Documentation

Beanframework

Thank you so much for using beanframework project. 100% open source project under MIT license.

Version: 3.0.0

Created: 2 May, 2021

Github: Beanframework

Update: 11 May, 2021

If you have any questions that are beyond the scope of this help file, Please feel free to post via Github Discussion Page .

License

```
MIT License

Copyright 2018-2021 Beanframework

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

Installation

Create a new database:

1. By default, this project supported unicode character such as Chinese language, therefore the database must use CHARACTER SET utf8 COLLATE utf8_unicode_ci

```
CREATE SCHEMA `beanframework` DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci
```

2. Configure project by copy and remove **.template** suffix:
   - `beanframework/bin/` pom.xml.template
   - `beanframework/bin/install/` server.bat.template
   - `beanframework/bin/install/` server.sh.template
   - `beanframework/bin/platform/` pom.xml.template
   - `beanframework/config/` pom.xml.template
   - `beanframework/config/src/main/resources/` *.template
   - `beanframework/config/src/main/resources/import/dev.template`

- Optional: `beanframework/bin/install/` app.xml.template

3. Open command prompt and navigate to `beanframework/bin` directory run:

```
mvnw clean install
```

4. Navigate to `beanframework/bin/install` directory and run:

```
server.bat
```

5. Access application endpoints:
   - Console: http://localhost:8080/console
   - Backoffice: http://localhost:8080/backoffice
   - Documentation: http://localhost:8080/documentation
