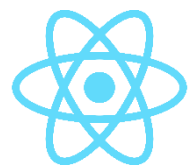
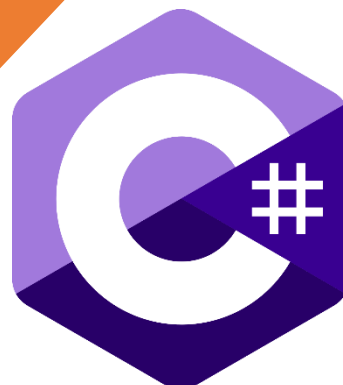


Mijn Reis Van C# Naar Java

Graduaatsproef
Indy Baeté



Inhoud

Voorpagina	1
Inhoud.....	2
Uitleg over de gekozen nieuwe technologie.....	3
Beschrijving van mijn project / opdracht	4
Hoe zit mijn project in elkaar / hoe ben ik tewerk gegaan	6
Backend (Java):	6
Frontend (React):	21
Resultaten	22
Conclusie	27
Bronnen	28

Uitleg over de gekozen nieuwe technologie

Ik heb ervoor gekozen om Java te leren als nieuwe technologie. Java is een zeer bekende programmeertaal waarin ik altijd al geïnteresseerd was. Echter, mijn keuze voor Java komt niet alleen uit mijn interesse in de taal. De voornaamste reden voor deze keuze is mijn ambitie om verder te studeren voor mijn bachelor toegepaste informatica. In deze opleiding is Java de meest gebruikte programmeertaal.

Aangezien ik van plan ben om de bachelor te volgen via een versneld traject, dat is mogelijk dankzij mijn diploma graduaat programmeren. Zal ik vrijstellingen krijgen voor veel vakken waarin Java wordt toegepast. Het wordt echter van mij wel verwacht dat ik voor de andere vakken weet hoe ik met Java moet werken. Daarom ging ik normaal gezien deze zomer de taal Java toch al moeten leren. De opdracht van mijn graduaatsproef, waarin we een nieuwe taal/technologie moesten leren, bleek dus bijzonder goed van pas te komen. Dan moet ik deze zomer Java niet meer leren en kan ik mij focussen op een vakantiejob.

Dus ik heb Java geleerd waarbij ik heb gewerkt met Maven en het Spring Boot-framework. Daarnaast heb ik het Spring Data JPA-framework gebruikt voor het beheren van mijn database. Ik heb dus met meerder nieuwe technologieën gewerkt.

Beschrijving van mijn project / opdracht

Ik heb een klanten en leden beheersysteem ontwikkeld voor een hotel. Hoewel de opdracht specifiek gericht was op het beheren van klanten en leden voor een hotel, is het mogelijk om dit beheer systeem licht aan te passen en te gebruiken voor andere applicaties waarbij je klanten kunt beheren. Mijn applicatie is ook volledig in het Engels geschreven.

De inspiratie voor deze opdracht kwam van een eerdere opdracht waarbij we een vergelijkbaar systeem moesten ontwikkelen, maar dan in C# met WPF als frontend. Mijn opdracht is niet exact hetzelfde. Maar het is wel heel erg gelijk.

Het klanten en leden beheersysteem is opgebouwd rondom de volgende functionaliteiten. Een lijst van klanten waarin je klanten kunt toevoegen, bijwerken, en verwijderen. Klanten hebben ook een lijst met leden onder zich waarin je leden kunt toevoegen, bijwerken, en verwijderen. Hierbij kan een klant bijvoorbeeld een familie zijn, en de leden zijn dan alle gezinsleden van de familie. Het is belangrijk op te merken dat als een klant of een lid wordt verwijderd, deze niet definitief uit de database worden verwijderd. In plaats daarvan wordt de status op 0 gezet, waardoor de gegevens behouden blijven.

De gegevens en validatie van klanten en leden omvatten:

Een klant heeft een identicator, naam, e-mailadres, telefoonnummer en postadres (gemeente, postcode, straatnaam, huisnummer). Leden van een klant hebben een identicator, naam en geboortedatum.

Voor validatie heb ik het niet al te ingewikkeld gemaakt. Alle velden mogen niet leeg zijn, er moet een '@' in een e-mailadres zitten en een geboortedatum mag niet in de toekomst liggen. Degene die de klanten en leden beheert, heeft dus redelijk veel vrijheid in welke data er ingevoerd wordt. Mocht ik extra tijd hebben gehad, dan had ik deze validatie strenger gemaakt, maar dit vond ik geen prioriteit.

Voor de implementatie van het project heb ik gebruik gemaakt van verschillende technologieën. Waaronder een backend geschreven in Java met Spring Boot en Maven, MySQL als database, en het Spring Data JPA-framework voor het beheren en communiceren tussen de database en de backend. Ik heb gewerkt met een 3-lagen model voor mijn backend, dit heb ik gedaan door alle lagen op te splitsen in verschillende packages. De frontend is ontwikkeld met React, waarbij ik Yarn heb gebruikt als package manager.

In tegenstelling tot de oorspronkelijke opdracht in C#, waar de backend rechtstreeks met de WPF-frontend communiceert, heb ik in mijn implementatie natuurlijk gebruik gemaakt van een REST API voor de communicatie tussen de backend en frontend. Ik vond dit zelf zeer interessant omdat ik graag wilde weten hoe je een REST API in Java kon ontwikkelen.

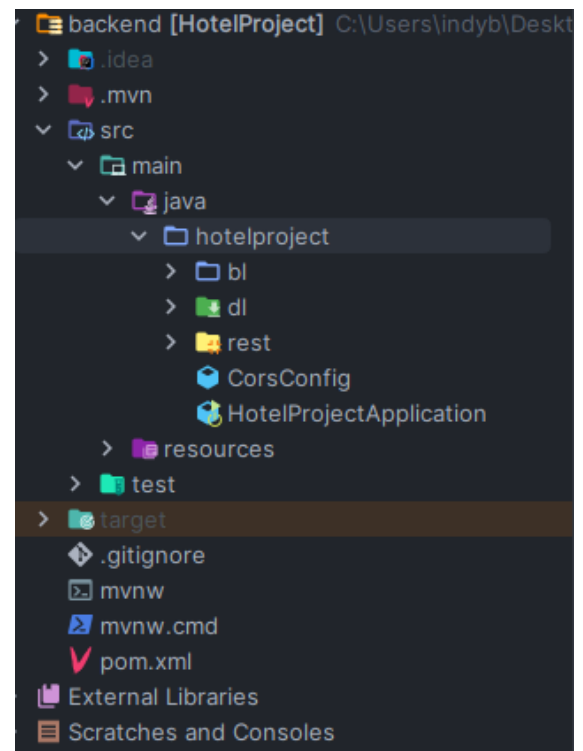
Hoe zit mijn project in elkaar / hoe ben ik tewerk gegaan

Backend (Java):

Voordat ik met de backend begon, heb ik me eerst beziggehouden met Java te leren. Dit ben ik beginnen doen sinds het begin van het semester, met wekelijks een aantal uurtjes hieraan te besteden. Gedurende deze tijd heb ik veel geleerd over Java, zoals Spring Boot, kennis over Maven, het werken met het Spring Data JPA-framework, en nog veel meer. Ik heb dit gedaan door het volgen van vele informatieve YouTube video's. Daarnaast heb ik talloze kleine oefenprojectjes uitgevoerd om mijn kennis te verdiepen. Dit heb ik niet alleen gedaan om Java te leren voor mijn project, maar ik wilde echt begrijpen wat ik aan het doen was. Dit was vooral belangrijk omdat ik deze taal heel goed moet kunnen voor mijn bachelor. In ongeveer twee maanden tijd heb ik ongeveer 35 uur besteed aan het leren van Java en het maken van oefenprojecten voordat ik begon aan de backend van mijn hotelproject. Deze voorbereiding heeft me enorm geholpen, omdat ik al een duidelijk idee had van hoe ik alles zou aanpakken voordat ik aan mijn backend begon.

