```java
/* src/entities/Person.java */
package entities;

import javax.xml.bind.annotation.XmlRootElement;

/**Represent a person, can be us or another person in an incident*/
@XmlRootElement
public class Person {

    private int personId;
    private String email;
    private String firstName;
    private String lastName;
    private String idNumber;
    private String city;
    private String streetName;
    private int houseNumber = -1;
    private String addressDetails;
    private int zipcode = -1;
    private String phone1;
    private String phone2;
    private String insuranceCompany;
    private String insuranceAgentName;
    private String insurancePhone1;
    private String insurancePhone2;
    private String insuranceNumber;

    /**Person constructor, without the person id that we got back from the database
     * Parameters:
     *      @param email String
     *      @param firstName String
     *      @param lastName String
     *      @param idNumber String, "teudat zehut"
     *      @param city String
     *      @param streetName String
     *      @param houseNumber integer
     *      @param addressDetails String, additional information, like if you have more then
     one entrance to the building
     *      @param zipcode integer
     *      @param phone1 String
     *      @param phone2 String
     *      @param insuranceCompany String, insurance company name
     *      @param insuranceAgentName String
     *      @param insurancePhone1 String
     *      @param insurancePhone2 String
     *      @param insuranceNumber String*/
    public Person(String email, String firstName,
            String lastName, String idNumber,String city, String streetName, int houseNumber,
            String addressDetails, int zipcode, String phone1, String phone2,
            String insuranceCompany, String insuranceAgentName, String insurancePhone1,
            String insurancePhone2, String insuranceNumber) {

        this.email = email;
        this.firstName = firstName;
        this.lastName = lastName;
        this.idNumber = idNumber;
        this.city = city;
        this.streetName = streetName;
        this.houseNumber = houseNumber;
        this.addressDetails = addressDetails;
        this.zipcode = zipcode;
        this.phone1 = phone1;
        this.phone2 = phone2;
        this.insuranceCompany = insuranceCompany;
        this.insuranceAgentName = insuranceAgentName;
        this.insurancePhone1 = insurancePhone1;
        this.insurancePhone2 = insurancePhone2;
        this.insuranceNumber = insuranceNumber;


    }

    /**Person constructor, a complete one
     * Parameters:
     *      @param personId integer
     *      @param email String
     *      @param firstName String
     *      @param lastName String
     *      @param idNumber String, "teudat zehut"
     *      @param city String
     *      @param streetName String
     *      @param houseNumber integer
     *      @param addressDetails String, additional information, like if you have more then
     one entrance to the building
     *      @param zipcode integer
     *      @param phone1 String
     *      @param phone2 String
     *      @param insuranceCompany String, insurance company name
     *      @param insuranceAgentName String
     *      @param insurancePhone1 String
     *      @param insurancePhone2 String
     *      @param insuranceNumber String*/
    public Person(int personId, String email, String firstName,
            String lastName, String idNumber, String city, String streetName, int houseNumber,
            String addressDetails, int zipcode, String phone1, String phone2,
            String insuranceCompany, String insuranceAgentName, String insurancePhone1,
            String insurancePhone2, String insuranceNumber) {

        this.personId = personId;
        this.email = email;
        this.firstName = firstName;
        this.lastName = lastName;
        this.idNumber = idNumber;
        this.city = city;
        this.streetName = streetName;
        this.houseNumber = houseNumber;
        this.addressDetails = addressDetails;
        this.zipcode = zipcode;
        this.phone1 = phone1;
        this.phone2 = phone2;
        this.insuranceCompany = insuranceCompany;
        this.insuranceAgentName = insuranceAgentName;
        this.insurancePhone1 = insurancePhone1;
        this.insurancePhone2 = insurancePhone2;
        this.insuranceNumber = insuranceNumber;
    }

    /**An empty constructor*/
    public Person() {
        //this constructor is needed so jaxb doesn't throw an exception related to no-arg
        constructor
    }

    /**A PersonId getter
     * Returns:
     *      @return integer*/
    public int getPersonId() {
        return personId;
    }

    /**A PersonId setter
     * Parameters:
     *      @param PersonId integer*/
    public void setPersonId(int personId) {
        this.personId = personId;
    }

    /**An email getter
     * Returns:
     *      @return integer*/
    public String getEmail() {
        return email;
    }

    /**An email setter
     * Parameters:
     *      @param email String*/
    public void setEmail(String email) {
        this.email = email;
    }

    /**A first name getter
     * Returns:
     *      @return String*/
    public String getFirstName() {
        return firstName;
    }

    /**A first name setter
     * Parameters:
     *      @param first name String*/
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    /**A last name getter
     * Returns:
     *      @return String*/
    public String getLastName() {
        return lastName;
    }

    /**A last name setter
     * Parameters:
     *      @param last name String*/
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    /**An id number ("tedudat zehout") getter
     * Returns:
     *      @return integer*/
    public String getIdNumber() {
        return idNumber;
    }

    /**An id number ("tedudat zehout") setter
     * Parameters:
     *      @param id number integer*/
    public void setIdNumber(String idNumber) {
        this.idNumber = idNumber;
    }

    /**A city getter
     * Returns:
     *      @return String*/
    public String getCity() {
        return city;
    }

    /**A city setter
     * Parameters:
     *      @param city String*/
    public void setCity(String city) {
        this.city = city;
    }

    /**A street name getter
     * Returns:
     *      @return String*/
    public String getStreetName() {
        return streetName;
    }

    /**A street name setter
     * Parameters:
     *      @param street name String*/
    public void setStreetName(String streetName) {
        this.streetName = streetName;
    }

    /**A house number getter
     * Returns:
     *      @return integer*/
    public int getHouseNumber() {
        return houseNumber;
    }

    /**A house number setter
     * Parameters:
     *      @param house number integer*/
    public void setHouseNumber(int houseNumber) {
        this.houseNumber = houseNumber;
    }

    /**An address details getter
```

```
     * Returns:
     *      @return String*/
    public String getAddressDetails() {
        return addressDetails;
    }

    /**An address details setter
     * Parameters:
     *      @param address details String*/
    public void setAddressDetails(String addressDetails) {
        this.addressDetails = addressDetails;
    }

    /**A zipcode getter
     * Returns:
     *      @return integer*/
    public int getZipcode() {
        return zipcode;
    }

    /**A zipcode setter
     * Parameters:
     *      @param zipcode integer*/
    public void setZipcode(int zipcode) {
        this.zipcode = zipcode;
    }

    /**A phone 1 getter
     * Returns:
     *      @return String*/
    public String getPhone1() {
        return phone1;
    }

    /**A phone 1 setter
     * Parameters:
     *      @param PersonId String*/
    public void setPhone1(String phone1) {
        this.phone1 = phone1;
    }

    /**A phone 2 getter
     * Returns:
     *      @return String*/
    public String getPhone2() {
        return phone2;
    }

    /**A phone 2 getter
     * Returns:
     *      @return String*/
    public void setPhone2(String phone2) {
        this.phone2 = phone2;
    }

    /**An insurance company name getter
     * Returns:
     *      @return String*/
    public String getInsuranceCompany() {
```

```
    return insuranceCompany;
}

/**An insurance company name setter
 * Parameters:
 *      @param insurance company name String*/
public void setInsuranceCompany(String insuranceCompany) {
    this.insuranceCompany = insuranceCompany;
}


/**An insurance agent name getter
 * Returns:
 *      @return String*/
public String getInsuranceAgentName() {
    return insuranceAgentName;
}

/**An insurance agent name setter
 * Parameters:
 *      @param insurance agent name String*/
public void setInsuranceAgentName(String insuranceAgentName) {
    this.insuranceAgentName = insuranceAgentName;
}

/**An insurance phone 1 getter
 * Returns:
 *      @return String*/
public String getInsurancePhone1() {
    return insurancePhone1;
}

/**An insurance phone 1 setter
 * Parameters:
 *      @param insurance phone 1 String*/
public void setInsurancePhone1(String insurancePhone1) {
    this.insurancePhone1 = insurancePhone1;
}


/**An insurance phone 2 getter
 * Returns:
 *      @return String*/
public String getInsurancePhone2() {
    return insurancePhone2;
}

/**An insurance phone 2 setter
 * Parameters:
 *      @param insurance phone 2 String*/
public void setInsurancePhone2(String insurancePhone2) {
    this.insurancePhone2 = insurancePhone2;
}


