Université Abdelmalek Essaadi
Faculté ses Sciences et techniques de Tanger
Département Génie Informatique
Cycle Master : SIM
Applications distribuées
Pr . ELAACHAk LOTFI

Mise en place d’une application distribuée JEE

Setting up a distributed JEE application

Réaliser Par : ELHoumaini Karim / P139481476

Summary

Project Name: EJB Web Application for Managing Etudiant Information

Project Description: This project involves the development of a web application for managing Etudiant (student) information. The application is divided into two main components: a backend EJB (Enterprise JavaBeans) application responsible for managing Etudiant entities, and a frontend web application that communicates with the EJBs using servlets.

Project Components:

1. Backend EJB Application

- **Entity Bean (Etudiant):** The project defines an Entity Bean named "Etudiant" to represent student information. This entity is managed by the EJB container and is responsible for database persistence.
- **Session Bean (EtudiantServiceBean):** The project includes a Session Bean named "EtudiantServiceBean" that provides business logic and services for creating, updating, deleting, and retrieving Etudiant entities. It implements the remote interface "EtudiantServiceRemote."

- **Remote Interface (EtudiantServiceRemote):** The "EtudiantServiceRemote" interface specifies the remote methods that can be called by clients to interact with the Etudiant service.
- **Data Source Configuration:** A MySQL database is used to store Etudiant data. The project establishes a data source in WildFly, which includes the JDBC driver configuration and connection pool settings. This data source is used to connect to the MySQL database.

2. Frontend Web Application:

- **Communication Class (EtudiantServiceClient):** The frontend project includes a communication class named "EtudiantServiceClient." This class facilitates remote communication with the EJBs using the "EtudiantServiceRemote" interface. It uses JNDI to look up and invoke EJB methods.
- **Servlet Classes (CreateEtudiantServlet, UpdateEtudiantServlet, DeleteEtudiantServlet, GetEtudiantServlet):** The project defines four servlet classes that act as endpoints for client requests. Each servlet corresponds to a specific method in the "EtudiantServiceRemote" interface. The servlets handle HTTP requests (e.g., POST, PUT, DELETE, GET) and communicate with the EJB backend to perform CRUD operations on Etudiant entities.

Project Goals: The primary goal of this project is to provide a web-based system for managing student information. It allows clients to create, update, delete, and retrieve student records using the EJB backend, while ensuring secure and efficient communication.

Technologies Used: Java EE (Enterprise Edition), EJB, Servlets, WildFly (JBoss), MySQL, JDBC, JNDI.

Outcome: The project results in a robust and scalable web application that allows efficient management of Etudiant data. Users can interact with the application through well-defined servlets, which communicate with the EJB backend, ensuring data integrity and security.

Future Enhancements: In future iterations, you can consider adding features like authentication, validation, and front-end improvements for a more user-friendly experience.

This summary provides an overview of the main components, technologies, and goals of your project. It outlines the high-level structure and functionality of your EJB-based web application for managing student data.

Introduction:

In an era driven by information technology, the efficient management of data and resources plays a pivotal role in the success of organizations, institutions, and enterprises. In the context of educational institutions, managing student information is a crucial aspect of ensuring smooth administrative processes and providing a seamless experience for both students and faculty. This report sheds light on the development of an EJB (Enterprise JavaBeans) based web application aimed at facilitating the management of student information, referred to as "Etudiants" (students).

The project encompasses two key components. The first is a backend EJB application, responsible for storing, retrieving, updating, and deleting student information. This application leverages the power of EJBs to provide the necessary business logic and data access. It includes an Entity Bean to represent student data, a Session Bean to serve as the business logic layer, and a Remote Interface defining the methods accessible to clients.

The second component is the frontend web application, which allows users to interact with the EJB-based backend. This interaction is facilitated through a set of servlets, each of which corresponds to a specific operation on student records, namely creating, updating, deleting, and retrieving. These servlets enable the application to respond to HTTP requests from clients, process the requests, and communicate with the EJB backend to perform the desired actions.

