#30 and #31 JPA Native Query, Criteria API and Specification API

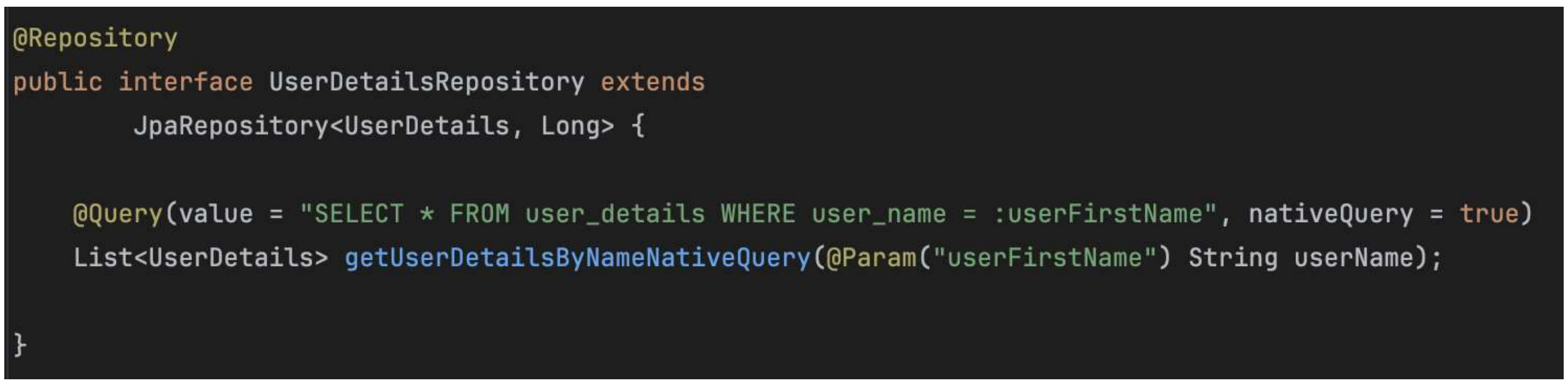
Saturday, 8 February 2025 12:21 PM

Native Query:

- Plain SQL queries.
- Directly interact with Database, thus if in future DB changes, code changes also requi
- No caching, lazy loading or entity life cycle management happens.

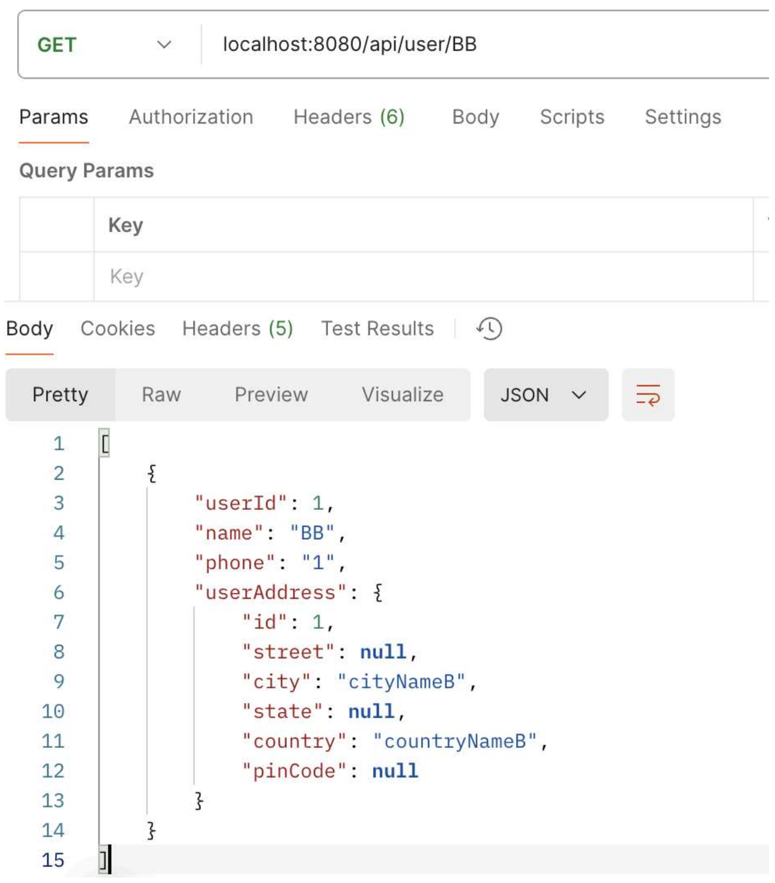
When to use over JPQL:

- More complex queries, including database specific features like JSONB, LATERAL JOIN
- Need to fetch non-entity results or joins table without any entity relationship.
- Query efficiency like Bulk operations.



```java
@Repository
public interface UserDetailsRepository extends
        JpaRepository<UserDetails, Long> {

    @Query(value = "SELECT * FROM user_details WHERE user_name = :userFirstName", nativeQuery = true)
    List<UserDetails> getUserDetailsByNameNativeQuery(@Param("userFirstName") String userName);

}
```

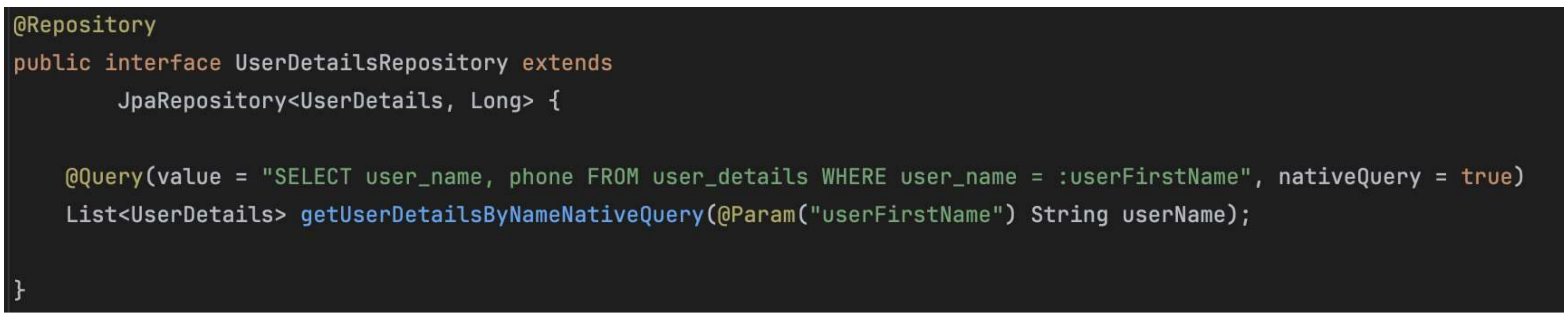
When all the fields (*) of the table are returned by Native Query, JPA internally does the mapping between DB column name and Entity fields.



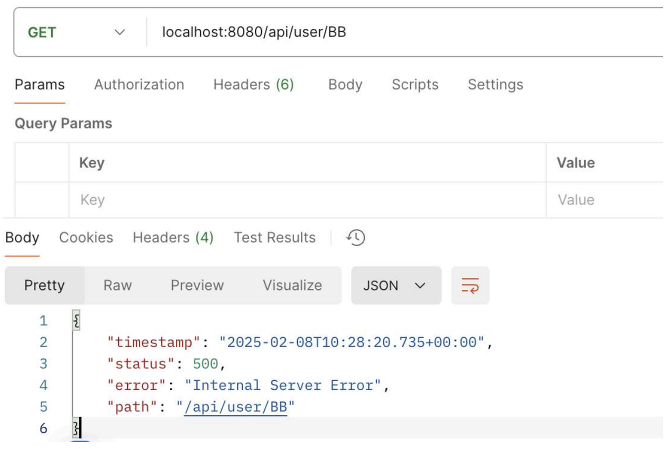
localhost:8080/api/user/BB

```json
[
    {
        "userId": 1,
        "name": "BB",
        "phone": "1",
        "userAddress": {
            "id": 1,
            "street": null,
            "city": "cityNameB",
            "state": null,
            "country": "countryNameB",
            "pinCode": null
        }
    }
]
```

But, when Native Query returned partial fields, then JPA don't map it to Entity by default.

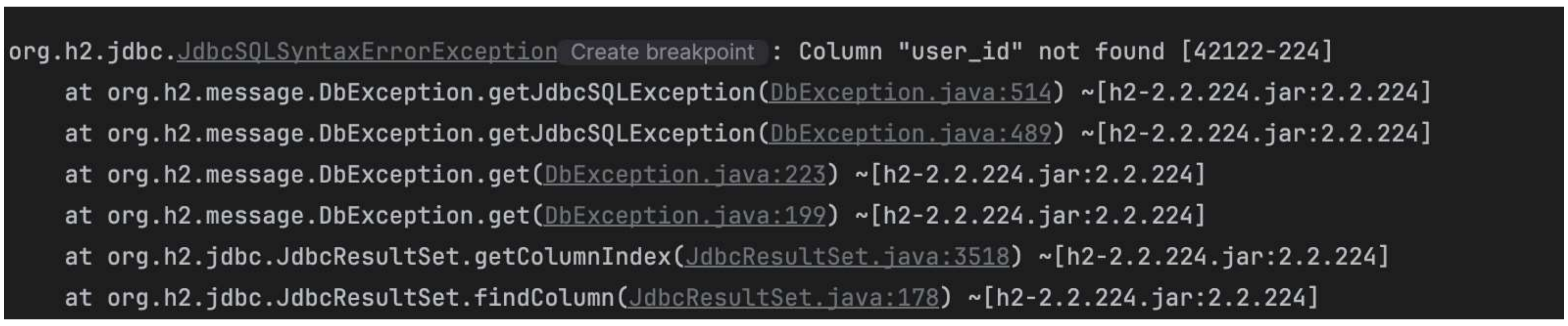


```
@Repository
public interface UserDetailsRepository extends
        JpaRepository<UserDetails, Long> {

    @Query(value = "SELECT user_name, phone FROM user_details WHERE user_name = :userFirstName", nativeQuery = true)
    List<UserDetails> getUserDetailsByNameNativeQuery(@Param("userFirstName") String userName);

}
```



GET localhost:8080/api/user/BB

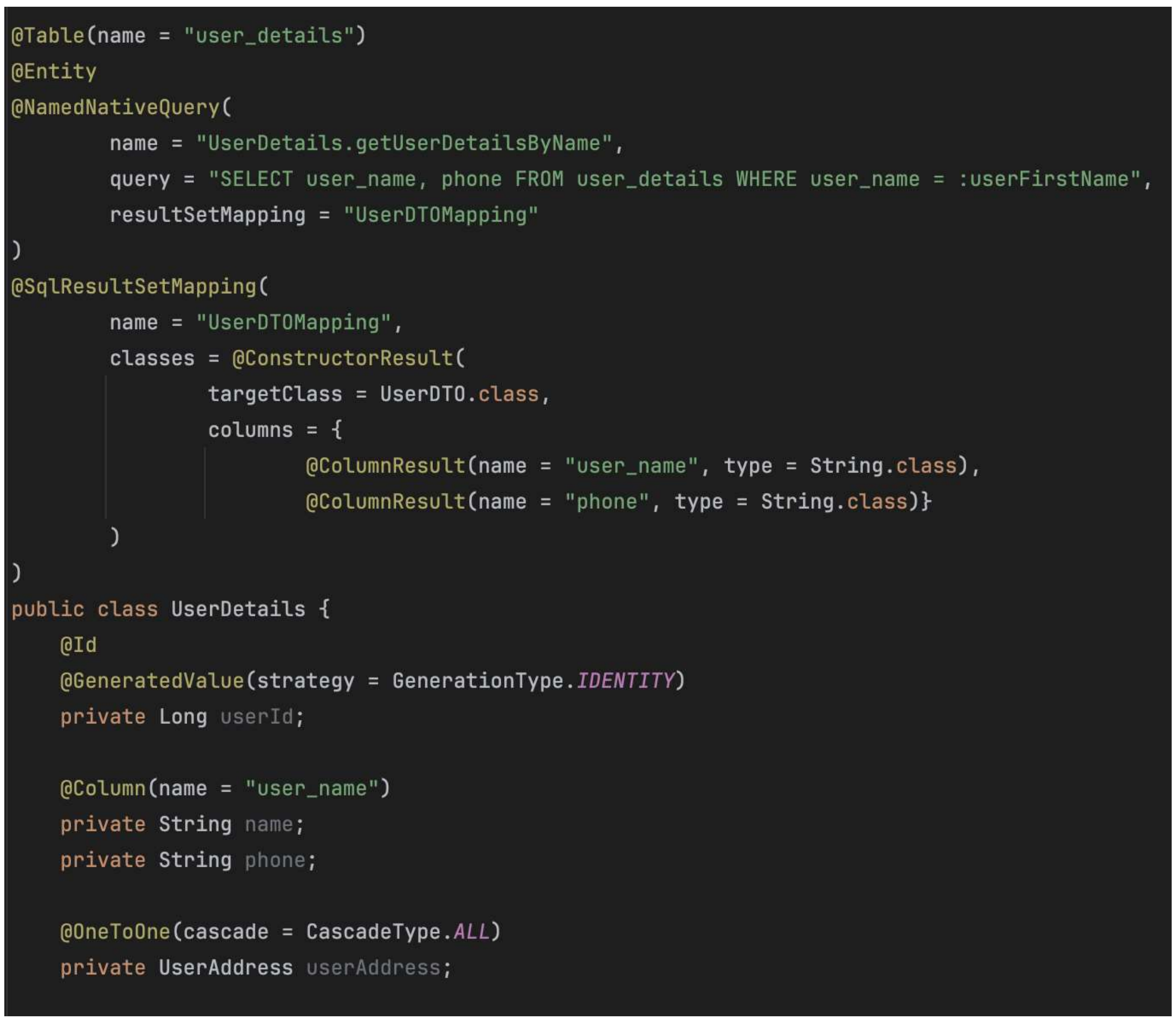
```
{
    "timestamp": "2025-02-08T10:28:20.735+00:00",
    "status": 500,
    "error": "Internal Server Error",
    "path": "/api/user/BB"
}
```



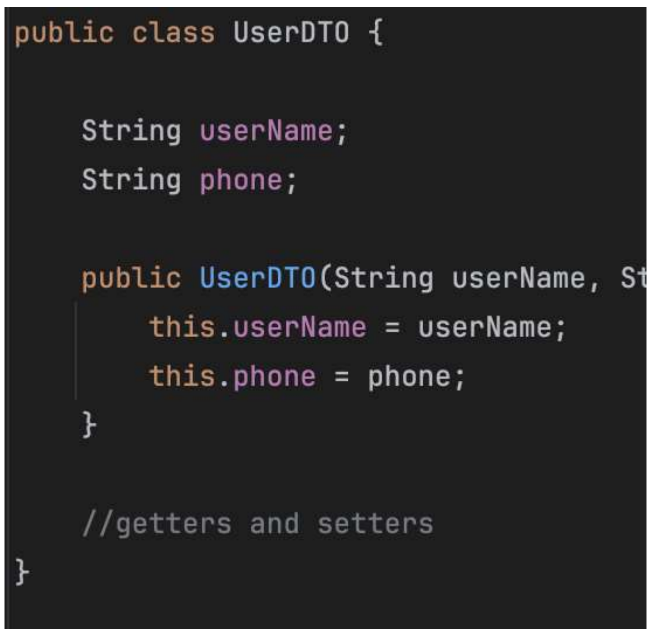
```
org.h2.jdbc.JdbcSQLSyntaxErrorException Create breakpoint : Column "user_id" not found [42122-224]
    at org.h2.message.DbException.getJdbcSQLException(DbException.java:514) ~[h2-2.2.224.jar:2.2.224]
    at org.h2.message.DbException.getJdbcSQLException(DbException.java:489) ~[h2-2.2.224.jar:2.2.224]
    at org.h2.message.DbException.get(DbException.java:223) ~[h2-2.2.224.jar:2.2.224]
    at org.h2.message.DbException.get(DbException.java:199) ~[h2-2.2.224.jar:2.2.224]
    at org.h2.jdbc.JdbcResultSet.getColumnIndex(JdbcResultSet.java:3518) ~[h2-2.2.224.jar:2.2.224]
    at org.h2.jdbc.JdbcResultSet.findColumn(JdbcResultSet.java:178) ~[h2-2.2.224.jar:2.2.224]
```

We need to manually tell JPA, how to do the mapping.

