FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Fleet Management System

Design

Documentation of a project for the purpose of the course BIE-SI1.

Authors: Vladimir Ananyev
Berker Katipoglu
Oleksii Tarbeiev
Atsamaz Akopyan
Muntaser Abu Zaid
Sanan Pashayev

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Contents

1. Architecture

This chapter describes the architecture of the web application of the Fleet Management System.

The web application will be implemented as Java using following technologies:
• Java 1.8
• Spring Boot Framework
• Vaadin Framework

The architecture is divided into three independent layers:
• Presentation layer - layer responsible for presentation of application data.
• Business layer- layer responsible for all business logic of the application.
• Data layer - layer responsible for data persistence.

The layers are isolated using interfaces:
• IBusiness - interface between the presentation and business layers. It consists of one main IBusiness to handle any request from presentation layer.
• IDao - interface between the business and data layer. It consists of several IDao interfaces defining the persistence operations for individual entities in the system.

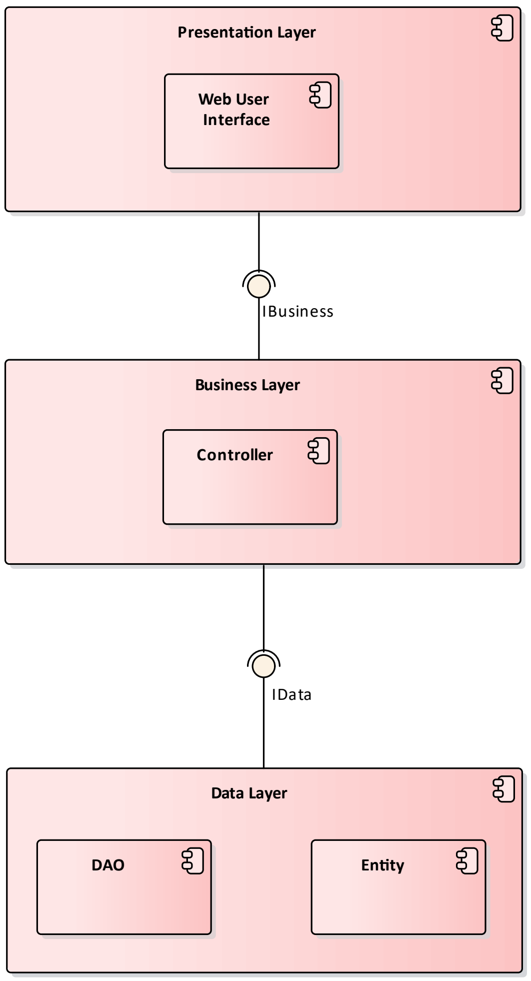


Figure 1 - Architecture

1.1 Presentation Layer

The presentation layer is responsible for sharing the information with a user or other systems. It consists of a web user interface and connected to business layer through business interface.

1.1.1 Web User Interface

Web User Interface is implemented in Java using Swing Boot and Vaadin frameworks.

1.2 Business Layer

The business layer contains implementation of the business logic and It consists of controllers implementing the system behavior. It is connected to data later through data interface.

1.2.1 Controller

Controller is implemented in Java using Spring Boot framework.

1.3 Data Layer

The data layer is responsible for data persistence. It consists of DAO classes implementing the persistence operations and entities representing the persistent data.

The implementation of the data layer is based on the Spring Boot Framework and its native support for database.

2. Design Class Model

This package describes the classes and packages of the implementation of the web application based on the Vaadin and Spring Boot Frameworks.

In the whole model, only important classes and their methods are shown. The primitive getters and setters are not shown.

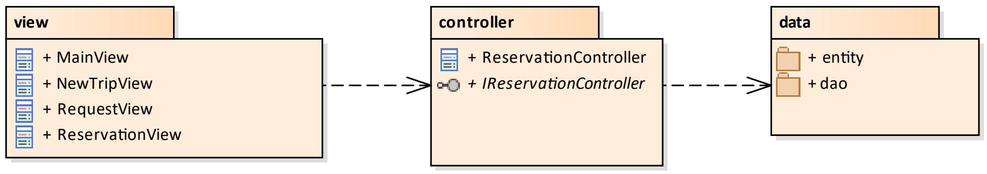


Figure 2 - Design Class Model

2.1 view

The web package contains classes implementing the web user interface.

