**Hands-On 4: Difference between JPA, Hibernate, and Spring Data JPA:**

When working with Java applications that need to persist data in a database, developers often encounter three related concepts: JPA, Hibernate, and Spring Data JPA.  
**Java Persistence API (JPA):**

**What is it?**

* JPA is a **specification (JSR 338)** that defines a set of interfaces and rules for ORM (Object-Relational Mapping).
* It standardizes how Java objects are mapped to relational database tables.

**Key points:**

* Part of the Java EE (now Jakarta EE) standard.
* Defines *what to do*, but **does not provide implementation**.
* Needs an implementation (like Hibernate, EclipseLink) to actually work.

**Hibernate:**

**What is it?**

* Hibernate is a **concrete implementation** of the JPA specification.
* It is also an ORM framework on its own — it existed even before JPA and influenced JPA’s creation.

**Key points:**

* Provides additional features beyond JPA, like caching, filters, interceptors, and better SQL support.
* You can use Hibernate as a JPA provider (implementation) or directly with its own APIs.

**Spring Data JPA**

**What is it?**

* Spring Data JPA is a **higher-level abstraction** built on top of JPA.
* It is a Spring project that reduces boilerplate code required to implement repositories.
* It does not provide its own JPA implementation — it relies on an underlying JPA provider like Hibernate.

**Key points:**

* Eliminates writing common DAO/Repository code.
* Provides out-of-the-box CRUD operations, custom query methods, pagination, and more.
* Integrated seamlessly with Spring’s transaction management.

**Using Hibernate directly:**

public Integer addEmployee(Employee employee) {

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employeeID;

}

Here you:

* Manually open and close sessions.
* Begin and commit/rollback transactions explicitly.
* Write boilerplate code to persist objects.

**Using Spring Data JPA:**

**Repository Interface:**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**Service:**

@Autowired

private EmployeeRepository employeeRepository;

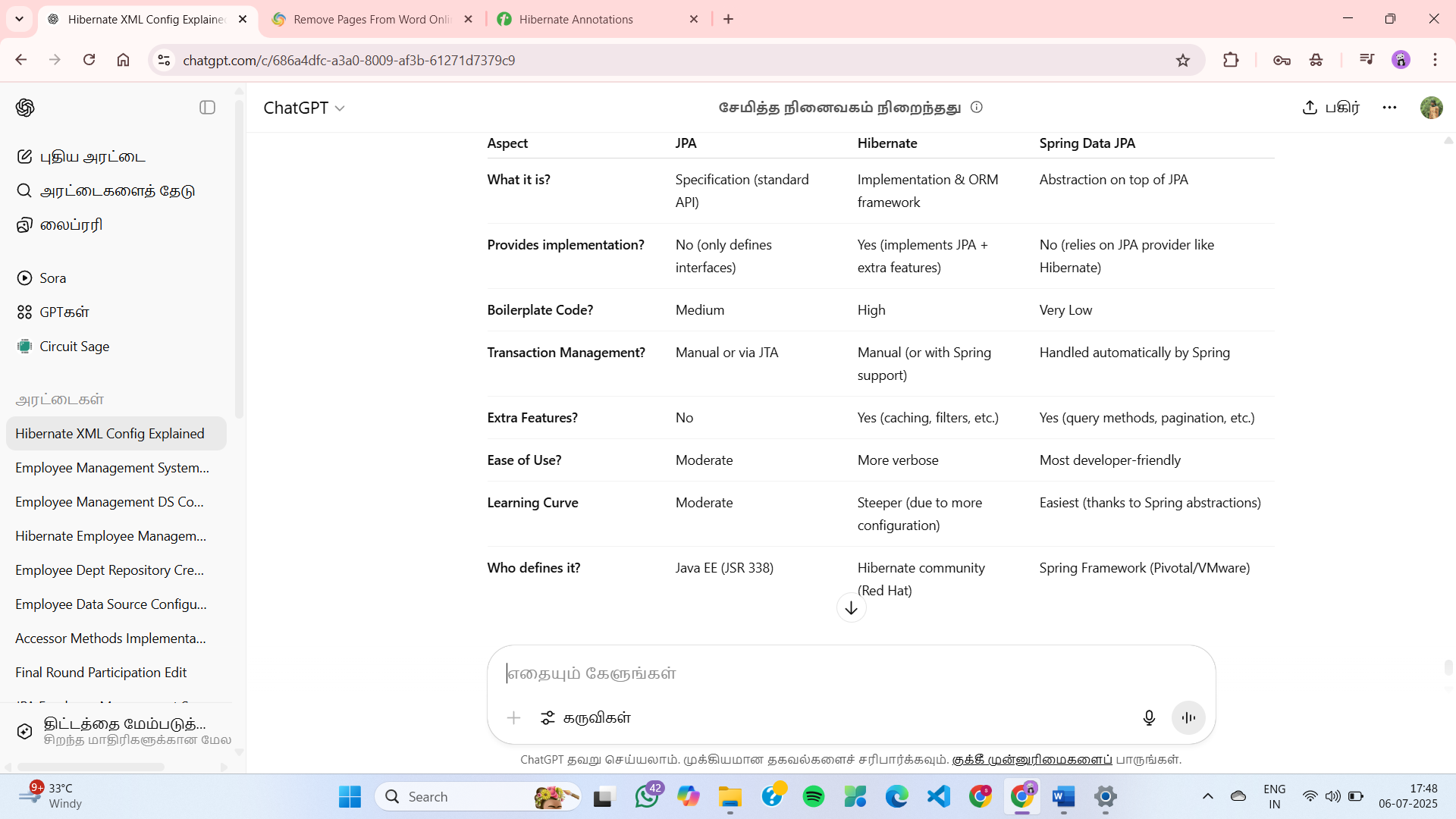
@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

Here you:

* Just extend JpaRepository, and Spring automatically implements CRUD methods for you.
* Use @Transactional to handle transactions automatically.
* Write almost no code to persist objects — **much cleaner and maintainable.**