DotNet 2021

ONLINE TECH CONFERENCE

#DotNet2021

22nd June 2021

Arquitectura (Hexagonal) con Typescript

en APIs web con Nodejs & Express

www.dotnet2021.com

DotNet 2021

ORGANIZATION

plain concepts

IN COOPERATION WITH

FUNDACIÓN GOMAESPUMA

"Educando con una sonrisa."

SPONSORS

Microsoft

intelequia

DevsDNA™

My Public Inbox®

#DotNet2021

Carlos Bastos Pérez-Cuadrado

Staff Software Engineer at eventbrite

- Developer since age 15
- 6 companies, > 20 projects/products
- Different sectors and roles
- Consultant, developer, software architect, trainer and manager

@cbastospc

1996

(POSA volumen 1) [1]

Presentation Layer

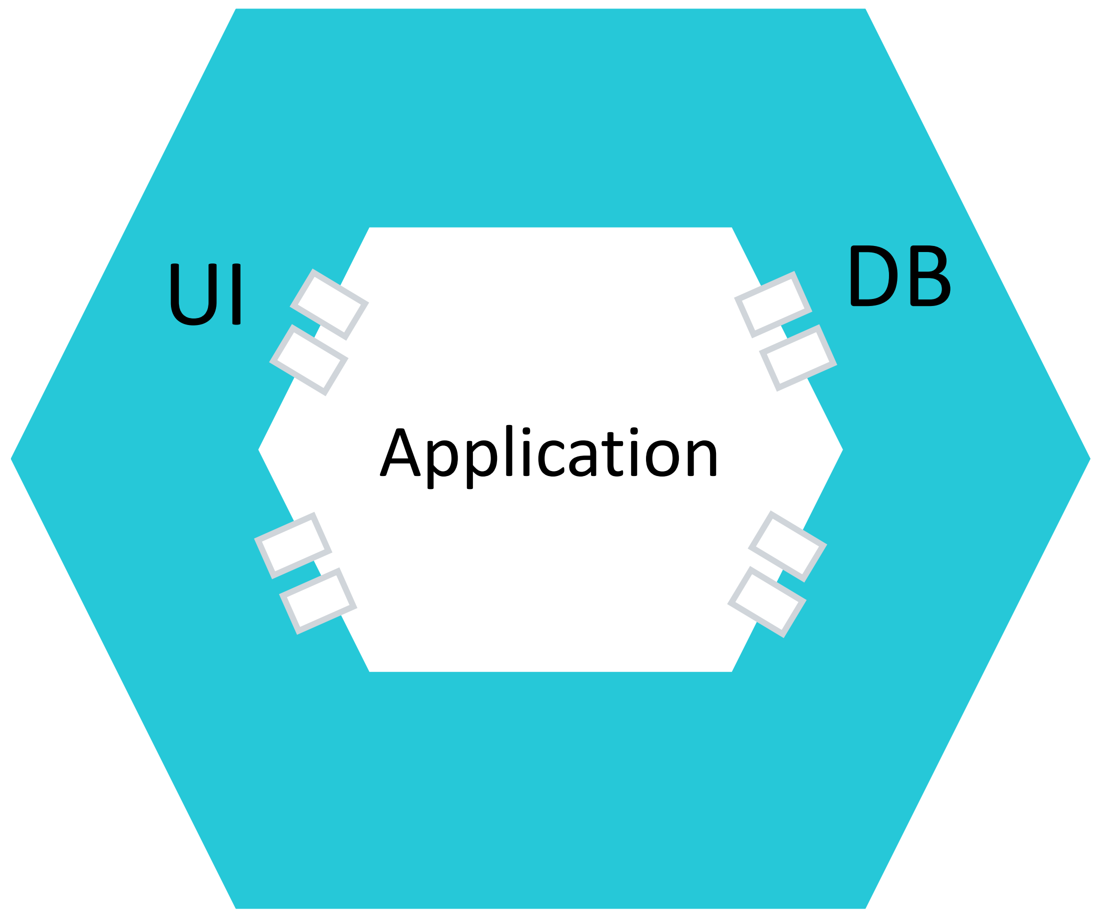
Business Layer

Data Access Layer

Layered Architecture
(Three layer)

2005

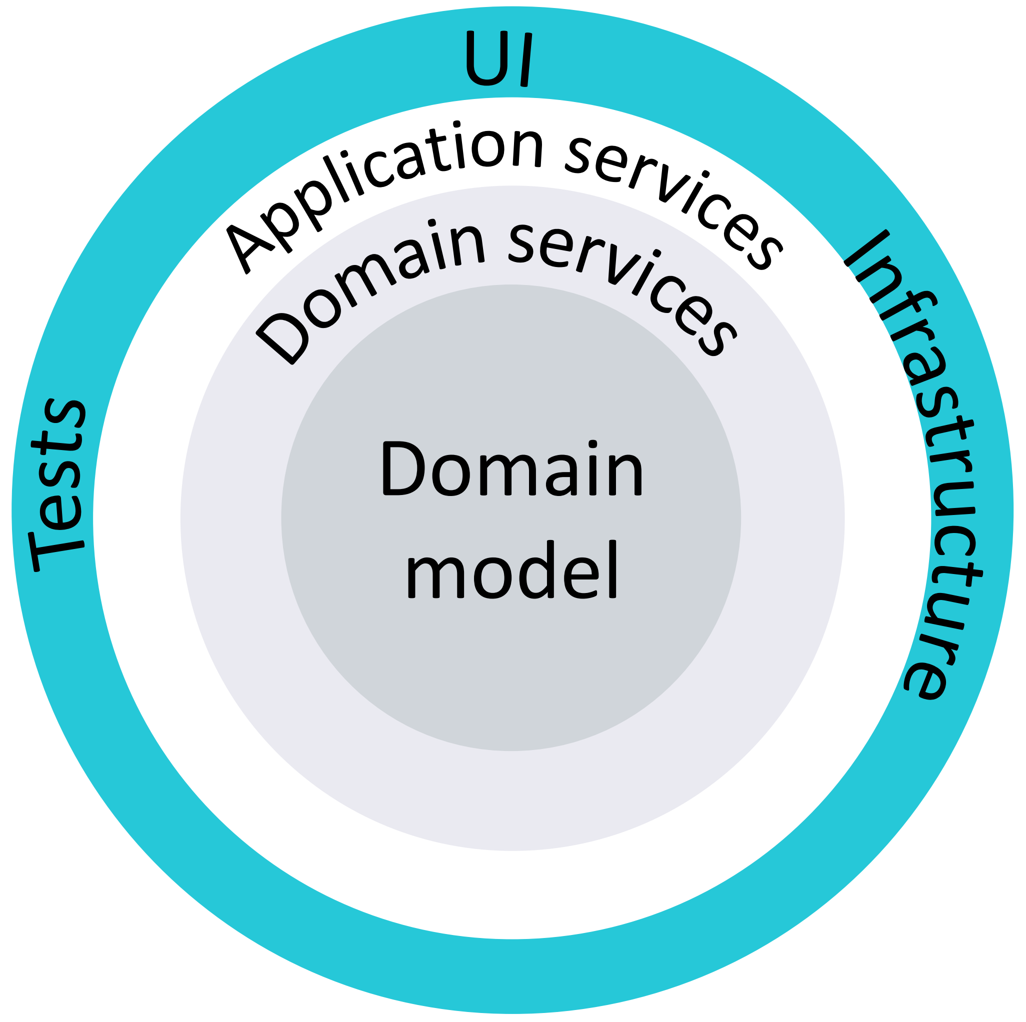
(Alistair Cockburn) [2]



Hexagonal Architecture
(or Ports & Adapters)

2008

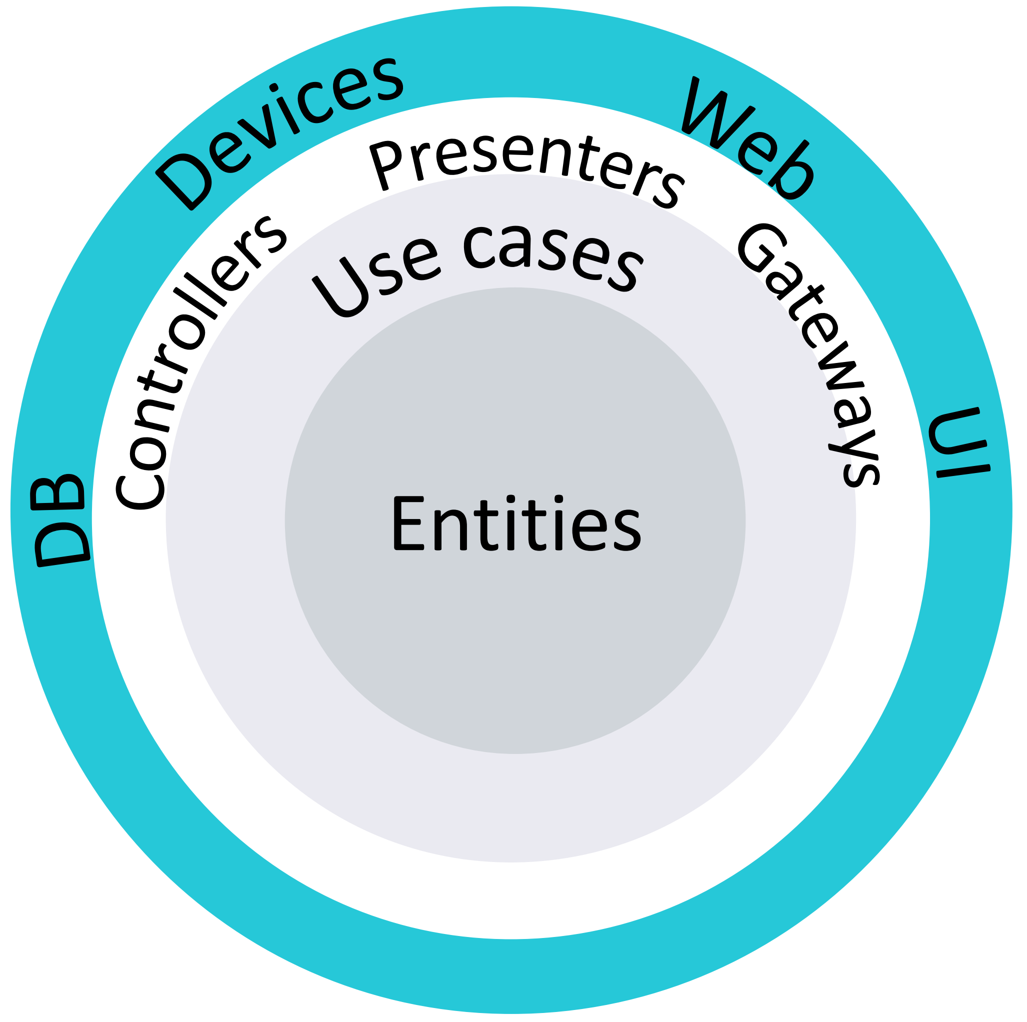
(Jeffrey Palermo) [3]



Onion Architecture

2012

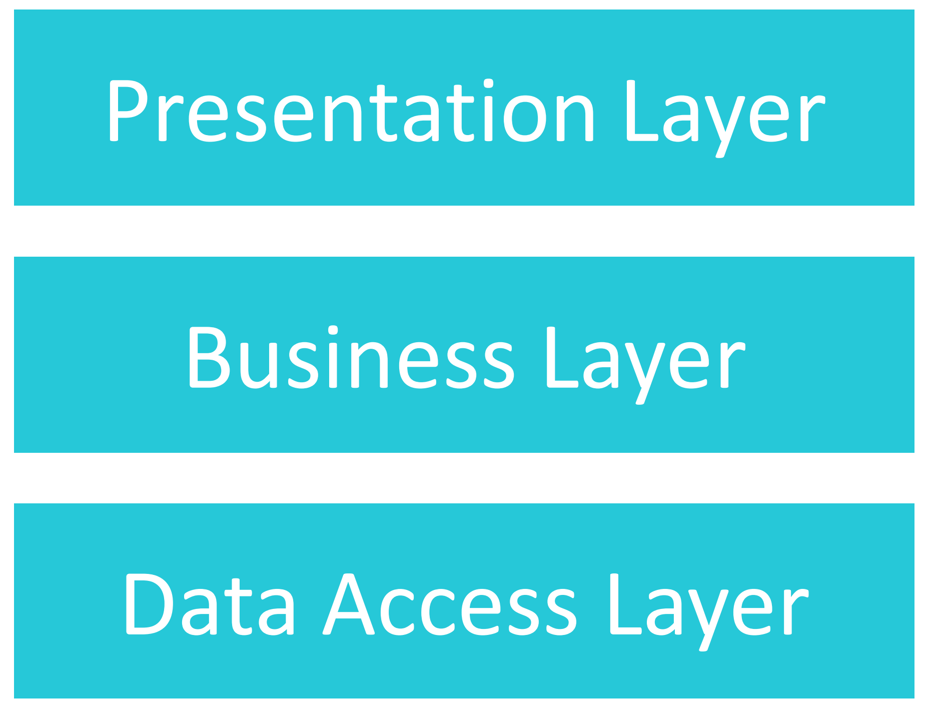
(Robert C. Martin) [4]



Clean architecture
(Hexagonal + Onion architectures)

1996

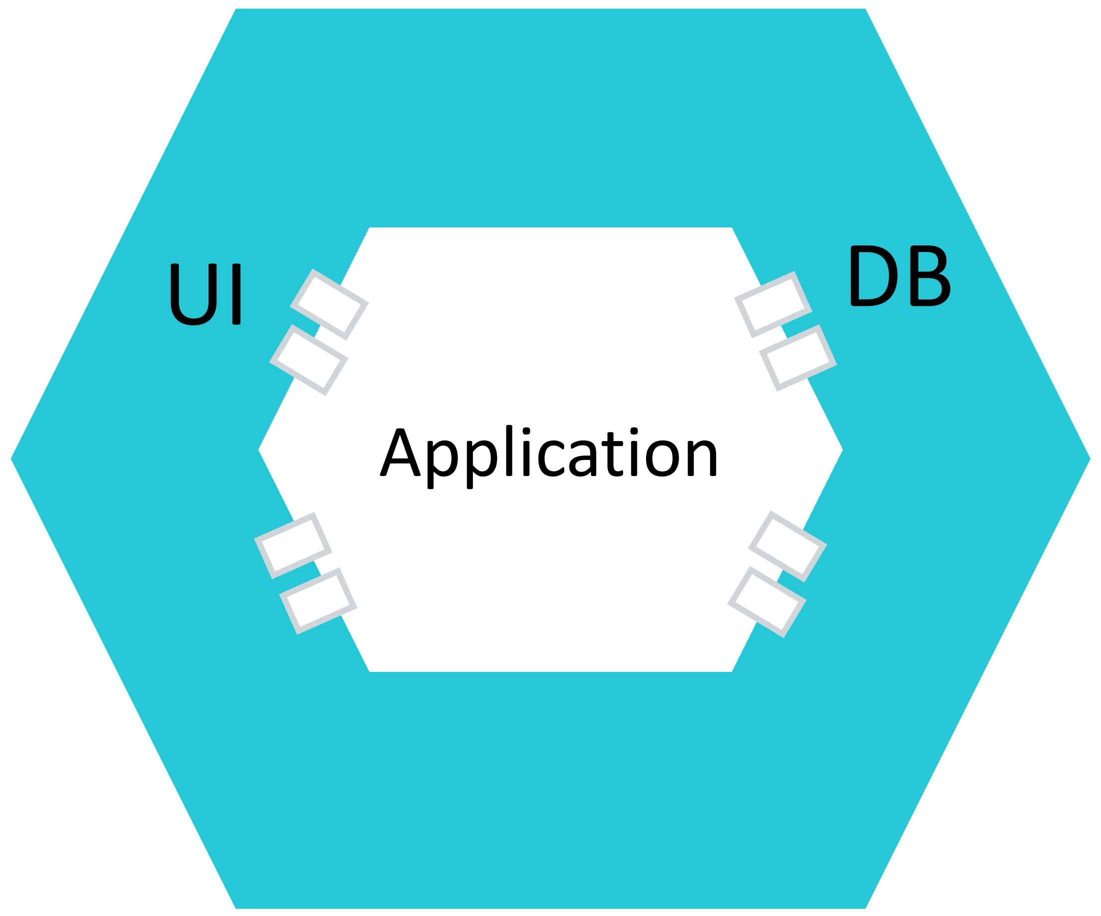
(POSA volumen 1) [1]



Layered Architecture
(Three layer)

2005

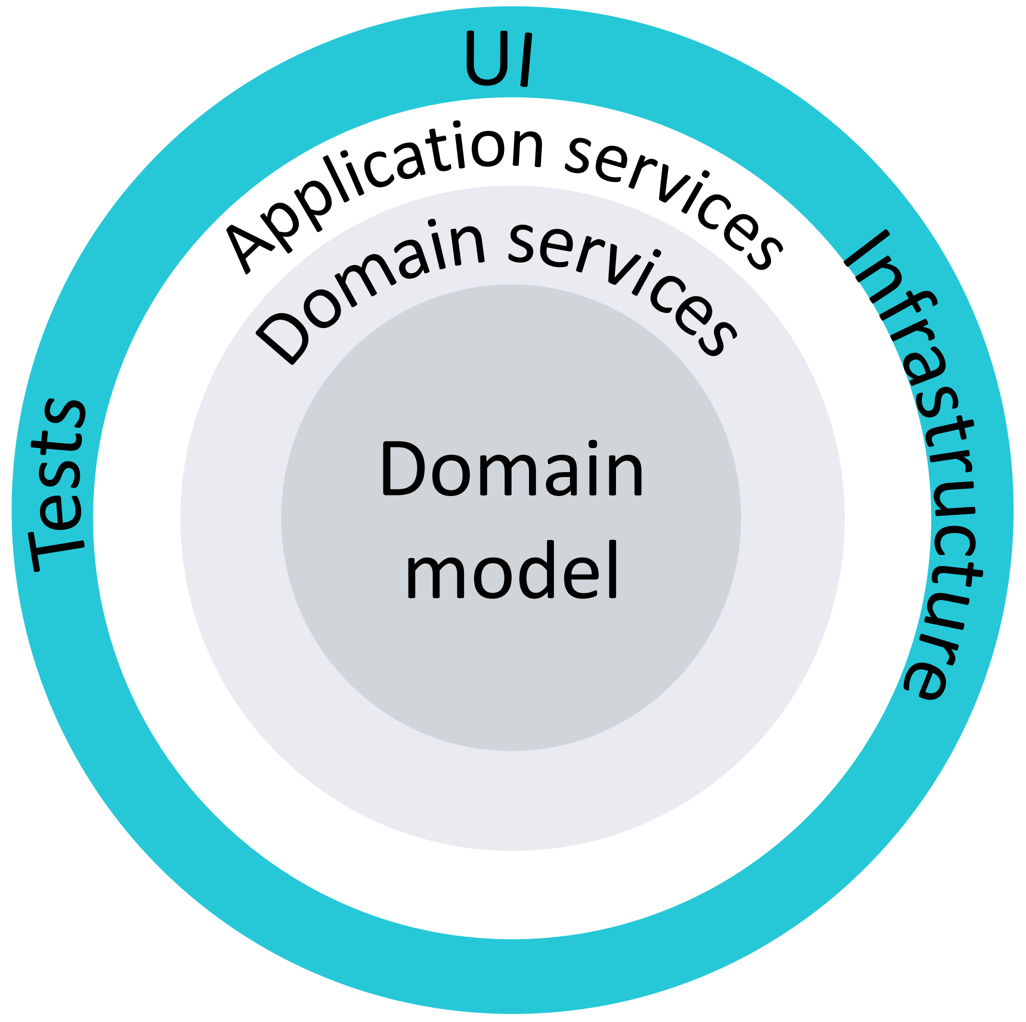
(Alistair Cockburn) [2]



Hexagonal Architecture
(or Ports & Adapters)

2008

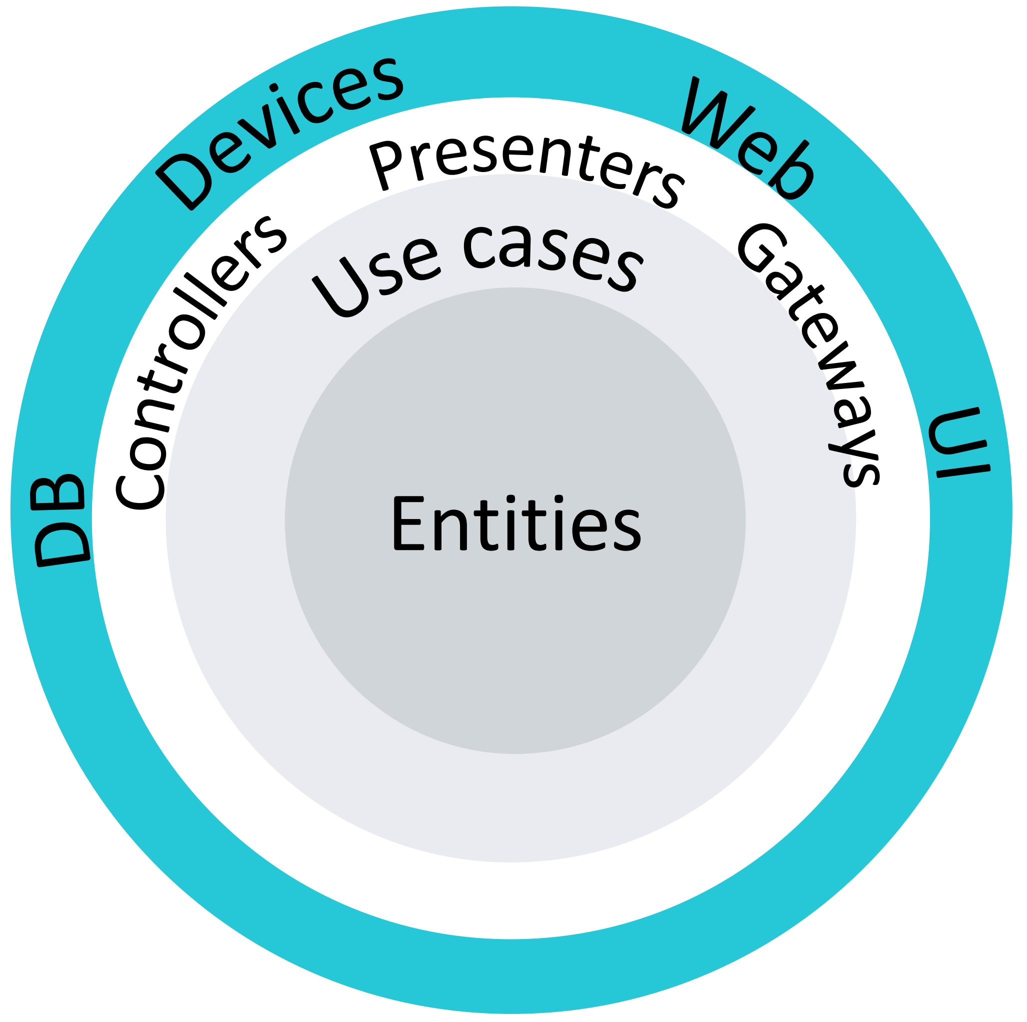
(Jeffrey Palermo) [3]



Onion Architecture

2012

(Robert C. Martin) [4]

