Path of the starting point of the program:

main\src\main\java\com\example\diplomenproekt\DiplomenProektApplication.java

Files located in packedge **com.example.diplomenproekt.config:**

ApplicationBeanConfiguration.java:

- Configuration class for defining application beans.
- Contains methods annotated with **@Bean** to define beans.
- Defines beans for **ModelMapper** and **PasswordEncoder**.
- Uses **ModelMapper** to facilitate object mapping.
- Uses **Pbkdf2PasswordEncoder** as the password encoder for security purposes.

Files located in package **com.example.diplomenproekt.modules:**

AddCompanyBindingModel.java:

- Represents a binding model for adding a company.
- Contains fields for company name, manager's name, VAT number, address, and registration status.

AddToCartBindingModel.java:

- Represents a binding model for adding a product to the shopping cart.
- Contains a field for the portion of the product to add to the cart.

CheckoutBindingModel.java:

- Represents a binding model for the checkout process.
- Contains fields for the delivery address and total price of the order.

ProductAddBindingModel.java:

- Represents a binding model for adding a product.
- Contains fields for product name, price, minimum price, quantity, category, company name, and URL.

ProfileUpdateBindingModel.java:

- Represents a binding model for updating user profile information.
- Contains fields for first name, last name, email, old password, new password, and password confirmation.

PromotionAddBindingModel.java:

- Represents a binding model for adding a promotion.
- Contains fields for promotion amount, category, and company name.

PromotionBindingModel.java:

- Represents a binding model for promotions.
- No specific fields are defined in this class.

UserLoginBindingModel.java:

- Represents a binding model for user login.
- Contains fields for email and password.

UserRegisterBindingModel.java:

- Represents a binding model for user registration.
- Contains fields for first name, last name, password, and email.

UserDto.java:

- This `UserDto` class represents a data transfer object (DTO) for user information. It contains fields for the user's first name, last name, and email address.
- The class provides getter and setter methods for each field to allow accessing and modifying the user's information.
- The getter methods (`getFirstName()`, `getLastName()`, and `getEmail()`) retrieve the corresponding field values, while the setter methods (`setFirstName()`, `setLastName()`,

and `setEmail()`) set the values of the fields and return the modified `UserDto` object to support method chaining.

- This class is used to transfer user data between different layers of an application.

Files located in the package **com.example.diplomenproekt.services.impl:**

UserServiceImpl.java:

- It handles user-related operations like login, registration, updating profile, checking password, finding product cart total price, and confirming user registration.
- Dependencies:
    - **UserRepository**: Used for database operations related to users.
    - **ModelMapper**: Utilized for mapping between different objects.
    - **PasswordEncoder**: Used for encoding passwords.
    - **JavaMailSender**: Utilized for sending confirmation emails.
- Notable methods:
    - **getAllUsers()**: Retrieves a list of all users.
    - **login ()**: Handles user login functionality.
    - **register ()**: Registers a new user.
    - **getUserInfo()**: Retrieves user information for profile update.
    - **updateProfile()**: Updates user profile information.
    - **checkIfPasswordExists()**: Checks if the provided password matches the user's password.
    - **findProductsCartTotalPrice()**: Calculates the total price of products in the user's cart.
    - **confirm ()**: Confirms user registration.

HistoryServiceImpl.java:

- It handles history-related operations like secure checkout, retrieving history products, and clearing history.
- Dependencies:
    - **UserRepository**: Used for database operations related to users.
    - **ProductRepository**: Used for database operations related to products.
    - **HistoryRepository**: Used for database operations related to user history.

    - **ShoppingCartRepository**: Used for database operations related to shopping carts.
- Notable methods:
    - **secureCheckout()**: Handles the secure checkout process, including saving order history and clearing the shopping cart.
    - **findAllHistoryProducts()**: Retrieves all historical products for a user.
    - **clearHistory()**: Clears the user's history, including removing history products and resetting user history fields.