1st: Using **@SqlResultSetMapping and @NamedNativeQuery** Annotation

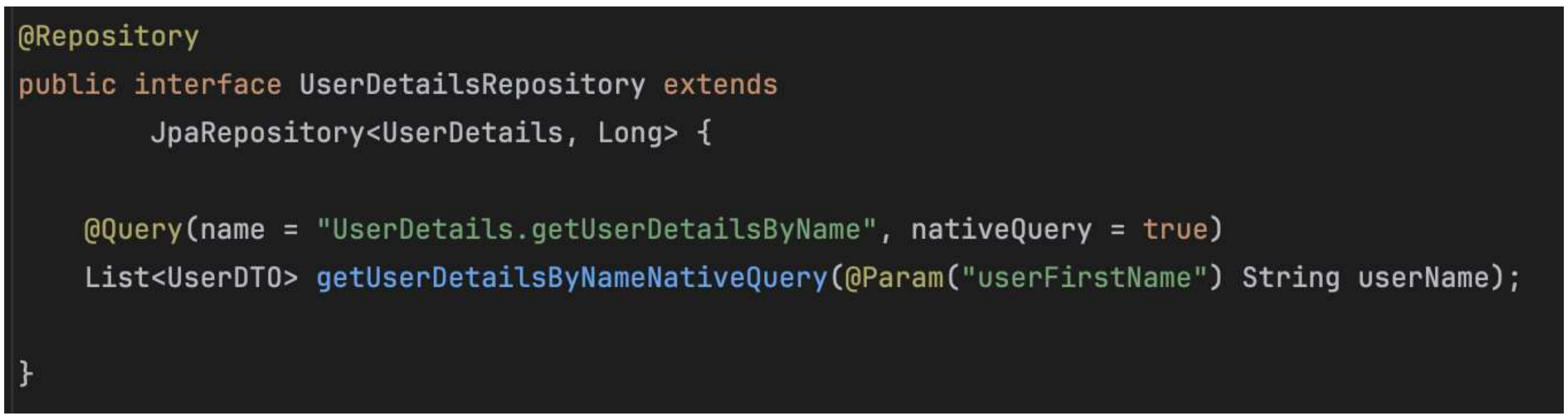


```
@Table(name = "user_details")
@Entity
@NamedNativeQuery(
        name = "UserDetails.getUserDetailsByName",
        query = "SELECT user_name, phone FROM user_details WHERE user_name = :userFirstName",
        resultSetMapping = "UserDTOMapping"
)
@SqlResultSetMapping(
        name = "UserDTOMapping",
        classes = @ConstructorResult(
                targetClass = UserDTO.class,
                columns = {
                        @ColumnResult(name = "user_name", type = String.class),
                        @ColumnResult(name = "phone", type = String.class)}
        )
)
public class UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;

    @Column(name = "user_name")
    private String name;
    private String phone;

    @OneToOne(cascade = CascadeType.ALL)
    private UserAddress userAddress;
```

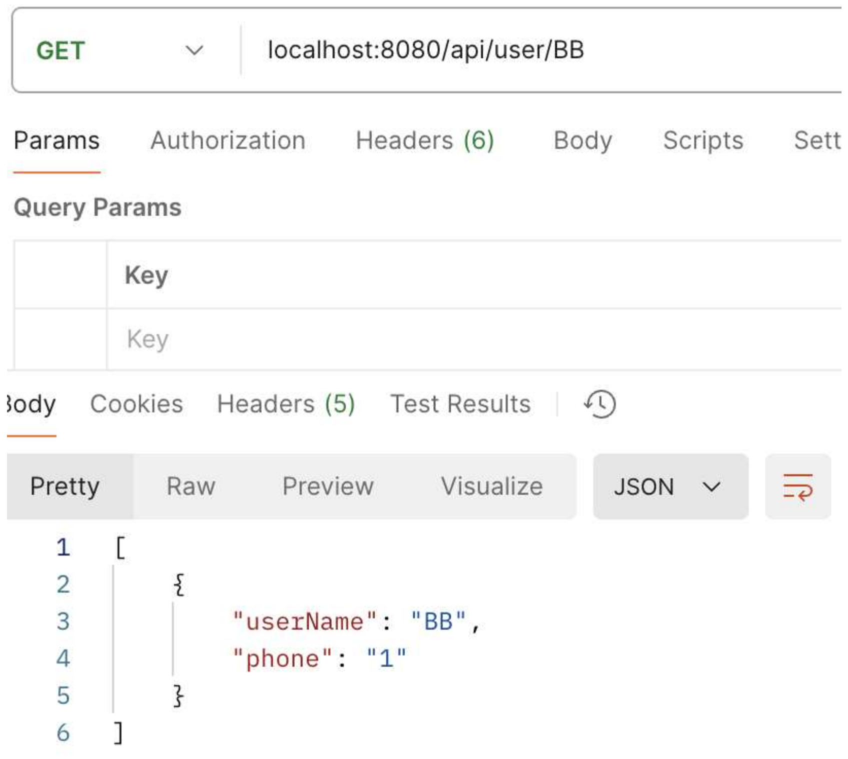


```
public class UserDTO {

    String userName;
    String phone;

    public UserDTO(String userName, St
        this.userName = userName;
        this.phone = phone;
    }

    //getters and setters
}
```

```
    //getters and setters
}
```



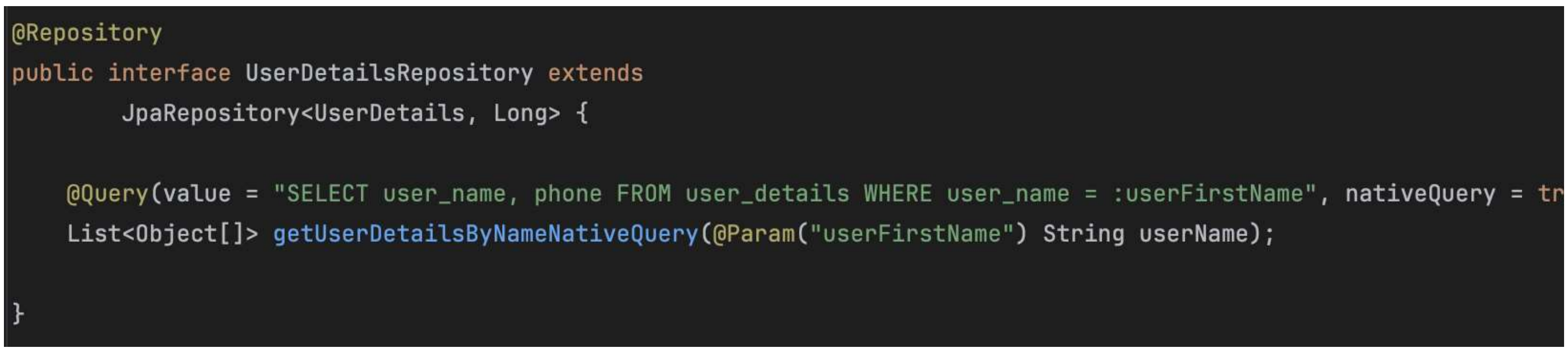
```
@Repository
public interface UserDetailsRepository extends
        JpaRepository<UserDetails, Long> {

    @Query(name = "UserDetails.getUserDetailsByName", nativeQuery = true)
    List<UserDTO> getUserDetailsByNameNativeQuery(@Param("userFirstName") String userName);

}
```



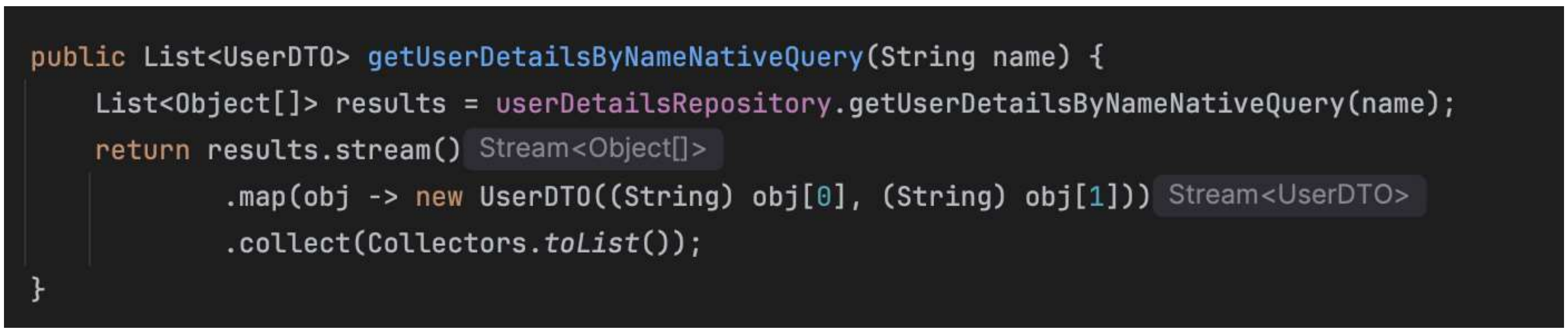
GET localhost:8080/api/user/BB

```
[
    {
        "userName": "BB",
        "phone": "1"
    }
]
```

2nd: With Manual mapping



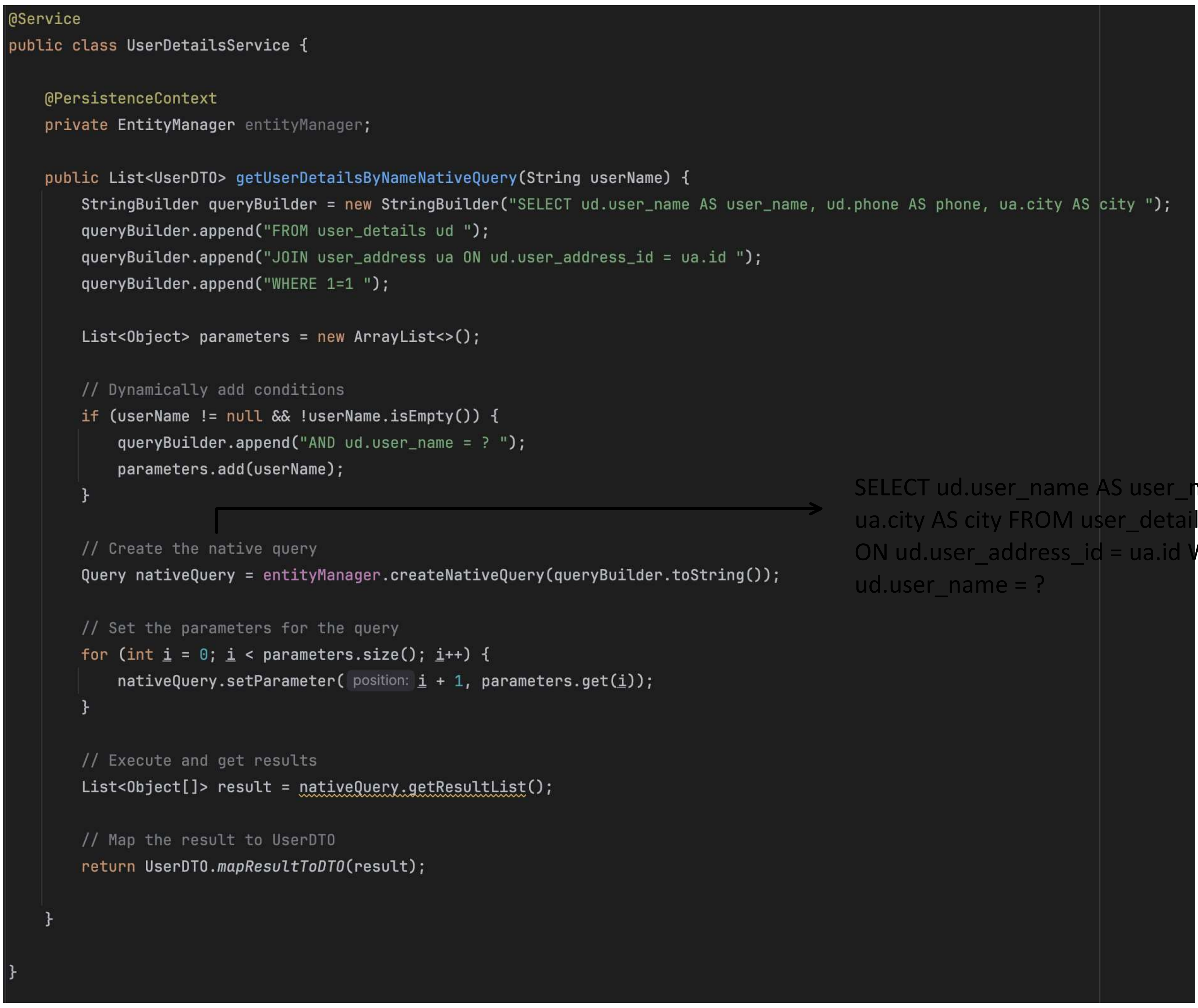
```
@Repository
public interface UserDetailsRepository extends
        JpaRepository<UserDetails, Long> {

    @Query(value = "SELECT user_name, phone FROM user_details WHERE user_name = :userFirstName", nativeQuery = tr
    List<Object[]> getUserDetailsByNameNativeQuery(@Param("userFirstName") String userName);

}
```



```
public List<UserDTO> getUserDetailsByNameNativeQuery(String name) {
    List<Object[]> results = userDetailsRepository.getUserDetailsByNameNativeQuery(name);
    return results.stream() Stream<Object[]>
            .map(obj -> new UserDTO((String) obj[0], (String) obj[1])) Stream<UserDTO>
            .collect(Collectors.toList());
}
```

Dynamic Native Query:

Dynamic Native Query:

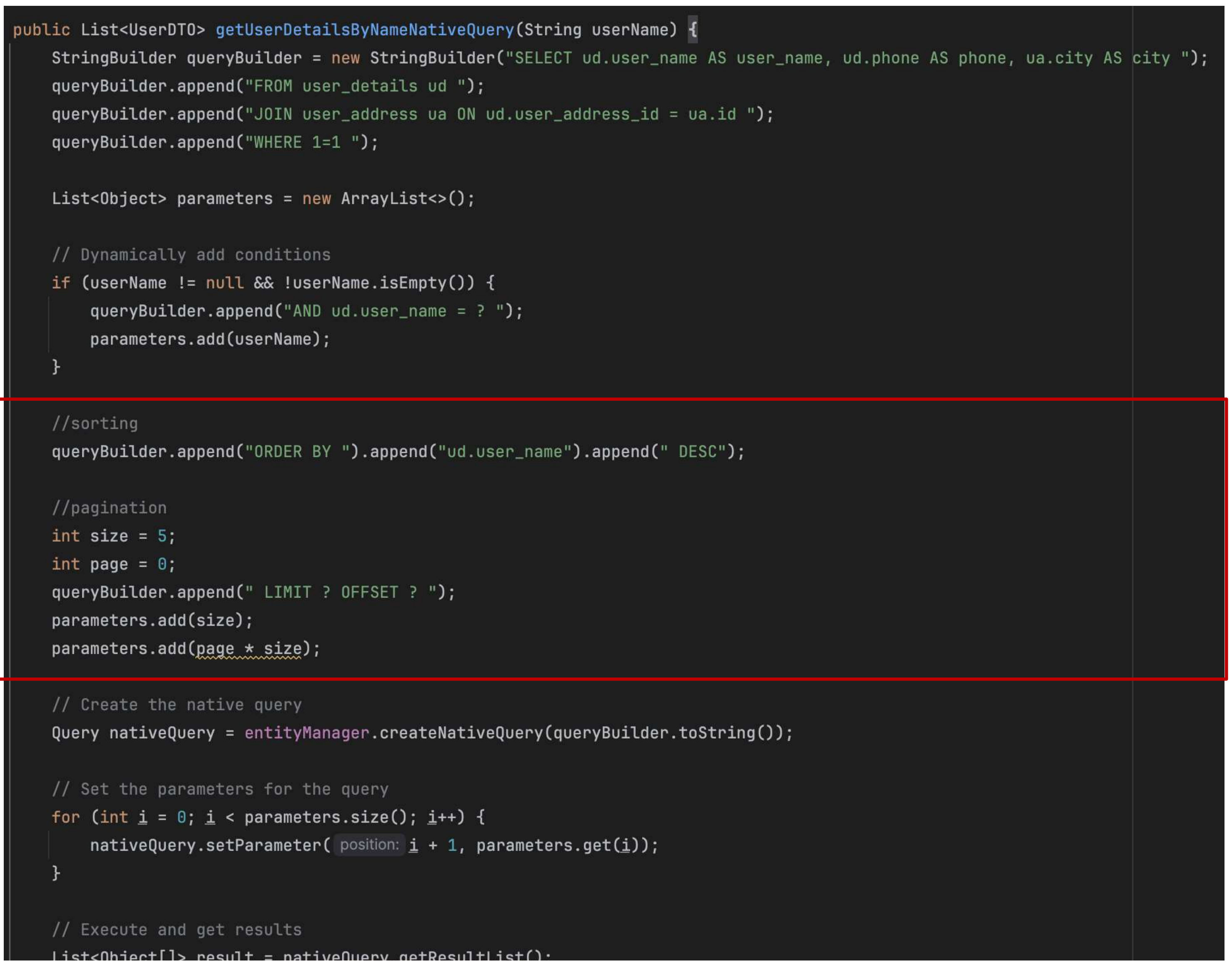


```java
@Service
public class UserDetailsService {

    @PersistenceContext
    private EntityManager entityManager;

    public List<UserDTO> getUserDetailsByNameNativeQuery(String userName) {
        StringBuilder queryBuilder = new StringBuilder("SELECT ud.user_name AS user_name, ud.phone AS phone, ua.city AS city ");
        queryBuilder.append("FROM user_details ud ");
        queryBuilder.append("JOIN user_address ua ON ud.user_address_id = ua.id ");
        queryBuilder.append("WHERE 1=1 ");

        List<Object> parameters = new ArrayList<>();

        // Dynamically add conditions
        if (userName != null && !userName.isEmpty()) {
            queryBuilder.append("AND ud.user_name = ? ");
            parameters.add(userName);
        }

        // Create the native query
        Query nativeQuery = entityManager.createNativeQuery(queryBuilder.toString());

        // Set the parameters for the query
        for (int i = 0; i < parameters.size(); i++) {
            nativeQuery.setParameter( position: i + 1, parameters.get(i));
        }

        // Execute and get results
        List<Object[]> result = nativeQuery.getResultList();

        // Map the result to UserDTO
        return UserDTO.mapResultToDTO(result);

    }

}
```

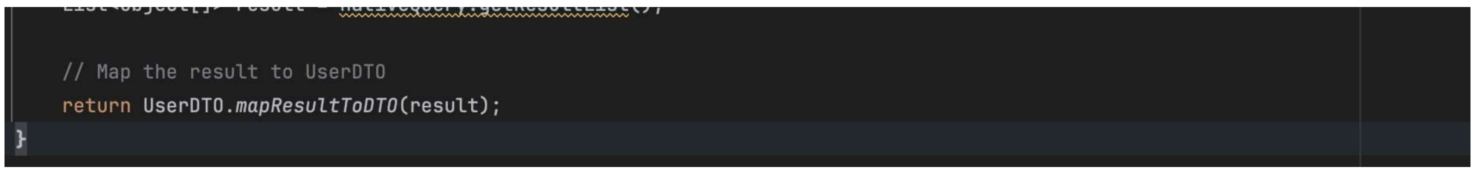
SELECT ud.user_name AS user_name, u
ua.city AS city FROM user_details ud JO
ON ud.user_address_id = ua.id WHERE
ud.user_name = ?

Pagination and Sorting in Native SQL:

1st way:

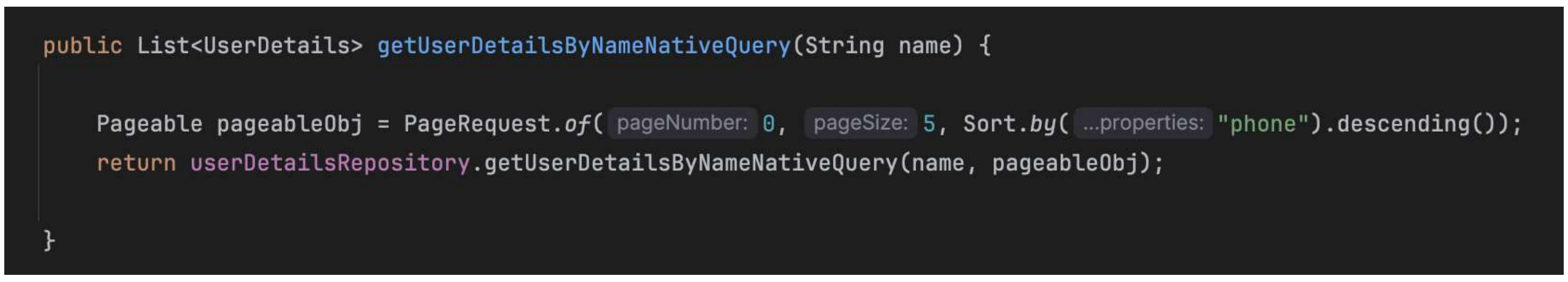


```java
public List<UserDTO> getUserDetailsByNameNativeQuery(String userName) {
    StringBuilder queryBuilder = new StringBuilder("SELECT ud.user_name AS user_name, ud.phone AS phone, ua.city AS city ");
    queryBuilder.append("FROM user_details ud ");
    queryBuilder.append("JOIN user_address ua ON ud.user_address_id = ua.id ");
    queryBuilder.append("WHERE 1=1 ");

    List<Object> parameters = new ArrayList<>();

    // Dynamically add conditions
    if (userName != null && !userName.isEmpty()) {
        queryBuilder.append("AND ud.user_name = ? ");
        parameters.add(userName);
    }

    //sorting
    queryBuilder.append("ORDER BY ").append("ud.user_name").append(" DESC");

    //pagination
    int size = 5;
    int page = 0;
    queryBuilder.append(" LIMIT ? OFFSET ? ");
    parameters.add(size);
    parameters.add(page * size);

    // Create the native query
    Query nativeQuery = entityManager.createNativeQuery(queryBuilder.toString());

    // Set the parameters for the query
    for (int i = 0; i < parameters.size(); i++) {
        nativeQuery.setParameter( position: i + 1, parameters.get(i));
    }

    // Execute and get results
    List<Object[]> result = nativeQuery.getResultList();
```

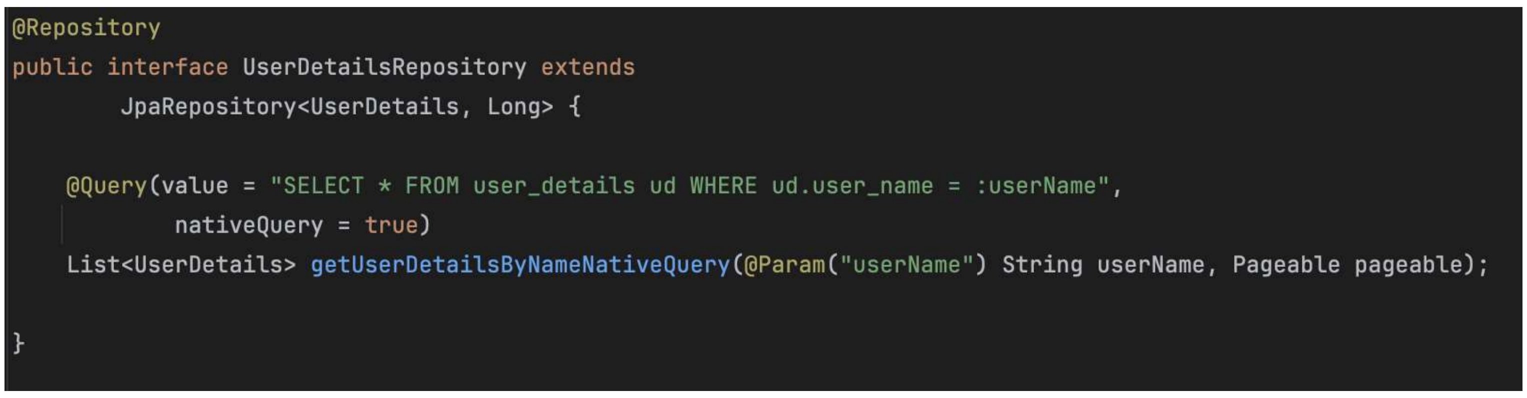


```java
    // Map the result to UserDTO
    return UserDTO.mapResultToDTO(result);
}
```

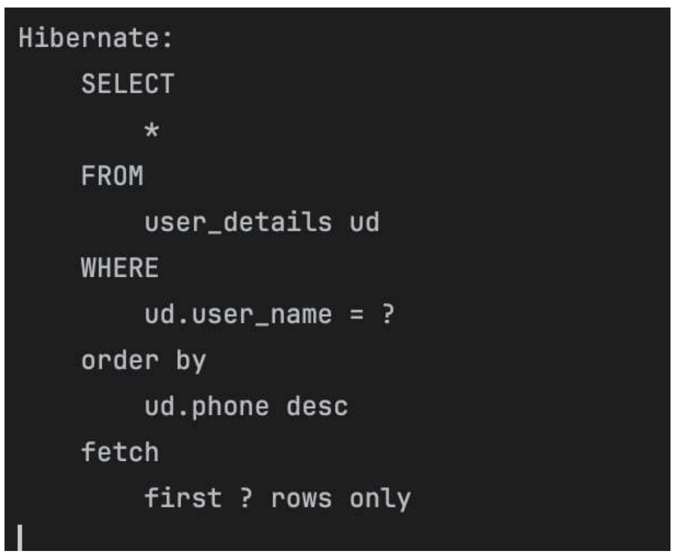
2nd way:



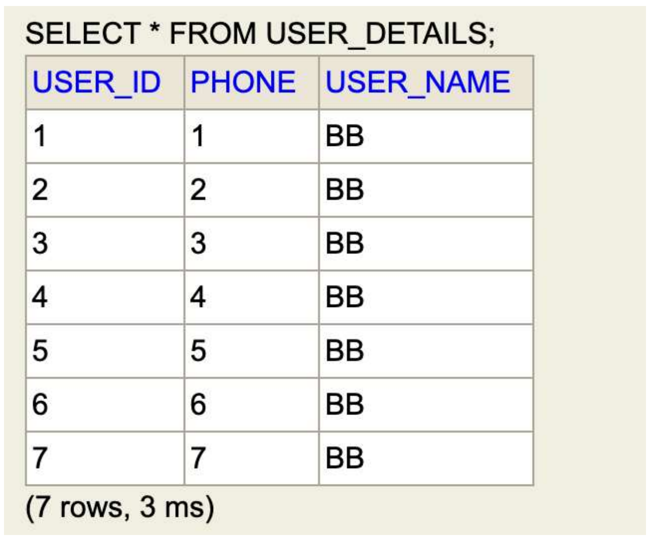
```java
public List<UserDetails> getUserDetailsByNameNativeQuery(String name) {

    Pageable pageableObj = PageRequest.of( pageNumber: 0,  pageSize: 5, Sort.by( ...properties: "phone").descending());
    return userDetailsRepository.getUserDetailsByNameNativeQuery(name, pageableObj);

}
```



```java
@Repository
public interface UserDetailsRepository extends
        JpaRepository<UserDetails, Long> {

    @Query(value = "SELECT * FROM user_details ud WHERE ud.user_name = :userName",
            nativeQuery = true)
    List<UserDetails> getUserDetailsByNameNativeQuery(@Param("userName") String userName, Pageable pageable);

}
```



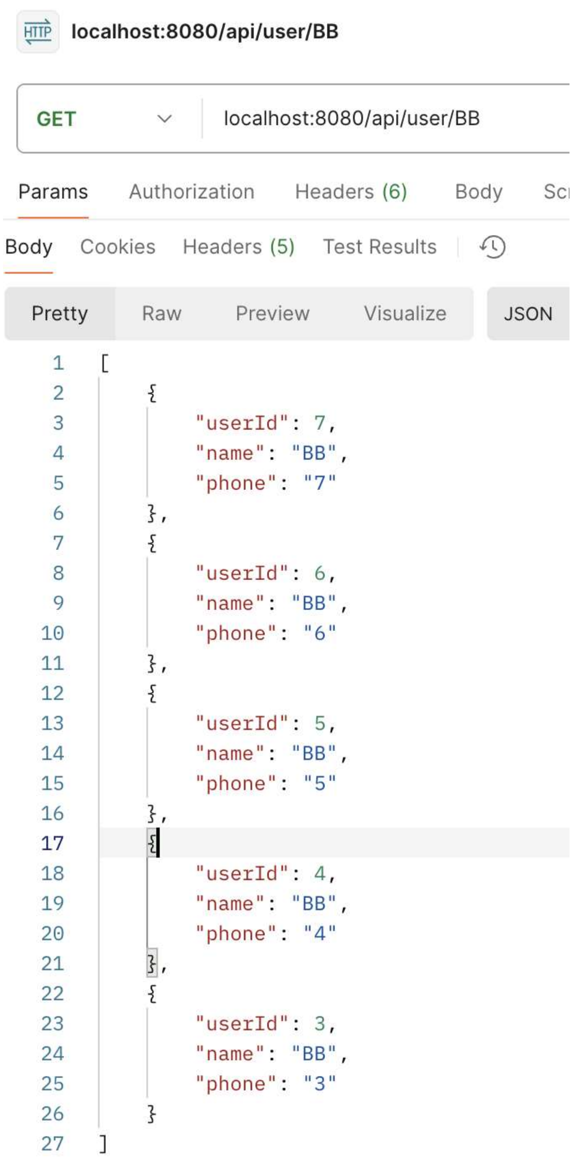
```
Hibernate:
    SELECT
        *
    FROM
        user_details ud
    WHERE
        ud.user_name = ?
    order by
        ud.phone desc
    fetch
        first ? rows only
```



SELECT * FROM USER_DETAILS;

