Set up the development environment:
Install Node.js on your machine.
Set up Jenkins for continuous integration.
Set up the Angular application.
Create a new Spring Boot project:
Use your preferred IDE (e.g., IntelliJ, Eclipse) to create a new Spring Boot project.
Set up the project with Maven or Gradle as the build tool.
Add Spring Security dependencies:
Update the project's `pom.xml` or `build.gradle` file to include the necessary dependencies for Spring Security.
Configure Spring Security:
Create a new configuration class that extends `WebSecurityConfigurerAdapter`.
Override the `configure()` method to customize the security configuration.
Implement authentication logic by extending `AbstractUserDetailsAuthenticationProvider`.
Implement the authentication logic:
Create a new class that implements the `AuthenticationProvider` interface.
Override the `authenticate()` method to perform the authentication logic.
Use the Node.js backend for authentication, such as verifying credentials against a database or an external service.
Implement the necessary logic to handle successful or failed authentication attempts.
Integrate Angular application with Spring Security:
Update the Angular application to include login and authentication functionality.
Use Angular's HTTP client to send authentication requests to the Spring Security backend.
Handle the response from the backend to display appropriate feedback to the user.
Set up Jenkins for continuous integration:
Configure Jenkins to build and deploy the Spring Security application automatically.
Set up a Jenkins pipeline or job that listens for changes in the GitHub repository.
Configure Jenkins to trigger a build whenever changes are pushed to the repository.
Track source code using GitHub repositories:
Create a new GitHub repository to track the source code for your project.
Initialize a Git repository locally and add the remote repository as a remote origin.
Add the necessary files and directories to be tracked by Git.
Create a `.gitignore` file to exclude files that should not be pushed to the repository, such as IDE-specific files or build artifacts.
Push code to the GitHub repository:
Commit your changes locally with descriptive commit messages.

Push your changes to the remote GitHub repository.
Document the repository link:
Once the code is pushed to the GitHub repository, document the repository link for submission.

This project aims to enhance the flexibility of building an Authentication Provider in Spring Security by incorporating Node.js, Jenkins, and an Angular Application. The objective is to develop a robust authentication system that integrates with an Angular frontend, uses Node.js for authentication logic, and employs continuous integration with Jenkins for seamless development workflow.
To begin, the development environment needs to be set up. This involves installing Node.js, configuring Jenkins for continuous integration, and setting up the Angular application.
Next, a new Spring Boot project is created using an IDE like IntelliJ or Eclipse, and the necessary Spring Security dependencies are added. These dependencies will enable the implementation of secure authentication mechanisms.
Spring Security is then configured by creating a new configuration class that extends `WebSecurityConfigurerAdapter`. This class allows customization of security configurations and serves as the entry point for authentication logic. The authentication logic is implemented by extending `AbstractUserDetailsAuthenticationProvider` and creating a class that implements the `AuthenticationProvider` interface. The Node.js backend is utilized to perform the actual authentication, which may involve verifying credentials against a database or an external service. The logic to handle successful or failed authentication attempts is also implemented.
The Angular application is integrated with Spring Security by adding login and authentication functionality. The Angular HTTP client is used to send authentication requests to the Spring Security backend, and appropriate feedback is displayed to the user based on the response received.
To ensure version control and collaboration, the source code is tracked using GitHub repositories. A new repository is created, and the necessary files and directories are added to be tracked by Git. A `.gitignore` file is created to exclude files that should not be pushed to the repository, such as IDE-specific files or build artifacts.
Once the development and testing are complete, the code is pushed to the GitHub repository. This involves committing the changes locally with descriptive commit messages and pushing the changes to the remote repository.
Lastly, the repository link is documented for submission. This link serves as a reference to access the codebase and track the project's progress.

It is important to note that the above description provides a general overview of the project objectives and steps involved. The specific implementation details may vary depending on the requirements and preferences of the development team.