**BankManagement**

**class** BankManagementApplication

کلاس BankManagementApplication:  
   این کلاس وظیفه‌ی مدیریت کلی برنامه بانکی را دارد. معمولاً در این کلاس روش‌های main() و لاگ‌های برنامه قرار می‌گیرد. ممکن است در این کلاس تنظیمات اولیهٔ برنامه و راه‌اندازی اولیهٔ سیستم نیز انجام شود.

**class** SecurityConfig

کلاس SecurityConfig:  
   این کلاس وظیفهٔ تنظیمات امنیتی مرتبط با برنامه بانکی را برعهده دارد. از جمله وظایف این کلاس میتوان به تنظیم دسترسی‌ها، تعیین سطوح دسترسی کاربران، مدیریت توکن‌ها، مدیریت جلسات (session) و رمزنگاری اطلاعات اشاره کرد.

**class** WebConfig

کلاس WebConfig:  
   این کلاس به تنظیمات مربوط به وب برنامه بانکی می‌پردازد. این شامل تنظیم پورت و هاست سرور، تنظیم اتصال به پایگاه داده، مدیریت APIها و تنظیمات مرتبط با فرانت‌اند و بک‌اند برنامه می‌باشد.

**package** com.example.BankManagement.**advice**;

public class ApplicationExceptionHandler {}

این کلاس با عنوان ApplicationExceptionHandler مسئول کنترل و مدیریت خطاها و استثنائات در اپلیکیشن بانکی می‌باشد. این کلاس توابع و روش های بررسی و پردازش خطا را در اختیار دارد.

**package** com.example.BankManagement.**enums**;

public enum Roles { ADMIN, USER }

این enum با نام Roles ، با دو عضو ADMIN و USER  نقش های مختلف کاربران در اپلیکیشن بانکی را نمایندگی می‌کند. این enum معمولاً در سیستم های احراز هویت و دسترسی به کاربران مورد استفاده قرار می‌گیرد و با تعریف نقش ها و مجوزها، اجازه‌ی دسترسی به عملکردها و منابع مختلف را به کاربران مدیریت می‌کند.

**package** com.example.BankManagement.**exception**;

public class UserNotFondException {}

 این کلاس با عنوان UserNotFondException، به عنوان یک استثناء به صورت خاص برای مدیریت و پردازش موارد یافت نشدن کاربر در سیستم بانکی استفاده می‌شود. به عبارت دیگر، وقتی که برنامه به هر دلیلی قادر به یافتن یک کاربر خاص در دیتابیس نمی‌باشد، این استثناء پرتاب می‌شود تا برنامه بتواند مصرف کننده این استثناء را مطلع کند و اقدامات لازم را برای مدیریت و پردازش آن انجام دهد.

**package** com.example.BankManagement.**modules**;

***package*** *account*

**class** AccountConvertor {}

کلاس AccountConvertor:  
 این کلاس برای تبدیل اطلاعات حساب بانکی به دیتا ترانسفر اشیا (DTO) و بالعکس استفاده می‌شود.

**class** AccountDTO {}

کلاس AccountDTO:  
 این کلاس حاوی اطلاعات حساب بانکی است که برای ارتباط با لایه‌های دیگر سیستم استفاده می‌شود.

**package** **repository;**

public interface AccountRepository {}

رابط AccountRepository:  
 این رابط متدهایی را برای انجام عملیات ماندگاری اطلاعات حساب بانکی در پایگاه داده ارائه می‌دهد.

**package model;**

public class Account {}

کلاس Account:  
 این کلاس مدلی از حساب بانکی را نمایش می‌دهد و شامل فیلدها و روشهای مربوط به حساب بانکی است.