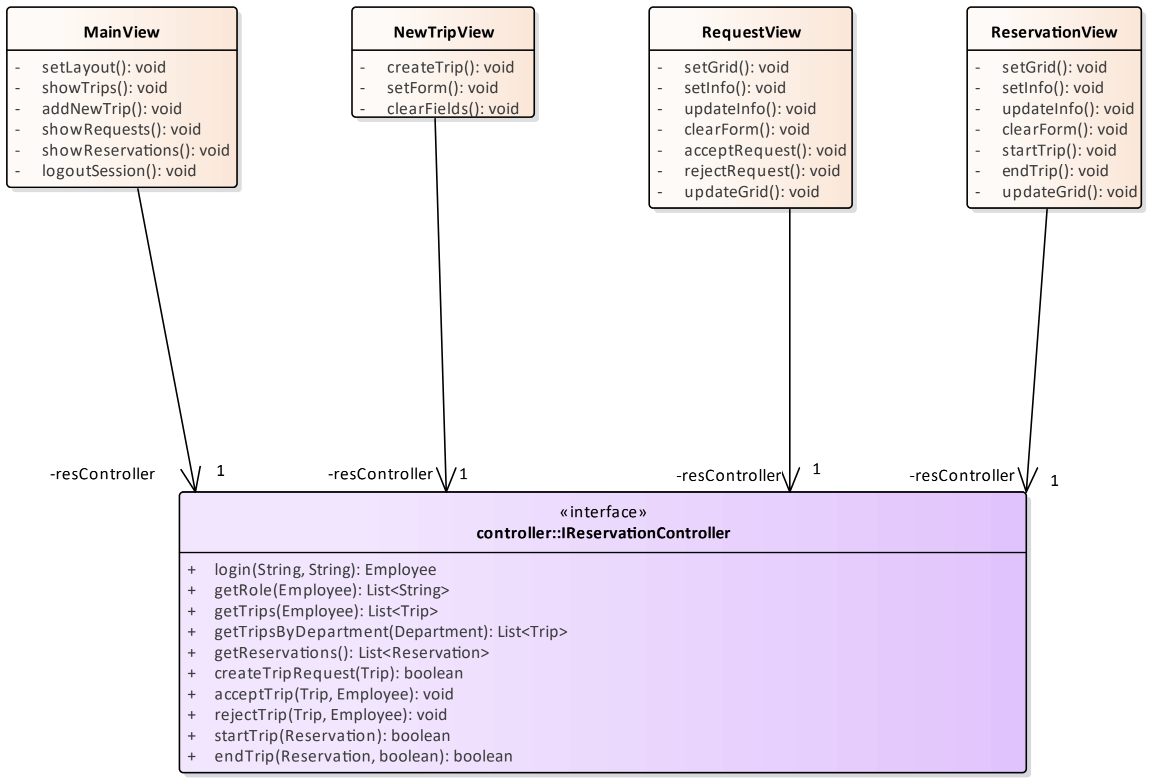


Figure 3 - view

2.1.1 NewTripView

This class is responsible for displaying necessary fields to make trip request and processing requests by passing arguments to control layer via reservation controller interface.

Method name	Return type	Description
createTrip	void	Button function to create new trip. Function uses IReservationController to realize request.
setForm	void	Function to set the fields in the form displayed on the screen.
clearFields	void	Function to clear the fields of the form displayed on the screen.

2.1.2 RequestView

This class is responsible for displaying all requests for a given department and processing confirmation requests by passing arguments to control layer via reservation controller interface.

Method name	Return type	Description
setGrid	void	Function to set the contents of the table showing trip requests.
setInfo	void	Function to set the contents of the form displaying the details of the trip request.
updateInfo	void	Function to update the contents of the form displaying the details of the trip request.
clearForm	void	Function to clear the contents of the form displaying the details of the trip request.
acceptRequest	void	Button function to accept trip requests. Request view uses IReservationController to realize the request.
rejectRequest	void	Button function to reject trip requests. Request view uses IReservationController to realize the request.
updateGrid	void	Function to update the table showing the requests.

2.1.3 ReservationView

This class is responsible for displaying all reservations and processing start, end and cancel requests by passing arguments to control layer via reservation controller interface.

Method name	Return type	Description
setGrid	void	Function to set the table showing all reservations.
setInfo	void	Function to set the form showing details of the selected reservation.
updateInfo	void	Function to update the form showing details of the selected

Method name	Return type	Description
		reservation.
clearForm	void	Function to clean the form showing details of the selected reservation.
startTrip	void	Button function to start the trip. It sets the reservation_start_date of the trip using IBusinessController.
endTrip	void	Button function to end the trip. It sets the reservation_end_date of the trip using IBusinessController.
updateGrid	void	Function to update the table showing all reservations.

2.2 controller

The model package contains the classes and packages of the business layer.

It contains the controller classes and interfaces, implementing the business logic of the system.

