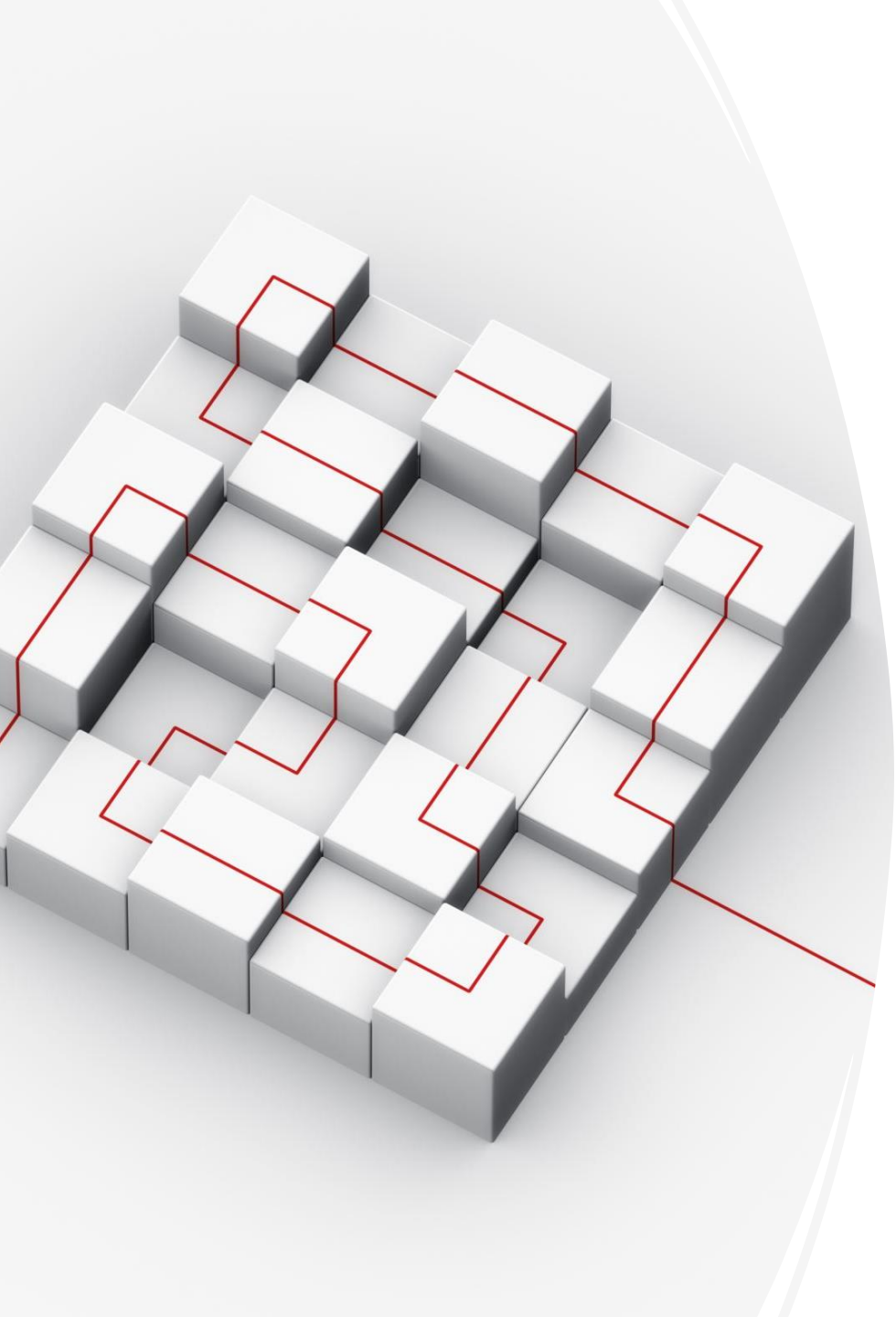
INT 161

Basic Backend Development

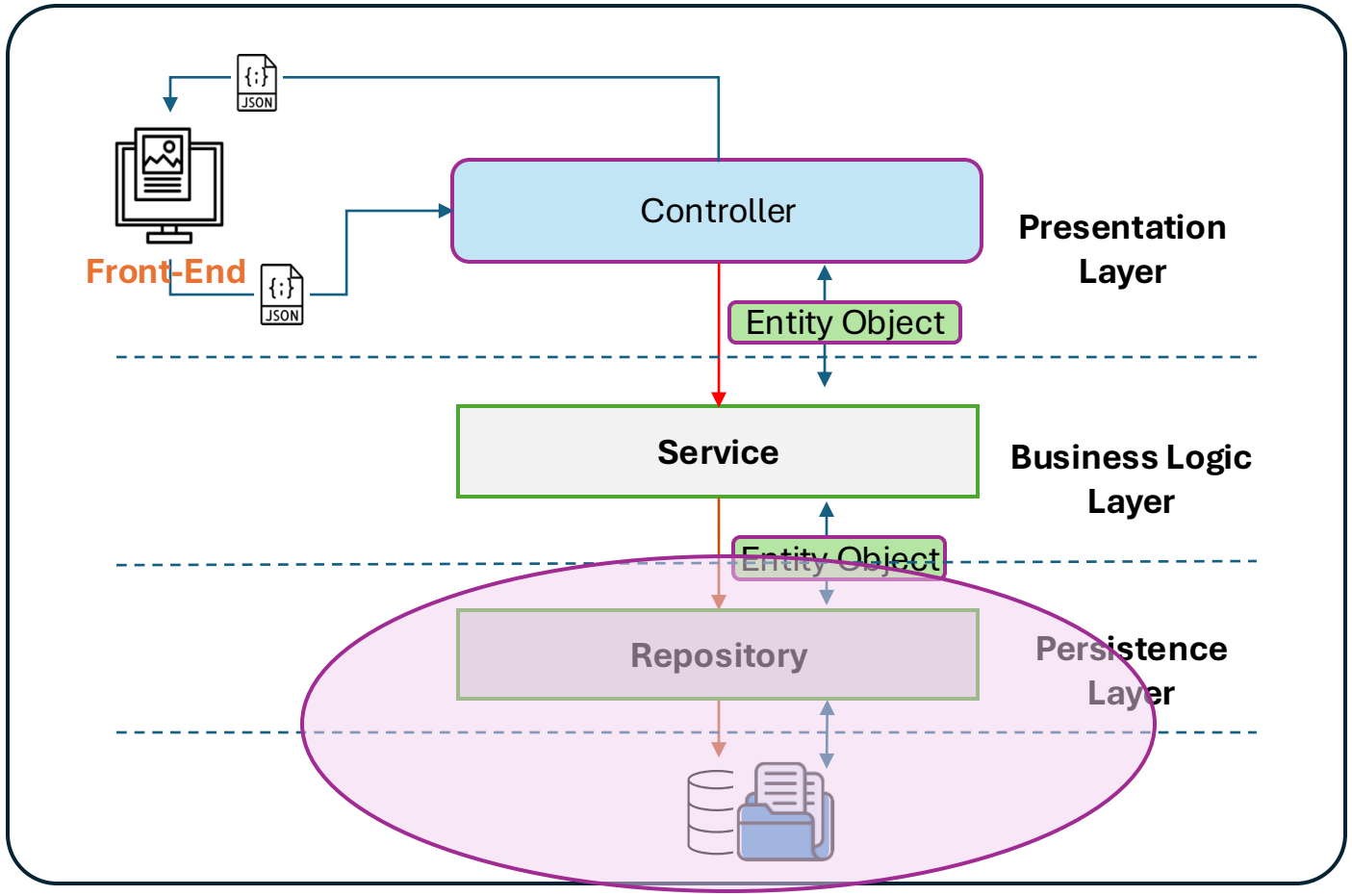
DATA MODELING



Unit Objectives

- After completing this unit, you should be able to:
  - Understand basic concept of Sorting and Filtering
  - Using Prisma Data Modelling for Sorting and Filtering