USER_ID	PHONE	USER_NAME
1	1	BB
2	2	BB
3	3	BB
4	4	BB
5	5	BB
6	6	BB
7	7	BB

(7 rows, 3 ms)



localhost:8080/api/user/BB

GET localhost:8080/api/user/BB

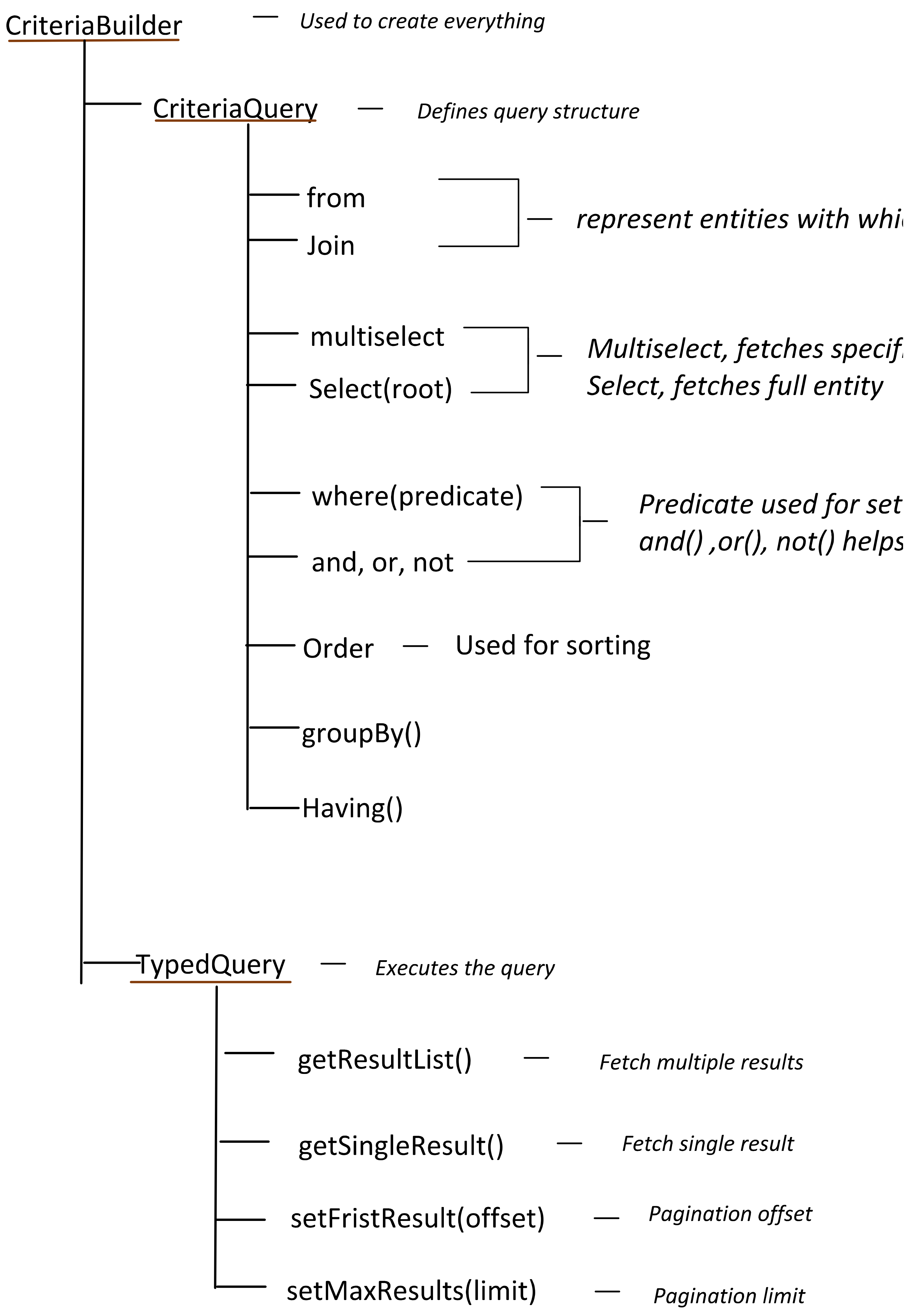
```json
[
    {
        "userId": 7,
        "name": "BB",
        "phone": "7"
    },
    {
        "userId": 6,
        "name": "BB",
        "phone": "6"
    },
    {
        "userId": 5,
        "name": "BB",
        "phone": "5"
    },
    {
        "userId": 4,
        "name": "BB",
        "phone": "4"
    },
    {
        "userId": 3,
        "name": "BB",
        "phone": "3"
    }
]
```

Criteria API:

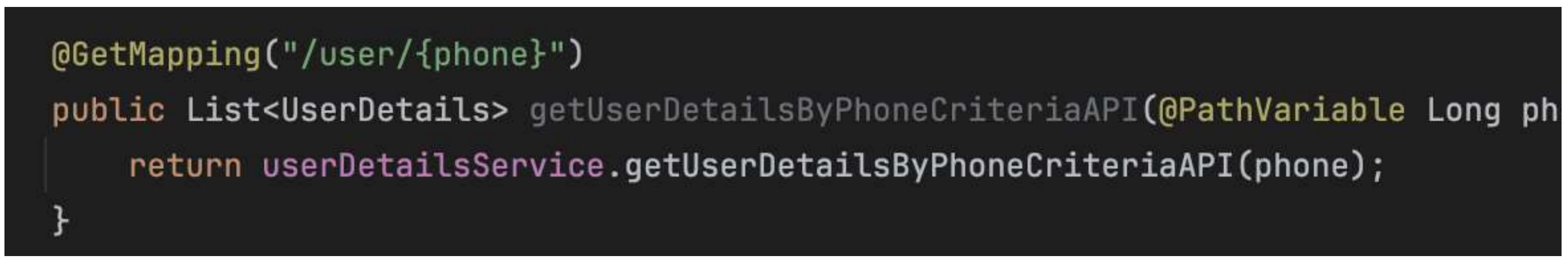
Native SQL queries support dynamic query building, but they are database-dependent and leverage JPA abstraction.

That’s why **JPA Criteria API** exists, it allows you to build dynamic, type-safe queries withou raw SQL.

Lets understand the Hierarchy

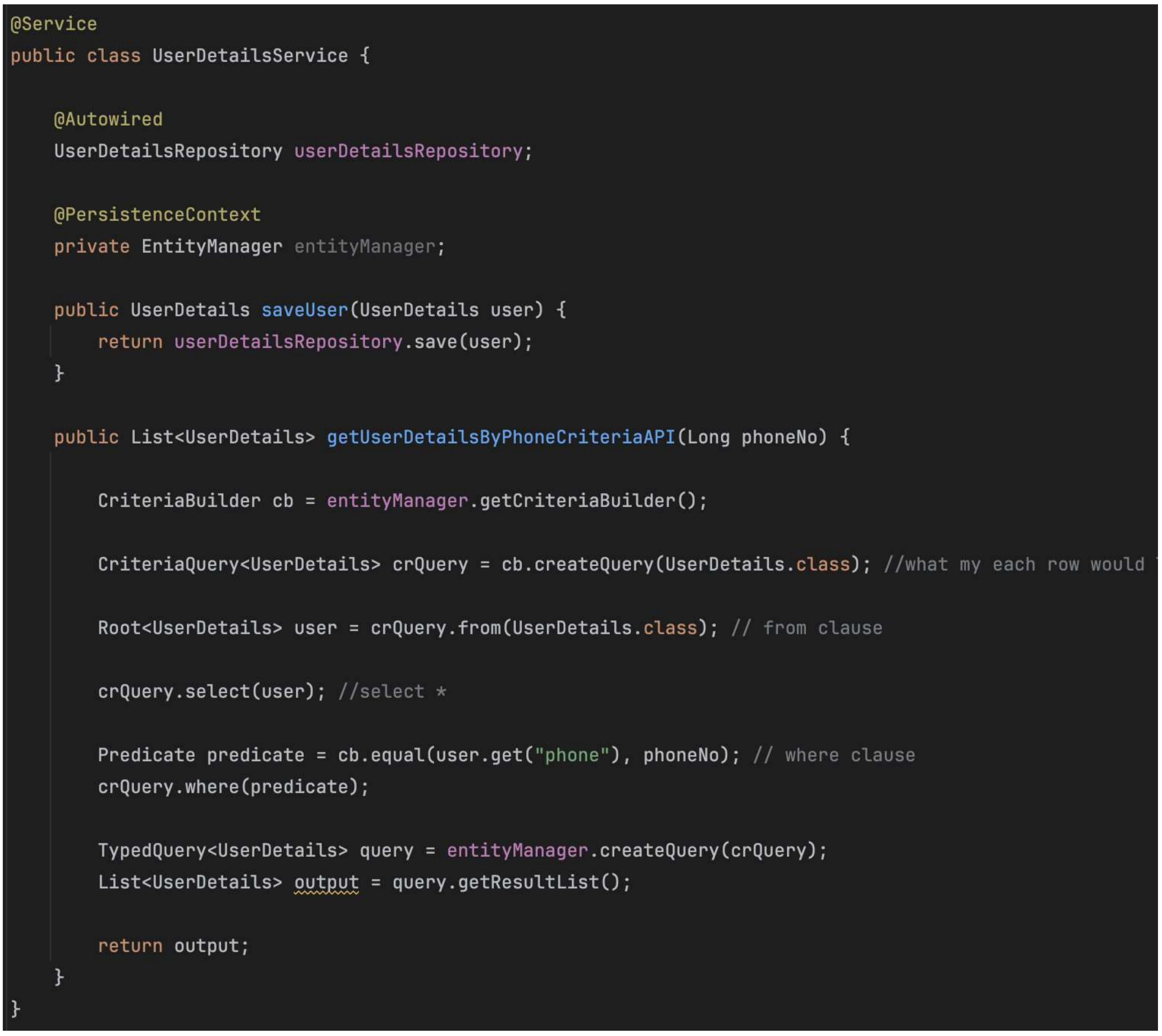

CriteriaBuilder — *Used to create everything*
CriteriaQuery — *Defines query structure*
from
Join
— *represent entities with whi*
multiselect
Select(root)
— *Multiselect, fetches specif*
Select, fetches full entity
where(predicate)
and, or, not
— *Predicate used for set*
and() ,or(), not() helps
Order — Used for sorting
groupBy()
Having()
TypedQuery — *Executes the query*
getResultList() — *Fetch multiple results*
getSingleResult() — *Fetch single result*
setFristResult(offset) — *Pagination offset*
setMaxResults(limit) — *Pagination limit*

Controller class:



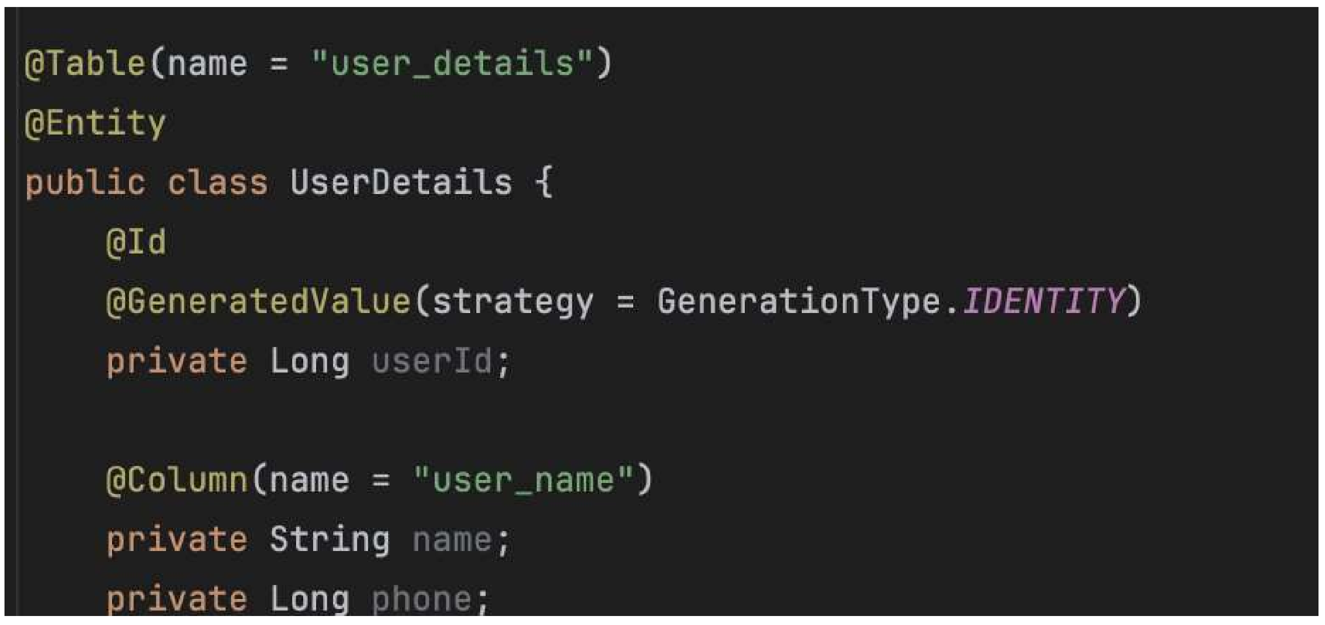
```
@GetMapping("/user/{phone}")
public List<UserDetails> getUserDetailsByPhoneCriteriaAPI(@PathVariable Long ph
    return userDetailsService.getUserDetailsByPhoneCriteriaAPI(phone);
}
```

Service class:



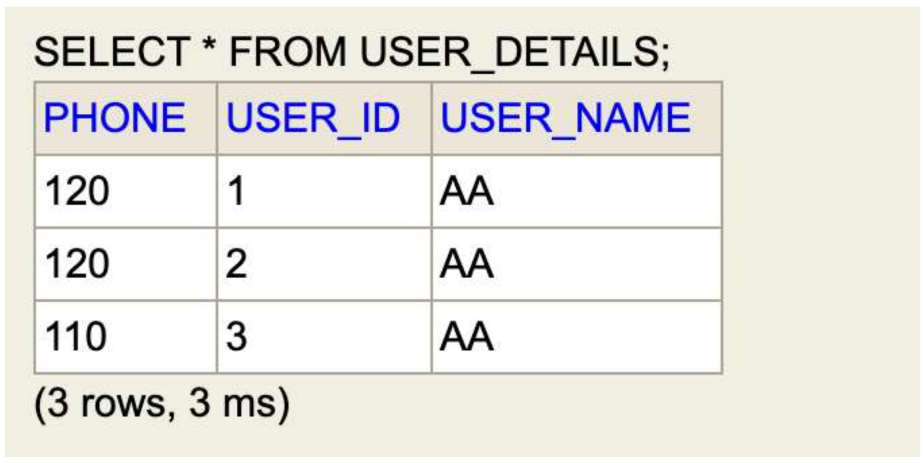
```
@Service
public class UserDetailsService {

    @Autowired
    UserDetailsRepository userDetailsRepository;

    @PersistenceContext
    private EntityManager entityManager;

    public UserDetails saveUser(UserDetails user) {
        return userDetailsRepository.save(user);
    }

    public List<UserDetails> getUserDetailsByPhoneCriteriaAPI(Long phoneNo) {

        CriteriaBuilder cb = entityManager.getCriteriaBuilder();

        CriteriaQuery<UserDetails> crQuery = cb.createQuery(UserDetails.class); //what my each row would

        Root<UserDetails> user = crQuery.from(UserDetails.class); // from clause

        crQuery.select(user); //select *

        Predicate predicate = cb.equal(user.get("phone"), phoneNo); // where clause
        crQuery.where(predicate);

        TypedQuery<UserDetails> query = entityManager.createQuery(crQuery);
        List<UserDetails> output = query.getResultList();

        return output;
    }
}
```

Entity class:



```
@Table(name = "user_details")
@Entity
public class UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;

    @Column(name = "user_name")
    private String name;
    private Long phone;
```

