الجمهورية العربية السورية
المعهد العالي للعلوم التطبيقية والتكنولوجيا
قسم النظم المعلوماتية
العام الدراسي 2024/2025

مشروع سنة رابعة

تطوير تطبيق حجز مواقف سيارات

تقديم
حسن أمين سلامي
أريج حيدر محمد

إشراف
د . سميح جمول
ما. عماد قرحيلي

الخلاصة

يتناول هذا المشروع تصميم وتنفيذ نظام حجز مواقف السيارات، يهدف إلى تبسيط عملية الحجز وتحسين تجربة المستخدم من خلال إدارة فعالة لمواقف السيارات. يركز النظام على توفير وظائف رئيسية تشمل حجز المواقف عبر تطبيق موبايل ومنصة ويب، إدارة الحجوزات مركزيًا، والتحقق من الوصول باستخدام تقنيتي Bluetooth وNFC.

تطلب تصميم النظام مراعاة متطلبات وظيفية مثل تسجيل المستخدمين، توليد Token آمن باستخدام Kerberos عند إتمام الدفع ، والدفع الإلكتروني، إلى جانب متطلبات غير وظيفية تشمل الأمان العالي. تم اعتماد منهجية تصميم تدريجية تشمل تطوير نموذج أولي (MVP) في المرحلة الأولى، ثم إضافة ميزات متقدمة هي Bluetooth وNFC في المرحلة الثانية، باستخدام تقنيات مثل Flutter للتطبيق، ASP.NET للخادم، وSQL Server لقاعدة البيانات.

شمل التنفيذ تطوير وحدات مترابطة تشمل تطبيق الموبايل، الخادم، وحساسات محاكاة. واجه الفريق تحديات مثل عدم توافر حساسات فعلية، مما استلزم إنشاء صفحة ويب بسيطة لمحاكاة دور الحساسات، ومحدودية دعم NFC في بعض الأجهزة ككاتب فقط وليس قارئ، فتم التعامل معها مؤقتًا عبر المحاكاة. تم استخدام Bluetooth Low Energy (BLE) للتحقق التلقائي عبر مسافات تصل إلى 10 أمتار، وNFC للتحقق اليدوي والدفع السريع على مسافات قصيرة، مع ضمان الأمان عبر التشفير وKerberos للتحقق من صحة توكن الحجز .

يبرهن المشروع على إمكانية بناء نظام حجز مواقف موثوق وقابل للتوسع، مع إمكانيات تطوير مستقبلية مثل تعزيز تكامل NFC للدفع المتقدم وإضافة حساسات فعلية لتحسين الأداء. النظام مصمم للتكيف مع بيئات التشغيل المتنوعة، مما يضمن تجربة مستخدم سلسة وفعالة.

Abstract

This venture involves designing and implementing a automobile parking reservation machine aimed at simplifying the booking method and improving person revel in thru green parking control. The machine makes a speciality of presenting center functionalities along with mobile and internet-primarily based parking reservations, centralized reserving management, and get admission to verification the use of Bluetooth and NFC technology.

The system layout required consideration of purposeful requirements like consumer registration, stable token generation, and digital payments, along non-practical requirements together with high security. A phased layout method changed into adopted, beginning with an MVP (Minimum Viable Product) in the first phase, followed via adding advanced capabilities like Bluetooth and NFC within the second phase, the use of technologies along with Flutter for the application, ASP.NET for the server, and SQL Server for the database .

Implementation involved growing interconnected modules including the mobile utility, server, and simulated sensors. The group faced demanding situations inclusive of the unavailability of physical sensors, which necessitated developing a easy internet page to simulate sensor functionality, and constrained NFC aid on some devices (write-only mode), quickly addressed via simulation. Bluetooth Low Energy (BLE) was used for automatic verification over distances up to 10 meters, even as NFC enabled manual verification and fast fee at short stages, with safety ensured thru encryption .

The undertaking demonstrates the feasibility of building a reliable and scalable parking reservation system, with future improvement potential including more suitable NFC integration for advanced bills and adding physical sensors to improve performance. The system is designed to adapt to numerous operational envi

المحتويات

الفصل الأول

التعريف بالمشروع

يتضمن هذا الفصل التعريف بالمشروع ومتطلباته.