Furthermore, the report also discusses the essential configuration steps, including the establishment of a data source for MySQL, which acts as the database for storing student information. User management in WildFly is configured to ensure the security of the application, granting appropriate access rights to users based on their roles and responsibilities.

The primary objective of this project is to provide a robust and scalable solution for managing student information in an educational context. It encapsulates the essence of Java EE (Enterprise Edition) and EJBs, illustrating the power of these technologies in simplifying data management tasks while ensuring security and performance.

In conclusion, the development of this EJB-based web application serves as an excellent example of how modern technology can streamline the management of student information, ultimately contributing to more efficient educational processes. The report is structured to guide readers through the project's key aspects, from its

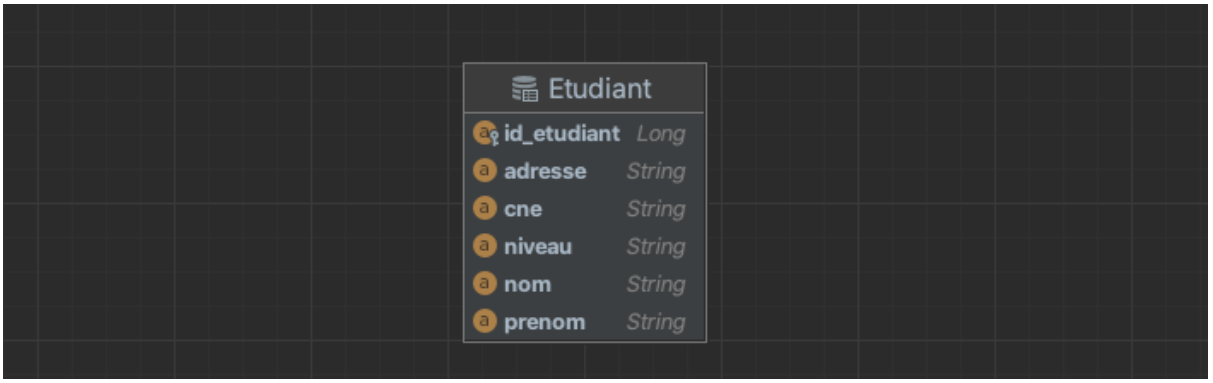
architecture to implementation details and further possibilities for enhancements and refinements.

Backend EJB Application

The backend EJB (Enterprise JavaBeans) application is the heart of our student information management system. It serves as the foundational component responsible for storing, retrieving, updating, and deleting student data. The application is designed to encapsulate business logic and data access, making use of EJBs, which are well-suited for creating robust, distributed, and scalable enterprise applications.

Entity Class: Etudiant

At the core of the backend application is the `Etudiant` entity class, which represents student information. The entity class is annotated with EJB 4 specifications and JPA (Java Persistence API) annotations, allowing it to be seamlessly integrated with the application's data source and manipulated as a Java object. The `Etudiant` entity class typically consists of attributes such as student ID, name, email, and other relevant details. The JPA annotations define the mapping of these attributes to corresponding columns in the underlying database.



Session Bean: EtudiantServiceBean

The business logic layer of the application is implemented through a Session Bean called `EtudiantServiceBean`. This bean contains methods that allow clients to perform various operations on student records. In the context of EJB, session beans are used to encapsulate business logic and manage the application's state. The `EtudiantServiceBean` is annotated as a stateless session bean, which makes it suitable for handling multiple client requests in a stateless and efficient manner.

Remote Interface: EtudiantServiceRemote

To make the business logic of the `EtudiantServiceBean` accessible to clients, a remote interface named `EtudiantServiceRemote` is defined. The remote interface specifies the methods that can be invoked remotely by clients, allowing for the execution of CRUD (Create, Read, Update, Delete) operations on student records.

Java Transaction API (JTA) and DataSource Setup

In our backend EJB application, we employ the Java Transaction API (JTA) and configure a DataSource to manage database transactions and connections efficiently. JTA provides the necessary infrastructure for distributed transaction management, ensuring data integrity and consistency when multiple operations are involved.