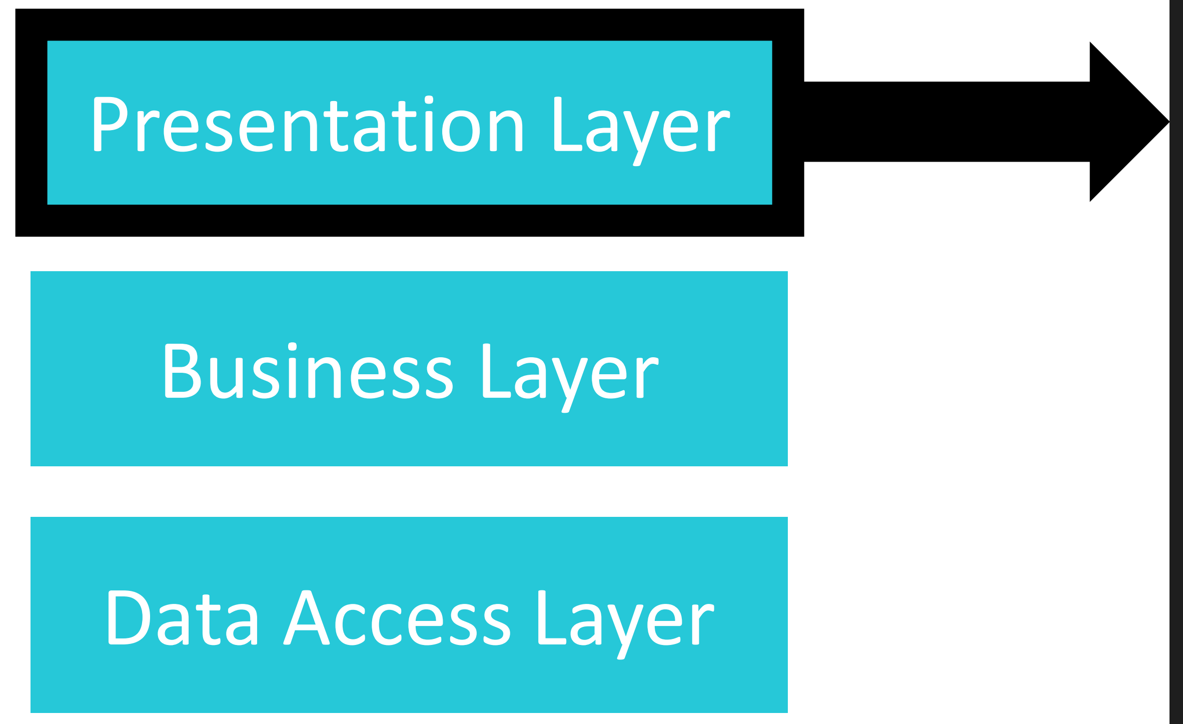


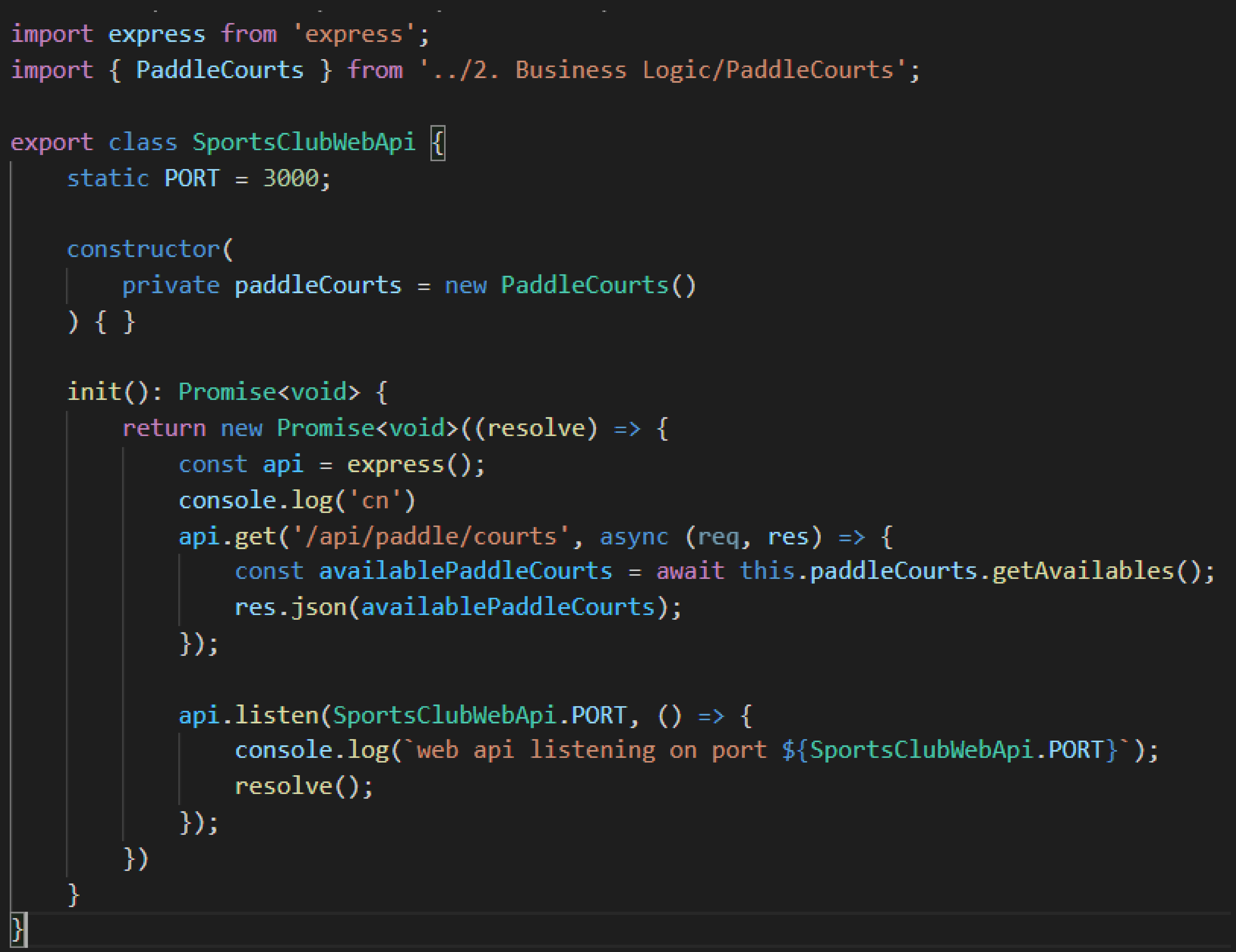
Clean architecture
(Hexagonal + Onion architectures)

DotNet2021

Presentation Layer

Layered Architecture (Three layer)



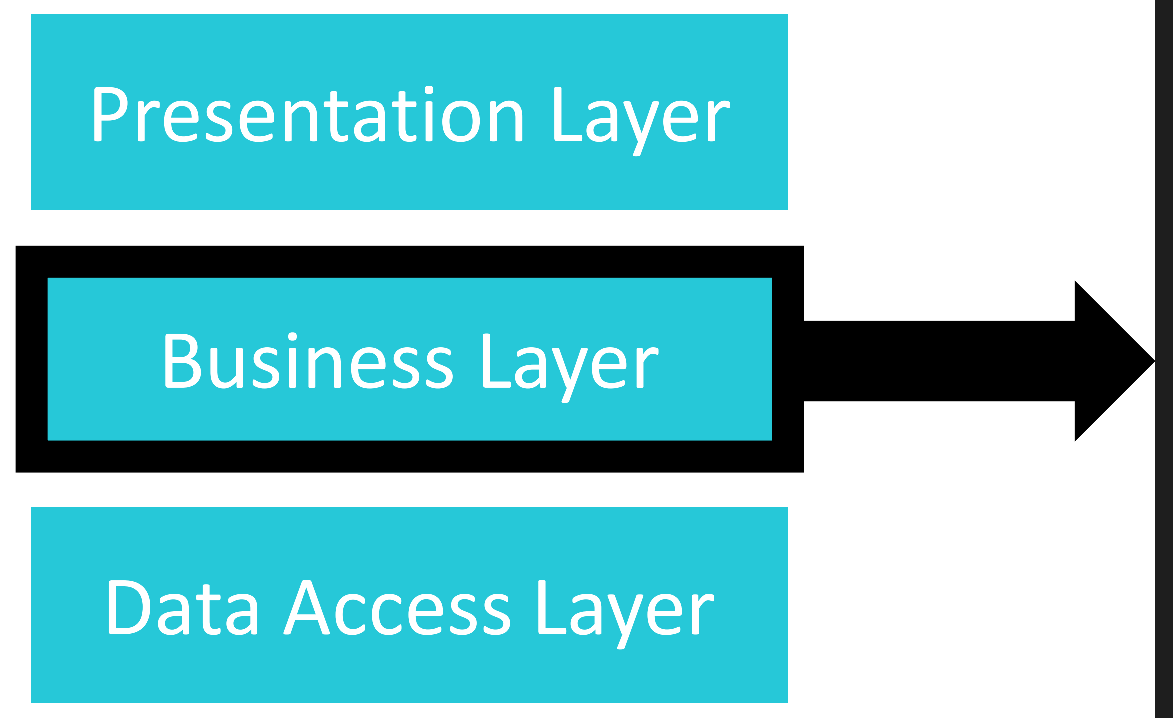


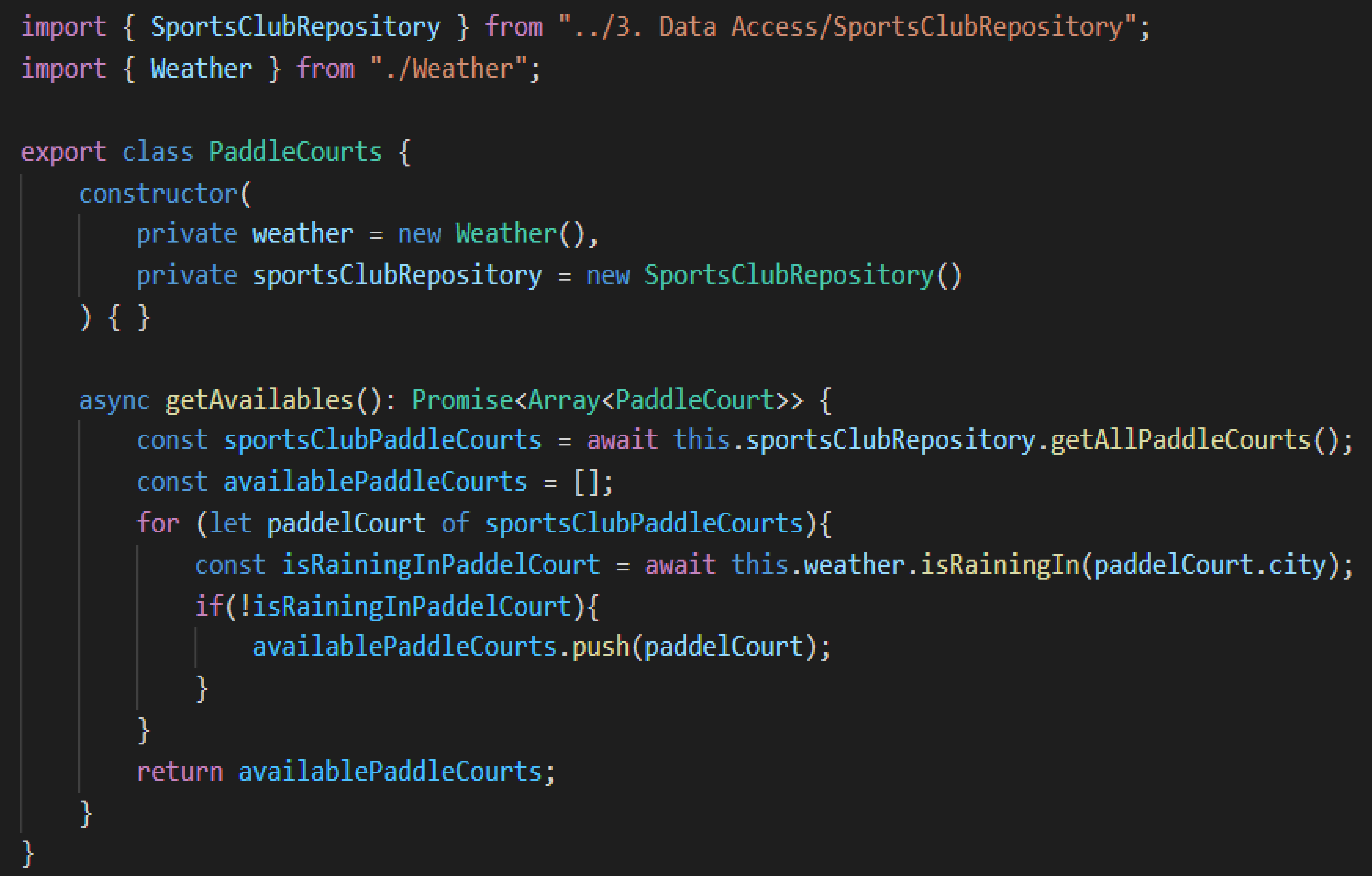
```
import express from 'express';
import { PaddleCourts } from '../2. Business Logic/PaddleCourts';

export class SportsClubWebApi {
    static PORT = 3000;

    constructor(
        private paddleCourts = new PaddleCourts()
    ) { }

    init(): Promise<void> {
        return new Promise<void>((resolve) => {
            const api = express();
            console.log('cn')
            api.get('/api/paddle/courts', async (req, res) => {
                const availablePaddleCourts = await this.paddleCourts.getAvailables();
                res.json(availablePaddleCourts);
            });

            api.listen(SportsClubWebApi.PORT, () => {
                console.log(`web api listening on port ${SportsClubWebApi.PORT}`);
                resolve();
            });
        })
    }
}
```

DotNet2021

Business Layer

Layered Architecture (Three layer)



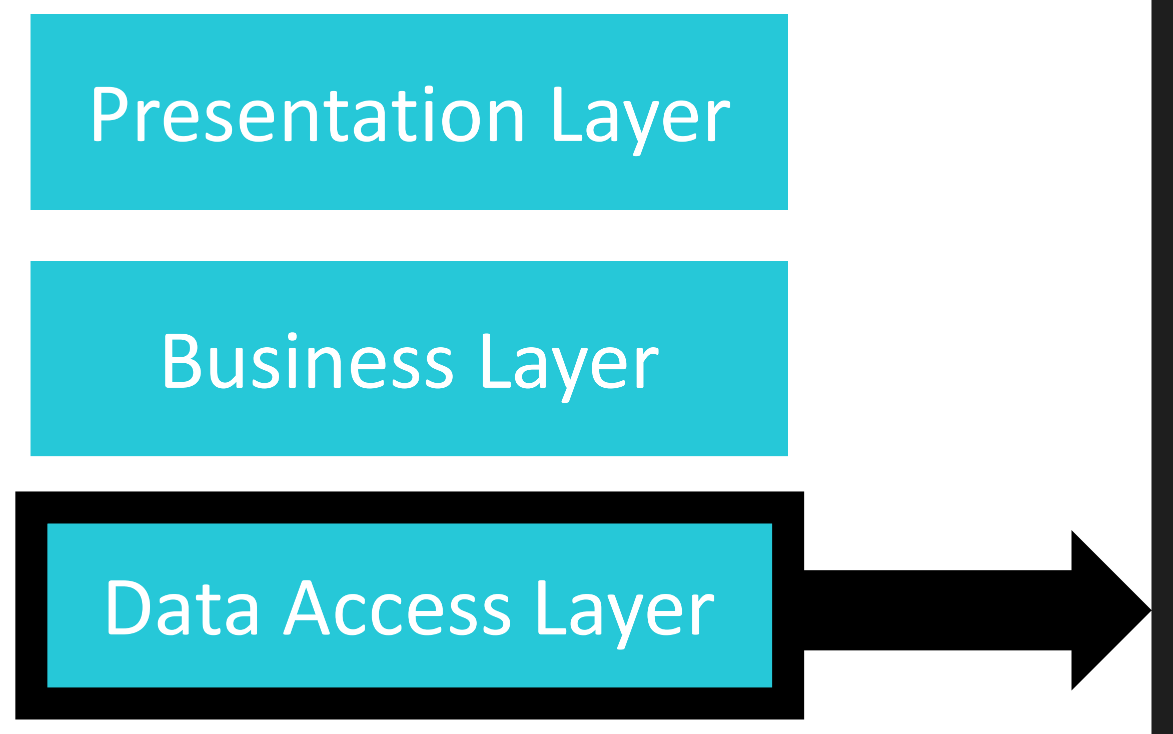


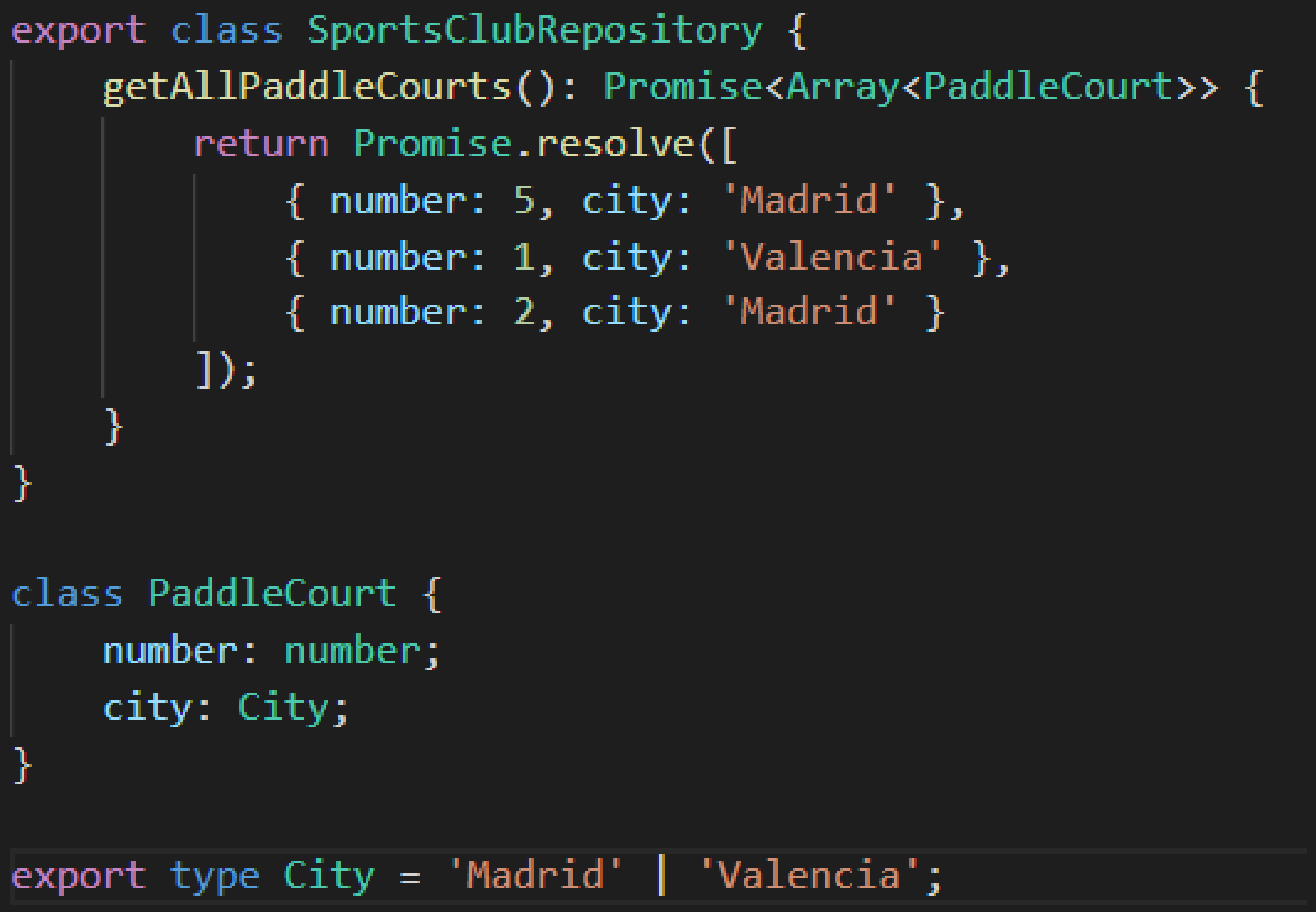
```
import { SportsClubRepository } from "../3. Data Access/SportsClubRepository";
import { Weather } from "./Weather";

export class PaddleCourts {
    constructor(
        private weather = new Weather(),
        private sportsClubRepository = new SportsClubRepository()
    ) { }

    async getAvailables(): Promise<Array<PaddleCourt>> {
        const sportsClubPaddleCourts = await this.sportsClubRepository.getAllPaddleCourts();
        const availablePaddleCourts = [];
        for (let paddelCourt of sportsClubPaddleCourts){
            const isRainingInPaddelCourt = await this.weather.isRainingIn(paddelCourt.city);
            if(!isRainingInPaddelCourt){
                availablePaddleCourts.push(paddelCourt);
            }
        }
        return availablePaddleCourts;
    }
}
```

Data Access Layer

DotNet2021

Layered Architecture (Three layer)





```
export class SportsClubRepository {
    getAllPaddleCourts(): Promise<Array<PaddleCourt>> {
        return Promise.resolve([
            { number: 5, city: 'Madrid' },
            { number: 1, city: 'Valencia' },
            { number: 2, city: 'Madrid' }
        ]);
    }
}

class PaddleCourt {
    number: number;
    city: City;
}

export type City = 'Madrid' | 'Valencia';
```

DotNet2021

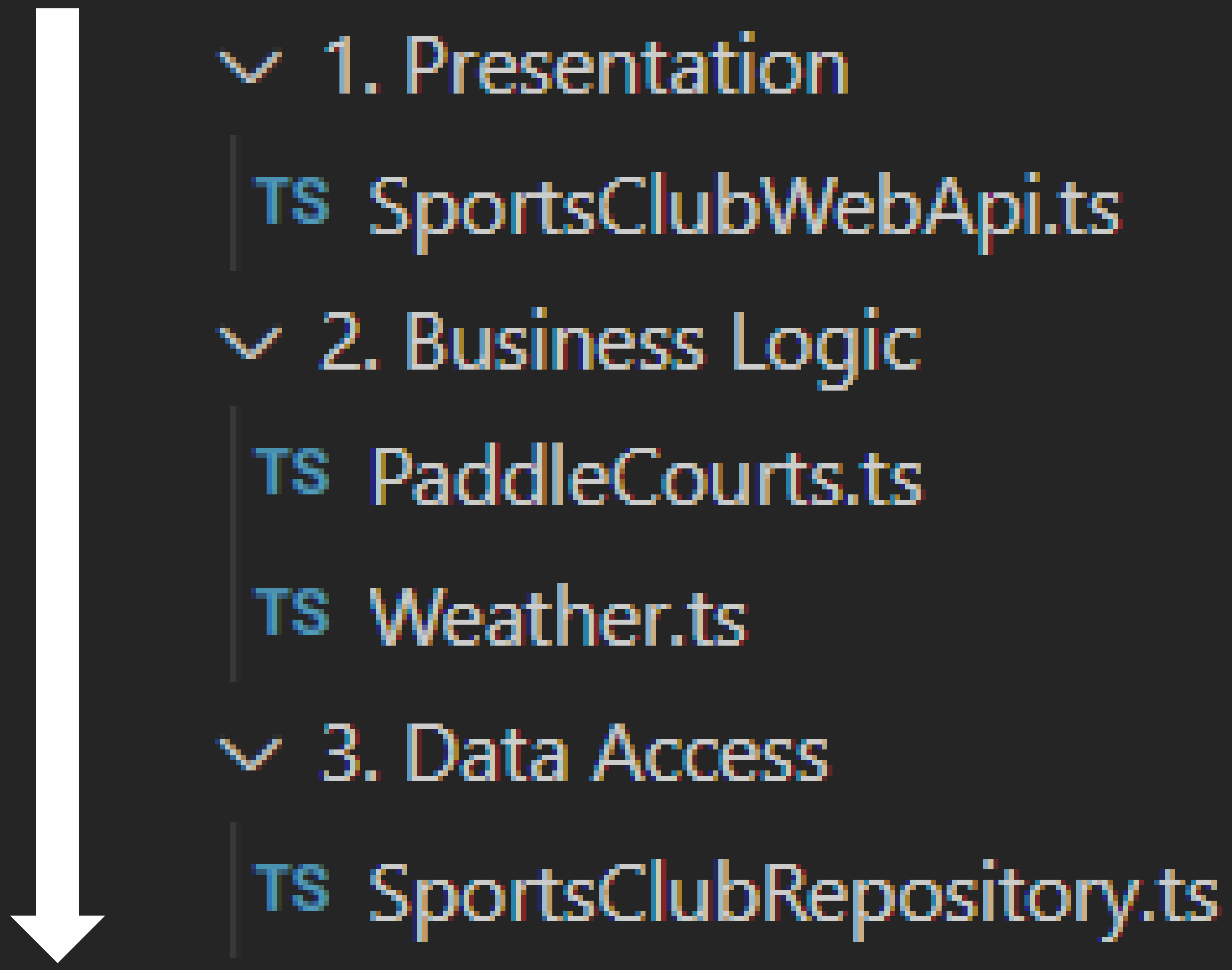
Dependency direction

Layered Architecture (Three layer)

Presentation Layer

Business Layer

Data Access Layer



```
⌄ 1. Presentation
    TS SportsClubWebApi.ts
⌄ 2. Business Logic
    TS PaddleCourts.ts
    TS Weather.ts
⌄ 3. Data Access
    TS SportsClubRepository.ts
```

1996

(POSA volumen 1) [1]