Layered System

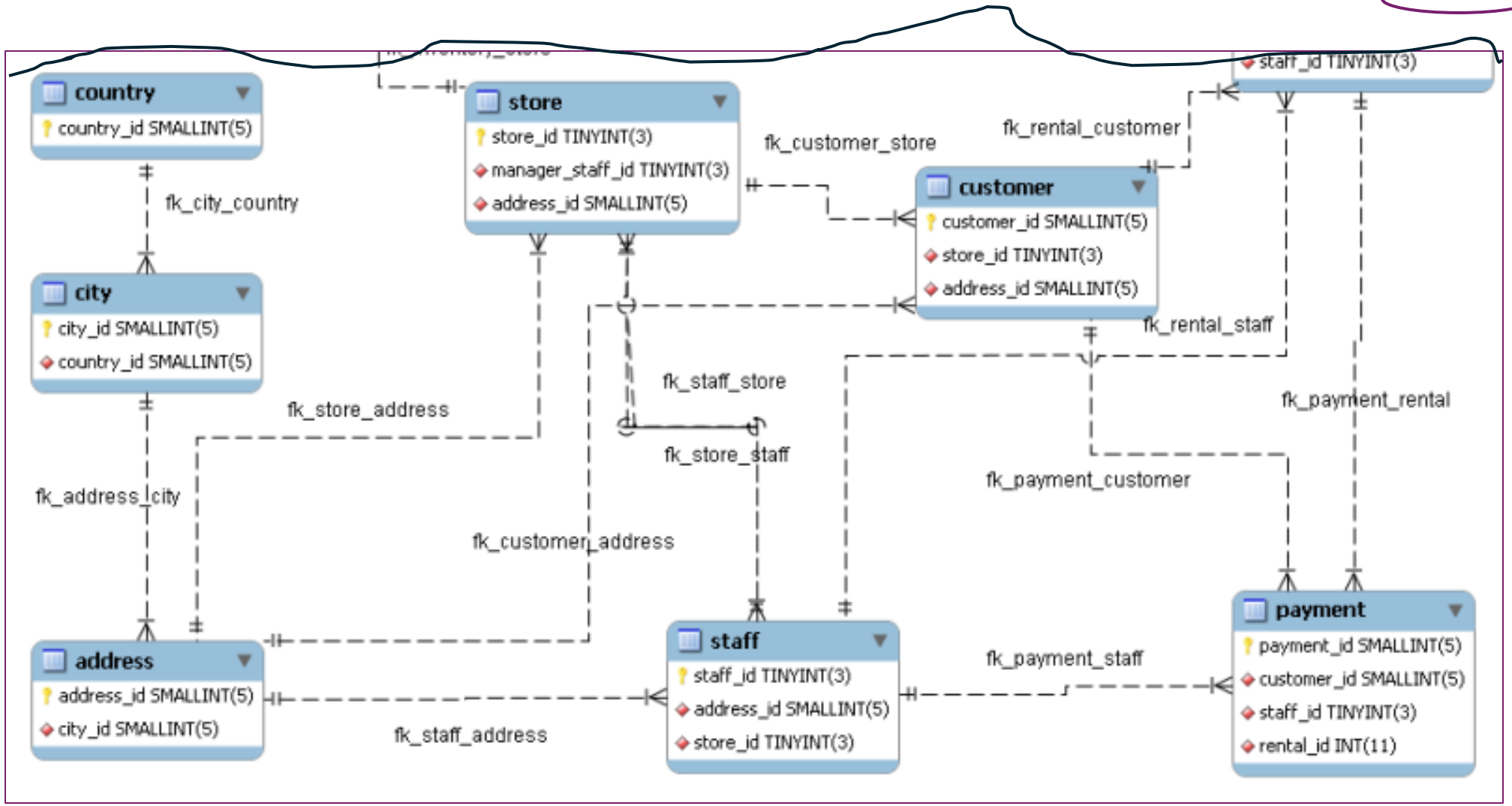


The **Layered System** principle means that a REST API is designed as a set of layers, where each layer has a specific role, and a client does not need to know whether it's communicating directly with the end server or through intermediaries.

This allows **scalability, flexibility, and separation of concerns**.

https://dev.mysql.com/doc/sakila/en/sakila-structure.html

Sakila



https://dev.mysql.com/doc/sakila/en/sakila-structure.html

film_actor
actor_id SMALLINT(5)
film_id SMALLINT(5)

film_category
film_id SMALLINT(5)
category_id TINYINT(3)

fk_film_category_category

category
category_id TINYINT(3)

fk_film_actor_actor

fk_film_actor_film

fk_film_category_film

fk_film_language_original
fk_film_language

language
language_id TINYINT(3)

actor
actor_id SMALLINT(5)

film
film_id SMALLINT(5)
language_id TINYINT(3)
original_language_id TINYINT(3)

fk_inventory_film

film_text
film_id SMALLINT(6)

inventory
inventory_id MEDIUMINT(8)
film_id SMALLINT(5)
store_id TINYINT(3)

fk_rental_inventory

rental
rental_id INT(11)
inventory_id MEDIUMINT(8)
customer_id SMALLINT(5)
staff_id TINYINT(3)

fk_inventory_store

Prisma CRUD summary - Read

Method	Description	Example

findUnique()	Fetches a single, unique record by a unique identifier, like id or email.	`await prisma.user.findUnique({`
		`  where: { id: 1, }`
		`});`

findFirst()	Fetches the first record matching the search criteria.	`await prisma.user.findFirst({`
		`  where: { name: 'Alice', }`
		`});`