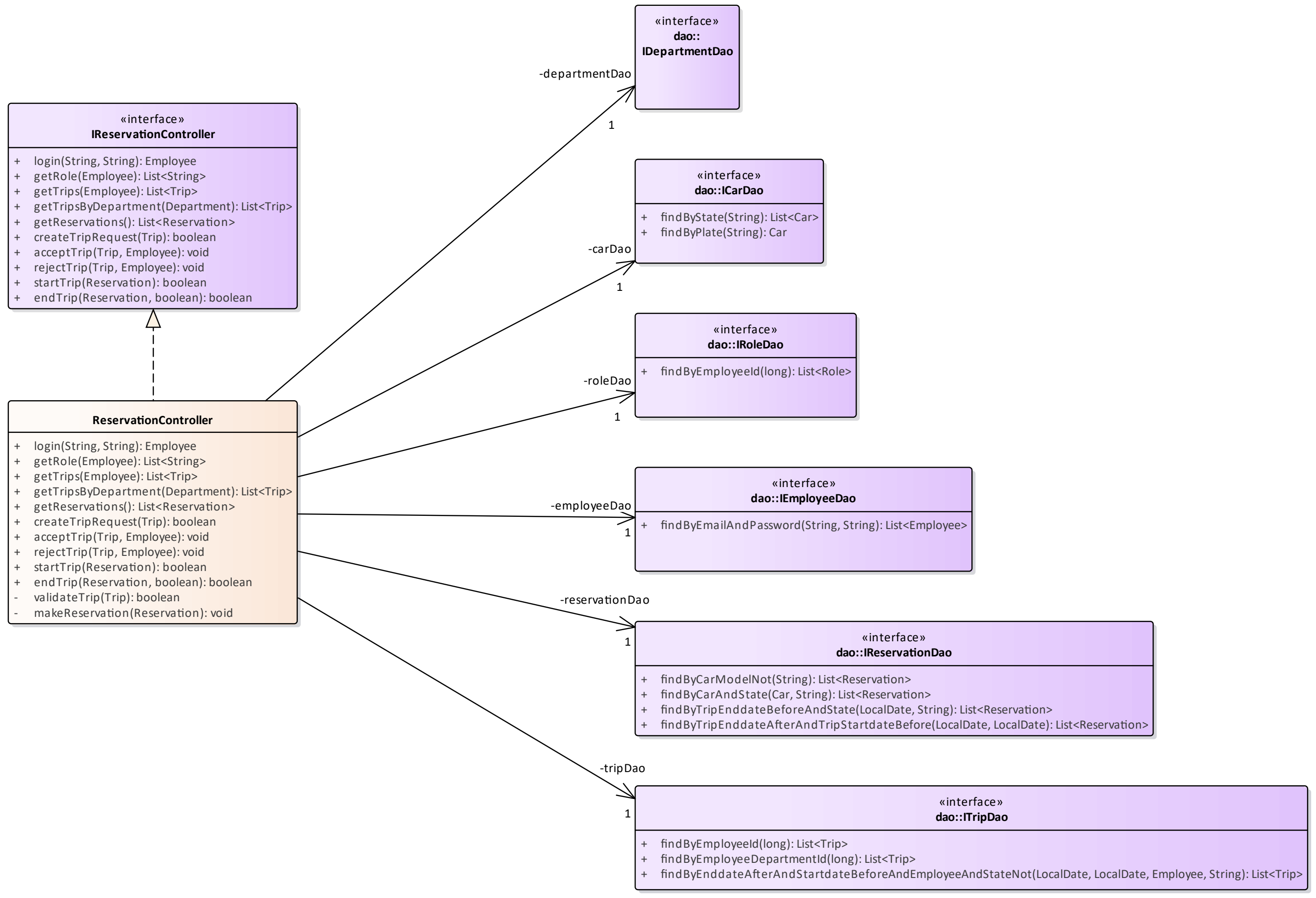

«interface»
IReservationController
+ login(String, String): Employee
+ getRole(Employee): List<String>
+ getTrips(Employee): List<Trip>
+ getTripsByDepartment(Department): List<Trip>
+ getReservations(): List<Reservation>
+ createTripRequest(Trip): boolean
+ acceptTrip(Trip, Employee): void
+ rejectTrip(Trip, Employee): void
+ startTrip(Reservation): boolean
+ endTrip(Reservation, boolean): boolean
ReservationController
+ login(String, String): Employee
+ getRole(Employee): List<String>
+ getTrips(Employee): List<Trip>
+ getTripsByDepartment(Department): List<Trip>
+ getReservations(): List<Reservation>
+ createTripRequest(Trip): boolean
+ acceptTrip(Trip, Employee): void
+ rejectTrip(Trip, Employee): void
+ startTrip(Reservation): boolean
+ endTrip(Reservation, boolean): boolean
- validateTrip(Trip): boolean
- makeReservation(Reservation): void
-departmentDao
1
«interface»
dao::
IDepartmentDao
-carDao
1
«interface»
dao::ICarDao
+ findByState(String): List<Car>
+ findByPlate(String): Car
-roleDao
1
«interface»
dao::IRoleDao
+ findByEmployeeId(long): List<Role>
-employeeDao
1
«interface»
dao::IEmployeeDao
+ findByEmailAndPassword(String, String): List<Employee>
-reservationDao
1
«interface»
dao::IReservationDao
+ findByCarModelNot(String): List<Reservation>
+ findByCarAndState(Car, String): List<Reservation>
+ findByTripEnddateBeforeAndState(LocalDate, String): List<Reservation>
+ findByTripEnddateAfterAndTripStartdateBefore(LocalDate, LocalDate): List<Reservation>
-tripDao
1
«interface»
dao::ITripDao
+ findByEmployeeId(long): List<Trip>
+ findByEmployeeDepartmentId(long): List<Trip>
+ findByEnddateAfterAndStartdateBeforeAndEmployeeAndStateNot(LocalDate, LocalDate, Employee, String): List<Trip>

Figure 4 - ReservationController

2.2.1 IReservationController

Interface operations related to reservation management use cases.