/**An insurance number getter
 * Returns:
 *      @return String*/
public String getInsuranceNumber() {
        return insuranceNumber;
    }

    /**An insurance number setter
     * Parameters:
     *       @param insurance number String*/
    public void setInsuranceNumber(String insuranceNumber) {
        this.insuranceNumber = insuranceNumber;
    }
}
```

```java
/* src/entities/User.java */
package entities;

import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;

/**Represent a user, which is an extend of a person*/
@XmlRootElement
public class User extends Person {

    private int userId;
    private String username;
    private String password;
    private String authHash;

    /**Incident constructor
     * Parameters:
     *       @param int userId
     *       @param username String
     *       @param password String
     *       @param personId integer
     *       @param email String
     *       @param firstName String
     *       @param lastName String
     *       @param idNumber String, "teudat zehut"
     *       @param city String
     *       @param streetName String
     *       @param houseNumber integer
     *       @param addressDetails String, additional information, like if you have more then
     one entrance to the building
     *       @param zipcode integer
     *       @param phone1 String
     *       @param phone2 String
     *       @param insuranceCompany String, insurance company name
     *       @param insuranceAgentName String
     *       @param insurancePhone1 String
     *       @param insurancePhone2 String
     *       @param insuranceNumber String
     * */
    public User(int userId, String username, String password, int personId, String email,
    String firstName,
            String lastName, String idNumber, String city, String streetName, int houseNumber,
            String addressDetails, int zipcode, String phone1, String phone2,
            String insuranceCompany, String insuranceAgentName, String insurancePhone1,
            String insurancePhone2, String insuranceId) {

        super(personId, email, firstName,
                lastName, idNumber, city, streetName, houseNumber,
                addressDetails, zipcode, phone1, phone2,
                insuranceCompany, insuranceAgentName, insurancePhone1,
                insurancePhone2, insuranceId);

        this.userId = userId;
        this.username = username;
        this.password = password;
    }

    /**An empty constructor*/
    public User() {
```

```
    //this constructor is needed so jaxb doesn't throw an exception related to no-arg
    constructor
}

/**A user id getter
 * Returns:
 *      @return integer*/
public int getUserId() {
    return userId;
}

/**A user id setter
 * Parameters:
 *      @param user id integer*/
public void setUserId(int userId) {
    this.userId = userId;
}

/**A user name getter
 * Returns:
 *      @return String*/
public String getUsername() {
    return username;
}

/**A user name setter
 * Parameters:
 *      @param user name String*/
public void setUsername(String username) {
    this.username = username;
}

/**A password getter
 * Returns:
 *      @return String*/
@XmlTransient
public String getPassword() {
    return password;
}

/**A password setter
 * Parameters:
 *      @param password String*/
public void setPassword(String password) {
    this.password = password;
}

/**An authentication Hash getter
 * Returns:
 *      @return String*/
public String getAuthHash() {
    return authHash;
}

/**An authentication Hash setter
 * Parameters:
 *      @param authentication Hash String*/
public void setAuthHash(String authHash) {
    this.authHash = authHash;
```

```
        }

}
```

```java
/* src/entities/Incident.java */
package entities;

import java.text.SimpleDateFormat;
import java.util.Date;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;

/**Represent an incident event, there can be more then one incident to a user*/
@XmlRootElement
public class Incident {

    private int incidentId;
    private int userId;
    private Date date;
    private String notes;
    private String location;
    private String vehicleLicensePlate;
    private String vehicleBrand;
    private String vehicleModel;
    private Person driver;
    private Person owner;


    /**Incident constructor
     * Parameters:
     *      @param incidentId integer
     *      @param userId integer
     *      @param date Date
     *      @param location String (will change to latitude and longitude in the future)
     *      @param note String
     *      @param driver Person, the "other" side of the incident
     *      @param owner Person, our data*/
    public Incident(int incidentId, int userId, Date date,String location, String notes,
    Person driver, Person owner) {
        this.incidentId = incidentId;
        this.userId = userId;
        this.date = date;
        this.location = location;
        this.notes = notes;
        this.driver = driver;
        this.owner = owner;
    }

    /**An empty constructor*/
    public Incident() {
        //this constructor is needed so jaxb doesn't throw an exception related to no-arg
        constructor
    }

    /**An incidentId getter
     * Returns:
     *      @return integer*/
    public int getIncidentId() {
        return incidentId;
    }

    /**An incidentId setter
```

```
 * Parameters:
 *      @param incidentId integer*/
public void setIncidentId(int incidentId) {
    this.incidentId = incidentId;
}

/**A userId getter
 * Returns:
 *      @return integer*/
public int getUserId() {
    return userId;
}

/**A userId setter
 * Parameters:
 *      @param userId integer*/
public void setUserId(int userId) {
    this.userId = userId;
}

/**A date getter
 * Returns:
 *      @return Date */
@XmlTransient
public Date getDate() {
    return date;
}

/**A date getter
 * Returns:
 *      @return String*/
@XmlElement(name="date")
public String getDateString() {
    if (null == date) {
        return null;
    }
    SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm");//yyyy-mm-dd
    hh:mm:ss.fffffffff
    return df.format(date);
}

/**A date setter
 * Parameters:
 *      @param date Date*/
public void setDate(Date date) {
    this.date = date;
}

/**A location getter
 * Returns:
 *      @return String*/
public String getLocation() {
    return location;
}

/**A location setter
 * Parameters:
 *      @param location String*/
public void setLocation(String location) {
    this.location = location;
}

/**A notes getter
 * Returns:
 *      @return String*/
public String getNotes() {
    return notes;
}

/**A notes setter
 * Parameters:
 *      @param notes String*/
public void setNotes(String notes) {
    this.notes = notes;
}

/**A vehicle license plate getter
 * Returns:
 *      @return String*/
public String getVehicleLicensePlate() {
    return vehicleLicensePlate;
}

/**A vehicle license plate setter
 * Parameters:
 *      @param vehicle license plate String*/
public void setVehicleLicensePlate(String vehicleLicensePlate) {
    this.vehicleLicensePlate = vehicleLicensePlate;
}

/**A vehicle brand getter
 * Returns:
 *      @return String*/
public String getVehicleBrand() {
    return vehicleBrand;
}

/**A vehicle brand setter
 * Parameters:
 *      @param vehicle brand String*/
public void setVehicleBrand(String vehicleBrand) {
    this.vehicleBrand = vehicleBrand;
}

/**A vehicle model getter
 * Returns:
 *      @return String*/
public String getVehicleModel() {
    return vehicleModel;
}

/**A vehicle model setter
 * Parameters:
 *      @param vehicle model String*/
public void setVehicleModel(String vehicleModel) {
    this.vehicleModel = vehicleModel;
}
```

```java
    /**A driver getter, the other side of the incident
     * Returns:
     *      @return Person*/
    public Person getDriver() {
        return driver;
    }

    /**An driver setter, the other side of the incident
     * Parameters:
     *      @param driver Person*/
    public void setDriver(Person driver) {
        this.driver = driver;
    }

    /**A owner getter, our data
     * Returns:
     *      @return Person*/
    public Person getOwner() {
        return owner;
    }

    /**A owner setter, our data
     * Parameters:
     *      @param owner Person*/
    public void setOwner(Person owner) {
        this.owner = owner;
    }
}
```

```java
/* src/entities/IncidentPage.java */
package entities;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementWrapper;
import javax.xml.bind.annotation.XmlRootElement;
import entities.Incident;

/**Represent a single page (web page) of incidents*/
@XmlRootElement
public class IncidentPage {

    private Incident[] incidents;

    private int totalLines;

    /**An empty constructor*/
    public IncidentPage () {
        //this constructor is needed so jaxb doesn't throw an exception related to no-arg
        constructor
    }

    /**Incident constructor
     * Parameters:
     *      @param incidents Incident[]
     *      @param totalLines integer
     * */
    public IncidentPage (Incident[] incidents, int totalLines){
        this.incidents = incidents;
        this.totalLines = totalLines;
    }

    /**An incidents getter
     * Returns:
     *      @return Incident[]*/
    @XmlElement(name="incident")
    @XmlElementWrapper(name="incidents")
    public Incident[] getIncidents() {
        return incidents;
    }

    /**An incidents setter
     * Parameters:
     *      @param incidents Incident[]*/
    public void setIncidents(Incident[] incidents) {
        this.incidents = incidents;
    }

    /**An total lines getter, return the total number of incidents that we have, for the
    pager.
