**blog-api-app-v3 project structure:**

**blog-api-app-v3/**

**├── src/**

**│ ├── api/**

**│ │** └── blog.controller.ts

**│ ├── application/**

**│ │ ├── commands/**

**│ │ │** ├── create-blog.command.ts

│ │ │ ├── delete-all-blogs.command.ts

│ │ │ └── delete-blog-by-id.command.ts

**│ │ ├── dto/**

**│ │ │** └── create-blog.dto.ts

**│ │ ├── handlers/**

**│ │ │** ├── all.handlers.ts

│ │ │ ├── create-blog.handler.ts

│ │ │ ├── delete-all-blogs.handler.ts

│ │ │ ├── delete-blog-by-id.handler.ts

│ │ │ ├── get-all-blogs.handler.ts

│ │ │ └── get-blog-by-id.handler.ts

│ │ │ └── get-joke.handler.ts

**│ │ ├── queries/**

**│ │ │** ├── get-all-blogs.query.ts

│ │ │ ├── get-blog-by-id.query.ts

│ │ │ └── get-joke.query.ts

**│ ├── domain/**

**│ │ └── entities/**

**│ │** └── blog.entity.ts

**│ ├── infrastructure/**

**│ │ ├── database/**

**│ │ │** ├── db-connection.ts

│ │ │ ├── db.module.ts

│ │ │ └── schema.ts

**│ │ └── repositories/**

**│ │** ├── blog-repository-interface.ts

│ │ └── drizzle-blog-repository.ts

**│ ├── app.module.ts**

**│** └── main.ts

├── .env

├── .gitignore

├── docker-compose.yml

├── drizzle.config.ts

├── package.json

├── tsconfig.json

└── README.md

**src/api/blog.controller.ts**

* **Purpose**: Defines the BlogController class, which handles HTTP requests related to blog operations.
* **Essence**: Acts as the entry point for API requests, routing them to appropriate handlers.

**src/application/commands/create-blog.command.ts**

* **Purpose**: Defines the CreateBlogCommand class, encapsulating data required to create a new blog.
* **Essence**: Represents a command in the CQRS pattern for creating a blog.

**src/application/commands/delete-all-blogs.command.ts**

* **Purpose**: Defines the DeleteAllBlogsCommand class, representing a command to delete all blogs.
* **Essence**: Represents a command in the CQRS pattern for deleting all blogs.

**src/application/commands/delete-blog-by-id.command.ts**

* **Purpose**: Defines the DeleteBlogByIdCommand class, encapsulating data required to delete a blog by its ID.
* **Essence**: Represents a command in the CQRS pattern for deleting a blog by ID.

**src/application/dto/create-blog.dto.ts**

* **Purpose**: Defines the CreateBlogDto class, a Data Transfer Object (DTO) used for creating a new blog.
* **Essence**: Ensures data validation and transfer between layers.

**src/application/handlers/all.handlers.ts**

* **Purpose**: Exports arrays of command and query handlers used in the application.
* **Essence**: Centralizes the registration of handlers for easy import.

**src/application/handlers/create-blog.handler.ts**

* **Purpose**: Defines the CreateBlogHandler class, handling the CreateBlogCommand.
* **Essence**: Implements the logic for creating a blog.

**src/application/handlers/delete-all-blogs.handler.ts**

* **Purpose**: Defines the DeleteAllBlogHandler class, handling the DeleteAllBlogsCommand.
* **Essence**: Implements the logic for deleting all blogs.

**src/application/handlers/delete-blog-by-id.handler.ts**

* **Purpose**: Defines the DeleteBlogByIdHandler class, handling the DeleteBlogByIdCommand.
* **Essence**: Implements the logic for deleting a blog by its ID.

**src/application/handlers/get-all-blogs.handler.ts**

* **Purpose**: Defines the GetAllBlogsHandler class, handling the GetAllBlogsQuery.
* **Essence**: Implements the logic for retrieving all blogs.

