```
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");
        emf.setJpaVendorAdapter(jpaVendorAdapters());
        emf.setJpaProperties(jpaProperties());

        return emf;
    }

    private JpaVendorAdapter jpaVendorAdapters() {
        HibernateJpaVendorAdapter hibernateJpaVendorAdapter = new
HibernateJpaVendorAdapter();
        hibernateJpaVendorAdapter.setDatabase(Database.H2);

        return hibernateJpaVendorAdapter;
    }

    private Properties jpaProperties() {
        Properties jpaProperties = new Properties();
        jpaProperties.setProperty("hibernate.show_sql", "true");
        jpaProperties.setProperty("hibernate.format_sql", "true");
        jpaProperties.setProperty("hibernate.use_sql_comments", "true");
        jpaProperties.setProperty("hibernate.globally_quoted_identifiers",
"true");

jpaProperties.setProperty("hibernate.temp.use_jdbc_metadata_defaults",
"false");

        return jpaProperties;
    }

    @Bean
    public PlatformTransactionManager
transactionManager(EntityManagerFactory entityManagerFactory) {
        JpaTransactionManager transactionManager = new
JpaTransactionManager();
        transactionManager.setEntityManagerFactory(entityManagerFactory);
```

```
            return transactionManager;
        }

    }
```