WriteUp For the project:

Objective:

As a developer, build Authentication Provider in Spring Security.

Steps/Description:

The objective of this project is to build an Authentication Provider in Spring Security that provides more flexibility than the standard scenario. The project utilizes the following technologies: Node.js, Jenkins, and an Angular application.

- Spring Boot Application: The core of the project is a Spring Boot application that handles authentication and user management using Spring Security. The application includes a custom authentication provider (CustomAuthenticationProvider) that implements the AuthenticationProvider interface to perform authentication logic. It interacts with a user repository (UserRepository) to retrieve user information and validate credentials.

- JSP Pages: The project includes JSP (JavaServer Pages) files to create the user interface. The login.jsp page provides a login form for users to enter their credentials. The home.jsp page is the landing page after successful authentication, displaying a welcome message. The users.jsp page lists all the users stored in the system, and the update-user.jsp page allows users to update their information.

- User Management: The project supports user management functionality, allowing users to view a list of all users (/users endpoint) and update their information (/users/{id}/edit endpoint). The UserController class handles these requests, interacting with the UserRepository to fetch and update user data.

- Continuous Integration with Jenkins: Jenkins, a popular continuous integration and delivery tool, is used to automate the build and testing process. A Jenkins job is configured to build and test the Spring Boot application whenever changes are pushed to the GitHub repository.

Algorithm:

1. Create a new Spring Boot project using Spring Initializr with the required dependencies: Spring Web, Spring Security, and Spring Data JPA.

2. Implement a custom authentication provider (CustomAuthenticationProvider) that implements the AuthenticationProvider interface.
3. Configure Spring Security in the SecurityConfig class, specifying the login page, success URL, and permit access to certain endpoints.
4. Create JSP pages for login, home, users list, and update user functionality.
5. Create a User entity class and a corresponding UserRepository interface to store and retrieve user information.
6. Implement a UserDetailsServiceImpl class that implements the UserDetailsService interface to load user details from the UserRepository.
7. Create frontend jsp pages with using bootstrap and css.
8. Create a UserController class to handle user-related requests, such as retrieving the users list and updating user information.
9. Set up Jenkins for continuous integration, configuring a job to build and test the Spring Boot application when changes are pushed to the GitHub repository.
10. Track the project source code on a GitHub repository and push the code to the repository.