스프링 있이 구현하는 파트

```
import javax.sql.DataSource;

@Configuration
@ComponentScan(basePackageClasses = Base.class,
        excludeFilters = @ComponentScan.Filter(Controller.class))
public class RootConfig {
    @Bean
    public DataSource dataSource() {
        BasicDataSource dataSource = new BasicDataSource();
        dataSource.setDriverClassName("org.h2.Driver");

        // 데이터베이스 접속 정보 설정
        dataSource.setUrl("jdbc:h2:~/spring-jpa;DATABASE_TO_UPPER=false;"
                + "INIT=RUNSCRIPT FROM 'classpath:/script/schema.sql'");
        dataSource.setUsername("sa");
        dataSource.setPassword("");

        // 초기 커넥션 풀 크기 설정
        dataSource.setInitialSize(10);

        // 최대 커넥션 풀 크기 설정
        dataSource.setMaxTotal(10);

        // 최소 유휴 커넥션 개수 설정
        dataSource.setMinIdle(10);

        // 최대 유휴 커넥션 개수 설정
        dataSource.setMaxIdle(10);

        // 커넥션 풀이 바쁠 때 대기 시간 설정 (밀리초)
        dataSource.setMaxWaitMillis(1000);

        // 커넥션 풀에서 커넥션을 가져올 때 살아있는지 확인
        dataSource.setTestOnBorrow(true);

        // 사용이 끝난 커넥션을 다시 풀에 반환할 때 해당 커넥션이 사용 가능한지 확인
        dataSource.setTestOnReturn(true);

        // 주기적으로 유휴 상태인 커넥션들을 검사하여 살아있는지 확인 (약간의 성능 저하가 있을
수 있음)
        dataSource.setTestWhileIdle(true);

        return dataSource;
    }
    @Bean
    //플랫폼 트랜잭션 매니저
    //트랜잭션 추상화를 위한 중심 API
    // jdbc를 쓰지만 다른걸 쓰면  DataSource가아니라 다른게 들어갈수도있음
```

```
    public PlatformTransactionManager transactionManager() {
        return new DataSourceTransactionManager(dataSource());
    }
}
```

```
- 두개의 유저정보를 미리 넣어둠

CREATE TABLE IF NOT EXISTS `Users` (
    `user_id`   VARCHAR(50) NOT NULL,
    `password`  VARCHAR(50) NOT NULL,

    PRIMARY KEY(`user_id`)
);

MERGE INTO `Users` KEY ( `user_id` ) VALUES ( 'admin', '12345' );
MERGE INTO `Users` KEY ( `user_id` ) VALUES ( 'dongmyo', '67890' );
```

```
@RestController
@RequestMapping("/users/{userId}")
public class UserRestController {
    private final UserRepository userRepository;

    public UserRestController(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @ModelAttribute(value = "user", binding = false)
    public User getUser(@PathVariable("userId") String userId) {
        User user = userRepository.getUser(userId);
        if (Objects.isNull(user)) {
            throw new UserNotFoundException();
        }

        return user;
    }

    @GetMapping
    public User getUser(@ModelAttribute("user") User user) {
        return user;
    }

    @PutMapping
    public User modifyUser(@ModelAttribute("user") User user,
                           @Valid @RequestBody UserModifyRequest request,
                           BindingResult bindingResult) {
        if (bindingResult.hasErrors()) {
            throw new ValidationFailedException(bindingResult);
```

```
        }

        if (!userRepository.modifyUser(user.getId(),
request.getPassword())) {
            throw new UserModifyFailedException();
        }

        return userRepository.getUser(user.getId());
    }

}
```

```
package com.nhnacademy.springjpa.repository;

import com.nhnacademy.springjpa.domain.User;
import com.nhnacademy.springjpa.domain.UserRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import javax.sql.DataSource;
import java.util.Objects;

@Repository("userRepository")
public class UserRepositoryImpl implements UserRepository {
    private final JdbcTemplate jdbcTemplate;

    public UserRepositoryImpl(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    @Override
    public boolean exists(String id) {
        Integer count = jdbcTemplate.queryForObject("SELECT count(*) FROM
Users WHERE user_id = ?1", Integer.class, id);
        return count != null && count == 1;
    }

    @Override
    public boolean matches(String id, String password) {
        User user = jdbcTemplate.queryForObject("SELECT user_id, password
FROM Users WHERE user_id = ?1 AND password = ?2",
                User.class, id, password);

        return Objects.nonNull(user) && user.getId().equals(id);
    }

    @Override
    public User getUser(String id) {
        return jdbcTemplate.queryForObject("SELECT user_id, password FROM
Users where user_id = ?1", new UserRowMapper(), id);
```

```
    }

    @Override
    public boolean addUser(String id, String password) {
        int result = jdbcTemplate.update("INSERT INTO Users (`user_id`,
`password`) VALUES (?, ?)",
                id,
                password);

        return result == 1;
    }

    @Override
    public boolean modifyUser(String id, String password) {
        int result = jdbcTemplate.update("UPDATE Users set password = ?1
WHERE user_id = ?2",
                password,
                id);

        return result == 1;
    }

}
```

```
GET /users/admin
Host: localhost:8080
Content-Type: application/json

###


POST /users HTTP/1.1
Host: localhost:8080
Content-Type: application/json

{
  "id": "nhn",
  "password": "academy"
}

###

GET /users/nhn
