optional products added
- CONDITION: None
- ACTION: Compute incremental cost of products to add

```
DELIMITER //
CREATE TRIGGER TRG_TOTAL_COST_WITH_PRODUCTS
AFTER INSERT ON db_Telco_DB2.SOP
FOR EACH ROW
BEGIN
			UPDATE db_Telco_DB2.Subscription
			SET Subscription.TotalPrice = Subscription.TotalPrice +
			Subscription.PeriodID * (SELECT COALESCE(SUM(MonthlyFee),0)
						FROM db_Telco_DB2.Product AS P
						WHERE new.OptionalProductID = P.ID AND new.SubscriptionID = Subscription.ID)
			WHERE Subscription.ID = new.SubscriptionID;
END;//
DELIMITER ;
```

- Trigger design motivation: trigger used to compute only the cost of the optional products for the selected period to add

Trigger design & code

TRIGGER NAME: **TRG_TOTAL_SALES_REPORT_1**

- MATERIALIZED VIEW:

```
CREATE TABLE Sales_Report_1(
ServicePackage VARCHAR(45),
NumberTotalPurchases INT,
PRIMARY KEY(ServicePackage)
);
```

- TRIGGER:

```
DELIMITER //
CREATE TRIGGER TRG_TOTAL_SALES_REPORT_1
AFTER INSERT ON db_Telco_DB2.Order
FOR EACH ROW
BEGIN
            REPLACE INTO db_Telco_DB2.Sales_Report_1
                        SELECT ServicePackageID, COUNT(ServicePackageID)
                        FROM db_Telco_DB2.Subscription
                        GROUP BY ServicePackageID;
END;//
DELIMITER ;
```

Trigger design & code

TRIGGER NAME: **TRG_TOTAL_SALES_REPORT_2**

- MATERIALIZED VIEW:

```
CREATE TABLE Sales_Report_2(
ServicePackage VARCHAR(45),
Period INT,
NumberTotalPurchasesPerPackageAndValidityPeriod INT,
PRIMARY KEY(ServicePackage, Period)
)
```

- TRIGGER:

```
DELIMITER //
CREATE TRIGGER TRG_TOTAL_SALES_REPORT_2
AFTER INSERT ON db_Telco_DB2.Order
FOR EACH ROW
BEGIN
            REPLACE INTO db_Telco_DB2.Sales_Report_2
                        SELECT DISTINCT ServicePackageID, PeriodID, COUNT(*)
                        FROM db_Telco_DB2.Subscription
                        GROUP BY ServicePackageID, PeriodID;
END;//
DELIMITER ;
```

Trigger design & code

TRIGGER NAME: **TRG_TOTAL_SALES_REPORT_3**

- MATERIALIZED VIEW:

```
CREATE TABLE Sales_Report_3(
ServicePackage VARCHAR(45),
TotalValueOfSales DECIMAL(10,2),
NTotalValueOfSalesWithoutOptionalProducts DECIMAL(10,2),
PRIMARY KEY(ServicePackage)
)
```

- TRIGGER:

```
DELIMITER //
CREATE TRIGGER TRG_TOTAL_SALES_REPORT_3
AFTER INSERT ON db_Telco_DB2.Order
FOR EACH ROW
BEGIN
  REPLACE INTO db_Telco_DB2.Sales_Report_3
                        SELECT DISTINCT ServicePackageID, SUM(TotalPrice),
                        SUM(TotalPrice)-PeriodID * (SELECT COALESCE(SUM(MonthlyFee),0)
                                                  FROM db_Telco_DB2.Product AS P JOIN db_Telco_DB2.SOP AS S
                                                  WHERE S.OptionalProductID = P.ID AND S.SubscriptionID = Subscription.ID
                                                  GROUP BY SubscriptionID)
                                    FROM db_Telco_DB2.Subscription
                                    GROUP BY ServicePackageID, PeriodID, ID;
END;//
DELIMITER ;
```

Trigger design & code

TRIGGER NAME: **TRG_TOTAL_SALES_REPORT_4**

- MATERIALIZED VIEW:

```
CREATE TABLE Sales_Report_4(
ServicePackage VARCHAR(45),
AverageNumberOfOptionalProducts DECIMAL(10,2),
PRIMARY KEY(ServicePackage))
```

- TRIGGER:

```
DELIMITER //
CREATE TRIGGER TRG_TOTAL_SALES_REPORT_4
AFTER INSERT ON db_Telco_DB2.Order
FOR EACH ROW
BEGIN
		REPLACE INTO db_Telco_DB2.Sales_Report_4
			SELECT ServicePackageID, AVG(Count)
			FROM (
				SELECT ServicePackageID, COUNT(*) AS Count
				FROM db_Telco_DB2.Subscription AS S JOIN db_Telco_DB2.SOP AS SOP
				WHERE S.ID = SOP.SubscriptionID
				GROUP BY ServicePackageID, ID
			) AS TMP
			GROUP BY ServicePackageID;
END;//
DELIMITER ;
```

Trigger design & code

TRIGGER NAME:
TRG_TOTAL_SALES_REPORT_5_InsolventUsers

- MATERIALIZED VIEW:

```
CREATE TABLE Sales_Report_5_InsolventUsers(
User VARCHAR(45),
PRIMARY KEY(User))
```

- TRIGGER:

```
DELIMITER //
CREATE TRIGGER TRG_TOTAL_SALES_REPORT_5_InsolventUsers
AFTER INSERT ON db_Telco_DB2.Order
FOR EACH ROW
BEGIN
            REPLACE INTO db_Telco_DB2.Sales_Report_5_InsolventUsers
                        SELECT DISTINCT Username
                        FROM db_Telco_DB2.User
                        WHERE Status = "insolvent";
END;//
DELIMITER ;
```

Trigger design & code

TRIGGER NAME:
TRG_TOTAL_SALES_REPORT_5_SuspendedOrders

- MATERIALIZED VIEW:

```
CREATE TABLE Sales_Report_5_SuspendedOrders(
OrderID INT,
PRIMARY KEY(OrderID))
```

- TRIGGER:

```
DELIMITER //
CREATE TRIGGER TRG_TOTAL_SALES_REPORT_5_SuspendedOrders
AFTER INSERT ON db_Telco_DB2.Order
FOR EACH ROW
BEGIN
		REPLACE INTO db_Telco_DB2.Sales_Report_5_SuspendedOrders
				SELECT DISTINCT OrderID
				FROM db_Telco_DB2.Order
				WHERE Status = "rejected";
END;//
DELIMITER ;
```

Trigger design & code

TRIGGER NAME: **TRG_TOTAL_SALES_REPORT_5_Alerts**

- MATERIALIZED VIEW:

```
CREATE TABLE Sales_Report_5_Alerts(
ReviewID INT,
PRIMARY KEY(ReviewID))
```

- TRIGGER:

```
DELIMITER //
CREATE TRIGGER TRG_TOTAL_SALES_REPORT_5_Alerts
AFTER INSERT ON db_Telco_DB2.Review
FOR EACH ROW
BEGIN
            REPLACE INTO db_Telco_DB2.Sales_Report_5_Alerts
                        SELECT ReviewID
                        FROM db_Telco_DB2.Review;
END;//
DELIMITER ;
```

Trigger design & code

TRIGGER NAME: **TRG_TOTAL_SALES_REPORT_6**

- MATERIALIZED VIEW:

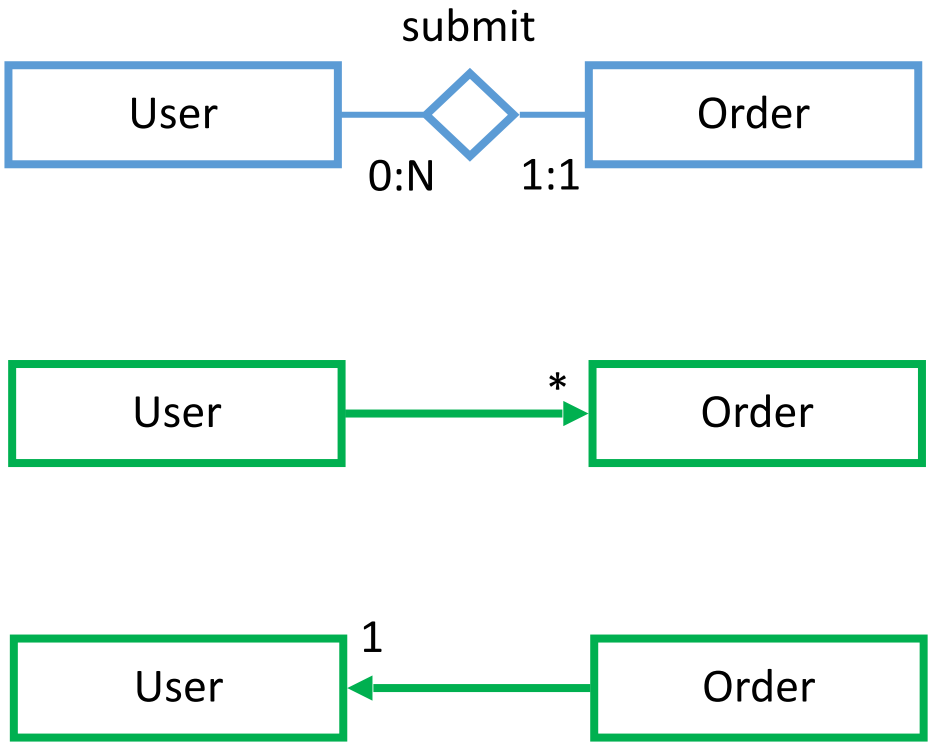
```
CREATE TABLE Sales_Report_6(
ProductID INT,
PRIMARY KEY(ProductID))
```

- TRIGGER:

```
DELIMITER //
CREATE TRIGGER TRG_TOTAL_SALES_REPORT_6
AFTER INSERT ON db_Telco_DB2.Order
FOR EACH ROW
BEGIN
                    REPLACE INTO db_Telco_DB2.Sales_Report_6
                                        SELECT ID FROM
                                        (
                                         SELECT ID, MonthlyFee*Count AS Sold
                                         FROM Product AS P JOIN
                                                            (
                                                            SELECT DISTINCT OptionalProductID, COUNT(OptionalProductID) AS Count
                                                            FROM db_Telco_DB2.SOP AS S JOIN db_Telco_DB2.Product AS P
                                                            GROUP BY OptionalProductID
                                                            ) AS TMP1
                                         WHERE P.ID = TMP1.OptionalProductID
                                         GROUP BY Sold, ID
                                        ) AS TMP2
                                        ORDER BY Sold DESC
                                        LIMIT 1;
END;//
DELIMITER ;
```

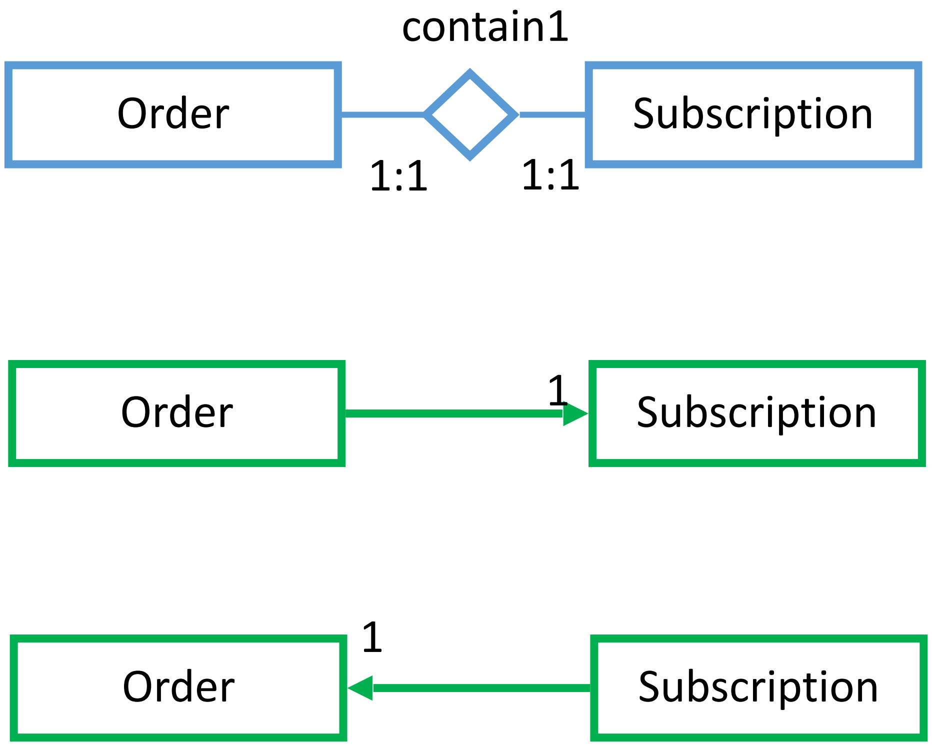
ORM design

Relationship "submit"