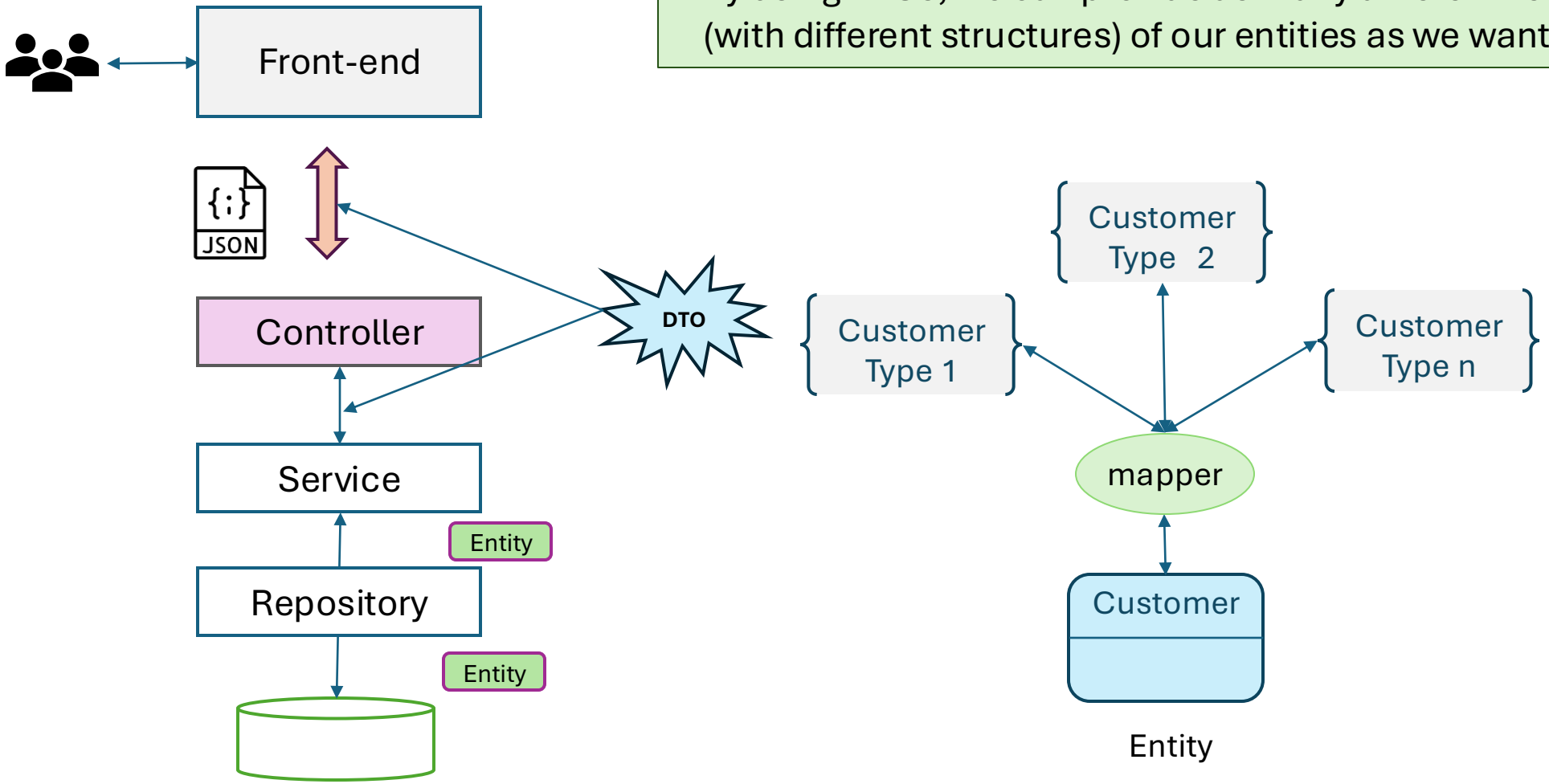
findMany()	Fetches all records matching the search criteria.	`// Find all users`
		`await prisma.user.findMany();`
		`// Find users with a specific name await`
		`prisma.user.findMany({`
		`  where: { name: 'Alice', }`
		`});`

Prisma Data Modelling (1:m, m:1 relationship)

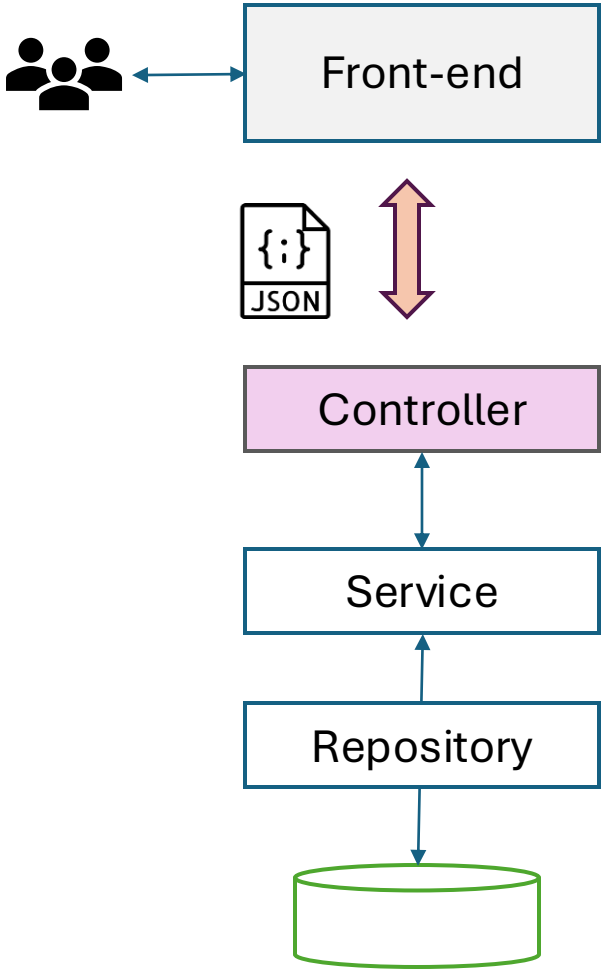
```
model City {
  id          Int       @id @default(autoincrement()) @db.UnsignedSmallInt @map("city_id")
  city        String    @db.VarChar(50)
  countryId   Int       @map("country_id") @db.UnsignedSmallInt
  lastUpdate  DateTime  @default(now()) @db.Timestamp(0) @map("last_update")
  addresses   Address[]
  country     Country   @relation(fields: [countryId], references: [id]
                        , map: "fk_city_country")

  @@index([id], map: "idx_fk_country_id")
  @@map("city")
}

model Country {
  id          Int       @id @default(autoincrement()) @db.UnsignedSmallInt @map("country_id")
  country     String    @db.VarChar(50)
  lastUpdate  DateTime  @default(now()) @map("last_update") @db.Timestamp(0)
  cities      City[]

  @@map("country")
}
```