Setting up JTA and DataSource:

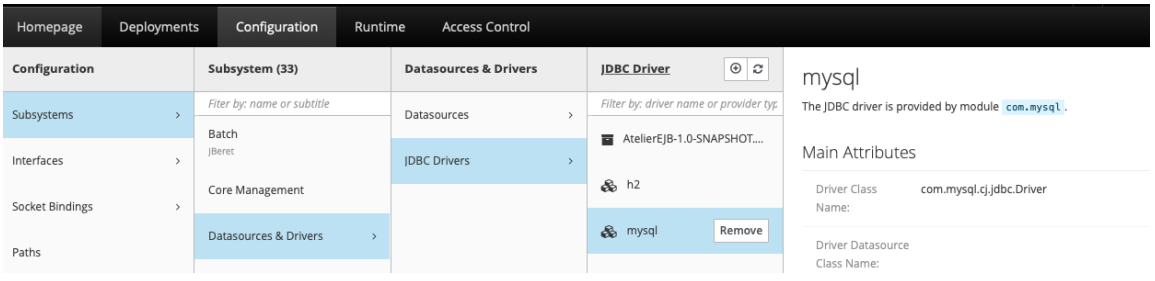
1. **Creating the MySQL JDBC Driver Module**:
    - To enable Wildfly to communicate with a MySQL database, we first create a module for the MySQL JDBC driver.

- We created a folder named `mysql` in the Wildfly `modules` directory (for example, `wildfly/modules/system/layers/base/com/mysql/main`).
- Inside the `main` subfolder, we placed the MySQL JDBC driver JAR file.
- Alongside the JAR file, we created a `module.xml` file with the following content:

```
<module xmlns="urn:jboss:module:1.5" name="com.mysql">
    <resources>
        <resource-root path="mysql-connector-j-8.0.33.jar" />
    </resources>
    <dependencies>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
    </dependencies>
</module>
```

module.xml

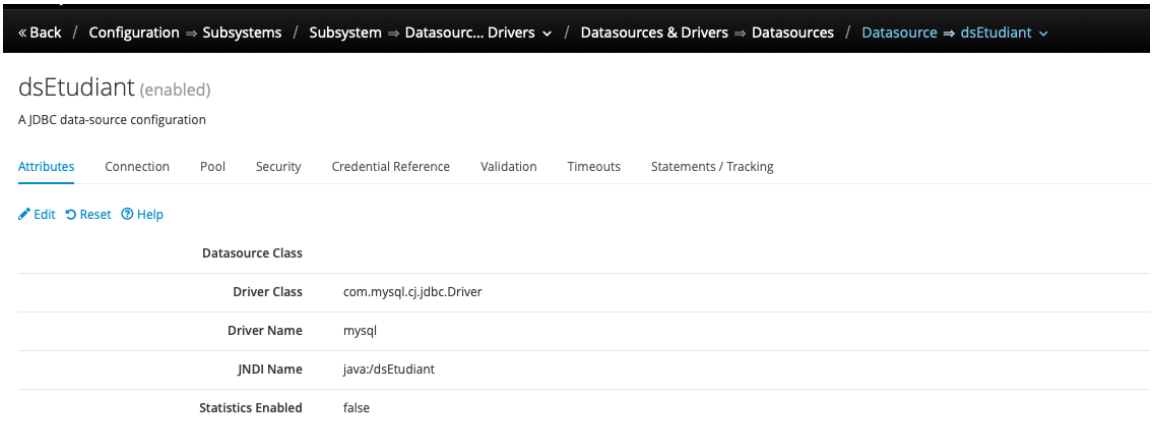
2. **Creating an Admin User**:
   - To manage the Wildfly application server and configure data sources, we created an admin user with the necessary permissions.
   - We logged into the Wildfly Management Console (usually accessible at `http://localhost:9990` or another configured URL)
3. **Creating the JDBC Driver**:
   - In the Wildfly Management Console, we created a new JDBC driver using the MySQL driver module we defined earlier.
   - We specified the driver class name and the driver module name .



4. **Setting up the Datasource**:
   - After configuring the JDBC driver, we proceeded to set up the DataSource.
   - In the Wildfly Management Console, under the "Datasources" section, we created a new DataSource using the MySQL JDBC driver.