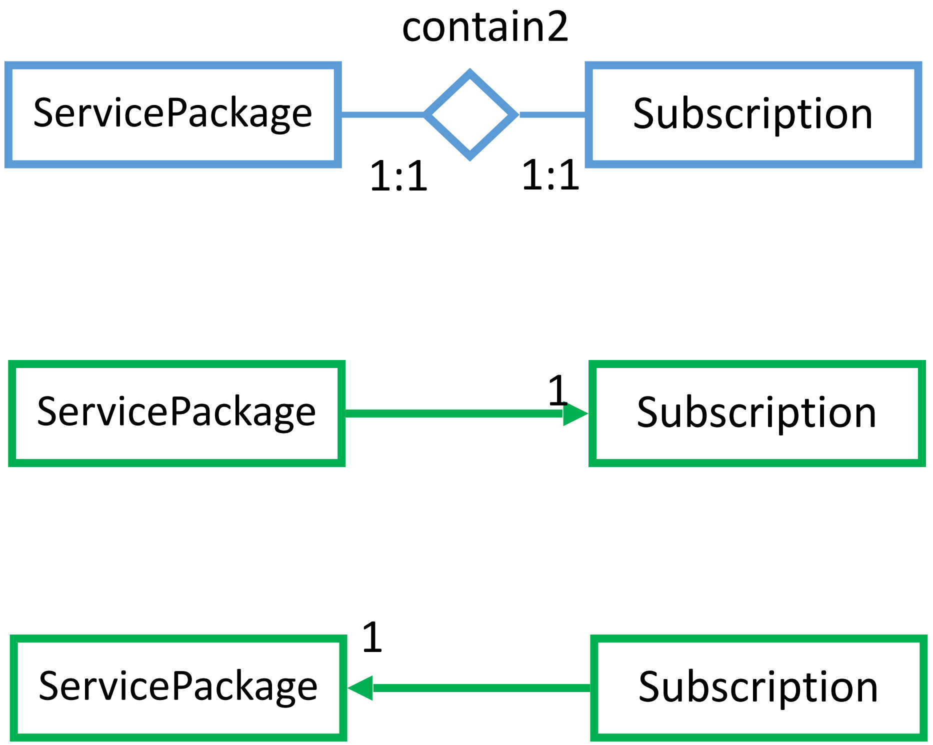
- User → Order
  - not mapped because it is not necessary to get the list of user's submits from the user entity
- Order →User
  - @ManyToOne: used to get the user that submit that order

Relationship “contain1 ”



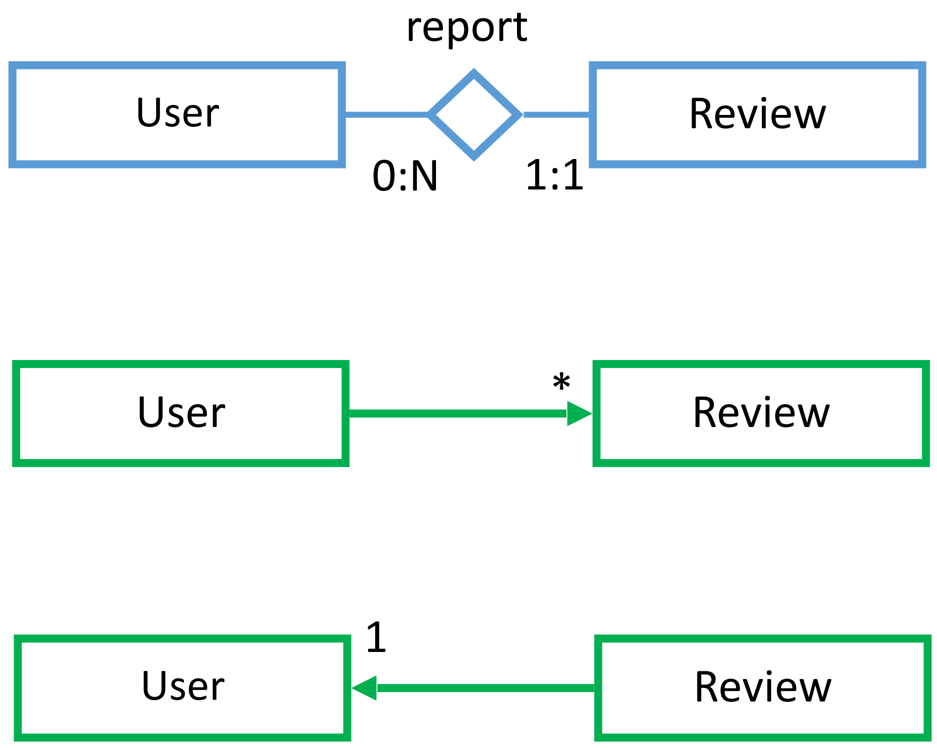
- Order → Subscription
  - not mapped because it is not necessary to get the order from the Subscription entity
- Subscription →Order
  - not mapped because it is not necessary to get the subscription from the Order entity

Relationship “contain2 ”



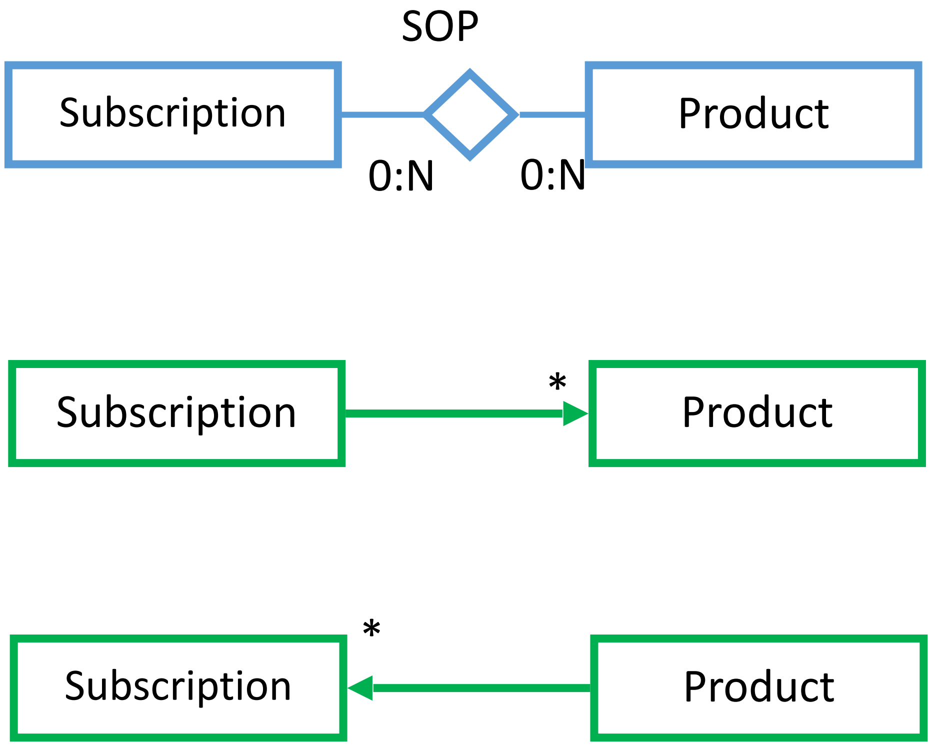
- ServicePackage → Subscription
  - not mapped because it is not necessary to get the service package from the Subscription entity
- Subscription →ServicePackage
  - not mapped because it is not necessary to get the subscription from the ServicePackage entity

Relationship “report ”



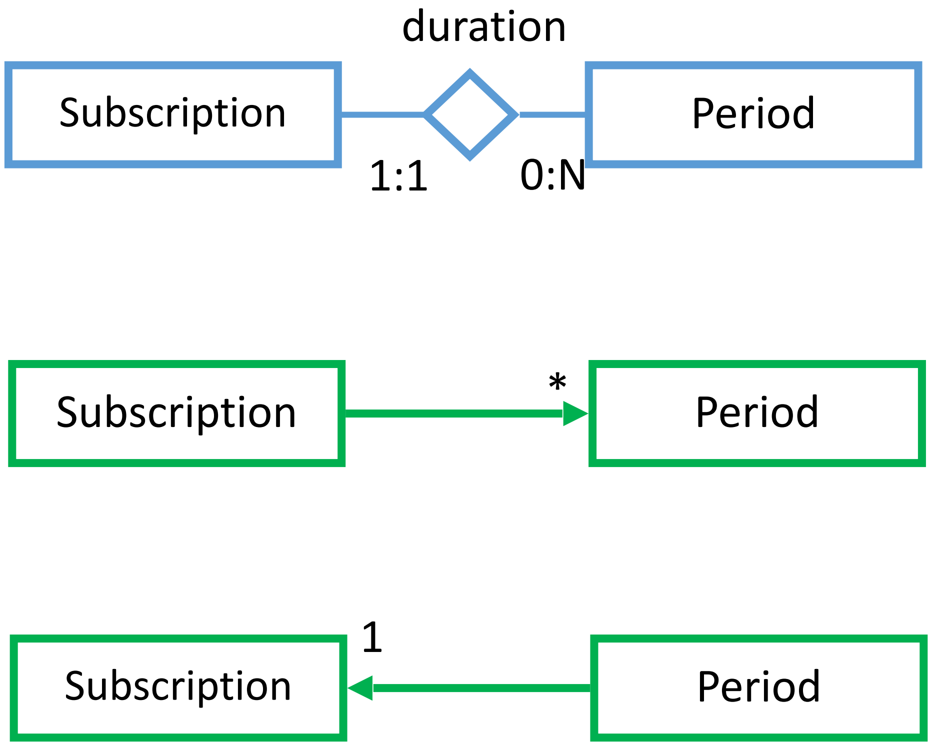
- User → Review
  - not mapped because it is not necessary to get the list of user’s reviews from the user entity
- Review →User
  - @ManyToOne: used to get the user reported in the review

Relationship “SOP ”