6. Import sample data:
   - Access Console: http://localhost:8080/console
   - Login with default admin account: username: **admin** , password: **admin**
   - Goto menu `Platform->Update` , check all and update.
7. You are good to go for run your project now!

Project Structure

1. Below is the `beanframework` folder structure:
   - `bin` - Source files
     - `custom` - Create a custom modules and put in this folder, best practice for not mixing with original project structure and codes, and easily to upgrade software in future.
     - `install` - Install/Run application manually or install as windows service.
     - `modules` - Project default modules.
     - `platform` - Platform that startup this application with configured modules.
   - `config` - Mainly use for application properties and configurations for different environments.
   - `data` - Data storage for this application
     - `media` - All the media stored in this directory.
     - `integration` - Spring Integration. Can be used for import data.
   - `log` - Application logging files, with archived and rotated logging files.
   - `temp` - Application temporary files.

Platform

Platform Configuration and Import Data

Configuration

`beanframework/bin/config/src/main/resources/`

- Configure environment:
  - `application.properties`
    - **spring.profiles.active=dev**
- Configure configurations and import data for environment:

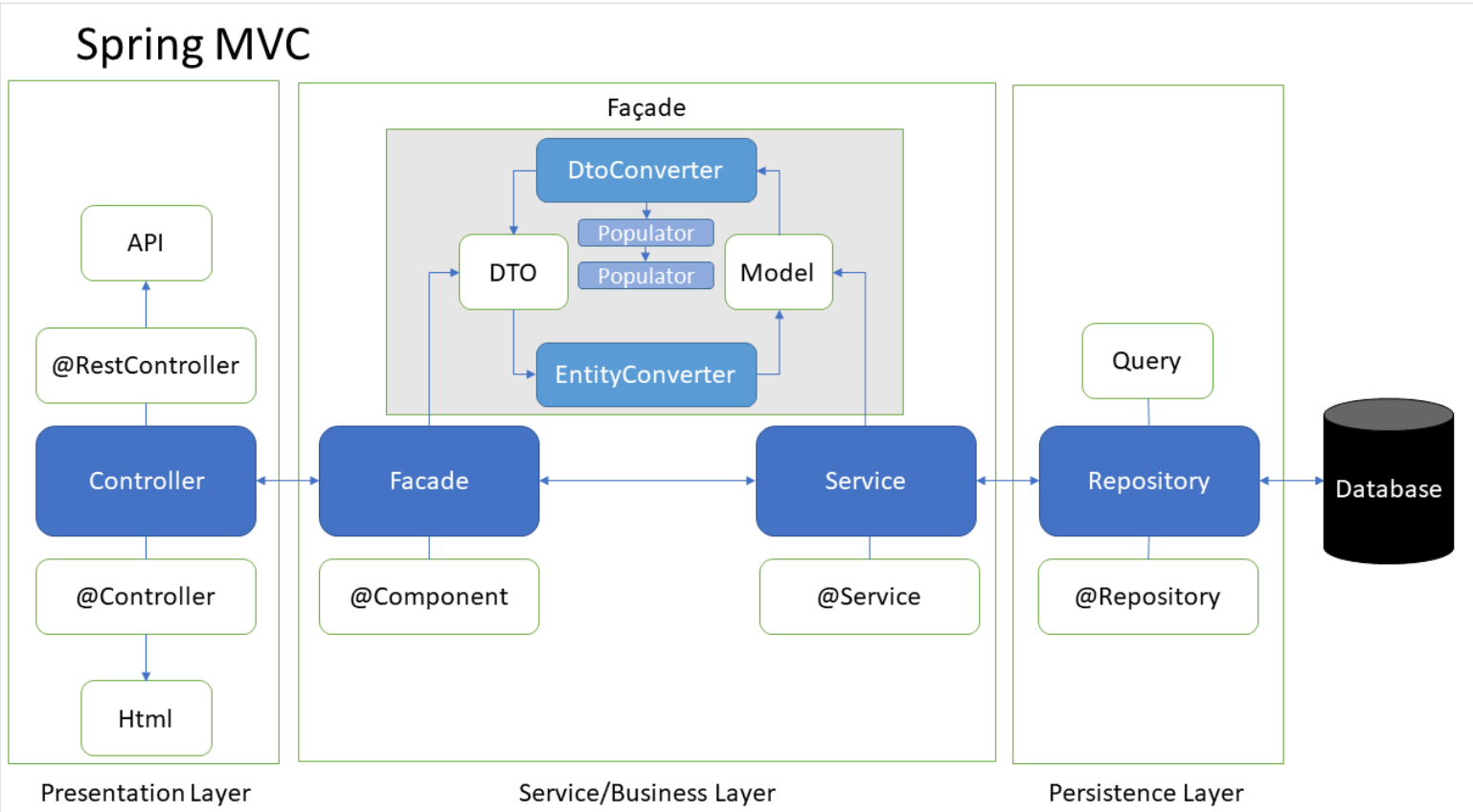
- `application-dev.properties` - Environment properties.
- `ehcache-dev.xml` - Cache configuration.
- `logback-dev.xml` - Logback configuration.
- `import/dev` - Contains data to import