- We provided the necessary connection URL, username, and password to access the MySQL database.

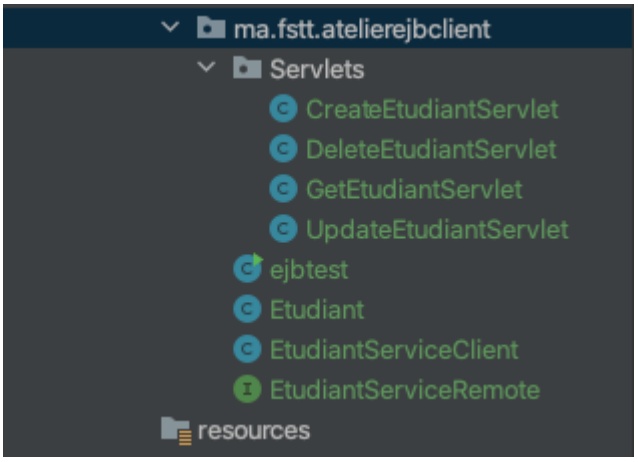
5. **Linking the DataSource to the Application**:
   - After setting up the DataSource, we ensured that the EJB application is aware of the JNDI name for the DataSource.
   - We configured the EJB application to look up the DataSource using JNDI, allowing it to establish database connections for performing CRUD operations on student records.



By following these steps, we established a robust infrastructure for managing database transactions, allowing the EJB application to interact with the MySQL database efficiently while ensuring data integrity and reliability through JTA. The defined DataSource serves as the bridge between the EJB application and the database, enabling seamless database operations.

Client EJB Application

The client project is responsible for interacting with the backend EJB application, specifically the `EtudiantService` EJB, and exposing this functionality through servlets. This section provides an overview of the key components in the client project.



1. `Etudiant` Class:

The `Etudiant` class represents an entity in the client project, typically matching the entity in the backend EJB application. It encapsulates data related to students. Properties like `etudiantId`, `nom`, `prenom`, `cne`, etc., allow the client to work with student records.

2. `EtudiantServiceRemote` Interface:

The `EtudiantServiceRemote` interface defines the remote service contract for working with student records. It typically mirrors the remote interface defined in the backend EJB application. This interface declares the methods that can be invoked remotely on the EJB, including operations like creating, updating, deleting, fetching, and listing student records.

3. `EtudiantServiceClient` Class:

The `EtudiantServiceClient` class plays a crucial role in establishing communication with the backend EJB application. It serves as the client-side counterpart of the remote interface `EtudiantServiceRemote` defined in the backend EJB application.

- This class is responsible for creating the EJB context and looking up the remote EJB instance via JNDI. It allows the client to access EJB methods on the server-side.
- The `EtudiantServiceClient` class encapsulates the logic for performing JNDI lookups and invoking methods on the remote EJB. It acts as a bridge between the client application's servlets and the remote EJB methods.

- When a servlet in the client project needs to interact with the EJB to perform operations on student records, it creates an instance of `EtudiantServiceClient`. The servlet can then use this client to access the EJB's functionality, such as creating, updating, deleting, or retrieving student records.

4. `ejbtest` Class - JNDI Communication:

The `ejbtest` class serves as a key component for JNDI communication between the client and the backend EJB application. It is typically a standalone Java class or a part of the client project, responsible for initializing the JNDI context, looking up the remote EJB instance, and invoking methods on the EJB. The `ejbtest` clas contains the following components:

- JNDI context initialization: It creates an initial context that allows the client to look up EJB instances.
- JNDI lookup: The class performs JNDI lookups to locate the `EtudiantServiceRemote` EJB instance. This lookup is based on the JNDI name associated with the EJB in the backend EJB application.

5. Servlets:

The client project contains a set of servlets that provide HTTP endpoints for client applications, enabling them to interact with the EJB application. These servlets handle incoming HTTP requests and communicate with the backend EJB application via `EtudiantServiceClient`.