Presentation Layer

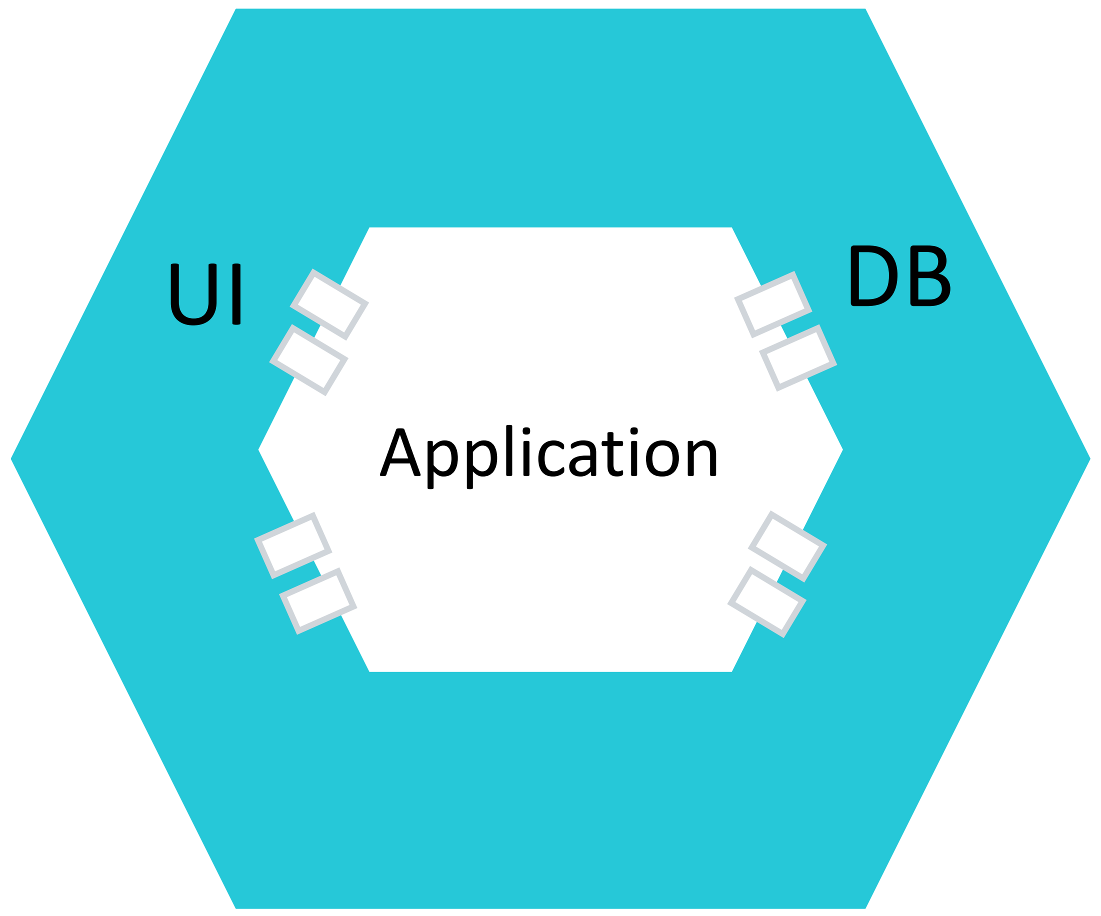
Business Layer

Data Access Layer

Layered Architecture
(Three layer)

2005

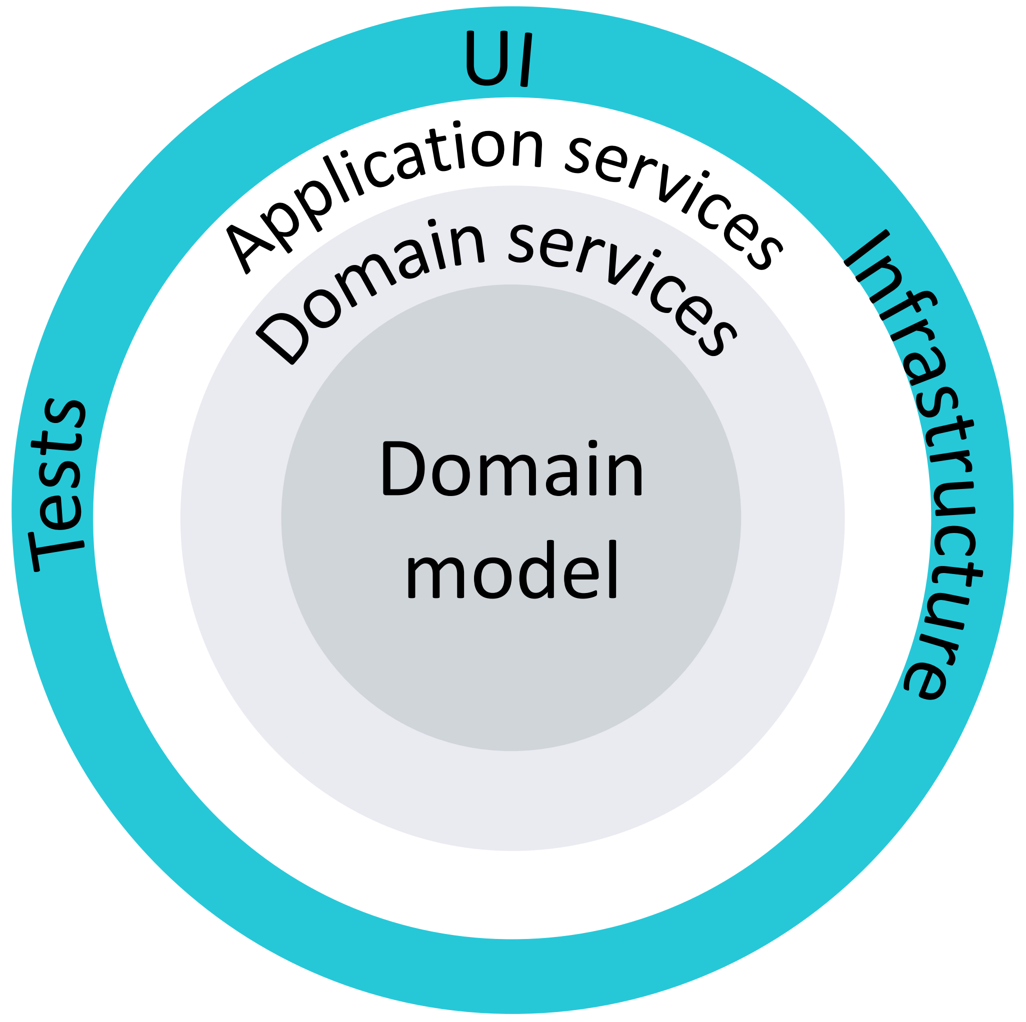
(Alistair Cockburn) [2]



Hexagonal Architecture
(or Ports & Adapters)

2008

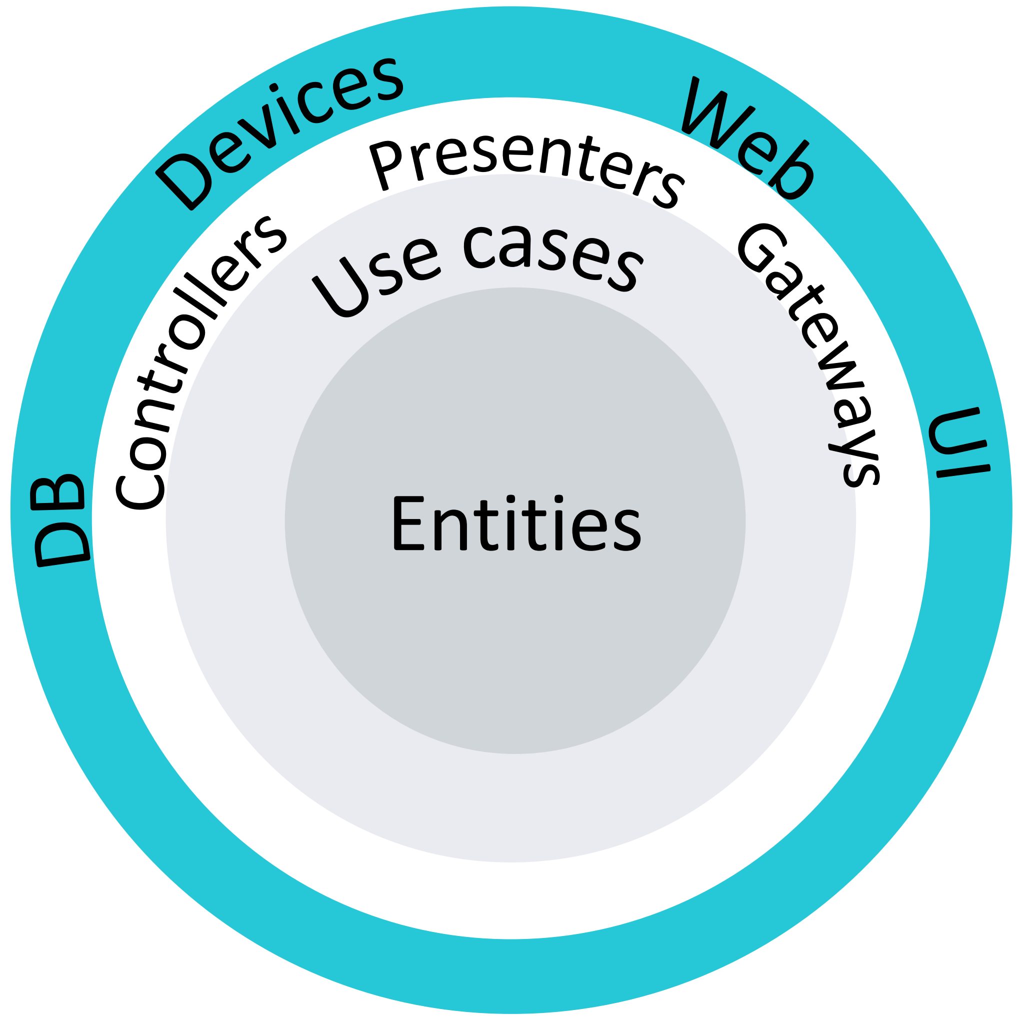
(Jeffrey Palermo) [3]



Onion Architecture

2012

(Robert C. Martin) [4]

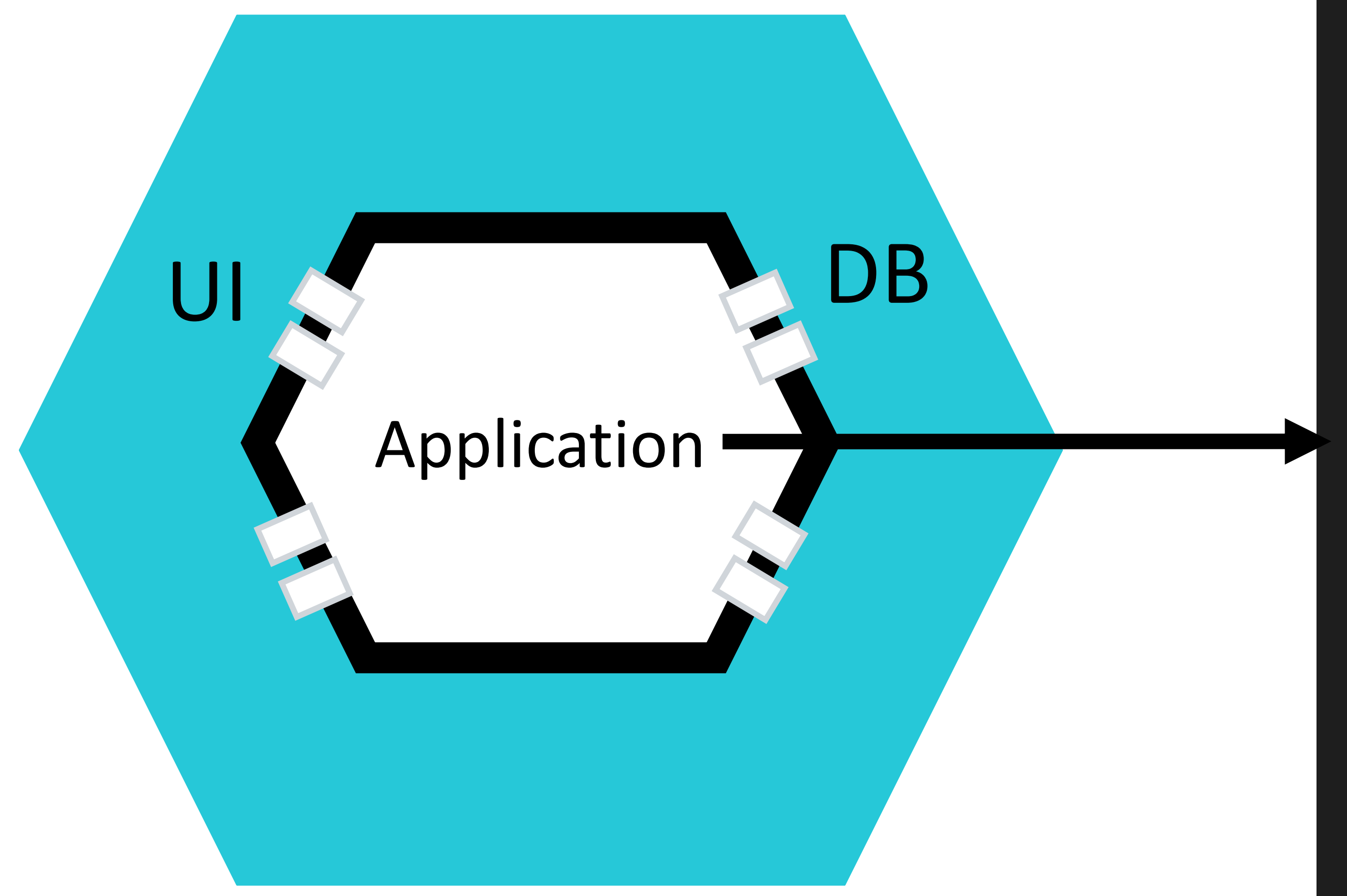


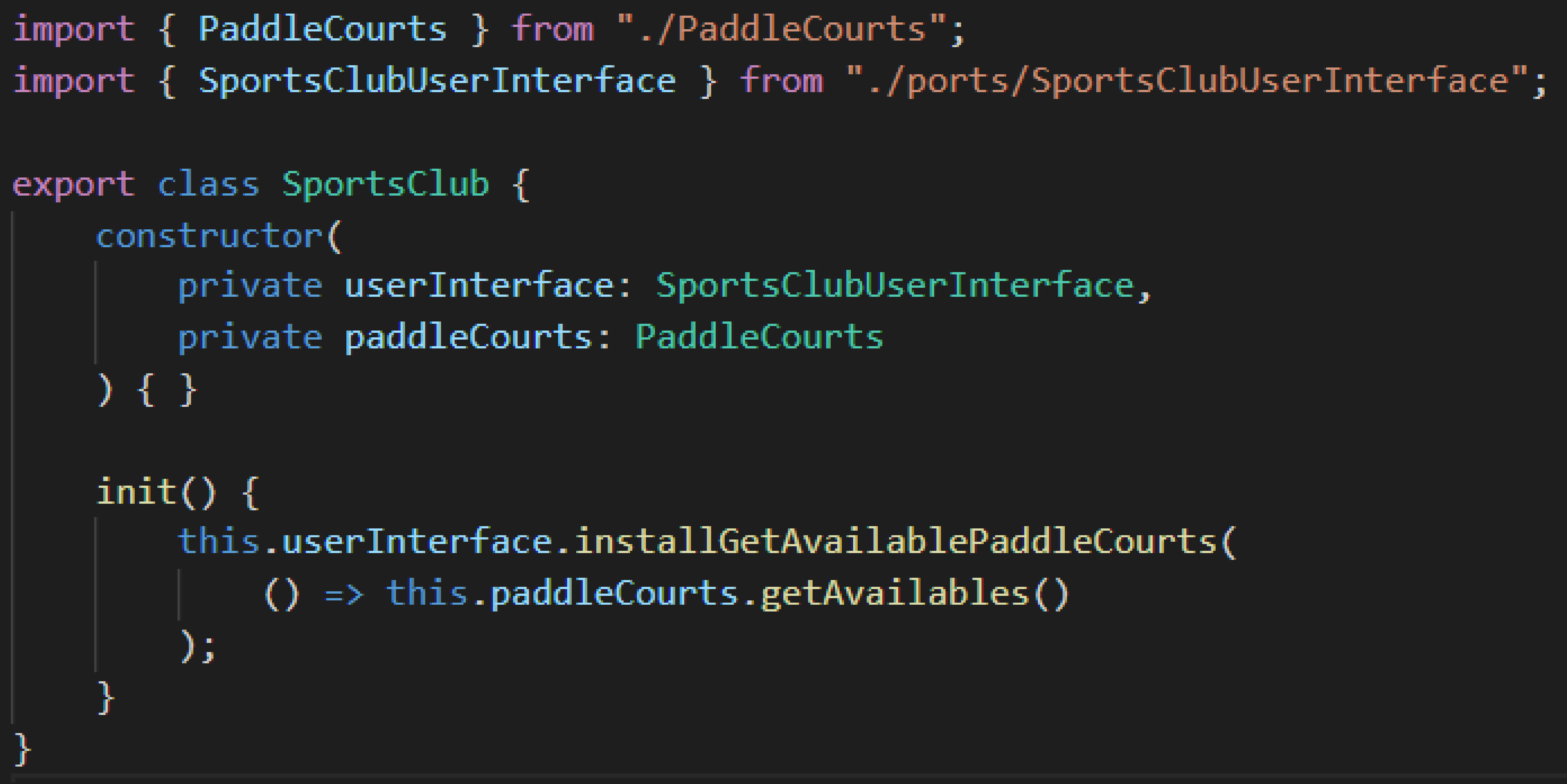
Clean architecture
(Hexagonal + Onion architectures)

Application

DotNet2021

Hexagonal Architecture
(or Ports & Adapters)

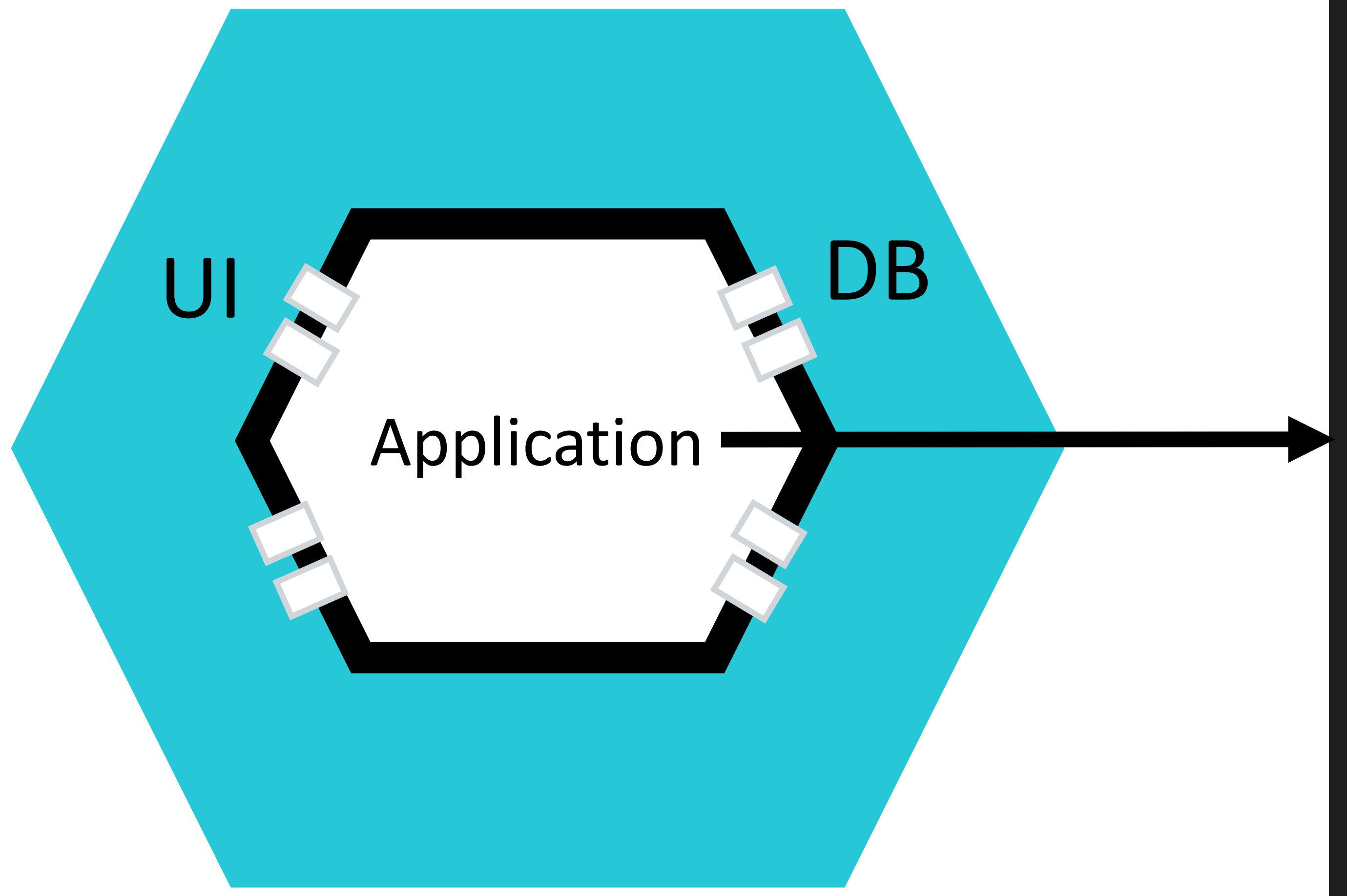




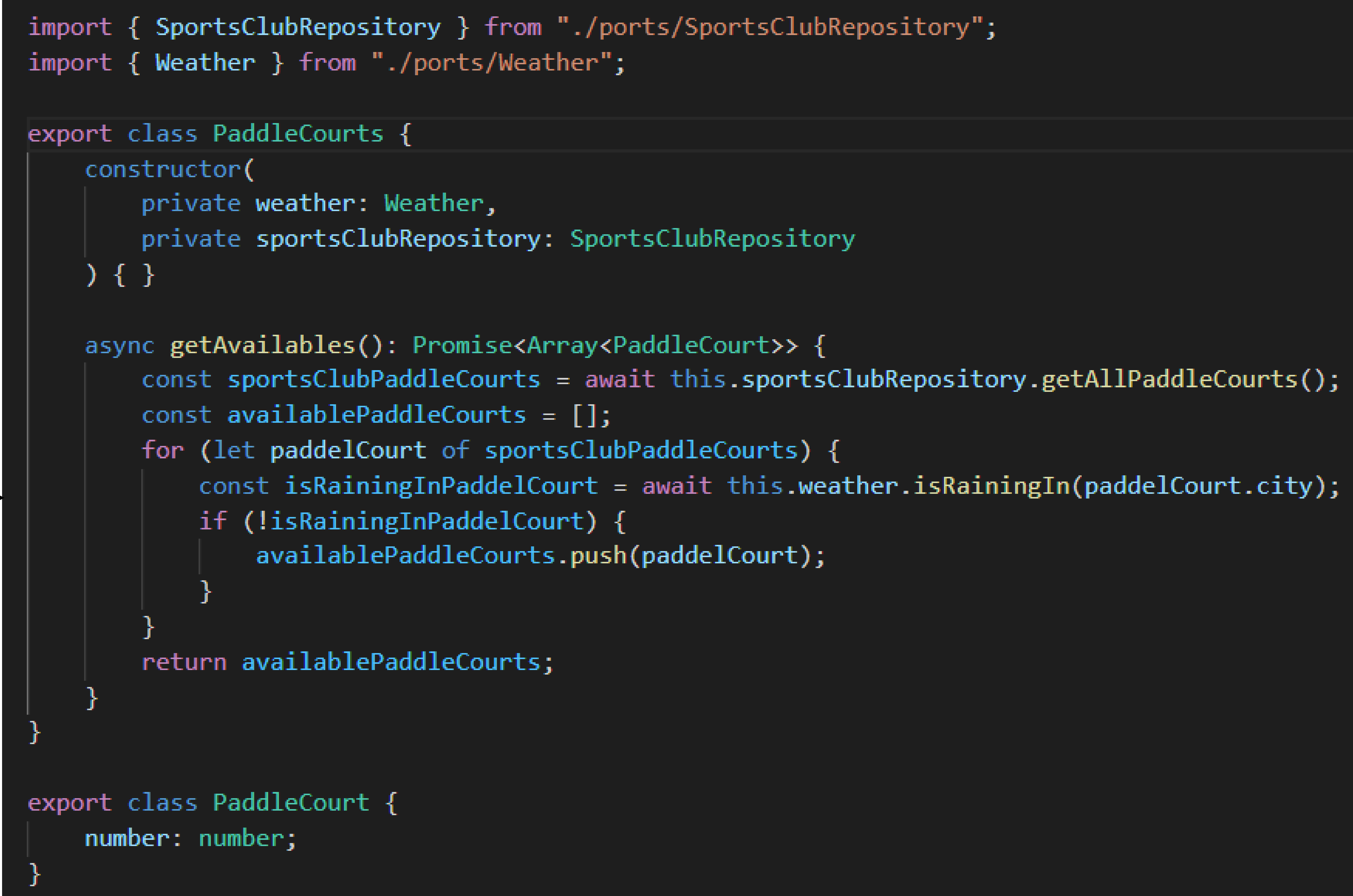
```
import { PaddleCourts } from "./PaddleCourts";
import { SportsClubUserInterface } from "./ports/SportsClubUserInterface";

export class SportsClub {
    constructor(
        private userInterface: SportsClubUserInterface,
        private paddleCourts: PaddleCourts
    ) { }

    init() {
        this.userInterface.installGetAvailablePaddleCourts(
            () => this.paddleCourts.getAvailables()
        );
    }
}
```