**src/application/handlers/get-blog-by-id.handler.ts**

* **Purpose**: Defines the GetBlogByIdHandler class, handling the GetBlogByIdQuery.
* **Essence**: Implements the logic for retrieving a blog by its ID.

**src/application/queries/get-all-blogs.query.ts**

* **Purpose**: Defines the GetAllBlogsQuery class, representing a query to retrieve all blogs.
* **Essence**: Represents a query in the CQRS pattern for retrieving all blogs.

**src/application/queries/get-blog-by-id.query.ts**

* **Purpose**: Defines the GetBlogByIdQuery class, encapsulating data required to retrieve a blog by its ID.
* **Essence**: Represents a query in the CQRS pattern for retrieving a blog by its ID.

**src/application/queries/get-joke.query.ts**

* **Purpose**: Defines the GetJokeQuery class, representing a query to retrieve a joke.
* **Essence**: Represents a query in the CQRS pattern for retrieving a joke.

**src/domain/entities/blog.entity.ts**

* **Purpose**: Defines the Blog class, representing the blog entity in the domain layer.
* **Essence**: Models the blog data and business logic.

**src/infrastructure/database/db-connection.ts**

* **Purpose**: Defines a constant DATABASE\_CONNECTION used as a token for injecting the database connection.
* **Essence**: Provides a token for dependency injection of the database connection.

**src/infrastructure/database/db.module.ts**

* **Purpose**: Defines the DatabaseModule class, setting up the database connection using ConfigService and drizzle-orm.
* **Essence**: Configures and provides the database connection.

**src/infrastructure/database/schema.ts**

* **Purpose**: Defines the database schema using drizzle-orm.
* **Essence**: Models the database structure and constraints.

**src/infrastructure/repositories/blog-repository-interface.ts**

* **Purpose**: Defines the BlogRepository interface, outlining methods for interacting with blog data.
* **Essence**: Provides an abstraction for blog data operations.

**src/infrastructure/repositories/drizzle-blog-repository.ts**

* **Purpose**: Defines the DrizzleBlogRepository class, implementing the BlogRepository interface.
* **Essence**: Implements data operations using drizzle-orm.

**src/app.module.ts**

* **Purpose**: Defines the main application module, importing necessary modules and setting up controllers and providers.
* **Essence**: Configures the application and its dependencies.

**src/main.ts**

* **Purpose**: Defines the entry point of the application, creating and starting the NestJS application.
* **Essence**: Boots up the application.

**.env**

* **Purpose**: Stores environment variables.
* **Essence**: Provides configuration values for the application.

**.gitignore**

* **Purpose**: Specifies files and directories to be ignored by Git.
* **Essence**: Prevents unnecessary files from being tracked by version control.

**docker-compose.yml**

* **Purpose**: Defines services, networks, and volumes for Docker containers.
* **Essence**: Facilitates the setup and orchestration of multi-container Docker applications.

**drizzle.config.ts**

* **Purpose**: Configures the drizzle-orm settings.
* **Essence**: Provides configuration for the ORM, such as database connection details and schema settings.

**package.json**

* **Purpose**: Defines the project's dependencies, scripts, and metadata.
* **Essence**: Manages the project's dependencies and scripts.

**tsconfig.json**

* **Purpose**: Configures TypeScript compiler options.
* **Essence**: Specifies TypeScript compilation settings.

**README.md**

* **Purpose**: Provides an overview and documentation for the project.
* **Essence**: Offers information and instructions for users and developers.