1.1- مقدمة

يعد تطبيق حجز مواقف السيارات هي حل تكنولوجي حديث صُمم لتبسيط وتعزيز عملية إيجاد وحجز مساحات الانتظار في المناطق الحضرية المزدحمة. يعتمد تطبيق جوال بديهي ومنصة إدارة متطورة قائمة على الويب، مما يُمكّن العملاء من اختيار مواقف السيارات بناءً على الموقع أو السعر أو التوفر، مع ضمان تجربة مستخدم سلسة وآمنة. تستفيد المنظومة من تقنيات متطورة مثل البلوتوث والاتصال قريب المدى (NFC) للتحقق من المستخدم عند الدخول والخروج من كراج الانتظار، إلى جانب بروتوكول كيربيروس لتوليد رموز تشفير (Tokens) من أجل عمليات آمنة. يهدف المشروع إلى تقليل الجهد والوقت المهدر في البحث عن مواقف السيارات، وتعزيز كفاءة إدارة كراجات الانتظار، وتوفير بيئة موثوقة لكل من العملاء ومديري المواقف. من خلال دمج التقنيات المتقدمة وقاعدة البيانات المركزية، يطمح النظام لتقديم حل مستدام يدعم المدن الذكية ويُحسّن تجارب التنقل اليومية.

.

2.1- هدف المشروع

هدف تطبيق "حجز مواقف سيارات" إلى تقديم حل تقني مبتكر لتسهيل عملية حجز مواقف السيارات وتحسين تجربة المستخدم من خلال استخدام تقنيات متقدمة. تشمل الأهداف الرئيسية للمشروع ما يلي:

- تسهيل حجز مواقف السيارات: تمكين المستخدمين من اختيار وحجز موقف سيارة بسهولة عبر تطبيق الهاتف الذكي، مع تحديد الوقت والموقع المناسبين، لتقليل الوقت المستغرق في البحث عن موقف.
- ضمان أمان الحجز باستخدام بروتوكول :Kerberos توليد توكن فريد لكل حجز باستخدام بروتوكولKerberos ، وهو بروتوكول أمان يعتمد على التذكرة (Ticket) للتحقق من هوية المستخدم وضمان صحة الحجز. يتم إرسال هذا التوكن إلى حساسات الكراج للتحقق منه عبر السيرفر.
- الاقتران الآمن مع الكراج: توفير نظام اقتران فعال باستخدام تقنيات Bluetooth أو Near Field Communication (NFC)، مما يتيح للمستخدم التواصل مع حساسات باب الكراج لتأكيد الحجز بسرعة وأمان.
- توفير نظام دفع إلكتروني: إتاحة إمكانية الدفع عبر محفظة إلكترونية مدمجة في التطبيق، مما يضمن سهولة وسرعة إتمام عمليات الدفع.
- تحسين تجربة المستخدم: تصميم واجهة مستخدم بسيطة وسلسة، مع تقليل الازدحام في الكراجات من خلال تنظيم عملية الحجز مسبقاً.

3.1- المتطلبات الوظيفية

❖ **يسمح النظام للزبائن بمايلي**

- تسجيل جديد وتسجيل الدخول إلى التطبيق باستخدام البريد الالكتروني
- عرض المواقف المتاحة .
- الفلترة والبحث عن المواقف وفق التوفر الزمني (بناءً على وقت الوصول والمغادرة المحدد).
- حجز الموقف باختيار تاريخ ووقت الوصول والمغادرة .
- عرض سعر الحجز .
- مشاهدة الحجوازات السابقة و الحالية
- تأكيد الحجز و دفع التكاليف عن طريق محفظة ضمن التطبيق .

- تعديل الحجز (تغيير الوقت او الموقف) و ذلك خارج الفترة الحرجة .