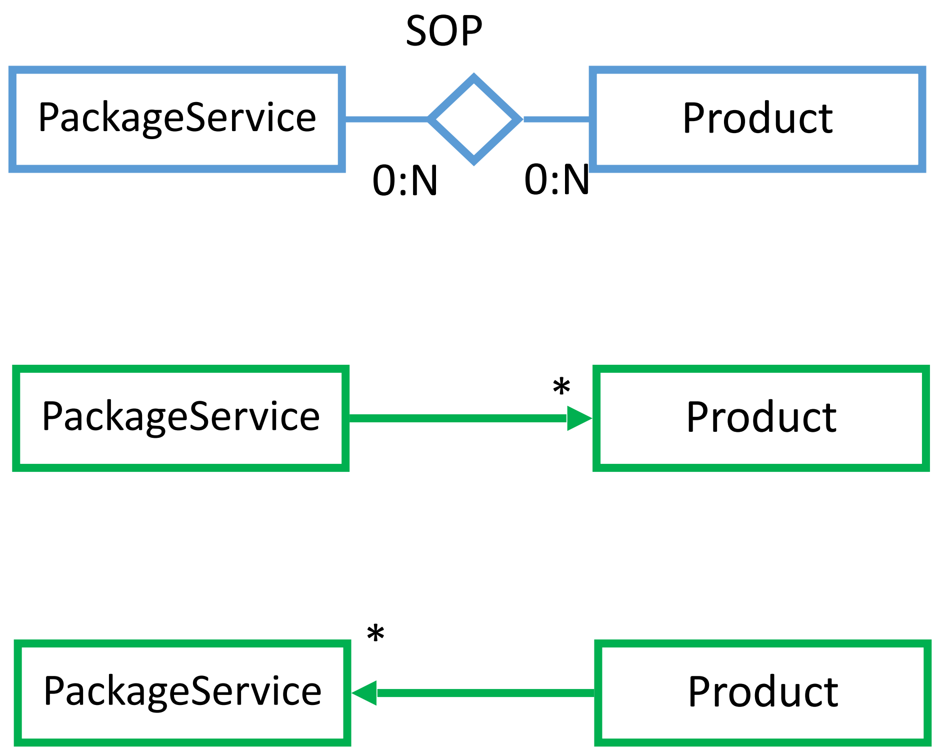
- Subscription → Product
  - @ManyToMany: used to get the optional products that are contained in a subscription
- Product →Subscription
  - @ManyToMany: used to get the subscriptions that contains the optional product

Relationship “duration ”



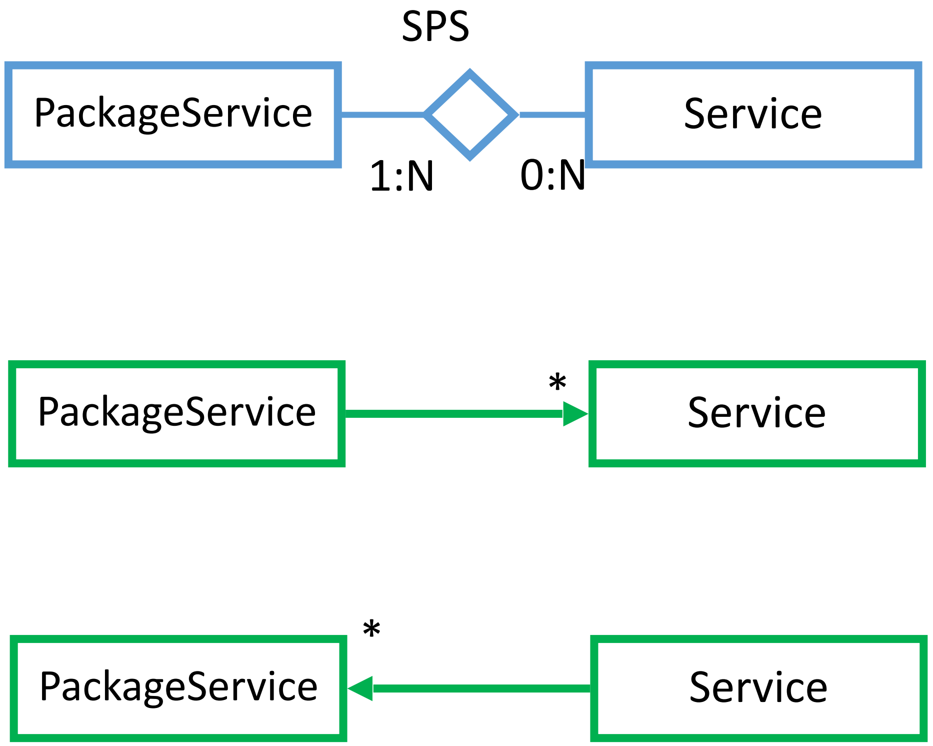
- Subscription → Period
  - not mapped because it is not necessary to get the subscription from the period entity
- Period →Subscription
  - not mapped because it is not necessary to get the list of periods from the subscription entity

Relationship “SPOP ”



- PackageService → Product
  - @ManyToMany: used to get the possible products that a package could contains
- Product →PackageService
  - @ManyToMany: used to get the packages that contains the optional product

Relationship “SPS ”



- PackageService → Service
  - @ManyToMany: used to get the possible services that a package could contains
- Service →PackageService
  - @ManyToMany: used to get the packages that contains the service

Entity Employee

```
@NamedQueries({
        @NamedQuery(name = "Employee.checkCredentials",
                query = "SELECT e FROM Employee e WHERE e.username = :usr AND
e.password = :pwd")
})

@Entity
public class Employee {
    @Id
    @Column(name = "EmployeeID", nullable = false)
    private Integer id;

    @Column(name = "Username", nullable = false, length = 45)
    private String username;

    @Column(name = "Password", nullable = false, length = 45)
    private String password;
}
```

Entity Order

```java
@Table(name = "`Order`", indexes = {
        @Index(name = "FK_Order_SUbscription_idx", columnList = "SubscriptionID"),
        @Index(name = "FK_Order_User_idx", columnList = "UserID")
})

@NamedQueries({
        @NamedQuery(name = "Order.getRejectedOrders",
                query = "SELECT o FROM Order o WHERE o.userID = :usr AND o.status = 'rejected'"),
        @NamedQuery(name = "Order.getOrder",
                query = "SELECT o FROM Order o WHERE o.userID = :usr AND o.subscriptionID = :sbcr")
})
@Entity
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "OrderID", nullable = false)
    private Integer id;

    @ManyToOne(optional = false)
    @JoinColumn(name = "UserID", nullable = false)
    private User userID;

    @ManyToOne(optional = false)
    @JoinColumn(name = "SubscriptionID", nullable = false)
    private Subscription subscriptionID;

    @Column(name = "CreationDate", nullable = false)
    private Timestamp creationDate;

    @Column(name = "Attempt", nullable = false)
    private Integer attempt;

    @Lob
    @Column(name = "Status", nullable = false)
    private String status;
}
```

Entity Period

```
NamedQueries({
        @NamedQuery(name = "Period.getAll",
                query = "SELECT p FROM Period p")
})

@Entity
public class Period {
    @Id
    @Column(name = "Month", nullable = false)
    private Integer id;

    @Column(name = "MonthlyFee", nullable = false, precision = 5, scale = 2)
    private BigDecimal monthlyFee;
}
```

Entity Product

```
@NamedQueries({
        @NamedQuery(name = "Product.getAll",
                query = "SELECT p FROM Product p")
})

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID", nullable = false)
    private Integer id;

    @Column(name = "Description", nullable = false, length = 200)
    private String description;

    @Column(name = "MonthlyFee", nullable = false, precision = 5, scale = 2)
    private BigDecimal monthlyFee;


    @ManyToMany(mappedBy = "possibleProductsToAdd")
    private Set<ServicePackage> packagesUseIt;

    @ManyToMany(mappedBy = "productChosen")
    private Set<Subscription> subscriptionsUseIt;
}
```

Entity Review

```
@NamedQueries({
        @NamedQuery(name = "Review.getByUser",
                query = "SELECT r FROM Review r WHERE r.userID = :usr")
})

@Entity
public class Review {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ReviewID", nullable = false)
    private Integer id;

    @ManyToOne(optional = false)
    @JoinColumn(name = "UserID", nullable = false)
    private User userID;

    @Column(name = "Email", nullable = false, length = 45)
    private String email;

    @Column(name = "Amount", precision = 10, scale = 2)
    private BigDecimal amount;

    @Column(name = "LastRejection", nullable = false)
    private Timestamp lastRejection;
}
```

Entity Service

```
@NamedQueries({
        @NamedQuery(name = "Service.getAll",
                query = "SELECT s FROM Service s")
})

@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Service {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ServiceID", nullable = false)
    private Integer id;

    @Lob
    @Column(name = "Name", nullable = false)
    private String name;

    @ManyToMany(mappedBy = "servicesInPackage")
    private Set<ServicePackage> packagesUseIt;
}
```

Entity InternetService

```
@Entity
public class InternetService extends Service{

    @Column(name = "DefaultGB")
    private Integer defaultGB;

    @Column(name = "ExtraGBFee", precision = 5, scale = 2)
    private BigDecimal extraGBFee;
}
```

Entity PhoneService

```
@Entity
public class PhoneService extends Service{

    @Column(name = "DefaultMinutes")
    private Integer defaultMinutes;

    @Column(name = "DefaultSMS")
    private Integer defaultSMS;

    @Column(name = "ExtraMinutesFee", precision = 5, scale = 2)
    private BigDecimal extraMinutesFee;

    @Column(name = "ExtraSMSFee", precision = 5, scale = 2)
    private BigDecimal extraSMSFee;
}
```

Entity ServicePackage

```java
@NamedQueries({
        @NamedQuery(name = "ServicePackage.getAll",
                query = "SELECT sp FROM ServicePackage sp")
})


@Entity
public class ServicePackage {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID", nullable = false)
    private Integer id;

    @Column(name = "Name", nullable = false, length = 45)
    private String name;


    @JoinTable(name = "SPOP", joinColumns = @JoinColumn(name = "ServicePackageID"),
            inverseJoinColumns = @JoinColumn(name = "OptionalProductID"))
    @ManyToMany
    private Set<Product> possibleProductsToAdd;


    @JoinTable(name = "SPS", joinColumns = @JoinColumn(name = "ServicePackageID"),
            inverseJoinColumns = @JoinColumn(name = "ServiceID"))
    @ManyToMany
    private Set<Service> servicesInPackage;
}
```

Entity Subscription

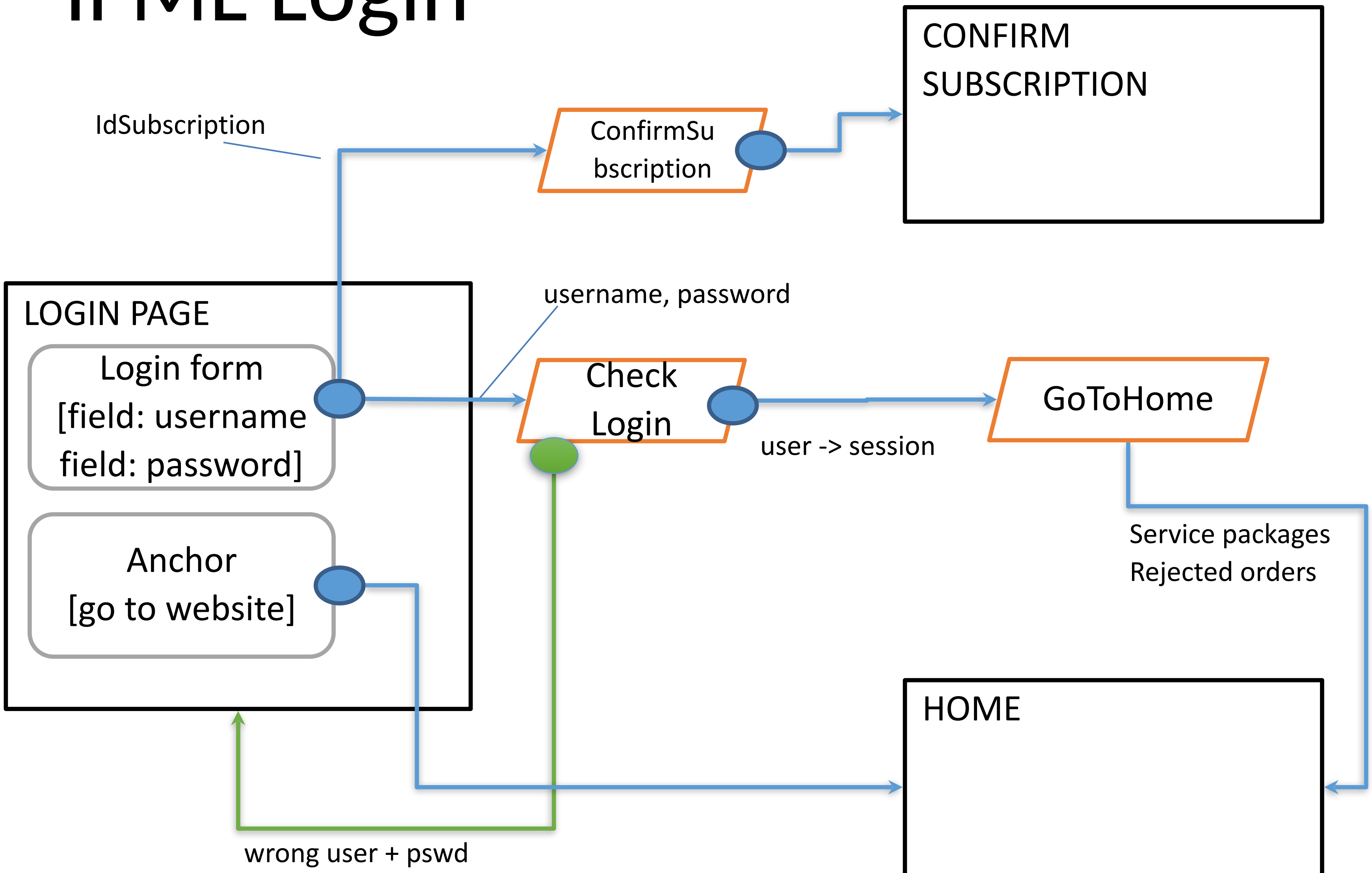
```
@Table(name = "Subscription", indexes = {
        @Index(name = "FK_Subscription_ServicePackage_idx", columnList = "ServicePackageID")
})

@Entity
public class Subscription {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID", nullable = false)
    private Integer id;

    @Column(name = "StartDate", nullable = false)
    private LocalDate startDate;

    @Column(name = "TotalPrice", nullable = false, precision = 10, scale = 2)
    private BigDecimal totalPrice;

    @ManyToOne(optional = false)
    @JoinColumn(name = "ServicePackageID", nullable = false)
    private ServicePackage servicePackageID;

    @ManyToOne(optional = false)
    @JoinColumn(name = "PeriodID", nullable = false)
    private Period periodID;

    @Column(name = "DeactivationDate")
    private LocalDate deactivationDate;


    @JoinTable(name = "SOP", joinColumns = @JoinColumn(name = "SubscriptionID"),
            inverseJoinColumns = @JoinColumn(name = "OptionalProductID"))
    @ManyToMany
    private Set<Product> productChosen;
}
```