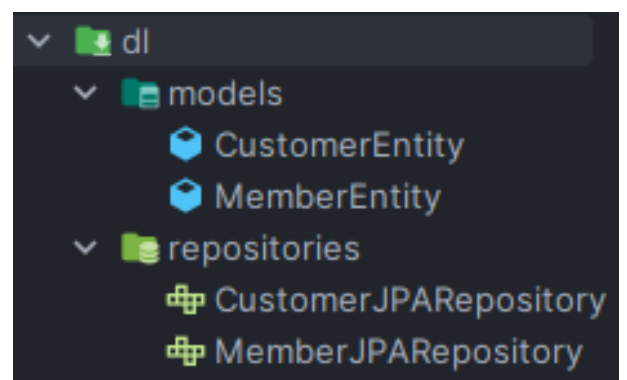
Vervolgens ben ik gestart met het opzetten van mijn backend. Nadat ik mijn spring boot project had aangemaakt, ben ik begonnen met het toevoegen van alle benodigde dependencies aan mijn pom.xml-bestand. Ik heb de dependency voor Spring Data JPA toegevoegd, evenals een dependency zodat ik kan verbinden met mijn MySQL-database. Daarnaast heb ik een dependency toegevoegd waarmee ik, net zoals bij ASP.NET Core, gebruik kan maken van Swagger UI om een goed overzicht te krijgen van al mijn requests.

Daarna heb ik het 3-lagenmodel opgezet in verschillende packages: data laag, business laag en rest laag, zoals je kunt zien in de bijvoegende afbeelding. De klasse `HotelProjectApplication` is de opstartklasse van mijn spring boot project, waarin ik niets hoefde te wijzigen. De klasse `CorsConfig` heb ik gemaakt om mijn applicatie te configureren, zodat mijn frontend toestemming had om verzoeken naar mijn backend te sturen.



Daarna heb ik tegelijkertijd gewerkt aan mijn data laag, business laag en rest laag. Maar ik zal eerst uitleggen hoe de data laag in elkaar zit, aangezien ik hier in het begin het meest aan heb gewerkt. Daarna zal ik uitleggen hoe de business laag in elkaar zit en dan hoe de rest laag in elkaar zit.

Omdat ik gebruik heb gemaakt van het Spring Data JPA-framework, was de data laag niet bijzonder moeilijk. In het begin was het wel uitdagend om alles te begrijpen, maar omdat ik dit al eerder had geleerd met mijn oefenprojectjes, viel het uiteindelijk mee. Het Spring Data JPA framework is eigenlijk vrij vergelijkbaar met Entity Framework in .NET. Zoals je kunt zien op de bijgevoegde foto, had ik slechts 2 klassen en 2 interfaces nodig voor de data laag.



In de CustomerEntity en MemberEntity klassen heb ik de klassen en de vereiste properties gemarkeerd met annotaties. Op deze manier kan het Spring Data JPA framework begrijpen hoe de database moet worden aangemaakt en hoe deze klassen moeten worden gekoppeld aan de database. Daarnaast heb ik ook de nodige constructors gemaakt en de nodige getters en setters in deze klassen geplaatst, zodat ik deze gegevens later kan mappen naar een domain model.

```
package hotelproject.dl.models;

import ...

18 usages  IndyBa
@Entity(name = "Customer")
public class CustomerEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    2 usages
    @Column
    private String name;

    2 usages
    @Column
    private String email;

    2 usages
    @Column
    private String phone;

    2 usages
    @Column
    private String address;

    2 usages
    @Column
    private boolean status;

    2 usages
    @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL)
    private List<MemberEntity> members;

    1 usage  IndyBa
    public CustomerEntity(Long id, String name, String email, String phone, String address, boolean status) {
        setId(id);
    }
}
```

```
1 package hotelproject.dl.models;
2
3 > import ...
4
5 19 usages  IndyBa
6 @Entity(name="Member")
7 public class MemberEntity {
8
9     @Id
10    @GeneratedValue(strategy = GenerationType.IDENTITY)
11    private Long id;
12
13    2 usages
14    @Column
15    private String name;
16
17    2 usages
18    @Column
19    private LocalDate birthDay;
20
21    2 usages
22    @Column
23    private boolean status;
24
25    2 usages
26    @ManyToOne
27    private CustomerEntity customer;
28
29    1 usage  IndyBa
30    public MemberEntity(Long id, String name, LocalDate birthDay, boolean status) {
31        setId(id);
32        setName(name);
33        setBirthDay(birthDay);
34        setStatus(status);
35        setCustomer(customer);
36    }
37
38    IndyBa
39    public MemberEntity() {
40        // Default constructor required by JPA
41    }
42
43 }
```


De JPA-repositories zijn interfaces die overerven van JpaRepository, zoals hieronder geïllustreerd. Bij het definiëren van de interface geef je de entity-klasse en het datatype van de primary key mee. Vervolgens kun je verschillende methoden uitvoeren op dit interface. In de businesslaag zul je later zien hoe dit wordt toegepast. JPA heeft ook al veel standaardmethoden die je niet zelf hoeft te implementeren. Je hebt ook de mogelijkheid om aangepaste methoden te definiëren, zoals hieronder gedemonstreerd. Deze methoden worden gegenereerd volgens naming conventions. JPA begrijpt zelf wat er moet gebeuren dankzij deze naming conventions.

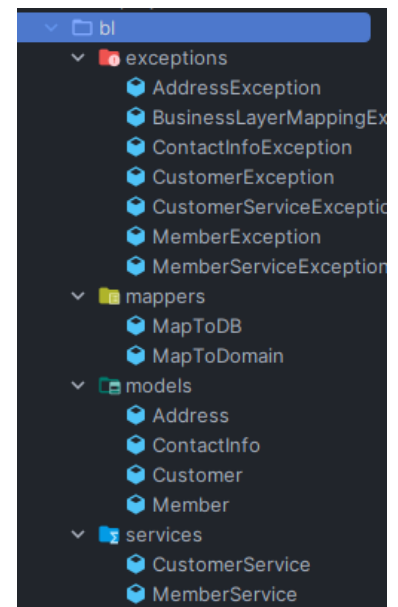
```
@Repository
public interface CustomerJpaRepository extends JpaRepository<CustomerEntity, Long> {
    1 usage  IndyBa
    boolean existsByIdAndStatusIsTrue(Long id);
    1 usage  IndyBa
    List<CustomerEntity> findByNameContainingAndStatusIsTrue(String name);
    4 usages  IndyBa
    CustomerEntity findByIdAndStatusIsTrue(Long id);
}

@Repository
10 public interface MemberJpaRepository extends JpaRepository<MemberEntity, Long> {
    1 usage  IndyBa
    11 boolean existsByCustomerIdAndIdAndStatusIsTrue(Long customerId, Long id);
    2 usages  IndyBa
    12 List<MemberEntity> findAllByCustomerIdAndStatusIsTrue(Long customerId);
    2 usages  IndyBa
    13 MemberEntity findByIdAndStatusIsTrue(Long id);
    14 }
```

Naast het maken van de JPA-repositories, moest ik ook een database/schema aanmaken in MySQL. Daarna moest ik alle instellingen configureren in het application.properties bestand dat zich onder de resources map bevindt. Hierdoor kan er verbinding worden gemaakt met de aangemaakte database, en wanneer ik het Spring Boot-programma uitvoer, worden automatisch alle tabellen gegenereerd.

```
spring.datasource.url=jdbc:mysql://localhost:3306/hotelldb?useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.hibernate.ddl-auto=update
```

En nu gaan we over naar de businesslaag. In de businesslaag heb ik voor alle klassen afzonderlijke exceptions gemaakt. Al deze exceptions erven over van de Exception klasse en nemen twee constructors over. De eerste constructor stelt je in staat om een foutbericht mee te geven, terwijl de tweede constructor je in staat stelt om een foutbericht en een inner-exception mee te geven.



