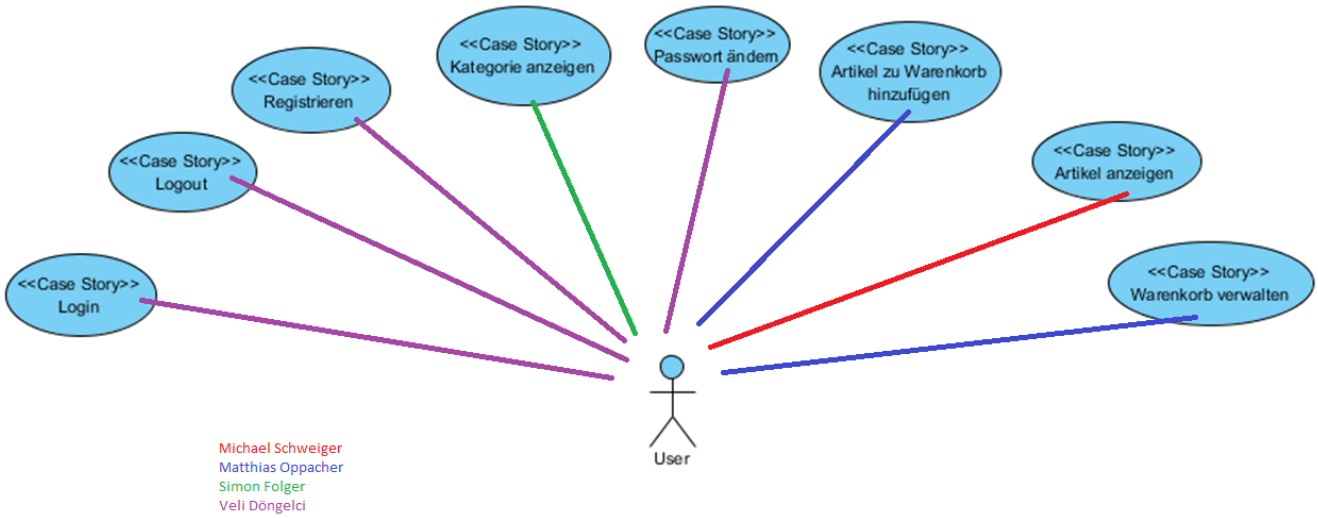
Web-Services 2016

Einzelbericht

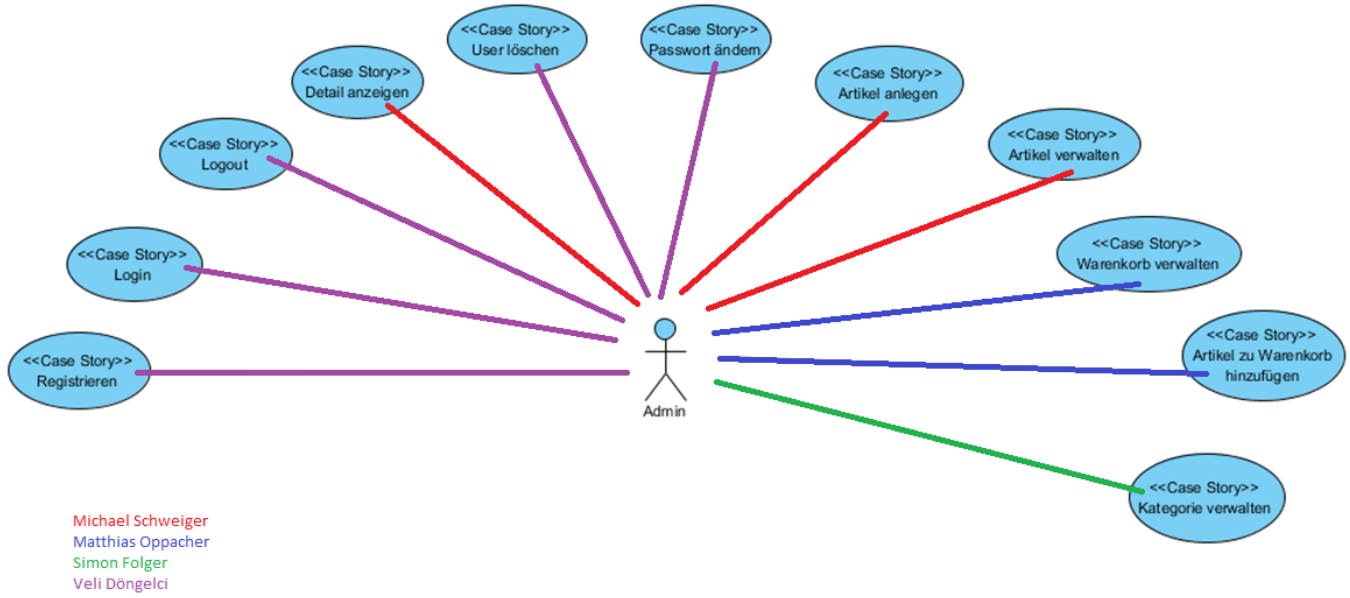
Team: ArtikelVerwaltung
Name: Veli Döngelci