Entity User

```
@NamedQueries({
        @NamedQuery(name = "User.checkCredentials",
                query = "SELECT u FROM User u WHERE u.username = :usr AND u.password = :pwd"),
        @NamedQuery(name = "User.getByUsername",
                query = "SELECT u FROM User u WHERE u.username = :usr")
})

@Entity
public class User {
    @Id
    @Column(name = "Username", nullable = false, length = 45)
    private String username;

    @Column(name = "Email", nullable = false, length = 45)
    private String email;

    @Column(name = "Password", nullable = false, length = 45)
    private String password;

    @Column(name = "Status", nullable = false, length = 45)
    private String status;

}
```

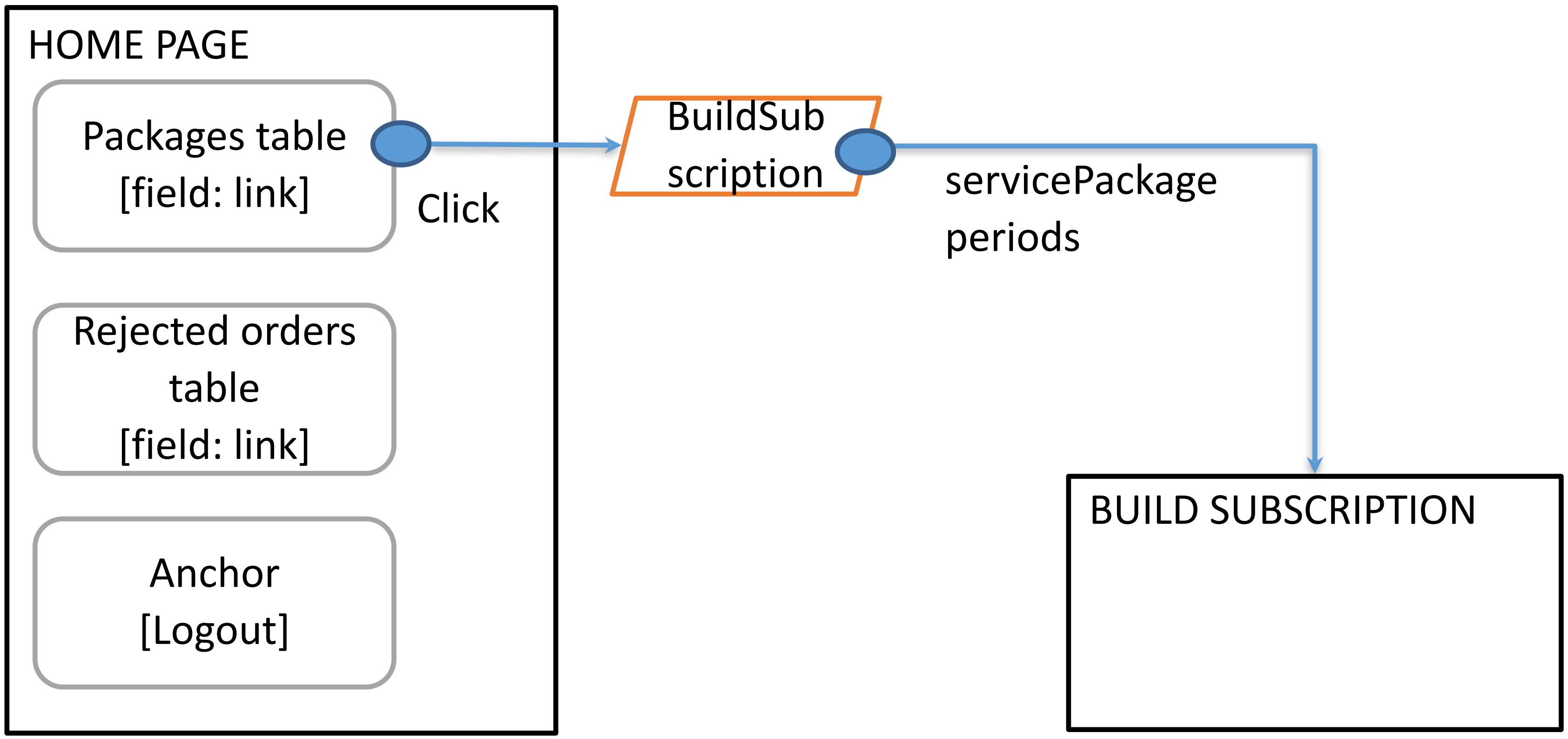
Functional analysis of the interaction

IFML Login

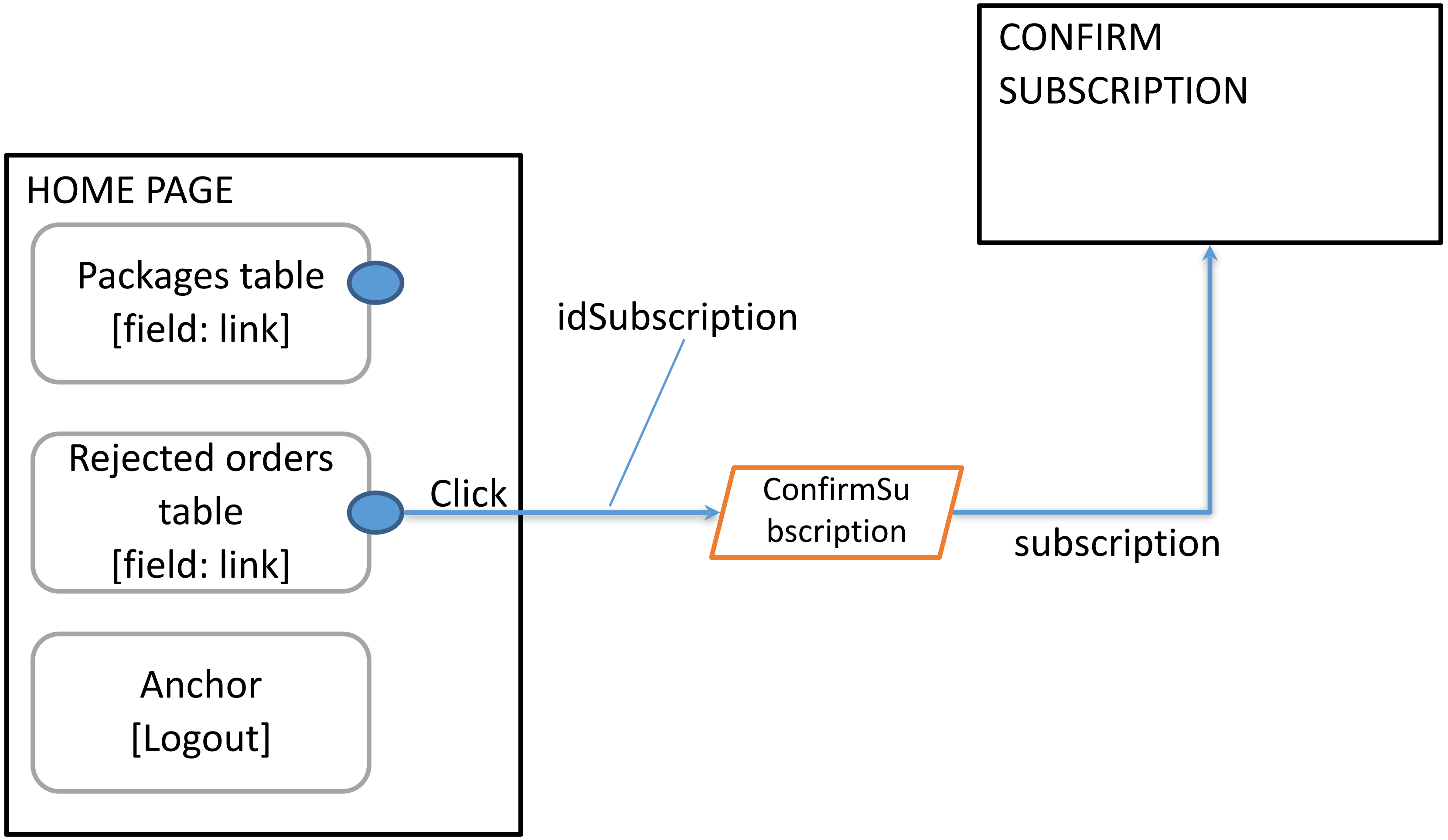


IdSubscription
ConfirmSubscription
CONFIRM SUBSCRIPTION
LOGIN PAGE
Login form [field: username field: password]
Anchor [go to website]
username, password
Check Login
GoToHome
user -> session
Service packages
Rejected orders
HOME
wrong user + pswd