Method name	Return type	Description
login	Employee	Parameters: **email: String** - Parameters: **pass: String** -
getRole	List<String>	Parameters: **user: Employee** -
getTrips	List<Trip>	Parameters: **user: Employee** -
getTripsByDepartment	List<Trip>	Parameters: **managing: Department** -
getReservations	List<Reservation>	
createTripRequest	boolean	Parameters: **trip: Trip** -
acceptTrip	void	Parameters: **trip: Trip** - Parameters: **user: Employee** -
rejectTrip	void	Parameters: **trip: Trip** - Parameters: **user: Employee** -
startTrip	boolean	Parameters: **reservation: Reservation** -
endTrip	boolean	Parameters: **reservation: Reservation** - Parameters:

Method name	Return type	Description
		damaged: boolean -

2.3 data

This package contains the classes and interfaces of the data layer.

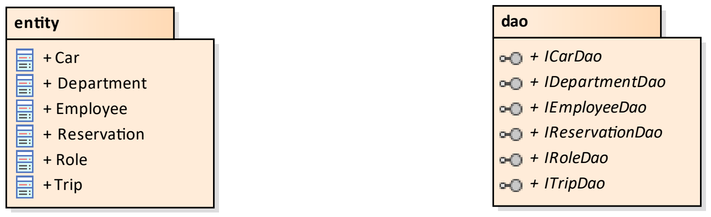


Figure 5 - data

2.3.1 entity

This package contains the classes of the entities, defining the data objects synchronized with a persistent storage.

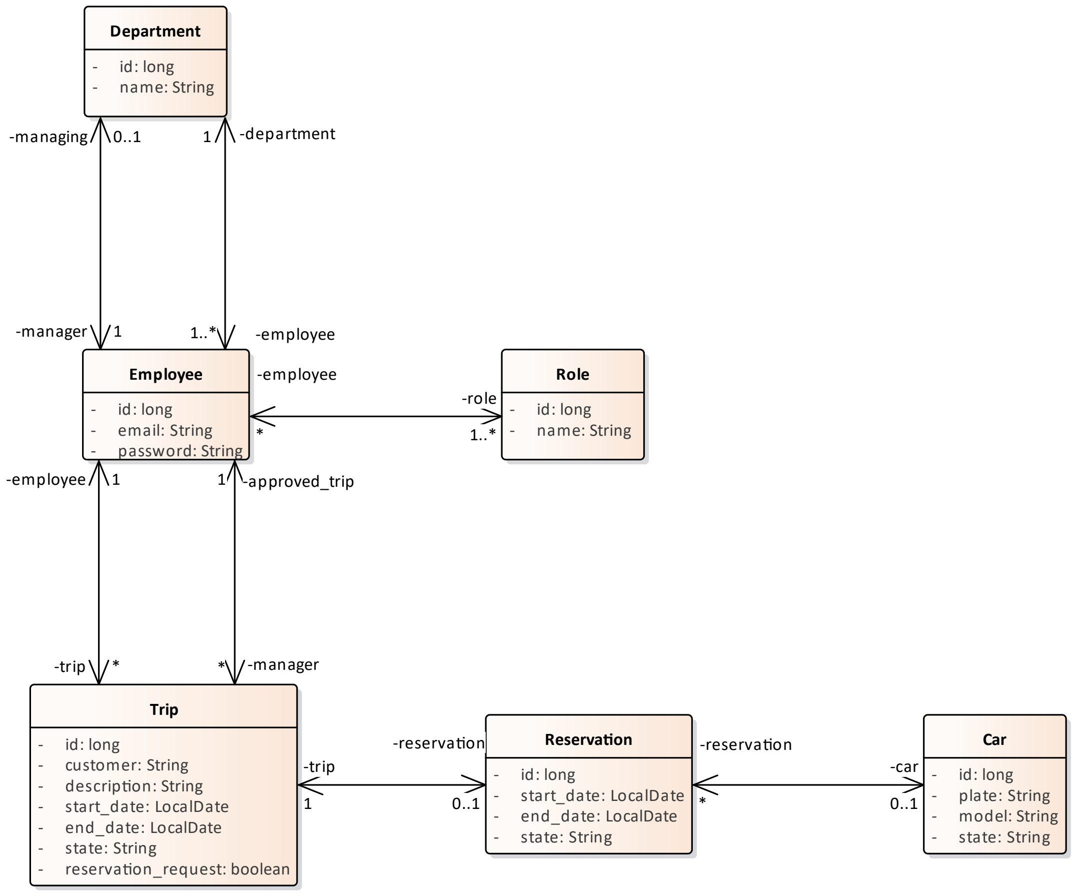


Figure 6 - entity

2.3.1.1 Car

The Car entity serves as a data holder for the data about car stored in a persistent storage.

Car represents information about a car which may be reserved for a business trip.

A car may be used in none or many reservations.

Attribute name	Data type	Description
id	long	Internal ID of car entity.
plate	String	Unique plate number of the car.
model	String	Model of the car.
state	String	State of the car (eg. OK, damaged).

2.3.1.2 Department

The Department entity serves as a data holder for the data about departments stored in a persistent storage.

Department represents information about a departments of the company.

A department must always have an employee managing it. It may also have one or many employees working.