CompanyServiceImpl.java:

- It handles operations related to companies, such as retrieving all companies for a user, adding a new company, and retrieving all profile companies for a user.
- Dependencies:
    - **CompanyRepository**: Used for database operations related to companies.
    - **ModelMapper**: Utilized for mapping between different objects.
    - **UserRepository**: Used for database operations related to users.
- Notable methods:
    - **getAllCompanies()**: Retrieves all companies associated with a specific user.
    - **addCompany()**: Adds a new company for a user.
    - **getAllProfileCompanies()**: Retrieves all companies associated with the user profile.

ProductServiceImpl.java:

- It handles operations related to products, such as adding a new product, retrieving all products, retrieving products by category, finding a product by ID, checking quantity availability, adding a product to the cart, and finding the maximum price of a product.
- Dependencies:
    - **ProductRepository**: Used for database operations related to products.
    - **ModelMapper**: Utilized for mapping between different objects.
    - **CompanyRepository**: Used for database operations related to companies.
    - **UserRepository**: Used for database operations related to users.
    - **ShoppingCartService**: Used for operations related to the shopping cart.
- Notable methods:
    - **addProduct()**: Adds a new product to the database.
    - **getAllProducts()**: Retrieves all products.

    - Methods like **getAllVideocards()**, **getAllProcessors()**, etc.: Retrieve products by specific categories.
    - **findProductById()**: Finds a product by its ID.
    - **checkQuantityAvailability()**: Checks if the requested quantity of a product is available.
    - **addProductToCart()**: Adds a product to the user's shopping cart.
    - **getProductMaxPrice()**: Retrieves the maximum price of a product.

PromotionServiceImpl.java:

- It handles operations related to promotions, such as adding a new promotion.
- Dependencies:
    - **PromotionRepository**: Used for database operations related to promotions.
    - **CompanyRepository**: Used for database operations related to companies.
- Notable methods:
    - **addPromotion()**: Adds a new promotion to the database. The promotion is in percentages and is added to all the products of the company. There could be more than one promotion for a company, but the price of a product cannot drop below the set minimum price.

ShoppingCartServiceImpl.java:

- It handles operations related to the shopping cart, such as adding a new shopping cart entity, retrieving all shopping cart entities, and saving an existing shopping cart entity.
- Dependencies:
    - **ShoppingCartRepository**: Used for database operations related to the shopping cart.
- Notable methods:
    - **addShoppingCartEntity()**: Adds a new shopping cart entity to the database.
    - **findAllShoppingCartEntities()**: Retrieves all shopping cart entities from the database.
    - **saveShoppingCart()**: Saves an existing shopping cart entity to the database.

Files located in the package **com.example.diplomenproekt.web:**

CompanyController.java:

- Controller class handling requests related to companies.
- Defines methods for adding a company and adding a promotion.
- Uses **CompanyService** and **PromotionService** for business logic.
- Utilizes **ModelMapper** for mapping objects.
- Handles HTTP sessions for user authentication.

HomeController.java:

- Controller class for handling home-related requests.
- Defines a method for accessing the home page.
- Redirects to the product catalog page if a user is logged in.
- Utilizes **ModelMapper** for mapping objects.

MaxPriceRestController.java:

- REST controller class for handling requests related to retrieving the maximum price of products.
- Defines a method to get the maximum price of products.
- Utilizes **ProductsService** for business logic.

ProductController.java:

- Controller class handling requests related to products.
- Defines methods for adding a product, viewing a product catalog, viewing product details, adding products to the shopping cart, checking out, viewing order details, and viewing purchase history.
- Uses services such as **ProductsService**, **CompanyService**, **ShoppingCartService**, **UserService**, and **HistoryService** for business logic.
- Manages HTTP sessions for user authentication.

UserController.java:

- Controller class handling requests related to users.
- Defines methods for managing user registration, login, logout, profile viewing, profile editing, confirming email, and viewing companies associated with a user.
- Utilizes services such as **UserService** and **CompanyService** for business logic.
- Manages HTTP sessions for user authentication.