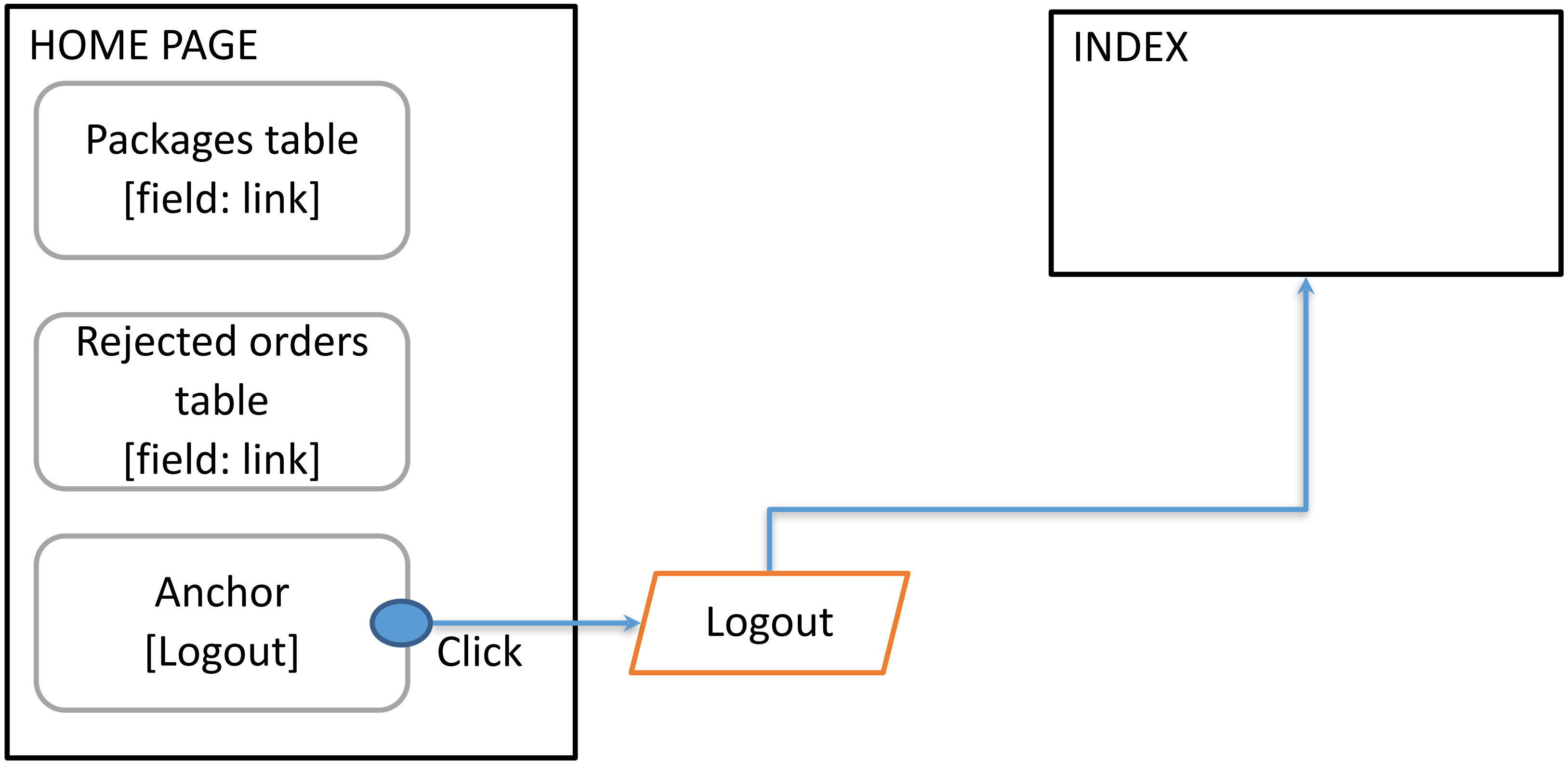
IFML Home page



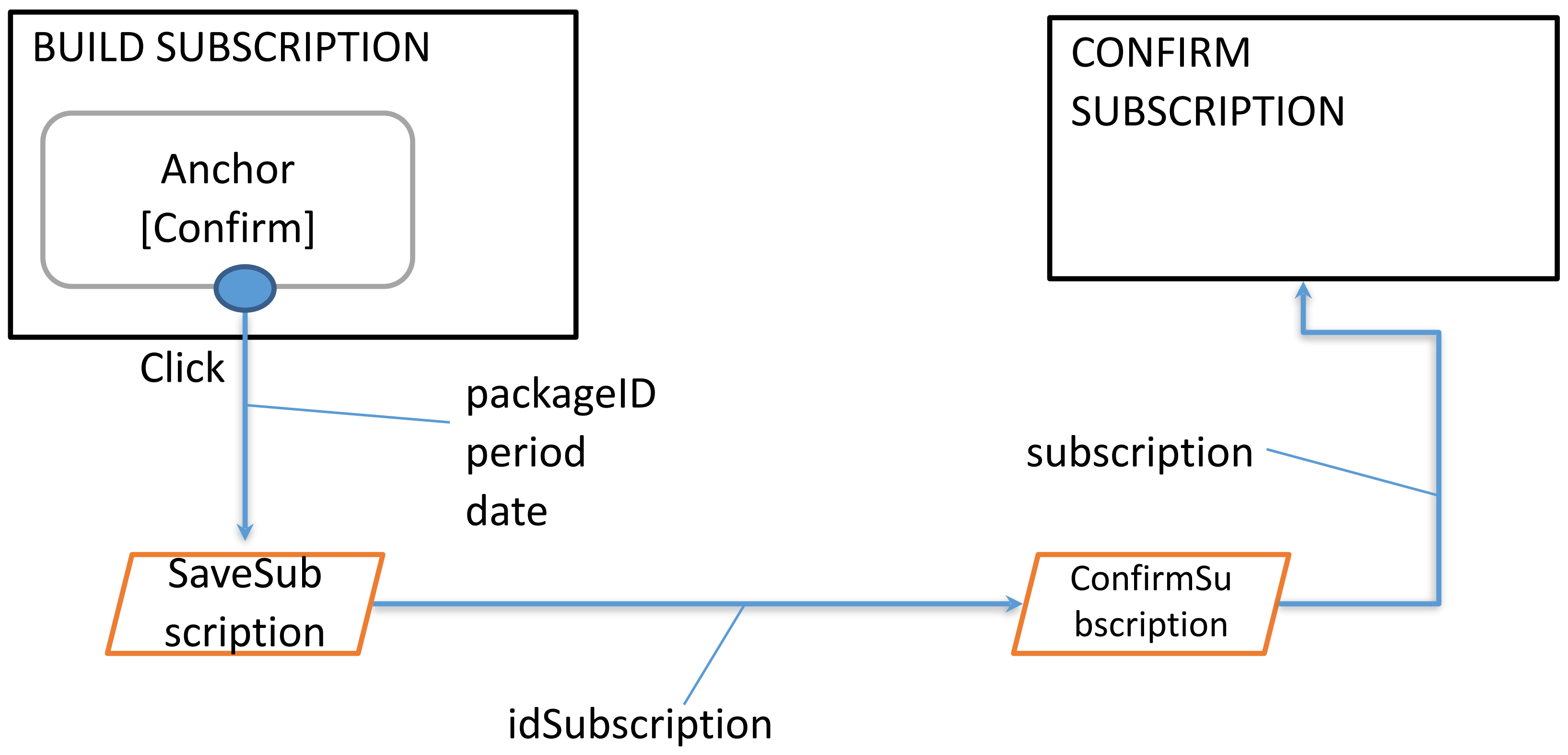
IFML Home page



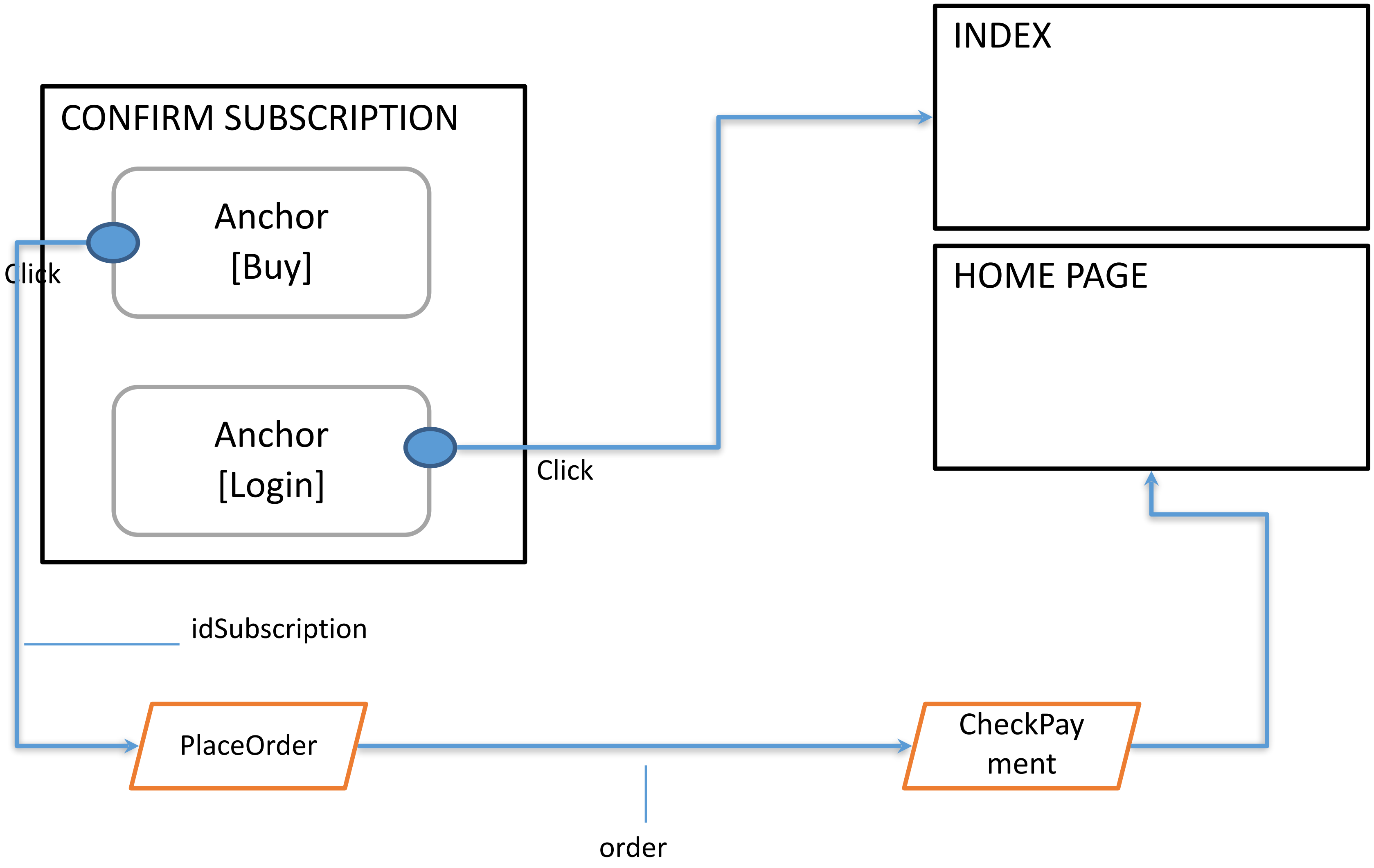
IFML Home page


HOME PAGE
Packages table [field: link]
Rejected orders table [field: link]
Anchor [Logout]
Click
Logout
INDEX

IFML Build Subscription



IFML Build Subscription



Textual notation

CONSUMER APPLICATION (Part 1)

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.

- Pages (views), view components, events, actions

Textual notation

CONSUMER APPLICATION (Part 2)

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.

- Pages (views), view components, events, actions

Textual notation

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.