Attribute name	Data type	Description
id	long	Internal ID of the department entity.
name	String	Name of the department.

2.3.1.3 Employee

The Employee entity serves as a data holder for the data about employees stored in a persistent storage.

Employee represents information about an employee working in the company.

An employee must have at least one role, also a department that s/he working for. It may also have a department that s/he is managing.

Attribute name	Data type	Description
id	long	Internal ID of the employee entity.
email	String	Unique email of the user.
password	String	Password of the user used to log in the system.

2.3.1.4 Reservation

The Reservation entity serves as a data holder for the data about reservations stored in a persistent storage.

Reservation represents information about a reservation which may be done for a business trip.

A reservation must always have a trip related to it, it may also have zero or one car according to availability.

Attribute name	Data type	Description
id	long	Internal ID of reservation.

Attribute name	Data type	Description
start_date	LocalDate	Realization start date of the reservation.
end_date	LocalDate	Realization end date of the reservation.
state	String	State of the reservation (eg. new, started, ended, expired).

2.3.1.5 Role

The Role entity serves as a data holder for the data about roles stored in a persistent storage.

Reservation represents information about a reservation which may be done for a business trip.

A reservation must always have a trip related to it, it may also have zero or one car according to availability.

Attribute name	Data type	Description
id	long	Internal ID of the role entity.
name	String	Name of the role.

2.3.1.6 Trip

The Trip entity serves as a data holder for the data about trips stored in a persistent storage.

Trip represents information about a business trips which are requested by employees.

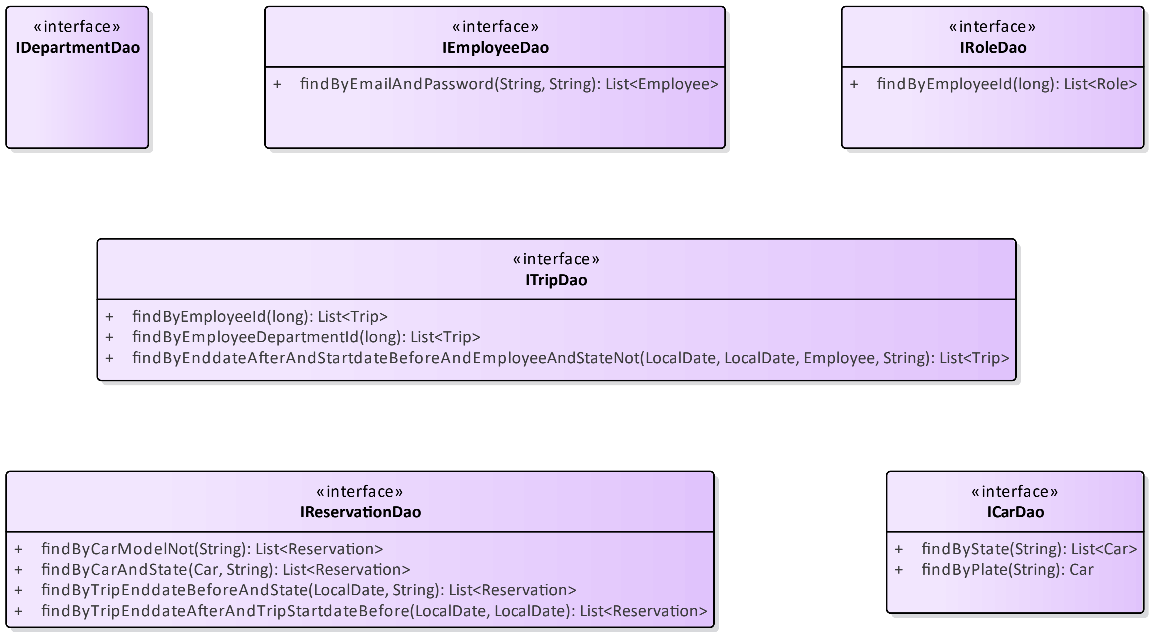
A trip must always have an employee that requested it and also an employee to approve it, it may also have zero or one reservation requests.

Attribute name	Data type	Description
id	long	Internal ID of the trip entity.
customer	String	Customer to be visited during the trip.
description	String	Optional short description about the trip.
start_date	LocalDate	Planned start date of the trip.
end_date	LocalDate	Planned end date of the trip.
state	String	State of the trip (eg. new, accepted, rejected).
reservation_request	boolean	Attribute to show if there is a car reservation request with the trip.

2.3.2 dao

This package defines the interfaces for DAOs - Data access objects - responsible for retrieving data from the persistent storage and persisting the changes. It maps the database tables to entities of the system.

Spring Boot's generic interfaces allow usage of the interface without implementation and also does not require to specify basic database operations such as getById() or findAll().


