// Java Spring Boot Course

Spring Boot 專題實作練習

專題練習：ATM

全端 Web API 整合開發

整合 Controller、Service、Repository 與 Security 的完整應用。

技術堆疊：

- `spring-boot-starter-web`
- `spring-boot-devtools (Hot Deploy)`
- `spring-boot-starter-security`
- `BCryptPasswordEncoder`

需排除 SecurityAutoConfiguration 以自定義安全邏輯。

Atm：需求規格

Entity 定義

- account (String, 20, PK, Not Null)
- password(String, 60, Not Null)
- balance (int)

使用BCryptPasswordEncoder

- 增加 security 的依賴，透過 web 呼叫 API 時，會強制要求登入帳號密碼
- 定義「允許所有」的 FilterChain

build.gradle 設定

```
dependencies {
    // 1. Web: 讓我們可以寫 Controller, 提供 REST API
    implementation 'org.springframework.boot:spring-boot-starter-web'

    // 2. JPA & MySQL: 資料庫存取
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    runtimeOnly 'com.mysql:mysql-connector-j'

    // 3. Security: 這裡我們主要為了使用BCryptPasswordEncoder
    implementation 'org.springframework.boot:spring-boot-starter-security'

    // 4. DevTools: 修改程式碼後自動重啟 (Hot Deployment)
    developmentOnly 'org.springframework.boot:spring-boot-devtools'

    // 5. Lombok & Test (建議加入)
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

排除 Spring Security 預設登入頁面

```java
@Configuration
@EnableWebSecurity // 開啟 Web 安全功能
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .anyRequest().permitAll() // 這裡定義了不需要登入
            );
        return http.build(); // Spring 會自動拿到這個回傳的物件並使用它
    }

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Entity

```java
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "atm")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Atm {
    @Id
    private String account;
    private String password; // 這裡存的是加密後的亂碼
    private Integer balance;
}
```

Repository

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface AtmRepository extends JpaRepository<Atm, String> {
    // 基本 CRUD 由 JPA 提供
}
```

DTO - 狀態碼 Enum

```
public enum RspCode {
    SUCCESS(200, "操作成功"),
    PARAM_ERROR(400, "參數錯誤"),
    ACCOUNT_NOT_FOUND(404, "帳號不存在"),
    PASSWORD_ERROR(401, "密碼錯誤"),
    INSUFFICIENT_BALANCE(400, "餘額不足"),
    DUPLICATE_ACCOUNT(409, "帳號已存在");

    private int code;
    private String message;

    RspCode(int code, String message) {
        this.code = code;
        this.message = message;
    }

    public int getCode() { return code; }
    public String getMessage() { return message; }
}
```

DTO - 統一回應物件

```java
import lombok.Data;

@Data
public class AtmResponse {
    private int code;
    private String message;
    private Atm data; // 回傳的帳戶資訊

    public AtmResponse(RspCode rspCode, Atm data) {
        this.code = rspCode.getCode();
        this.message = rspCode.getMessage();
        this.data = data;
        // 題目要求:Response 中可以有 password 欄位，但不能有值 (資安考量)
        if (this.data != null) {
            this.data.setPassword(null);
        }
    }
}
```

DTO - 請求物件

```java
import lombok.Data;

@Data
public class AtmRequest {
    private String account;
    private String password;
    private String newPassword; // 修改密碼用
    private Integer amount;     // 存款/提款金額
}
```

Service(1) - 注入

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.Optional;

@Service
public class AtmService {
@Autowired
    private AtmRepository atmRepository;

    @Autowired
    private PasswordEncoder passwordEncoder; // 注入 BCrypt 編碼器

}
```

Service(2) - 新增帳戶

```
// === 1. 新增帳戶 (addInfo) ===
    public AtmResponse addInfo(AtmRequest req) {
        String account = req.getAccount();
        String password = req.getPassword();

        // 防呆檢查
        if (!isValidAccount(account) || !isValidPassword(password)) {
            return new AtmResponse(RspCode.PARAM_ERROR, null);
        }
        if (atmRepository.existsById(account)) {
            return new AtmResponse(RspCode.DUPLICATE_ACCOUNT, null);
        }

        // 密碼加密並儲存
        String encodedPwd = passwordEncoder.encode(password);
        Atm newAtm = new Atm(account, encodedPwd, 0); // 初始餘額 0
        atmRepository.save(newAtm);

        return new AtmResponse(RspCode.SUCCESS, newAtm);
    }
```

Service(3) - 查詢餘額

```
public AtmResponse getBalanceByAccount(AtmRequest req) {
        Atm atm = checkAccountAndPassword(req.getAccount(), req.getPassword());
        if (atm == null) {
            return new AtmResponse(RspCode.PASSWORD_ERROR, null); // 或 帳號不存在
        }
        return new AtmResponse(RspCode.SUCCESS, atm);
    }
```