- Pages (views), view components, events, actions

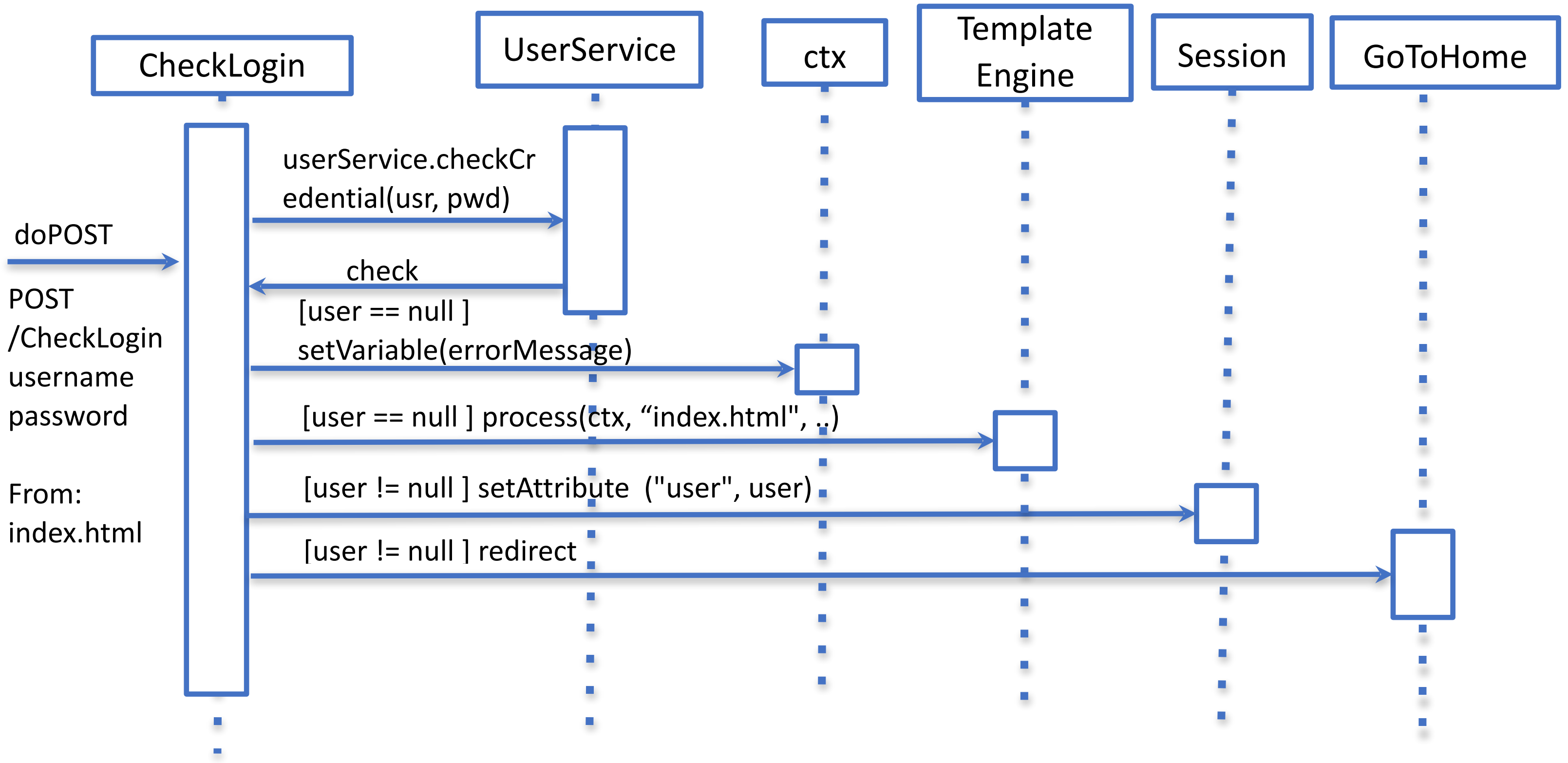
Consumer Components

- Client components
  - Servlets
    - CheckLogin
    - Register
    - Logout
    - GoToHome
    - BuildSubscription
    - SaveSubscription
    - ConfirmSubscription
    - PlaceOrder
    - CheckPayment
  - Views
    - index.html
    - home.html
    - buildSubscription.html
    - ConfirmationPage.html
- Back end components
  - Entities
    - User
    - ServicePackage
    - Service
    - Subscription
    - Product
    - Period
    - Order
    - Review
    - PhoneService
    - InternetService
  - Business Components (EJBs)
    - User_Service
    - ServicePackage_Service
    - Service_Service
    - Subscription_Service
    - Product_Service
    - Period_Service
    - Order_Service
    - Review_service

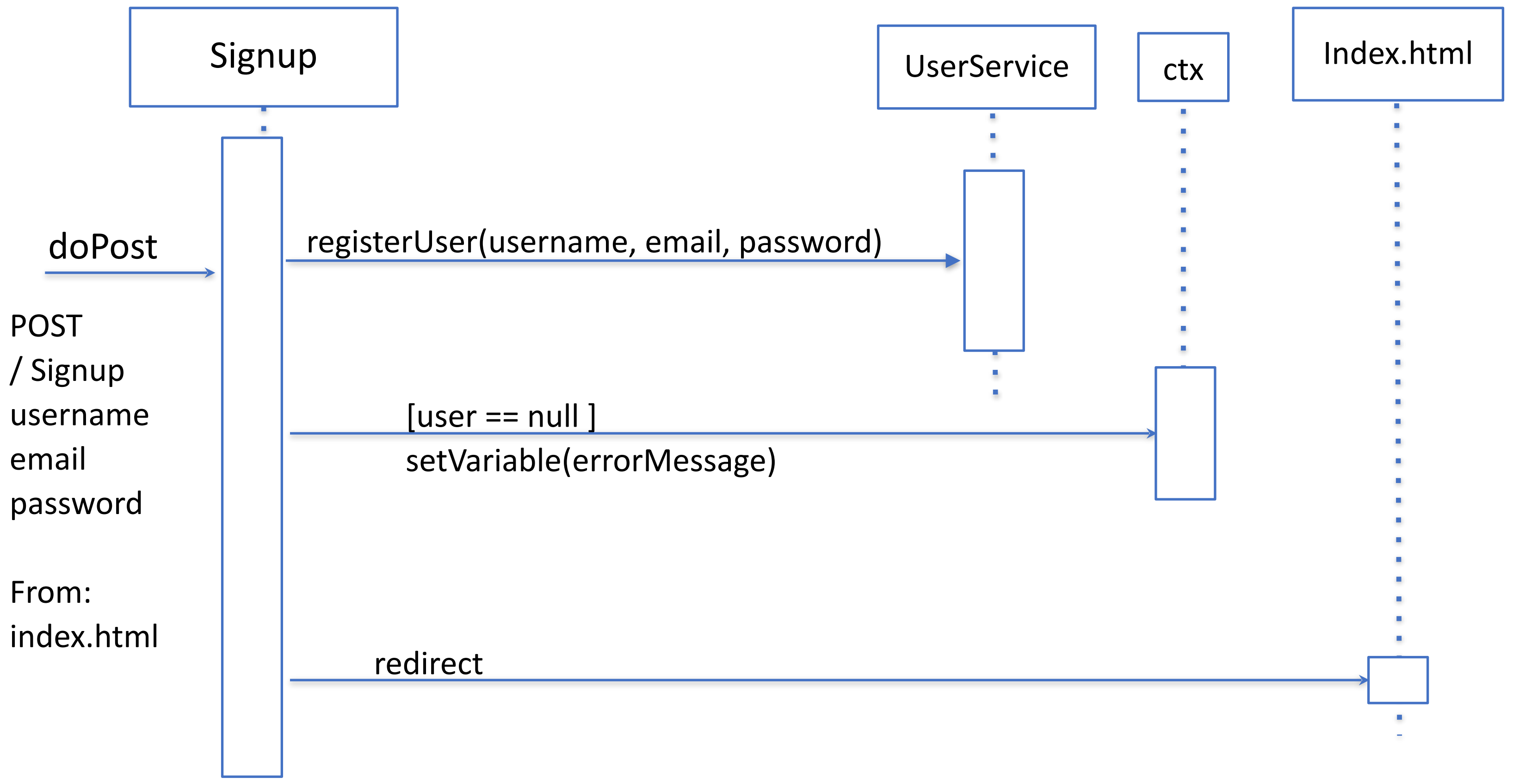
Employee Components

- Client components
  - Servlets
    - CheckLogin
    - Logout
    - GoToHome
    - CreatePackage
    - CreateProduct
    - GoToSalesReport
  - Views
    - index.html
    - home.html
    - salesReport.html
- Back end components
  - Entities
    - User
    - ServicePackage
    - Service
    - Product
    - Period
    - Review
  - Business Components (EJBs)
    - Employee_Service
    - ServicePackage_Service
    - Service_Service
    - Product_Service
    - Period_Service
    - Review_service

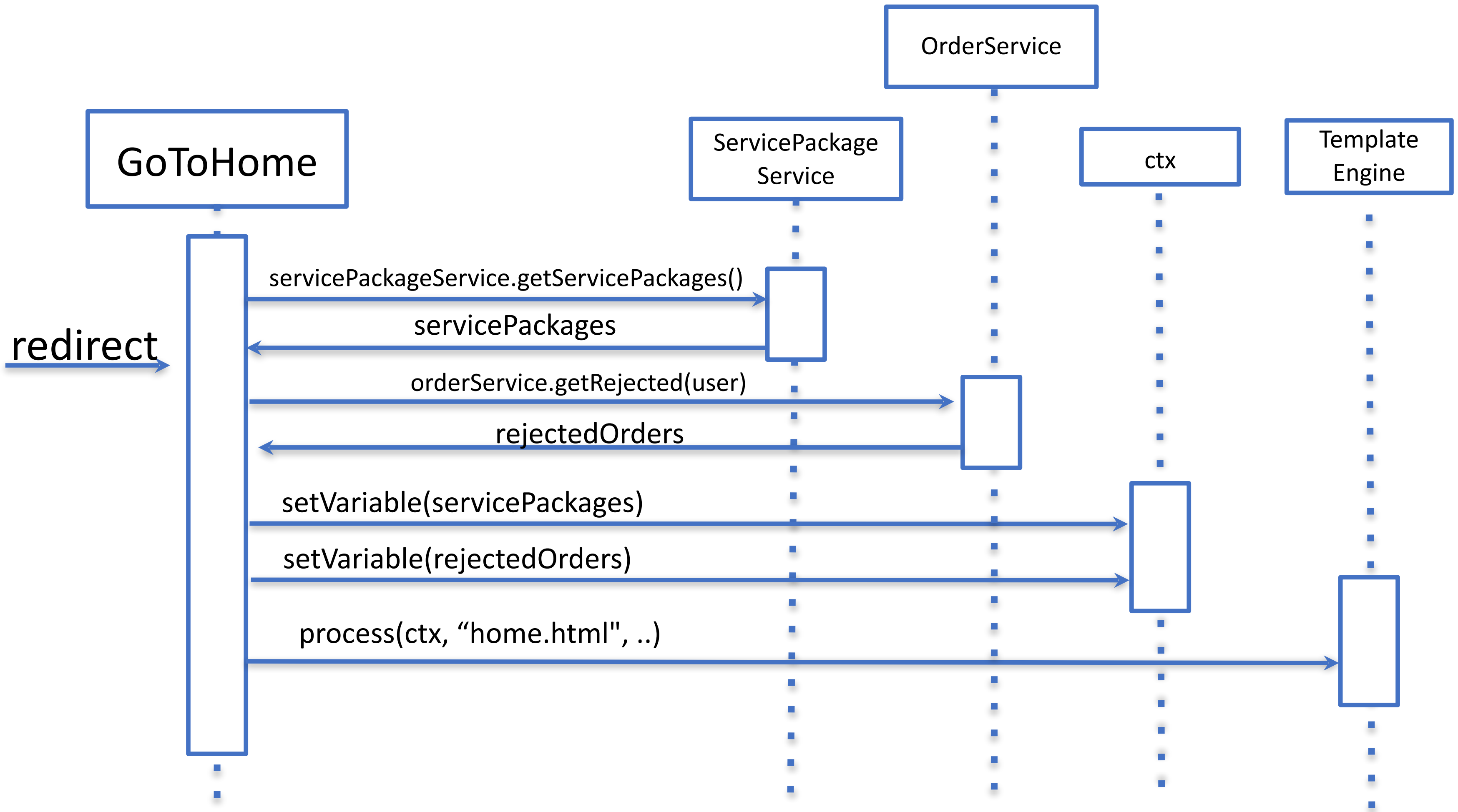
Event: Login


CheckLogin
UserService
ctx
Template Engine
Session
GoToHome
userService.checkCredential(usr, pwd)
doPOST
check
POST /CheckLogin username password
[user == null] setVariable(errorMessage)
[user == null] process(ctx, “index.html", ..)
From: index.html
[user != null] setAttribute ("user", user)
[user != null] redirect