Architecture

Architecture for development

Spring MVC

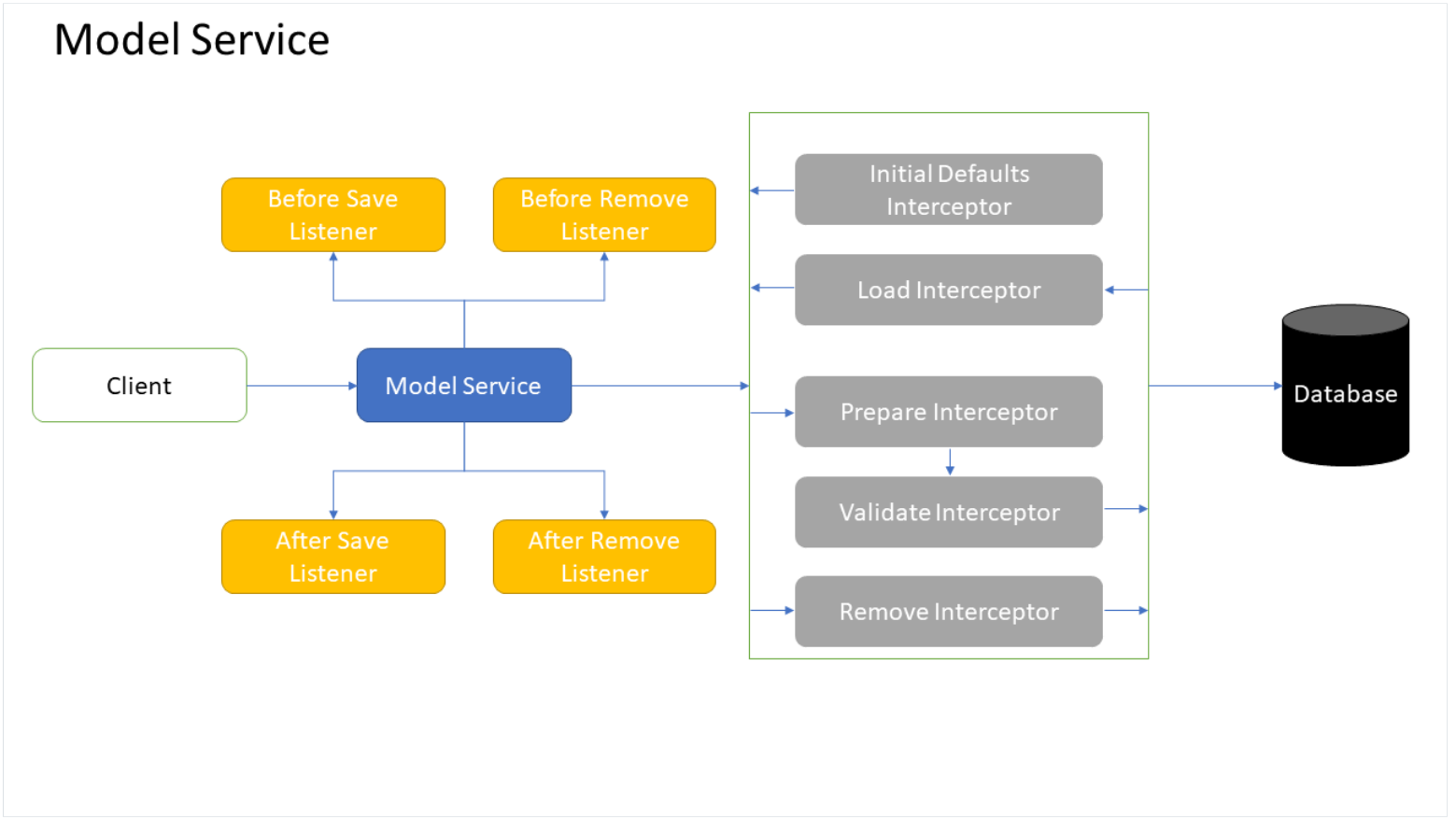
Design Pattern used in this platform.



- Presentation Layer
  - **Controller** :
    `@Controller` - Render view html
    `@RestController` - Response for web services
- Service/Business Layer
  - **Facade** : `@Component` - To mark the beans as Spring's managed components.
    - **DtoConverter** - Convert Model to DTO
      - **Populator** - Multiple populators to populate data from source Model to target DTO
    - **EntityConverter** - Convert DTO to Model
  - **Service** : `@Service` - To indicate that it's holding the business logic
- Persistence Layer
  - **Repository** : `@Repository` - To catch persistence specific exceptions and rethrow them as one of Spring's unified unchecked exception.

Model Service

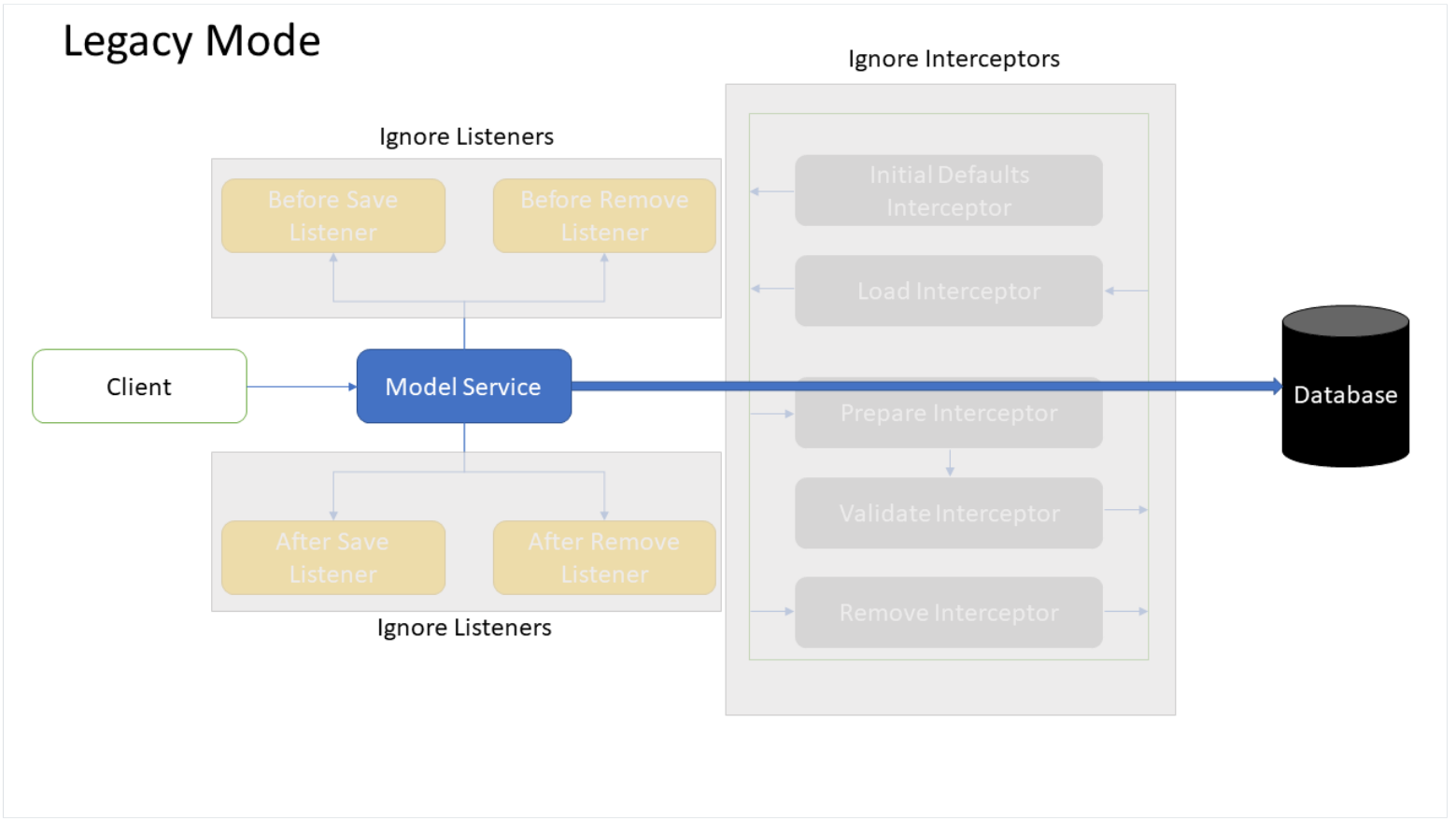
Model's life cycle involving listeners and interceptors.



- Initial a new entity object operation in model service:
  - `modelService.create(...)`
  - **Initial Defaults Interceptor** - To create an new object with defaults attributes.
- Find entity operation in model service:
  - `modelService.find...(...)`
  - **Database**
  - **Load Interceptor** - To modify an object loaded from database.
- Create/update entity operation in model service:
  - `modelService.saveEntity(...)`
  - **Prepare Interceptor** - To modify an object before save to database.
  - **Validate Interceptor** - To validate an object before save to database. (Do not modify attached model)
  - **Before Save Listener** - Perform business logic. (Do not modify attached model)