Service(4) - 修改密碼

```
public AtmResponse updatePasswordByAccount(AtmRequest req) {
        Atm atm = checkAccountAndPassword(req.getAccount(), req.getPassword());
        if (atm == null) {
            return new AtmResponse(RspCode.PASSWORD_ERROR, null);
        }

        // 檢查新密碼格式
        if (!isValidPassword(req.getNewPassword())) {
            return new AtmResponse(RspCode.PARAM_ERROR, null);
        }

        // 加密新密碼並儲存
        atm.setPassword(passwordEncoder.encode(req.getNewPassword()));
        atmRepository.save(atm);
        return new AtmResponse(RspCode.SUCCESS, atm);
    }
```

Service(5) - 存款

```
public AtmResponse deposit(AtmRequest req) {
        Atm atm = checkAccountAndPassword(req.getAccount(), req.getPassword());
        if (atm == null) {
            return new AtmResponse(RspCode.PASSWORD_ERROR, null);
        }

        if (req.getAmount() == null || req.getAmount() <= 0) {
            return new AtmResponse(RspCode.PARAM_ERROR, null);
        }

        atm.setBalance(atm.getBalance() + req.getAmount());
        atmRepository.save(atm);
        return new AtmResponse(RspCode.SUCCESS, atm);
    }
```

Service(6) - 提款

```
public AtmResponse withdraw(AtmRequest req) {
        Atm atm = checkAccountAndPassword(req.getAccount(), req.getPassword());
        if (atm == null) {
            return new AtmResponse(RspCode.PASSWORD_ERROR, null);
        }

        if (req.getAmount() == null || req.getAmount() <= 0) {
            return new AtmResponse(RspCode.PARAM_ERROR, null);
        }

        if (atm.getBalance() < req.getAmount()) {
            return new AtmResponse(RspCode.INSUFFICIENT_BALANCE, atm);
        }

        atm.setBalance(atm.getBalance() - req.getAmount());
        atmRepository.save(atm);
        return new AtmResponse(RspCode.SUCCESS, atm);
    }
```

Service(7) - 驗證帳號密碼

```
private Atm checkAccountAndPassword(String account, String rawPassword) {
    Optional<Atm> optional = atmRepository.findById(account);
    if (optional.isPresent()) {
        Atm atm = optional.get();
        // 使用 matches 方法比對:(明文, 加密文)
        if (passwordEncoder.matches(rawPassword, atm.getPassword())) {
            return atm;
        }
    }
    return null;
}
// Regex 驗證帳號: 3~8碼, 英文數字底線
private boolean isValidAccount(String acc) {
    return acc != null && acc.matches("^[a-zA-Z0-9_]{3,8}$");
}
// Regex 驗證密碼: 8~16碼, 英文數字底線
private boolean isValidPassword(String pwd) {
    return pwd != null && pwd.matches("^[a-zA-Z0-9_]{8,16}$");
}
```

Controller

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
@RestController
@RequestMapping("/api/atm")
public class AtmServiceController {
    @Autowired
    private AtmService atmService;
     @PostMapping("/add")  // 1. 新增帳戶
    public AtmResponse add(@RequestBody AtmRequest request) {
        return atmService.addInfo(request);
    }
    @PostMapping("/balance")// 2. 查詢餘額 (需驗證密碼，故用POST)
    public AtmResponse getBalance(@RequestBody AtmRequest request) {
        return atmService.getBalanceByAccount(request);
    }
    @PostMapping("/update_pwd")// 3. 修改密碼
    public AtmResponse updatePwd(@RequestBody AtmRequest request) {
        return atmService.updatePasswordByAccount(request);
    }
    @PostMapping("/deposit")// 4. 存款
    public AtmResponse deposit(@RequestBody AtmRequest request) {
        return atmService.deposit(request);
    }
    @PostMapping("/withdraw")// 5. 提款
    public AtmResponse withdraw(@RequestBody AtmRequest request) {
        return atmService.withdraw(request);
    }
}
```

使用 Postman 測試

1. 新增帳戶 (Add)

- **Method:** POST
- **URL:** http://localhost:8080/api/atm/add
- **Body (JSON):**

```JSON
{
  "account": "user01",
  "password": "Password123"
}
```

- **預期結果:** code: 200, data.password: null

3. 提款 (Withdraw)

- **URL:** http://localhost:8080/api/atm/withdraw
- **Body:** (同上，將 amount 改為 500)

5. 測試防呆 (故意打錯)

- 試試看帳號只有 2 碼。
- 試試看密碼輸入錯誤。
- 試試看提款超過餘額。

2. 存款 (Deposit)

- **URL:** http://localhost:8080/api/atm/deposit
- **Body:**

```JSON
{
  "account": "user01",
  "password": "Password123",
  "amount": 1000
}
```

4. 修改密碼

- **URL:** http://localhost:8080/api/atm/update_pwd
- **Body:**

```JSON
{
  "account": "user01",
  "password": "Password123",
  "newPassword": "NewPassword888"
}
```