1 Use Cases

User



Admin



Ich war für die folgende Use-Cases und Features:

- Authentifizierung & Autorisierung für alle Use-Cases
  - Token-basierte Autorisierung
  - Lokale Authentifizierung mit Mailadresse und einem min. 8-stelligen Passwort
- Registrierung
  - Benutzer gibt seine Kontodaten sowie ein 8-stelliges Passwort an.

- Passwort vergessen
    - Hier gibt der Benutzer Mailadresse und Geheimfrage und -antwort mit dem neuen Passwort an, um das Passwort zu ändern.
- Benutzerverwaltung
    - Admins können alle Benutzer auflisten, einen Benutzer löschen, die Daten von einem Benutzer ändern und einem Benutzer Admin-Rechte geben oder beheben.

Auf Frontend-Seite war ich für die Views von den entsprechenden Features verantwortlich. Außerdem habe ich User-Repository, Services und Tabellen erstellt. Interceptor für die Autorisierung war auch ein Teil meiner Aufgaben.

2 Schnittstelle

BaseApiController ist die Vaterklasse für alle davon vererbte Klassen. Meine Aufgabe ist hier, ein Auth- und User Controller zu implementieren. AuthController ist für Authentifizierung, Registrierung und „Passwort vergessen“ zuständig. Nach der Authentifizierung wird für den Benutzer ein Token generiert und in Response zurückgeschickt. Token und ID vom Benutzer werden immer mitgeschickt für Benutzer-Aktionen. Sie werden dann im Backend anhand eines Interceptors, ApiAuthFilterAttribute, bearbeitet. Wenn die Autorisierung nicht erfolgt, wird Status Code 401 zurückgeschickt.

Für die Autorisierung sind zwei Rollen vorgesehen, Benutzer und Admin. Die Rolle ist anhand des Filters festzulegen, ApiAuthFilterAttribute. Eine Schnittstelle darf nur eine Rolle besitzen. Filter ist defaultmäßig für jede Anfrage aktiv. Das kann aber mit AllowAnonymous-Attribute einfach deaktiviert werden.