  - **Database**
  - **After Save Listener** - Perform business logic. (Do not modify attached model)
- Delete entity operation in model service:
  - `modelService.deleteEntity(...)`
  - **Remove Interceptor** - To modify an object before remove from database.
  - **Before Remove Listener** - Perform business logic. (Do not modify attached model)
  - **Database**
  - **After Remove Listener** - Perform business logic. (Do not modify attached model)

Legacy Mode

Performing model service with legacy mode ignoring listeners and interceptors.



Development

Development guide for Spring Boot developer

Module Gen

`beanframework/bin/module/modulegen/`

- Set module configuration `src/main/resources/application.properties`
- Run `src/main/java/com/beanframework/modulegen/ModulegenApplication.java`
- Generated module output: `beanframework/bin/module/custom/`

Configure new module in project:

- Add new module in `beanframework/bin/pom.xml`
- Add new dependency in `beanframework/config/pom.xml`
- Add new profile and packages in `beanframework/config/src/main/resources/application.properties`
  - **spring.profiles.include=platform,console,backoffice,documentation,training**
  - **spring.scanBasePackages=com.beanframework,com.sample**
  - **jpa.domain.packagetoscans=com.beanframework.*.domain,com.sample.*.domain**

Model Attribute

Example of adding new attribute in Model

- `Training.java` - Add new attribute
- `TrainingDto.java` - Add new attribute
- `TrainingPopulator.java` - Add new attribute
- `TrainingEntityConverter.java` - Add new attribute
- `trainingForm.html` - Add new input

