PreTender

A software for tender application

Notes of design

About design style

This software was designed under the principles of SOA, aka. Service Oriented Architecture. This derective aims the separation the software components into smaller, independent or semi-independent services with high autonomy, while each of them serves a single, well defined purpose. In SOA, various business logic are fragmented into smaller, atomic components, and these components are organized into services, or capabilities of services. Four types of services exists: task, micro, entity and utility.

Design of PreTender

This software has 14 services:

- 5 Task Services
- 2 Micro Services
- 4 Entity Services
- 3 Utility Services

The 5 task services are the following:

- Apply Tender
- Authorize
- Finalize Application
- Register
- Login

Register Task Service

This task service provides registration to the software, which is required in order to use it. Provided with 3 inputs: username, password and e-mail address, it checkes if the username is already used or not, then initializes a new user in the database with the given parameters. The task service uses two other services: namely the User Entity Service, and the Hasher Utility Service. The User Entity Service represents the user entities in the process, and the Hasher Utility Service provides sha256 or md5 hashing algorithms in order to encode a given string. In this particular case, it is used to encode the passwords of users, for they are preserved in the database and for basic security reasions it is natural, that no password is stored in plain text.

While storing the user entities in the database does not corresponds to any capabilites of the User Entity Service[1], it is used to check wether the given username is already used by a user or not. In the former case, the registration process is not completed, as username is identical for each user.

Login Task Service

The Login Task Service executes the login business process. It is given a username and a password, then it checks whether the two is valid: the username is in the database, and the given password matches with the one in the database. It uses the User Entity Service in order to read the user from the database (and also to check if the given username is valid, and indeed used by a user). Once the user is found in the database, it uses the Hasher Utility Service to create a hashed version of the password. Since sha256 algorithm is used both in the registartion and here, if the original two passwords were the same, their hashed version will be the same, so authentication is indeed valid. Once this is done, JSON Web Token and Cookies are used to make login presistent. A JWT is created by the Tokener Utility Service, and the username of the logged in user is stored in the token (encoded of course). Once it's created, it will be stored in a cookie for 24 hours via the Baker Utility Service, which performs cookie management.

Authorization Task Service

Once logged in, it is preserved in a cookie for a day. After that, the Authorization Task Service is responsible for authorizing a user to use the many features of the software. It is a relatively simple Task Service, once it found the cookie on the computer, from it's value it can recover the JWT. Using the Tokener Utility Service, it decodes the token. Once decoded, the token holds the username of the logged in user. As another check of security, it uses a capability of the User Entity Service to read that particular user from the database. Only if it does indeed exists, will be the authorization granted.

Apply Tender Task Service

This service corresponds for applying to a tender by a user. Many tender can be found in the database, with various deadlines and structures. Since the Authorization Task Service grants the software the current username, this service first checks whether the username is valid by reading it using the User Entity Service. Then it checks whether the type of tender and the issued tender is correct using a capability of the Tender Type and Issued Tender Entity Services. Then it creates the new document, that should be stored as an application, by adding the properties which is in the corresponding tender type to a neutral scheme. Once done, it saves the tender in the database using the Save Application Micro Service. The save application business logic was not added as a capability of the Application Entity Service for the same reasons the creation of a user were not a capability of the User Entity Service (see footnotes). After all this is done, the Save Application Micro Services presents an ID, which belongs to the newly created application in the database, and this ID is inserted to the applications of the user in the database via the Modify capability of the User Entity Service.

Finalize Tender Task Service

Finalization of a tender can be separated into two subtask: the validation of it, and then writing the finalization into the database. While the latter is a relatively atomic task, and can be easly done by

[1] The creation of user was scrapped from it's entity service, beacuse after a long consideration, it was found that a user entity can be spoken of only after it exists. As long as it does not, it's not an entity, therefore should not be included in the entity service. In addition, the creation of a user is used by the registration business process and nothing else, therefore it cannot classified as non-agnostic or reusable. The same logic decided that the creation of application is not included in the Application Entity Service.

the Modify capability of the Application Entity Service, the former is more robust, but also very strongly tied to this particular task and so non-agnostic, therefore it is organized into a Micro Service. This Micro Service (Validate Application) checks if every single field of data, if they are matching to the well-defined requirements by the issued tender. Once it's done and everything is correct and the Task Service gets this answer from the Micro Service, it finalizes the application in the database.

Other Notes

All components are written in Javascript. Various services use MongoDB driver in order to connect and communicate with the database, which is – obviously - a MongoDB. Other libraries are used, such as Got and Express in order to make the services REST-compliant. Convict is used to provide a clean way for the configuration of each service.