Daarnaast heb ik verschillende model klassen gemaakt. In deze klassen is alle businesslogica opgenomen.

```
public class Customer {
    2 usages
    private String name;
    2 usages
    private long id;
    2 usages
    private ContactInfo contactInfo;
    2 usages
    private boolean status;
    6 usages
    private final List<Member> members = new ArrayList<>();

    1 usage
    public Customer(long id, String name, ContactInfo contactInfo) throws CustomerException {
        setId(id);
        setName(name);
        setContactInfo(contactInfo);
        setStatus(true);
    }

    1 usage
    public Customer(String name, ContactInfo contactInfo) throws CustomerException {
        setName(name);
        setContactInfo(contactInfo);
        setStatus(true);
    }

    public long getId() { return id; }
    public void setId(long id) throws CustomerException {
        if (id <= 0) {
            throw new CustomerException("Invalid id");
        }
        this.id = id;
    }

    public String getName() { return name; }
    public void setName(String name) throws CustomerException {
        if (name == null || name.trim().isEmpty()) {
            throw new CustomerException("Invalid name");
        }
        this.name = name;
    }

    2 usages
    public ContactInfo getContactInfo() { return contactInfo; }
    2 usages
    public void setContactInfo(ContactInfo contactInfo) throws CustomerException {
        if (contactInfo == null) {
            throw new CustomerException("ContactInfo is null");
        }
    }
}
```

```

    2 usages
    public ContactInfo getContactInfo() { return contactInfo; }
    2 usages
    public void setContactInfo(ContactInfo contactInfo) throws CustomerException {
        if (contactInfo == null) {
            throw new CustomerException("ContactInfo is null");
        }
        this.contactInfo = contactInfo;
    }

    1 usage
    public boolean isStatus() { return status; }
    3 usages
    public void setStatus(boolean status) { this.status = status; }

    3 usages
    public List<Member> getMembers() { return members; }
    3 usages
    public void addMember(Member member) throws CustomerException {
        if (!members.contains(member) && member != null) {
            members.add(member);
        } else {
            throw new CustomerException("Couldn't add the member");
        }
    }

    2 usages
    public void updateMember(Member updatedMember) throws CustomerException {
        if (updatedMember != null) {
            for (Member member : members) {
                if (member.getId() == updatedMember.getId()) {
                    members.remove(member);
                    members.add(updatedMember);
                    break;
                }
            }
        } else {
            throw new CustomerException("Couldn't update the member");
        }
    }
}
```

Zoals te zien is in de afbeeldingen, heb ik een Customer klasse. Deze klasse omvat een ContactInfo klasse en een lijst met Members. In deze ContactInfo klasse zit ook een Address klasse. Alle inkomende en uitgaande data wordt omgezet naar deze domein model klassen met behulp van mappers, zodat alle data gecontroleerd wordt door de businesslogica.

```
public class ContactInfo {
    2 usages
    private String email;
    2 usages
    private String phone;
    2 usages
    private Address address;

    2 usages
    public ContactInfo(String email, String phone, Address address) throws ContactInfoException {
        setEmail(email);
        setPhone(phone);
        setAddress(address);
    }

    2 usages
    public String getEmail() { return email; }

    1 usage
    public void setEmail(String email) throws ContactInfoException {
        if (email == null || !email.contains("@")) {
            throw new ContactInfoException("Invalid email");
        }
        this.email = email;
    }

    2 usages
    public String getPhone() { return phone; }

    1 usage
    public void setPhone(String phone) throws ContactInfoException {
        if (phone == null || phone.trim().isEmpty()) {
            throw new ContactInfoException("Invalid phone");
        }
        this.phone = phone;
    }

    2 usages
    public Address getAddress() { return address; }

    1 usage
    public void setAddress(Address address) throws ContactInfoException {
        if (address == null) {
            throw new ContactInfoException("Address is null");
        }
        this.address = address;
    }
}
```

```
36 usages
public class Member {
    5 usages
    private long id;
    5 usages
    private String name;
    5 usages
    private LocalDate birthDay;
    2 usages
    private boolean status;

    1 usage
    public Member(long id, String name, LocalDate birthDay) throws MemberException {
        setId(id);
        setName(name);
        setBirthDay(birthDay);
        setStatus(true);
    }

    1 usage
    public Member(String name, LocalDate birthDay) throws MemberException {
        setName(name);
        setBirthDay(birthDay);
        setStatus(true);
    }

    public long getId() { return id; }

    public void setId(long id) throws MemberException {
        if (id <= 0) {
            throw new MemberException("Invalid ID");
        }
        this.id = id;
    }

    public String getName() { return name; }
}
```

```
public String getName() { return name; }

public void setName(String name) throws MemberException {
    if (name == null || name.trim().isEmpty()) {
        throw new MemberException("Name is empty");
    }
    this.name = name;
}

public LocalDate getBirthDay() { return birthDay; }

public void setBirthDay(LocalDate birthDay) throws MemberException {
    if (birthDay.isAfter(LocalDate.now())) {
        throw new MemberException("Birthday is invalid");
    }
    this.birthDay = birthDay;
}

public boolean isStatus() { return status; }

public void setStatus(boolean status) { this.status = status; }

@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;

    Member member = (Member) obj;

    if (id != member.id) return false;
    if (!name.equals(member.name)) return false;
    return birthDay.equals(member.birthDay);
}

@Override
public int hashCode() { return Objects.hash(id, name, birthDay); }
```

```

public class Address {
    3 usages
    private String municipality;
    3 usages
    private String zipCode;
    3 usages
    private String houseNumber;
    3 usages
    private String street;

    1 usage
    1 IndyBa
    public Address(String municipality, String zipCode, String street, String houseNumber) {
        setMunicipality(municipality);
        setZipCode(zipCode);
        setStreet(street);
        setHouseNumber(houseNumber);
    }

    1 usage
    1 IndyBa
    public Address(String addressLine) throws AddressException {
        String[] parts = addressLine.split(Regex: "\\|");
        setMunicipality(parts[0]);
        setZipCode(parts[1]);
        setStreet(parts[2]);
        setHouseNumber(parts[3]);
    }

    1 usage
    1 IndyBa
    public String getMunicipality() { return municipality; }

    2 usages
    1 IndyBa
    public void setMunicipality(String municipality) throws AddressException {
        if (municipality == null || municipality.trim().isEmpty()) {
            throw new AddressException("Invalid municipality");
        }
        this.municipality = municipality;
    }

    1 usage
    1 IndyBa
    public String getZipCode() { return zipCode; }

    public String getZipCode() { return zipCode; }

    2 usages
    1 IndyBa
    public void setZipCode(String zipCode) throws AddressException {
        if (zipCode == null || zipCode.trim().isEmpty()) {
            throw new AddressException("Invalid zipcode");
        }
        this.zipCode = zipCode;
    }

    1 usage
    1 IndyBa
    public String getHouseNumber() { return houseNumber; }

    2 usages
    1 IndyBa
    public void setHouseNumber(String houseNumber) throws AddressException {
        if (houseNumber == null || houseNumber.trim().isEmpty()) {
            throw new AddressException("Invalid house number");
        }
        this.houseNumber = houseNumber;
    }

    1 usage
    1 IndyBa
    public String getStreet() { return street; }

    2 usages
    1 IndyBa
    public void setStreet(String street) throws AddressException {
        if (street == null || street.trim().isEmpty()) {
            throw new AddressException("Invalid street");
        }
        this.street = street;
    }

    1 IndyBa
    @Override
    public String toString() { return municipality + "|" + zipCode + "|" + street + "|" + houseNumber; }
}