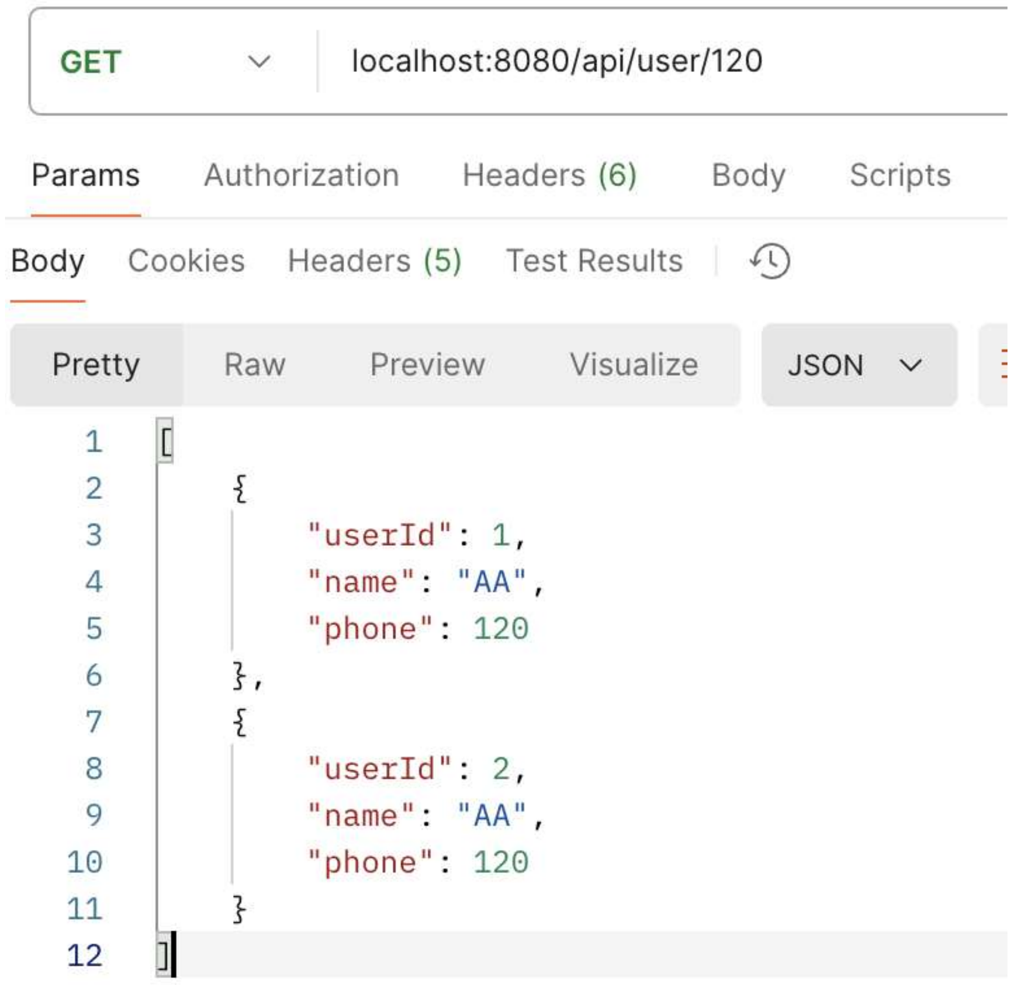
```
    //getters and setters
}
```



SELECT * FROM USER_DETAILS;

PHONE	USER_ID	USER_NAME
120	1	AA
120	2	AA
110	3	AA

(3 rows, 3 ms)



GET localhost:8080/api/user/120

```
[
    {
        "userId": 1,
        "name": "AA",
        "phone": 120
    },
    {
        "userId": 2,
        "name": "AA",
        "phone": 120
    }
]
```

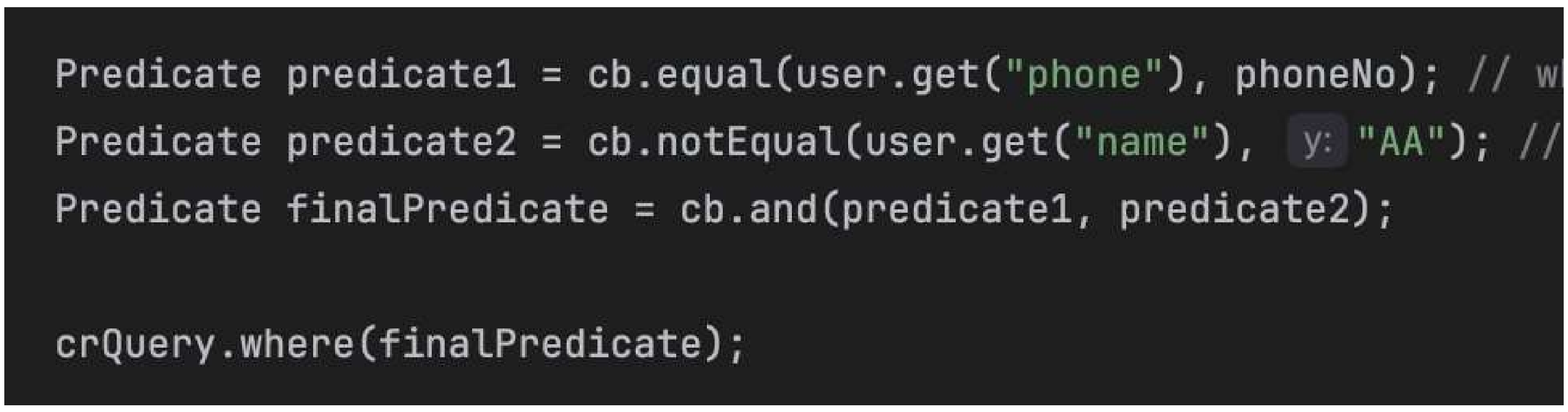
Comparison operator:

Root<UserDetails>user=crQuery.from(UserDetails.class); //from clau

Method	Description
cb.equal(user.get("phone"),123);	Check for equality
cb.notEqual(user.get("phone"),123);	Check for in-equality
cb.gt(user.get("phone"),123);	Greater than
cb.ge(user.get("phone"),123);	Greater than or equal
cb.lt(user.get("phone"),123);	Less than
cb.le(user.get("phone"),123);	Less than or equal

Logical operator:

Method	Description	
cb.and(predicate1, predicate2);	Combining two conditions using and	W
cb.or(predicate1, predicate2);	Combining two conditions using or	V
cb.not(predicate1,);	Negate the condition	



```
Predicate predicate1 = cb.equal(user.get("phone"), phoneNo); // w
Predicate predicate2 = cb.notEqual(user.get("name"),  y: "AA"); //
Predicate finalPredicate = cb.and(predicate1, predicate2);

crQuery.where(finalPredicate);
```

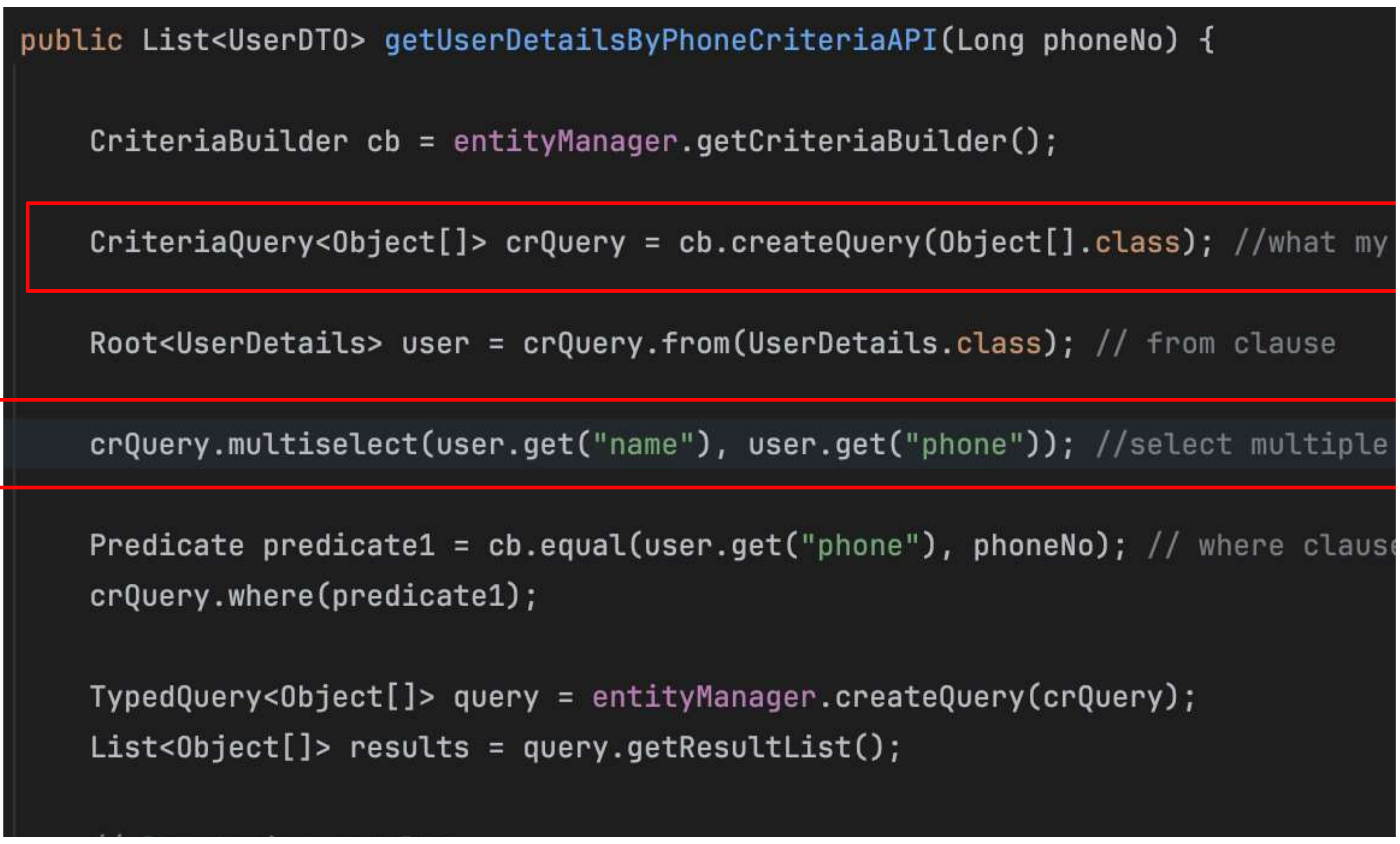
Strings Operations:

Method	Description
cb.like(user.get("name"), "S%");	Name starts with J
cb.notLike(user.get("name"), "S%");	Name do not start with J

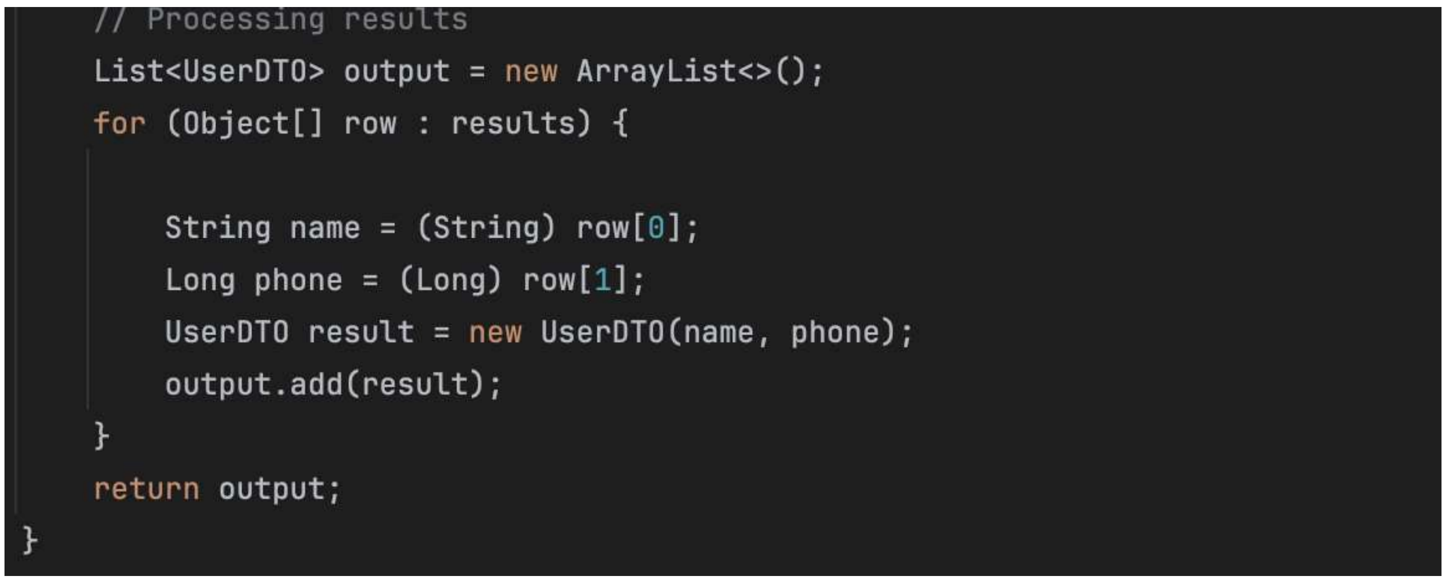
Collection Operations:

Method	
cb.in(user.get("phone")).value(11).value(7);	Che
cb.not(user.get("phone").in(11,7));	Che

Select Multiple fields:

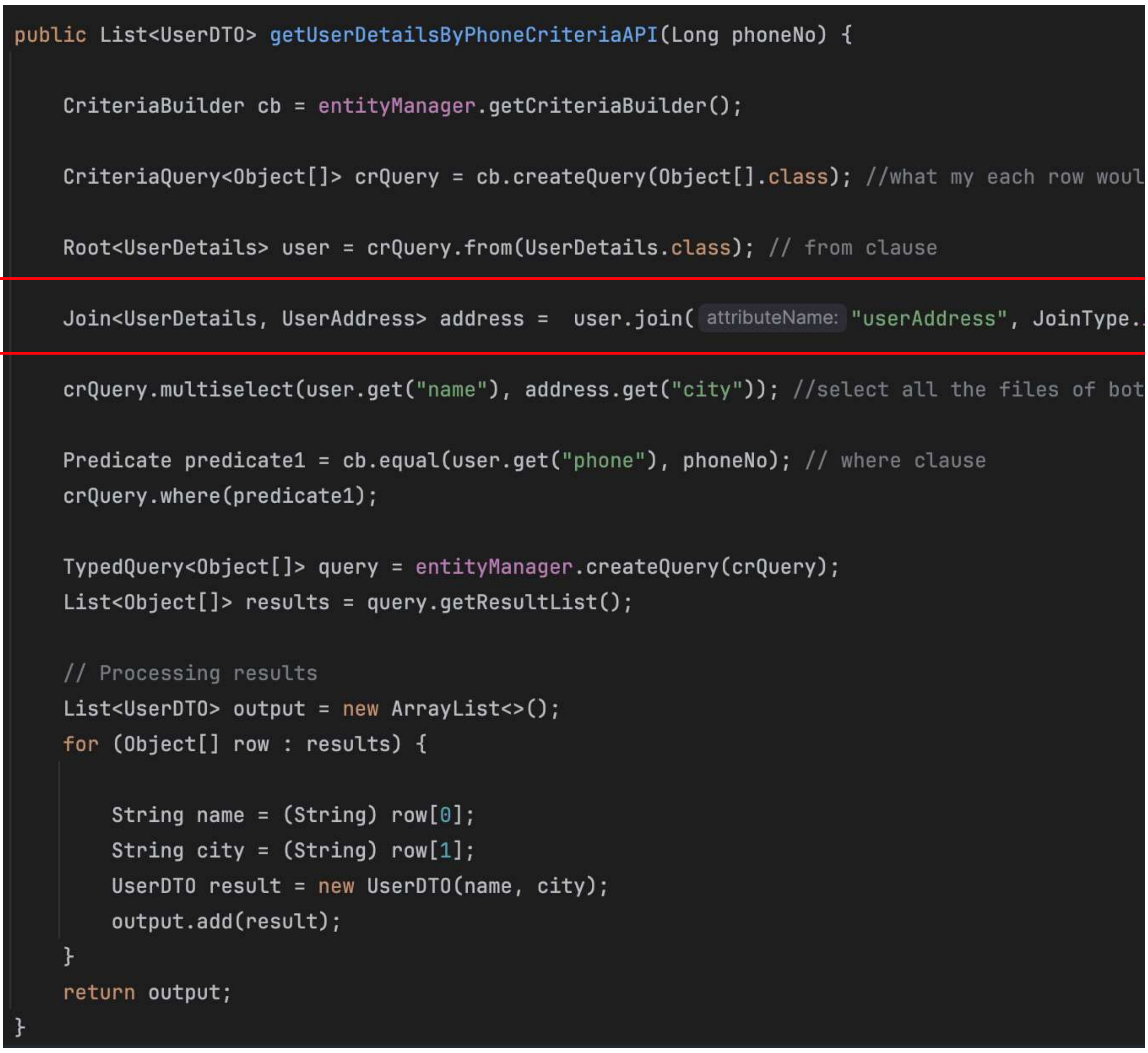


```
public List<UserDTO> getUserDetailsByPhoneCriteriaAPI(Long phoneNo) {

    CriteriaBuilder cb = entityManager.getCriteriaBuilder();

    CriteriaQuery<Object[]> crQuery = cb.createQuery(Object[].class); //what my

    Root<UserDetails> user = crQuery.from(UserDetails.class); // from clause

    crQuery.multiselect(user.get("name"), user.get("phone")); //select multiple

    Predicate predicate1 = cb.equal(user.get("phone"), phoneNo); // where claus
    crQuery.where(predicate1);

    TypedQuery<Object[]> query = entityManager.createQuery(crQuery);
    List<Object[]> results = query.getResultList();
```