DotNet2021

Hexagonal Architecture
(or Ports & Adapters)



Application

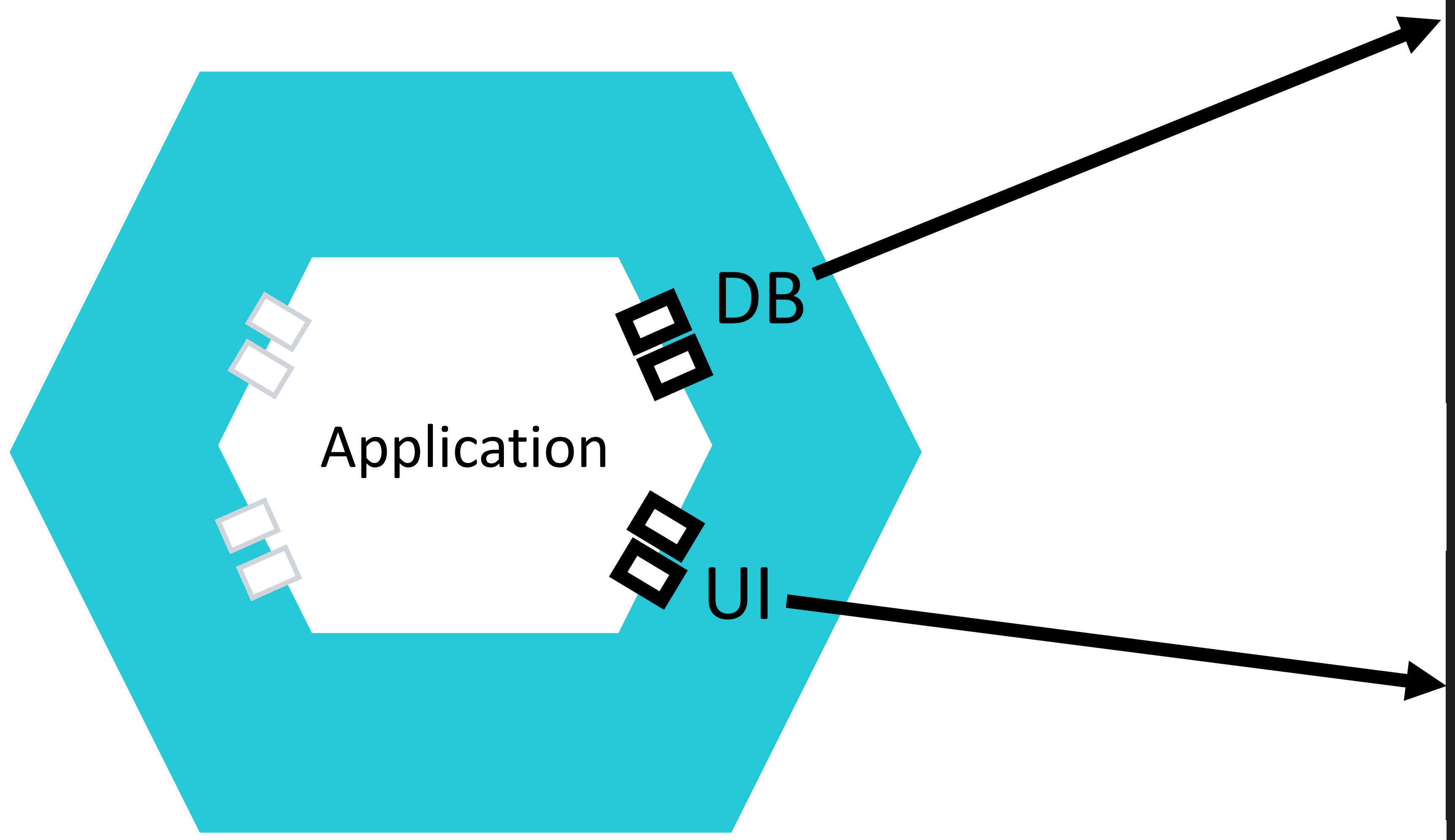


```
import { SportsClubRepository } from "./ports/SportsClubRepository";
import { Weather } from "./ports/Weather";

export class PaddleCourts {
    constructor(
        private weather: Weather,
        private sportsClubRepository: SportsClubRepository
    ) { }

    async getAvailables(): Promise<Array<PaddleCourt>> {
        const sportsClubPaddleCourts = await this.sportsClubRepository.getAllPaddleCourts();
        const availablePaddleCourts = [];
        for (let paddelCourt of sportsClubPaddleCourts) {
            const isRainingInPaddelCourt = await this.weather.isRainingIn(paddelCourt.city);
            if (!isRainingInPaddelCourt) {
                availablePaddleCourts.push(paddelCourt);
            }
        }
        return availablePaddleCourts;
    }
}

export class PaddleCourt {
    number: number;
}
```

DotNet2021

Ports

Hexagonal Architecture
(or Ports & Adapters)

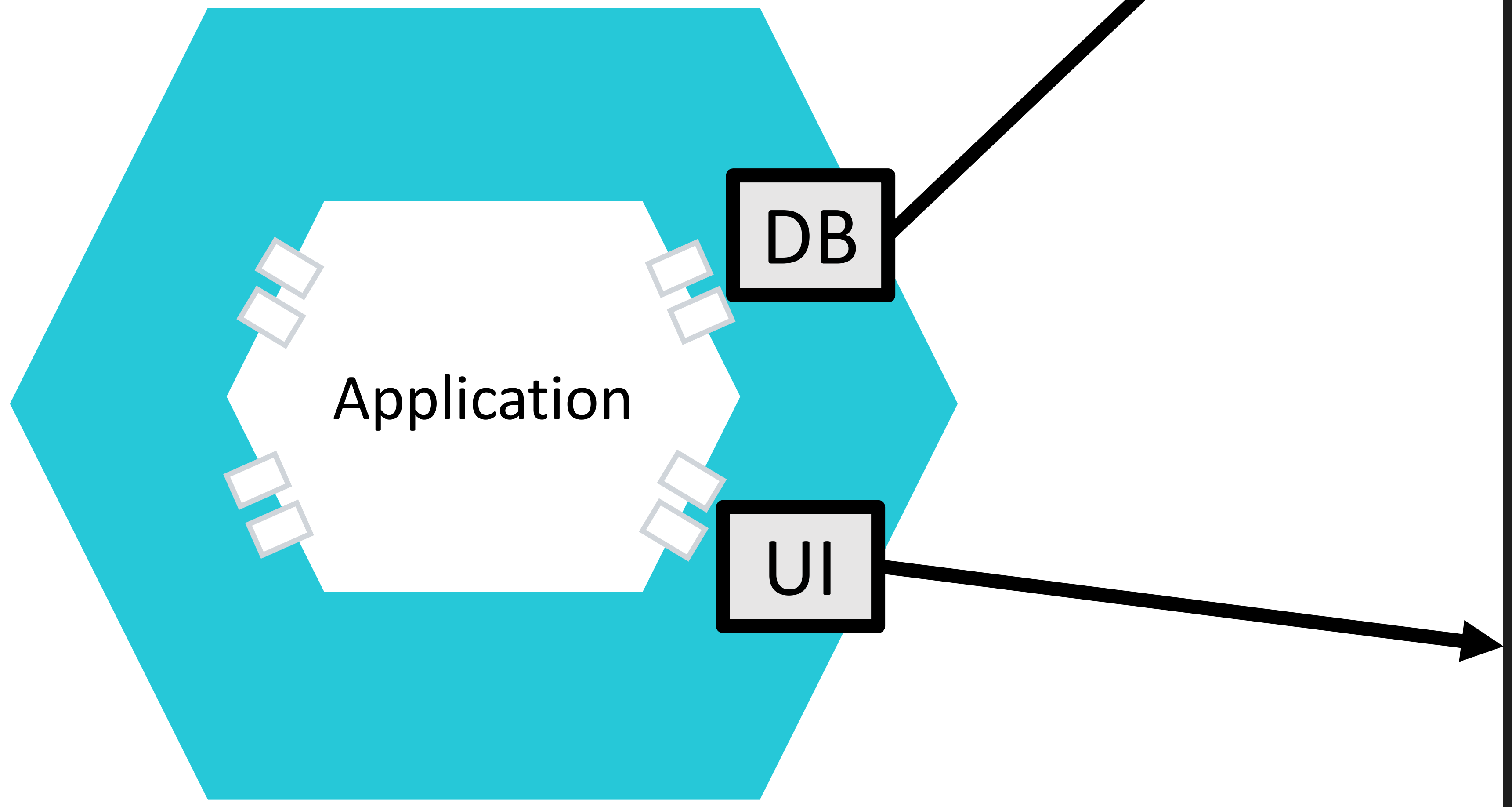


```
export interface SportsClubRepository {
    getAllPaddleCourts(): Promise<Array<PaddleCourt>>;
}

export class PaddleCourt {
    number: number;
    city: City;
}

export type City = 'Madrid' | 'Valencia';
```

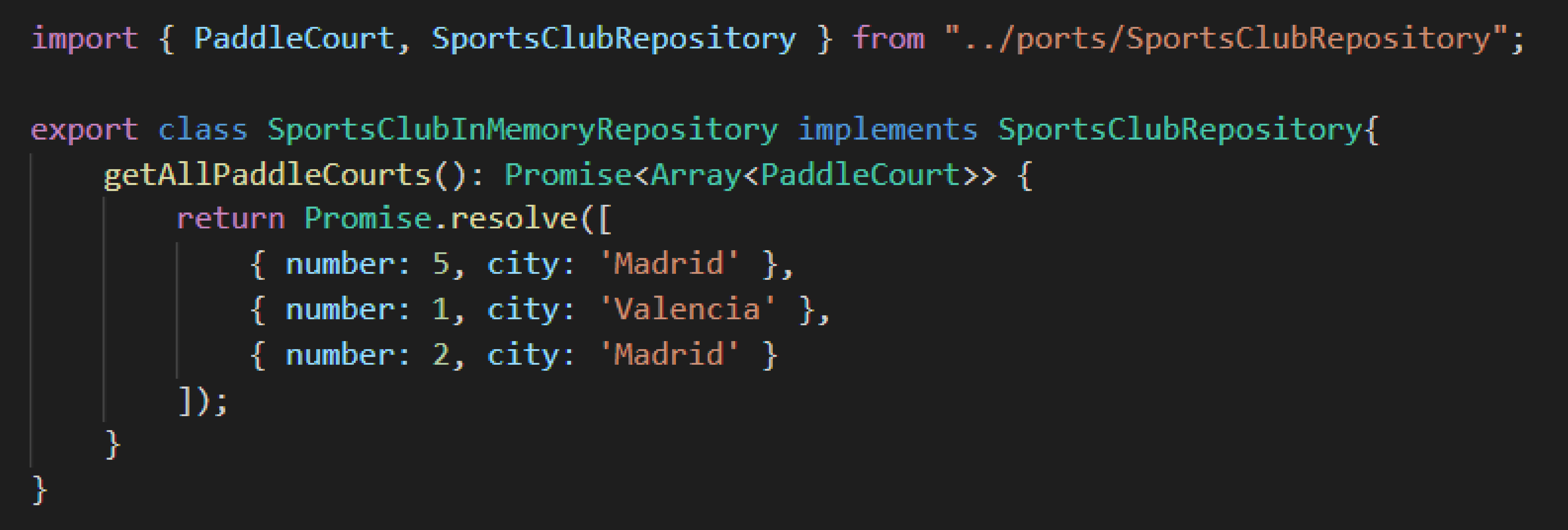
```
export interface SportsClubUserInterface {
    installGetAvailablePaddleCourts(callback: () => Promise<PaddleCourt[]>): void;
}
```

DotNet2021

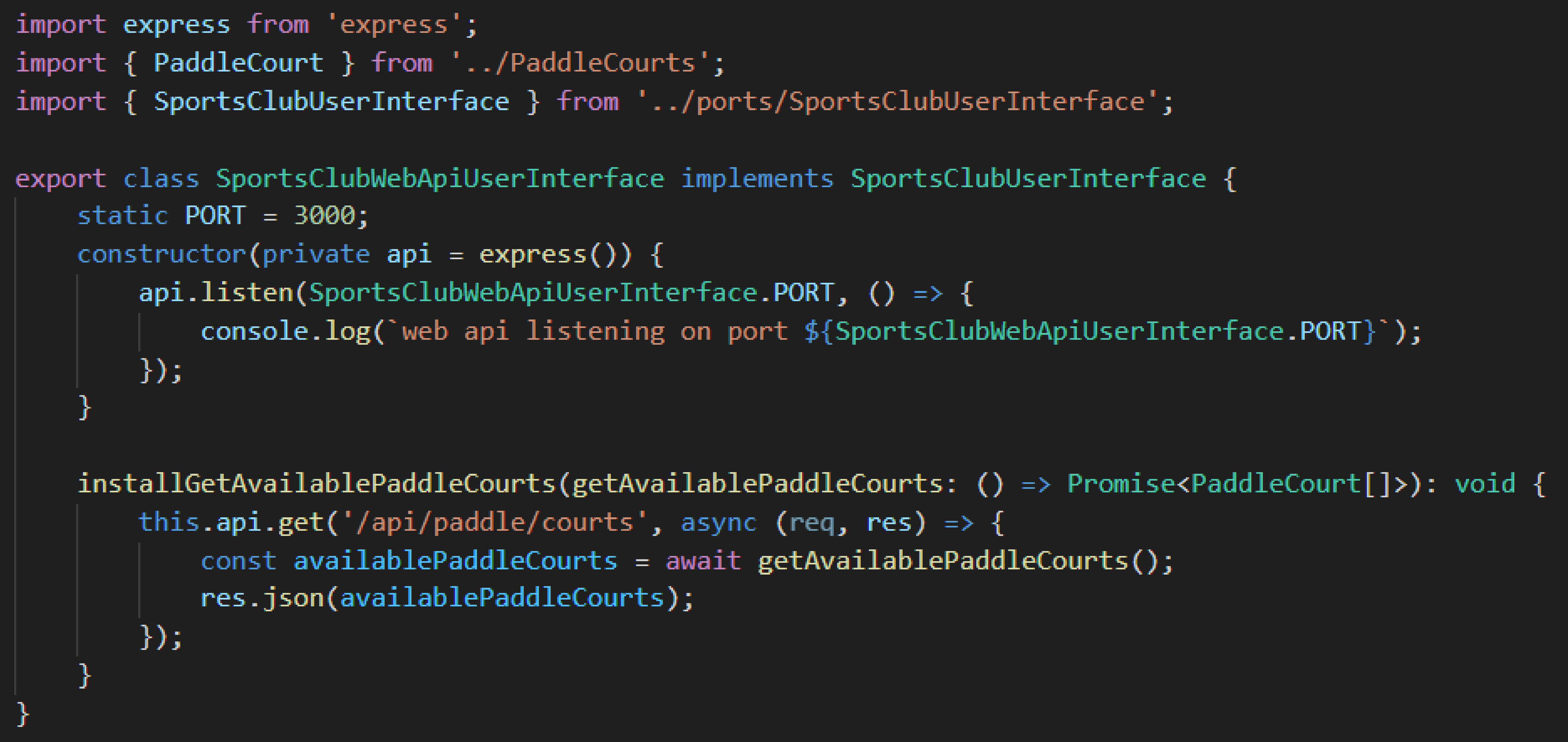
Hexagonal Architecture
(or Ports & Adapters)



Adapters



```
import { PaddleCourt, SportsClubRepository } from "../ports/SportsClubRepository";

export class SportsClubInMemoryRepository implements SportsClubRepository{
    getAllPaddleCourts(): Promise<Array<PaddleCourt>> {
        return Promise.resolve([
            { number: 5, city: 'Madrid' },
            { number: 1, city: 'Valencia' },
            { number: 2, city: 'Madrid' }
        ]);
    }
}
```

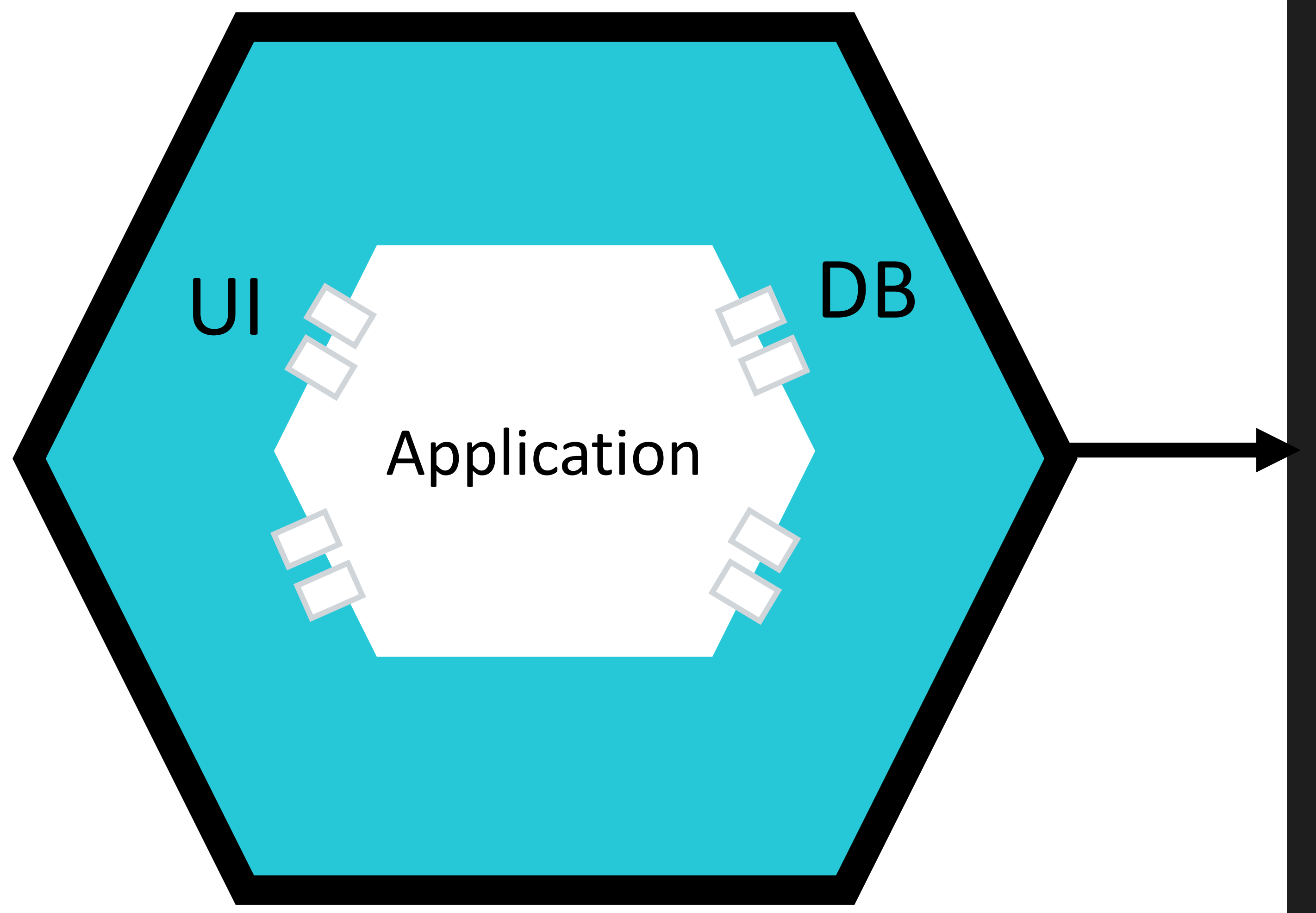


```
import express from 'express';
import { PaddleCourt } from '../PaddleCourts';
import { SportsClubUserInterface } from '../ports/SportsClubUserInterface';

export class SportsClubWebApiUserInterface implements SportsClubUserInterface {
    static PORT = 3000;
    constructor(private api = express()) {
        api.listen(SportsClubWebApiUserInterface.PORT, () => {
            console.log(`web api listening on port ${SportsClubWebApiUserInterface.PORT}`);
        });
    }

    installGetAvailablePaddleCourts(getAvailablePaddleCourts: () => Promise<PaddleCourt[]>): void {
        this.api.get('/api/paddle/courts', async (req, res) => {
            const availablePaddleCourts = await getAvailablePaddleCourts();
            res.json(availablePaddleCourts);
        });
    }
}
```

Concrete Application

DotNet2021

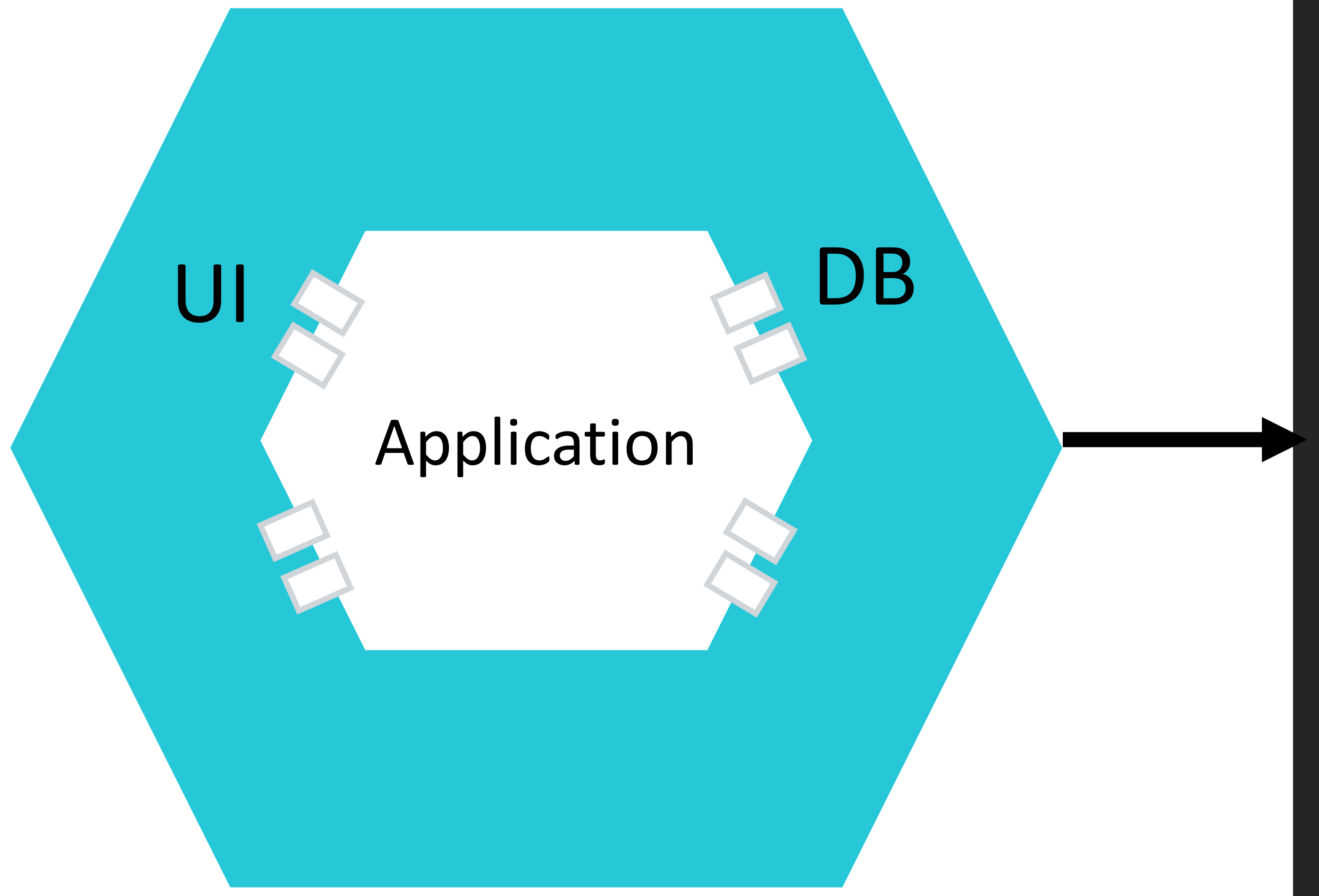
Hexagonal Architecture
(or Ports & Adapters)