```

Alle mapping tussen een domeinklasse en een entiteitsklasse wordt gedaan in de businesslaag. Dit komt omdat de JPA-repositories interfaces zijn, waardoor het gemakkelijker is om alle mapping in de services uit te voeren.

```

public class MapToDB {
    7 usages
    1 IndyBa
    public static CustomerEntity MapToCustomerDB(Customer c) throws BusinessLayerMappingException {
        try {
            ContactInfo ci = c.getContactInfo();
            CustomerEntity customerDB = new CustomerEntity(c.getId(), c.getName(), ci.getEmail(), ci.getPhone(), ci.getAddress().toString(), c.isStatus());
            customerDB.setMembers(MapToMembersDB(c.getMembers(), customerDB));
            return customerDB;
        } catch (Exception ex) {
            throw new BusinessLayerMappingException(ex.getMessage(), ex);
        }
    }

    1 usage
    1 IndyBa
    public static List<MemberEntity> MapToMembersDB(List<Member> members, CustomerEntity customerDB) throws BusinessLayerMappingException {
        try {
            List<MemberEntity> membersDB = new ArrayList<>();
            for (Member member : members) {
                membersDB.add(MapToMemberDB(member, customerDB));
            }
            return membersDB;
        } catch (Exception ex) {
            throw new BusinessLayerMappingException(ex.getMessage(), ex);
        }
    }

    5 usages
    1 IndyBa
    public static MemberEntity MapToMemberDB(Member m, CustomerEntity customerDB) throws BusinessLayerMappingException {
        try {
            return new MemberEntity(m.getId(), m.getName(), m.getBirthDay(), m.isStatus(), customerDB);
        } catch (Exception ex) {
            throw new BusinessLayerMappingException(ex.getMessage(), ex);
        }
    }
}

```

```

public class MapToDomain {
    1 usage 1x IndyBa
    public static List<Customer> MapToCustomersDomain(List<CustomerEntity> customersDB) throws CustomerException, AddressException, ContactInfoException, MemberException, BusinessLayerMappingException {
        try {
            List<Customer> customers = new ArrayList<>();
            for (CustomerEntity customerDB : customersDB) {
                customers.add(MapToCustomerDomain(customerDB));
            }
            return customers;
        } catch (CustomerException | ContactInfoException | AddressException | MemberException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new BusinessLayerMappingException(ex.getMessage(), ex);
        }
    }

    8 usages 1x IndyBa
    public static Customer MapToCustomerDomain(CustomerEntity c) throws AddressException, ContactInfoException, CustomerException, MemberException, BusinessLayerMappingException {
        try {
            Customer customer = new Customer(c.getId(), c.getName(), new ContactInfo(c.getEmail(), c.getPhone(), new Address(c.getAddress())));
            List<Member> members = MapToMembersDomain(c.getMembers());
            for (Member member : members) {
                customer.addMember(member);
            }
            return customer;
        } catch (CustomerException | ContactInfoException | AddressException | MemberException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new BusinessLayerMappingException(ex.getMessage(), ex);
        }
    }

    3 usages 1x IndyBa
    public static List<Member> MapToMembersDomain(List<MemberEntity> membersDB) throws MemberException, BusinessLayerMappingException {

```

```

    3 usages 1x IndyBa
    public static List<Member> MapToMembersDomain(List<MemberEntity> membersDB) throws MemberException, BusinessLayerMappingException {
        try {
            List<Member> members = new ArrayList<>();
            for (MemberEntity memberDB : membersDB) {
                if (memberDB.isStatus()) {
                    members.add(MapToMemberDomain(memberDB));
                }
            }
            return members;
        } catch (MemberException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new BusinessLayerMappingException(ex.getMessage(), ex);
        }
    }

    5 usages 1x IndyBa
    public static Member MapToMemberDomain(MemberEntity m) throws MemberException, BusinessLayerMappingException {
        try {
            return new Member(m.getId(), m.getName(), m.getBirthDay());
        } catch (MemberException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new BusinessLayerMappingException(ex.getMessage(), ex);
        }
    }
}

```

Vervolgens hebben we een CustomerService en een MemberService. In de constructor voegen we een CustomerJpaRepository en een MemberJpaRepository toe, en door @Autowired boven deze constructor te zetten, implementeert Spring Boot automatisch een singleton van deze repositories in de services. In de methoden van de service wordt ook alles gemapt en wordt er gebruikgemaakt van de repositories om gegevens toe te voegen en bij te werken in de database. Uiteindelijk zullen we deze methoden van de service gebruiken in onze restlaag.

```

@Service
public class CustomerService {

    7 usages
    private final CustomerJpaRepository customerRepository;

    2 usages
    private final MemberJpaRepository memberRepository;

    1 usages
    @Autowired
    public CustomerService(CustomerJpaRepository customerRepository, MemberJpaRepository memberRepository ) {
        this.customerRepository = customerRepository;
        this.memberRepository = memberRepository;
    }

    5 usages
    public boolean existsCustomer(Long id) throws CustomerServiceException {
        try {
            return customerRepository.existsByIdAndStatusIsTrue(id);
        } catch (Exception ex) {
            throw new CustomerServiceException(ex.getMessage(), ex);
        }
    }

    1 usages
    public List<Customer> getCustomers(String searchTerm) throws CustomerException, AddressException, ContactInfoException, MemberException, CustomerServiceException {
        try {
            if (searchTerm != null) { searchTerm = searchTerm.trim(); }
            return MapToCustomersDomain(customerRepository.findByNameContainingAndStatusIsTrue(Objects.requireNonNullElse(searchTerm, defaultObj: "")));
        } catch (CustomerException | ContactInfoException | AddressException | MemberException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new CustomerServiceException(ex.getMessage(), ex);
        }
    }

    2 usages
    public Customer getCustomer(Long id) throws CustomerException, AddressException, ContactInfoException, MemberException, CustomerServiceException {
        try {
            return MapToCustomerDomain(customerRepository.findByIdAndStatusIsTrue(id));
        } catch (CustomerException | ContactInfoException | AddressException | MemberException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new CustomerServiceException(ex.getMessage(), ex);
        }
    }
}

```

```

    public Customer addCustomer(Customer customer) throws CustomerException, AddressException, ContactInfoException, MemberException, CustomerServiceException {
        try {
            return MapToCustomerDomain(customerRepository.save(MapToCustomerDB(customer)));
        } catch (CustomerException | ContactInfoException | AddressException | MemberException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new CustomerServiceException(ex.getMessage(), ex);
        }
    }

    1 usages
    public Customer updateCustomer(Customer customer) throws CustomerException, AddressException, ContactInfoException, MemberException, CustomerServiceException {
        try {
            for (Member member : MapToMembersDomain(memberRepository.findAllByCustomerIdAndStatusIsTrue(customer.getId()))) {
                customer.addMember(member);
            }

            return MapToCustomerDomain(customerRepository.save(MapToCustomerDB(customer)));
        } catch (CustomerException | ContactInfoException | AddressException | MemberException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new CustomerServiceException(ex.getMessage(), ex);
        }
    }

    1 usages
    public void deleteCustomer(Long id) throws CustomerException, AddressException, ContactInfoException, MemberException, CustomerServiceException {
        try {
            Customer customer = getCustomer(id);
            customer.setStatus(false);

            for (Member member : customer.getMembers()) {
                member.setStatus(false);
            }

            MapToCustomerDomain(customerRepository.save(MapToCustomerDB(customer)));
        } catch (CustomerException | ContactInfoException | AddressException | MemberException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new CustomerServiceException(ex.getMessage(), ex);
        }
    }
}

