📝 Time Constraints: Commitments and Trade-offs

🎯 Context

The development of the digital wallet service was carried out under a **tight deadline**, requiring strategic decisions to prioritize delivering a functional version of the system. Some features and improvements were simplified or postponed to future iterations.

⚖️ Commitments Made

1. In-Memory Database (H2)

- **Commitment**: We opted to use the **H2 Database** in in-memory mode instead of a complete relational database like PostgreSQL or MySQL, to simplify the setup and accelerate development.
- **Reason**: H2 is lightweight, easy to configure, and meets the main objective: providing local persistence for quick development and testing.
- **Impact**:
  - The database is **not persistent** between executions, making it unsuitable for production environments.
  - There might be inconsistencies when migrating to a production-grade database.

Planned Compensation:

- Plan the migration to a relational database like **PostgreSQL** or **MySQL**.
- Implement migration scripts using **Flyway** or **Liquibase**.

2. Basic Idempotency Key Persistence

- **Commitment**: Persistence of the Idempotency-Key was implemented directly in the **H2 database** using a simple table (idempotency_keys).
- **Reason**: This approach satisfies idempotency requirements in the short term without implementing advanced mechanisms (e.g., distributed caching).
- **Impact**:
  - **Scalability and performance limitations** in distributed environments.
  - The current solution does not support clusters or load balancers.

Planned Compensation:

- Introduce a **distributed cache** (e.g., Redis) to store idempotency keys, enabling scalability and improved performance.
- Add an **expiration policy** to prevent uncontrolled data growth.

3. Reduced Test Coverage

- **Commitment**: Due to time constraints, we prioritized **basic unit tests** only for critical components and services (e.g., wallet creation and idempotency validation).
- **Reason**: Ensure that the main logic is validated, even with reduced coverage.
- **Impact**:
  - Lack of integration and load testing might expose the application to unforeseen failures.
  - Edge cases and error scenarios are not fully covered.

Planned Compensation:

- Expand test coverage by adding:
  - **Integration tests** using **Spring Boot Test**.
  - **Load and performance testing** with **JMeter** or **Gatling**.
  - **Automated API tests** with **Postman Collections** or **Rest Assured**.

4. Basic Data Validation

- **Commitment**: Implemented limited validations for input parameters. Conditions like null or negative values were addressed, but no robust validation policies (e.g., valid userId) were added.
- **Reason**: Simplify initial development without compromising basic functionality.
- **Impact**:
    - Lack of rigorous validation might allow **invalid or malicious data** into the system.

Planned Compensation:

- Implement **Bean Validation** (e.g., @NotNull, @Positive) with **custom validators**.
- Add **global error handling** using **ControllerAdvice**.

5. Basic Swagger Documentation

- **Commitment**: Swagger documentation was implemented for **essential endpoints** (wallet creation). Additional endpoints, like querying wallets, are not yet documented.
- **Reason**: Prioritize documenting the core functionality of the system.
- **Impact**:
    - Incomplete documentation may hinder integration with other systems.

Planned Compensation:

- Complete the documentation for all existing endpoints and parameters.
- Add **examples of responses** and **error statuses** in Swagger.

📊 Summary of Commitments and Trade-offs

Area	Commitment	Impact	Planned Compensation
Database	Use of H2 in-memory	No persistence between executions	Migrate to PostgreSQL with migration scripts
Idempotency Persistence	Simple table in H2	Limited scalability in distributed environments	Use Redis for distributed caching
Test Coverage	Limited to unit tests	Scenarios not fully validated	Add integration and load tests
Data Validation	Basic validations	Possible invalid input	Implement Bean Validation and ControllerAdvice
Swagger Documentation	Basic documentation of endpoints	Incomplete documentation	Expand documentation with full examples

🚀 Conclusion

The decisions made during the development of the project meet the **main functional requirements**, ensuring a functional **MVP** within the stipulated timeline. However, simplified solutions were necessary in areas such as persistence, validation, and testing.

These limitations have been documented and include **planned compensations** for future iterations, prioritizing **scalability, robustness, and quality**.

With **Swagger documentation** and functional idempotency, the system is ready for delivery and provides a solid foundation for continuous improvements. 🚀