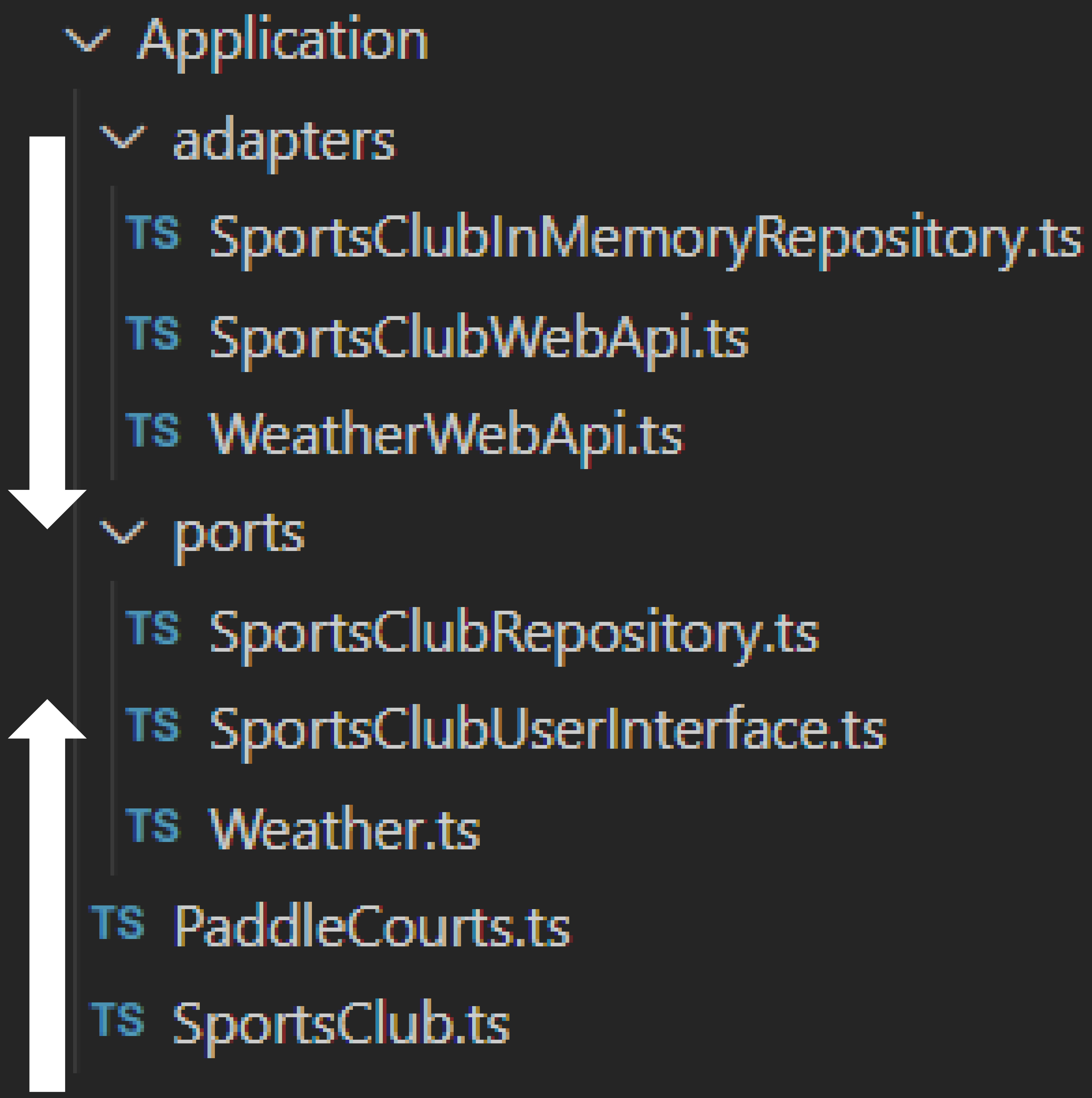
```
import { SportsClubInMemoryRepository } from './Application/adapters/SportsClubInMemoryRepository';
import { SportsClubWebApiUserInterface } from './Application/adapters/SportsClubWebApi';
import { WeatherWebApi } from './Application/adapters/WeatherWebApi';
import { PaddleCourts } from './Application/PaddleCourts';
import { SportsClub } from './Application/SportsClub';

const sportsClub = new SportsClub(
    new SportsClubWebApiUserInterface(),
    new PaddleCourts(
        new WeatherWebApi(),
        new SportsClubInMemoryRepository()
    )
);

sportsClub.init();
```

DotNet2021

Hexagonal Architecture
(or Ports & Adapters)



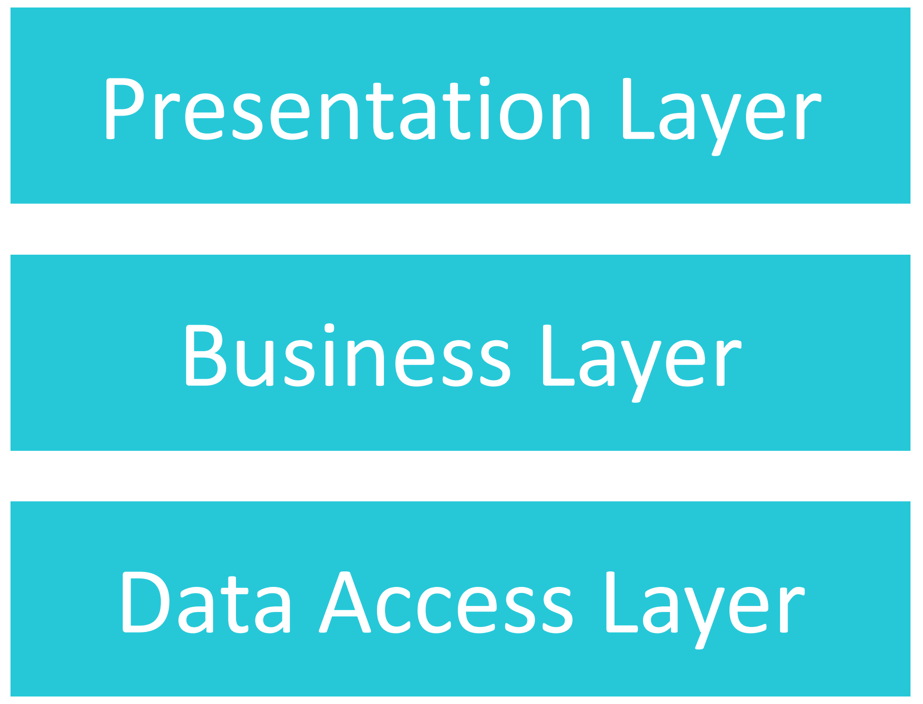
Dependency direction



```
⌄ Application
  ⌄ adapters
     TS SportsClubInMemoryRepository.ts
     TS SportsClubWebApi.ts
     TS WeatherWebApi.ts
  ⌄ ports
     TS SportsClubRepository.ts
     TS SportsClubUserInterface.ts
     TS Weather.ts
  TS PaddleCourts.ts
  TS SportsClub.ts
```

1996

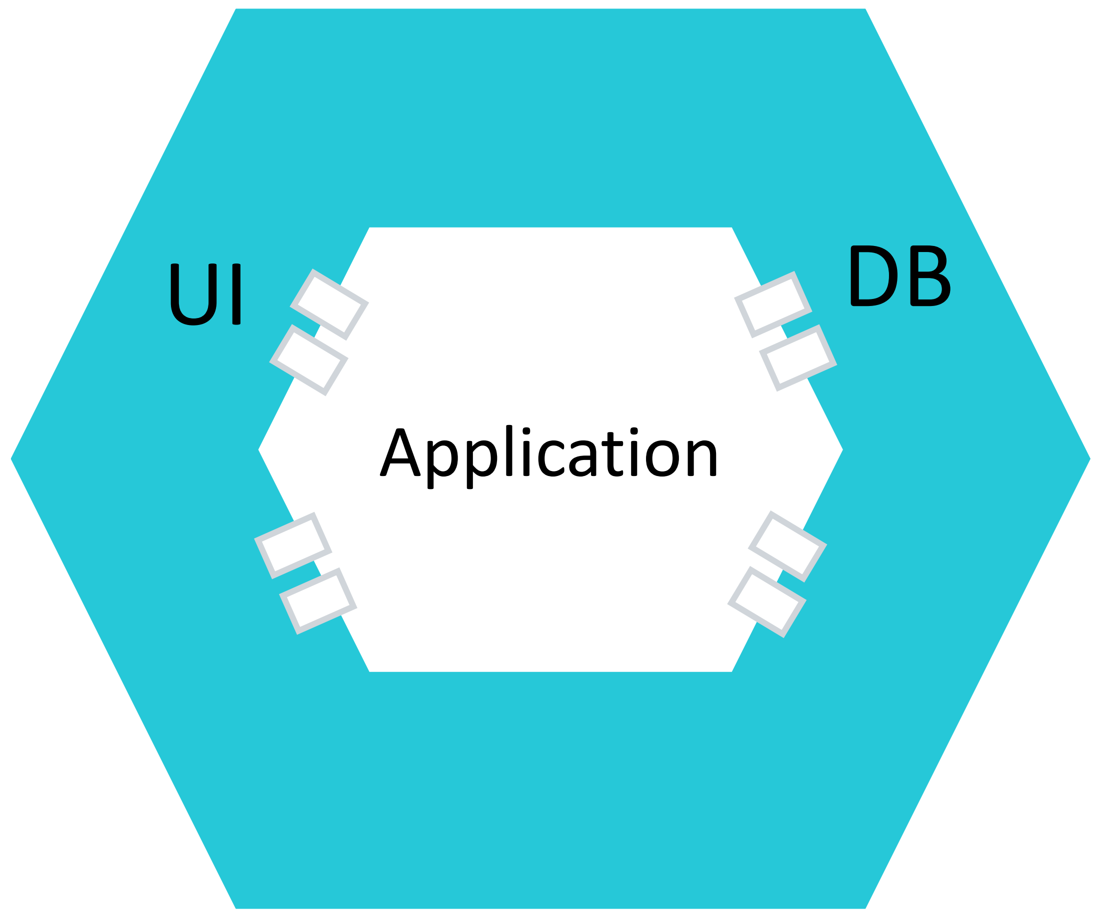
(POSA volumen 1) [1]



Layered Architecture
(Three layer)

2005

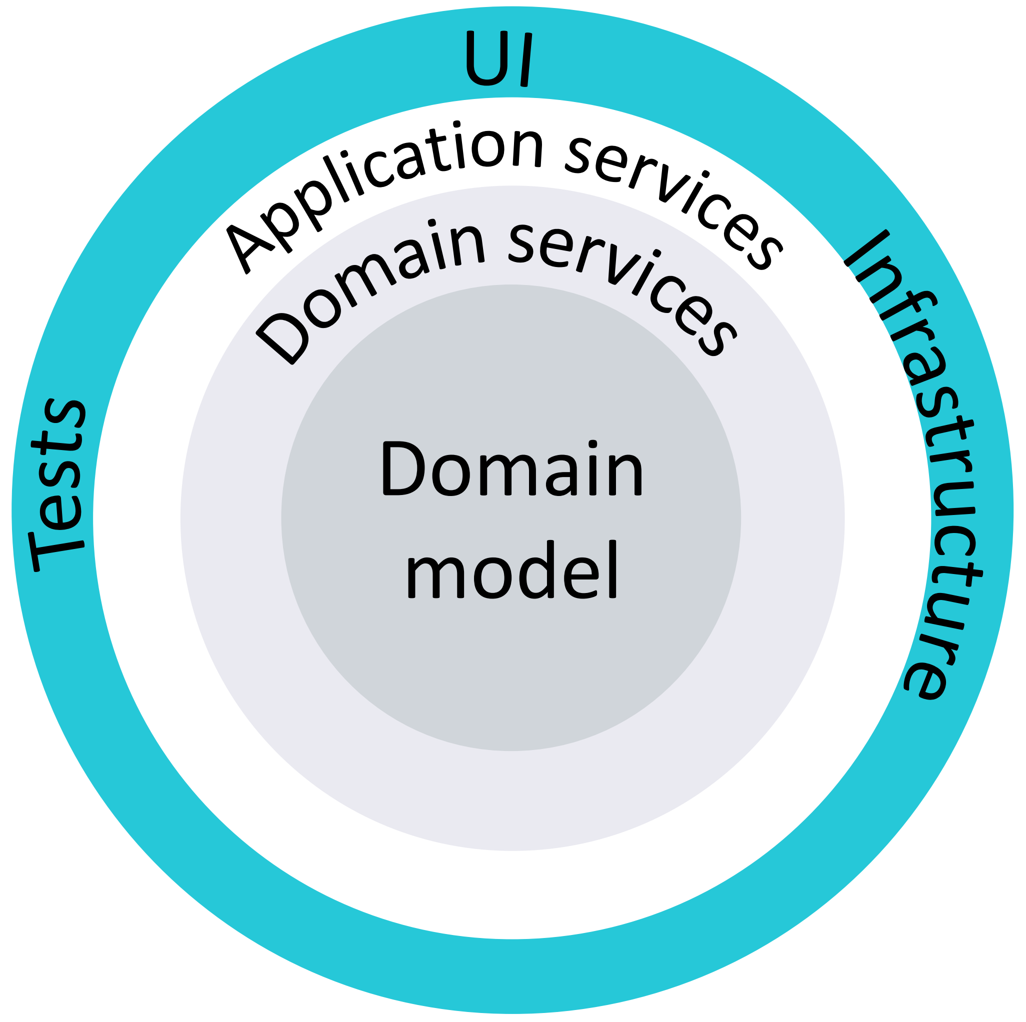
(Alistair Cockburn) [2]



Hexagonal Architecture
(or Ports & Adapters)

2008

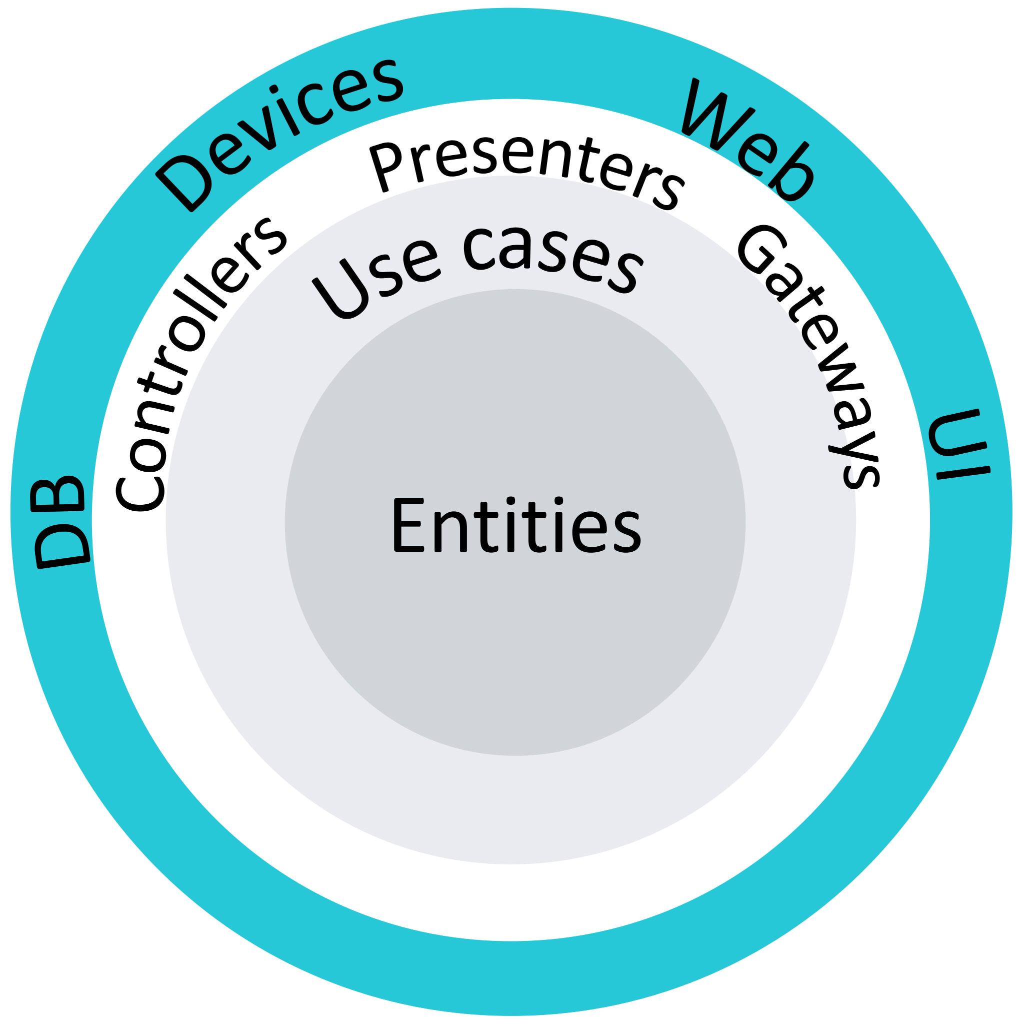
(Jeffrey Palermo) [3]



Onion Architecture

2012

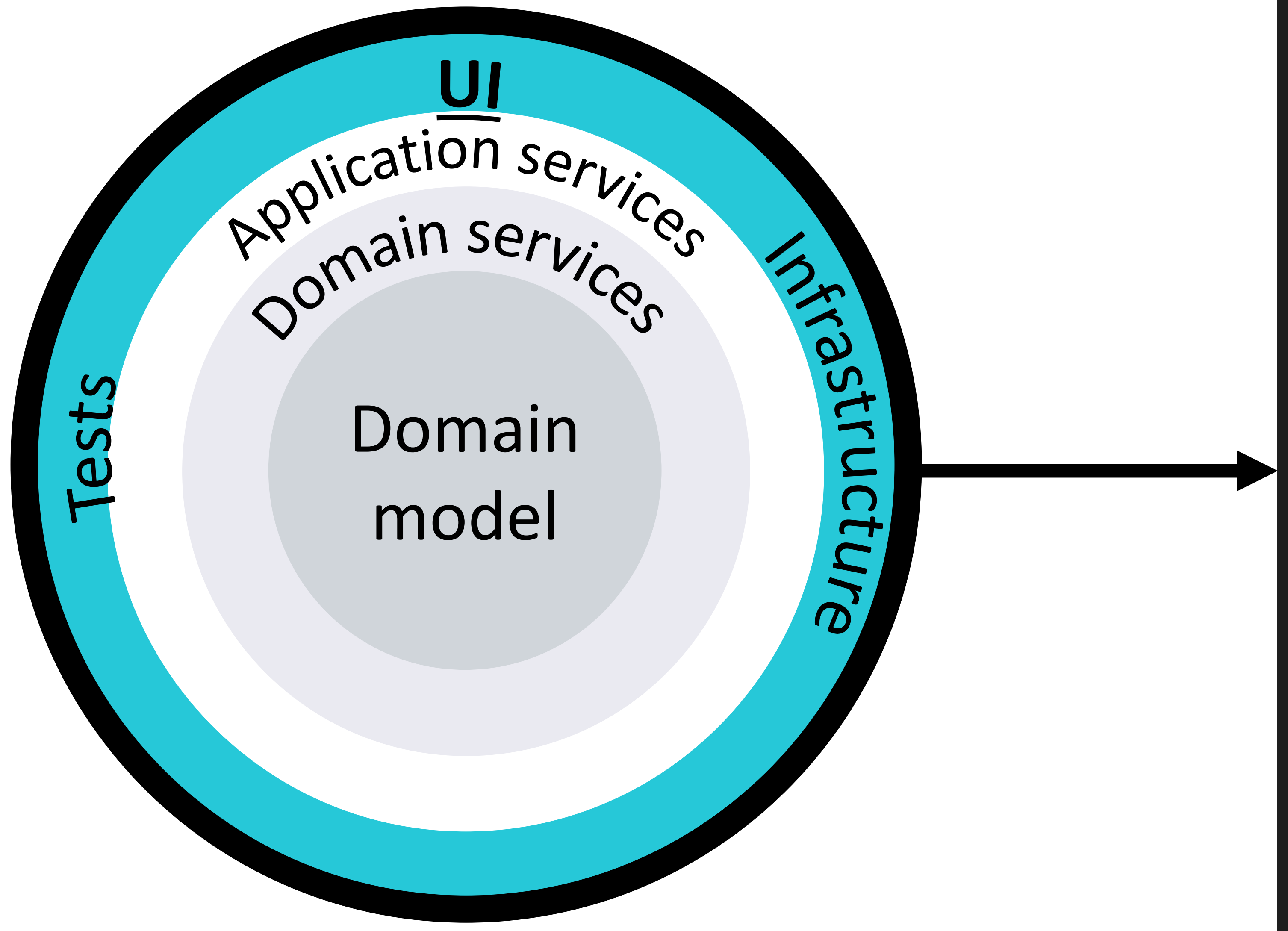
(Robert C. Martin) [4]



Clean architecture
(Hexagonal + Onion architectures)

DotNet2021

Onion Architecture



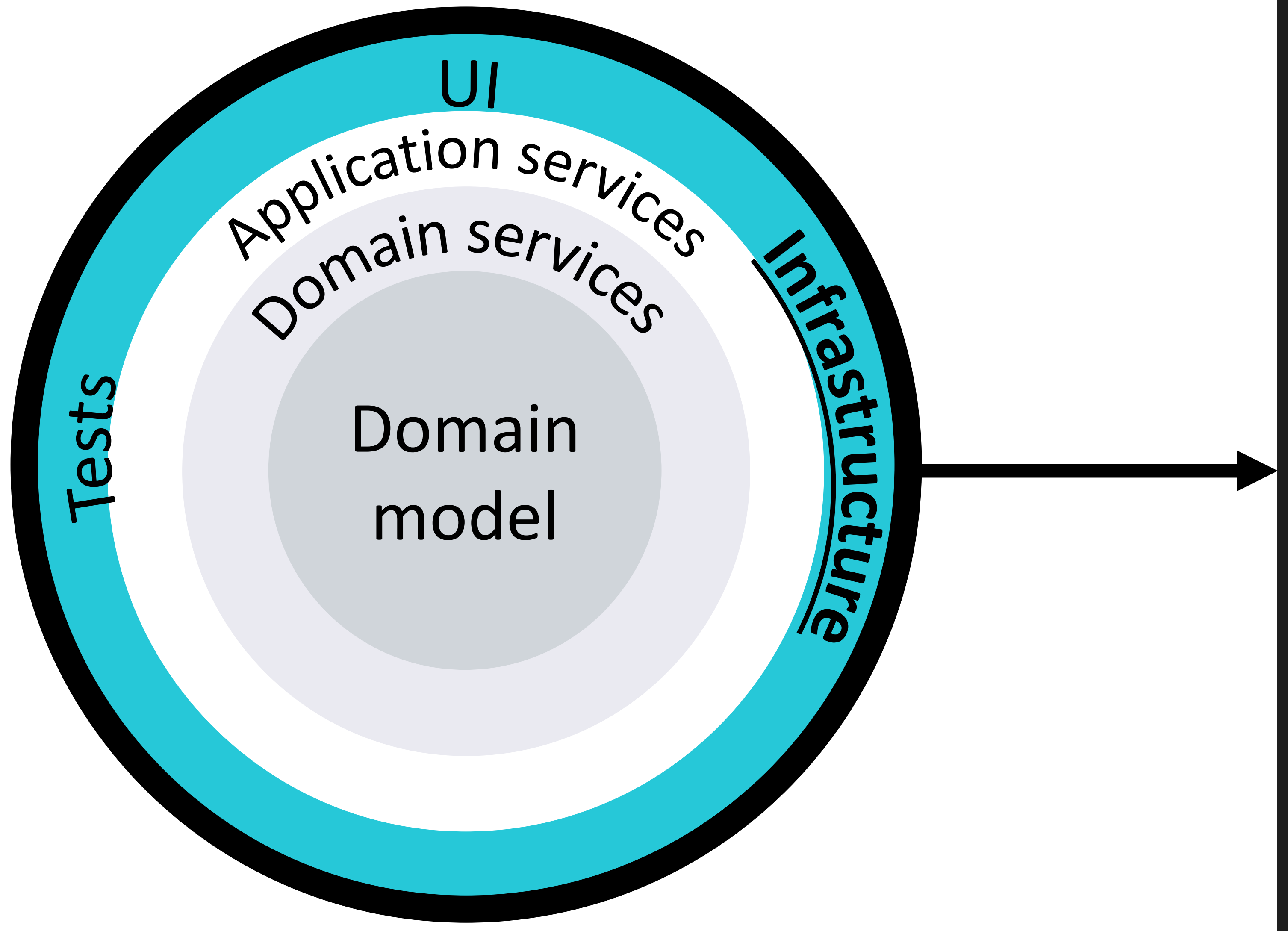
User Interface

```
import express from 'express';
import { inject, injectable } from 'inversify';
import { PaddleCourts } from '../3. ApplicationServices/PaddleCourts';
import TYPES from '../container.types';

@injectable()
export class SportsClubWebApiUserInterface {
    static PORT = 3000;
    api = express();

    constructor(@inject(TYPES.PaddleCourts) private paddleCourts: PaddleCourts) { }

    init() {
        this.api.get('/api/paddle/courts', async (req, res) => {
            const availablePaddleCourts = await this.paddleCourts.getAvailables();
            res.json(availablePaddleCourts);
        });
        this.api.listen(SportsClubWebApiUserInterface.PORT, () => {
            console.log(`web api listening on port ${SportsClubWebApiUserInterface.PORT}`);
        });
    }

}
```

