Banking App

Making a baking app with the feature of ->

1. Create Account Holder Name & balance
2. Show account holders by id
3. Show all saved account holders
4. Deposit
5. Withdrawal
6. Delete the account

**Work Flow:**

Controller

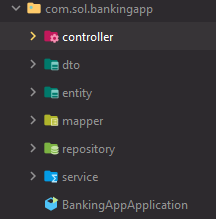
DB

API

Service

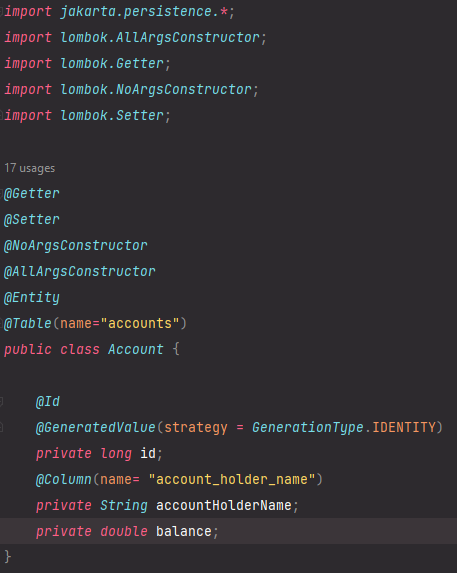
**Program Methodology:**

1. Make a jar file in spring Initializer. Open in IDE. In com.sol.banking package make these following pakages.

using MVC architecture, in model layer we use service package. Entity, dto, repository for story bean var. and controller to connect with api which is the view here.

1. Make a Account Class in Entity. Using following annotations

* *@Getter, @Setter* for auto Generate getter and Setter
* *@NoArgsConstructor, @AllArgsConstructor* to create constructor dependency injection
* *@Entity* for define Entity Class & *@Table(name="accounts")* to create accounts table in DB
* @*Id* – for unique Value key
* *@GeneratedValue*(strategy = *GenerationType*.IDENTITY) -Auto generated Identity key
* *@Column*(name= "account\_holder\_name") – make thw Coloumn name account holder name

save the entity data as an encapsulation , using private access specifier. Then id as a long , accountholdername as string and balance as adouble.

1. now to connect with database, Mysql . in application properties

*spring.datasource.url*=jdbc:mysql://localhost:3306/banking\_app  
*spring.datasource.username*=root  
*spring.datasource.password*=Helios2110@  
*spring.jpa.show-sql*=true  
  
*spring.jpa.hibernate.ddl-auto*=update

and create a database in mysql with the name banking\_app

1. make a AccountRepository interface in repository fetch or push the data from Database. By Using the JPA repository it connects with entity beans and uses in service layer.

*public interface AccountRepository extends JpaRepository*<*Account*,*Long*> { }

1. in dto/playload create a AccountDto class. The dto beans makes the connection between Controller layer and service layer.

*@Data  
@AllArgsConstructor  
public class AccountDto* {  
 *private long* id;  
 *private String* accountHolderName;  
 *private double* balance;  
}

1. Now make a mapper class to reduce the line of codes in service layer.

*public static Account* mapToAccount(*AccountDto accountDto*){  
 *Account* account =*new* Account(  
 *accountDto*.getId(),  
 *accountDto*.getAccountHolderName(),  
 *accountDto*.getBalance());

*return* account;

map the beans for Dto to entity

*public static AccountDto* mapToAccountDto(*Account account*){  
 *AccountDto* accountDto = *new* AccountDto(  
 *account*.getId(),  
 *account*.getAccountHolderName(),  
 *account*.getBalance());  
 *return* accountDto;}

map the beans Entity to Dto

1. Making the service layer with service interface, and create all service methods

*public interface AccountService* {  
 *AccountDto* createAccount(*AccountDto accountDto*);  
 *AccountDto* getAccountById(*Long id*);  
 *AccountDto* deposit(*Long id* , *double amount*);  
 *AccountDto* withdraw(*Long id*, *double amount*);  
 *List*<*AccountDto*> getAllAccounts();  
 *void* deleteAccount (*Long id*);}

then make a serviceImpl class to implement the service interface

*@Service  
public class AccountServiceImpl implements AccountService* {  
  
 *private AccountRepository* accountRepository;  
  
 *public* AccountServiceImpl(*AccountRepository accountRepository*) {  
 *this*.accountRepository = *accountRepository*;}