```



```

@Service
public class MemberService {

    4 usages
    private final CustomerJpaRepository customerRepository;

    8 usages
    private final MemberJpaRepository memberRepository;

    @Autowired
    public MemberService(CustomerJpaRepository customerRepository, MemberJpaRepository memberRepository) {
        this.customerRepository = customerRepository;
        this.memberRepository = memberRepository;
    }

    3 usages
    public boolean existsMember(long memberId, long customerId) throws MemberServiceException {
        try {
            return memberRepository.existsByCustomerIdAndIdAndStatusIsTrue(memberId, customerId);
        } catch (Exception ex) {
            throw new MemberServiceException(ex.getMessage(), ex);
        }
    }

    1 usage
    public List<Member> getMembers(long customerId) throws MemberException, MemberServiceException {
        try {
            return MapToMembersDomain(memberRepository.findAllByCustomerIdAndStatusIsTrue(customerId));
        } catch (MemberException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new MemberServiceException(ex.getMessage(), ex);
        }
    }

    1 usage
    public Member getMember(long memberId) throws MemberException, MemberServiceException {
        try {
            return MapToMemberDomain(memberRepository.findByIdAndStatusIsTrue(memberId));
        } catch (MemberException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new MemberServiceException(ex.getMessage(), ex);
        }
    }

    public Member addMember(Member member, long customerId) throws MemberException, CustomerException, AddressException, ContactInfoException, MemberServiceException {
        try {
            Customer customer = MapToCustomerDomain(customerRepository.findByIdAndStatusIsTrue(customerId));
            customer.addMember(member);

            MemberEntity memberDB = MapToMemberDB(member, MapToCustomerDB(customer));
            return MapToMemberDomain(memberRepository.save(memberDB));
        } catch (MemberException | CustomerException | AddressException | ContactInfoException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new MemberServiceException(ex.getMessage(), ex);
        }
    }

    1 usage
    public Member updateMember(Member member, long customerId) throws CustomerException, AddressException, ContactInfoException, MemberException, MemberServiceException {
        try {
            Customer customer = MapToCustomerDomain(customerRepository.findByIdAndStatusIsTrue(customerId));
            customer.updateMember(member);

            MemberEntity memberDB = MapToMemberDB(member, MapToCustomerDB(customer));
            return MapToMemberDomain(memberRepository.save(memberDB));
        } catch (CustomerException | ContactInfoException | AddressException | MemberException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new MemberServiceException(ex.getMessage(), ex);
        }
    }

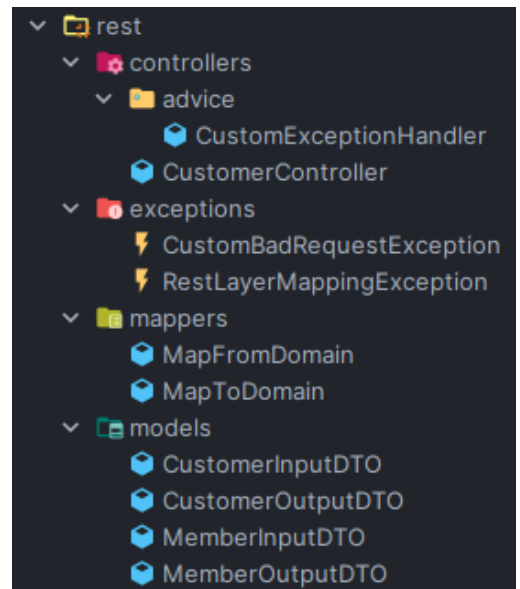
    1 usage
    public void deleteMember(long memberId, long customerId) throws CustomerException, AddressException, ContactInfoException, MemberException, MemberServiceException {
        try {
            Member member = MapToMemberDomain(memberRepository.findByIdAndStatusIsTrue(memberId));
            member.setStatus(false);

            Customer customer = MapToCustomerDomain(customerRepository.findByIdAndStatusIsTrue(customerId));
            customer.updateMember(member);

            memberRepository.save(MapToMemberDB(member, MapToCustomerDB(customer)));
        } catch (CustomerException | ContactInfoException | AddressException | MemberException ex) {
            throw ex;
        } catch (Exception ex) {
            throw new MemberServiceException(ex.getMessage(), ex);
        }
    }
}

```

Vervolgens hebben we de restlaag. In deze laag heb ik ook modellen voor DTO's (Data Transfer Objects) gemaakt. Voor zowel Member als Customer heb ik zowel een input als output DTO moeten creëren. Dit was noodzakelijk omdat de gegevens die uiteindelijk moesten worden weergegeven en de gegevens die moesten worden ingevoerd niet overeenkwamen met de domeinklassen.



(In de DTO-klassen waren ook getters en setters aanwezig. Deze staan echter niet in de afbeeldingen, aangezien er hier geen businesslogica op was toegepast en het toevoegen ervan veel ruimte in beslag zou nemen.)

```
public class CustomerInputDTO {
    no usages  IndyBa
    public CustomerInputDTO(String name, String email, String phone, String municipality, String zipCode, String street, String houseNumber) {
        this.name = name;
        this.email = email;
        this.phone = phone;
        this.municipality = municipality;
        this.zipCode = zipCode;
        this.street = street;
        this.houseNumber = houseNumber;
    }
}
```

```
3 usages
private String name;
3 usages
private String email;
3 usages
private String phone;
3 usages
private String municipality;
3 usages
private String zipCode;
3 usages
private String street;
3 usages
private String houseNumber;
```

```
public class CustomerOutputDTO {
    1 usage  IndyBa
    public CustomerOutputDTO(long id, String name, String email, String phone, String municipality, String zipCode, String street, String houseNumber, int nrOfMembers) {
        this.id = id;
        this.name = name;
        this.email = email;
        this.phone = phone;
        this.municipality = municipality;
        this.zipCode = zipCode;
        this.street = street;
        this.houseNumber = houseNumber;
        this.nrOfMembers = nrOfMembers;
    }
}
```

```
3 usages
private long id;
3 usages
private String name;
3 usages
private String email;
3 usages
private String phone;
3 usages
private String municipality;
3 usages
private String zipCode;
3 usages
private String street;
3 usages
private String houseNumber;
3 usages
private int nrOfMembers;
```

```
public class MemberInputDTO {
    3 usages
    private String name;
    3 usages
    private LocalDate birthDay;

    no usages  IndyBa
    public MemberInputDTO(String name, LocalDate birthDay) {
        this.name = name;
        this.birthDay = birthDay;
    }
}
```

```
public class MemberOutputDTO {
    3 usages
    private long id;
    3 usages
    private String name;
    3 usages
    private LocalDate birthDay;

    1 usage  IndyBa
    public MemberOutputDTO(long id, String name, LocalDate birthDay) {
        this.id = id;
        this.name = name;
        this.birthDay = birthDay;
    }
}
```