DotNet2021

Infrastructure

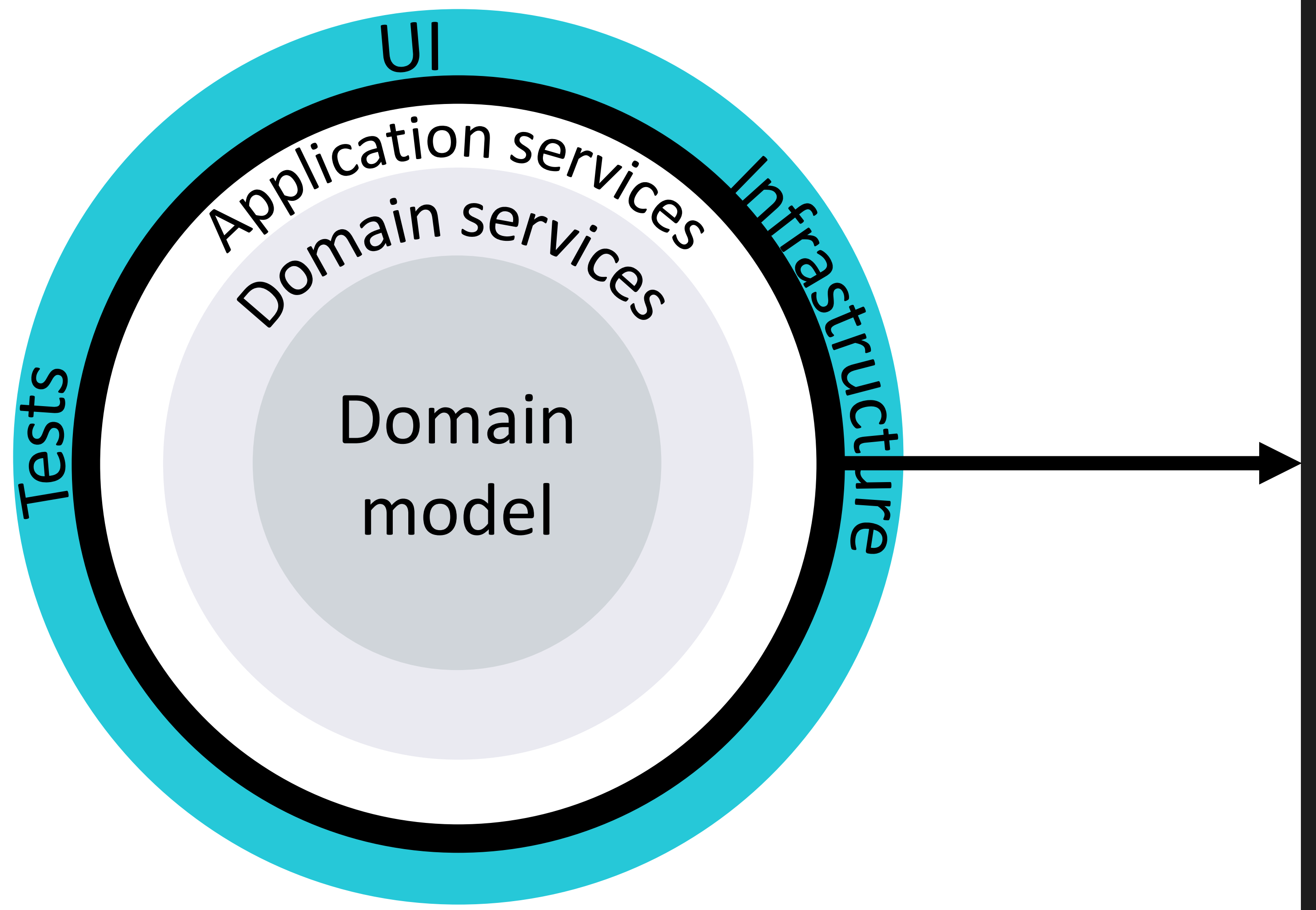
Onion Architecture



```
import { PaddleCourt } from "../1. DomainModel/PaddleCourt";
import { SportsClubRepository } from "../2. DomainServices/SportsClubRepository";
import { injectable } from "inversify";

@injectable()
export class SportsClubInMemoryRepository implements SportsClubRepository {
    getAllPaddleCourts(): Promise<Array<PaddleCourt>> {
        return Promise.resolve([
            { number: 5, city: 'Madrid' },
            { number: 1, city: 'Valencia' },
            { number: 2, city: 'Madrid' }
        ]);
    }
}


import { Weather } from "../3. ApplicationServices/Weather";
import { City } from "../1. DomainModel/City";
// import axios from 'axios';
import { inject, injectable } from "inversify";

@injectable()
export class WeatherWebApiClient implements Weather {
    isRainingIn(city: City): Promise<boolean> {

        // HTTP call logic: await axios.get('https://weather.com/api/israining')
        return Promise.resolve(false);
    }
}
```

DotNet2021

Onion Architecture



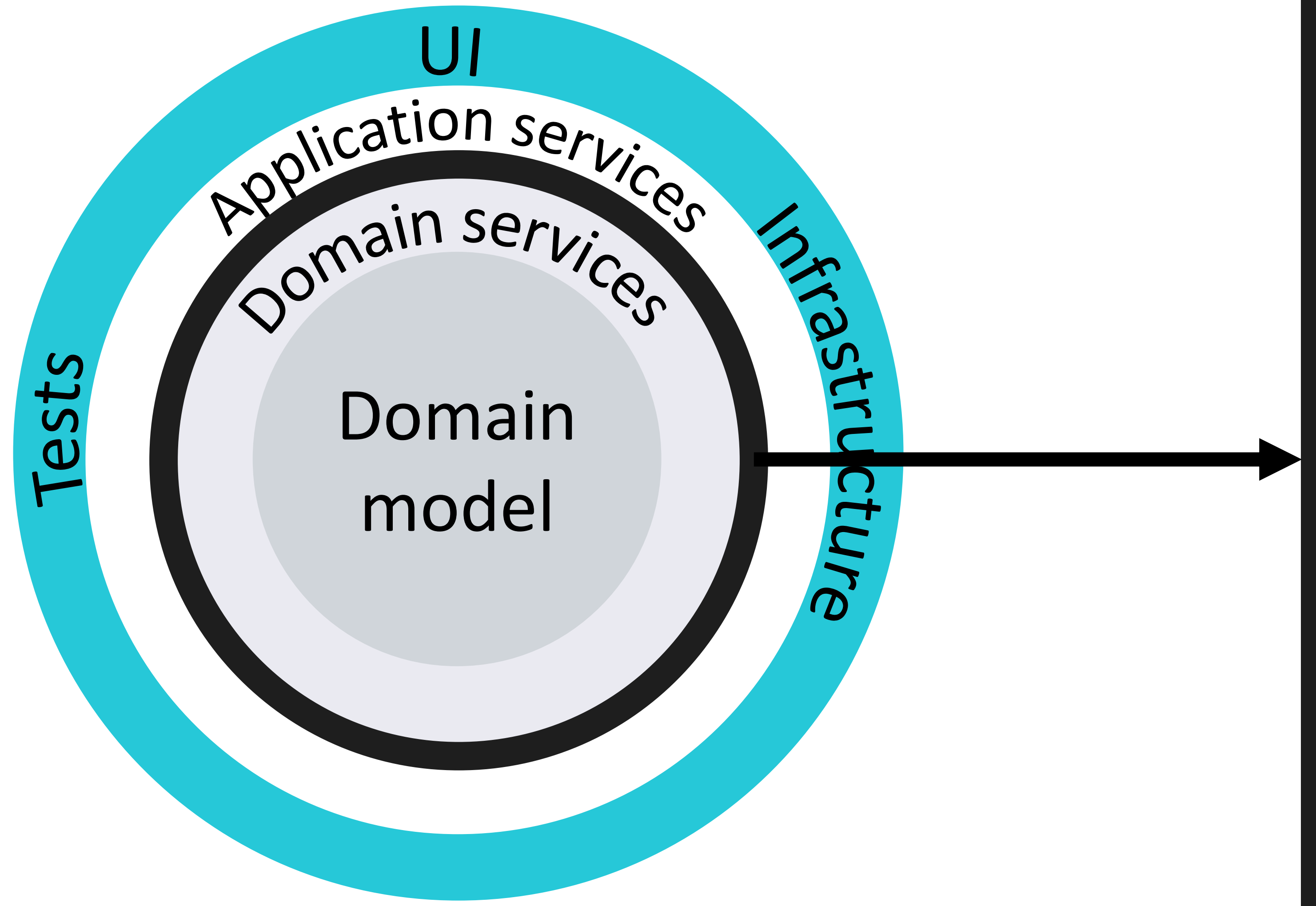
Application Services

```
import { SportsClubRepository } from "../2. DomainServices/SportsClubRepository";
import { Weather } from "./Weather";
import { PaddleCourt } from "../1. DomainModel/PaddleCourt";
import { inject, injectable } from 'inversify';
import TYPES from "../container.types";

@injectable()
export class PaddleCourts {
    constructor(
        @inject(TYPES.Weather) private weather: Weather,
        @inject(TYPES.SportsClubRepository) private sportsClubRepository: SportsClubRepository
    ) { }

    async getAvailables(): Promise<Array<PaddleCourt>> {
        const sportsClubPaddleCourts = await this.sportsClubRepository.getAllPaddleCourts();
        const availablePaddleCourts = [];
        for (let paddelCourt of sportsClubPaddleCourts) {
            const isRainingInPaddelCourt = await this.weather.isRainingIn(paddelCourt.city);
            if (!isRainingInPaddelCourt) {
                availablePaddleCourts.push(paddelCourt);
            }
        }
        return availablePaddleCourts;
    }
}
```

```
import { City } from "../1. DomainModel/City";

export interface Weather {
    isRainingIn(city: City): Promise<boolean>;
}
```

DotNet2021

Onion Architecture



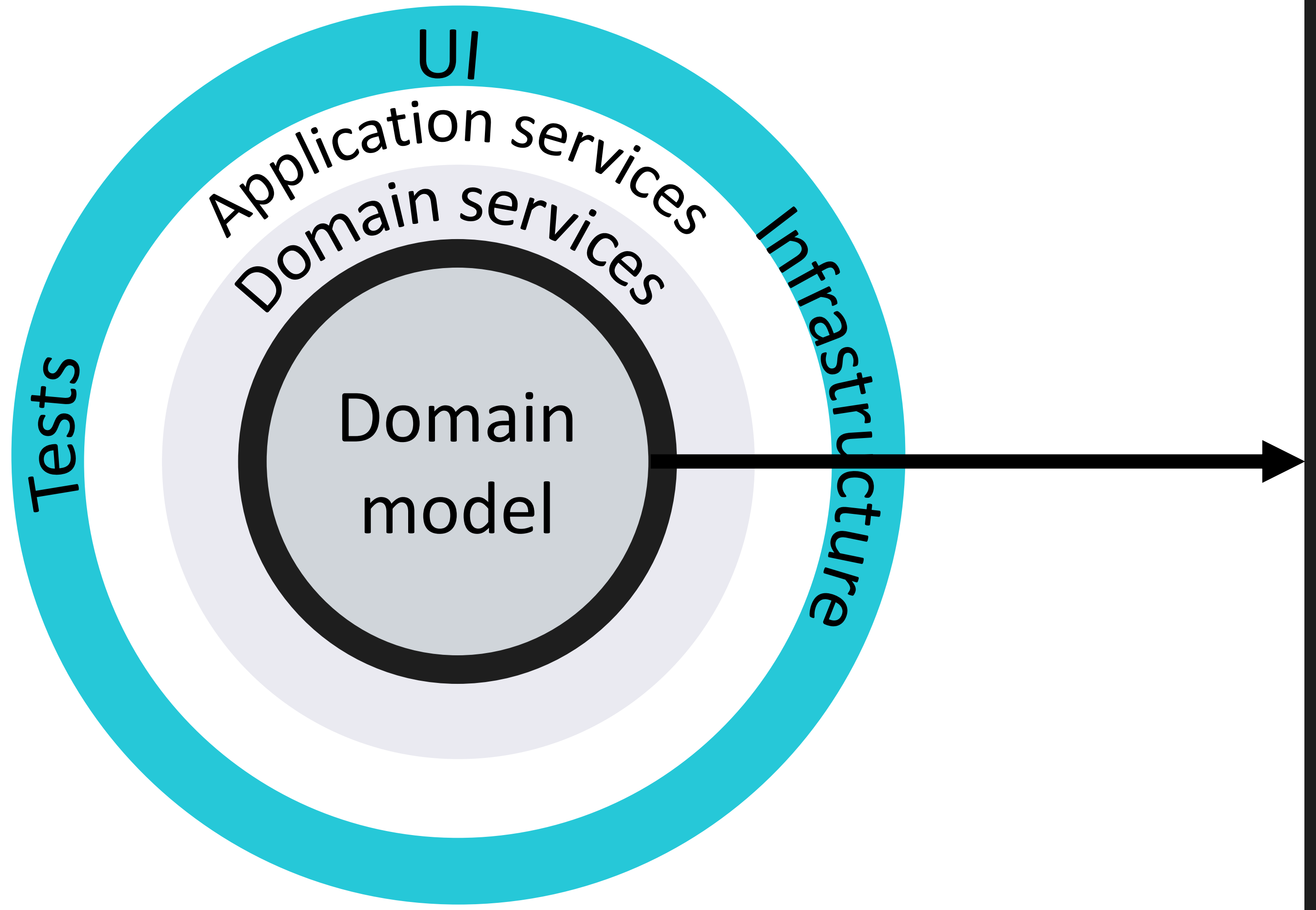
Domain Services

```
import { PaddleCourt } from "../1. DomainModel/PaddleCourt";

export interface SportsClubRepository {
    getAllPaddleCourts(): Promise<Array<PaddleCourt>>;
}
```

DotNet2021

Domain Model

Onion Architecture

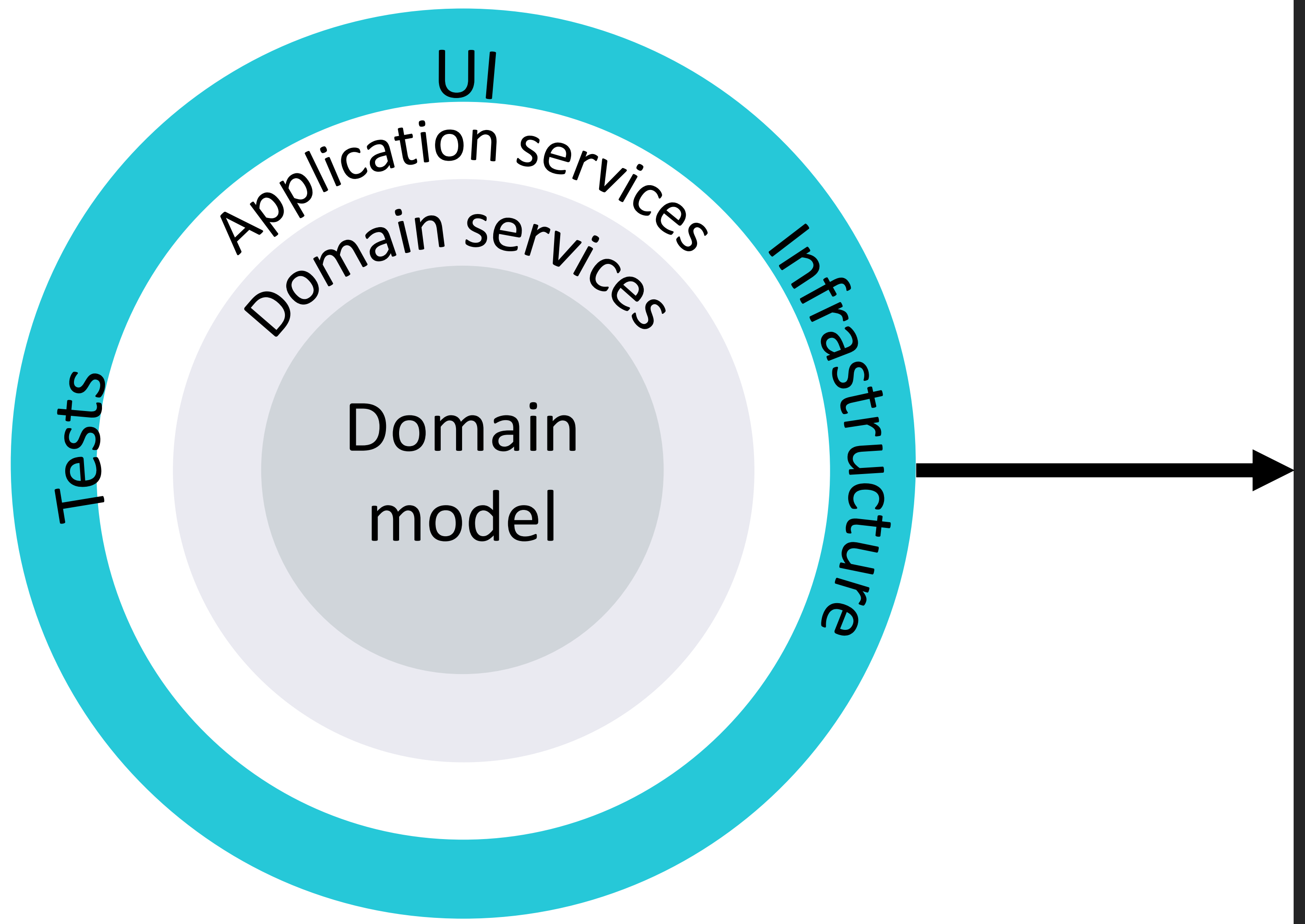


```
export class PaddleCourt {
    number: number;
    city: City
}
```

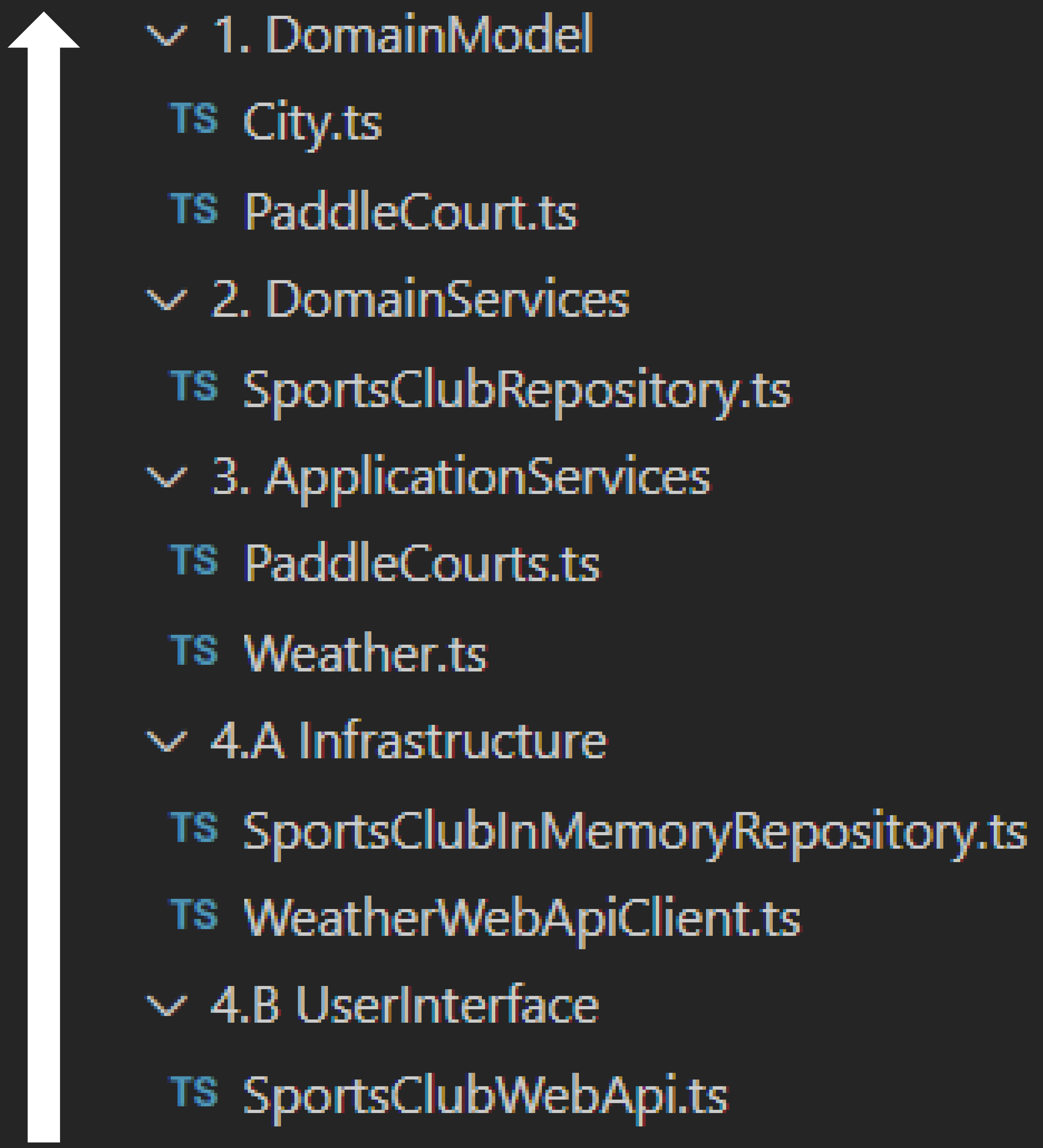
```
export type City = 'Madrid' | 'Valencia';
```

DotNet2021

Onion Architecture



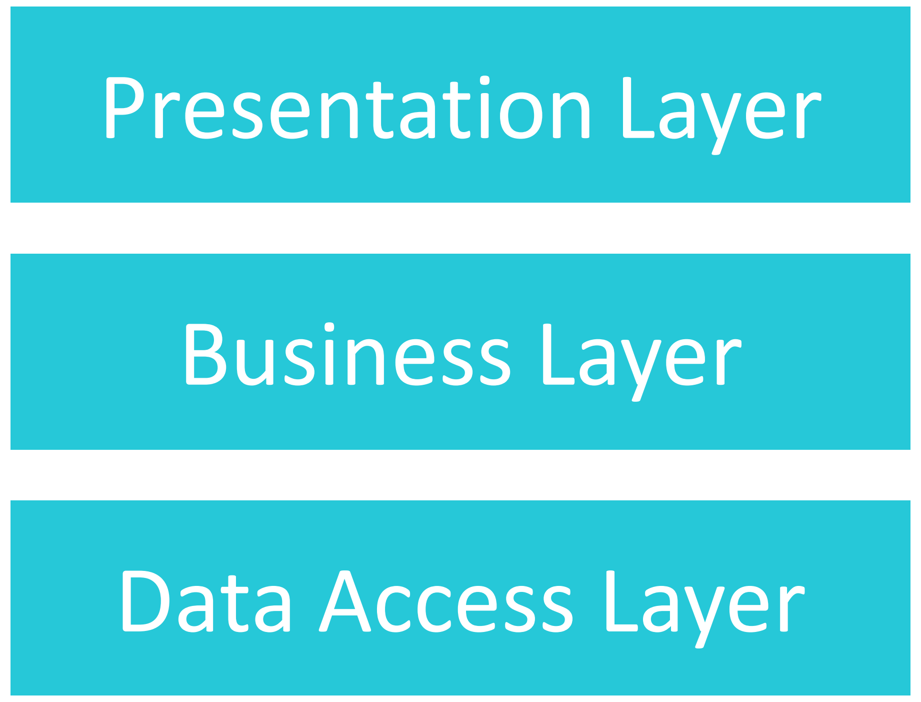
Dependency direction



```
⌄ 1. DomainModel
  TS City.ts
  TS PaddleCourt.ts
⌄ 2. DomainServices
  TS SportsClubRepository.ts
⌄ 3. ApplicationServices
  TS PaddleCourts.ts
  TS Weather.ts
⌄ 4.A Infrastructure
  TS SportsClubInMemoryRepository.ts
  TS WeatherWebApiClient.ts
⌄ 4.B UserInterface
  TS SportsClubWebApi.ts
```

