People Simple Preferences

RESTful API to manage personal info (name, age) and personal preferences (food, color).

Built using:

- Jersey
- Grizzly
- SQLite
- Hibernate
- Gson
- Maven
- JUnit
- JBehave
- Swagger.io

Install

```
mvn install
```

Details

Implemented date of birth instead of age, because age is dynamic value. But also "age" field added - it's read-only & auto-calculated from date of birth.

Implemented multiple preferences for color & food. Color preferences are sub-set of pre-defined colors so I used `enum`. Food preferences are user-generated so I've used a separate DB table & relation one-to-many.

Added constraint for unique first name + last name + date of birth. You can't create two persons with same fields.

Try it!

Run the web app (for dev/test only):

```
mvn clean test  # run just once
mvn exec:java
```

JSON or XML

By default, all results will be returned in JSON format. You can ask for it explicitly via `-H 'Accept: application/json'` (that's for `curl`). To get data in XML, use `-H 'Accept: application/xml'`.

To submit data, you should always provide `Content-Type` header (there are no defaults for it). To send data in JSON, use `-H 'Content-Type: application/json'`. To send data in XML, use `-H 'Content-Type: application/xml'`.

People list

Get people list:

```
curl -i http://localhost:8080/preferences/people
```

To see all possible RESTful options for `/preferences/people`, use `-X OPTIONS` (it'll show options in WADL format).

Search the people list:

```
curl -i http://localhost:8080/preferences/people/search?lastName=Ivanov
```

You can search by `firstName` and/or `lastName`.

Create new person:

```
curl -i -X POST -H 'Content-Type: application/json' -d '{"firstName": "John",
"lastName": "Smith", "dateOfBirth": "1985-12-18", "favoriteColor": ["yellow"],
"favoriteFood": [{"name": "chocolate"}]}' http://localhost:8080/preferences/people
```

You can send data in XML format too:

```
curl -i -X POST -H 'Content-Type: application/xml' -d
'<person><id>1</id><firstName>Petr</firstName><lastName>Petrov</lastName><dateOfBir
th>1985-12-18</dateOfBirth><favoriteFood><food><name>chocolate</name></food><food><
name>banana</name></food></favoriteFood></person>'
http://localhost:8080/preferences/people
```

If the person *John Smith (1985-12-18)* already exists, there will be `HTTP 409 Conflict` with error message in body. If the color is *unexpected* - there will be `HTTP 400 Bad Request` with error message in body. Otherwise, there will be `HTTP 201 Created` with empty body & `Location: link-to-created-person` header.

Person

Get person details:

```
curl -i http://localhost:8080/preferences/people/1
```

To see all possible RESTful options for `/preferences/people/1`, use `-X OPTIONS`.

Update person details:

```
curl -i -X PUT -H 'Content-Type: application/json' -d
'{"id":1,"firstName":"John","lastName":"Smith","dateOfBirth":"1988-12-23","favorite
Color":["yellow"],"favoriteFood":[{"name":"chocolate"}]}'
http://localhost:8080/preferences/people/1
```

There is the same validation for the person update as for its creation. If you'll try to update *not existing person* - it will return `HTTP 404 Not Found`.

Delete person:

```
curl -X DELETE http://localhost:8080/preferences/people/4
```

There will be `HTTP 204 No Content` with empty body in case of success.

Food

As I've mentioned, there is a separate entity for food preferences.

Get food list:

```
curl -i http://localhost:8080/preferences/foodlist
```

To see all possible RESTful options for `/preferences/foodlist`, use `-X OPTIONS`.

Search the food list:

```
curl -i http://localhost:8080/preferences/foodlist/search?name=ca
```

You can search by `name`.

Get food details:

```
curl -i http://localhost:8080/preferences/foodlist/1
```

To see all possible RESTful options for `/preferences/foodlist/1`, use `-X OPTIONS`.

Update food details:

```
curl -i -X PUT -H 'Content-Type: application/json' -d '{"name": "apple"}'
http://localhost:8080/preferences/foodlist/1
```

I've not implemented creation and deletion of food record because it's a part of person workflow.

Dev extras

Special test page created, served by dev server, to test the API in the browser:

```
http://localhost:8080/test/web/index.html
```

First you need to choose HTTP method, then - fill the form & send.

Tests

JUnit for unit tests and JBehave for application-level tests used. Run tests command:

```
mvn clean test integration-test
```

To view the results of JBehave tests go to "target/jbehave/view/reports.html".

Docs

Swagger.io used to build RESTful API documentation. Build the docs:

```
mvn compile
```

Look for the generated docs in "target/generated-resources/swagger.json".