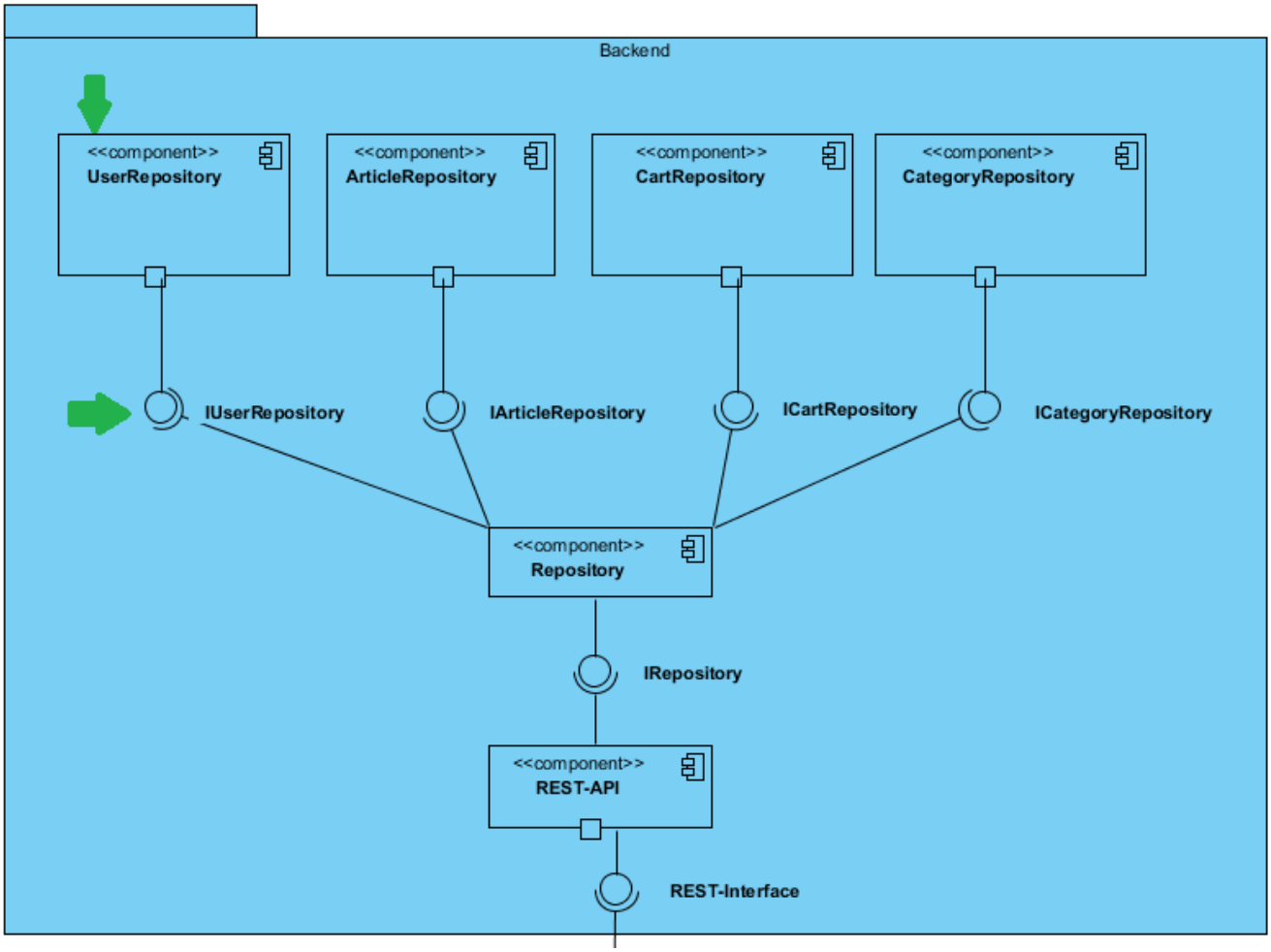
METHODE	**ROLLEN**		
	Anonymous	**Admin**	**User**
POST	/api/v1/auth/register		
POST	/api/v1/auth/login		
POST	/api/v1/auth/forgot		
DELETE	/api/v1/auth/logout		
GET		/api/v1/user/all	
PUT		/api/v1/user/{userid}/adminRights/grant	
DELETE		/api/v1/user/{userid}/adminRights/remove	
DELETE		/api/v1/user/{userid}/remove	
GET		/api/v1/user/{userid}/get	
PUT		/api/v1/user/{userid}/update	
DELETE			/api/v1/user/deleteAccount
PUT			/api/v1/user/editAccount
GET			/api/v1/user/accountDetails

UserController ist für Benutzer auflisten, ändern und löschen zuständig. Admins können diese Aktionen für alle Benutzer durchführen, wobei die Benutzer nur für ihr eigenes Konto.

3 Technische Architektur – Blockschaubild

Da meine Aufgaben um Benutzer-Aktionen gingen, habe ich UserRepository implementiert. UserRepository ist zuständig für alle Datenbank-Operationen von Benutzern.