     * Returns:
     *      @return integer*/
    public int getTotalLines() {
        return totalLines;
    }

    /**An total lines setter, sets the total number of incidents that we have, for the pager.
     * Parameters:
     *      @param total lines integer*/
    public void setTotalLines(int totalLines) {
        this.totalLines = totalLines;
    }



}
```

```java
/* src/db/MySql.java */
package db;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import app.Config;

public class MySql {

    static private MySql instance = null;

    private Connection connection;

    // protected so creation is only handled through getInstance
    protected MySql() {
        try {
            Config c = Config.getInstance();
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            //Class.forName("com.mysql.jdbc.Driver");
            try {
                connection = DriverManager.getConnection("jdbc:mysql://"+c.getDbHost()+"/"+c.
                getDbSchema()+"?user="+c.getDbUser()+"&password="+c.getDbPassword());
            } catch (SQLException e) {
                System.out.print("\ncan't get connection\n");
                e.printStackTrace();
            }
        } catch (Exception e) {
            System.out.print("\ncan't get driver\n");
            e.printStackTrace();
        }

    }

    public Connection getConnection() {
        return connection;
    }

    static public MySql getInstance() {
        if (null == instance) {
            instance = new MySql();
        }
        return instance;
    }
}
```

```java
/* src/dao/PersonDao.java */
package dao;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import db.MySql;
import entities.Person;
import entities.ResourceError;

public class PersonDao {

    static private PersonDao instance = null;
    private Connection connection;

    protected PersonDao() {
        connection = MySql.getInstance().getConnection();
    }

    /**Return a Person object given an id
     * Parameters:
     *      @param personId integer
     * Returns:
     *      @return Person
     * Throws:
     *      @exception SQLException
     * */
    public Person getById(int id) {
        Person person = new Person();
        try {
            CallableStatement callable = connection.prepareCall("call sp_getPersonById(?)");
            callable.setInt(1, id);
            ResultSet result = callable.executeQuery();
            if (!result.first()) {
                return null;
            }
            person.setPersonId(result.getInt("personId"));
            person.setEmail(result.getString("email"));
            person.setFirstName(result.getString("firstName"));
            person.setLastName(result.getString("lastName"));
            person.setIdNumber(result.getString("idNumber"));
            person.setCity(result.getString("city"));
            person.setStreetName(result.getString("streetName"));
            person.setHouseNumber(result.getInt("houseNumber"));
            person.setAddressDetails(result.getString("addressDetails"));
            person.setZipcode(result.getInt("zipcode"));
            person.setPhone1(result.getString("phone1"));
            person.setPhone2(result.getString("phone2"));
            person.setInsuranceCompany(result.getString("insuranceCompany"));
            person.setInsuranceAgentName(result.getString("insuranceAgentName"));
            person.setInsurancePhone1(result.getString("insurancePhone1"));
            person.setInsurancePhone2(result.getString("insurancePhone2"));
            person.setInsuranceNumber(result.getString("insuranceNumber"));
            return person;
        }
        catch (SQLException e) {
            //e.printStackTrace();
            ResourceError err = ResourceError.getInstance();
            err.setMessage("internal error");
            err.setStatusCode(500);
            err.setReasonCode(ResourceError.REASON_UNKNOWN);
            return null;
        }
    }

    /**Insert a Person object to the DB
     * Parameters:
     *      @param person Person
     * Returns:
     *      @return boolean
     * Throws:
     *      @exception SQLException*/
    public boolean save(Person person) {
        // insert the person to the db
        try {

            CallableStatement callable;
            int i = 0;
            if (0 < person.getPersonId()) {
                callable = connection.prepareCall("call
                sp_updatePerson(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)");
                callable.setInt(++i,person.getPersonId());
            }
            else {
                callable = connection.prepareCall("call
                sp_createPerson(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)");
            }

            callable.setString(++i,person.getEmail());
            callable.setString(++i,person.getFirstName());
            callable.setString(++i,person.getLastName());
            callable.setString(++i,person.getIdNumber());
            callable.setString(++i,person.getStreetName());
            callable.setString(++i,person.getCity());
            callable.setInt(++i,person.getHouseNumber());
            callable.setString(++i,person.getAddressDetails());
            callable.setInt(++i,person.getZipcode());
            callable.setString(++i,person.getPhone1());
            callable.setString(++i,person.getPhone2());
            callable.setString(++i,person.getInsuranceCompany());
            callable.setString(++i,person.getInsuranceAgentName());
            callable.setString(++i,person.getInsurancePhone1());
            callable.setString(++i,person.getInsurancePhone2());
            callable.setString(++i,person.getInsuranceNumber());
            callable.registerOutParameter(++i, java.sql.Types.INTEGER);
            callable.execute();

            if (0 < person.getPersonId()) { // update mode
                int rowsUpdated = callable.getInt(i);
                if (1 != rowsUpdated) {
                    ResourceError err = ResourceError.getInstance();
                    if (!err.isSet()) {
                        err.setMessage("user update failed, invalid input");
                        err.setStatusCode(400);
                        err.setReasonCode(ResourceError.REASON_INVALID_INPUT);
                    }
                    return false;
                }
            }
            else { // insert mode
                int personId = callable.getInt(i);
                person.setPersonId(personId);
            }

            return true;
        }
        catch (SQLException e) {
            ResourceError err = ResourceError.getInstance();
            if (!err.isSet()) {
                err.setMessage("internal error");
                err.setStatusCode(500);
                err.setReasonCode(ResourceError.REASON_UNKNOWN);
            }
            return false;
        }
    }

    /**Handle a person object, created so we won't have to use static methods
     * Return:
     *      @return Person
     * */
    static public PersonDao getInstance() {
        if (null == instance) {
            instance = new PersonDao();
        }
        return instance;
    }

}
```

```java
/* src/dao/UserDao.java */
package dao;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import db.MySql;
import entities.ResourceError;
import entities.User;

public class UserDao {

    static private UserDao instance = null;
    private Connection connection;

    protected UserDao() {
        connection = MySql.getInstance().getConnection();
    }

    /**Return a User object given an id
     * Parameters:
     *      @param userId integer
     * Returns:
     *      @return User
     * Throws:
     *      @exception SQLException
     * */
    public User getById(int id) {
        try {
            CallableStatement callable = connection.prepareCall("call sp_getUserById(?)");
            callable.setInt(1, id);
            ResultSet result = callable.executeQuery();
            return getByResult(result);
        }
        catch (SQLException e) {
            ResourceError err = ResourceError.getInstance();
            err.setMessage("internal error");
            err.setStatusCode(500);
            err.setReasonCode(ResourceError.REASON_UNKNOWN);
            return null;
        }
    }

    /**Set authentication string given a password string, if it is related to the user and
    valid
     *  Parameters:
     *      @param username String
     *      @param password String
     * Returns:
     *      @return null
     * Throws:
     *      @exception SQLException
     * */
    public User authenticate(String username, String password) {
        try {
            CallableStatement callable = connection.prepareCall("call
            sp_authenticateUser(?,?,?,?)");
            callable.setString(1, username);
            callable.setString(2, password);
            callable.registerOutParameter(3, java.sql.Types.INTEGER);
            callable.registerOutParameter(4, java.sql.Types.VARCHAR);