**package service;**

public class AccountService {

کلاس AccountService:  
 این کلاس عملیات مربوط به مدیریت حساب بانکی را ارائه می‌دهد، مانند ایجاد حساب برای مشتری، دریافت حساب بر اساس شناسه و دریافت لیست تمام حسابها.

public AccountDTO createAccountForCustomer() {}

این متد برای ایجاد یک حساب جدید برای مشتری استفاده میشود. این متد مقادیر مورد نیاز برای ایجاد حساب را از درخواست دریافتی دریافت میکند و سپس حساب جدید را ایجاد کرده و DTO مربوط به آن را برمیگرداند.

public Account getById() {}

این متد برای دریافت یک حساب بر اساس شناسه آن مورد استفاده قرار میگیرد. شناسه مورد نظر از درخواست دریافت شده به عنوان ورودی دریافت میشود و سپس حساب متناظر با آن شناسه را برمیگرداند.

public List<AccountDTO> getAllAccounts() {}

این متد برای دریافت لیستی از تمامی حساب ها استفاده میشود. ممکن است نیاز به جستجو، فیلترینگ یا مرتب سازی روی حساب ها باشد. معمولاً این متد تمامی حساب ها را از مخزن مورد استفاده دریافت کرده و یک لیست از DTO های مربوط به حساب ها را برمیگرداند.

}

**package controller;**

public class AccountController {

کلاس AccountController:  
 این کلاس برای ارائه و مدیریت روت‌ها و درخواستهای ورودی و خروجی مربوط به مدیریت حساب بانکی استفاده می‌شود. شامل متدهایی مانند ثبت حساب و دریافت لیست تمام حسابها.

public ResponseEntity<AccountDTO> registerAccount() { }

این متد برای ثبت یک حساب کاربری جدید استفاده می‌شود. آرگومان‌های ورودی این متد شامل اطلاعات حسابی که کاربر می‌خواهد ثبت کند می‌باشد. نتیجه بازگشتی این متد شامل شیء AccountDTOاست که مشخصات حساب کاربری جدید ثبت شده را نشان می‌دهد.

public List<AccountDTO> getAllAccounts() { }

این متد برای دریافت تمامی حساب‌های کاربری موجود استفاده می‌شود. نتیجه بازگشتی این متد یک لیست از شیء‌های AccountDTO  می‌باشد که مشخصات تمام حساب‌های کاربری موجود را نشان می‌دهد.

}

***package*** *bankBranch*

**class** BankBranchConvertor {}

کلاس BankBranchConvertor:  
این کلاس مسئول تبدیل اطلاعات بانک به DTO (Data Transfer Object) میباشد. یعنی اطلاعات بانک (BankBranch) را به شکلی مناسب برای انتقال در محیط های دیگر تبدیل میکند.

**class** BankBranchDTO {}

کلاس BankBranchDTO:  
این کلاس حاوی اطلاعات مربوط به بانک (BankBranch) است و برای انتقال داده ها در محیط های دیگر استفاده میشود.

**package repository;**

public interface BankBranchRepository {}

رابط BankBranchRepository:  
این رابط مسئول ارتباط با پایگاه داده برای عملیات CRUD (Create, Read, Update, Delete) بر روی اطلاعات بانک (BankBranch) میباشد.

**package model;**

public class BankBranch {}

کلاس BankBranch:  
این کلاس حاوی اطلاعات مربوط به بانک (BankBranch) است و شامل ویژگی هایی مانند نام بانک و آدرس است.

**package** **service;**

public class BankBranchService {

کلاس BankBranchService:  
این کلاس مسئول ارائه خدمات مربوط به بانک (BankBranch) است. در این کلاس متدهایی برای ایجاد بانک جدید، حذف بانک، اضافه کردن کارمند به بانک و دریافت لیست کارمندان براساس شناسه بانک ارائه شده است.

public String save() {}

متد save() به عنوان یک متد عملیاتی در کلاس BankBranchService قرار دارد و وظیفه‌ی ذخیره کردن یک شاخه بانکی را بر عهده دارد. خروجی این متد یک رشته است که نشان‌دهنده‌ی نتیجه‌ی عملیات ذخیره سازی است.

public void delete() {}

متد delete() نیز به عنوان یک متد عملیاتی در کلاس BankBranchService قرار دارد که وظیفه‌ی آن حذف کردن یک شاخه بانکی است. این متد هیچ خروجی را برنمی‌گرداند.

public BankBranch addEmpToBank() {}

متد addEmpToBank() در کلاس BankBranchService تعریف شده است و عملیات افزودن یک کارمند جدید به یک شاخه بانکی را انجام می‌دهد. خروجی این متد یک نمونه از کلاس BankBranch است که وضعیت آخرین ویرایش اطلاعات شاخه بانکی را نشان می‌دهد.

public List<Employee> getEmployeesByBankId(){}

متد getEmployeesByBankId() نیز در کلاس BankBranchService تعریف شده است و وظیفه‌ی آن بازگرداندن لیستی از کارمندانی است که به یک شاخه بانکی اختصاص یافته‌اند. خروجی این متد یک لیست از شیئ‌های کلاس Employee است.