Wir als Team haben uns für Repository Pattern entschieden, da die Mehrheit im Team sich damit besser auskennt. Das hat natürlich Nachteile, dass die Datenbank manuell angelegt werden. Der Vorteil ist aber, dass die Entities mithilfe von ADO.Net automatisch erstellt werden kann.



4 Technische Architektur – Datenhaltung

In unserem Projekt haben wir SQL-Server von Microsoft genutzt. Als ORM-Framework haben wir Entity-Framework. Im Hintergrund ermöglicht ORM-Technologie, Entity Framework, alle Prozesse einfach zu implementieren. Alle Änderungen werden in ORM durchgeführt. Danach werden mit SaveChanges()-Methode alle Änderungen in Datenbank persisted oder gelöscht.

FELDNAME	TYP	BESCHREIBUNG
ID	Int	Eindeutige ID vom Benutzer
NAME	nvarchar(255)	Name des Benutzers
ISADMIN	bit	Admin-Flag
EMAIL	nvarchar(255)	Mailadresse vom Benutzer
PASSWORD	nvarchar(255)	Kennwort des Benutzers (SHA256)
SECRETQUESTION	nvarchar(255)	Geheimfrage, Kennwort zurückzusetzen
SECRETANSWER	nvarchar(255)	Geheimantwort, Kennwort zurückzusetzen
TOKEN	nvarchar(255)	Eindeutiges Token für die Sitzung
TOKANDATE	datetime	Erstelldatum des Tokens

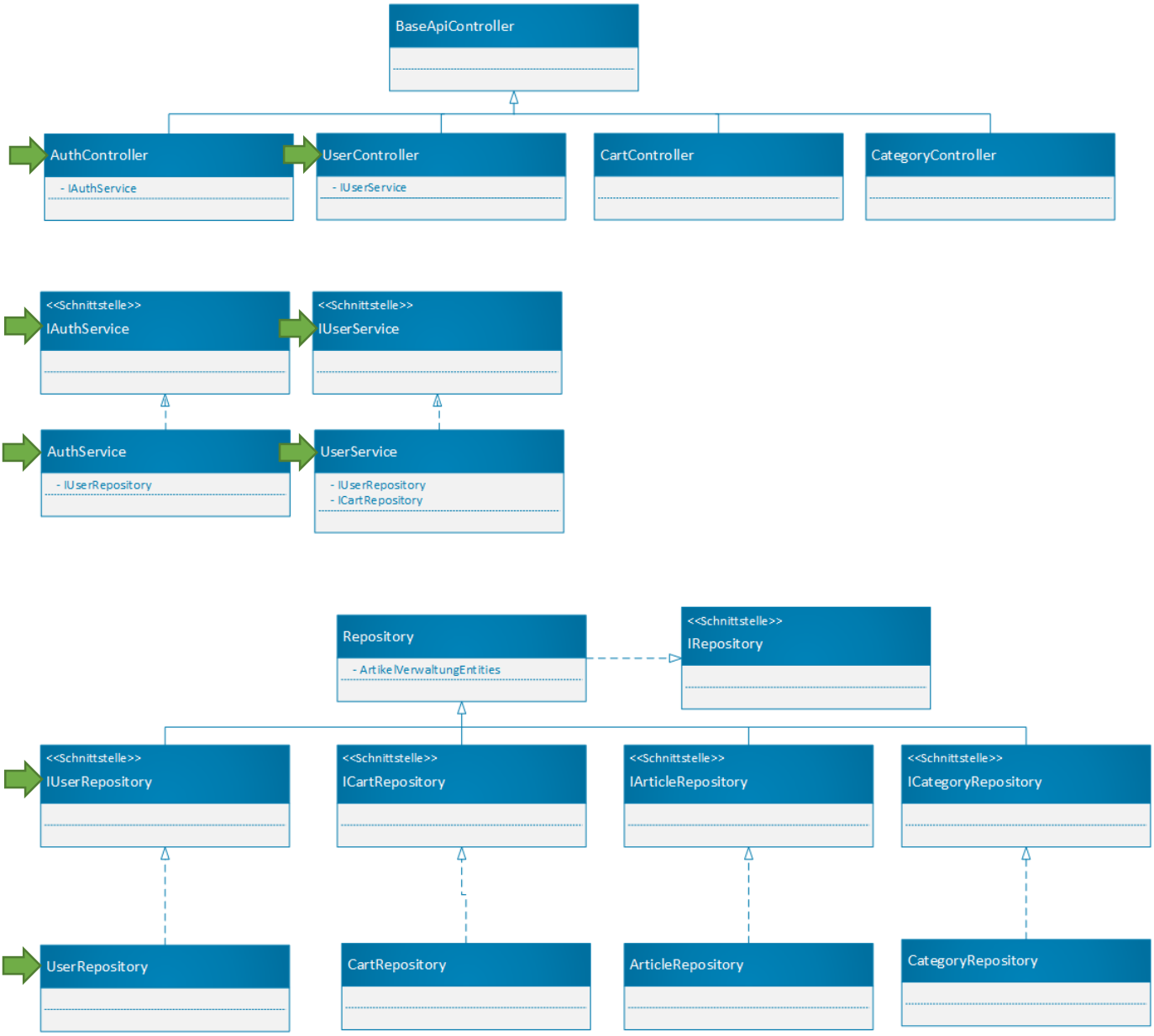
In Runtime wird TokenDate bei der Autorisierung verwendet. Wenn Token nicht älter als eine Stunde ist, erfolgt Autorisierung.