DTO: Data Transfer Object

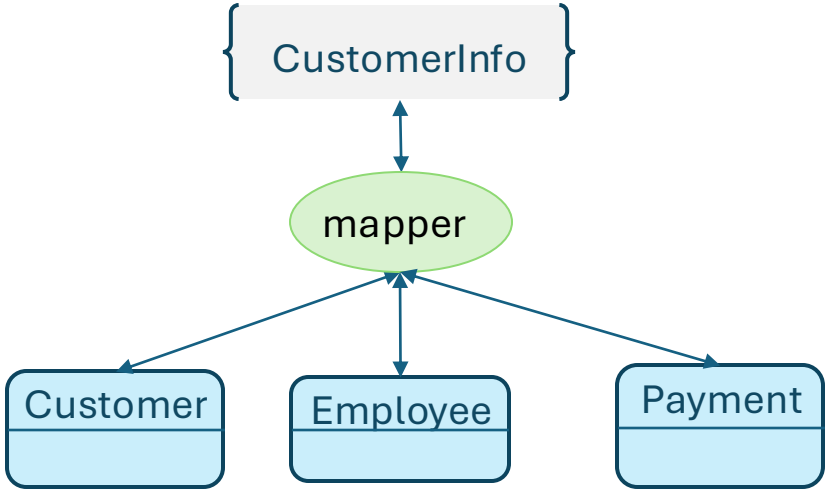
- The data is mapped from the domain models to the DTOs.
- By using DTOs, we can provide as many different versions (with different structures) of our entities as we want.



DTO: Data Transfer Object



DTO is a design pattern conceived to reduce the number of calls when working with remote interfaces.



Another advantage of using DTOs on RESTful APIs is that they can help hiding implementation details of domain objects (aka. entities). Exposing entities through endpoints can become a security issue if we do not carefully handle what properties can be changed through what operations.

DTO: Example

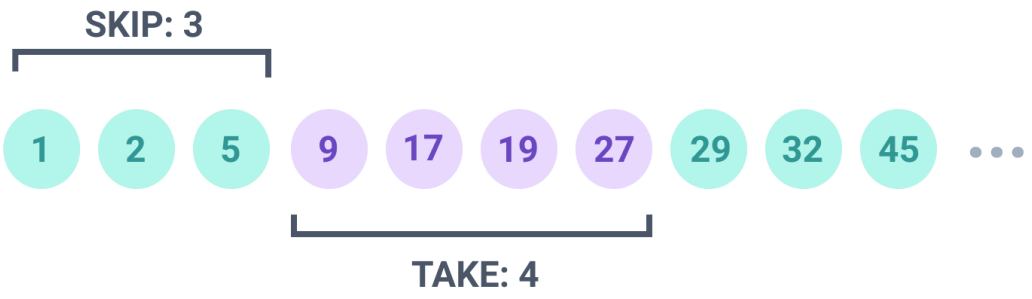
```
const uniqueOne = await service.getById(id);
res.json(new SimpleCustomerDto(uniqueOne));
```

```
class SimpleCustomerDto {
  constructor(customer = {}) {
    const { id, firstName, lastName, email, address, active } = customer;
    this.id = id ?? null;
    this.name = [firstName, lastName].filter(Boolean).join(" ") || null;
    this.email = email ?? null;
    this.active = active ?? false;
    if (address) {
      this.address = {
        id: address.id ? address.id : null,
        address: address.address ? `${address.address} ${address.address2}`.trim() : null,
        district: address.district ? address.district : '-',
        city: address.city && address.city.city ? address.city.city : null,
        postalCode: address.postalCode ? address.postalCode : '-',
        country: address.city && address.city.country ? address.city.country.country : null,
        phone: address.phone ? address.phone : '-',
      }
    }
  }
}
module.exports = SimpleCustomerDto;
```

Prisma Pagination

- Prisma Client supports both offset pagination and cursor-based pagination.
- Offset pagination
  - Offset pagination uses skip and take to skip a certain number of results and select a limited range.
  - The following query skips the first 3 Customer records and returns records 4 - 7:

```
const results = await prisma.customer.findMany({
  skip: 3,
  take: 4,
})
```



- To implement pages of results, you would just skip the number of pages multiplied by the number of results you show per page.