Create a constructor to dependency injections. Then override the service methods

*@Override  
public AccountDto* createAccount(*AccountDto accountDto*){  
 *Account* account = *AccountMapper*.*mapToAccount*(*accountDto*);  
 *Account* savedAccount = accountRepository.save(account);  
 *return AccountMapper*.*mapToAccountDto*(savedAccount);}

*@Override  
public AccountDto* getAccountById(*Long id*) {  
*Account* account = accountRepository.findById(*id*).orElseThrow(

() -> *new* RuntimeException("Account does not exists"));  
 *return AccountMapper*.*mapToAccountDto*(account);}

*@Override  
public AccountDto* deposit(*Long id*, *double amount*) {  
*Account* account = accountRepository.findById(*id*).orElseThrow(

() -> *new* RuntimeException("Account does not exists"));  
*double* total = account.getBalance() + *amount*;  
account.setBalance(total);

*Account* savedAccount = accountRepository.save(account);  
*return AccountMapper*.*mapToAccountDto*(savedAccount);}

*@Override  
public AccountDto* withdraw(*Long id*, *double amount*) {  
  
 *Account* account = accountRepository  
 .findById(*id*)  
 .orElseThrow(() -> *new* RuntimeException("Account does not exists"));  
  
 *if*(account.getBalance() < *amount*){  
 *throw new* RuntimeException ("Insufficient amount");  
 }  
 *double* total = account.getBalance() - *amount*;  
 account.setBalance(total);  
 *Account* savedAccount = accountRepository.save(account);  
 *return AccountMapper*.*mapToAccountDto*(savedAccount);

*@Override  
public List*<*AccountDto*> getAllAccounts() {  
 *List*<*Account*> accounts =accountRepository.findAll();  
 *return* accounts.stream().map((*account*) -> *AccountMapper*.*mapToAccountDto*(*account*)).collect(*Collectors*.*toList*());  
}

*@Override  
public void* deleteAccount(*Long id*) {  
  
 *Account* account = accountRepository  
 .findById(*id*)  
 .orElseThrow(() -> *new* RuntimeException("Account does not exists"));  
  
 accountRepository.deleteById(*id*);  
}

1. In controller layer we conncet the data with the API using

*@RestController  
@RequestMapping*("/api/accounts")  
*public class AccountCrontroller* {  
 *private AccountService* accountService;  
  
 *public* AccountCrontroller(*AccountService accountService*) {  
 *this*.accountService = *accountService*;  
 }

For create using

*@PostMapping  
public ResponseEntity*<*AccountDto*> addAccount(*@RequestBody AccountDto accountDto*){  
 *return new* ResponseEntity<>(accountService.createAccount(*accountDto*), *HttpStatus*.CREATED);  
}

*@GetMapping*("/{id}")  
*public ResponseEntity*<*AccountDto*> getAccountById(*@PathVariable Long id*){  
 *AccountDto* accountDto = accountService.getAccountById(*id*);  
 *return ResponseEntity*.*ok*(accountDto);  
}

*@PutMapping*("/{id}/deposit")  
*public ResponseEntity*<*AccountDto*> deposit(*@PathVariable Long id*, *@RequestBody Map*<*String*,*Double*> *request*){  
 *Double* amount = *request*.get("amount");  
 *AccountDto* accountDto= accountService.deposit(*id*, amount);  
 *return ResponseEntity*.*ok*(accountDto);  
}

*@PutMapping*("/{id}/withdraw")  
*public ResponseEntity*<*AccountDto*> withdraw(*@PathVariable Long id*,*@RequestBody Map*<*String*,*Double*> *request*){  
 *Double* amount = *request*.get("amount");  
 *AccountDto* accountDto = accountService.deposit(*id*,amount);  
 *return ResponseEntity*.*ok*(accountDto);  
};

*@GetMapping  
public ResponseEntity*<*List*<*AccountDto*>> getAllAccount(){  
 *List*<*AccountDto*> accounts = accountService. getAllAccounts();  
 *return ResponseEntity*.*ok*(accounts);  
}

*@DeleteMapping*("/{id}")  
*public ResponseEntity*<*String*> deleteById(*@PathVariable Long id*){  
 accountService.deleteAccount(*id*);  
 *return ResponseEntity*.*ok*("Account is deleted successfully!!");  
}

1. Open Postman Use the all services using the url.

Ok