- **CreateEtudiantServlet**: This servlet is responsible for processing HTTP requests to create new student records. It takes input data from clients, such as student information, and sends it to the EJB application for insertion into the database.
- **UpdateEtudiantServlet**: Handles requests for updating existing student records. Clients provide the updated data, and the servlet communicates with the EJB application to modify the corresponding records.
- **DeleteEtudiantServlet**: Manages requests to delete student records. It receives student IDs to identify the records to be removed and communicates with the EJB application to perform the deletion.
- **GetEtudiantServlet**: Processes requests for fetching specific student records by their unique IDs. The servlet interacts with the EJB application to retrieve the

requested data.

- **ListEtudiantsServlet**: Provides endpoints for listing all student records. Clients can use this servlet to retrieve a complete list of student records from the EJB application.

In summary, the `EtudiantServiceClient` class bridges the client and the backend EJB application by allowing JNDI communication. The `ejbtest` class is responsible for initializing JNDI and invoking EJB methods. The servlets in the client project expose HTTP endpoints for performing create, update, delete, retrieve, and list operations on student records, effectively communicating with the EJB application through `EtudiantServiceClient`. This architecture allows the client to interact seamlessly with the EJB application hosted on the Wildfly server.

Conclusion

Throughout this project, our primary objective was to construct a distributed Java EE application, harnessing the capabilities of Enterprise JavaBeans (EJB) technology and the Wildfly application server. We set out to develop a comprehensive solution comprising a backend EJB application, which served as the epicenter for our business logic, and a client project that effortlessly communicated with these EJBs via the Java Naming and Directory Interface (JNDI).

Our journey began by establishing the foundations of our backend EJB application. The fundamental building block was the `Etudiant` entity class, which underpinned the core structure of our application. We introduced stateless session beans as the workhorses of our business logic, meticulously handling operations such as entity creation, retrieval, modification, and deletion for `Etudiant` entities. Our robust design extended to configuring a `DataSource`, accomplished by setting up a MySQL connector with the aid of the Java Transaction API (JTA). This `DataSource` was pivotal, serving as the vital bridge connecting our application to the MySQL database.

JTA, an integral component of our project, played a crucial role in ensuring the consistency and reliability of database transactions. This critical capability was central to maintaining data integrity and enabling seamless recovery in case of unforeseen failures. We systematically outlined the steps involved in creating the `DataSource`, from the configuration of the MySQL connector module to the definition of the JDBC driver, culminating in the successful establishment of a link to our

MySQL database. This process was instrumental in enabling our EJB application to interact seamlessly with the database.

In the client project, we introduced the `EtudiantServiceRemote` interface. This interface defined the contract for interacting with the backend EJB application, acting as the intermediary that facilitated communication between the client and the EJBs. To ensure this communication was seamless, we introduced the `Communication` class, responsible for establishing a JNDI connection with the EJBs, enabling access to and utilization of the services offered by the EJB application.

With a strong foundation laid, we proceeded to develop a suite of servlets within the client project. Each servlet corresponded to one of the key methods outlined in the `EtudiantServiceRemote` interface. These servlets served as the bridge between the user interface and the EJB application, ensuring a user-friendly web interface and seamless communication with the backend EJBs.

To complete the client project, we introduced the `etudiantserviceClient` class, which illustrated how to initialize and employ the EJB client. It offered insights into performing JNDI lookups to access remote EJB interfaces and invoke methods effectively.

In summary, this project revolved around the creation of a distributed Java EE application. Our journey encompassed the development of a comprehensive solution, effectively bridging the gap between the backend EJB application and the client project. The utilization of Wildfly as our application server streamlined the development and deployment process, delivering a seamless and efficient development lifecycle.

In conclusion, this project not only showcased the capabilities and potential of Java EE, EJB, JNDI, and Wildfly, but it also exemplified the creation of a robust, scalable, and distributed enterprise application. By effectively connecting the client project with a powerful backend application, we demonstrated the creation of an integrated and efficient solution for data management and manipulation. This journey illuminated the versatility and power of Java EE and its practical application in real-world scenarios. It represents a significant stride in our endeavor to master Java EE and enterprise-level application development.