            callable.execute();
            int userId = callable.getInt("userIdOut");
            String hashResult = callable.getString("hashResultOut");
            if (0 >= userId || "" == hashResult) {
                ResourceError err = ResourceError.getInstance();
                err.setMessage("authentication failed, wrong username or password");
                err.setStatusCode(400);
                err.setReasonCode(ResourceError.REASON_AUTHENTICATION_FAILED);
                return null;
            }
            User user = getById(userId);
            if (null == user) {
                ResourceError err = ResourceError.getInstance();
                if (!err.isSet()) {
                    err.setMessage("user retrieval failed with provided input");
                    err.setStatusCode(400);
                    err.setReasonCode(ResourceError.REASON_INVALID_INPUT);
                }
                return null;
            }
            user.setAuthHash(hashResult);
            return user;
        }
        catch (SQLException e) {
            ResourceError err = ResourceError.getInstance();
            if (!err.isSet()) {
                err.setMessage("internal error");
                err.setStatusCode(500);
                err.setReasonCode(ResourceError.REASON_UNKNOWN);
            }
            return null;
        }
    }

    /**Checks if the authentication string is related to the user and valid
     *  Parameters:
     *      @param userId integer
     *      @param hash String
     * Returns:
     *      @return boolean
     * Throws:
     *      @exception SQLException
     * */
    public boolean isAuthorized(int userId, String hash) {
        try {
            CallableStatement callable = connection.prepareCall("call
            sp_authorizeUser(?,?,?)");
            callable.setInt(1, userId);
            callable.setString(2, hash);
            callable.registerOutParameter(3, java.sql.Types.BOOLEAN);
            callable.execute();
            return callable.getBoolean("authResultOut");

        }
        catch (SQLException e) {
            ResourceError err = ResourceError.getInstance();
            if (!err.isSet()) {
                err.setMessage("internal error");
                err.setStatusCode(500);
                err.setReasonCode(ResourceError.REASON_UNKNOWN);
            }
            return false;
        }
    }

    /**Insert a User object to the DB
     * Parameters:
     *      @param user User
     * Returns:
     *      @return boolean
     * Throws:
     *      @exception SQLException
     * */
    public boolean save(User user) {
        // insert the user to the db
        try {

            CallableStatement callable;
            int i = 0;
            boolean updateMode = false;
            if (0 < user.getUserId() && 0 < user.getPersonId()) {
                updateMode = true;
                callable = connection.prepareCall("call
                sp_updateUser(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)");
                callable.setInt(++i,user.getPersonId());
                callable.setInt(++i,user.getUserId());
            }
            else {
                callable = connection.prepareCall("call
                sp_createUser(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)");
            }

            callable.setString(++i,user.getEmail());
            callable.setString(++i,user.getFirstName());
            callable.setString(++i,user.getLastName());
            callable.setString(++i,user.getIdNumber());
            callable.setString(++i,user.getStreetName());
            callable.setString(++i,user.getCity());
            callable.setInt(++i,user.getHouseNumber());
            callable.setString(++i,user.getAddressDetails());
            callable.setInt(++i,user.getZipcode());
            callable.setString(++i,user.getPhone1());
            callable.setString(++i,user.getPhone2());
            callable.setString(++i,user.getInsuranceCompany());
            callable.setString(++i,user.getInsuranceAgentName());
            callable.setString(++i,user.getInsurancePhone1());
            callable.setString(++i,user.getInsurancePhone2());
            callable.setString(++i,user.getInsuranceNumber());
            if (!updateMode) { // pass username to the sp only if we are saving it in the
            first time
                callable.setString(++i,user.getUsername());
                callable.setString(++i,user.getPassword()); //TODO password should be md5ed
                before inserting or updating
            }
            callable.registerOutParameter(++i, java.sql.Types.INTEGER); // register
            personIdOut or rowsUpdatedPersonOut (if in updateMode)
            callable.registerOutParameter(++i, java.sql.Types.INTEGER); // register
            userIdOut or rowsUpdatedUserOut (if in updateMode)
            if (!updateMode) {
                callable.registerOutParameter(++i, java.sql.Types.VARCHAR); // register
                authHashOut
            }
            callable.execute();

            if (updateMode) { // update mode
                int rowsUpdatedPerson = callable.getInt("rowsUpdatedPersonOut");
                int rowsUpdatedUser = callable.getInt("rowsUpdatedUserOut");
                //TODO should roll back if rowsUpdated greater then 1
                if (1 != rowsUpdatedPerson || 1 != rowsUpdatedUser) {
                    ResourceError err = ResourceError.getInstance();
                    if (!err.isSet()) {
                        err.setMessage("user update failed, invalid input");
                        err.setStatusCode(400);
                        err.setReasonCode(ResourceError.REASON_INVALID_INPUT);
                    }
                    return false;
                }
            }
            else { // insert mode
                int personId = callable.getInt("personIdOut");
                int userId = callable.getInt("userIdOut");
                String authHash = callable.getString("authHashOut");
                user.setUserId(userId);
                user.setPersonId(personId);
                user.setAuthHash(authHash);
            }

            return true;
        }
        catch (SQLException e) {
            ResourceError err = ResourceError.getInstance();
            if (!err.isSet()) {
                err.setMessage("internal error");
                err.setStatusCode(500);
                err.setReasonCode(ResourceError.REASON_UNKNOWN);
            }
            return false;
        }
    }

    /**Handles the result set from the DB, via callable.executeQuery(), and creating the
    user instance
     * Parameters:
     *      @param result ResultSet
     * Return:
     *      @return User
     * Throws:
     *      @exception SQLException
     * */
    private User getByResult(ResultSet result) {
        User user = new User();
        try {
            if (!result.first()) {
                ResourceError err = ResourceError.getInstance();
```

```
                err.setMessage("user retrieval failed with provided input");
                err.setStatusCode(400);
                err.setReasonCode(ResourceError.REASON_INVALID_INPUT);
                return null;
            }
            user.setUserId(result.getInt("userId"));
            user.setPersonId(result.getInt("personId"));
            user.setUsername(result.getString("username"));
            user.setPassword(result.getString("password"));
            user.setEmail(result.getString("email"));
            user.setFirstName(result.getString("firstName"));
            user.setLastName(result.getString("lastName"));
            user.setIdNumber(result.getString("idNumber"));
            user.setCity(result.getString("city"));
            user.setStreetName(result.getString("streetName"));
            user.setHouseNumber(result.getInt("houseNumber"));
            user.setAddressDetails(result.getString("addressDetails"));
            user.setZipcode(result.getInt("zipcode"));
            user.setPhone1(result.getString("phone1"));
            user.setPhone2(result.getString("phone2"));
            user.setInsuranceCompany(result.getString("insuranceCompany"));
            user.setInsuranceAgentName(result.getString("insuranceAgentName"));
            user.setInsurancePhone1(result.getString("insurancePhone1"));
            user.setInsurancePhone2(result.getString("insurancePhone2"));
            user.setInsuranceNumber(result.getString("insuranceNumber"));
        } catch (SQLException e) {
            ResourceError err = ResourceError.getInstance();
            err.setMessage("internal error");
            err.setStatusCode(500);
            err.setReasonCode(ResourceError.REASON_UNKNOWN);
            return null;
        }
        return user;
    }

    /**Handle a user object, created so we won't have to use static methods
     * Return:
     *      @return User
     * */
    static public UserDao getInstance() {