Host: localhost:8080
Content-Type: application/json


###

PUT /users/nhn
```

```
Host: localhost:8080
Content-Type: application/json

{
  "password": "hahaha"
}
```

```
// Component 스캔과 비슷한 기능을 한다.
// SPRING에서 jpa 쓰려면 이게 필요하다.
@EnableJpaRepositories(basePackageClasses = RepositoryBase.class)
@Configuration
public class JpaConfig {
    //
    @Bean
    public LocalContainerEntityManagerFactoryBean
entityManagerFactory(DataSource dataSource) {
        LocalContainerEntityManagerFactoryBean emf = new
LocalContainerEntityManagerFactoryBean();
        emf.setDataSource(dataSource);
        emf.setPackagesToScan("com.nhnacademy.springjpa.entity"); // jpa가
관리하는 entity를 찾기 위한 경로 (현재 교육자료엔 entity 패키지에 있다. 하나의 앤티티매니저가
다루는 앤티티의 종류는 한정이 없고, 트랜잭션당 하나의 엔티티매니저가 생성된다.)
        emf.setJpaVendorAdapter(jpaVendorAdapters());
        emf.setJpaProperties(jpaProperties());

        return emf;
    }

    private JpaVendorAdapter jpaVendorAdapters() {
        HibernateJpaVendorAdapter hibernateJpaVendorAdapter = new
HibernateJpaVendorAdapter();
        hibernateJpaVendorAdapter.setDatabase(Database.H2);//방언을 지정해주는
역할. mysql, h2, oracle 등등

        return hibernateJpaVendorAdapter;
    }

  private Properties jpaProperties() {
    Properties jpaProperties = new Properties();
    jpaProperties.setProperty("hibernate.show_sql", "true"); // 생성되는 SQL
을 로그에 출력
    jpaProperties.setProperty("hibernate.format_sql", "true"); // SQL을 잘
보기 좋게 포맷팅
    jpaProperties.setProperty("hibernate.use_sql_comments", "true"); //
SQL 쿼리에 대한 주석 추가
    jpaProperties.setProperty("hibernate.globally_quoted_identifiers",
"true"); // 모든 데이터베이스 객체에 대해 쿼리 작성 시 인용 부호(따옴표) 사용
    jpaProperties.setProperty("hibernate.temp.use_jdbc_metadata_defaults",
"false"); // JDBC 메타데이터 기본값 사용하지 않음

    return jpaProperties;
```

```
    }// 하이버네이트 속성값 지정해주는 역할

    //jpa에서 사용하기 위한 트랜잭션 매니저
    // datasource는 jdbc꺼였고 얘는 jp
    // root컨피그에선 platformtransactionManager가 필요없다.
    //setEntityManagerFactory(entityManagerFactory); 이게 필요하더라
    @Bean
    public PlatformTransactionManager
transactionManager(EntityManagerFactory entityManagerFactory) {
        JpaTransactionManager transactionManager = new
JpaTransactionManager();
        transactionManager.setEntityManagerFactory(entityManagerFactory);

        return transactionManager;
    }

}
```

```
package com.nhnacademy.springjpa.entity;


import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Transient;
import lombok.Getter;
import lombok.Setter;

// TODO #1: `Items` 테이블과 맵핑될 `Item` Entity 클래스를 작성하세요.
/*
 *
 * create table if not exists `Items` (
 *   `item_id` bigint not null auto_increment,
 *   `item_name` varchar(40) not null,
 *   `price` bigint not null,
 *
 *   primary key(`item_id`)
 * );
 *
 */
@Entity
@Table(name="Items")
@Getter
@Setter
public class Item {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    // 객체가 생성될때 오토 인크리먼트로 아이디를 설정해준다?
    @Column(name="item_id")
    private Long itemId;
    @Column(name="item_name")
    private String itemName;
    private Long price;
    @Transient
    private String test;

}
```

```
package com.nhnacademy.springjpa.entity;

import java.sql.Timestamp;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import lombok.Getter;
import lombok.Setter;

// TODO #1: `Orders` 테이블과 맵핑될 `Order` Entity 클래스를 작성하세요.
/*
 * create table if not exists `Orders` (
 *   `order_id` bigint not null auto_increment,
 *   `order_date` timestamp not null,
 *
 *   primary key(`order_id`)
 * );
 *
 */
@Entity
@Table(name="Orders")
@Getter
@Setter
public class Order {
  @Id
  @Column(name = "order_id")
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long orderId;
  @Column(name = "order_date")
  private LocalDate orderDate;
}
```

```
package com.nhnacademy.springjpa.entity;

import static org.assertj.core.api.Assertions.assertThat;

import com.nhnacademy.springjpa.config.RootConfig;
import com.nhnacademy.springjpa.config.WebConfig;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.ContextHierarchy;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import org.springframework.test.context.web.WebAppConfiguration;
import org.springframework.test.util.ReflectionTestUtils;
import org.springframework.transaction.annotation.Transactional;

// TODO #2: 아래 `@Disabled` 어노테이션을 삭제하고 테스트를 통과시키세요.
@ExtendWith(SpringExtension.class)
@WebAppConfiguration
@Transactional
@ContextHierarchy({
    @ContextConfiguration(classes = RootConfig.class),
    @ContextConfiguration(classes = WebConfig.class)
})
public class OrderEntityTest {
    @PersistenceContext
    private EntityManager entityManager;

    @Test
    public void testOrderEntity() {
        Order order1 = entityManager.find(Order.class, 1001L);

        assertThat(ReflectionTestUtils.invokeGetterMethod(order1,
"orderId")).isEqualTo(1001L);
        assertThat(ReflectionTestUtils.invokeGetterMethod(order1,
"orderDate")).isNotNull();
    }

}
```