Daarnaast heb ik ook mappers geïmplementeerd om alle input DTO's naar domeinmodellen om te zetten, evenals mappers om domeinmodellen naar output DTO's om te zetten.

```
public class MapFromDomain {  
    1 usage  IndyBa  
    public static List<CustomerOutputDTO> MapFromCustomersDomain(List<Customer> customersDomain) throws RestLayerMappingException {  
        try {  
            List<CustomerOutputDTO> customers = new ArrayList<>();  
            for (Customer customerDomain : customersDomain) {  
                customers.add(MapFromCustomerDomain(customerDomain));  
            }  
            return customers;  
        } catch (Exception ex) {  
            throw new RestLayerMappingException(ex.getMessage(), ex);  
        }  
    }  
  
    4 usages  IndyBa  
    public static CustomerOutputDTO MapFromCustomerDomain(Customer c) throws RestLayerMappingException {  
        try {  
            ContactInfo ci = c.getContactInfo();  
            Address a = ci.getAddress();  
            return new CustomerOutputDTO(c.getId(), c.getName(), ci.getEmail(), ci.getPhone(), a.getMunicipality(), a.getZipCode(), a.getStreet(), a.getHouseNumber(), c.getMembers().size());  
        } catch (Exception ex) {  
            throw new RestLayerMappingException(ex.getMessage(), ex);  
        }  
    }  
}  
  
1 usage  IndyBa  
public static List<MemberOutputDTO> MapFromMembersDomain(List<Member> membersDomain) throws RestLayerMappingException {  
    try {  
        List<MemberOutputDTO> members = new ArrayList<>();  
        for (Member memberDomain : membersDomain) {  
            members.add(MapFromMemberDomain(memberDomain));  
        }  
        return members;  
    } catch (Exception ex) {  
        throw new RestLayerMappingException(ex.getMessage(), ex);  
    }  
}  
  
4 usages  IndyBa  
public static MemberOutputDTO MapFromMemberDomain(Member m) throws RestLayerMappingException {  
    try {  
        return new MemberOutputDTO(m.getId(), m.getName(), m.getBirthDay());  
    } catch (Exception ex) {  
        throw new RestLayerMappingException(ex.getMessage(), ex);  
    }  
}
```

```
1 usage  IndyBa  
public class MapToDomain {  
    2 usages  IndyBa  
    public static Customer MapToCustomerDomain(CustomerInputDTO c) throws CustomerException, AddressException, ContactInfoException, RestLayerMappingException {  
        try {  
            return new Customer(c.getName(), new ContactInfo(c.getEmail(), c.getPhone(), new Address(c.getMunicipality(), c.getZipCode(), c.getStreet(), c.getHouseNumber())));  
        } catch (CustomerException | ContactInfoException | AddressException ex) {  
            throw ex;  
        } catch (Exception ex) {  
            throw new RestLayerMappingException(ex.getMessage(), ex);  
        }  
    }  
  
    2 usages  IndyBa  
    public static Member MapToMemberDomain(MemberInputDTO m) throws MemberException, RestLayerMappingException {  
        try {  
            return new Member(m.getName(), m.getBirthDay());  
        } catch (MemberException ex) {  
            throw ex;  
        } catch (Exception ex) {  
            throw new RestLayerMappingException(ex.getMessage(), ex);  
        }  
    }  
}
```

Vervolgens heb ik een CustomerController waarin alle GET, POST, PUT en DELETE requests in staan. Deze controller maakt ook gebruik van @Autowired om automatisch een singleton van de services te verkrijgen. Hierdoor kan de controller communiceren met de onderliggende services om de benodigde bewerkingen uit te voeren.

```

@IndyBa
@RestController
@RequestMapping("/api")
public class CustomerController {

    // 11 usages
    private final CustomerService customerService;

    // 9 usages
    private final MemberService memberService;

    // IndyBa
    @Autowired
    public CustomerController(CustomerService customerService, MemberService memberService) {
        this.customerService = customerService;
        this.memberService = memberService;
    }

    // IndyBa
    @GetMapping("/customer")
    public ResponseEntity<List<CustomerOutputDTO>> getCustomers(@RequestParam(name = "searchTerm", required = false) String searchTerm) {
        try {
            List<CustomerOutputDTO> customers = MapFromCustomersDomain(customerService.getCustomers(searchTerm));
            if (customers.isEmpty()) { return ResponseEntity.notFound().build(); }

            return ResponseEntity.ok(customers);
        } catch (Exception ex) {
            return ResponseEntity.notFound().build();
        }
    }

    // IndyBa
    @PostMapping("/customer")
    public ResponseEntity<CustomerOutputDTO> postCustomer(@RequestBody CustomerInputDTO customerInput) {
        try {
            CustomerOutputDTO createdCustomer = MapFromCustomerDomain(customerService.addCustomer(MapToCustomerDomain(customerInput)));

            URI location = URI.create("/customer/" + createdCustomer.getId());

            return ResponseEntity.created(location).body(createdCustomer);
        } catch (CustomerException | ContactInfoException | AddressException | MemberException ex) {
            throw new CustomBadRequestException(ex.getMessage());
        } catch (Exception ex) {
            return ResponseEntity.badRequest().build();
        }
    }
}

```

```

// IndyBa
@GetMapping("/customer/{id}")
public ResponseEntity<CustomerOutputDTO> getCustomer(@PathVariable long id) {
    try {
        if (!customerService.existsCustomer(id)) { return ResponseEntity.notFound().build(); }

        return ResponseEntity.ok(MapFromCustomerDomain(customerService.getCustomer(id)));
    } catch (Exception ex) {
        return ResponseEntity.notFound().build();
    }
}

// IndyBa
@PutMapping("/customer/{id}")
public ResponseEntity<CustomerOutputDTO> putCustomer(@PathVariable long id, @RequestBody CustomerInputDTO customerInput) {
    try {
        if (!customerService.existsCustomer(id)) { return ResponseEntity.notFound().build(); }

        Customer updatedCustomer = MapToCustomerDomain(customerInput);
        updatedCustomer.setId(id);

        CustomerOutputDTO updatedCustomerOutputDto = MapFromCustomerDomain(customerService.updateCustomer(updatedCustomer));

        return ResponseEntity.ok().body(updatedCustomerOutputDto);
    } catch (CustomerException | ContactInfoException | AddressException | MemberException ex) {
        throw new CustomBadRequestException(ex.getMessage());
    } catch (Exception ex) {
        return ResponseEntity.badRequest().build();
    }
}
}

```

```

@DeleteMapping (@"/customer/{id}")
public ResponseEntity<Void> deleteCustomer(@PathVariable Long id) {
    try {
        if (!customerService.existsCustomer(id)) { return ResponseEntity.notFound().build(); }

        customerService.deleteCustomer(id);

        return ResponseEntity.noContent().build();
    } catch (Exception ex) {
        return ResponseEntity.badRequest().build();
    }
}

@IndyBa
@GetMapping (@"/customer/{id}/member")
public ResponseEntity<List<MemberOutputDTO>> getMembers(@PathVariable Long id) {
    try {
        if (!customerService.existsCustomer(id)) { return ResponseEntity.notFound().build(); }

        List<MemberOutputDTO> members = MapFromMembersDomain(memberService.getMembers(id));
        if (members.isEmpty()) { return ResponseEntity.notFound().build(); }

        return ResponseEntity.ok(members);
    } catch (Exception ex) {
        return ResponseEntity.notFound().build();
    }
}

@IndyBa
@PostMapping (@"/customer/{id}/member")
public ResponseEntity<MemberOutputDTO> postMembers(@PathVariable Long id, @RequestBody MemberInputDTO memberInput) {
    try {
        if (!customerService.existsCustomer(id)) { return ResponseEntity.notFound().build(); }

        MemberOutputDTO createdMember = MapFromMemberDomain(memberService.addMember(MapToMemberDomain(memberInput), id));

        URI location = URI.create("/customer/" + id + "/" + createdMember.getId());

        return ResponseEntity.created(location).body(createdMember);
    } catch (CustomerException | ContactInfoException | AddressException | MemberException ex) {
        throw new CustomBadRequestException(ex.getMessage());
    } catch (Exception ex) {
        return ResponseEntity.badRequest().build();
    }
}

@IndyBa
@GetMapping (@"/customer/{customerId}/member/{memberId}")
public ResponseEntity<MemberOutputDTO> getMembers(@PathVariable Long customerId, @PathVariable Long memberId) {
    try {
        if (!memberService.existsMember(customerId, memberId)) { return ResponseEntity.notFound().build(); }

        return ResponseEntity.ok(MapFromMemberDomain(memberService.getMember(memberId)));
    } catch (Exception ex) {
        return ResponseEntity.notFound().build();
    }
}

```

```

@IndyBa
@PutMapping (@"/customer/{customerId}/member/{memberId}")
public ResponseEntity<MemberOutputDTO> putCustomer(@PathVariable Long customerId, @PathVariable Long memberId, @RequestBody MemberInputDTO memberInput) {
    try {
        if (!memberService.existsMember(customerId, memberId)) { return ResponseEntity.notFound().build(); }

        Member member = MapToMemberDomain(memberInput);
        member.setId(memberId);

        MemberOutputDTO updatedMemberOutputDto = MapFromMemberDomain(memberService.updateMember(member, customerId));

        return ResponseEntity.ok().body(updatedMemberOutputDto);
    } catch (CustomerException | ContactInfoException | AddressException | MemberException ex) {
        throw new CustomBadRequestException(ex.getMessage());
    } catch (Exception ex) {
        return ResponseEntity.badRequest().build();
    }
}

@IndyBa
@DeleteMapping (@"/customer/{customerId}/member/{memberId}")
public ResponseEntity<Void> putCustomer(@PathVariable Long customerId, @PathVariable Long memberId) {
    try {
        if (!memberService.existsMember(customerId, memberId)) { return ResponseEntity.notFound().build(); }

        memberService.deleteMember(memberId, customerId);

        return ResponseEntity.noContent().build();
    } catch (Exception ex) {
        return ResponseEntity.badRequest().build();
    }
}
}