        if (null == instance) {
            instance = new UserDao();
        }
        return instance;
    }

}
```

```java
/* src/dao/IncidentDao.java */
package dao;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Date;
import db.MySql;
import entities.Incident;
import entities.Person;
import entities.ResourceError;
import entities.IncidentPage;

public class IncidentDao {

    static private IncidentDao instance = null;
    private Connection connection;

    protected IncidentDao() {
        connection = MySql.getInstance().getConnection();
    }

    /**Return a Incident object given an id
     * Parameters:
     *      @param incidentId integer
     *      @param userId integer
     * Returns:
     *      @return Incident
     * Throws:
     *      @exception SQLException
     * */
    public Incident getById(int incidentId, int userId) {
        Incident incident = new Incident();
        ResourceError err = ResourceError.getInstance();
        try {
            CallableStatement callable = connection.prepareCall("call sp_getIncidentById(?,
            ?)");
            callable.setInt(1, incidentId);
            callable.setInt(2, userId);
            ResultSet result = callable.executeQuery();
            if (!result.first()) {
                return null;
            }

            incident.setIncidentId(result.getInt("incidentId"));
            incident.setUserId(result.getInt("userId"));
            incident.setDate(new Date(result.getTimestamp("date").getTime()));
            incident.setNotes(result.getString("notes"));
            incident.setVehicleLicensePlate(result.getString("vehicleLicensePlate"));
            incident.setVehicleBrand(result.getString("vehicleBrand"));
            incident.setVehicleModel(result.getString("vehicleModel"));

            int driverId = result.getInt("driverPersonId");
            int ownerId = result.getInt("ownerPersonId");

            PersonDao personDao = PersonDao.getInstance();
            if (0 < driverId) {
                Person driver = personDao.getById(driverId);
                if (null != driver) {
                    incident.setDriver(driver);
                }
            }

            if (0 < ownerId) {
                Person owner = personDao.getById(ownerId);
                if (null != owner) {
                    incident.setOwner(owner);
                }
            }

            return incident;
        }
        catch (SQLException e) {
            //e.printStackTrace();
            err.setMessage("internal error");
            err.setStatusCode(500);
            err.setReasonCode(ResourceError.REASON_UNKNOWN);
            return null;
        }
    }

    /**Insert a Incident object to the DB
     * Parameters:
     *      @param incident Incident
     * Returns:
     *      @return boolean
     * Throws:
     *      @exception SQLException
     * */
    public boolean save(Incident incident) {
        // insert or update the incident to the db
        ResourceError err = ResourceError.getInstance();
        try {
            CallableStatement callable;
            int i = 0;
            boolean updateMode = false;
            PersonDao personDao = PersonDao.getInstance();
            Person driver = incident.getDriver();
            Person owner = incident.getOwner();

            if (driver != null) {
                personDao.save(driver);
                if (err.isSet()) {
                    return false;
                }

            }

            if (owner != null) {
                personDao.save(owner);
                if (err.isSet()) {
                    return false;
                }
            }

            if (0 < incident.getIncidentId()) {
                updateMode = true;
                callable = connection.prepareCall("call
                sp_updateIncident(?,?,?,?,?,?,?,?,?,?,?)");
                callable.setInt(++i,incident.getIncidentId());
            }
            else {
                callable = connection.prepareCall("call
                sp_createIncident(?,?,?,?,?,?,?,?,?,?)");
            }

            callable.setInt(++i,incident.getUserId());
            callable.setTimestamp(++i, new java.sql.Timestamp(incident.getDate().getTime()));
            callable.setString(++i,incident.getNotes());
            callable.setString(++i,incident.getLocation());
            callable.setString(++i,incident.getVehicleLicensePlate());
            callable.setString(++i,incident.getVehicleBrand());
            callable.setString(++i,incident.getVehicleModel());
            callable.setInt(++i, null == driver ? 0 : driver.getPersonId());
            callable.setInt(++i, null == owner ? 0 : owner.getPersonId());

            callable.registerOutParameter(++i, java.sql.Types.INTEGER); // register
            incidentIdOut or rowsUpdatedOut

            callable.execute();

            if (updateMode) { // update mode
                int rowsUpdatedOut = callable.getInt("rowsUpdatedOut");
                if (1 != rowsUpdatedOut) {
                    if (!err.isSet()) {
                        err.setMessage("incident update failed, invalid input");
                        err.setStatusCode(400);
                        err.setReasonCode(ResourceError.REASON_INVALID_INPUT);
                    }
                    return false;
                }
            }
            else { // insert mode
                int incidentId = callable.getInt("incidentIdOut");
                incident.setIncidentId(incidentId);
            }
            return true;
        }
        catch (SQLException e) {
            if (!err.isSet()) {
                err.setMessage("internal error");
                err.setStatusCode(500);
                err.setReasonCode(ResourceError.REASON_UNKNOWN);
            }
            return false;
        }
    }

    /**Return an Incident page from the DB
     * Parameters:
     *      @param userId integer
     *      @param authHash String
     *      @param linesInPage short
     *      @param page short
     * Returns:
     *      @return IncidentPage
```

```java
     * Throws:
     *      @exception SQLException
     * */
    public IncidentPage getIncidentHistory(int userId, int page, int linesInPage){
        IncidentPage result = new IncidentPage();
        Incident[] incidentsTmp = new Incident[linesInPage];
        Incident[] incidents;
        short totalLines;
        CallableStatement callable;
        ResultSet rs;
        int counter = 0;
        Incident incident;
        PersonDao personDao = PersonDao.getInstance();
        int driverId;
        int ownerId;
        Person driver;
        Person owner;
        ResourceError err = ResourceError.getInstance();

        try {
            callable = connection.prepareCall("call sp_getIncidentHistory(?,?,?,?)");

            callable.setInt(1,userId);
            callable.setInt(2,page);
            callable.setInt(3,linesInPage);
            callable.registerOutParameter(4, java.sql.Types.SMALLINT);

            rs=callable.executeQuery();
            totalLines = callable.getShort("numberOfLines");

            while(rs.next()) {
                incident = new Incident();
                incidentsTmp[counter] = incident;
                incident.setIncidentId(rs.getInt("incidentId"));
                incident.setUserId(rs.getInt("userId"));
                incident.setDate(new Date(rs.getTimestamp("date").getTime()));
                incident.setNotes(rs.getString("notes"));
                incident.setLocation(rs.getString("location"));
                incident.setVehicleLicensePlate(rs.getString("vehicleLicensePlate"));
                incident.setVehicleBrand(rs.getString("vehicleBrand"));
                incident.setVehicleModel(rs.getString("vehicleModel"));


                driverId = rs.getInt("driverPersonId");
                ownerId = rs.getInt("ownerPersonId");


                if (0 < driverId) {
                    driver = personDao.getById(driverId);
                    if (null != driver) {
                        incident.setDriver(driver);
                    }
                }

                if (0 < ownerId) {
                    owner = personDao.getById(ownerId);
                    if (null != owner) {
                        incident.setOwner(owner);
                    }
                }

                counter++;
            }
            incidents = new Incident[counter];
            int i = 0;
            for (i=0;i<counter;i++) {
                incidents[i]=incidentsTmp[i];
            }
            result = new IncidentPage(incidents,totalLines);

        }
        catch (SQLException e) {
            if (!err.isSet()) {
                err.setMessage("internal error");
                err.setStatusCode(500);
                err.setReasonCode(ResourceError.REASON_UNKNOWN);

            }
            return null;
        }

        return result;
    }