1996

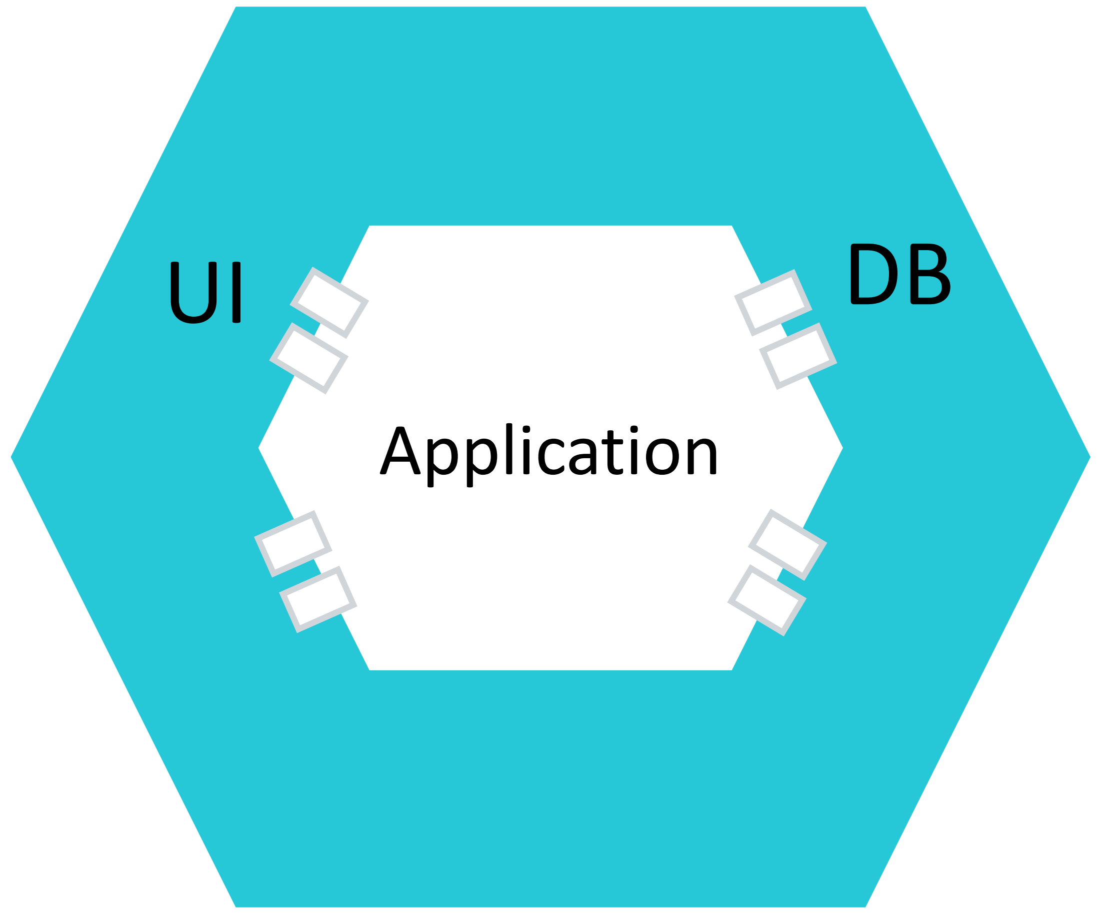
(POSA volumen 1) [1]



Layered Architecture
(Three layer)

2005

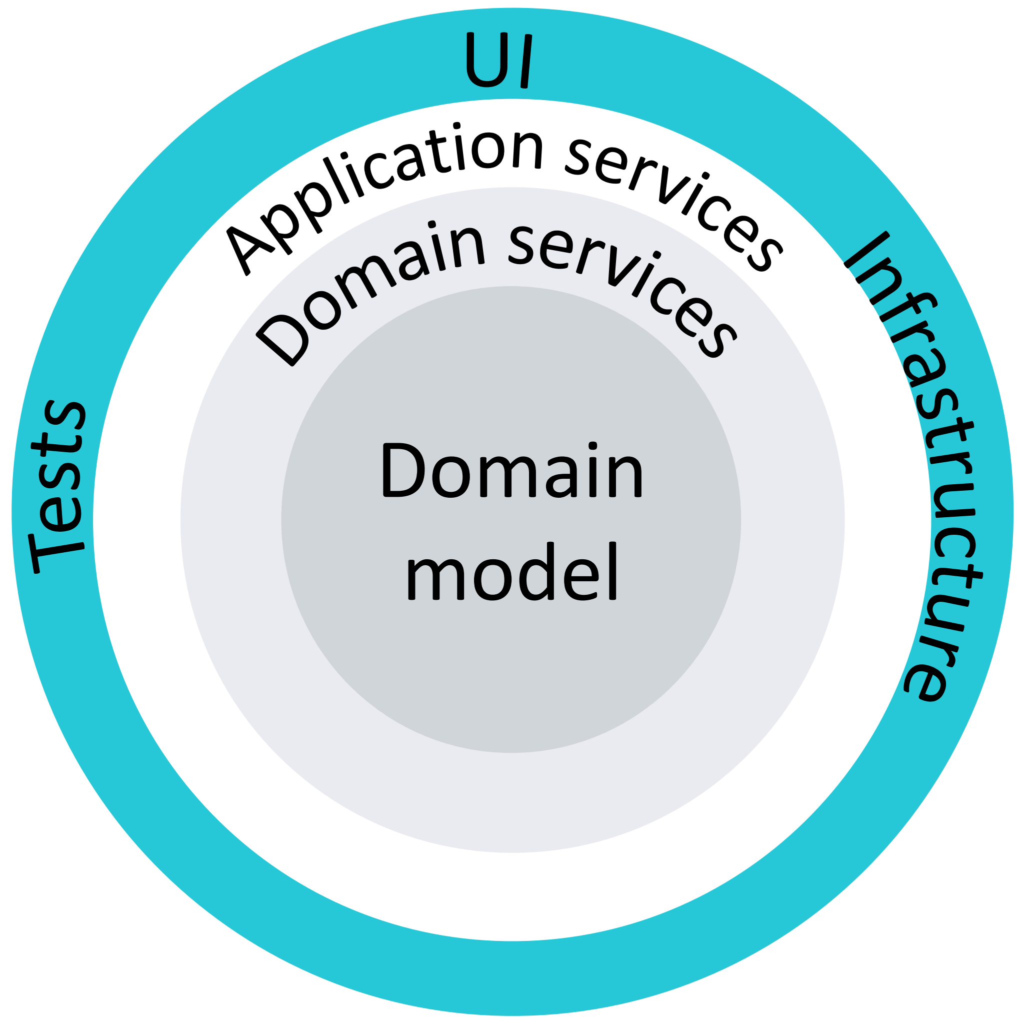
(Alistair Cockburn) [2]



Hexagonal Architecture
(or Ports & Adapters)

2008

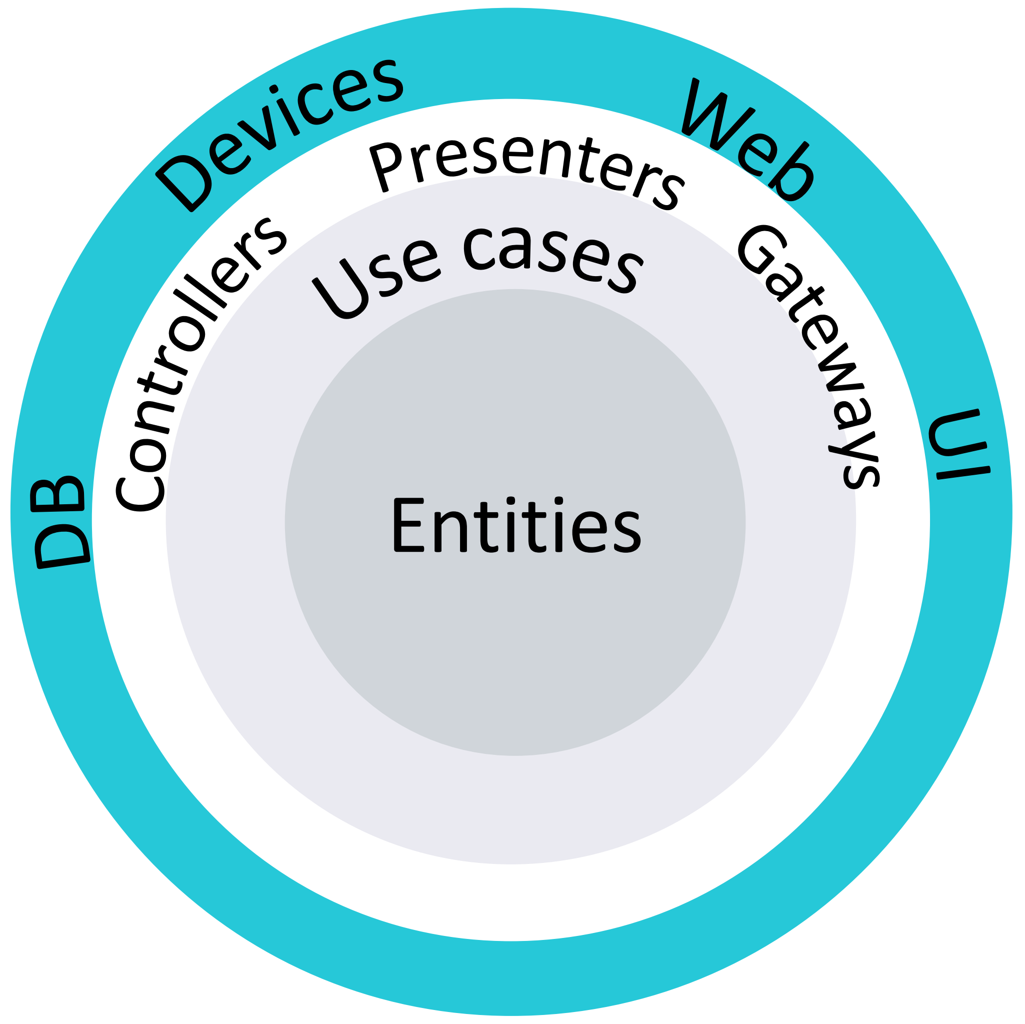
(Jeffrey Palermo) [3]



Onion Architecture

2012

(Robert C. Martin) [4]

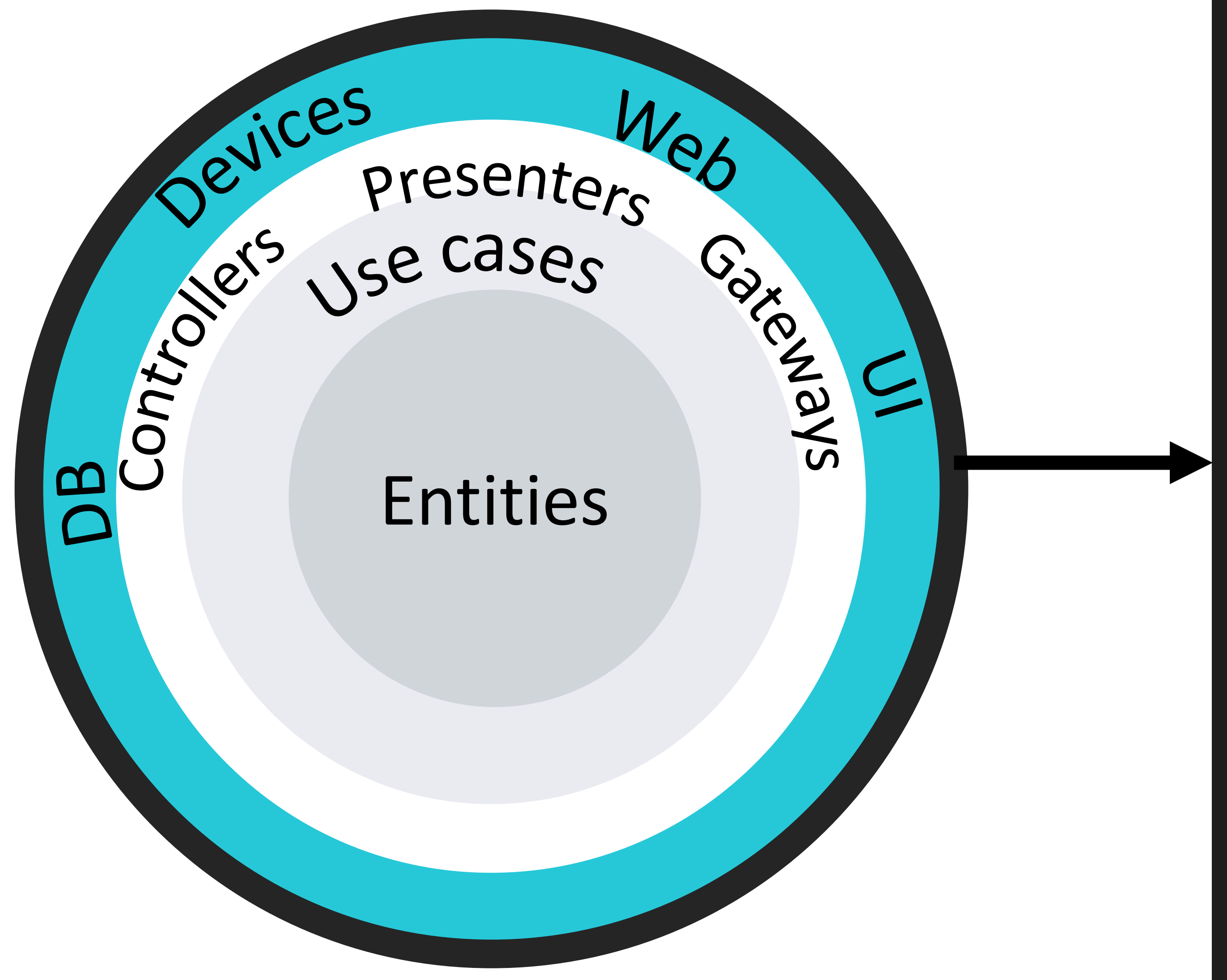


Clean architecture
(Hexagonal + Onion architectures)

DotNet2021

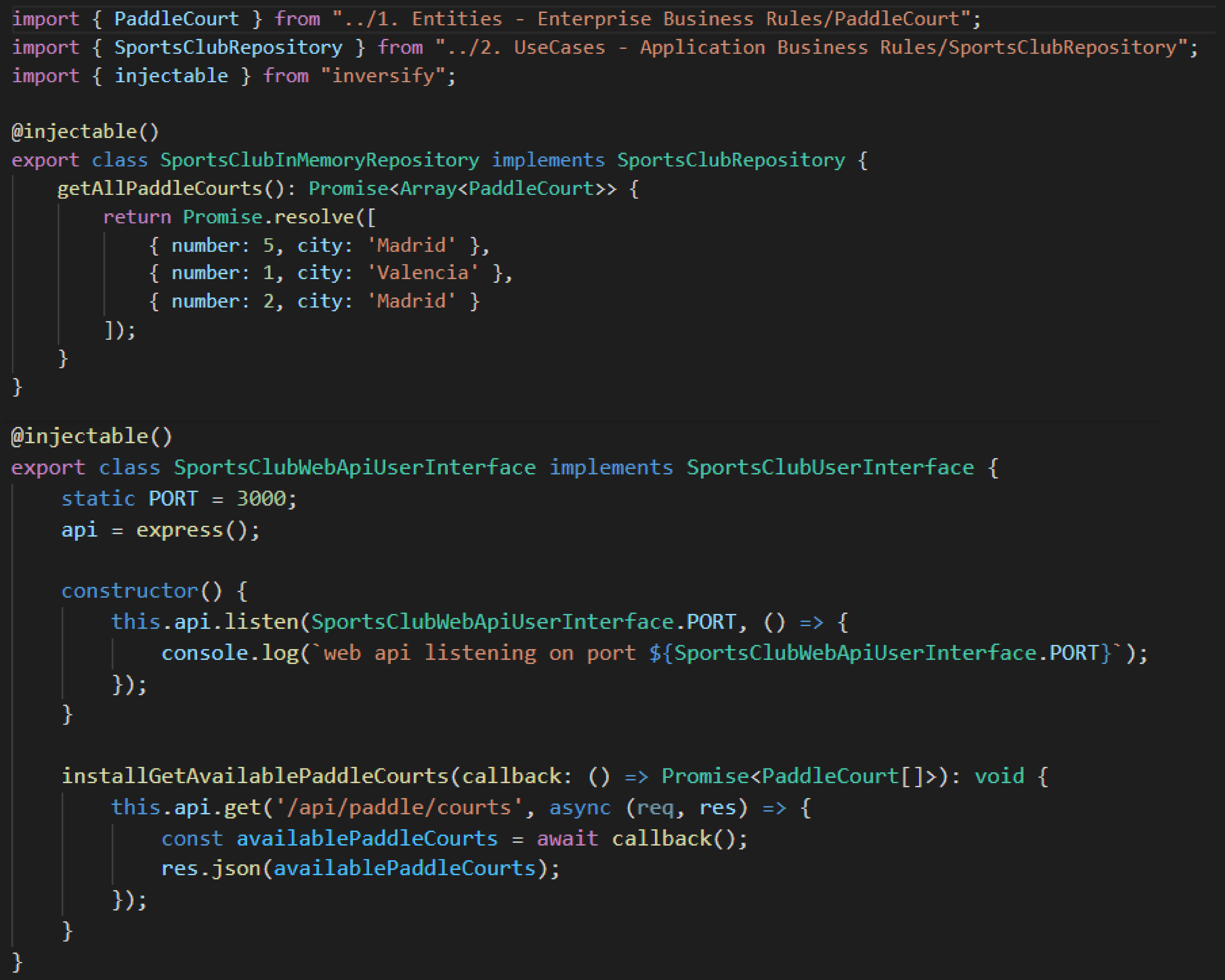
2012

(Robert C. Martin) [4]



Clean architecture
(Hexagonal + Onion architectures)

Frameworks & Drivers

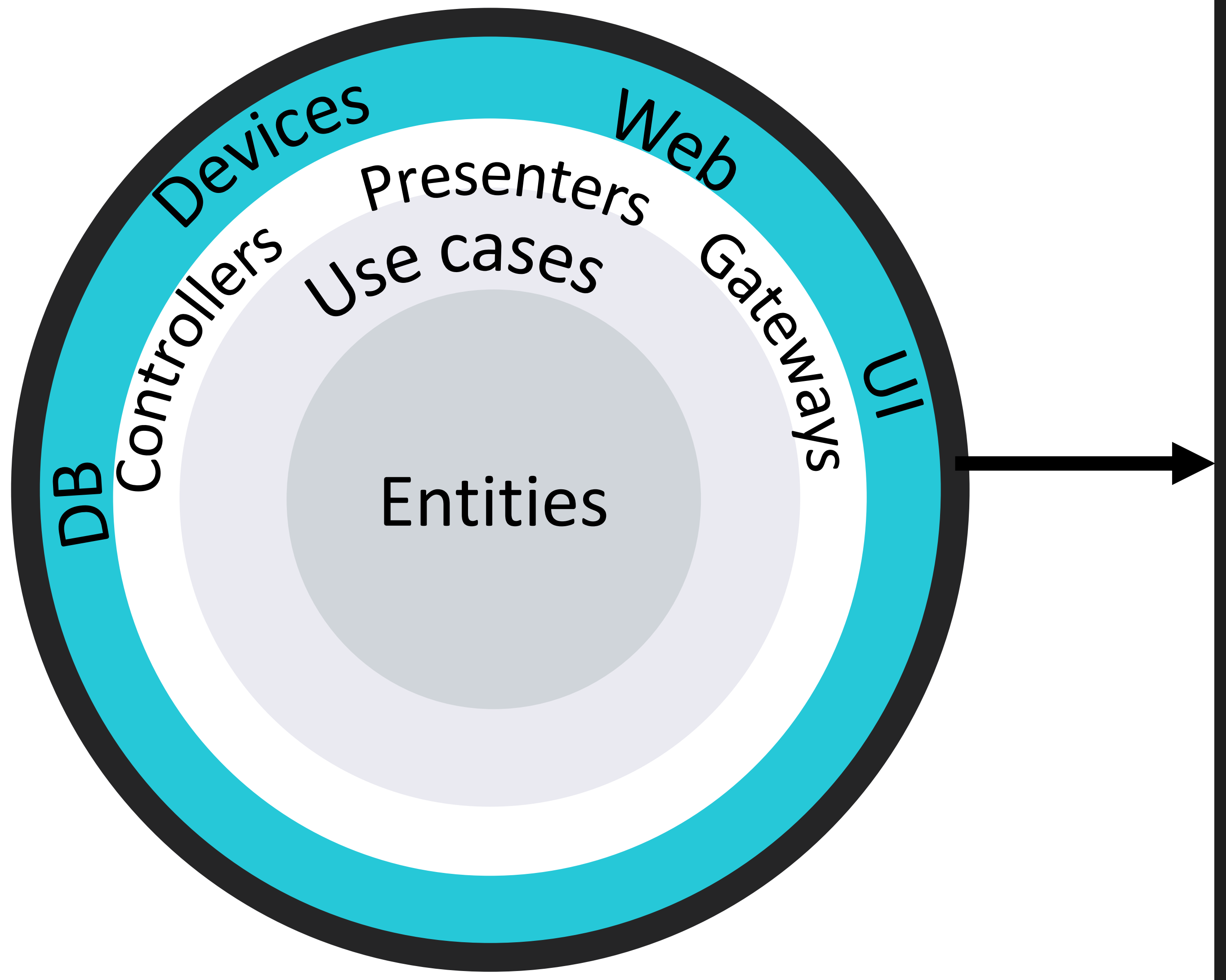


```
import { PaddleCourt } from "../1. Entities - Enterprise Business Rules/PaddleCourt";
import { SportsClubRepository } from "../2. UseCases - Application Business Rules/SportsClubRepository";
import { injectable } from "inversify";

@injectable()
export class SportsClubInMemoryRepository implements SportsClubRepository {
    getAllPaddleCourts(): Promise<Array<PaddleCourt>> {
        return Promise.resolve([
            { number: 5, city: 'Madrid' },
            { number: 1, city: 'Valencia' },
            { number: 2, city: 'Madrid' }
        ]);
    }
}

@injectable()
export class SportsClubWebApiUserInterface implements SportsClubUserInterface {
    static PORT = 3000;
    api = express();

    constructor() {
        this.api.listen(SportsClubWebApiUserInterface.PORT, () => {
            console.log(`web api listening on port ${SportsClubWebApiUserInterface.PORT}`);
        });
    }

    installGetAvailablePaddleCourts(callback: () => Promise<PaddleCourt[]>): void {
        this.api.get('/api/paddle/courts', async (req, res) => {
            const availablePaddleCourts = await callback();
            res.json(availablePaddleCourts);
        });
    }
}
```

DotNet2021

2012

(Robert C. Martin) [4]



Clean architecture
(Hexagonal + Onion architectures)