Offset pagination Example

```
const data = await
prisma.customer.findMany({
    skip: (page - 1) * pageSize,
    take: pageSize,
    orderBy: sortBy,
    include: {
        address: includeAddress ? {
            include: {
                city: {
                    include: {
                        country: true
                    }
                },
            }
        } : false,
    },
```

page	pageSize	skip	take
1	10	0	1-10
2	10	10	11-20
3	10	20	21-30
9	10	80	81-90

Film Data (1)

```
model Actor {
  actor_id    Int          @id @default(autoincrement()) @db.UnsignedSmallInt
  first_name  String       @db.VarChar(45)
  last_name   String       @db.VarChar(45)
  last_update DateTime     @default(now()) @db.Timestamp(0)
  film_actor  FilmActor[]

  @@index([last_name], map: "idx_actor_last_name")
}

model Category {
  category_id   Int            @id @default(autoincrement()) @db.UnsignedTinyInt
  name          String         @db.VarChar(25)
  last_update   DateTime       @default(now()) @db.Timestamp(0)
  film_category FilmCategory[]
}
```

Film Data (2)

```
model Film {
  id                    Int            @id @default(autoincrement())
                                       @db.UnsignedSmallInt @map("film_id")
  title                 String         @db.VarChar(128)
  description           String?        @db.Text
  release_year          Int?           @db.Year
  language_id           Int            @db.UnsignedTinyInt
  original_language_id  Int?           @db.UnsignedTinyInt
  rental_duration       Int            @default(3) @db.UnsignedTinyInt
  rental_rate           Decimal        @default(4.99) @db.Decimal(4, 2)
  length                Int?           @db.UnsignedSmallInt
  replacement_cost      Decimal        @default(19.99) @db.Decimal(5, 2)
  rating                FilmRating?    @default(G)
  special_features      String?
  last_update           DateTime       @default(now()) @db.Timestamp(0)
  film_actor            FilmActor[]
  film_category         FilmCategory[]

  @@index([language_id], map: "idx_fk_language_id")
  @@index([original_language_id], map: "idx_fk_original_language_id")
  @@index([title], map: "idx_title")
}
```

```
enum FilmRating {
  G
  PG
  PG_13 @map("PG-13")
  R
  NC_17 @map("NC-17")
}
```

Film Data (3)

```
model FilmActor {
  actor_id    Int      @db.UnsignedSmallInt
  film_id     Int      @db.UnsignedSmallInt
  last_update DateTime @default(now()) @db.Timestamp(0)
  actor       Actor    @relation(fields: [actor_id], references: [actor_id],
                        map: "fk_film_actor_actor")
  film        Film     @relation(fields: [film_id], references: [id],
                        map: "fk_film_actor_film")


  @@id([actor_id, film_id])
  @@index([film_id], map: "idx_fk_film_id")
  @@map("film_actor")
}
```

Film Data (4)

```
model FilmCategory {
  film_id     Int      @db.UnsignedSmallInt
  category_id Int      @db.UnsignedTinyInt
  last_update DateTime @default(now()) @db.Timestamp(0)
  category    Category @relation(fields: [category_id], references: [category_id],
                        map: "fk_film_category_category")
  film        Film     @relation(fields: [film_id], references: [id],
                       map: "fk_film_category_film")


  @@id([film_id, category_id])
  @@index([category_id], map: "fk_film_category_category")
  @@map("film_category")
}
```

Prisma: Filtering and Sorting

- Prisma Client supports:
  - Filtering with the where query option,
  - Sorting with the orderBy query option.

String Filters

- equals → ค่าตรงกัน
- contains → มีข้อความที่กำหนดอยู่ภายใน
- startsWith → ขึ้นต้นด้วย
- endsWith → ลงท้ายด้วย
- in → อยู่ใน list
- notIn → ไม่อยู่ใน list
- mode → ระบุการเปรียบเทียบ case (default หรือ "insensitive")
  - ใช้กับ MySql ที่ไม่ได้ setting ให้รองรับ case sensitive อาจจะมี error เนื่องจาก MySql มี mode เป็น insensitive โดย default

String Filters: example

```
const result = await prisma.user.findMany({
  where: {
    OR: [
      { email: {endsWith: 'gmail.com’,} , },
      { email: { endsWith: 'company.com' } },
    ],
    NOT: {
      email: {
        endsWith: 'admin.company.com’,
      },
    },
  },
})
```

Filter for non-null fields

The following query returns all posts whose `content` field is **not** `null`:

```
const posts = await prisma.post.findMany({
  where: {
    content: { not: null },
  },
})
```

Filter on null fields

The following query returns all posts whose `content` field is `null`:

```
const posts = await prisma.post.findMany({
  where: {
    content: null,
  },
})
```

Number / BigInt / Decimal / DateTime Filters

- equals
- in / notIn
- lt (Less than)
- lte (Less than or equals)
- gt (Greater than)
- gte (Greater than or equals)
- not

```
const users = await prisma.user.findMany({
  where: {
    id: {
      notIn: [1, 2, 3]
    }
  }
});
```

```
const users = await prisma.user.findMany({
  where: {
    email: {
      notIn: ["test@example.com", "admin@example.com"]
    }
  }
});
```

Boolean / Relation Filters

- Boolean
  - equals: true | false
- Relation Filters
  - some → มีบาง record ที่ match
  - every → ทุก record ต้อง match
  - none → ไม่มี record ไหน match

```
const users = await prisma.user.findMany({
  where: {
    email: { endsWith: "@gmail.com" },
    age: { gte: 18 },
    posts: { some: { published: true } }
  }
});
```