```

Ik heb twee exceptions geïmplementeerd in de restlaag, een exception voor mapping-fouten en een `BadRequestException`. Deze `BadRequestException` is ontworpen om het mogelijk te maken een bericht mee te geven bij het gooien van een bad request. Deze aanpak was nodig omdat in Spring Boot het lastig is om direct een bericht mee te geven in de body als de return type een output DTO is. Om dit te omzeilen, heb ik in de `RestControllerAdvice` iets ingesteld. Als er een `CustomBadRequestException` wordt gegooid in de controller, zal er een response van het type string terug worden gestuurd. Hierdoor kan ik een bad request terug sturen met een string in de body.

```
IndyBa
@RestControllerAdvice
public class CustomExceptionHandler {
    IndyBa
    @ExceptionHandler({CustomBadRequestException.class})
    public ResponseEntity<String> handleBadRequestException(CustomBadRequestException ex) {
        return ResponseEntity.badRequest().body(ex.getMessage());
    }
}
```

Zoals je misschien al eerder had opgemerkt, worden `CustomerExceptions`, `ContactInfoExceptions`, `AddressExceptions` en `MemberException` altijd opgevangen en opnieuw gegooid. Vervolgens worden deze exceptions in de controller opgevangen, waarbij een `CustomBadRequestException` wordt gegooid met de foutmelding. Dit maakt het mogelijk om fouten terug te geven wanneer er bijvoorbeeld een onjuiste waarde wordt ingevoerd bij het maken van een POST-request van een Customer.

```
"name": "",
"email": "familytest@gmail.com",
"phone": "0400000000",
"municipality": "Oost-Vlaanderen",
"zipCode": "9000",
"street": "Test Straat",
"houseNumber": "1"
}
```

Request URL

`http://localhost:8080/api/customer`

Server response

Code	Details
400 <i>Undocumented</i>	Error: response status is 400
	Response body
	Invalid name

Dit was het dan voor de backend, hier heb ik ongeveer 65 uur aan besteed. Alle resultaten van de backend zul je later in dit verslag terugvinden bij het hoofdstuk "Resultaten".

Frontend (React):

Ik heb ongeveer 30 uur besteed aan de frontend, hoewel dit oorspronkelijk niet de bedoeling was. Mijn plan was eigenlijk om snel een frontend samen te stellen voor de presentatie, zodat ik niet alleen alle HTTP-verzoeken moest laten tonen bij het presenteren van mijn graduaatsproef. Ik had oorspronkelijk bedacht hier 15 uur aan te besteden, maar ik ben er dieper op ingegaan dan gepland. Want als ik iets doe, wil ik het altijd goed wil doen. React was ook nieuw voor mij. Net als bij Java, ben ik pas in september voor het eerst met React in aanraking gekomen, en dit project was mijn eerste grote project met React. Maar React behoort officieel niet tot de nieuwe technologie voor mijn graduaatsproef. Daarom zal ik niet alle code tonen zoals ik wel heb gedaan voor de backend. Als je mijn code toch wilt bekijken, kun je dat natuurlijk zelf doen door mijn project te openen in een IDE.

Ik heb een lijst van klanten gemaakt, met rechtsboven een knop om een nieuwe klant toe te voegen. Naast elke klant heb je twee knoppen om de klant bij te werken of te verwijderen. Wanneer je een klant bijwerkt, krijg je een menu waar je alle gegevens van een klant kunt aanpassen en komt er ook een lijst met alle leden van de klant in het menu. Bij deze leden kun je opnieuw rechtsboven een nieuw lid toevoegen en naast elk lid zijn er opnieuw knoppen om het lid bij te werken en te verwijderen.

Om deze CRUD-operaties uit te voeren, heb ik Axios gebruikt om de GET, POST, PUT en DELETE verzoeken naar de backend te sturen. Ik heb er ook voor gezorgd dat de juiste foutmeldingen op de frontend worden weergegeven als er iets misgaat met de backend. Dit omvat scenario's zoals het niet kunnen verbinden met de backend of bijvoorbeeld het proberen bijwerken van een klant die al is verwijderd.

Alle resultaten van de frontend zul je later in dit verslag terugvinden bij het hoofdstuk "Resultaten".





























Resultaten

Als resultaat van de backend heb ik een goed functionerend REST API ontwikkeld waar alle verzoeken goed werken. Bovendien worden correcte foutstatussen en foutberichten weergegeven in het geval van problemen. Ik zal niet elk verzoek afzonderlijk tonen in het Swagger UI, aangezien alle resultaten van deze verzoeken gebruikt zullen worden op de website (frontend) van het hotel klanten en leden beheersysteem.

GET	/api/customer/{id}
PUT	/api/customer/{id}
DELETE	/api/customer/{id}
GET	/api/customer/{customerId}/member/{memberId}
PUT	/api/customer/{customerId}/member/{memberId}
DELETE	/api/customer/{customerId}/member/{memberId}
GET	/api/customer
POST	/api/customer
GET	/api/customer/{id}/member
POST	/api/customer/{id}/member

Het eerste wat je ziet wanneer je de frontend opent, is een lijst met alle klanten.

Hotel Project

Search Customer										Add Customer
Id	Name	Email	Phone	Municipality	Zip Code	Street	House Number	Members	Actions	
1	Family Smith	family.smith@example.com	32400000001	Oost-Vlaanderen	9000	Test Straat	1	3	 	
2	Company IT	company.it@example.com	32400000002	Oost-Vlaanderen	9000	Test Straat	2	2	 	
3	Group Members	group.members@example.com	32400000003	Oost-Vlaanderen	9000	Test Straat	3	3	 	
4	Team Alpha	team.alpha@example.com	32400000004	Oost-Vlaanderen	9000	Test Straat	4	1	 	
5	Organization XYZ	organizationxyz@example.com	32400000005	Oost-Vlaanderen	9000	Test Straat	5	2	 	
6	Family Jones	family.jones@example.com	32400000006	Oost-Vlaanderen	9000	Test Straat	6	3	 	
7	Family White	family.white@example.com	32400000007	Oost-Vlaanderen	9000	Test Straat	7	1	 	
8	Family Black	family.black@example.com	32400000008	Oost-Vlaanderen	9000	Test Straat	8	3	 	
9	Family Green	family.green@example.com	32400000009	Oost-Vlaanderen	9000	Test Straat	9	2	 	
10	Family Brown	family.brown@example.com	32400000010	Oost-Vlaanderen	9000	Test Straat	10	1	 	
11	Family Johnson	family.johnson@example.com	32400000011	Oost-Vlaanderen	9000	Test Straat	11	3	 	
12	Family Miller	family.miller@example.com	32400000012	Oost-Vlaanderen	9000	Test Straat	12	2	 	
13	Family Davis	family.davis@example.com	32400000013	Oost-Vlaanderen	9000	Test Straat	13	2	 	
14	Family Garcia	family.garcia@example.com	32400000014	Oost-Vlaanderen	9000	Test Straat	14	1	 	

Je kunt ook zoeken naar klanten op naam door een zoekterm in te voeren. In dat geval wordt er een verzoek verzonden naar het pad `/customer?searchTerm=(je zoektekst)`.