**src/app.module.ts**

import { Module } from '@nestjs/common';

import { CqrsModule } from '@nestjs/cqrs';

import { ConfigModule } from '@nestjs/config';

import { BlogController } from './api/blog.controller';

import { DatabaseModule } from './infrastructure/database/db.module';

import { DrizzleBlogRepository } from './infrastructure/repositories/drizzle-blog-repository';

import { BLOG\_REPOSITORY } from './infrastructure/repositories/blog-repository-interface';

import { CommandHandlers, QueryHandlers } from './application/handlers/all.handlers';

@Module({

  imports: [

    ConfigModule.forRoot(), // Loads environment variables from a .env file into process.env

    CqrsModule, // Enables CQRS (Command Query Responsibility Segregation) pattern

    DatabaseModule, // Imports the DatabaseModule which sets up the database connection

  ],

  controllers: [BlogController], // Registers the BlogController to handle incoming HTTP requests

  providers: [

    {

      provide: BLOG\_REPOSITORY, // Provides the BLOG\_REPOSITORY token

      useClass: DrizzleBlogRepository, // Uses DrizzleBlogRepository as the implementation for BLOG\_REPOSITORY

    },

    ...CommandHandlers, // Registers all command handlers

    ...QueryHandlers, // Registers all query handlers

  ],

})

export class AppModule {}

**Explanation**

1. **Imports**:
   * Module: A decorator that marks a class as a NestJS module.
   * CqrsModule: A module that provides CQRS (Command Query Responsibility Segregation) support.
   * ConfigModule: A module that loads environment variables from a .env file into process.env.
   * BlogController: The controller that handles HTTP requests related to blogs.
   * DatabaseModule: A module that sets up the database connection.
   * DrizzleBlogRepository: The implementation of the blog repository using Drizzle ORM.
   * BLOG\_REPOSITORY: A token used for dependency injection of the blog repository.
   * CommandHandlers and QueryHandlers: Arrays of command and query handlers.
2. **@Module Decorator**:
   * imports: Specifies the modules to import.
     + ConfigModule.forRoot(): Loads environment variables from a .env file into process.env. This is useful for configuration management.
     + CqrsModule: Enables the CQRS pattern, which separates read and write operations.
     + DatabaseModule: Sets up the database connection.
   * controllers: Registers the controllers that handle incoming HTTP requests.
     + BlogController: Handles HTTP requests related to blogs.
   * providers: Registers the providers (services, repositories, etc.) that can be injected into other parts of the application.
     + { provide: BLOG\_REPOSITORY, useClass: DrizzleBlogRepository }: Provides the BLOG\_REPOSITORY token and uses DrizzleBlogRepository as its implementation.
     + ...CommandHandlers: Registers all command handlers.
     + ...QueryHandlers: Registers all query handlers.

**Next File: src/api/blog.controller.ts**

import { Controller, Get, Post, Delete, Param, Body } from '@nestjs/common';

import { CommandBus, QueryBus } from '@nestjs/cqrs';

import { CreateBlogDto } from '../application/dto/create-blog.dto';

import { CreateBlogCommand } from '../application/commands/create-blog.command';

import { DeleteBlogByIdCommand } from '../application/commands/delete-blog-by-id.command';

import { GetAllBlogsQuery } from '../application/queries/get-all-blogs.query';

import { GetBlogByIdQuery } from '../application/queries/get-blog-by-id.query';

@Controller('blogs')

export class BlogController {

  constructor(

    private readonly commandBus: CommandBus,

    private readonly queryBus: QueryBus,

  ) {}

  @Post()

  async createBlog(@Body() createBlogDto: CreateBlogDto) {

    const command = new CreateBlogCommand(createBlogDto.title, createBlogDto.content);

    return await this.commandBus.execute(command);

  }

  @Get()

  async getAllBlogs() {

    return await this.queryBus.execute(new GetAllBlogsQuery());

  }

  @Get(':id')

  async getBlogById(@Param('id') id: string) {

    return await this.queryBus.execute(new GetBlogByIdQuery(id));

  }

  @Delete(':id')

  async deleteBlogById(@Param('id') id: string) {

    const command = new DeleteBlogByIdCommand(id);

    return await this.commandBus.execute(command);