    /**Handle a instance object, created so we won't have to use static methods
     * Return:
     *      @return User
     * */
    static public IncidentDao getInstance() {
        if (null == instance) {
            instance = new IncidentDao();
        }
        return instance;
    }

}
```

```java
/* src/resources/UserResource.java */
package resources;

import java.net.URI;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.ws.rs.Consumes;
import javax.ws.rs.DefaultValue;
import javax.ws.rs.FormParam;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.UriInfo;
import dao.UserDao;
import entities.ResourceError;
import entities.User;

/**Handles the User entity, retrieve and save a user*/
@Path("/user")
public class UserResource {

    /**Authenticate a user password
     * Parameters:
     *      @param username String
     *      @param password String
     * Returns:
     *      @return user
     *  Throws:
     *      @exception ResourceException*/
    @GET
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
    public User authenticate(@QueryParam("username") String username,
            @QueryParam("password") String password) {
        User user = UserDao.getInstance().authenticate(username, password);
        if (null == user) {
            ResourceError err = ResourceError.getInstance();
            if (!err.isSet()) {
                err.setMessage("unable to get user");
                err.setStatusCode(500);
                err.setReasonCode(ResourceError.REASON_UNKNOWN);
            }
            throw new ResourceException(err);
        }
        return user;
    }

    /**Return a user given an Id, verify with local hash
     * Parameters:
     *      @param userId integer
     *      @param authHash String
     * Returns:
     *      @return user
     * Throws:
     *      @exception ResourceException*/
```

```java
@GET
@Path("{id}")
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public User getById(@PathParam("id") int id, @QueryParam("authHash") String authHash) {
    UserDao userDao = UserDao.getInstance();
    if (!userDao.isAuthorized(id, authHash)){
        ResourceError err = ResourceError.getInstance();
        err.setMessage("unauthorized reason:mismatch userId, authHash");
        err.setStatusCode(400);
        err.setReasonCode(ResourceError.REASON_AUTHENTICATION_FAILED);
        throw new ResourceException(err);
    }
    User user = userDao.getById(id);
    if (null == user) {
        ResourceError err = ResourceError.getInstance();
        if (!err.isSet()) {
            err.setMessage("unable to get user");
            err.setStatusCode(500);
            err.setReasonCode(ResourceError.REASON_UNKNOWN);
        }
        throw new ResourceException(err);
    }
    user.setAuthHash(authHash);
    return user;
}

/**Inserts a user data to the DB
 * Parameters:
 *      @param UriInfo uriInfo
 *      @param HttpServletRequest servletRequest
 *      @param HttpServletResponse servletResponse
 *      @param personeId String, default value -1
 *      @param userid String, default -1
 *      @param email String
 *      @param firstName String
 *      @param lastName String
 *      @param idNumber String, "teudat zehut"
 *      @param city String
 *      @param streetName String
 *      @param houseNumber String
 *      @param addressDetails String, additional information, like if you have more then
 one entrance to the building
 *      @param zipcode String
 *      @param phone1 String
 *      @param phone2 String
 *      @param insuranceCompany String, insurance company name
 *      @param insuranceAgentName String
 *      @param insurancePhone1 String
 *      @param insurancePhone2 String
 *      @param insuranceNumber String
 *      @param username String
 *      @param password String
 *      @param authHash String
 * Throws:
 *      @exception ResourceException
 **/
@POST
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
```

```
public void save(
        @Context UriInfo uriInfo,
        @Context HttpServletRequest servletRequest,
        @Context HttpServletResponse servletResponse,
        @FormParam("personId") @DefaultValue("-1") String personId,  // validate integer
        @FormParam("userId") @DefaultValue("-1") String userId,  // validate integer
        @FormParam("email") @DefaultValue("") String email ,
        @FormParam("firstName") @DefaultValue("") String firstName,
        @FormParam("lastName") @DefaultValue("") String lastName,
        @FormParam("idNumber") @DefaultValue("") String idNumber,
        @FormParam("city") @DefaultValue("") String city,
        @FormParam("streetName") @DefaultValue("") String streetName,
        @FormParam("houseNumber") @DefaultValue("-1") String houseNumber,  // validate
        integer
        @FormParam("addressDetails") @DefaultValue("") String addressDetails,
        @FormParam("zipcode") @DefaultValue("-1") String zipcode, // validate integer
        @FormParam("phone1") @DefaultValue("") String phone1,
        @FormParam("phone2") @DefaultValue("") String phone2,
        @FormParam("insuranceCompany") @DefaultValue("") String insuranceCompany,
        @FormParam("insuranceAgentName") @DefaultValue("") String insuranceAgentName,
        @FormParam("insurancePhone1") @DefaultValue("") String insurancePhone1,
        @FormParam("insurancePhone2") @DefaultValue("") String insurancePhone2,
        @FormParam("insuranceNumber") @DefaultValue("") String insuranceNumber,
        @FormParam("username") @DefaultValue("") String username,
        @FormParam("password") @DefaultValue("") String password,
        @FormParam("authHash") @DefaultValue("") String authHash
        ) throws Exception {

    // validate input:
    int personIdOk, userIdOk, houseNumberOk, zipcodeOk;
    try {
        personIdOk = Integer.parseInt(personId);
        userIdOk = Integer.parseInt(userId);
        houseNumberOk = Integer.parseInt(houseNumber);
        zipcodeOk = Integer.parseInt(zipcode);
    }
    catch(Exception e) {

        ResourceError err = ResourceError.getInstance();
        if (!err.isSet()) {
            err.setMessage("user not saved, one of the following is not an int
            (personId, userId, houseNumber, zipcode)");
            err.setStatusCode(400);
            err.setReasonCode(ResourceError.REASON_INVALID_INPUT);
        }
        throw new ResourceException(err);
    }

    UserDao userDao = UserDao.getInstance();
    User user = new User();

    if (personIdOk <= 0 || userIdOk <= 0) { // if we dont't know the userId or personId
    it means we intend to create a new user
        if (username == "" || password == "") { // therefor verify that username and
        password are not empty
            ResourceError err = ResourceError.getInstance();
            err.setMessage("user not saved, username or password is missing)");
            err.setStatusCode(400);
            err.setReasonCode(ResourceError.REASON_INVALID_INPUT);
            throw new ResourceException(err);
        }
    }
    else  { // check if authorized to make a change to the user
        if (!userDao.isAuthorized(userIdOk, authHash)) {
            ResourceError err = ResourceError.getInstance();
            if (!err.isSet()) {
                err.setMessage("user not save, reason:authorization failed");
                err.setStatusCode(400);
                err.setReasonCode(ResourceError.REASON_AUTHENTICATION_FAILED);
            }
            throw new ResourceException(err);
        }
        user.setPersonId(personIdOk);
        user.setUserId(userIdOk);
        user.setAuthHash(authHash);
    }

    user.setEmail((String)email);
    user.setFirstName(firstName);
    user.setLastName(lastName);
    user.setIdNumber(idNumber); //TODO handle length
    user.setCity(city);
    user.setStreetName(streetName);
    //TODO handle better integers, it crushes the application if non integer string is
    passed here
    user.setHouseNumber(houseNumberOk);
    user.setAddressDetails(addressDetails);
    //TODO handle better integers, it crushes the application if non integer string is
    passed here
    user.setZipcode(zipcodeOk);
    user.setPhone1(phone1);
    user.setPhone2(phone2);
    user.setInsuranceCompany(insuranceCompany);
    user.setInsuranceAgentName(insuranceAgentName);
    user.setInsurancePhone1(insurancePhone1);
    user.setInsurancePhone2(insurancePhone2);
    user.setInsuranceNumber(insuranceNumber);
    user.setUsername(username);
    user.setPassword(password); //TODO hash the password before persisting to the
    database

    if (!userDao.save(user) || 0 >= user.getUserId() || "" == user.getAuthHash()) {
        ResourceError err = ResourceError.getInstance();
        if (!err.isSet()) {
            err.setMessage("user not saved");
            err.setStatusCode(500);
            err.setReasonCode(ResourceError.REASON_UNKNOWN);
        }
        throw new ResourceException(err);
    }

    URI userUri = uriInfo.getBaseUriBuilder().path(UserResource.class).path("/" + user.