«interface» IDepartmentDao
«interface» IEmployeeDao
+ findByEmailAndPassword(String, String): List<Employee>
«interface» IRoleDao
+ findByEmployeeId(long): List<Role>
«interface» ITripDao
+ findByEmployeeId(long): List<Trip>
+ findByEmployeeDepartmentId(long): List<Trip>
+ findByEnddateAfterAndStartdateBeforeAndEmployeeAndStateNot(LocalDate, LocalDate, Employee, String): List<Trip>
«interface» IReservationDao
+ findByCarModelNot(String): List<Reservation>
+ findByCarAndState(Car, String): List<Reservation>
+ findByTripEnddateBeforeAndState(LocalDate, String): List<Reservation>
+ findByTripEnddateAfterAndTripStartdateBefore(LocalDate, LocalDate): List<Reservation>
«interface» ICarDao
+ findByState(String): List<Car>
+ findByPlate(String): Car
Figure 7 - dao

2.3.2.1 ICarDao

Interface defining the operations available for persistence of cars.

Method name	Return type	Description
findByState	List<Car>	Parameters: **state: String** -
findByPlate	Car	Parameters: **model: String** -

2.3.2.2 IDepartmentDao

Interface defining the operations available for persistence of departments.

2.3.2.3 IEmployeeDao

Interface defining the operations available for persistence of employee.

Method name	Return type	Description
findByEmailAndPassword	List<Employee>	Parameters: **email: String** - Parameters: **password: String** -

Method name	Return type	Description

2.3.2.4 IReservationDao

Interface defining the operations available for persistence of reservations.

Method name	Return type	Description
findByCarModelNot	List<Reservation>	Parameters: **model: String** -
findByCarAndState	List<Reservation>	Parameters: **car: Car** - Parameters: **state: String** -
findByTripEnddateBeforeAndState	List<Reservation>	Parameters: **date: LocalDate** - Parameters: **state: String** -
findByTripEnddateAfterAndTripStartdateBefore	List<Reservation>	Parameters: **start_date: LocalDate** - Parameters: **end_date: LocalDate** -

2.3.2.5 IRoleDao

Interface defining the operations available for persistence of roles.

Method name	Return type	Description
findByEmployeeId	List<Role>	Parameters: **id: long** -

2.3.2.6 ITripDao

Interface defining the operations available for persistence of trips.

Method name	Return type	Description
findByEmployeeId	List<Trip>	Parameters: **id: long** -
findByEmployeeDepartmentId	List<Trip>	Parameters:

Method name	Return type	Description
		id: long -
findByEnddateAfterAndStartdateBeforeAndEmployeeAndStateNot	List<Trip>	Parameters: **start_date: LocalDate** - Parameters: **end_date: LocalDate** - Parameters: **employee: Employee** - Parameters: **state: String** -

3. Realization Model

In this part, the realization of some of the use cases of the Company Fleet System is described, showing the communication of classes of the Web application.

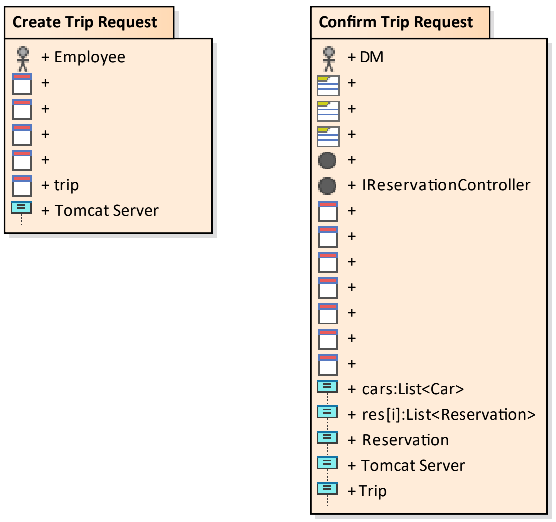


Figure 8 - Realization Model

3.1 Create Trip Request

This section describes realization of UC to create new trip requests within selected dates.