❖ **يجب أن يسمح النظام لتطبيق الزبون بما يلي :**

- يجب أن يسمح الحساس بالتحقق من هوية السائق، ومعلومات الحجز
- ارسال اشعارات للمستخدمين عند القيام باي عملية (فشل/نجاح)

❖ **يسمح النظام للمديرين بما يلي :**

- تسجيل الدخول إلى صفحة الويب باستخدام بريد الكتروني .
- إدارة المواقف
  - أضافة موقف بمواصفاته (سعة , اسعار , موقع)
  - تعديل مواصفات موقف (سعة , اسعار , موقع)
  - حذف موقف .
- إدارة الحجوزات :
  - عرض الحجوزات النشطة و الملغاة .
  - إلغاء الحجز يدوياً (حالات طوارئ) .
- إدارة المستخدمين :
  - حظر مؤقت لزبون (عند الطلب و التخلف في نفس اليوم أكثر من خمس مرات) .
  - فك حظر مستخدم (بعد اسبوع من الحظر المؤقت) .
  - حظر نهائي لزبون (عند تعرضه لمرتي حظر مؤقت) .
- إظهار احصائيات حول حجز المواقف في ايام محددة و شهر محدد و سنة محددة .

❖ **يجب أن يسمح للخادم المركزي بما يلي :**

- إدارة Token :
  - توليد Token مشفرة باستخدام بروتوكول Kerberos .
  - ارسال Token إلى الزبون وضمان وصولها .

- تحديد فترة حرجة (±15 دقيقة) للسماح بالوصول المبكر أو المتأخر .
- تخزين Token محلياً حتى الانتهاء من فترة الحجز و مغادرة الزبون .
- ترك 3 مواقف احتياطية فارغة دوماً من دون حجز لضمان عدم انتظار الأشخاص خارج الموقف في حال تأخر شخص اخر داخل الموقف.
- حذف Token بعد انتهاء الحجز أو إلغائه من قبل الزبون .
- تحديد صلاحية Token وهي الوقت الذي تم حجزه من قبل الزبون .
- ارسال Token إلى الحساسات .
- تحديث حالة الحجز تلقائيًا (من "معلق" إلى "منتهي") وذلك حسب البيانات الواصلة من الحساسات.

❖ **يجب ان يتحقق حساس الدخول /الخروج من صحة ال token المرسلة إلى التطبيق .**

4.1- المتطلبات الغير الوظيفية

❖ يجب أن يكون النظام أمناً: ان تكون جميع الاتصالات مشفرة

❖ يجب ان يكون السيرفر قادر على استقبال بيانات الدخول /الخروج من الحساسات بالزمن الحقيقي.

❖ زمن الاستجابة السيرفر: يجب ألا يتجاوز 5 ثوان في كل تفاعل مع النظام.

❖ يجب أن يدعم حتى 10,000 مستخدم نشط في نفس الوقت.

❖ يجب أن تكون الحساسات متصلة بالتطبيق عبر Bluetooth وتفعيل Bluetooth تلقائياً عند الاقتراب من الموقف.

❖ يجب أن يكون النظام سهل الاستخدام

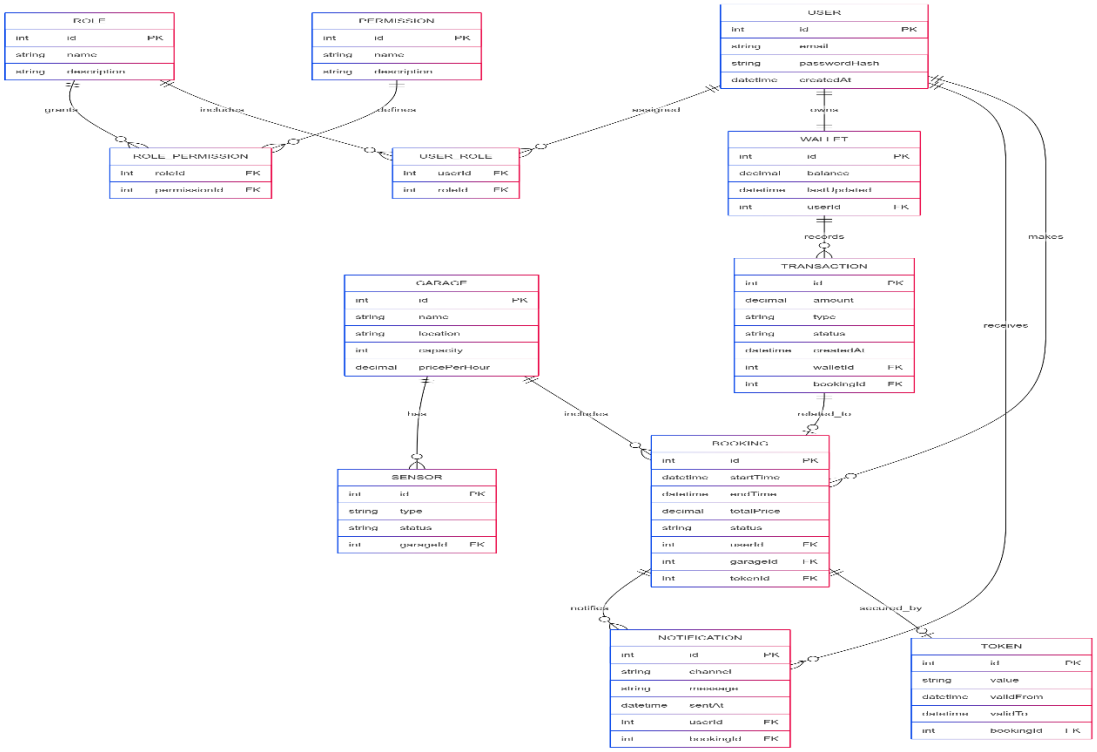
❖ .يجب ان يكون هناك إمكانية تحديث النظام دون توقف الخدمة (Zero Downtime Deployment).

الفصل الثاني

الدراسة التحليلية

يوضح هذا الفصل عمليّة تحليل النظام ودراسة متطلباته.

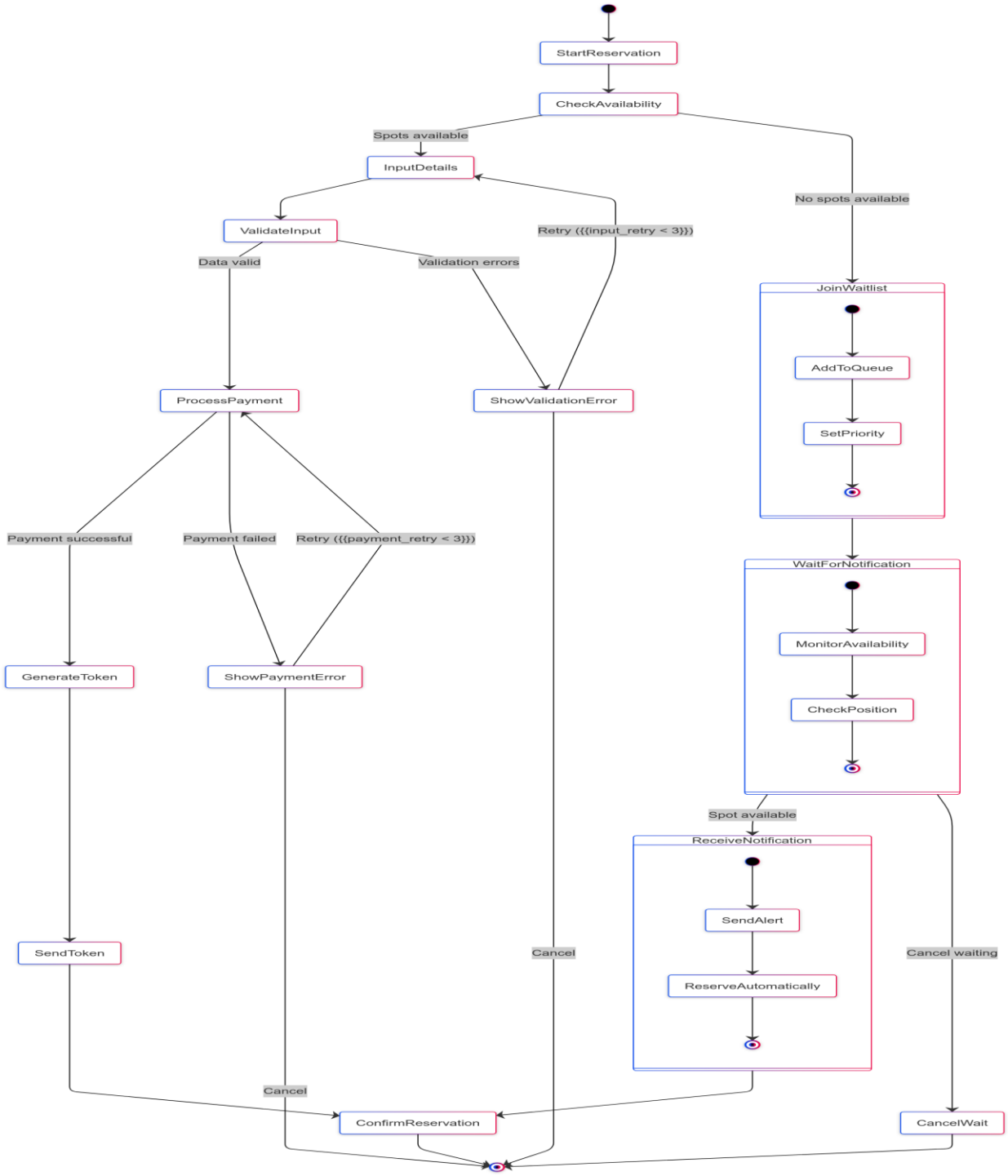
1.2- مخطط قاعدة بيانات علاقة الكيانات ERD



مخطط علاقة الكيانات

يمثل المخطط السابق قاعدة بيانات لنظام حجز مواقف سيارات ذكي، ويشمل جميع المكونات الأساسية لإدارة المستخدمين والحجوزات والصلاحيات. يحتوي المخطط على جدول المستخدمين (User) الذي يحتفظ بمعلوماتهم مثل الاسم والبريد الإلكتروني، وهو مرتبط بالحجوزات (Transaction) التي تسجل تفاصيل الحجز والدفع. كل حجز مرتبط بموقف سيارات (Parking)يحدد مكان الوقوف، ويرتبط الموقف بمستشعر (Sensor) يراقب حالته بشكل مباشر. كما يحتوي النظام على جدول الإشعارات (Notification) لإرسال التنبيهات للمستخدمين مثل تذكيرهم بالحجز، وجداول لإدارة الأدوار والصلاحيات (Role, Permission, Role_Permission, User_Role) لتنظيم الوصول والتحكم في صلاحيات المستخدمين. ويوجد أيضًا جدول (Token) لتخزين رموز الدخول الخاصة بالمستخدمين للتحقق من هويتهم. العلاقات بين هذه الجداول توضح كيفية ربط المستخدم بالحجز، والموقف بالمستشعر، وكذلك الصلاحيات بالأدوار لضمان إدارة متكاملة ومرنة للنظام، مما يسمح بتقديم خدمة ذكية لحجز مواقف السيارات ومتابعة حالتها بشكل دقيق.

2.2- مخطط النشاط لعملية الحجز



مخطط تدفق عملية حجز موقف السيارة

يمثل المخطط **تدفق عملية حجز موقف السيارة (Reservation Process Flow)** في النظام. يوضح المخطط الخطوات بالتسلسل من بداية طلب الحجز وحتى تأكيده أو إلغائه، ويعالج جميع السيناريوهات الممكنة مثل التحقق من التوافر، معالجة الدفع، وإدارة قائمة الانتظار.

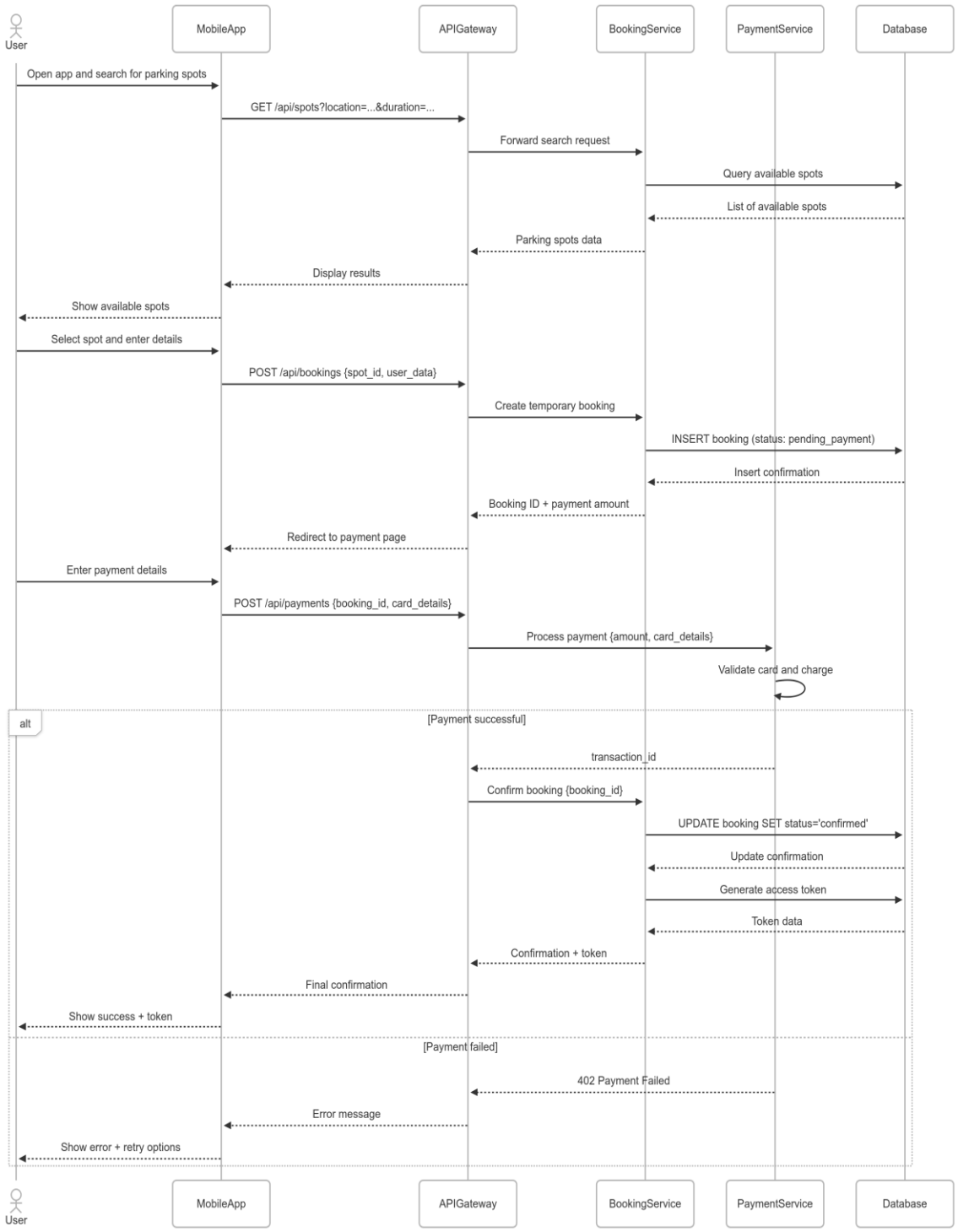
تبدأ العملية بـ **StartReservation** ثم **CheckAvailability** للتحقق من توفر الموقف. إذا كان هناك موقف متاح، ينتقل المستخدم إلى إدخال بياناته في **InputDetails**، يليها **ValidateInput** للتحقق من صحة البيانات المدخلة. إذا كانت البيانات صحيحة، تتم معالجة الدفع في **ProcessPayment**، حيث يؤدي نجاح الدفع إلى **GenerateToken** ثم إرسال الرمز للمستخدم في **SendToken**، وبعدها تأكيد الحجز في **.ConfirmReservation** أما في حال فشل الدفع، تعرض **ShowPaymentError** مع إمكانية إعادة المحاولة حتى 3 مرات.

