Architecture - backend

backend, architecture

indexmenu_n_10

Backend is based primarily on the Spring and Activiti workflow technologies.

Application layers

Going from the bottom:

Entity (`AbstractEntity` ⇒ repository (`AbstractEntityRepository`) ⇒ dto (`AbstractDto`) ⇒ service (`AbstractReadWriteDtoService`) ⇒ REST (`DefaultReadWriteDtoController`).

Creating a child of each one of the classes above is sufficient for issuing a new REST endpoint for the new agenda (entity).

And now in a bit more detail …

Relational database

We are expecting to use the relational databese through a jdbc connector. Primarily, we are going to use `PosgreSQL` and h2 for demo purposes (e.g. the default spring profile is going to be connected to h2, so that nothing would need to be configured after downloading the application from Git and the application would work properly after deploying to Tomcat). Spring profiles for the individual supported databases will be created. There is going to be a set of database scripts (create + change scripts) for each database. We expect use the flyway tool for their administration.

Hibernate ORM

Using of Hibernate is a logical choice, nothing changes/ will change about that. We will keep creating `entities` which will map relational database tables into java classes. Moreover, entities will be supplemented with `jsr303` (@NotNull, @Email ..) validation annotations, which will be automatically validated (validation annotations are used in dto too and propagated to REST).

Setting the cache selectively for the individual operations: https://spring.io/blog/2015/06/15/cache-auto-configuration-in-spring-boot-1-3

Spring

The application is superordinate to the spring boot platform and, besides the core functions, it uses a number of spring frameworks, libraries and procedures:

- Spring Data

- Spring Security
- Spring Plugin
- AOP (aspectj)
- Events
- …

Spring Data

Spring Data facilitates the work with the (not only) relational database. The data can be accessed simply by issuing the interface according to certain conventions. I find this being combined with the securing through annotations and the possibility of adding SpEl expressions into queries very strong.

Intended extensions: - support of searching through standard jpa criteria for more difficult queries via Querydsl

Spring Data REST

Spring Data REST are used as helper for rest layer initialization only (convertors, patch method) - to make use of some of the advantages of spring data REST (patch method, json mapping, converters) and, at the same time, to be able to add a service layer,. we are exposing rest endpoint ourself as `@RestController` - we are using dto layer above entities, entities are not exposed directly on endpoint.

Dto layer is created above entities - services (`AbstractReadWriteDtoService`) works and exposed dtos (`AbstractDto`) only. Entities (`AbstractEntity`) are used on repository layer and internally in services. Rest endpoints work only with dtos, respectivelly with services using dtos. Dto can be identified by `uuid` identifier or by code (string code ⇒ `Codeable` identifier - e.g. identity is identified by username. For tutorial how to set codeable evaluator visit this tutorial). `LookupService` can be used for getting dto by uuid identifier or code.

Advantages of Spring Data REST (not used now):

- hal - uses Spring hateoas
- documentation directly at REST - annotation, apls
- projection(in the manner of trimmed view),
- securing through annotations,
- ~~event handlers (planned use of provisioning)~~
- patch method
- versing(ETag) - usable for optimistic locks
- last modification of resource

Disadvantages of Spring Data REST (not used now):

- Non-existence of service layer (only repositories)
- Non-existence of service layer (mapping to dtos) - I consider this so important that I have mentioned it twice. REST controllers are issued directly from entities, or rather from spring data repositories, which makes creating anything more complex than CRUD methods complicated.

An API cannot be created for services created in such a way - the entity itself which is fixed to implementation (database) flows everywhere.

- links with an absolute pathway to BE are used as entity identifiers (e.g. http://localhost:8080/idm/api/identities/username). If BE is built behind a **proxy**, it **should not be a problem**, it is only a matter of adding a configuration according to the manual (needs to be tested).

Spring Security

The Spring Data layer, REST or services themselves can be supplemented with annotations evaluating security (before or after calling). A logged-in user can be used favorably as an input for search queries (e.g. queries like "give me subordinates"). The SpEl expressions are used in the annotations, so that is will be possible to create and call any function which decides whether someone has or does not have a right for something.

Authorities model

Authorities (hasAuthority), not roles themselves, will be used for evaluating the target permissions. Similarly to the previous backend, there is a configuration of individual authorities (permissions) with each role. When an identity is assigned a role, the identity acquires the target authorities. Authorities are returned at login as a unique list of base permissions (e.g. USER_READ) ⇒ the authorities are available in the user's security context.

To make assigning permissions to a role easier the following interfaces are created:

- `BasePermission` - base permissions (e.g. READ, WRITE …)
- `GroupPermision` - group (target) permission (e.g. USER, ROLE …)

Assigning permissions to a role is represented by this pair:

- **target** - target permission - an instance of GroupPermision
- **action** - base permission - an instance of BasePermission

When evaluating, target and action acts as an authority **target_action** (e.g. USER_READ).

Each module can register its own set of permissions (both a new group, and adding new base permissions to an already existing group).

Each role has its own type (system, business, technical …) according to application settings.

Authorization policies can be used for securing data.

Workflow

We have chosen Activiti BPM Platform as our workflow engine. It is a widely-used workflow engine, which focuses mainly on the speed and simplicity of usage for developers. Its main purpose is controlling and executing procedures described using the BPMN 2.0 language in Java.

One of the advantages is its integration with Spring framework and therefore its easier application in our devstack. Activiti Platform allows us to issue a REST interface through which most functions can be controlled, i.e. it is possible to have Activiti Platform + REST interface running on the application server and communicate with it even from a non-Java environment. A disadvantage of this solution is the loss of direct Java integration, i.e. the possibility of calling Spring beans directly from a workflow process (using Expression Language), which considerably extends the potential of the workflow and speeds up development.

Backend functional requirements

Logging

Documentation

Notifications

Documentation

Audit

Documentation

Log locking

We expect to use optimistic locks on key entities. Auxiliary entities will not be locked - the last to come is the winner. If needed, it will be possible to engage an optimistic lock on any entity. The use of pessimistic locks is not expected yet.

Regarding the user interface, the `ETag` and `LastModified` headings are planned to be used for informing the user about changes made by somebody else in his currently edited log + a mechanism for using my changes or reloading the changed log.

The solution assumes that the objects will not be kept loaded (outside of the UI) for long in order not to break the optimistic log. At the same time, it is necessary to keep the transaction processing over the object being changed.

Optimistic lock is engaged on entities:

- Identity
- Organization (IdmTreeNode)
- Role
- System

Provisioning a synchronization

Documentation

Performing actions always under a user

TODO: Guest, system …

Non-functional requirements

CI/CD

Documentation

Open source

Project on GitHub

Public demo

Application profiles configured:

- Default application profile configured to db h2 is used for issuing a demo.

Testing

Documentation