×

Add Customer

Id	Name	Email	Phone	Municipality	Zip Code	Street	House Number	Members	Actions
1	Family Smith	family.smith@example.com	32400000001	Oost-Vlaanderen	9000	Test Straat	1	3	

Je hebt ook de mogelijkheid om klanten te verwijderen door op het prullenbakje te klikken, dat zich bevindt helemaal rechts onder "Actions". Als je daarna op "OK" klikt, wordt de klant verwijderd en wordt de lijst automatisch bijgewerkt, waardoor je direct ziet dat de klant is verwijderd. Natuurlijk worden deze klanten niet echt uit de database worden verwijderd, ze worden gewoon op status 0 gezet.

localhost:5173 meldt het volgende
Are you sure you want to delete this customer? (Customer Id: 2)
OK Annuleren

Add Customer

Id	Name	Email	Phone	Municipality	Zip Code	Street	House Number	Members	Actions
1	Family Smith	family.smith@example.com	32400000001	Oost-Vlaanderen	9000	Test Straat	1	3	
2	Company IT	company.it@example.com	32400000002	Oost-Vlaanderen	9000	Test Straat	2	2	

Als je rechtsboven op "Add Customer" klikt, komt er een menu op beeld en kun je een klant toevoegen. In deze velden is ook front-end validatie geïmplementeerd, waardoor foutieve waarden in principe niet naar de backend kunnen worden verzonden. Mocht er echter toch iets fout gaan, wordt er een passende foutmelding weergegeven. Deze foutmeldingen zal ik later laten zien. Wanneer alle waarden correct zijn ingesteld en je op de knop "Add" klikt, wordt de lijst automatisch bijgewerkt, waarbij je meteen ziet dat er een nieuwe klant is toegevoegd.

Add Customer

NAME

Family Test

EMAIL

test@gmail.com

PHONE

Phone number is required

MUNICIPALITY

Municipality is a required field!

ZIP CODE

Zip code is a required field!

STREET

Street is a required field!

HOUSE NUMBER

House number is a required field!

Cancel

Add

Als je op het pen pictogram klikt, komt er een menu op beeld waar je een klant kunt bijwerken. Alle oorspronkelijke waarden van de klant worden automatisch ingevuld in de bijbehorende velden. Net zoals bij het toevoegen van een klant is er ook front-end validatie geïmplementeerd. Als je op "Update" klikt, wordt de lijst met klanten automatisch bijgewerkt, waarbij je direct ziet dat de klant is bijgewerkt.

Update Customer

NAME

Family Smith

EMAIL

family.smith@example.com

PHONE

32400000001

MUNICIPALITY

Oost-Vlaanderen

ZIP CODE

9000

STREET

Test Straat







HOUSE NUMBER

1

Members:

(changes to members will automatically be saved)

Add Member

Id	Name	Birthday	Actions
1	Max	1990-10-02	 
2	Emma	1985-05-15	 
3	Jack	1988-08-20	 

Cancel

Update

In het "Update Customer" menu is er ook een lijst met alle leden van de klant. Aanpassingen aan de leden worden automatisch opgeslagen. Hier werkt alles op dezelfde manier als bij de klanten. Je kunt leden toevoegen met de knop rechtsboven, leden verwijderen door op het prullenbakpictogram te klikken, en leden bijwerken door op het penpictogram te klikken. Net als bij de klanten is er ook frontend validatie bij het updaten en toevoegen van leden.

Add Member

NAME

BIRTHDAY

dd-mm-jjjj

Cancel

Add

Update Member

NAME

Max

BIRTHDAY

02-10-1990

Cancel

Update

localhost:5173 meldt het volgende

Are your sure you want to delete this Member? (Member Id: 1)



OK

Annuleren

Members:

(changes to members will automatically be saved)

Add Member

Id	Name	Birthday	Actions
1	Max	1990-10-02	 

Als er zich fouten voordoen bij het opvragen, toevoegen, bijwerken of verwijderen van klanten en leden in de backend, worden deze fouten weergegeven in de frontend. Deze fouten kunnen optreden wanneer bijvoorbeeld twee personen tegelijkertijd in de frontend werken en één persoon een klant verwijdert terwijl iemand anders dezelfde gebruiker probeert bij te werken. Ook kunnen deze fouten ontstaan als er op een of andere reden geen verbinding met de backend kan worden gemaakt.

Hoewel ik niet veel tijd heb besteed aan de frontend-implementatie van deze foutmeldingen, zijn ze wel aanwezig. Hieronder staan enkele afbeeldingen met voorbeelden van deze foutmeldingen.

×

Add Customer

Id	Name	Email	Phone	Municipality	Zip Code	Street	House Number	Members	Actions
No Customers Found									

Add Customer

Id	Name	Email	Phone	Municipality	Zip Code	Street	House Number	Members	Actions
Error: Couldn't connect to the server									

localhost:5173 meldt het volgende
AxiosError: Network Error

OK

Add Customer

NAME
Test

EMAIL
test@gmail.com

PHONE
3240000000

MUNICIPALITY
Oost-Vlaanderen

ZIP CODE
9000

STREET
Test Straat

HOUSE NUMBER
10

Cancel Adding...

Update Customer

NAME
Family White

EMAIL
family.white@example.com

PHONE
32400000020

MUNICIPALITY
Oost-Vlaanderen

ZIP CODE
9000

STREET
Test Straat

HOUSE NUMBER
20

Members:
(changes to members will automatically be saved)

Add Member

Id	Name	Birthday	Actions
42	Ella	1994-04-18	✎ 🗑
43	Noah	1980-12-30	✎ 🗑
44	Sophia	1992-06-22	✎ 🗑
45	Liam	1983-11-15	✎ 🗑
46	Olivia	1997-09-08	✎ 🗑

Cancel Updating...

Update Customer

Error with loading customer: Network Error

Cancel Update

De foutmeldingen voor de leden zijn identiek aan die voor de klanten. Ik zal ze echter niet weergeven omdat ze veel ruimte in beslag nemen.

Dit was het dan voor mijn resultaten. Ik heb alle 10 verzoeken van mijn backend in mijn frontend gebruikt, en alles werkt zoals ik wilde dat het zou werken.

Conclusie

Ik heb in totaal 130 uur in mijn graduaatsproef gestoken, waarbij ik in eerste instantie 35 uur heb besteed aan het leren van de taal en het uitvoeren van oefenprojecten, gevolgd door 65 uur aan de ontwikkeling van de backend en daarna heb ik nog eens 30 uur tijd in de frontend gestoken.

Gedurende dit proces heb ik heel erg veel geleerd. In het begin van mijn leerproces had ik gemengde gevoelens over Java, ondanks mijn grote interesse in de taal. Echter, na een tijdje werken met Java vond ik het uiteindelijk een zeer leuke taal om mee te werken. Deze ervaring heeft mijn zelfvertrouwen voor mijn bachelor toegepaste informatica enorm vergroot. Hoewel React niet de focus was van mijn graduaatsproef, was dit project ook mijn eerste grote project in React. Dit heeft mij ook enorm veel bijgeleerd.

Het is mij uiteindelijk gelukt om een succesvolle full-stack applicatie te creëren. Ik ben heel erg tevreden met het project dat ik heb gemaakt en heb er enorm veel plezier aan beleefd. Ik kijk uit naar verdere uitdagingen en leermogelijkheden in de toekomst op het gebied van Java.

Bronnen

In dit verslag heb ik als enige bron mijn kennis en het project zelf gebruikt.

Maar om mijn project te ontwikkelen en alles over Java te leren, heb ik gebruik gemaakt van de volgende bronnen:

- youtube.com (Ik heb talloze tutorials en cursussen op YouTube gevolgd voor uitleg over verschillende onderwerpen.)
- reddit.com (Wanneer ik vastliep op complexe problemen waarvoor ik geen directe oplossing kon vinden, zocht ik vaak naar antwoorden op Reddit.)
- chat.openai.com (ChatGPT heeft me enorm geholpen door dingen die ik niet goed begreep uit te leggen en te verduidelijken.)