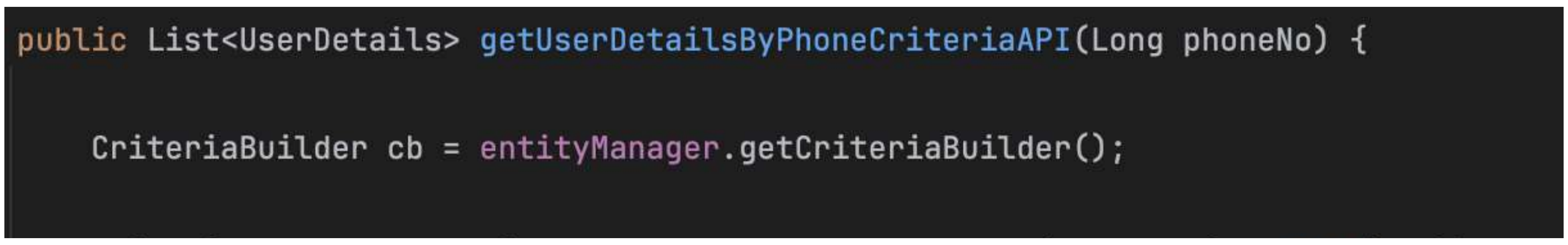
```
    // Processing results
    List<UserDTO> output = new ArrayList<>();
    for (Object[] row : results) {

        String name = (String) row[0];
        Long phone = (Long) row[1];
        UserDTO result = new UserDTO(name, phone);
        output.add(result);
    }
    return output;
}
```

Join

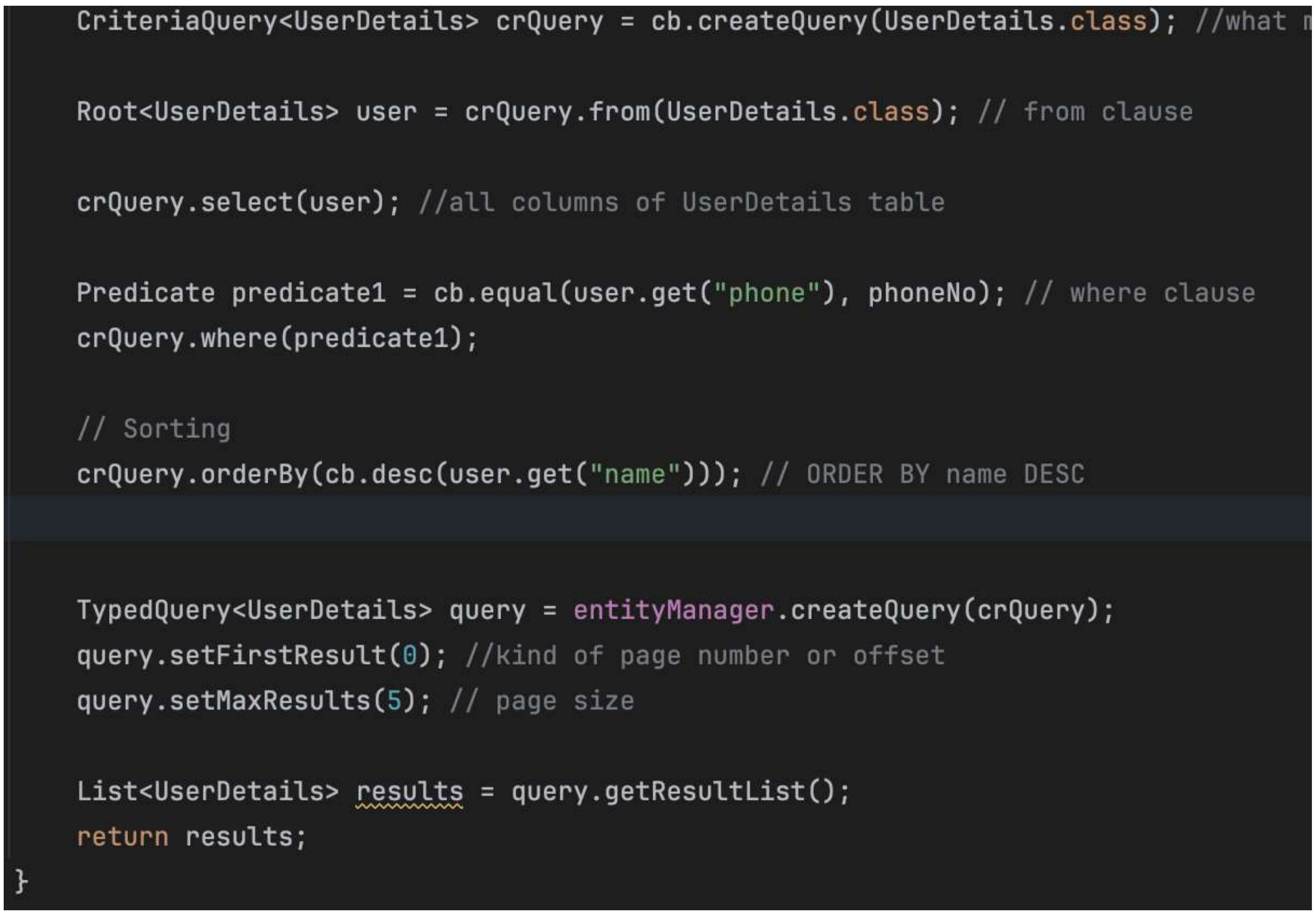


```
public List<UserDTO> getUserDetailsByPhoneCriteriaAPI(Long phoneNo) {

    CriteriaBuilder cb = entityManager.getCriteriaBuilder();

    CriteriaQuery<Object[]> crQuery = cb.createQuery(Object[].class); //what my each row woul

    Root<UserDetails> user = crQuery.from(UserDetails.class); // from clause

    Join<UserDetails, UserAddress> address =  user.join( attributeName: "userAddress", JoinType.

    crQuery.multiselect(user.get("name"), address.get("city")); //select all the files of bot

    Predicate predicate1 = cb.equal(user.get("phone"), phoneNo); // where clause
    crQuery.where(predicate1);

    TypedQuery<Object[]> query = entityManager.createQuery(crQuery);
    List<Object[]> results = query.getResultList();

    // Processing results
    List<UserDTO> output = new ArrayList<>();
    for (Object[] row : results) {

        String name = (String) row[0];
        String city = (String) row[1];
        UserDTO result = new UserDTO(name, city);
        output.add(result);
    }
    return output;
}
```

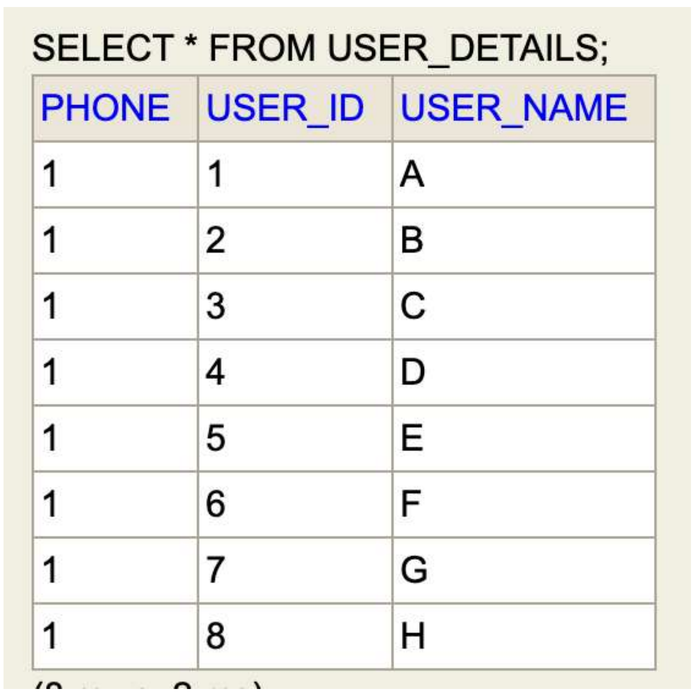
Pagination and Sorting



```
public List<UserDetails> getUserDetailsByPhoneCriteriaAPI(Long phoneNo) {

    CriteriaBuilder cb = entityManager.getCriteriaBuilder();
```

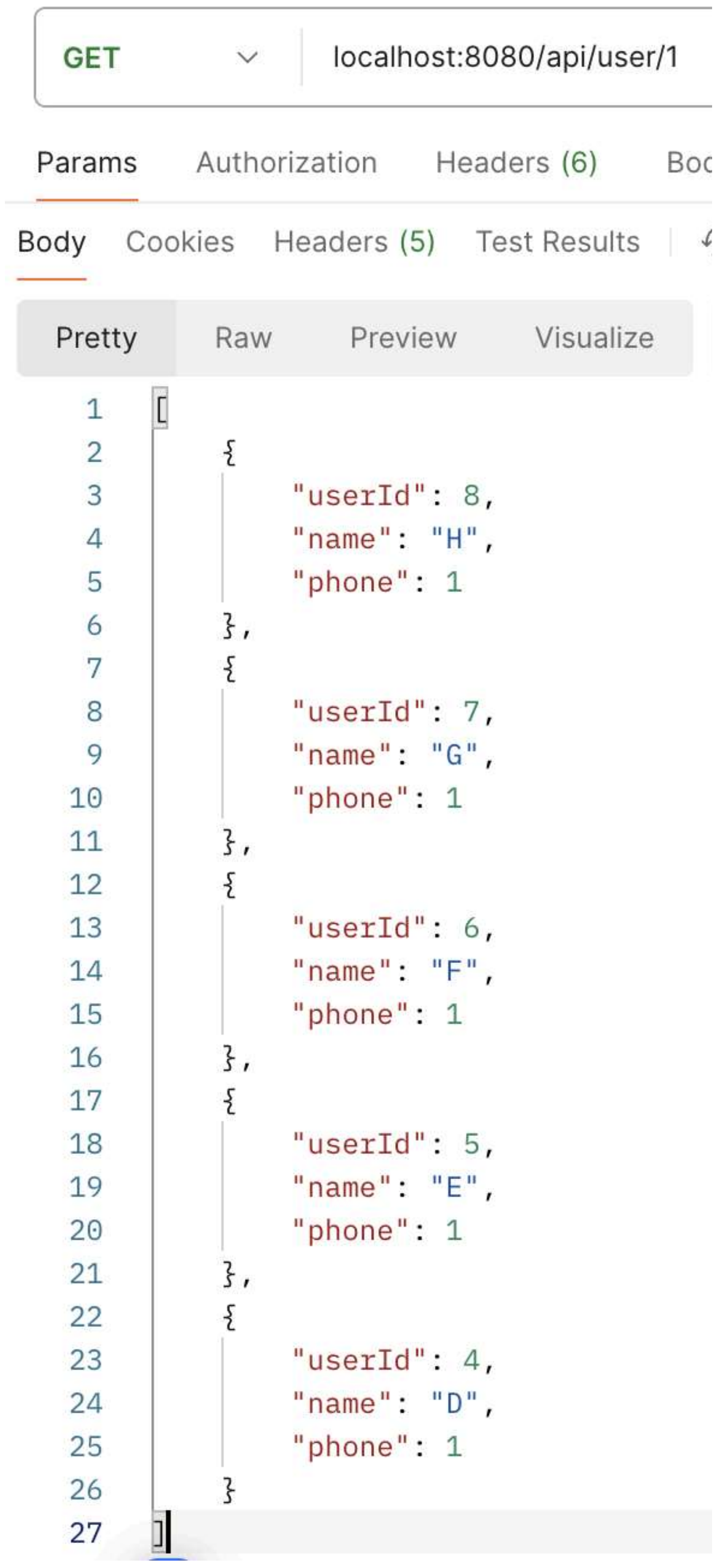


```java
    CriteriaQuery<UserDetails> crQuery = cb.createQuery(UserDetails.class); //what m

    Root<UserDetails> user = crQuery.from(UserDetails.class); // from clause

    crQuery.select(user); //all columns of UserDetails table

    Predicate predicate1 = cb.equal(user.get("phone"), phoneNo); // where clause
    crQuery.where(predicate1);

    // Sorting
    crQuery.orderBy(cb.desc(user.get("name"))); // ORDER BY name DESC


    TypedQuery<UserDetails> query = entityManager.createQuery(crQuery);
    query.setFirstResult(0); //kind of page number or offset
    query.setMaxResults(5); // page size

    List<UserDetails> results = query.getResultList();
    return results;
}
```



SELECT * FROM USER_DETAILS;