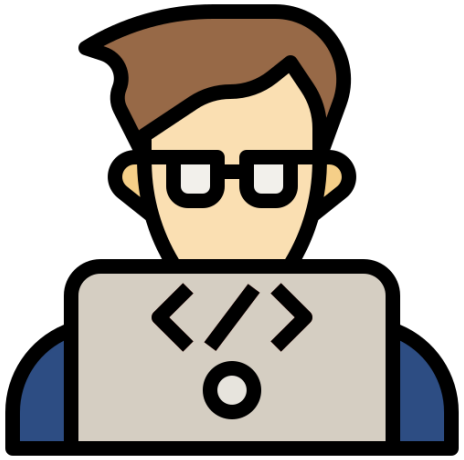
มี อย่างน้อย 1 โพสต์ ที่ published

Sorting

- Use orderBy to sort a list of records or a nested list of records by a particular field or set of fields.

```
const usersWithPosts = await prisma.user.findMany({
 orderBy: [
   { role: 'desc'},
   { name: 'desc'},
 ],
 include: {
   posts: {
     orderBy: {
       title: 'desc',
     },
   },
 } ,
})
```

Practices



Film Repository : `film-repository.js`

```
const {PrismaClient} = require("../generated/prisma");
const prisma = new PrismaClient();
module.exports = {
   findAll: async function (includeActor = false, pageRequet = {page: 1, pageSize: 10}) {
      const {page, pageSize} = pageRequet;
      const totalItems = await prisma.film.count();
      const data = await prisma.film.findMany({
         skip: (page - 1) * pageSize,
         take: pageSize,
         include: {
            film_actor: includeActor ? {include: {actor: true}} : false,
         },
      });
      return {
         data: data,
         page: page,
         pageSize: pageSize,
         totalItems: totalItems,
         totalPages: Math.ceil(totalItems / pageSize),
      }
   },
```

```
findById: async function (fid) {
      return await prisma.film.findUnique({
         where: {id: fid},
         include: {
            film_actor: {include: {actor: true}},
            film_category: {include: {category: true}},
         },});},
}
```

Film Server : film-service.js

```
const repo = require('../repositories/film-repository');

module.exports = {
    getAll: async function (includeActor = false, pageRequest = {}) {
        const results = await repo.findAll(includeActor, pageRequest);
        return results;
    },
    getById: async function (id) {
        const uniqueOne = await repo.findById(id);
        if (!uniqueOne) {
            const err = new Error(`Film not found for ID ${id}`);
            err.code = 'NOT_FOUND';
            err.status = 404;
            throw err;
        }
        return uniqueOne;
    },
}
```

Film DTO (1/4) : `simple-film-dto.js`

```
class SimpleFilmDto {
  constructor(film = {}) {
    const {id, title, release_year, rating, film_actor, film_category} = film;
    this.id = id ?? null;
    this.title = title ?? '-';
    this.releaseYear = release_year ?? '-';
    this.rating = rating ?? '-';
    if (film_actor) {
      this.actors = film_actor.map(actor => {
        return {
          id: actor.actor_id,
          name: actor.actor.first_name.charAt(0) + actor.actor.first_name.slice(1).toLowerCase()
            + ' ' + actor.actor.last_name.charAt(0)+ actor.actor.last_name.slice(1).toLowerCase()
        }
      });
    }
```

Film DTO (2/4) : `simple-film-dto.js`

```
 if (film_category) {
        this.categories = film_category.map(category => {
           return {
              id: category.category_id,
              name: category.category.name,
           }
        })
     }
   }
}

module.exports = SimpleFilmDto;
```

Film DTO (3/4) : `film-detail-dto.js`

```
class FilmDetailDto {
  constructor(film = {}) {
    const {id, title, description, length, release_year, special_features,rating, film_actor, film_category} = film;
    this.id = id ?? null;
    this.title = title ?? '-';
    this.releaseYear = release_year ?? '-';
    this.rating = rating ?? '-';
    this.description = description ?? '-';
    this.length = length ?? '-';
    this.specialFeatures = special_features ?? '-’;
    if (film_actor) {
      this.actors = film_actor.map(actor => {
        return {
          id: actor.actor_id,
          name: actor.actor.first_name.charAt(0) + actor.actor.first_name.slice(1).toLowerCase()
            + ' ' + actor.actor.last_name.charAt(0) + actor.actor.last_name.slice(1).toLowerCase()
        }
      });
    }
```

Film DTO (4/4) : `film-detail-dto.js`

```
        if (film_category) {
            this.categories = film_category.map(category => {
                return {
                    id: category.category_id,
                    name: category.category.name,
                }
            })
        }
    }
}

module.exports = FilmDetailDto;
```

Film Controller (1/2): `film-controller.js`

```
const service = require('../services/film-service');
const FilmDetailDto = require('../dtos/film-detail-dto');
const SimpleFilmDto = require('../dtos/simple-film-dto’);
```

```
module.exports = {
  list: async function (req, res) {
    try {
      const includeActor = req.query.includeActor|| false;
      const {page, pageSize} = req.query;
      pageRequest = { page: Number(page) || 1, pageSize: Number(pageSize) || 10 };
      const pageFilm = await service.getAll(includeActor, pageRequest);
      const simpleFilms = pageFilm.data.map(film => new SimpleFilmDto(film));
      pageFilm.data = simpleFilms;
      res.json(pageFilm);
    } catch (e) {
      console.log(e, e.status);
      res.status(e.status||500).json({code: e.code, message: e.message, status: e.status});
    }
  },
```