    getUserId()).queryParam("authHash", user.getAuthHash()).build();
    servletResponse.sendRedirect(userUri.toString());
  }
}
```

```java
/* src/resources/IncidentResource.java */
package resources;

import java.net.URI;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.ws.rs.Consumes;
import javax.ws.rs.DefaultValue;
import javax.ws.rs.FormParam;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.UriInfo;
import dao.IncidentDao;
import dao.UserDao;
import entities.Incident;
import entities.IncidentPage;
import entities.Person;
import entities.ResourceError;

/**Handles the Incident entity, retrieve and save an incident*/
@Path("/incident")
public class IncidentResource {


    /**Retrieve an incident data from the DB by id
     * Parameters:
     *      @param incidentId String
     *      @param userId String
     *      @param authHash String
     * Returns:
     *      @return Incident*/
    @GET
    @Path("{id}")
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
    public Incident getById(
            @PathParam("id") @DefaultValue("-1") String incidentId
            , @QueryParam("userId") @DefaultValue("-1") String userId
            , @QueryParam("authHash") @DefaultValue("") String authHash) {
        //return new Incident();
        IncidentDao incidentDao = IncidentDao.getInstance();
        UserDao userDao = UserDao.getInstance();
        ResourceError err = ResourceError.getInstance();
        int userIdOk, incidentIdOk;

        // validate input:
        try {
            userIdOk = Integer.parseInt(userId);
            incidentIdOk = Integer.parseInt(incidentId);
        }
        catch(Exception e) {
            if (!err.isSet()) {
                err.setMessage("invalid input, one of the following is not an int (userId,
                incidentId)");//, driverPersonId, ownerPersonId)");
                err.setStatusCode(400);
                err.setReasonCode(ResourceError.REASON_INVALID_INPUT);
            }
            throw new ResourceException(err);
        }

        if (!userDao.isAuthorized(userIdOk, authHash)){
            err.setMessage("unauthorized reason:mismatch userId, authHash");
            err.setStatusCode(400);
            err.setReasonCode(ResourceError.REASON_AUTHENTICATION_FAILED);
            throw new ResourceException(err);
        }

        Incident incident = incidentDao.getById(incidentIdOk, userIdOk);
        if (null == incident) {
            if (!err.isSet()) {
                err.setMessage("unable to get incident");
                err.setStatusCode(500);
                err.setReasonCode(ResourceError.REASON_UNKNOWN);
            }
            throw new ResourceException(err);
        }

        return incident;
    }


    @GET
    @Path("history")
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
    public IncidentPage getHistoryByUserId(@QueryParam("userId") @DefaultValue("-1") String
    userId,
            @QueryParam("page") @DefaultValue("1") String page,
            @QueryParam("linesInPage") @DefaultValue("20") String linesInPage,
            @QueryParam("authHash") @DefaultValue("") String authHash) {
        //return new Incident();
        IncidentDao incidentDao = IncidentDao.getInstance();
        UserDao userDao = UserDao.getInstance();
        ResourceError err = ResourceError.getInstance();
        int userIdOk, pageOk, linesInPageOk;

        // validate input:
        try {
            userIdOk = Integer.parseInt(userId);
            pageOk = Integer.parseInt(page);
            linesInPageOk = Integer.parseInt(linesInPage);
        }
        catch(Exception e) {
            if (!err.isSet()) {
                err.setMessage("invalid input, one of the following is not an int (userId,
                page, linesInPage)");
                err.setStatusCode(400);
                err.setReasonCode(ResourceError.REASON_INVALID_INPUT);
            }
            throw new ResourceException(err);
        }
        if (!userDao.isAuthorized(userIdOk, authHash)){
            err.setMessage("unauthorized reason:mismatch userId, authHash");
            err.setStatusCode(400);
            err.setReasonCode(ResourceError.REASON_AUTHENTICATION_FAILED);
            throw new ResourceException(err);
        }

        IncidentPage IncidentPage = incidentDao.getIncidentHistory(userIdOk, pageOk,
        linesInPageOk);
        if (null == IncidentPage) {
            if (!err.isSet()) {
                err.setMessage("unable to get incidents history");
                err.setStatusCode(500);
                err.setReasonCode(ResourceError.REASON_UNKNOWN);
            }
            throw new ResourceException(err);
        }

        return IncidentPage;
    }

    /**Inserts an incident data to the DB
     * Parameters:
     *      @param UriInfo uriInfo
     *      @param HttpServletRequest servletRequest
     *      @param HttpServletResponse servletResponse
     *      @param incidentId String
     *      @param userId String
     *      @param authHash String
     *      @param date String
     *      @param notes String
     *      @param location String
     *      @param vehicleLicensePlate String
     *      @param vehicleBrand String
     *      @param vehicleModel String
     *      @param driverIdNumber String
     *      @param driverFirstName String
     *      @param driverLastName String
     *      @param driverPhone1 String
     *      @param driverPhone2 String
     *      @param driverInsuranceCompany String
     *      @param driverInsuranceAgentName String
     *      @param driverInsurancePhone1 String
     *      @param driverInsurancePhone2 String
     *      @param driverInsuranceNumber String
     *      @param driverEmail String
     *      @param ownerIdNumber String
     *      @param ownerFirstName String
     *      @param ownerLastName String
     *      @param ownerPhone1 String
     *      @param ownerPhone2 String
     *      @param ownerInsuranceCompany String
     *      @param ownerInsuranceAgentName String
     *      @param ownerInsurancePhone1 String
     *      @param ownerInsurancePhone2 String
     *      @param ownerInsuranceNumber String
     *      @param ownerEmail String
     * Throws:
```

```
 *      @exception ResourceException
 **/
@POST
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
public void save(
        @Context UriInfo uriInfo,
        @Context HttpServletRequest servletRequest,
        @Context HttpServletResponse servletResponse,


        @FormParam("incidentId") @DefaultValue("-1") String incidentId,

        // authorization data
        @FormParam("userId") @DefaultValue("-1") String userId,
        @FormParam("authHash") @DefaultValue("") String authHash,

        // incident data
        @FormParam("date") @DefaultValue("") String date,
        @FormParam("notes") @DefaultValue("") String notes,
        @FormParam("location") @DefaultValue("") String location,
        @FormParam("vehicleLicensePlate") @DefaultValue("") String vehicleLicensePlate,
        @FormParam("vehicleBrand") @DefaultValue("") String vehicleBrand,
        @FormParam("vehicleModel") @DefaultValue("") String vehicleModel,

        // car's driver data
        @FormParam("driverIdNumber") @DefaultValue("") String driverIdNumber,
        @FormParam("driverFirstName") @DefaultValue("") String driverFirstName,
        @FormParam("driverLastName") @DefaultValue("") String driverLastName,
        @FormParam("driverPhone1") @DefaultValue("") String driverPhone1,
        @FormParam("driverPhone2") @DefaultValue("") String driverPhone2,
        @FormParam("driverInsuranceCompany") @DefaultValue("") String
        driverInsuranceCompany,
        @FormParam("driverInsuranceAgentName") @DefaultValue("") String
        driverInsuranceAgentName,
        @FormParam("driverInsurancePhone1") @DefaultValue("") String
        driverInsurancePhone1,
        @FormParam("driverInsurancePhone2") @DefaultValue("") String
        driverInsurancePhone2,
        @FormParam("driverInsuranceNumber") @DefaultValue("") String
        driverInsuranceNumber,
        @FormParam("driverEmail") @DefaultValue("") String driverEmail,

        // car's owner data
        @FormParam("ownerIdNumber") @DefaultValue("") String ownerIdNumber,
        @FormParam("ownerFirstName") @DefaultValue("") String ownerFirstName,
        @FormParam("ownerLastName") @DefaultValue("") String ownerLastName,
        @FormParam("ownerPhone1") @DefaultValue("") String ownerPhone1,
        @FormParam("ownerPhone2") @DefaultValue("") String ownerPhone2,
        @FormParam("ownerInsuranceCompany") @DefaultValue("") String
        ownerInsuranceCompany,
        @FormParam("ownerInsuranceAgentName") @DefaultValue("") String
        ownerInsuranceAgentName,
        @FormParam("ownerInsurancePhone1") @DefaultValue("") String ownerInsurancePhone1,
        @FormParam("ownerInsurancePhone2") @DefaultValue("") String ownerInsurancePhone2,
        @FormParam("ownerInsuranceNumber") @DefaultValue("") String ownerInsuranceNumber,
        @FormParam("ownerEmail") @DefaultValue("") String ownerEmail

        ) throws Exception {
```

```java
// declarations
IncidentDao incidentDao = IncidentDao.getInstance();
UserDao userDao = UserDao.getInstance();
ResourceError err = ResourceError.getInstance();
Person driver = new Person();
Person owner = new Person();
Incident incident = new Incident();
int userIdOk, incidentIdOk;

// validate input:
try {
    userIdOk = Integer.parseInt(userId);
    incidentIdOk = Integer.parseInt(incidentId);
}
catch(Exception e) {
    if (!err.isSet()) {
        err.setMessage("incident not saved, one of the following is not an int
        (userId, incidentId)");
        err.setStatusCode(400);