}

**package controller;**

کلاس BankBranchController:  
این کلاس مسئول دریافت درخواست ها از طریق واسط کاربری (مانند API) و ارسال پاسخ ها به کاربر میباشد. در این کلاس متدهایی برای ایجاد بانک جدید، حذف بانک، اضافه کردن کارمند به بانک و دریافت لیست کارمندان براساس شناسه بانک پیاده سازی شده است.

public class BankBranchController {

public String CreateBranch() {}

متد CreateBranch() در کلاس BankBranchController مسئول ساختن یک شاخه بانکی است. این متد بر اساس اطلاعات ورودی که توسط کاربر ارسال می‌شود، یک شاخه بانکی جدید ایجاد می‌کند و یک رشته حاوی پیام موفقیت یا شکست را برمی‌گرداند.

public ResponseEntity<?> delete() {}

متد delete() در کلاس BankBranchController مسئول حذف یک شاخه بانکی است. این متد یک ورودی را دریافت می‌کند که شناسه‌ی شاخه بانکی مورد نظر برای حذف است و یک ResponseEntity را برمی‌گرداند که نتیجه‌ی عملیات حذف را نشان می‌دهد.

public BankBranch addEmpToBank() {}

متد addEmpToBank() در کلاس BankBranchController به شاخه بانکی یک کارمند جدید اضافه می‌کند و شاخه بانکی بعد از اضافه شدن کارمند را برمی‌گرداند.

List<Employee> getEmployyesById() {}

متد getEmployyesById() در کلاس BankBranchController لیستی از کارمندان موجود در یک شاخه بانکی را براساس شناسه‌ی شاخه بانکی مورد نظر بر می‌گرداند.

}

***package*** *customer*

**class** CustomerConvertor {}

کلاسی است که وظیفه تبدیل اطلاعات یک مشتری به شکل مورد نیاز دیگر قسمت‌ها را دارد.

**class** CustomerDTO {}

یک کلاس DTO (Data Transfer Object) است که برای انتقال داده‌های مشتری بین لایه‌های مختلف برنامه استفاده می‌شود.

**package repository;**

public interface CustomerRepository {}

یک رابط است که تعامل با داده‌های مشتری را بر عهده دارد، مانند ذخیره، حذف و دریافت اطلاعات مشتری.

**package model;**

public class Customer {}

 یک مدل است که اطلاعات یک مشتری را نگهداری می‌کند.

**package** **service;**

public class CustomerService {

کلاسی است که عملیات مربوط به مشتریان را مدیریت می‌کند، مانند ذخیره، به‌روزرسانی، حذف و دریافت لیستی از مشتریان.

public CustomerDTO save() {}

متد save()در کلاس CustomerService ، مشتری جدیدی را در سیستم ثبت می‌کند و یک CustomerDTO حاوی اطلاعات مشتری ثبت شده را برمی‌گرداند.

public CustomerDTO update() {}

متد update() در کلاس CustomerService ، اطلاعات یک مشتری موجود را به روزرسانی می‌کند و یک CustomerDTO حاوی اطلاعات به‌روزرسانی شده را برمی‌گرداند.

public void delete() {}

متد delete() در کلاس CustomerService ، مشتری مورد نظر را از سیستم حذف می‌کند.

public List<CustomerDTO> getAllCustomers() {}

متد getAllCustomers() در کلاس CustomerService ، همه‌ی مشتریان موجود در سیستم را برمی‌گرداند.

public void blockedCustomer() {}

متد blockedCustomer() در کلاس CustomerService ، وضعیت مشتری را به وضعیت بلوکه شده تغییر می‌دهد.

public void unBlockCustomer() {}

متد unBlockCustomer() در کلاس CustomerService، وضعیت مشتری را به وضعیت غیر بلوکه شده تغییر می‌دهد.