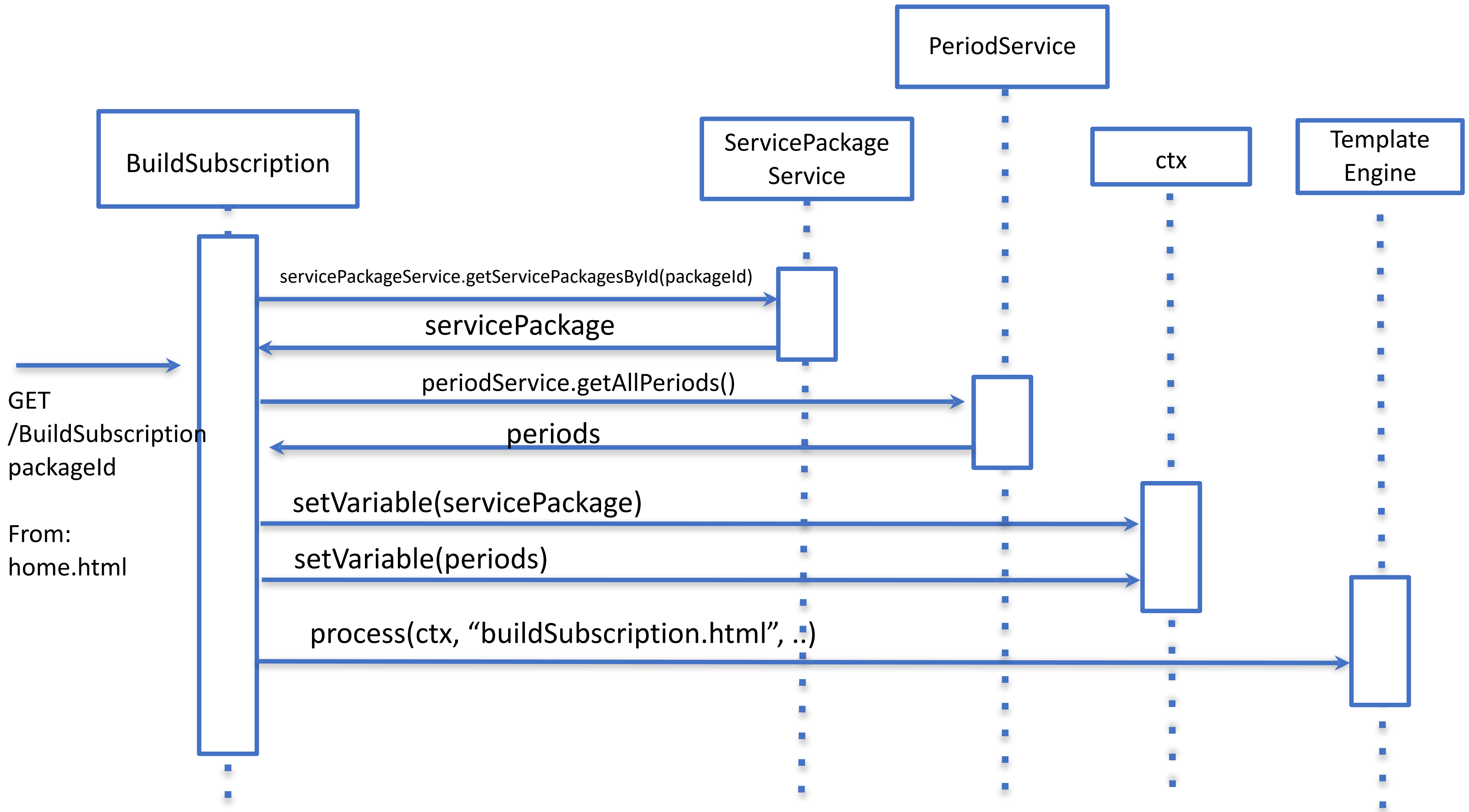
Event: Signup



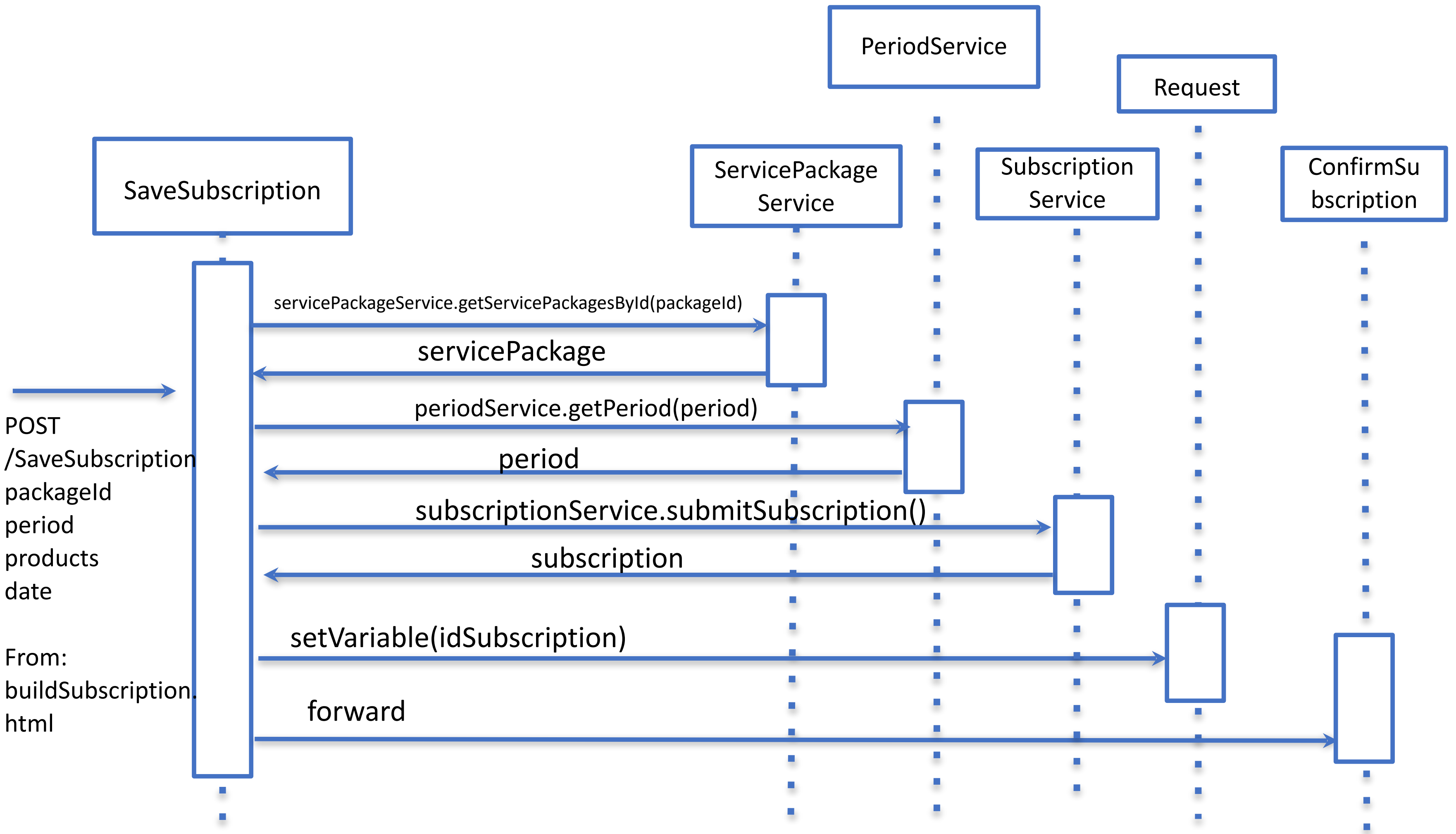
Event: GoToHome


OrderService
GoToHome
ServicePackage Service
ctx
Template Engine
servicePackageService.getServicePackages()
servicePackages
redirect
orderService.getRejected(user)
rejectedOrders
setVariable(servicePackages)
setVariable(rejectedOrders)
process(ctx, “home.html", ..)

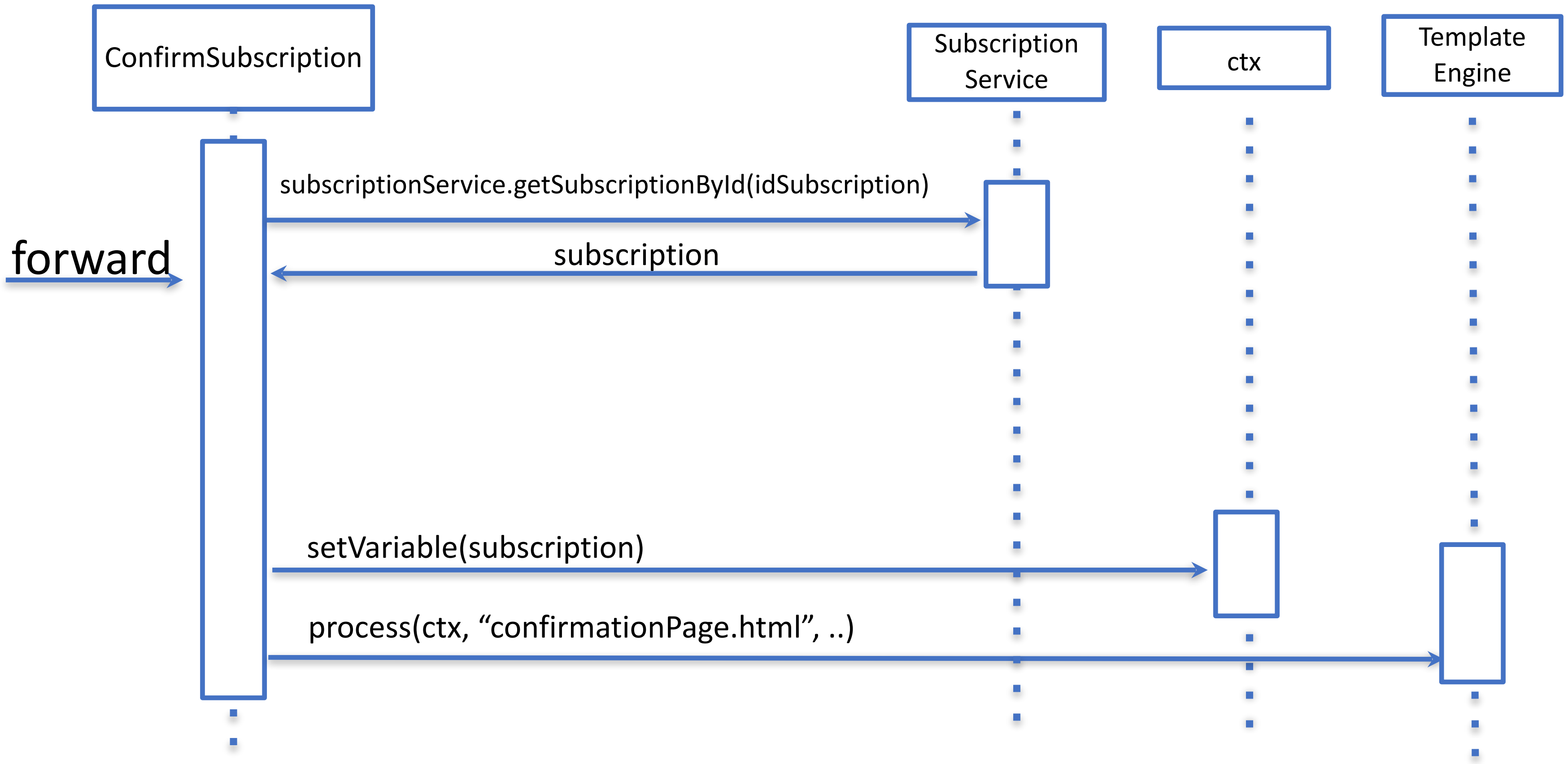
Event: BuildSubscription


PeriodService
BuildSubscription
ServicePackage Service
ctx
Template Engine
servicePackageService.getServicePackagesById(packageId)
servicePackage
periodService.getAllPeriods()
periods
GET /BuildSubscription packageId
From: home.html
setVariable(servicePackage)
setVariable(periods)
process(ctx, “buildSubscription.html”, ..)