PHONE	USER_ID	USER_NAME
1	1	A
1	2	B
1	3	C
1	4	D
1	5	E
1	6	F
1	7	G
1	8	H



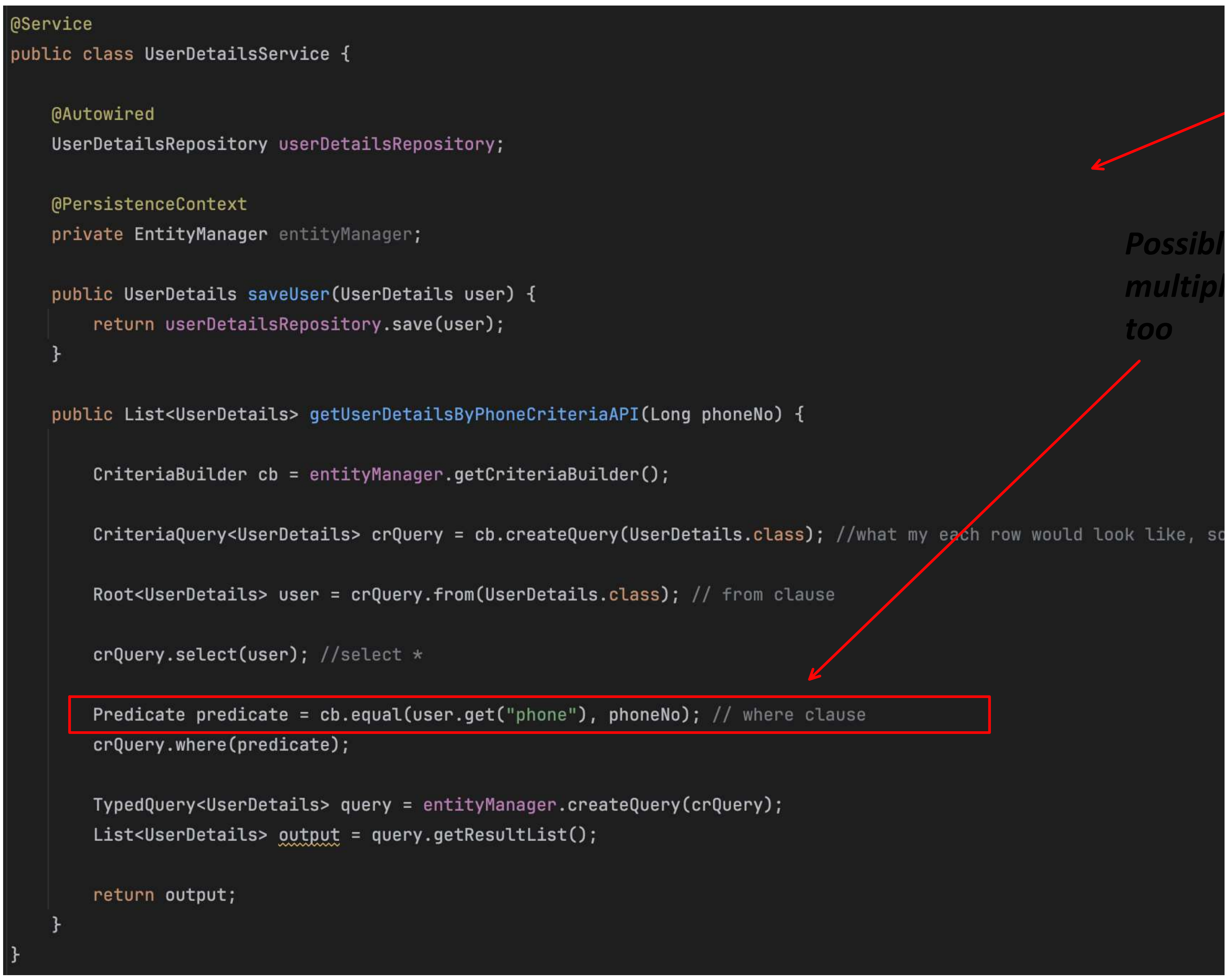
GET localhost:8080/api/user/1

```json
[
    {
        "userId": 8,
        "name": "H",
        "phone": 1
    },
    {
        "userId": 7,
        "name": "G",
        "phone": 1
    },
    {
        "userId": 6,
        "name": "F",
        "phone": 1
    },
    {
        "userId": 5,
        "name": "E",
        "phone": 1
    },
    {
        "userId": 4,
        "name": "D",
        "phone": 1
    }
]
```

Specification API

1st problem it solves is: ***CODE DUPLICITY***

- In Criteria API its possible that, same filter (predicate) used at multiple methods an of Code Duplicity.



```java
@Service
public class UserDetailsService {

    @Autowired
    UserDetailsRepository userDetailsRepository;

    @PersistenceContext
    private EntityManager entityManager;

    public UserDetails saveUser(UserDetails user) {
        return userDetailsRepository.save(user);
    }

    public List<UserDetails> getUserDetailsByPhoneCriteriaAPI(Long phoneNo) {

        CriteriaBuilder cb = entityManager.getCriteriaBuilder();

        CriteriaQuery<UserDetails> crQuery = cb.createQuery(UserDetails.class); //what my each row would look like, sc

        Root<UserDetails> user = crQuery.from(UserDetails.class); // from clause

        crQuery.select(user); //select *

        Predicate predicate = cb.equal(user.get("phone"), phoneNo); // where clause
        crQuery.where(predicate);

        TypedQuery<UserDetails> query = entityManager.createQuery(crQuery);
        List<UserDetails> output = query.getResultList();

        return output;
    }
}
```

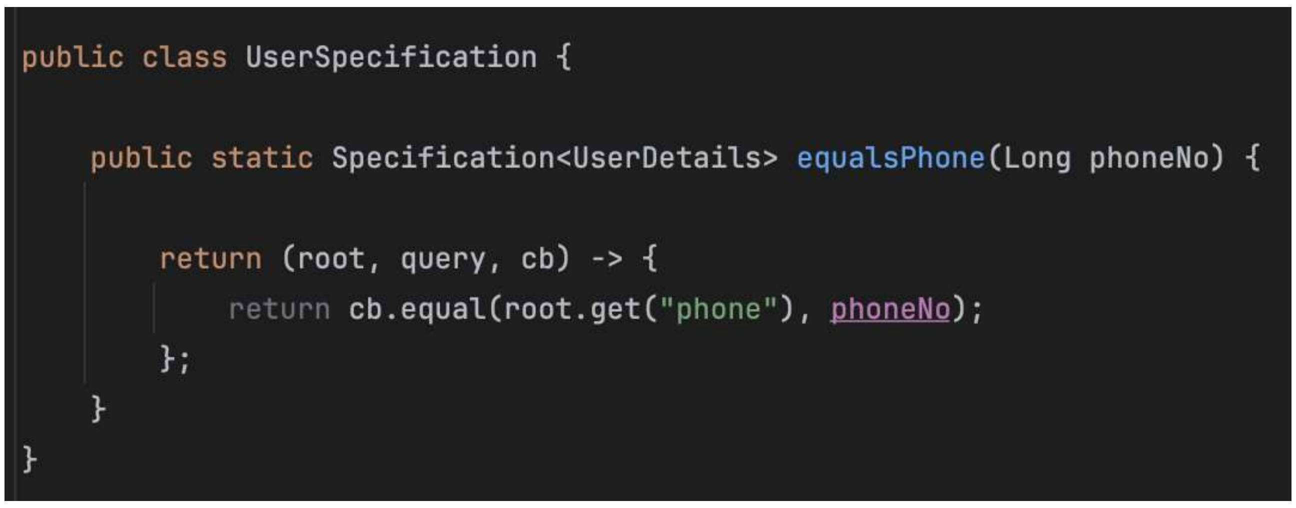
Possibl multipl too

Through Specification API, we can solve this:

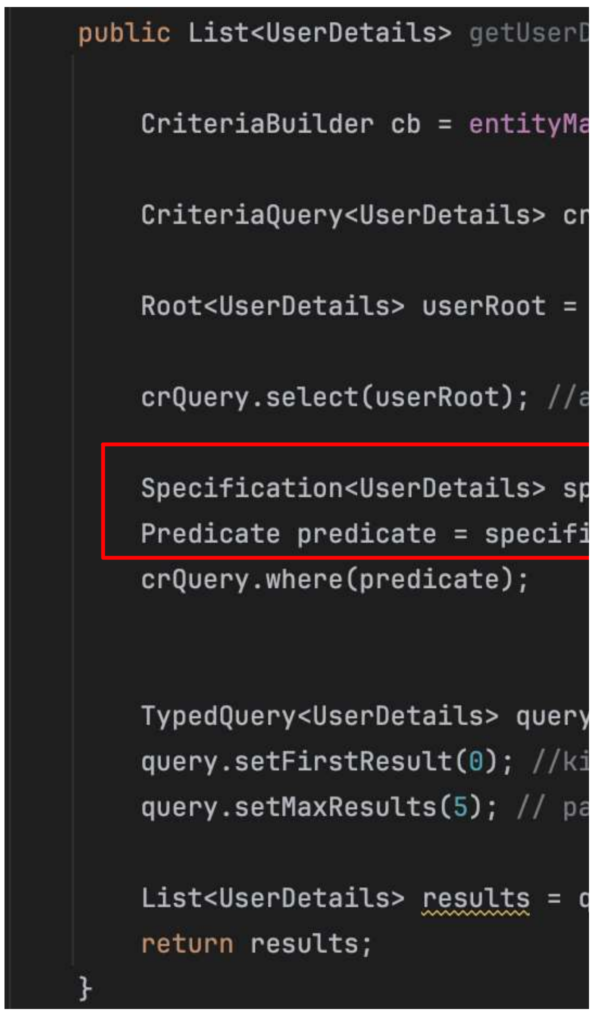
Specification Interface support following methods

Method	Description
toPredicate()	Abstract method, for which we need to provide implemen

and()	specf1.and(spec2)
or()	specf1.or(spec2)
not()	Specification.not(spec1)



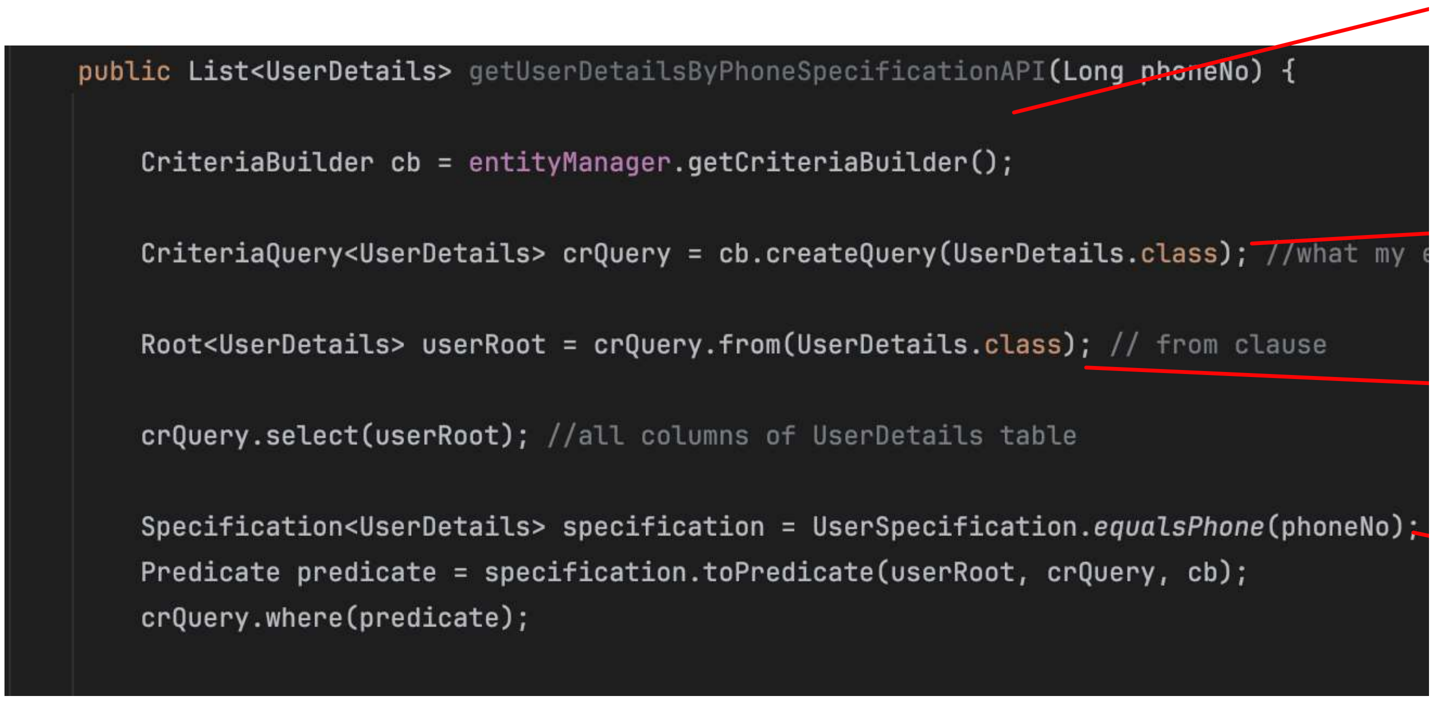
```
public class UserSpecification {

    public static Specification<UserDetails> equalsPhone(Long phoneNo) {

        return (root, query, cb) -> {
            return cb.equal(root.get("phone"), phoneNo);
        };
    }
}
```



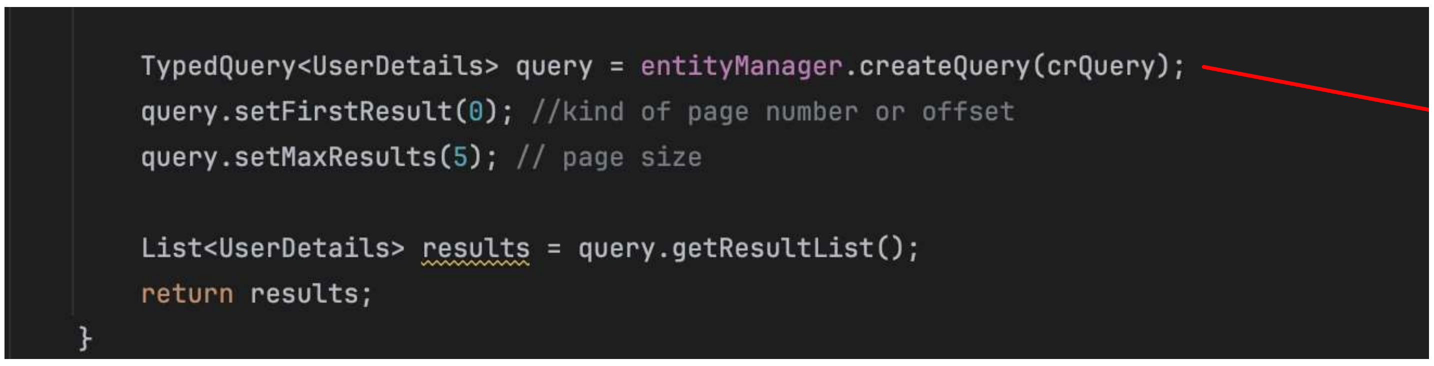
```
public List<UserDetails> getUserD

    CriteriaBuilder cb = entityMa

    CriteriaQuery<UserDetails> cr

    Root<UserDetails> userRoot =

    crQuery.select(userRoot); //a

    Specification<UserDetails> sp
    Predicate predicate = specifi
    crQuery.where(predicate);


    TypedQuery<UserDetails> query
    query.setFirstResult(0); //ki
    query.setMaxResults(5); // pa

    List<UserDetails> results = c
    return results;
}
```

2nd problem it solves is: ***CODE BOILERPLATE***

Even though we have taken out the predicate logic / filtering logic out, still the
present here



```
public List<UserDetails> getUserDetailsByPhoneSpecificationAPI(Long phoneNo) {

    CriteriaBuilder cb = entityManager.getCriteriaBuilder();

    CriteriaQuery<UserDetails> crQuery = cb.createQuery(UserDetails.class); //what my e

    Root<UserDetails> userRoot = crQuery.from(UserDetails.class); // from clause

    crQuery.select(userRoot); //all columns of UserDetails table

    Specification<UserDetails> specification = UserSpecification.equalsPhone(phoneNo);
    Predicate predicate = specification.toPredicate(userRoot, crQuery, cb);
    crQuery.where(predicate);
```



```
        TypedQuery<UserDetails> query = entityManager.createQuery(crQuery);
        query.setFirstResult(0); //kind of page number or offset
        query.setMaxResults(5); // page size

        List<UserDetails> results = query.getResultList();
        return results;
    }
```

All, we need to tell JPA that:

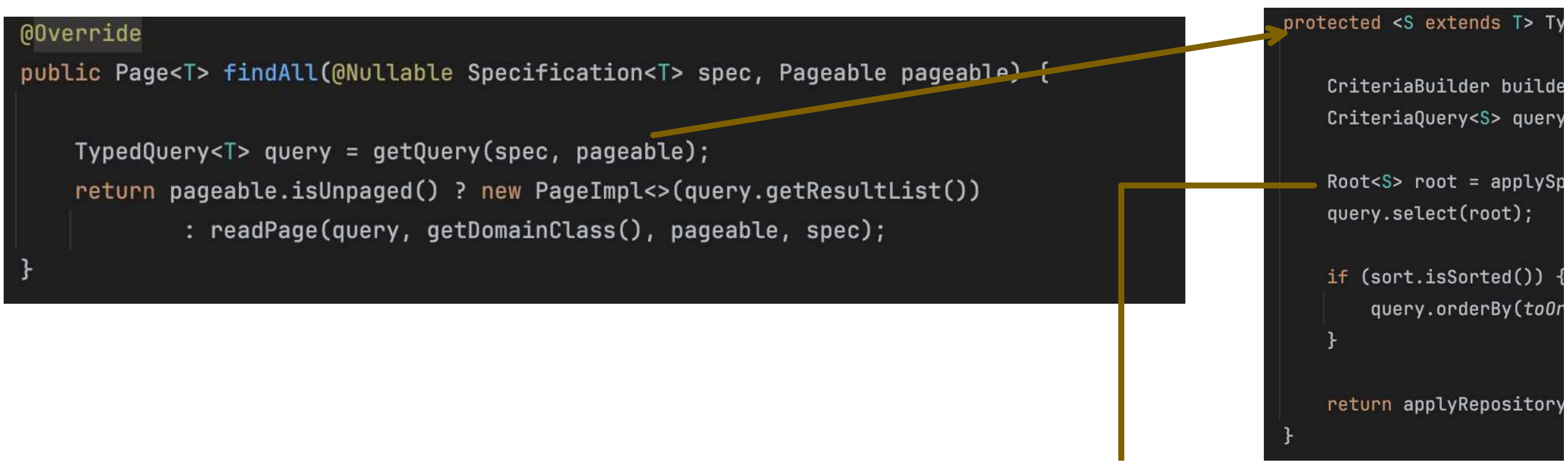
- From Which table we have to fetch the data, including joins
- What all columns
- Filtering in where clause

  that's it, JPA should take care of everything like object creation, query b

JpaSpecificationExecutor

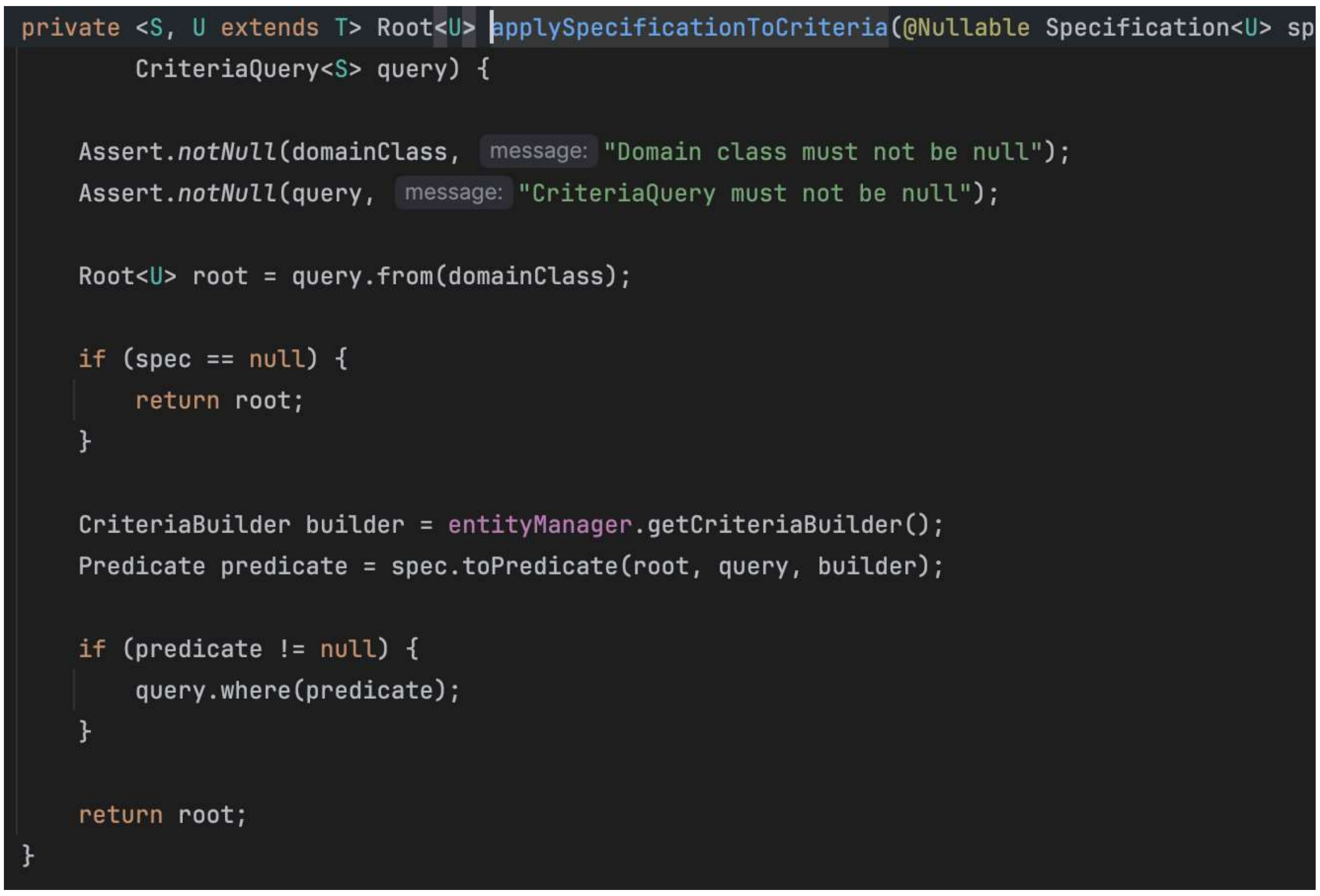
- Optional<T> findOne (Specification<T> spec);
- List<T> findAll (Specification<T> spec);
- Page<T> findAll (Specification<T> spec, Pageable pageab
- Boolean exists (Specification<T> spec);

JpaSpecificationExecutor Framework code

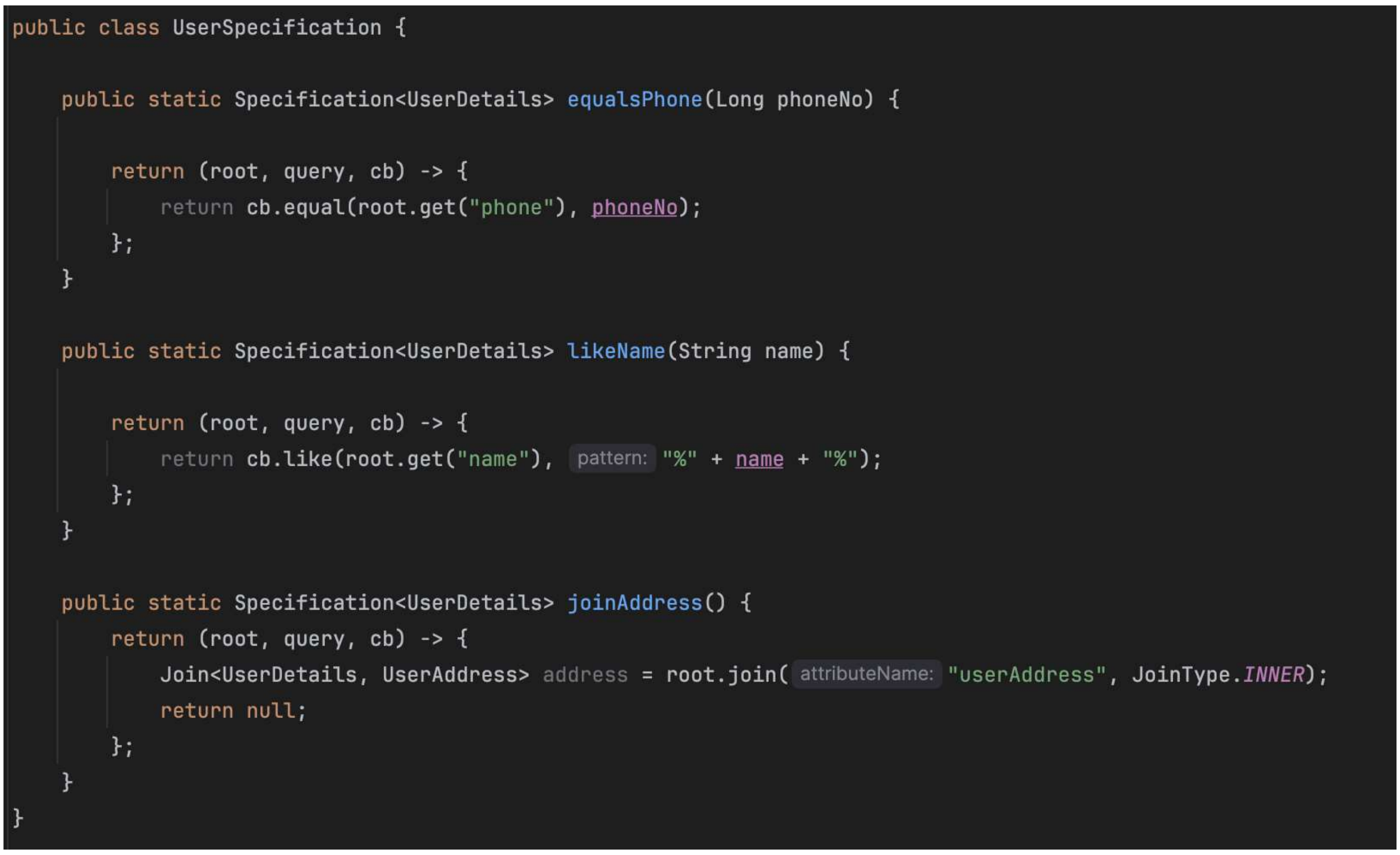


```
@Override
public Page<T> findAll(@Nullable Specification<T> spec, Pageable pageable) {

    TypedQuery<T> query = getQuery(spec, pageable);
    return pageable.isUnpaged() ? new PageImpl<>(query.getResultList())
            : readPage(query, getDomainClass(), pageable, spec);
}
```

```
protected <S extends T> Ty

    CriteriaBuilder builde
    CriteriaQuery<S> query

    Root<S> root = applySp
    query.select(root);

    if (sort.isSorted()) {
        query.orderBy(toOr
    }

    return applyRepository
}
```

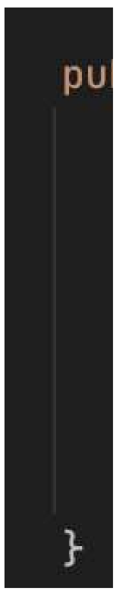


```java
private <S, U extends T> Root<U> applySpecificationToCriteria(@Nullable Specification<U> sp
        CriteriaQuery<S> query) {

    Assert.notNull(domainClass,  message: "Domain class must not be null");
    Assert.notNull(query,  message: "CriteriaQuery must not be null");

    Root<U> root = query.from(domainClass);

    if (spec == null) {
        return root;
    }

    CriteriaBuilder builder = entityManager.getCriteriaBuilder();
    Predicate predicate = spec.toPredicate(root, query, builder);

    if (predicate != null) {
        query.where(predicate);
    }

    return root;
}
```



```java
public class UserSpecification {

    public static Specification<UserDetails> equalsPhone(Long phoneNo) {

        return (root, query, cb) -> {
            return cb.equal(root.get("phone"), phoneNo);
        };
    }

    public static Specification<UserDetails> likeName(String name) {

        return (root, query, cb) -> {
            return cb.like(root.get("name"),  pattern: "%" + name + "%");
        };
    }

    public static Specification<UserDetails> joinAddress() {
        return (root, query, cb) -> {
            Join<UserDetails, UserAddress> address = root.join( attributeName: "userAddress", JoinType.INNER);
            return null;
        };
    }
}
```

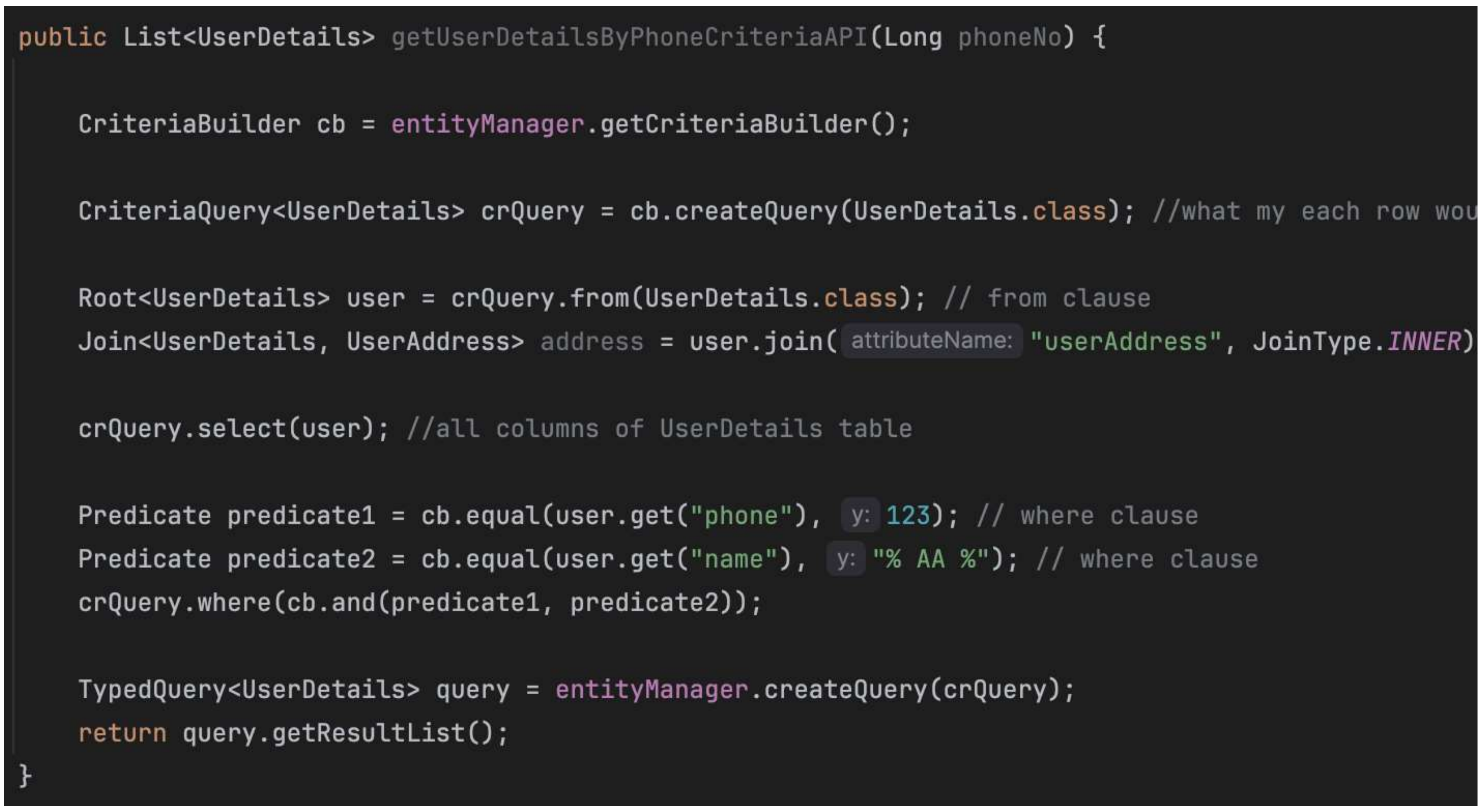
```
@Repo
publi

}
```



```
pul

}
```

Compa

API is r



```java
public List<UserDetails> getUserDetailsByPhoneCriteriaAPI(Long phoneNo) {

    CriteriaBuilder cb = entityManager.getCriteriaBuilder();

    CriteriaQuery<UserDetails> crQuery = cb.createQuery(UserDetails.class); //what my each row wou

    Root<UserDetails> user = crQuery.from(UserDetails.class); // from clause
    Join<UserDetails, UserAddress> address = user.join( attributeName: "userAddress", JoinType.INNER)

    crQuery.select(user); //all columns of UserDetails table

    Predicate predicate1 = cb.equal(user.get("phone"),  y: 123); // where clause
    Predicate predicate2 = cb.equal(user.get("name"),  y: "% AA %"); // where clause
    crQuery.where(cb.and(predicate1, predicate2));

    TypedQuery<UserDetails> query = entityManager.createQuery(crQuery);
    return query.getResultList();
}
```