}

**package controller;**

public class CustomerController {

کلاسی است که واسط برای ارتباط با مشتریان در بخش کنترلر است.

public ResponseEntity<CustomerDTO> registerCustomer() {}

متد registerCustomer() در کلاس CustomerController، عملیات ثبت نام یک مشتری را انجام می‌دهد و یک ResponseEntity<CustomerDTO> حاوی اطلاعات مشتری ثبت شده را در قالب یک پیام HTTP برمی‌گرداند.

public ResponseEntity<CustomerDTO> updateCustomer() {}

متد updateCustomer() در کلاس CustomerController، اطلاعات یک مشتری موجود را به روزرسانی می‌کند و یک ResponseEntity<CustomerDTO> حاوی اطلاعات به‌روزرسانی شده را در قالب یک پیام HTTP برمی‌گرداند.

public ResponseEntity<?> deleteById() {}

متد deleteById() در کلاس CustomerController، مشتری مورد نظر را با استفاده از شناسه آن حذف می‌کند و یک ResponseEntity<?> حاوی پیام موفقیت یا شکست را در قالب یک پیام HTTP برمی‌گرداند.

public List<CustomerDTO> getAllCustomers() {}

متد getAllCustomers() در کلاس CustomerController، همه‌ی مشتریان موجود را در سیستم را برمی‌گرداند.

public void Blocked() {}

متد Blocked() در کلاس CustomerController ، وضعیت مشتری را به وضعیت بلوکه شده تغییر می‌دهد.

public void unBlocked() {}

متد unBlocked() در کلاس CustomerController، وضعیت مشتری را به وضعیت غیر بلوکه شده تغییر می‌دهد.

}

***package*** *employee*

**class** EmployeeConvertor {}

یک کلاس کمکی است که برای تبدیل بین اشیا Employee و DTO آن استفاده میشود.

**class** EmployeeDTO {}

یک کلاس DTO (Data Transfer Object) است که اطلاعات مربوط به یک کارمند را نگهداری میکند.

**package repository;**

public interface EmployeeRepository {}

یک رابط است که متدهای ارتباط با دیتابیس برای کارمندان را تعریف میکند.

**package model;**

public class Employee {}

یک کلاس مدل است که اطلاعات مربوط به یک کارمند را نگهداری می‌کند.

**package service;**

public class EmployeeService {

public EmployeeDTO save(){}

یک کلاس سرویس است که عملیات مربوط به مدیریت کارمندان را انجام می‌دهد. متد save() این کلاس یک کارمند را ذخیره میکند و یک EmployeeDTO حاوی اطلاعات کارمند ثبت شده را برمیگرداند.

}

**package controller;**

public class EmployeeController {

public ResponseEntity<EmployeeDTO> save (){}

یک کلاس کنترلر است که وظیفه‌ی ارتباط با درخواست‌های مربوط به کارمندان را دارد، متد save() این کلاس یک کارمند جدید را ذخیره میکند و یک ResponseEntity<EmployeeDTO> حاوی اطلاعات کارمند ثبت شده را برمیگرداند.

}

***package*** *transaction*

**class** TransactionConvertor {}

این کلاس مسئول تبدیل داده ها می باشد.

**class** TransactionDTO {}

این کلاس مدل داده DTO (Data Transfer Object) برای تراکنش ها است و اطلاعات مربوط به تراکنش را به صورت Serializable برای مراحل انتقال در سرویس ها و کنترل کننده ها فراهم می کند

**package repository;**

public interface TransactionRepository {}

این رابط پیش نیازی برای مدل داده های مربوط به تراکنش ها در بانک اطلاعاتی است.

**package model;**

public class Transaction {}

این کلاس مدلی است که ویژگی ها و عملیاتی که برای یک تراکنش در نظر گرفته می شود را تعریف می کند.