Error handling: **main\src\main\resources\templates\error**

In the file are the definition of the handling of the most common error caused by users:

- **400 Bad Request**: The server cannot process the request due to malformed syntax or invalid request message framing.
- **401 Unauthorized**: The request requires user authentication. The client must provide proper authentication credentials to access the resource.
- **403 Forbidden**: The server understood the request, but it refuses to authorize it. The client does not have permission to access the requested resource.
- **404 Not Found**: The server cannot find the requested resource. It may be removed, moved, or temporarily unavailable.
- **405 Method Not Allowed**: The request method is not supported for the requested resource. For example, trying to POST to a resource that only supports GET requests.
- **500 Internal Server Error**: A generic error message indicating that the server encountered an unexpected condition that prevented it from fulfilling the request.

Configuration files:

DiplomenProekt.imp:

XML file that configures JPA and Spring facets for a project. It specifies JPA settings such as validation and provider, along with data source mappings. The Spring facet has no additional configuration.

mvmw.cmd:

This is a batch script used to start up Maven on Windows systems. It sets up environment variables, determines the location of Java and Maven, and executes the Maven wrapper if necessary to download the Maven wrapper JAR file. Then it launches Maven using the specified JVM options and classpath.

DiplomenProekt.imp, mvmw, mvmw.cmd, packedge-lock.json and pom.xml go to a configuration file.

Database Structure:

Tables:

1. **users:**
   - Stores information about users.
   - Fields:
     - id: Primary key
     - firstName: First name of the user
     - lastName: Last name of the user
     - password: Password of the user
     - isConfirmed: Boolean indicating whether the user's account is confirmed
     - email: Email address of the user
2. **company:**
   - Stores information about companies.
   - Fields:
     - id: Primary key
     - name: Name of the company
     - mol: Manager of the company
     - vat: VAT (Value Added Tax) of the company
     - address: Address of the company
     - is_registered: Boolean indicating whether the company is registered
     - income: Income of the company
     - user: Foreign key referencing the user who owns the company
3. **products:**
   - Stores information about products.
   - Fields:
     - id: Primary key
     - name: Name of the product
     - price: Price of the product
     - minprice: Minimum price of the product
     - quantity: Quantity of the product available
     - url: URL of the product
     - category: Category of the product

- company: Foreign key referencing the company that owns the product

4. **promotion:**

- Stores information about promotions.
- Fields:
  - id: Primary key
  - amount: Amount of the promotion
  - category: Category of the promotion
  - company: Foreign key referencing the company associated with the promotion

5. **shopping_cart:**

- Stores information about shopping cart items.
- Fields:
  - id: Primary key
  - wishedQuantityForOrder: Quantity of the product wished for order
  - product: Foreign key referencing the product in the cart
  - user: Foreign key referencing the user who owns the cart