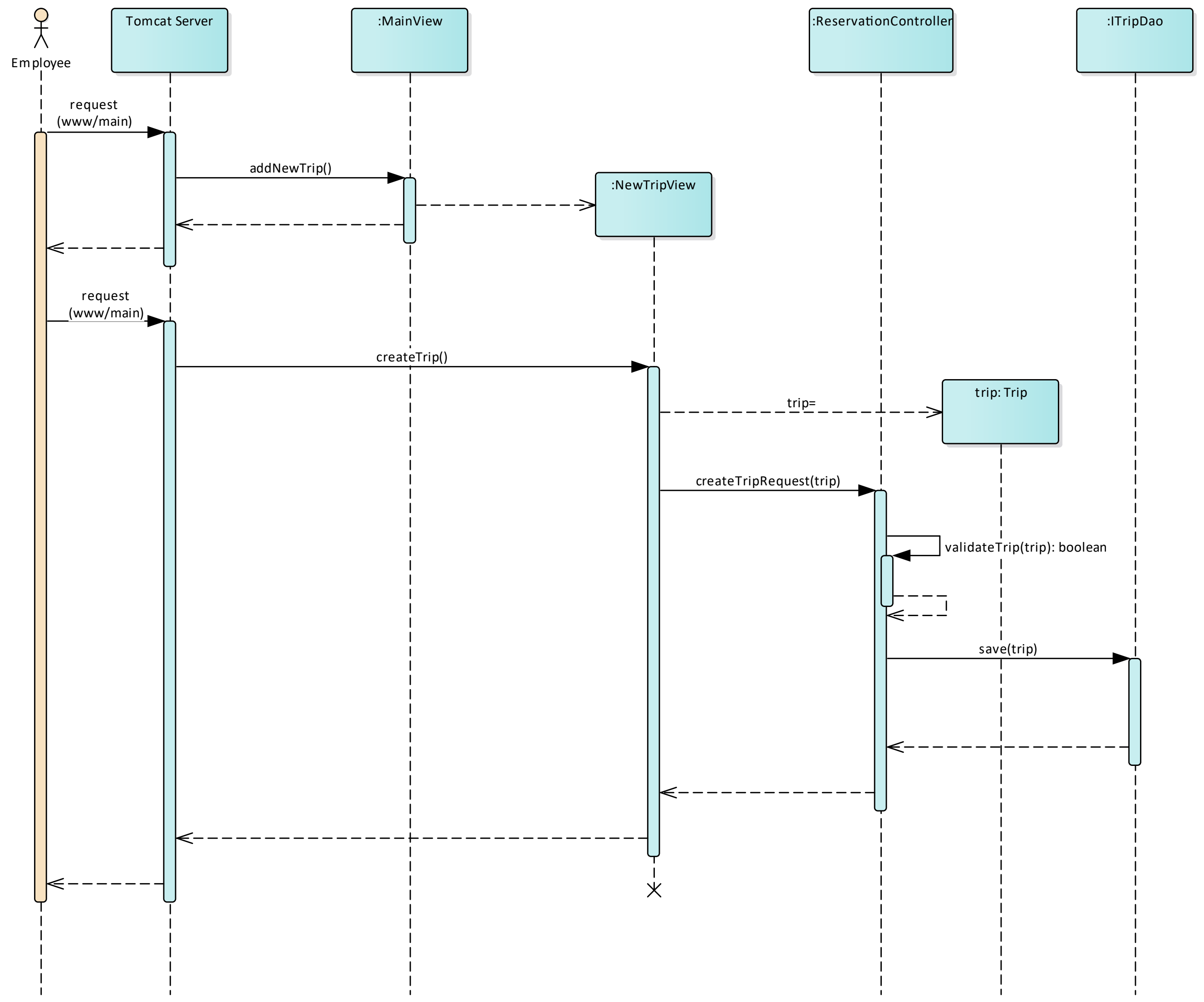


Figure 9 - Create Trip Request

This diagram describes the realization of creating trip request.

The user sends a request to the Tomcat server via HTTP. This request is processed by the web server and a method addNewTrip is called upon the NewTripView instance.

The NewTripView first creates an instance of Trip entity and send the message createTripRequest to the instance of the IReservationController.

ReservationController validates the trip and saves the trip via ITripDao by calling save and returns boolean value to indicate success.

3.2 Confirm Trip Request

This section describes realization of UC to confirm new trip requests.

