

Design and Implementation choices for ACCA7005.

14/05/2021

ROKAS RUDYS

ID: 2012818

WORDS: 831

***Note the ReadME.md within the files contains all instructions and implementation structure and personal review including testing, Docker, KeyCloak and general structure.**

System plan ->

A RESTful Web Service with OAuth2 authentication using KeyCloak.

Five entities will exist within the application with the following logic:

1. Vehicle – the object that is being taxed and owned. There exists only one vehicle with the same identity, this identity carries on even when the vehicle is updated. Originally, I planned for multiple CarRegistration for a vehicle, but upon further consideration the relationship to a CarRegistration is oneToOne as historical information would be stored within unchangeable logs.
2. CarRegistration – has a dateFrom and dateEnd that vehicle has registered for, this registration carries on with the vehicle until a different CarOwner will register the vehicle, which is when a new CarRegistration will be created. This means Vehicles can exist not registered for a certain time frame, but CarRegistrations will have an unchangeable vehicle they were first registered to. CarRegistration will contain a list of TaxInformation objects that allow for a large store of time and to also determine if the vehicle is currently taxed by looking at the ascended order of TaxInformation by endDate and seeing if the EndDate is in the past or future, this will allow for a Boolean flag to determine if vehicle is currently taxes. This also allows for the structure of taxing different date periods. In addition, it will reference a CarOwner who has registered the vehicle and connect them both as a many-to-many type of table with extra information.

3. CarOwner object will contain all an individual's information, they will store a list of references to CarRegistrations and that is all. This allows for securing of personal information by limiting a single point of connection, which can be attentively managed, allowing for CarRegistrations to provide info to anyone about vehicle tax without revealing details of the individual who owns it.
4. TaxInformation contains a time frame that it covers, and the total cost of that time frame, within it exists a List of TaxPayments and a reference back to the CarRegistration. Having a list of TaxPayments allows for possible monthly plans, quarterly and such and will also allow checking if all the payments add up to the desired amount which is registered in TaxPayments.
5. Finally, the TaxPayment entity will contain the payment amount and paid date, and will follow a paymentStatus structute, which determines the state that the payment is in using enums. This provides better tracking to see if any payments have failed or cancelled, and it will also provide a history of failures/successes if new payment is created for every state although that would be a future business decision, but it provides the versatility to do both.

KeyCloak will provide a large quantity of roles allowing a separation of many features and possible use groups, these are as follows:

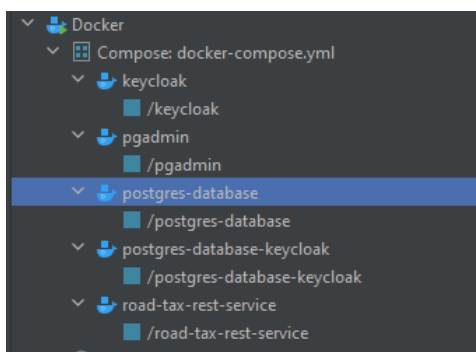
Role Name	Composite	Description
CarOwnerManager	True	Create, update and delete Car owners
CarOwnerViewer	False	View all car owners.
CarRegistrationManager	True	Create, update and delete car registrations
CarRegistrationViewer	False	View all car registrations
FullServiceManager	True	
PaymentManager	True	initiate, update and change payments.
PaymentViewer	False	View payments
TaxInformationViewer	False	View taxationInformation
TaxationManager	True	Create, delete and update taxInformation
VehicleManager	True	Create, update and delete vehicles
VehicleViewer	False	view all vehicles
default-roles-taxationservice	True	\${role_default-roles}
offline_access	False	\${role_offline-access}
uma_authorization	False	\${role_uma_authorization}

This large layer of separation will allow the RESTful service to be used by many use cases through the creation of accounts with specific roles. As an example, a CarOwnerViewer, VehicleViewer and TaxInformationViewer can become an authorized client that is able to check who owns a car and if they are currently taxed, which could be used by authorities.

As the entities and roles focus on providing as much use-cases and reusability so does the implementation and code, there exists 5 repositories, but most of the control will happen at the RestController layer, each controller provides the basic Create, Read, Update and Delete functionality, with the addition of specialization for certain entities, they will follow the same mapping structure and allow for common knowledge reuse.

To follow further with the maximization of reusability, Data Transfer Objects exist within both ways. Meaning they can be exchanged from entity to DTO as required. But also, they will provide a hierarchy relationship as discussed in the entity plan. DTO's also exist to create different type of authentication layers, for example a PresentCarRegistration that does not include the CarOwner, and the AuthorisedCarRegistration that does.

A docker-compose file exists to deploy the service fully, it includes separated database containers for the web service and KeyCloak security, more information about usage/development and login can be found in the readme.



A good testing coverage exists, although integration testing is lacking and only web layer and service layers are tested indepth. This is due to issues faced with overcoming KeyCloak adapter configuration.

A screenshot of a code coverage tool interface. At the top, it says 'Coverage: All in service x'. Below that, a summary bar shows '91% classes, 72% lines covered in package 'edu''. A table below this provides a breakdown of coverage for the 'vehicle' package.

Element	Class, % ▲	Method, %	Line, %
vehicle	91% (42/46)	76% (209/272)	72% (574/793)

ExceptionHandler handles thrown exceptions in a single place, this then converts it to a list of ApiErrors and produces a pleasant status outcome.

Swagger documentation exists for all controllers, I have also left entity documentation in place and have allowed for all of it to be displayed at once, an authorization token input is available allowing for authenticated calls directly from Swagger-ui. In a non-educational setting, all documentation would be hidden until authorization tokens are added to reveal and then only the token role documentation would be revealed.