Film Controller (2/2): `film-controller.js`

```
  get: async function (req, res) {
     const id = Number(req.params.id);
     try {
        const uniqueOne = await service.getById(id);
        res.json(new FilmDetailDto(uniqueOne));
     } catch (e) {
        res.status(e.status||500).json(
           {code: e.code, message: e.message, status: e.status});
     }
  },
}
```

Filter Film By Title (1) : `film-repository.js`

```
findAll: async function (includeActor = false,
                pageRequet = {page: 1, pageSize: 10}, filmTitle = null) {

  const {page, pageSize} = pageRequet;
  const totalItems = await prisma.film.count();

  const data = await prisma.film.findMany({
    skip: (page - 1) * pageSize,
    take: pageSize,
    include: {
        film_actor: includeActor ? {include: {actor: true}} : false,
    },
    where: filmTitle ? {title: {contains: filmTitle}} : {},
  });
```

Filter Film By Title : `film-controller.js`

```
list: async function (req, res) {
   try {
      const includeActor = req.query.includeActor|| false;
      const {page, pageSize} = req.query;

      const {filmTitle} = req.query;

      console.log(filters);
      pageRequest = {page: Number(page) || 1, pageSize: Number(pageSize) || 10};
      const pageFilm = await service.getAll(includeActor, pageRequest, filmTitle);
```

Filter Film By Title (2) : `film-repository.js`

```
findAll: async function (includeActor = false,
        pageRequet = {page: 1, pageSize: 10}, filmTitle = null) {

    const {page, pageSize} = pageRequet;
    const filmFiters = filmTitle ? {title: {contains: filmTitle}} : {}

    const totalItems = await prisma.film.count({where: filmFiters});

    const data = await prisma.film.findMany({
        skip: (page - 1) * pageSize,
        take: pageSize,
        include: {
            film_actor: includeActor ? {include: {actor: true}} : false,
        },
        where: filmFiters,
    });
```

Filter Film By Title & Rating : `film-controller.js`

```
list: async function (req, res) {
  try {
    const includeActor = req.query.includeActor|| false;
    const {page, pageSize} = req.query;
    const {filmTitle} = req.query;
    const filmRating = req.query.filmRating
      ? Array.isArray(req.query.filmRating)
        ? req.query.filmRating
        : [req.query.filmRating]
      : [];
    filters = {filmTitle, filmRating};
    console.log(filters);
    pageRequest = {page: Number(page) || 1, pageSize: Number(pageSize) || 10};
    const pageFilm = await service.getAll(includeActor, pageRequest, filters);
```

Filter Film By Title & Rating : `film-repository.js`

```
findAll: async function (includeActor = false, pageRequet = {page: 1, pageSize: 10},
                         filmFiters = {filmTitle: null, filmRating : []}) {
    const {page, pageSize} = pageRequet;
    const {filmTitle, filmRating} = filmFiters;
    const titleFiter = filmTitle ? {title: {contains: filmTitle}} : {}
    const ratingFilter = Object.keys(filmRating).length>0 ? {rating: {in: filmRating}}:null ;
    const filters = {...titleFiter?titleFiter: {} , ...(ratingFilter ? ratingFilter : {})};
    console.log(filters);
    const totalItems = await prisma.film.count({where: filters});
    const data = await prisma.film.findMany({
      skip: (page - 1) * pageSize,
      take: pageSize,
      include: {
        film_actor: includeActor ? {include: {actor: true}} : false,
    },
    where: filters,
});
```

Sorting Film By Title : `film-controller.js`

```
list: async function (req, res) {
   try {
      const sortBy = req.query.sortBy || null;
      const [key, value] = sortBy.split(":");
      const sortObj = { [key]: value };

      const includeActor = req.query.includeActor|| false;
      const {page, pageSize} = req.query;
      const {filmTitle} = req.query;
      const filmRating = req.query.filmRating ? Array.isArray(req.query.filmRating)
            ? req.query.filmRating : [req.query.filmRating] : [];
      filters = {filmTitle, filmRating, sortBy:sortObj};

console.log(filters);
console.log(sortObj);
```

```
const pageFilm = await service.getAll(includeActor, pageRequest, filters);
```

Sorting Film By Title : `film-repository.js`

```
findAll: async function (includeActor = false, pageRequet = {page: 1, pageSize: 10},
                         filmFiters = {filmTitle: null, filmRating : [],sortBy: null}) {
    :
    :
   const ratingFilter = Object.keys(filmRating).length>0 ?{rating: {in: filmRating}}:null ;

   const {sortBy} = filmFiters;    // <----

   const filters = {...titleFiter?titleFiter: {} , ...(ratingFilter ? ratingFilter : {})};
   console.log(filters);
   console.log('sort by:', sortBy);
```

```
const data = await prisma.film.findMany({
    :
    :
   where: filters,
   orderBy: sortBy,
});
```