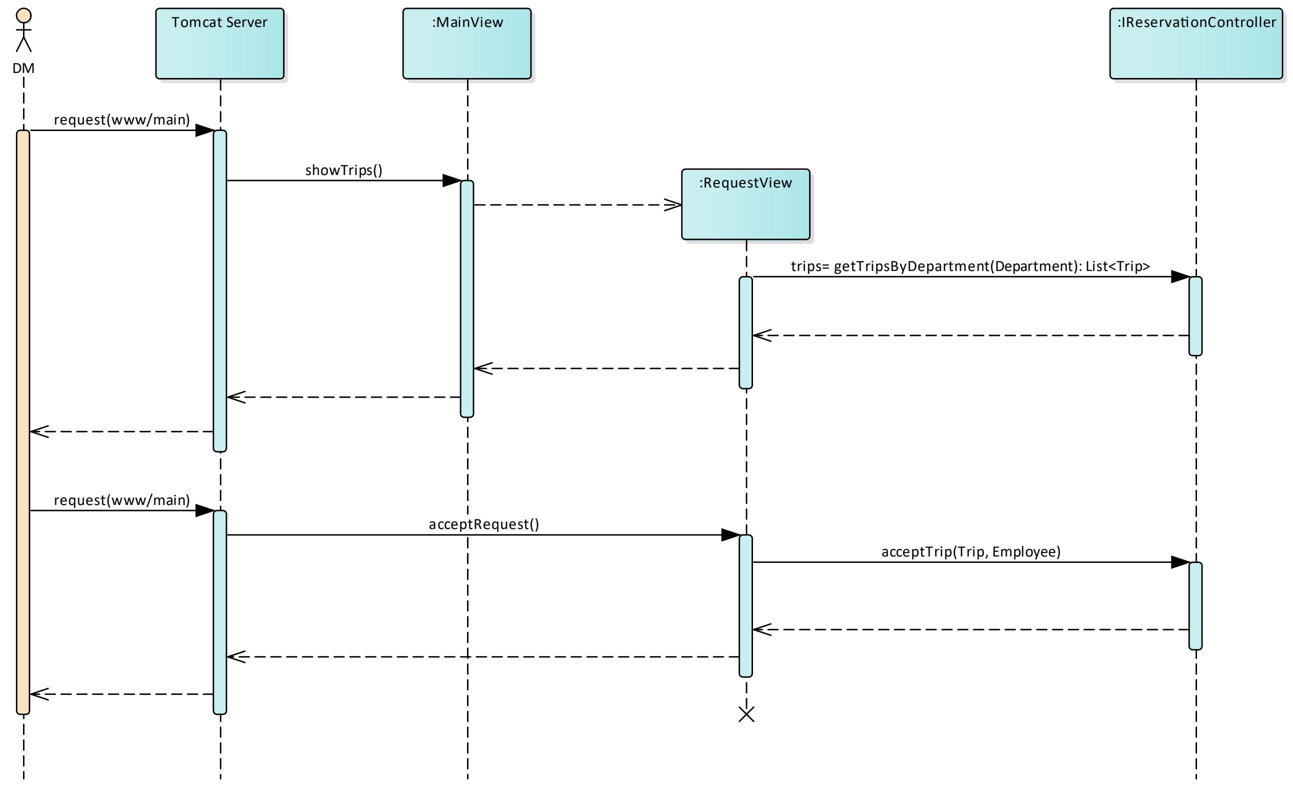


Figure 10 - Confirm Trip Request PL

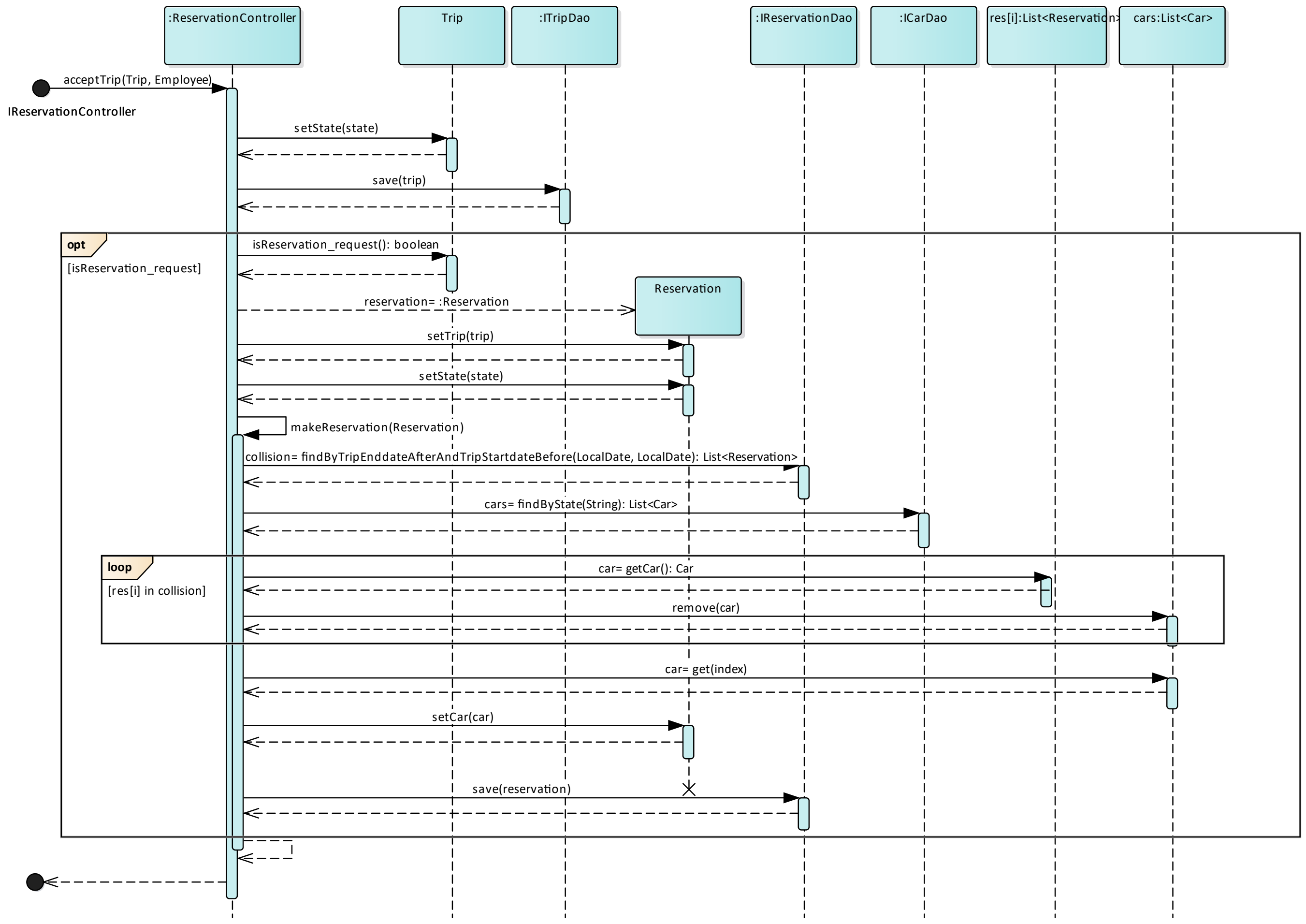


Figure 11 - Confirm Trip Request BL

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE