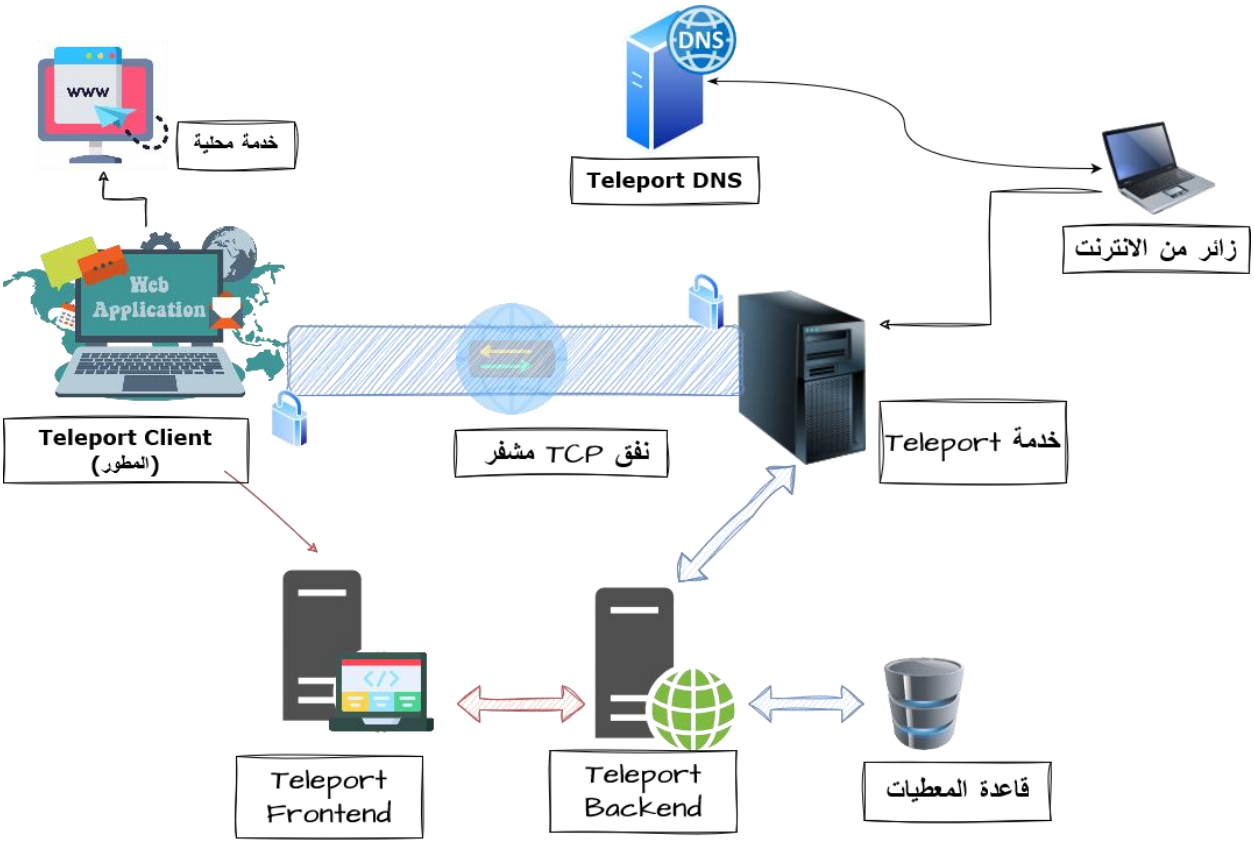


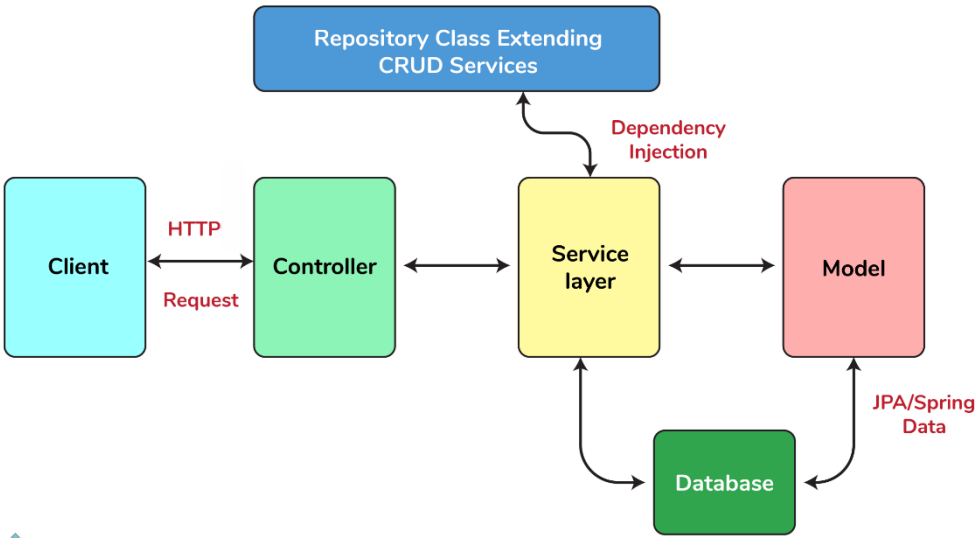
هذا الجزء خاص بالخادم الخلفي في نظام Teleport الظاهر في الصورة السابقة باسم TeleportBackend

شرح التصميم والتنفيذ موجود في التقرير 😈
التصميم في الفقرة 4.2.6-

التنجيز في الفقرة 3.7

4.2.6– النمط التصميمي المستخدم في تصميم Backend

في تصميم الـ Backend لنظام Teleport، تم الاعتماد على نمط Maven في إطار العمل Spring Boot، وهو ما يضمن تنظيماً واضحاً للمشروع وتسهيلاً لإدارة التبعيات وبناء التطبيق. يتبع النظام تصميماً متعدد الطبقات (Multi-Layered Architecture) يتضمن ثلاث طبقات رئيسية:



الشكل6. 1 Spring Boot Flow Architecture

طبقة الـ Controller: تعمل هذه الطبقة كوسيط بين واجهة المستخدم وباقي أجزاء النظام، حيث تستقبل الطلبات من العميل (Client) وتقوم بتمريرها إلى الخدمات المعنية (Services). وهي المسؤولة عن معالجة البيانات وإرجاع النتائج المناسبة.

طبقة الخدمات (Service Layer): تحتوي على منطق الأعمال (Business Logic) وتنسق بين طبقة التحكم وطبقة الوصول إلى البيانات. (Repository Layer) هذه الطبقة هي المسؤولة عن تنفيذ العمليات المختلفة على البيانات وفقاً لمتطلبات النظام.

طبقة الوصول إلى البيانات (Repository Layer) : تُعدّ هذه الطبقة مخصصة للتعامل مع قاعدة البيانات، وتوفر واجهة محددة للتفاعل مع البيانات المخزنة في MySQL. من خلال هذه الطبقة، يتم التحكم في الوصول إلى البيانات بشكل صارم، مما يضمن حماية البيانات وسلامتها.

تم أيضاً استخدام تقنية JWT (JSON Web Token) لضمان أمان الاتصالات داخل النظام. تُستخدم JWT لتوفير آلية فعّالة للتحقق من هوية المستخدمين وتأكيد صلاحياتهم عند محاولة الوصول إلى موارد معينة في النظام. بفضل هذه التقنية.

يمكن للنظام التأكد من صحة هوية المستخدم والحفاظ على أمان الجلسات (Sessions) بشكل فعال، مما يعزز من أمان وسرية المعلومات المتداولة في النظام.

3.7Backend- الخادم الخلفي

سنتحدث في هذه الفقرة عن هيكيلية بناء وتنجيز الخادم الخلفي لمظام Teleport.

3.3.7- إنشاء المخدم والاتصال مع قاعدة المعطيات:

تم تطوير المخدم الخلفي باستخدام Spring Boot وبيئة Maven، بهدف بناء نظام متكامل يتميز بالقوة والمرونة. في البداية، تم إنشاء قاعدة بيانات MySQL مخصصة لتخزين البيانات المتعلقة بالتطبيق. بعدها، تم تكوين إعدادات الاتصال بقاعدة البيانات داخل ملف التكوين application.yml. تم اعتماد الربط مع قاعدة البيانات بناءً على المتغيرات البيئية كما يظهر في الشكل8.7 ، مما يوفر مرونة عالية عند نشر التطبيق في بيئات مختلفة. بفضل هذه الطريقة، يمكن تعديل معلمات الاتصال بقاعدة البيانات مثل عنوان الـ URL، واسم المستخدم، وكلمة المرور، دون الحاجة إلى تعديل الكود البرمجي نفسه. هذه المتغيرات البيئية تتيح التكيف مع بيئات التشغيل المختلفة بسهولة، سواء كانت بيئة تطوير، اختبار، أو إنتاج.

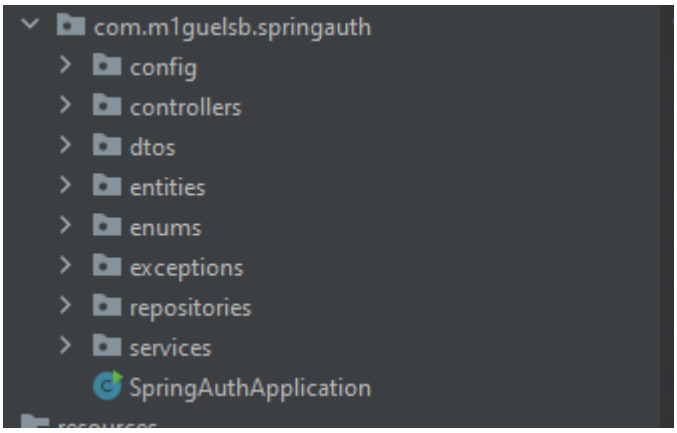
في ملف application.yml، تم ضبط إعدادات الاتصال بقاعدة البيانات كالتالي:

- **عنوان قاعدة البيانات**: jdbc:mysql://localhost:3306/teleport_db ، حيث يتم تخصيص عنوان URLلقاعدة البيانات بحيث ان لم تتوفر المتغيرات البيئية يتم اتباع هذا الرابط.
- **اسم المستخدم وكلمة المرور**: تم توفير بيانات الاعتماد المطلوبة لقاعدة البيانات من خلال المتغيرات البيئية وكذلك قيم ثابتة في حال عدم توفر متحولات بيئية.
- **إعدادات Hibernate**: تم تفعيل خاصية ORM لتحويل الكيانات إلى جداول في قاعدة البيانات.

```yaml
security:
  jwt:
    token:
      secret-key: ${JWT_SECRET}

server:
  port: ${DATASOURCE_PORT:9090}
  error:
    include-stacktrace: never

spring:
  application:
    name: authentication
  datasource:
    url: ${DATASOURCE_URL:jdbc:mysql://localhost:3306/teleport_db}
    username: ${DATASOURCE_USERNAME:root}
    password: ${DATASOURCE_PASSWORD:kheder}
    driver-class-name: ${DATASOURCE_DRIVER_CLASS_NAME:com.mysql.cj.jdbc.Driver}

  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
```

الشكل 7. 1 ملف application.yml

بعد إعداد الاتصال بقاعدة البيانات، تم تنظيم المشروع وفقاً لمبادئ المعمارية النظيفة (Clean Architecture) لضمان هيكلية واضحة وقابلة للصيانة. وقد تم تقسيم المشروع إلى الطبقات الأساسية التالية:



الشكل 7. 2 المستوى الأول من مجلدات الخادم الخلفي

1. **Entity**:

- تتضمن هذه الطبقة الكيانات التي تمثل الهياكل الأساسية للبيانات في التطبيق، مثل كيان User الذي يمثل جدول المستخدمين في قاعدة البيانات وكيان UserDetails وأخيرا UserUrlDetails. يتم تعريف الكيانات كفئات java مع تحديد الحقول التي تتوافق مع أعمدة الجداول.

2. **Repositories**:

- تحتوي هذه الطبقة على واجهات لعمليات الوصول إلى **البيانات** CRUD تم استخدام JpaRepository من Spring Data JPA لتسهيل تنفيذ هذه العمليات بشكل تلقائي.

3. **Services**:

- مسؤولة عن تنفيذ منطق العمل الرئيسي للتطبيق، حيث تتم معالجة البيانات والتأكد من صحة العمليات وتنسيقها بين الطبقات المختلفة. يتم حقن مستودعات البيانات (Repositories) داخل الخدمات (Services)باستخدام.Dependency Injection

4. **Controllers**:

- تتعامل مع الطلبات الواردة من المستخدمين عبر الويب .(HTTP Requests) تحتوي كل وحدة تحكم (Controller)على نقاط نهاية (Endpoints) تعالج الطلبات وتعيد الردود المناسبة. يتم توجيه الطلبات إلى الخدمات (Services) المناسبة بناءً على نوع العملية المطلوبة.

5. **DTOs (Data Transfer Objects)**:

- تُستخدم لنقل البيانات بين الطبقات المختلفة بطريقة آمنة ومنظمة، مما يعزز الأمان ويقلل من التعقيد.

6. **Config**:

- تحتوي هذه الطبقة على جميع ملفات التكوين والإعدادات الخاصة بالتطبيق، بما في ذلك إعدادات الأمان وتهيئة التوصيلات مع الخدمات الخارجية.

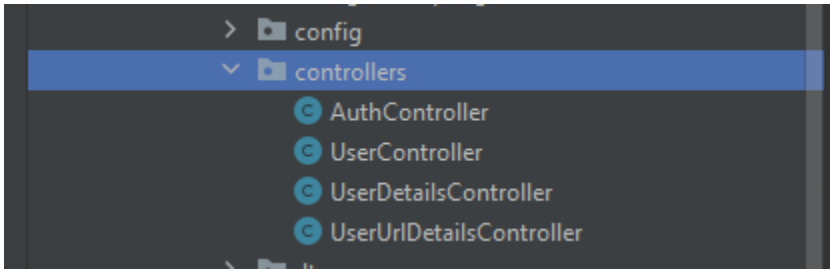
7. **Exception**:

- تمثل هذه الطبقة معالجة الاستثناءات المخصصة، حيث يتم تحديد الأخطاء المحتملة وتوفير ردود فعل مناسبة للمستخدمين.

من خلال هذه الهيكلية المنظمة والمرنة، يتمكن المشروع من الحفاظ على استقراره وتوسعه بشكل فعال، مما يسمح بإضافة ميزات جديدة أو تعديل العمليات الحالية بسهولة دون التأثير على الأجزاء الأخرى من التطبيق.

3.3.7- طبقة المتحكمات (controllers)

وتحتوي جميع التوابع المعنية بمعالجة الطلبات، والتي تم تقسيمها وظيفيا لأربعة أقسام موضحة في الشكل 10.7. سوف نسلط الضوء على بعض هذه التوابع والتي تم تصميمها وتنجيزها بنفس المنهجية والمعمارية النظيفة، بحيث لاجل كل تابع هناك كائن DTO خاص يستخدم كدخل لهذا التابع وendpoint تحدد نوع الطلب هل هوي GET أو POST ويتم تنجيز التابع في طبقة ال services المقسمة أيضا وظيفا بما يتوافق مع تقسيمات المتحكمات. وبدورها تتخاطب مع طبقة ال Repository الخاص بالجدول الذي السمتهدف في قاعدة المعطيات ومن ثم تنفذ العملية.



الشكل 7. 3 التقسيم الوظيفي لطبقة المتحكمات

- AuthenticationController:

تحتوي على تابعين sign in و sign up وكلاهما يحتاج اسم مستخدم وكلمة مرور كدخل ويعيد AccessToken واسم المستخدم ونوع الاشتراك والدور (role)