  }

}

**Explanation**

1. **Imports**:
   * Controller, Get, Post, Delete, Param, Body: NestJS decorators for defining controllers and routes.
   * CommandBus, QueryBus: CQRS buses for executing commands and queries.
   * CreateBlogDto: DTO for creating a blog.
   * CreateBlogCommand: Command for creating a blog.
   * DeleteBlogByIdCommand: Command for deleting a blog by ID.
   * GetAllBlogsQuery: Query for retrieving all blogs.
   * GetBlogByIdQuery: Query for retrieving a blog by ID.
2. **@Controller Decorator**:
   * @Controller('blogs'): Defines a controller with the base route /blogs.
3. **Constructor**:
   * constructor(private readonly commandBus: CommandBus, private readonly queryBus: QueryBus): Injects the CommandBus and QueryBus.
4. **Methods**:
   * @Post(): Defines a POST route for creating a blog.
     + async createBlog(@Body() createBlogDto: CreateBlogDto): Handles the creation of a blog.
     + const command = new CreateBlogCommand(createBlogDto.title, createBlogDto.content): Creates a new CreateBlogCommand.
     + return await this.commandBus.execute(command): Executes the command using the CommandBus.
   * @Get(): Defines a GET route for retrieving all blogs.
     + async getAllBlogs(): Handles the retrieval of all blogs.
     + return await this.queryBus.execute(new GetAllBlogsQuery()): Executes the query using the QueryBus.
   * @Get(':id'): Defines a GET route for retrieving a blog by ID.
     + async getBlogById(@Param('id') id: string): Handles the retrieval of a blog by ID.
     + return await this.queryBus.execute(new GetBlogByIdQuery(id)): Executes the query using the QueryBus.
   * @Delete(':id'): Defines a DELETE route for deleting a blog by ID.
     + async deleteBlogById(@Param('id') id: string): Handles the deletion of a blog by ID.
     + const command = new DeleteBlogByIdCommand(id): Creates a new DeleteBlogByIdCommand.
     + return await this.commandBus.execute(command): Executes the command using the CommandBus.

**src/application/commands/create-blog.command.ts**

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי**Explanation**

1. **Imports**:
   * ICommand: An interface from @nestjs/cqrs that marks a class as a command.
2. **Class Definition**:
   * export class CreateBlogCommand implements ICommand: Defines a command class for creating a blog.
   * constructor(public readonly title: string, public readonly content: string): The constructor initializes the command with the blog's title and content. The readonly keyword ensures that these properties cannot be modified after initialization.

**src/application/commands/delete-all-blogs.command.ts**

תמונה שמכילה טקסט, גופן, צילום מסך, גרפיקה

התיאור נוצר באופן אוטומטי**Explanation**

1. **Imports**:
   * ICommand: An interface from @nestjs/cqrs that marks a class as a command.
2. **Class Definition**:
   * export class DeleteAllBlogsCommand implements ICommand: Defines a command class for deleting all blogs. This command does not require any additional data, so the class is empty.

**src/application/commands/delete-blog-by-id.command.ts**

תמונה שמכילה טקסט, גופן, צילום מסך

התיאור נוצר באופן אוטומטי**Explanation**

1. **Imports**:
   * ICommand: An interface from @nestjs/cqrs that marks a class as a command.
2. **Class Definition**:
   * export class DeleteBlogByIdCommand implements ICommand: Defines a command class for deleting a blog by its ID.
   * constructor(public readonly id: string): The constructor initializes the command with the blog's ID. The readonly keyword ensures that this property cannot be modified after initialization.

**src/application/dto/create-blog.dto.ts**

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי**Explanation**

1. **Imports**:
   * IsNotEmpty, IsString: Decorators from class-validator used for validating class properties.
2. **Class Definition**:
   * export class CreateBlogDto: Defines a Data Transfer Object (DTO) for creating a blog.
   * @IsNotEmpty() @IsString() title: string: Validates that the title property is a non-empty string.
   * @IsNotEmpty() @IsString() content: string: Validates that the content property is a non-empty string.