```
...

<div class="col-md-3">
        <div class="form-group">
                <label for="name" th:text="#{module.training.name}"></label>
                <input type="text" class="form-control" name="name" th:value="*{name}">
        </div>
</div>

...
```

Datatable

Example of adding new column in datatable

- `training.html` - Add new attribute, for example "name":

```
...

<th:block th:insert="~{backoffice/adminlte/common/fragment/content-datatable :: datatable (title=#
{module.training}, tableId='trainingTable', columnsi18n=${#strings.arraySplit(id+','+name, ',')},
selectAuthority=true )}"></th:block>

...

<th:block th:insert="~{backoffice/adminlte/common/fragment/content-datatable :: js (tableId='trainingTable',
columns=${#strings.arraySplit('id,name', ',')}, pageUrl=@{${@environment.getProperty('path.api.training')}},
formUrl=@{${@environment.getProperty('path.training.form')}}, addAuthority=${addAuthority}, selectAuthority=true
)}"></th:block>


...
```

- `Training.java` - Add new attribute, for example "name":

```
...

public Training(UUID uuid, String id, String name) {
                super();
                setUuid(uuid);
                setId(id);
                setName(name);
        }
...
```

- `TrainingSpecification.java` - Add new attribute, for example "name":

```
...

@Override
public Predicate toPredicate(Root<T> root, CriteriaQuery<?>; query, CriteriaBuilder cb) {
                List<Predicate> predicates = new ArrayList<Predicate>();

                String search = clean(dataTableRequest.getSearch());

                if (StringUtils.isNotBlank(search)) {
                        predicates.add(cb.or(cb.like(root.get(TRAINING.ID), convertToLikePattern(search))));
                        predicates.add(cb.or(cb.like(root.get(TRAINING.NAME), convertToLikePattern(search))));
                }

                if (predicates.isEmpty()) {
                        return cb.and(predicates.toArray(new Predicate[predicates.size()]));
                } else {
                        return cb.or(predicates.toArray(new Predicate[predicates.size()]));
                }
}
```

```
...

@Override
public List<Selection<?>> toSelection(Root<T> root) {
                List<Selection<?>> multiselect = new ArrayList<Selection<?>>();
                multiselect.add(root.get(Training.UUID));
                multiselect.add(root.get(Training.ID));
                multiselect.add(root.get(Training.NAME));
                return multiselect;
}
...
```

- `TrainingDataTableResponseData.java` - Add new property, for example "name"
- `TrainingResource.java` - Add new data, for example "name":

```
...

for (TrainingDto dto : pagination.getContent()) {

                TrainingDataTableResponseData data = new TrainingDataTableResponseData();
                data.setUuid(dto.getUuid().toString());
                data.setId(StringUtils.stripToEmpty(dto.getId()));
                data.setName(StringUtils.stripToEmpty(dto.getName()));
                dataTableResponse.getData().add(data);
}
...
```

Dynamic Attributes

*Please check example of dynamic attributes used in UserGroup module in project.

Example of creating Dynamic Field for a Training Model:

- Create/Reuse a new entry in DynamicField:
  name_en
- Create a new entry in DynamicFieldSlot:
  training_name_en

- Create a new entry in DynamicFieldTemplate:
  training.dynamicfield.template

Example of creating Attributes for a Training model:

- `TrainingAttribute.java` - Create new entity
- `Training.java` - Insert attributes property
- `TrainingAttributeDto.java` - Insert attributes property
- `TrainingDto.java` - Insert attributes property
- `TrainingPopulator.java` - Insert attributes property
- `TrainingEntityConvTraining.java` - Insert attributes property
- `TrainingServiceImpl.java` - Create generateTrainingAttribute(...)
- `TrainingLoadInterceptor.java` - Insert trainingService.generateTrainingAttribute(...)
- `TrainingInitialDefaultsInterceptor.java` - Insert trainingService.generateTrainingAttribute(...)
- `trainingForm.html` - Insert:

```
...

<li class="nav-item">
        <a class="nav-link" id="custom-tabs-attribute-tab" data-toggle="pill" href="#custom-tabs-attribute"
role="tab" aria-controls="custom-tabs-attribute" aria-selected="false" th:text="#{module.common.attribute}"></a>
</li>

...

<div class="tab-pane fade" id="custom-tabs-attribute" role="tabpanel" aria-labelledby="custom-tabs-attribute-
tab">
        <th:block th:insert="~{backoffice/adminlte/common/fragment/content-form-attribute :: form(attributes=*
{attributes})}"></th:block>
</div>

...
```

Import Data

`beanframework/bin/config/src/main/resources/import/`

- `dev/initdata/**/*.csv` - Contains data to import every time when application startup.
  - `module.imex.import.init.locations=classpath*:import/dev/initdata/**/*.csv`
- `dev/initsql/**/*.sql` - Contains sql to execute every time when application startup.
  - `platform.import.startup.enabled=true`
  - `platform.import.sql.locations=classpath*:import/dev/initsql/**/*.sql`
- `dev/updatedata` - Contains data to import when manually performing platform update.
  - `module.imex.import.update.locations=classpath*:import/dev/updatedata`

- Data Syntax in CSV:
  - `INSERT` - To create new data only, no data created if exists in database,
  - `UPDATE` - To modify existing data only, no data updated if not exists in database.
  - `INSERT_UPDATE` - To create or modify data in database.
  - `REMOVE` - To remove existing data in database.

  Example:

```
INSERT_UPDATE Training,id      ,name
                      ,training,Training
```

- To add more data column in CSV:

- `TrainingCsv.` - Add new data attribute.
- `TrainingEntityCsvConverter.java` - Add new data attribute.
- `TrainingImportListener` - Optional: To customizing a new import logic.