        err.setReasonCode(ResourceError.REASON_INVALID_INPUT);
    }
    throw new ResourceException(err);
}


// check if user is authorized to save an incident
if (!userDao.isAuthorized(userIdOk, authHash)) {
    if (!err.isSet()) {
        err.setMessage("incident not saved, reason:authorization failed");
        err.setStatusCode(400);
        err.setReasonCode(ResourceError.REASON_AUTHENTICATION_FAILED);
    }
    throw new ResourceException(err);
}

if (incidentIdOk > 0) {
    incident = incidentDao.getById(incidentIdOk, userIdOk);

    if (null == incident) {
        if (!err.isSet()) {
            err.setMessage("unable to save incident");
            err.setStatusCode(500);
            err.setReasonCode(ResourceError.REASON_UNKNOWN);
        }
        throw new ResourceException(err);
    }


    Person driverTmp = incident.getDriver();
    Person ownerTmp = incident.getOwner();

    if (null != driverTmp) {
        driver = driverTmp;
    }
    if (null != ownerTmp) {
        owner = ownerTmp;
    }
}
```

```java
else {
    incident.setUserId(userIdOk);
}

SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm");
try {
    incident.setDate(df.parse(date));
}
catch (ParseException e) {
    incident.setDate(new Date());
}

incident.setLocation(location);
incident.setNotes(notes);
incident.setVehicleLicensePlate(vehicleLicensePlate);
incident.setVehicleBrand(vehicleBrand);
incident.setVehicleModel(vehicleModel);

if (driver.getPersonId() > 0
        || !driverFirstName.equals("")
        || !driverLastName.equals("")
        || !driverIdNumber.equals("")
        || !driverPhone1.equals("")
        || !driverPhone2.equals("")
        || !driverInsuranceCompany.equals("")
        || !driverInsuranceAgentName.equals("")
        || !driverInsurancePhone1.equals("")
        || !driverInsurancePhone2.equals("")
        || !driverInsuranceNumber.equals("")
        || !driverEmail.equals("")
    ) {
    driver.setFirstName(driverFirstName);
    driver.setLastName(driverLastName);
    driver.setIdNumber(driverIdNumber);
    driver.setPhone1(driverPhone1);
    driver.setPhone2(driverPhone2);
    driver.setInsuranceCompany(driverInsuranceCompany);
    driver.setInsuranceAgentName(driverInsuranceAgentName);
    driver.setInsurancePhone1(driverInsurancePhone1);
    driver.setInsurancePhone2(driverInsurancePhone2);
    driver.setInsuranceNumber(driverInsuranceNumber);
    driver.setEmail(driverEmail);

    incident.setDriver(driver);
}

if (owner.getPersonId() > 0
        || !ownerFirstName.equals("")
        || !ownerLastName.equals("")
        || !ownerIdNumber.equals("")
        || !ownerPhone1.equals("")
        || !ownerPhone2.equals("")
        || !ownerInsuranceCompany.equals("")
        || !ownerInsuranceAgentName.equals("")
        || !ownerInsurancePhone1.equals("")
        || !ownerInsurancePhone2.equals("")
        || !ownerInsuranceNumber.equals("")
        || !ownerEmail.equals("")
        ) {
            owner.setFirstName(ownerFirstName);
            owner.setLastName(ownerLastName);
            owner.setIdNumber(ownerIdNumber);
            owner.setPhone1(ownerPhone1);
            owner.setPhone2(ownerPhone2);
            owner.setInsuranceCompany(ownerInsuranceCompany);
            owner.setInsuranceAgentName(ownerInsuranceAgentName);
            owner.setInsurancePhone1(ownerInsurancePhone1);
            owner.setInsurancePhone2(ownerInsurancePhone2);
            owner.setInsuranceNumber(ownerInsuranceNumber);
            owner.setEmail(ownerEmail);

            incident.setOwner(owner);
        }


        if (!incidentDao.save(incident) || 0 >= incident.getIncidentId()) {
            if (!err.isSet()) {
                err.setMessage("incident not saved");
                err.setStatusCode(500);
                err.setReasonCode(ResourceError.REASON_UNKNOWN);
            }
            throw new ResourceException(err);
        }

        URI uri = uriInfo.getBaseUriBuilder().path(IncidentResource.class).path("/" +
        incident.getIncidentId()).queryParam("userId", userId).queryParam("authHash",
        authHash).build();
        servletResponse.sendRedirect(uri.toString());
    }
}
```

```java
/* src/entities/ResourceError.java */
package entities;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class ResourceError {

    static final public int REASON_UNKNOWN = -1;
    static final public int REASON_INVALID_INPUT = 1;
    static final public int REASON_AUTHENTICATION_FAILED = 2;

    static private ResourceError instance;
    private int statusCode;
    private int reasonCode;
    private String message;

    public ResourceError() {
        //this constructor is needed so jaxb doesn't throw an exception related to no-arg
        constructor
    }

    public int getStatusCode() {
        return statusCode;
    }

    public void setStatusCode(int statusCode) {
        this.statusCode = statusCode;
    }

    public int getReasonCode() {
        return reasonCode;
    }

    public void setReasonCode(int reasonCode) {
        this.reasonCode = reasonCode;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public boolean isSet() {
        return getStatusCode() != 0;
    }

    static public ResourceError getInstance() {
        if (null == instance) {
            instance = new ResourceError();
        }
        return instance;
    }
}
```

```java
/* src/resources/ResourceException.java */
package resources;

import entities.ResourceError;

/**Handles the project exceptions*/
public class ResourceException extends RuntimeException {

    private static final long serialVersionUID = 3222518051740261790L;

    private ResourceError error;
    private int statusCode;

    /**ResourceException constructor
     * Parameters:
     *      @param error ResourceError*/
    public ResourceException(ResourceError error) {
        super(error.getMessage());
        this.error = error;
        this.statusCode = error.getStatusCode();
    }

    /**An Error getter
     * Returns:
     *      @return ResourceError*/
    public ResourceError getError() {
        return error;
    }

    /**An Error setter
     * Parameters:
     *      @param error ResourceError*/
    public void setError(ResourceError error) {
        this.error = error;
    }

    /**A StatusCode getter
     * Returns:
     *      @return integer*/
    public int getStatusCode() {
        return statusCode;
    }

    /**A StatusCode setter
     * Parameters:
     *      @param statusCode integer*/
    public void setStatusCode(int statusCode) {
        this.statusCode = statusCode;
    }
}
```

```java
/* src/resources/ResourceExceptionMapper.java */
package resources;

import javax.ws.rs.core.Context;
import javax.ws.rs.core.HttpHeaders;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.ResponseBuilder;
import javax.ws.rs.ext.ExceptionMapper;
import javax.ws.rs.ext.Provider;
import java.util.List;

/**Handles the project http errors*/
@Provider
public class ResourceExceptionMapper implements ExceptionMapper<ResourceException> {

    @Context
    private HttpHeaders headers;

    /**Get a caught error and build a response
     * Parameters:
     *      @param e ResourceException
     * Returns:
     *      @return ResponseBuilder*/
    public Response toResponse(ResourceException e) {
        ResponseBuilder rb = Response.status(e.getStatusCode()).entity(e.getError());

        List<MediaType> accepts = headers.getAcceptableMediaTypes();
        if (accepts!=null && accepts.size() > 0) {
            //just pick the first one
            MediaType m = accepts.get(0);
            rb = rb.type(m);
        }
        else {
            //if not specified, use the entity type
            rb = rb.type(headers.getMediaType()); // set the response type to the entity type.
        }

        return rb.build();
    }
}
```

```java
/* src/app/Config.java */
package app;

import java.util.Properties;

public class Config {

    static private Config instance = null;
    private Properties config;
    private String appName;
    private String dbUser;
    private String dbPassword;
    private String dbHost;
    private String dbSchema;

    public Config() {
        try {
            config = new Properties();
            config.load(Thread.currentThread().getContextClassLoader().getResourceAsStream(
            "/config.properties"));

            appName = config.getProperty("app.name");
            dbUser = config.getProperty("db.user");
            dbPassword = config.getProperty("db.password");
            dbHost = config.getProperty("db.host");
            dbSchema = config.getProperty("db.schema");

        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    /**
     * singleton access method
     * @return Properties
     */
    static public Config getInstance() {
        if (null == instance) {
            instance = new Config();
        }
        return instance;
    }

    public String getAppName() {
        return appName;
    }

    public String getDbUser() {
        return dbUser;
    }

    public String getDbPassword() {
        return dbPassword;
    }

    public String getDbHost() {
        return dbHost;
    }
    public String getDbSchema() {
        return dbSchema;
    }
}
```

```java
/* src/controllers/Beenbumped.java */
package controllers;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**Active the JSON for the main web page*/
public class Beenbumped extends HttpServlet {

    private static final long serialVersionUID = -4260962579726500851L;

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
        RequestDispatcher view = request.getRequestDispatcher("/beenbumped.jsp");
        view.forward(request, response);
    }


}
```