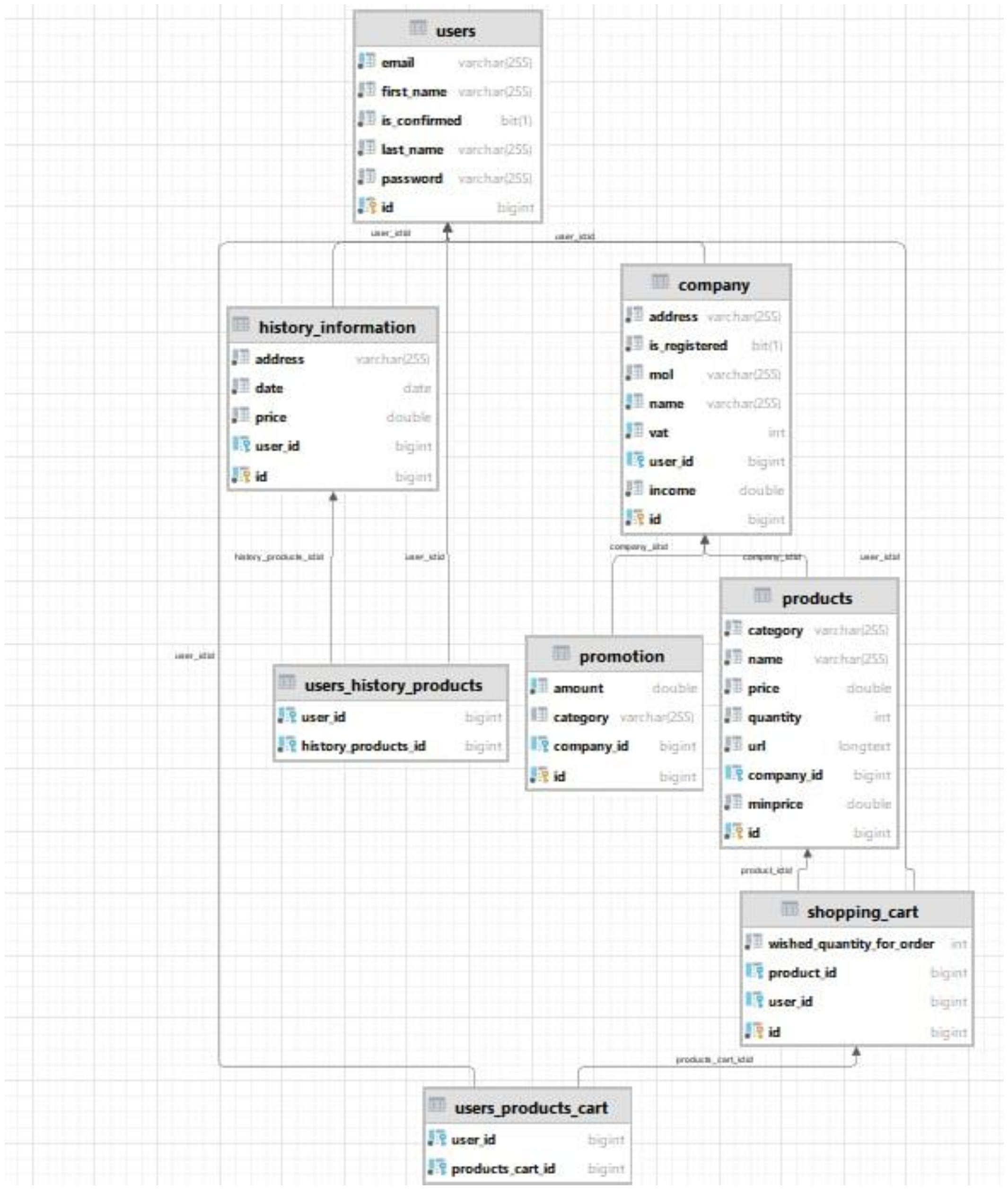
6. **history_information:**

- Stores historical information about transactions.
- Fields:
  - id: Primary key
  - address: Address associated with the transaction
  - date: Date of the transaction
  - price: Price of the transaction
  - user: Foreign key referencing the user associated with the transaction

Relationships:

- **User-Company:** One-to-Many relationship where a user can own multiple companies.
- **Company-Product:** One-to-Many relationship where a company can have multiple products.
- **Company-Promotion:** One-to-Many relationship where a company can have multiple promotions.
- **User-ShoppingCart:** One-to-Many relationship where a user can have multiple items in the shopping cart.
- **User-History:** One-to-Many relationship where a user can have multiple historical transactions.

ER Diagram (Entity Relationship Diagram):


users
email varchar(255)
first_name varchar(255)
is_confirmed bit(1)
last_name varchar(255)
password varchar(255)
id bigint
user_id:id
user_id:id
company
address varchar(255)
is_registered bit(1)
mol varchar(255)
name varchar(255)
vat int
user_id bigint
income double
id bigint
history_information
address varchar(255)
date date
price double
user_id bigint
id bigint
history_products_id:id
user_id:id
company_id:id
company_id:id
user_id:id
products
category varchar(255)
name varchar(255)
price double
quantity int
url longtext
company_id bigint
minprice double
id bigint
user_id:id
users_history_products
user_id bigint
history_products_id bigint
promotion
amount double
category varchar(255)
company_id bigint
id bigint
product_id:id
shopping_cart
wished_quantity_for_order int
product_id bigint
user_id bigint
id bigint
products_cart_id:id
users_products_cart
user_id bigint
products_cart_id bigint