5 Technische Architektur – Klassendiagramm

	AUTHENTIFIZIERUNG UND AUTORISIERUNG	**BENUTZERVERWALTUNG**
CONTROLLER	AuthController	UserController
SERVICE	IAuthService => AuthService	IUserService => UserService
DTOS	LoginDTO, RegisterDTO, ForgotDTO, UserDTO, UserEditDTO, TokenAuthenticationIdetitiy	
REPOSITORY	IUserRepository, UserRepository	
UTILS	AuthUtil	
FILTER	ApiAuthFilterAttribute	

Die oben genannten Klassen gehören mir. Außerdem habe ich in den folgenden Klassen Änderungen gemacht oder neue Methoden hinzugefügt.

- ArtikelVerwaltung.API/
  - App_Start/
    - NinjectWebCommon.cs // Dependency hinzugefügt
    - WebApiConfig.cs // Filter hinzugefügt
  - Models
    - ModelFactory.cs



6 Testkonzept

Bei der Implementierung habe ich möglichst Seperation-of-Concept-Prinzip berücksichtigt. Dazu habe ich Repository-Zugriffe in Service-Klassen gekapselt, die in Controller-Klassen aufgerufen werden.

Aus diesem Sinne habe ich zwei Test-Projekte erstellt, wo Repository und API separat getestet werden. In ArtikelVerwaltung.Repository.UnitTest werden alle Repository-Aktionen getestet. Implementiert sind aber nur Repository- und UserRepository-Testklassen, die von mir implementiert wurden.

In ArtikelVerwaltung.API.UnitTest werden alle API-Aktionen getestet. Mithilfe von Sepration-of-Concern sind UserService- und AuthService-Testklassen implementiert.

7 Implementierung der App

Auf Frontend-Seite habe ich die folgenden Dateien erstellt, bzw. implementiert.

	AUTHENTIFIZIERUNG UND AUTORISIERUNG	**BENUTZERVERWALTUNG**
CONTROLLER	register.js, login.js, forgot.js	admin.js, user.js
SERVICE	authService.js	userService.js
DIRECTIVE	compareTo.js	
RESOURCE	authResource.js	userResource.js
VIEWS	register.html, login.html, forgot.html	user/*.html, admin/useredit.html, admin/userlist.html
CSS	user.scss	

Die obigen Dateien gehören mir. Außerdem habe ich in den folgenden Dateien Änderungen gemacht.

- ArtikelVerwaltung.Client/
  - app/
    - controllers
      - app.js
      - cart.js
    - app.js
    - config.js
    - config.router.js
  - views/
    - layout.html
  - index.html
  - bower.json

Zusätzlich habe ich materialicons-Unterstützung hinzugefügt.