**src/application/handlers/all.handlers.ts**

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי**Explanation**

1. **Imports**:
   * Imports all the command and query handlers.
2. **Exports**:
   * export const CommandHandlers: Exports an array of command handlers.
   * export const QueryHandlers: Exports an array of query handlers.

**src/application/handlers/create-blog.handler.ts**

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי**Explanation**

1. **Imports**:
   * CommandHandler, ICommandHandler: Decorators and interfaces from @nestjs/cqrs for handling commands.
   * Inject: A decorator from @nestjs/common for dependency injection.
   * CreateBlogCommand: The command class for creating a blog.
   * BlogRepository, BLOG\_REPOSITORY: The blog repository interface and token.
2. **Class Definition**:
   * @CommandHandler(CreateBlogCommand): Marks the class as a handler for the CreateBlogCommand.
   * export class CreateBlogHandler implements ICommandHandler<CreateBlogCommand>: Defines a handler class for the CreateBlogCommand.
   * constructor(@Inject(BLOG\_REPOSITORY) private readonly blogRepository: BlogRepository): Injects the blog repository.
   * async execute(command: CreateBlogCommand): Promise<void>: Executes the command by calling the createBlog method on the repository.

**src/application/handlers/delete-all-blogs.handler.ts**

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי**Explanation**

1. **Imports**:
   * CommandHandler, ICommandHandler: Decorators and interfaces from @nestjs/cqrs for handling commands.
   * Inject: A decorator from @nestjs/common for dependency injection.
   * DeleteAllBlogsCommand: The command class for deleting all blogs.
   * BlogRepository, BLOG\_REPOSITORY: The blog repository interface and token.
2. **Class Definition**:
   * @CommandHandler(DeleteAllBlogsCommand): Marks the class as a handler for the DeleteAllBlogsCommand.
   * export class DeleteAllBlogsHandler implements ICommandHandler<DeleteAllBlogsCommand>: Defines a handler class for the DeleteAllBlogsCommand.
   * constructor(@Inject(BLOG\_REPOSITORY) private readonly blogRepository: BlogRepository): Injects the blog repository.
   * async execute(command: DeleteAllBlogsCommand): Promise<void>: Executes the command by calling the deleteAllBlogs method on the repository.

**src/application/handlers/delete-blog-by-id.handler.ts**

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי**Explanation**

1. **Imports**:
   * CommandHandler, ICommandHandler: Decorators and interfaces from @nestjs/cqrs for handling commands.
   * Inject: A decorator from @nestjs/common for dependency injection.
   * DeleteBlogByIdCommand: The command class for deleting a blog by its ID.
   * BlogRepository, BLOG\_REPOSITORY: The blog repository interface and token.
2. **Class Definition**:
   * @CommandHandler(DeleteBlogByIdCommand): Marks the class as a handler for the DeleteBlogByIdCommand.
   * export class DeleteBlogByIdHandler implements ICommandHandler<DeleteBlogByIdCommand>: Defines a handler class for the DeleteBlogByIdCommand.
   * constructor(@Inject(BLOG\_REPOSITORY) private readonly blogRepository: BlogRepository): Injects the blog repository.
   * async execute(command: DeleteBlogByIdCommand): Promise<void>: Executes the command by calling the deleteBlogById method on the repository.