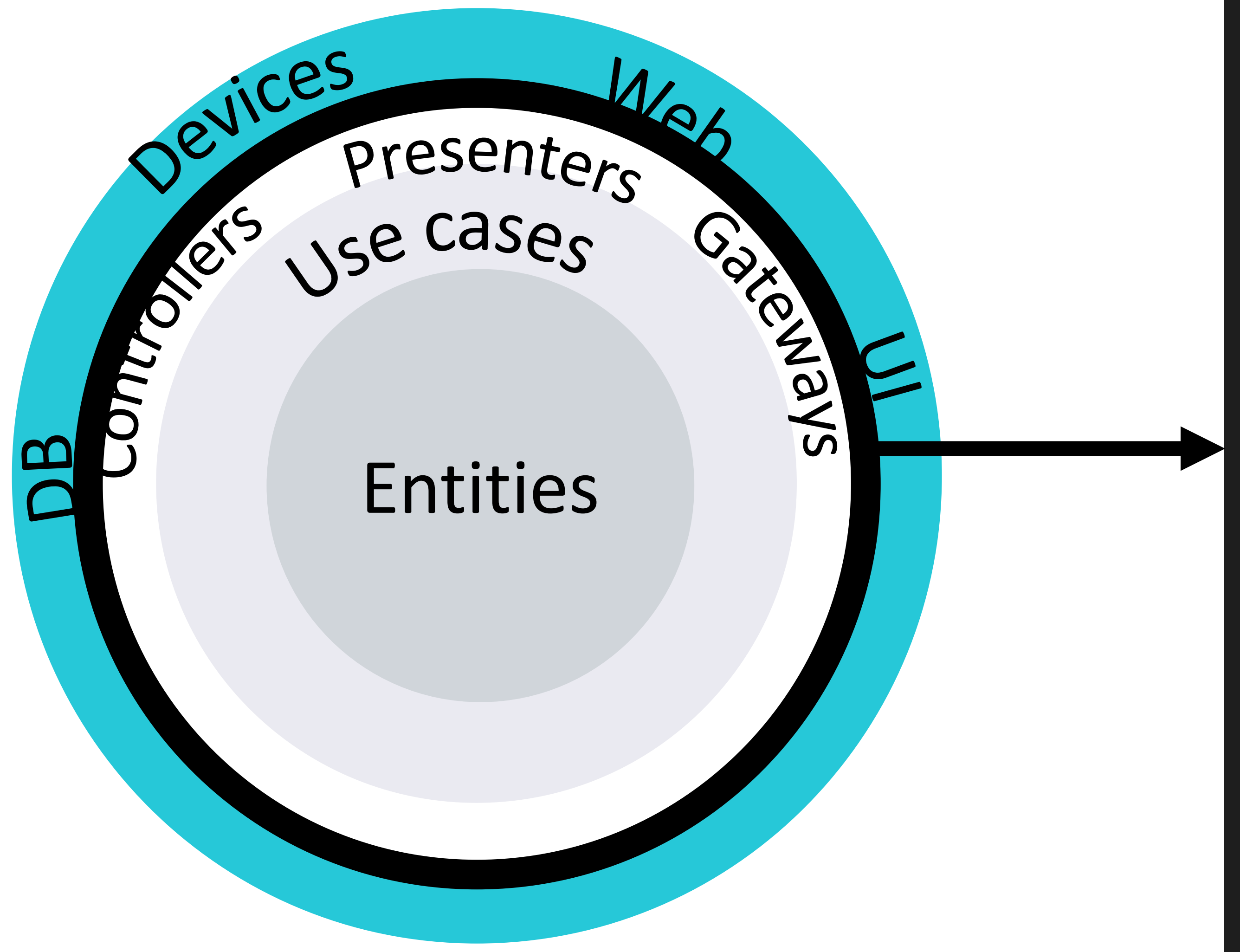
Frameworks & Drivers

```
import { Weather } from "../2. UseCases - Application Business Rules/Weather";
import { City } from "../1. Entities - Enterprise Business Rules/City";
// import axios from 'axios';
import { injectable } from "inversify";

@injectable()
export class WeatherWebApiClient implements Weather {
    isRainingIn(city: City): Promise<boolean> {

        // HTTP call logic: await axios.get('https://weather.com/api/israining')
        return Promise.resolve(false);
    }
}
```

DotNet2021

2012

(Robert C. Martin) [4]



Clean architecture
(Hexagonal + Onion architectures)

Interface Adapters

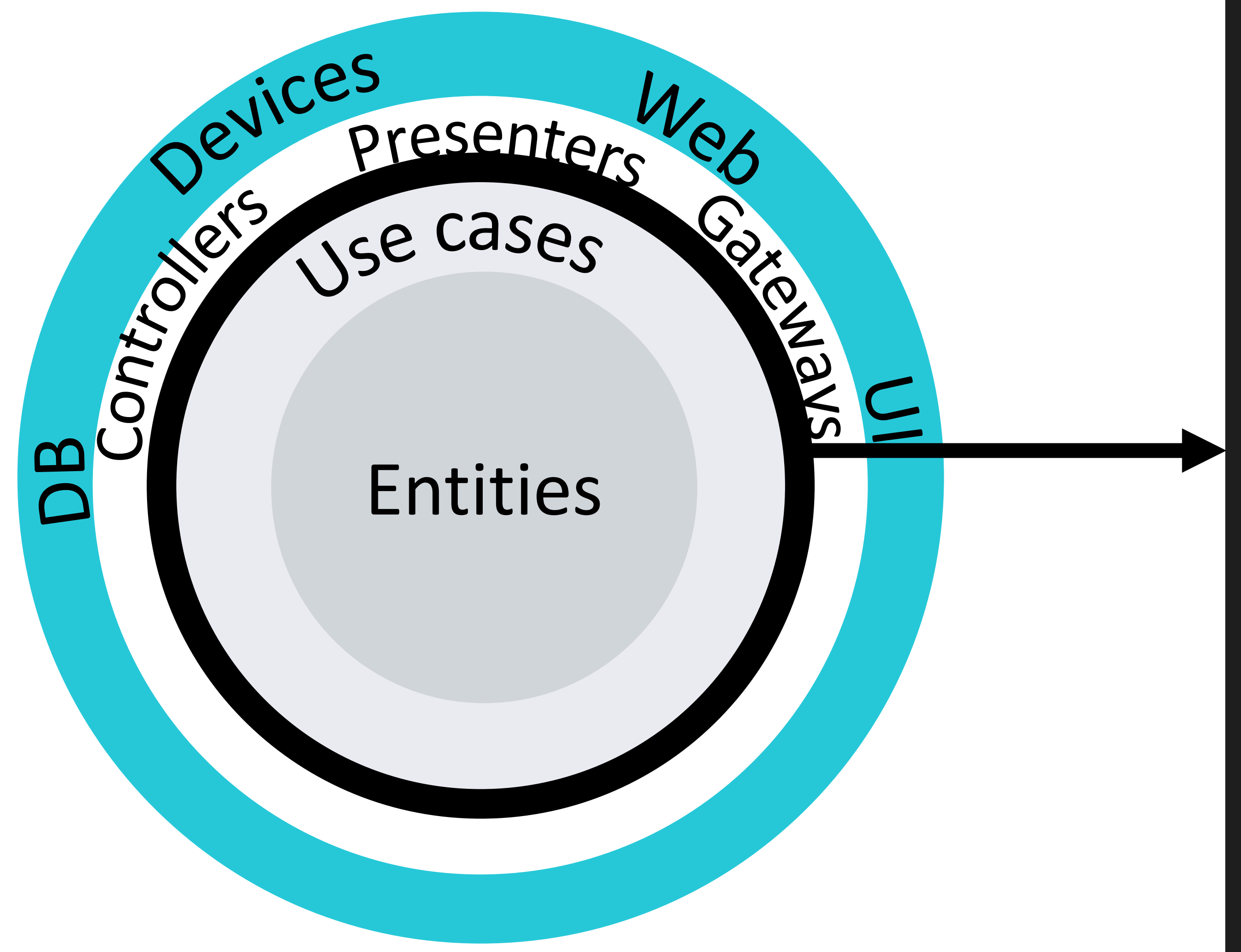
```
@injectable()
export class SportsClub {
    constructor(
        @inject(TYPES.SportsClubUserInterface) private userInterface: SportsClubUserInterface,
        @inject(TYPES.PaddleCourts) private paddleCourts: PaddleCourts
    ) { }

    init() {
        this.userInterface.installGetAvailablePaddleCourts(
            () => this.paddleCourts.getAvailables() // Adapter logic here: from entities to ui structures
        );
    }
}
```

```
import { PaddleCourt } from "../1. Entities - Enterprise Business Rules/PaddleCourt";

export interface SportsClubUserInterface {
    installGetAvailablePaddleCourts(callback: () => Promise<PaddleCourt[]>): void;
}
```

DotNet2021

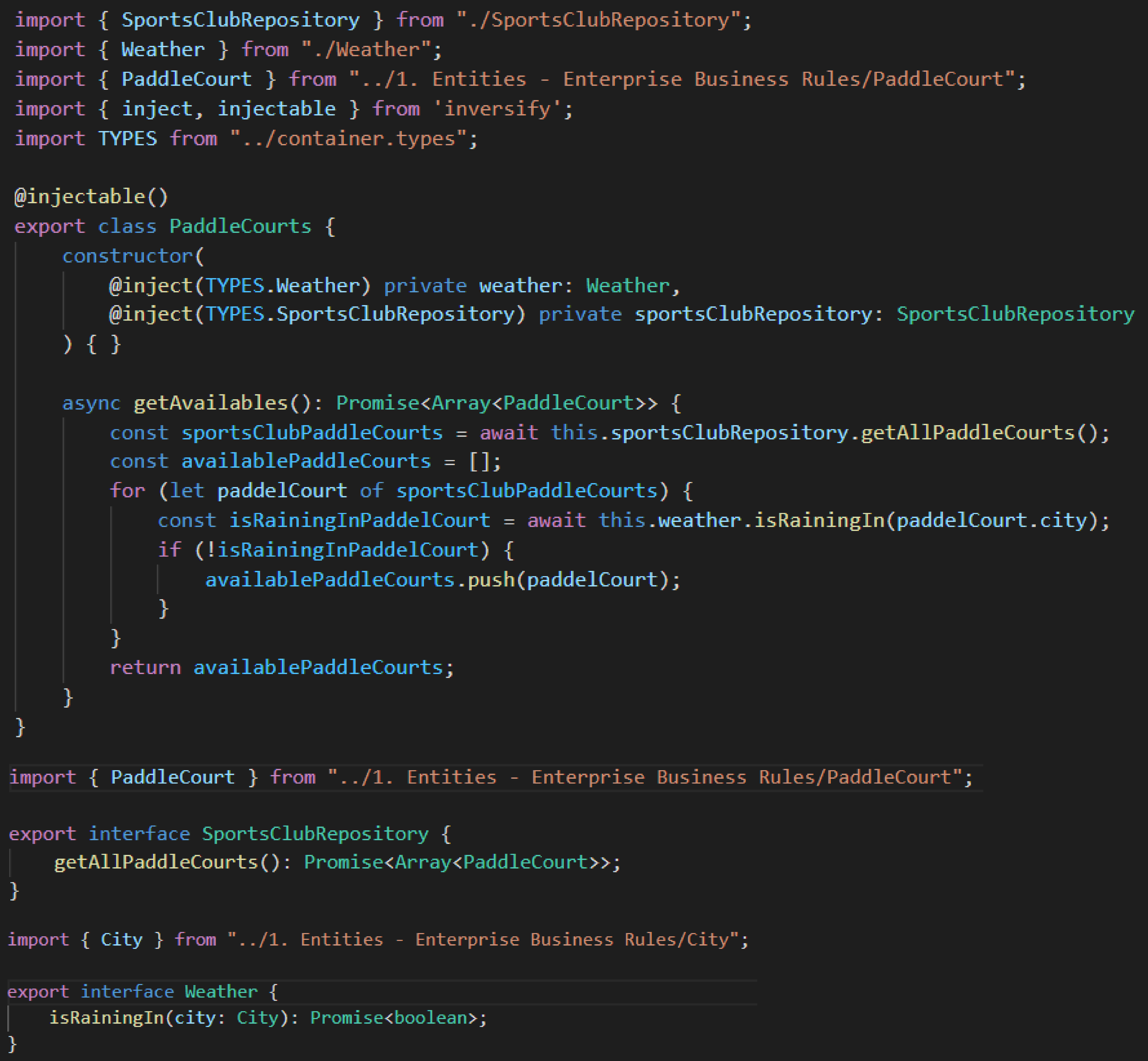
2012

(Robert C. Martin) [4]



Clean architecture
(Hexagonal + Onion architectures)

Application business rules



```
import { SportsClubRepository } from "./SportsClubRepository";
import { Weather } from "./Weather";
import { PaddleCourt } from "../1. Entities - Enterprise Business Rules/PaddleCourt";
import { inject, injectable } from 'inversify';
import TYPES from "../container.types";

@injectable()
export class PaddleCourts {
    constructor(
        @inject(TYPES.Weather) private weather: Weather,
        @inject(TYPES.SportsClubRepository) private sportsClubRepository: SportsClubRepository
    ) { }

    async getAvailables(): Promise<Array<PaddleCourt>> {
        const sportsClubPaddleCourts = await this.sportsClubRepository.getAllPaddleCourts();
        const availablePaddleCourts = [];
        for (let paddelCourt of sportsClubPaddleCourts) {
            const isRainingInPaddelCourt = await this.weather.isRainingIn(paddelCourt.city);
            if (!isRainingInPaddelCourt) {
                availablePaddleCourts.push(paddelCourt);
            }
        }
        return availablePaddleCourts;
    }
}
```

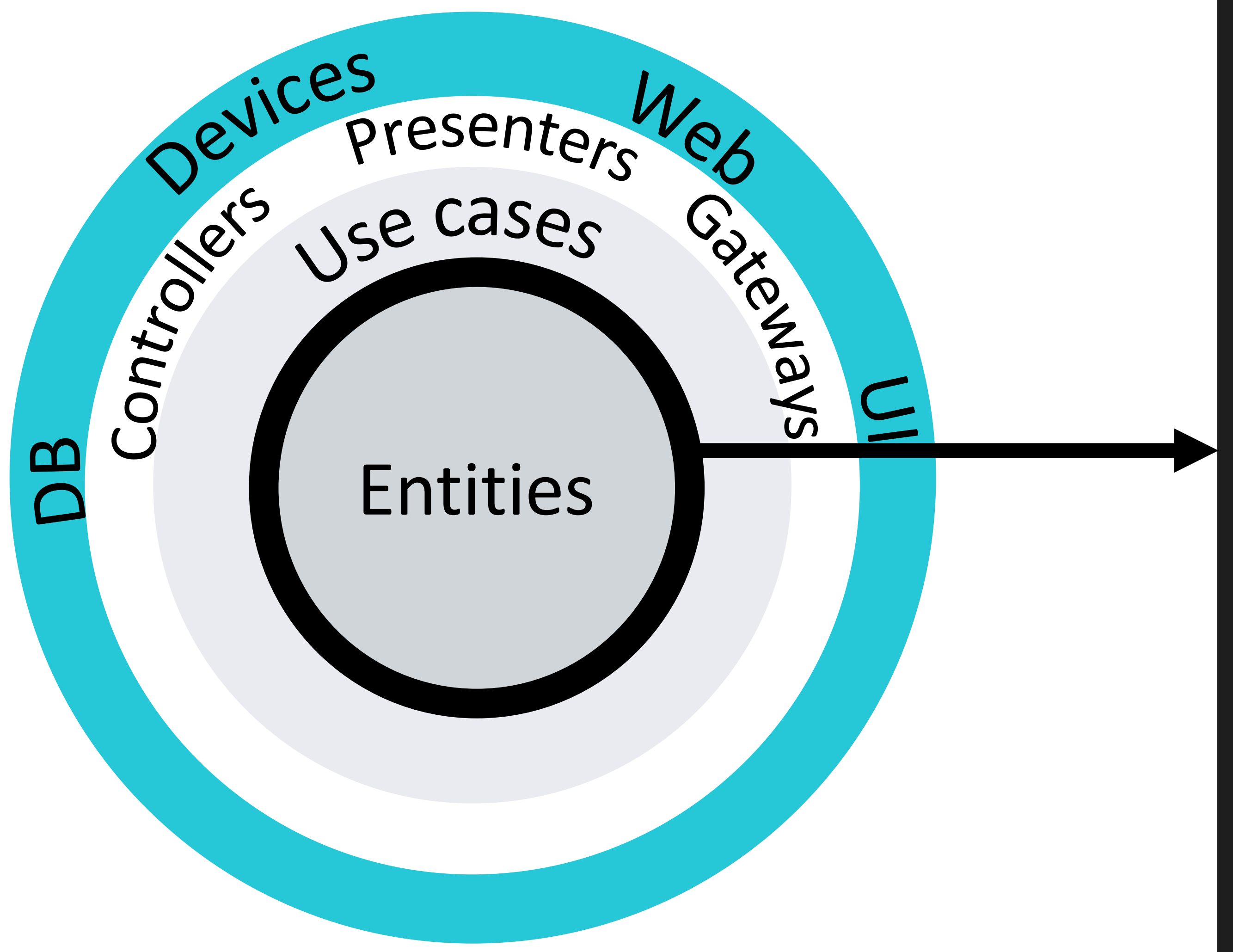
```
import { PaddleCourt } from "../1. Entities - Enterprise Business Rules/PaddleCourt";

export interface SportsClubRepository {
    getAllPaddleCourts(): Promise<Array<PaddleCourt>>;
}
```

```
import { City } from "../1. Entities - Enterprise Business Rules/City";

export interface Weather {
    isRainingIn(city: City): Promise<boolean>;
}
```

DotNet2021

2012

(Robert C. Martin) [4]



Clean architecture
(Hexagonal + Onion architectures)

Enterprise business rules

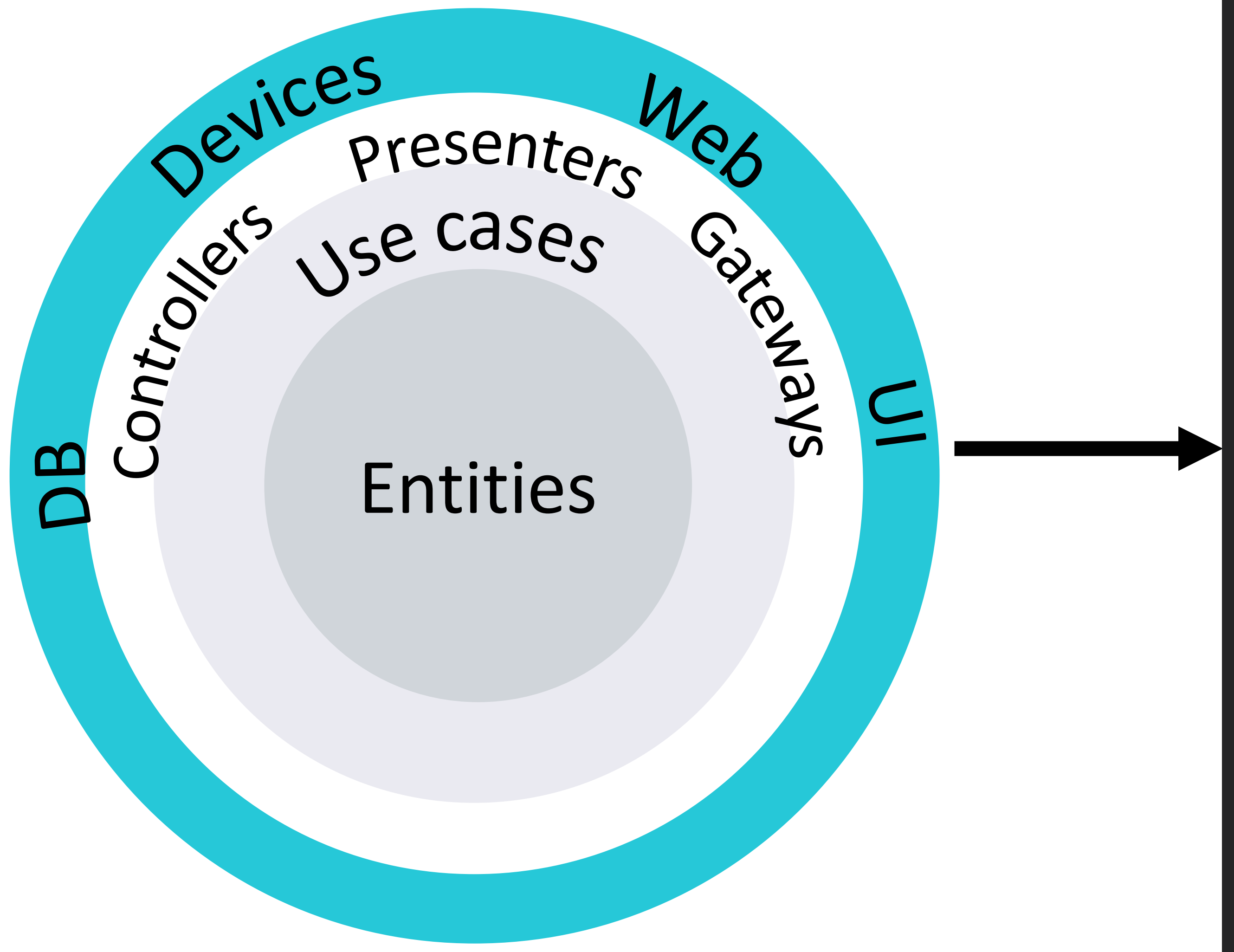
```
export class PaddleCourt {
    number: number;
    city: City
}
```

```
export type City = 'Madrid' | 'Valencia';
```

DotNet2021

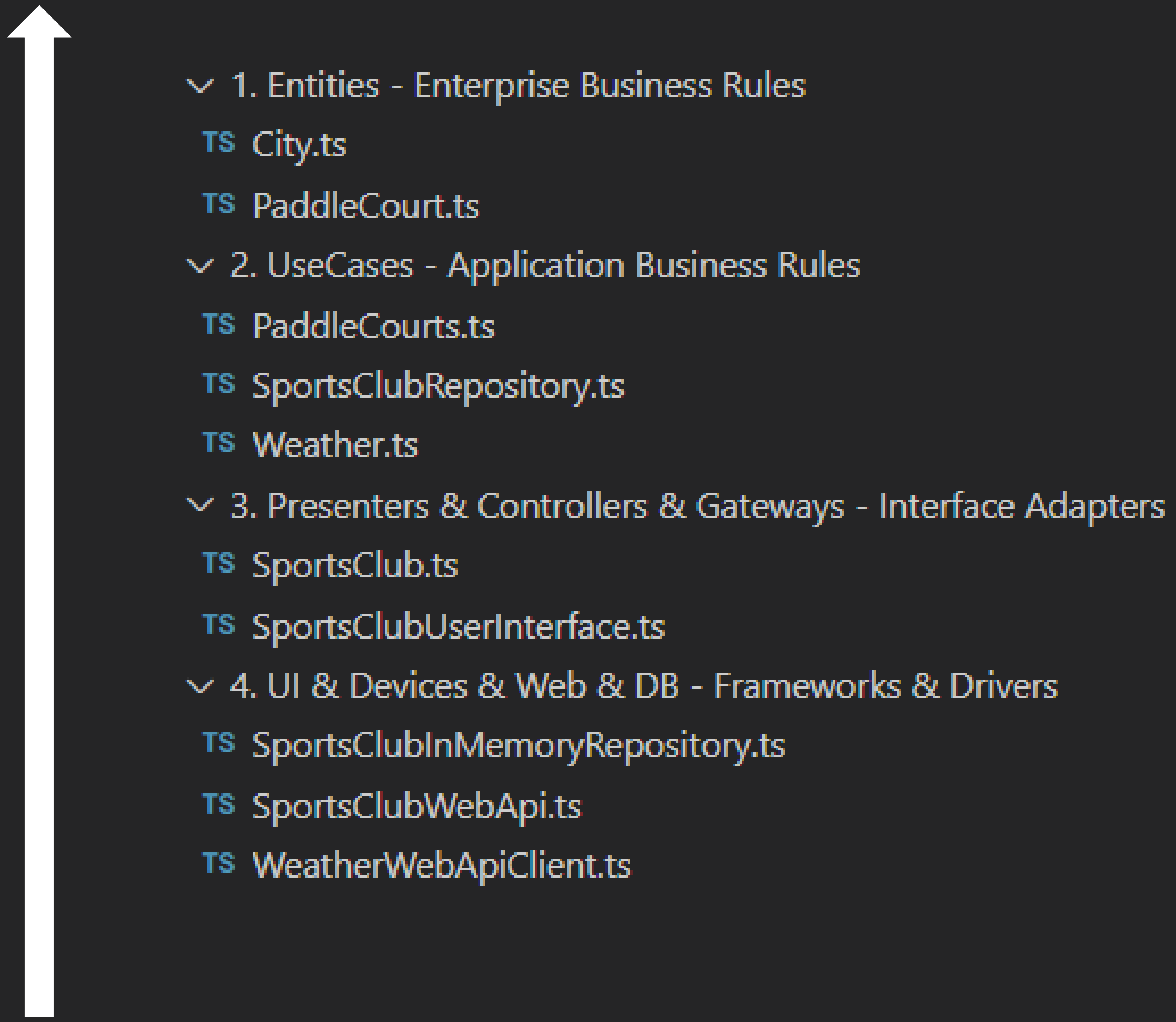
2012

(Robert C. Martin) [4]



Clean architecture
(Hexagonal + Onion architectures)

Dependency direction



- 1. Entities - Enterprise Business Rules
  - City.ts
  - PaddleCourt.ts
- 2. UseCases - Application Business Rules
  - PaddleCourts.ts
  - SportsClubRepository.ts
  - Weather.ts
- 3. Presenters & Controllers & Gateways - Interface Adapters
  - SportsClub.ts
  - SportsClubUserInterface.ts
- 4. UI & Devices & Web & DB - Frameworks & Drivers
  - SportsClubInMemoryRepository.ts
  - SportsClubWebApi.ts
  - WeatherWebApiClient.ts

GitHub

https://github.com/cbastos/dotnet-2021-web-api-architectures

@cbastospc

DotNet
2021

ONLINE TECH CONFERENCE

www.dotnet2021.com

#DotNet2021

Thanks and …
See you soon!

Thanks also to the sponsors. Without whom this would not have been posible.

plain concepts · FUNDACIÓN GOMAESPUMA "Educando con una sonrisa." · Microsoft · intelequia · My Public Inbox · DevsDNA