스프링없이 구현하는 파트

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
```

```
http://xmlns.jcp.org/xml/ns/persistence/persistence_1.0.xsd"
  version="2.2">
  <persistence-unit name="default">
<class>com.nhnacademy.jpa.entity.User</class>
    <properties>
      <property name="javax.persistence.jdbc.driver"
value="org.h2.Driver"/>
      <property name="javax.persistence.jdbc.url" value="jdbc:h2:~/spring-
jpa;DATABASE_TO_UPPER=false;"/>
      <property name="javax.persistence.jdbc.user" value="sa"/>
      <property name="javax.persistence.jdbc.password" value=""/>

      <property name="hibernate.hbm2ddl.auto" value="create"/>
<!--       테스트 용도로만 써야한다.-->
<!--       hibernate 매핑을 자동으로 할거냐. 드랍 업데이트 크리에이트 밸리데이트-->
<!--       크리에이트는 엔티티를 새로만든다-->
<!--       크리에이트 드랍은 엔티티를 새로만들고 업데이트될때 드랍-->
<!--       지금 엔티티 매핑되잇는거랑 틀린거랑 수정?-->
<!--       엔티티 매핑에 있는 내용이랑 실제 디비 스키마랑 안맞으면 에러내고 끝낸다-->

    </properties>
  </persistence-unit>

</persistence>
```