**src/application/handlers/get-all-blogs.handler.ts**

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי**Explanation**

1. **Imports**:
   * IQueryHandler, QueryHandler: Decorators and interfaces from @nestjs/cqrs for handling queries.
   * Inject: A decorator from @nestjs/common for dependency injection.
   * GetAllBlogsQuery: The query class for retrieving all blogs.
   * BlogRepository, BLOG\_REPOSITORY: The blog repository interface and token.
2. **Class Definition**:
   * @QueryHandler(GetAllBlogsQuery): Marks the class as a handler for the GetAllBlogsQuery.
   * export class GetAllBlogsHandler implements IQueryHandler<GetAllBlogsQuery>: Defines a handler class for the GetAllBlogsQuery.
   * constructor(@Inject(BLOG\_REPOSITORY) private readonly blogRepository: BlogRepository): Injects the blog repository.
   * async execute(query: GetAllBlogsQuery): Promise<any>: Executes the query by calling the getAllBlogs method on the repository.

**src/application/handlers/get-blog-by-id.handler.ts**

**Explanation**

1. **Imports**:
   * IQueryHandler, QueryHandler: Decorators and interfaces from @nestjs/cqrs for handling queries.
   * Inject: A decorator from @nestjs/common for dependency injection.
   * GetBlogByIdQuery: The query class for retrieving a blog by its ID.
   * BlogRepository, BLOG\_REPOSITORY: The blog repository interface and token.
2. **Class Definition**:
   * @QueryHandler(GetBlogByIdQuery): Marks the class as a handler for the GetBlogByIdQuery.
   * export class GetBlogByIdHandler implements IQueryHandler<GetBlogByIdQuery>: Defines a handler class for the GetBlogByIdQuery.
   * constructor(@Inject(BLOG\_REPOSITORY) private readonly blogRepository: BlogRepository): Injects the blog repository.
   * async execute(query: GetBlogByIdQuery): Promise<any>: Executes the query by calling the getBlogById method on the repository.

**src/application/queries/get-all-blogs.query.ts**

תמונה שמכילה טקסט, גופן, צילום מסך, קו

התיאור נוצר באופן אוטומטי**Explanation**

1. **Imports**:
   * IQuery: An interface from @nestjs/cqrs that marks a class as a query.
2. **Class Definition**:
   * export class GetAllBlogsQuery implements IQuery: Defines a query class for retrieving all blogs. This query does not require any additional data, so the class is empty.

**src/application/queries/get-blog-by-id.query.ts**

תמונה שמכילה טקסט, גופן, צילום מסך

התיאור נוצר באופן אוטומטי**Explanation**

1. **Imports**:
   * IQuery: An interface from @nestjs/cqrs that marks a class as a query.
2. **Class Definition**:
   * export class GetBlogByIdQuery implements IQuery: Defines a query class for retrieving a blog by its ID.
   * constructor(public readonly id: string): The constructor initializes the query with the blog's ID. The readonly keyword ensures that this property cannot be modified after initialization.

**src/domain/entities/blog.entity.ts**

תמונה שמכילה טקסט, צילום מסך, גופן

התיאור נוצר באופן אוטומטי}

**Explanation**

1. **Class Definition**:
   * export class Blog: Defines a class representing a blog entity.
   * constructor(public readonly id: string, public title: string, public content: string): The constructor initializes the blog with an ID, title, and content. The readonly keyword ensures that the ID cannot be modified after initialization.
   * updateTitle(newTitle: string): Updates the blog's title.
   * updateContent(newContent: string): Updates the blog's content.

**src/infrastructure/database/db-connection.ts**

**Explanation**

1. **Constant Definition**:
   * export const DATABASE\_CONNECTION = 'DATABASE\_CONNECTION': Defines a constant DATABASE\_CONNECTION which is used as a token for injecting the database connection.