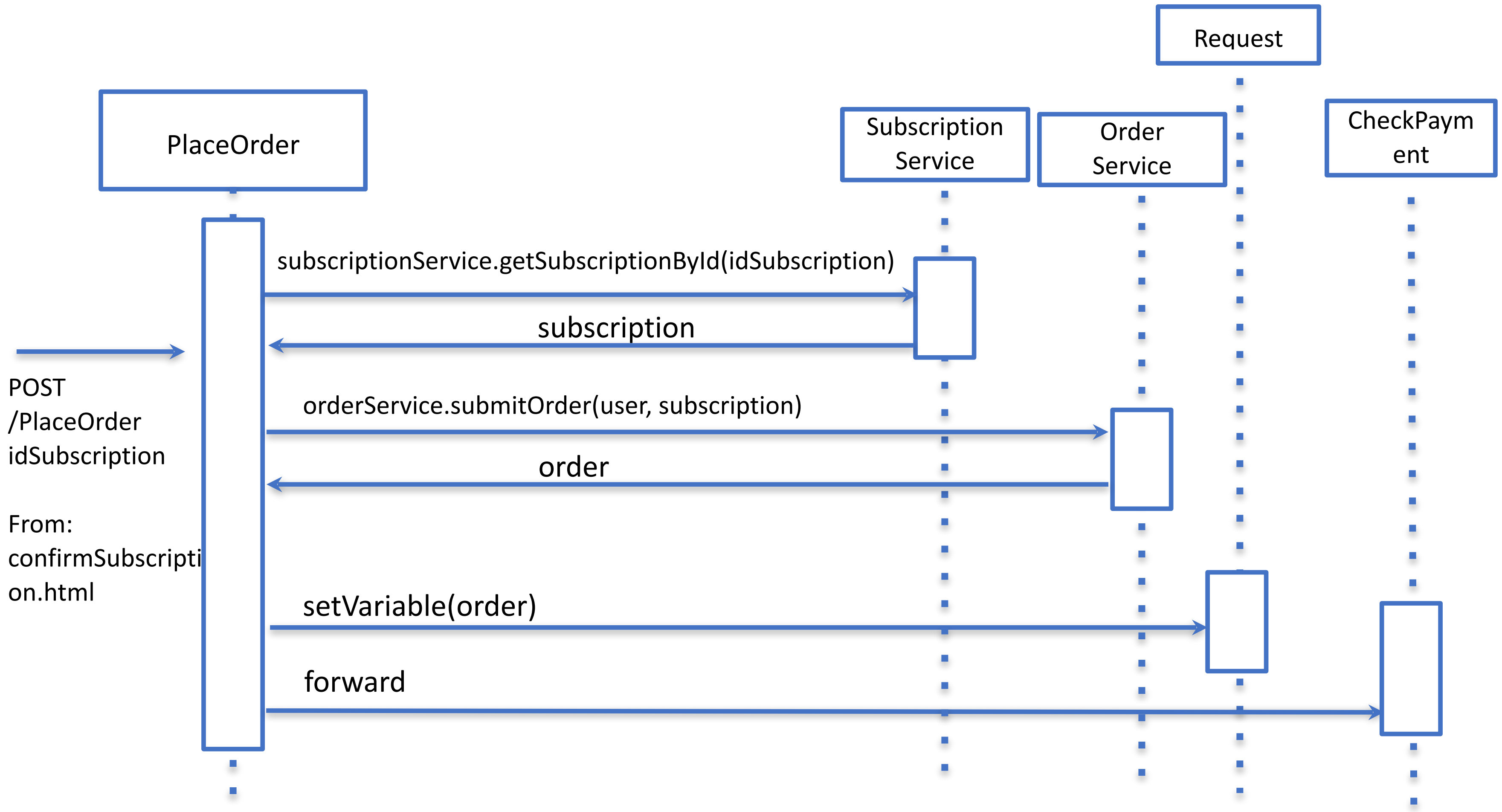
Event: SaveSubscription



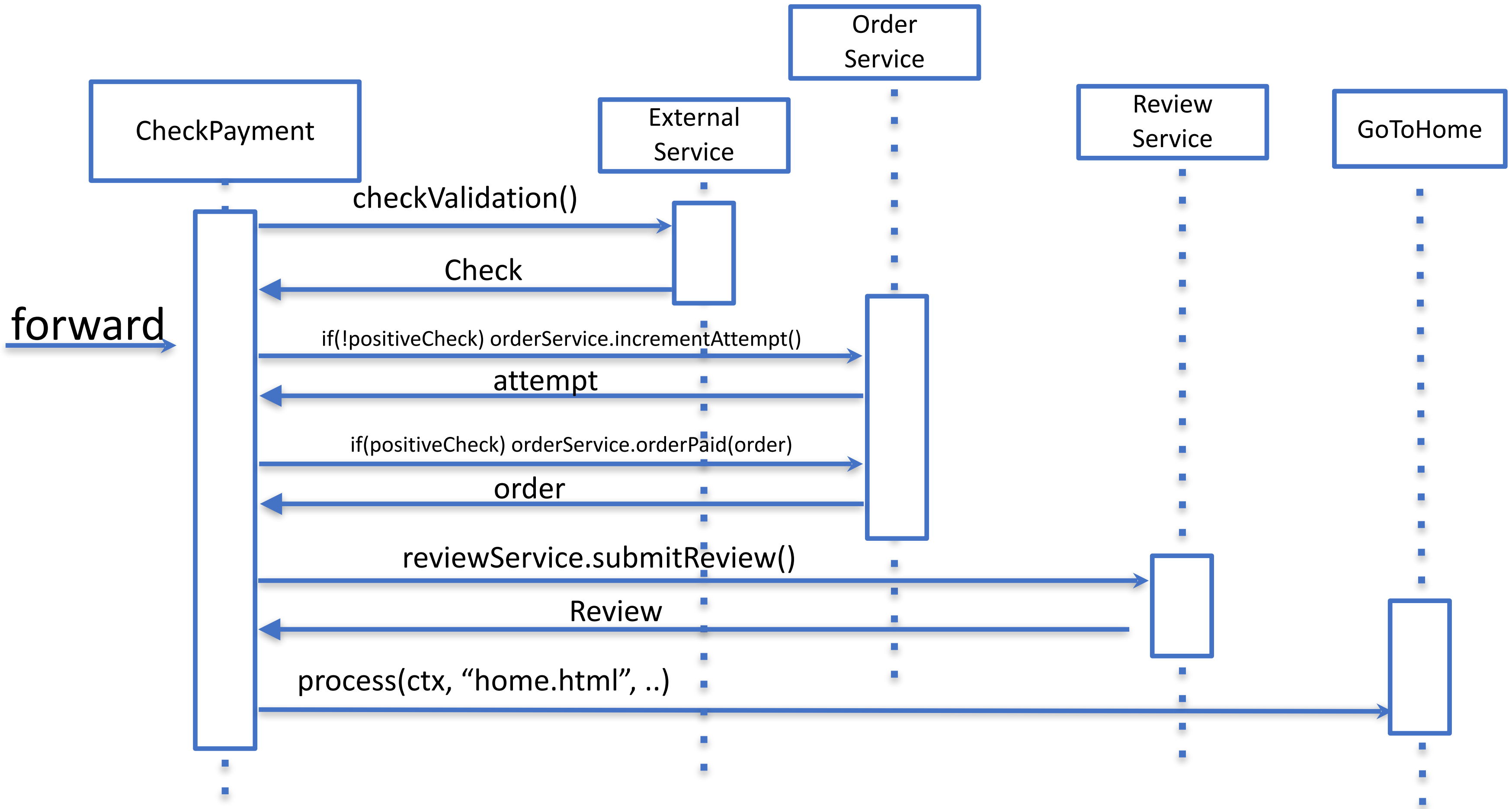
Event: ConfirmSubscription


ConfirmSubscription
Subscription Service
ctx
Template Engine
forward
subscriptionService.getSubscriptionById(idSubscription)
subscription
setVariable(subscription)
process(ctx, “confirmationPage.html”, ..)

Event: PlaceOrder

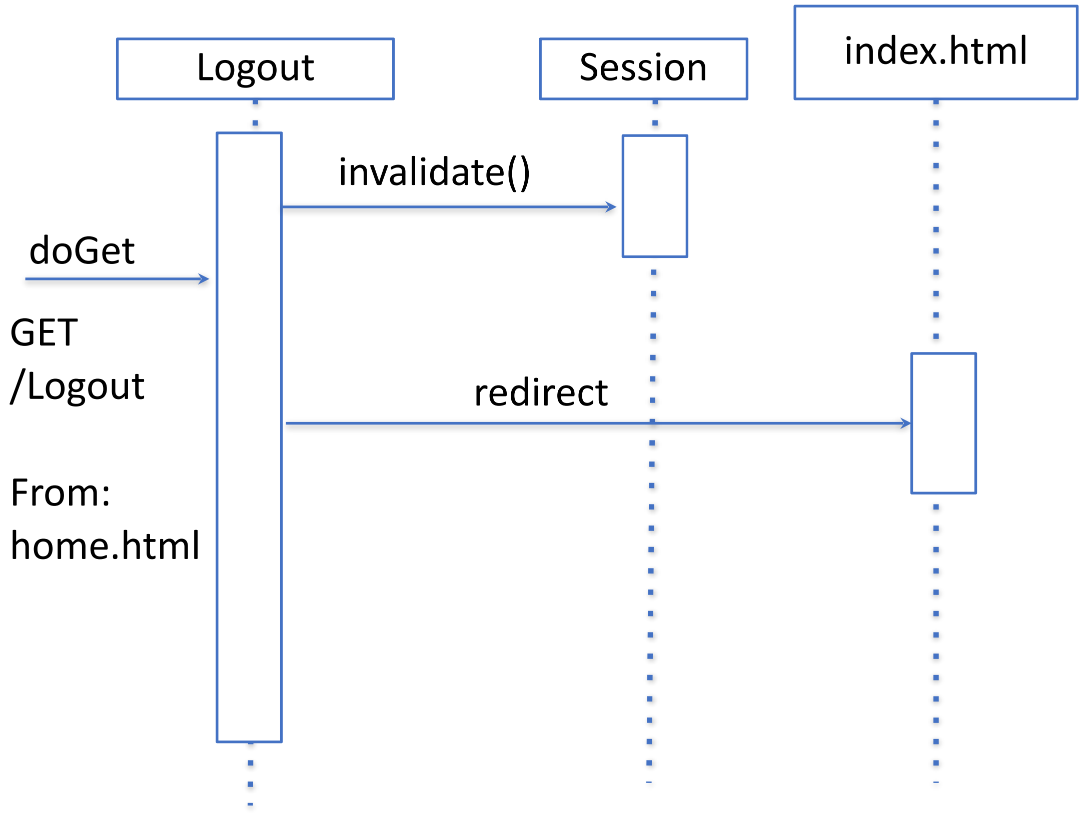

Request
PlaceOrder
Subscription Service
Order Service
CheckPayment
subscriptionService.getSubscriptionById(idSubscription)
subscription
POST /PlaceOrder idSubscription
From: confirmSubscription.html
orderService.submitOrder(user, subscription)
order
setVariable(order)
forward

Event: CheckPayment



CheckPayment
External Service
Order Service
Review Service
GoToHome
checkValidation()
Check
forward
if(!positiveCheck) orderService.incrementAttempt()
attempt
if(positiveCheck) orderService.orderPaid(order)
order
reviewService.submitReview()
Review
process(ctx, “home.html”, ..)

Event: logout


Logout
Session
index.html
invalidate()
doGet
GET
/Logout
From:
home.html
redirect