**package** **service;**

public class TransactionService {

این کلاس مسئول اجرای عملیات مربوط به تراکنش ها است. متدهای مختلفی از جمله ساخت تراکنش جدید، واریز و برداشت و بازیابی تراکنش ها را پیاده سازی می کند.

public String Card\_by\_card() {}

این متد برای انجام عملیات کارت به کارت استفاده می شود.

public Transaction creatTransaction() {}

این متد برای ایجاد یک تراکنش جدید استفاده می شود.

public String deposit() {}

این متد برای انجام عملیات واریز استفاده می شود.

public String withdraw() {}

این متد برای انجام عملیات برداشت استفاده می شود.

public List<TransactionDTO> getTransactions() {}

این متد برای بازیابی لیست تمامی تراکنش ها استفاده می شود.

public List<TransactionDTO> getSourceTransactionsByCardId() {}

این متد برای بازیابی لیست تمامی تراکنش هایی است که منبع آنها یک کارت با شناسه مشخص است.

public List<TransactionDTO> getDestinationTransactionsByCardId() {}

این متد برای بازیابی لیست تمامی تراکنش هایی است که مقصد آنها یک کارت با شناسه مشخص است.

public TransactionDTO getByUUID(){}

این متد برای بازیابی یک تراکنش بر اساس UUID آن استفاده می شود.

public List<TransactionDTO> getSourceAndDestinationByCardId (){}

این متد برای بازیابی لیست تمامی تراکنش هایی است که یک کارت با شناسه مشخص هم منبع آنها باشد و هم مقصد آنها.

}

**package controller;**

public class TransactionController {

این کلاس مسئول ارتباط با درخواست های دریافت شده از طریق درگاه های کاربری است. متدهای مختلفی از جمله برداشت، واریز و انتقال بین حساب ها را پیاده سازی می کند. همچنین قادر است لیستی از تراکنش ها را بازیابی و نمایش دهد و بازیابی تراکنش های مشخص ارائه شده توسط کاربر را فراهم می کند.

public ResponseEntity<String> withdraw() {}

این متد برای انجام عملیات برداشت مبلغ از حساب کاربر استفاده می شود. پاسخ به درخواست به صورت ResponseEntity<String> بازگردانده می شود که حاوی اطلاعاتی درباره نتیجه عملیات (موفقیت یا شکست) است.

public ResponseEntity<String> deposit() {}

این متد برای انجام عملیات واریز مبلغ به حساب کاربر استفاده می شود. پاسخ به درخواست به صورت ResponseEntity<String> بازگردانده می شود که حاوی اطلاعاتی درباره نتیجه عملیات (موفقیت یا شکست) است.

public ResponseEntity<String> TransferFromAccountToAccount() {}

این متد برای انجام عملیات انتقال وجه از یک حساب به حساب دیگر استفاده می شود. پاسخ به درخواست به صورت ResponseEntity<String> بازگردانده می شود که حاوی اطلاعاتی درباره نتیجه عملیات (موفقیت یا شکست) است.

public List<TransactionDTO> getAllTransaction() {}

این متد برای بازیابی لیست تمامی تراکنش ها استفاده می شود.

public List<TransactionDTO> getSourceByCardId() {}

این متد برای بازیابی لیست تمامی تراکنش هایی است که منبع آنها یک کارت با شناسه مشخص است.

public List<TransactionDTO> getDestinationByCardId() {}

این متد، لیستی از تراکنش های مقصدی را بر اساس شناسه کارت برگشت می دهد. اطلاعات تراکنش ها به صورت TransactionDTO برگشت داده می شود.

public ResponseEntity<TransactionDTO> getByUUId() {}

این متد، یک تراکنش را بر اساس شناسه یونیک برمی گرداند. اطلاعات تراکنش به صورت TransactionDTO برگشت داده می شود. این متد می تواند به صورت ResponseEntity برگشت داده شود تا اطلاعات بیشتری در قالب پاسخ HTTP وجود داشته باشد.

public ResponseEntity<List<TransactionDTO>> getAlltransactionByCardId() {}

این متد، لیستی از تمامی تراکنش ها را بر اساس شناسه کارت برگشت می دهد. اطلاعات تراکنش ها به صورت TransactionDTO برگشت داده می شود. این متد نیز می تواند به صورت ResponseEntity برگشت داده شود تا اطلاعات بیشتری در قالب پاسخ HTTP وجود داشته باشد.

}