**src/infrastructure/database/db.module.ts**

תמונה שמכילה טקסט, צילום מסך, תוכנה

התיאור נוצר באופן אוטומטי: [DATABASE\_CONNECTION],

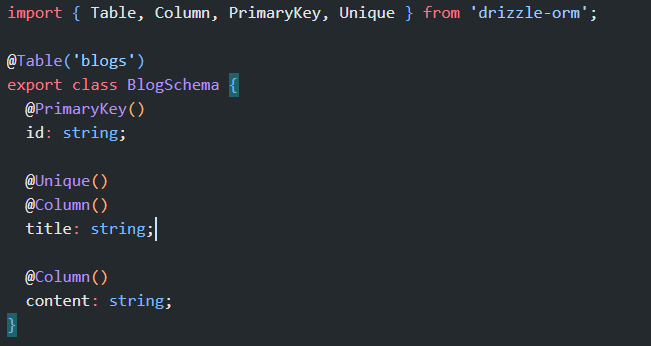
})

export class DatabaseModule {}

**Explanation**

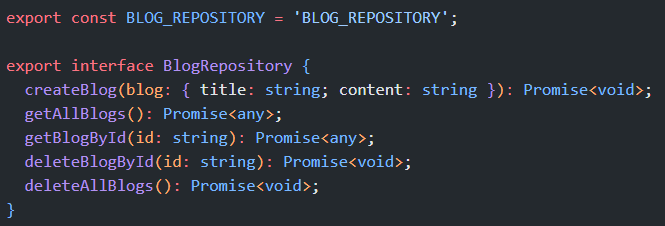
1. **Imports**:
   * Module: A decorator that marks a class as a NestJS module.
   * ConfigService, ConfigModule: Services and modules from @nestjs/config for configuration management.
   * DATABASE\_CONNECTION: The token for injecting the database connection.
   * createConnection: A function from drizzle-orm for creating a database connection.
2. **@Module Decorator**:
   * imports: Specifies the modules to import.
     + ConfigModule: Imports the configuration module.
   * providers: Registers the providers (services, repositories, etc.) that can be injected into other parts of the application.
     + { provide: DATABASE\_CONNECTION, useFactory: async (configService: ConfigService) => { ... }, inject: [ConfigService] }: Provides the DATABASE\_CONNECTION token and uses a factory function to create the database connection. The ConfigService is injected to access configuration values.
   * exports: Specifies the providers to export.
     + DATABASE\_CONNECTION: Exports the DATABASE\_CONNECTION token.

**src/infrastructure/database/schema.ts**

**Explanation**

1. **Imports**:
   * Table, Column, PrimaryKey, Unique: Decorators from drizzle-orm for defining database schema.
2. **Class Definition**:
   * @Table('blogs'): Marks the class as a table named blogs.
   * export class BlogSchema: Defines a class representing the blog schema.
   * @PrimaryKey() id: string: Marks the id property as the primary key.
   * @Unique() @Column() title: string: Marks the title property as a unique column.
   * @Column() content: string: Marks the content property as a column.

**src/infrastructure/repositories/blog-repository-interface.ts**



**Explanation**

1. **BLOG\_REPOSITORY Constant**:
   * export const BLOG\_REPOSITORY = 'BLOG\_REPOSITORY': Defines a constant for the blog repository token.
2. **BlogRepository Interface**:
   * export interface BlogRepository: Defines the blog repository interface.
   * createBlog(blog: { title: string; content: string }): Promise<void>: Method for creating a blog.
   * getAllBlogs(): Promise<any>: Method for retrieving all blogs.
   * getBlogById(id: string): Promise<any>: Method for retrieving a blog by its ID.
   * deleteBlogById(id: string): Promise<void>: Method for deleting a blog by its ID.
   * deleteAllBlogs(): Promise<void>: Method for deleting all blogs.

**src/infrastructure/repositories/drizzle-blog-repository.ts**

תמונה שמכילה טקסט, צילום מסך

התיאור נוצר באופן אוטומטי**Explanation**

1. **Imports**:
   * Injectable: A decorator that marks a class as a provider.
   * BlogRepository: The blog repository interface.
   * DATABASE\_CONNECTION: The database connection token.
   * Inject: A decorator for injecting dependencies.
   * Connection: A class from drizzle-orm representing a database connection.
2. **DrizzleBlogRepository Class**:
   * @Injectable(): Marks the class as a