إذا لم يكن هناك موقف متاح عند التحقق من التوافر، ينتقل المستخدم إلى **JoinWaitlist**، حيث تتم إضافته إلى قائمة الانتظار عبر **AddToQueue** مع تحديد الأولوية باستخدام التي تراقب توفر الموقف عبر **MonitorAvailability**وتحدد موقعه في القائمة عبر **.CheckPosition**عند توفر موقف، يستقبل المستخدم إشعارًا في **ReceiveNotification**، ويمكنه إما حجزه تلقائيًا عبر **ReserveAutomatically** أو تلقي تنبيهًا في **SendAlert**بعدها يتم تأكيد الحجز في **ConfirmReservation**أو إلغاء الانتظار في **.CancelWait**

يحتوي المخطط أيضًا على آليات لإعادة المحاولة عند إدخال بيانات خاطئة أو عند فشل الدفع، وكذلك إمكانية الإلغاء في أي مرحلة. هذه التفاصيل توضح جميع الحالات التي قد يواجهها المستخدم أثناء الحجز، بما في ذلك التعامل مع عدم توفر مواقف وإدارة قائمة الانتظار بشكل ذكي وآلي.

باختصار، هذا المخطط مصمم لتقديم تجربة متكاملة وسلسة للمستخدم بدءًا من محاولة الحجز وحتى تأكيده أو إلغائه، مع تغطية جميع السيناريوهات المحتملة لضمان حصول المستخدم على الموقف بأفضل طريقة ممكنة.

3.2- مخطط تتالي النظام



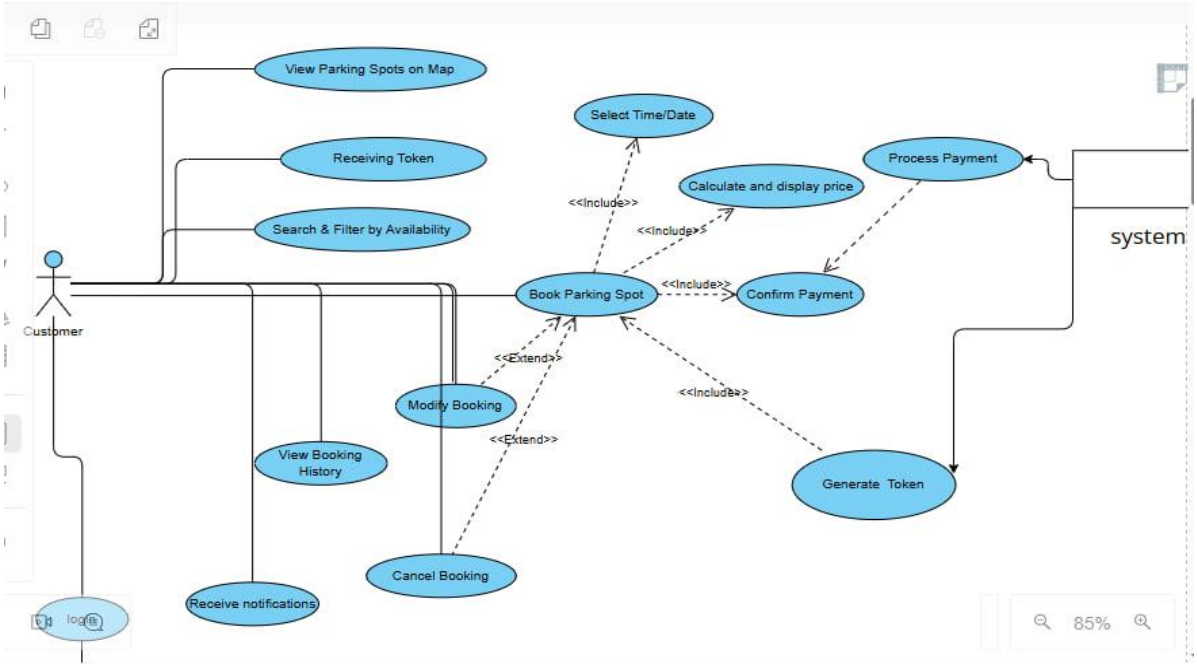
مخطط تتالي النظام

يُظهر المخطط **تسلسل العمليات (Sequence Diagram)** لعملية حجز موقف سيارات ودفع الرسوم عبر تطبيق الهاتف المحمول، بدءًا من بحث المستخدم وحتى تأكيد الحجز أو فشل الدفع. يوضح التفاعل بين المكونات الرئيسية: المستخدم، تطبيق الهاتف(MobileApp) ، بوابة الـ(APIGateway) API ، خدمة الحجز(BookingService) ، خدمة الدفع (PaymentService)، وقاعدة البيانات .(Database) إليك الشرح في فقرة متكاملة:

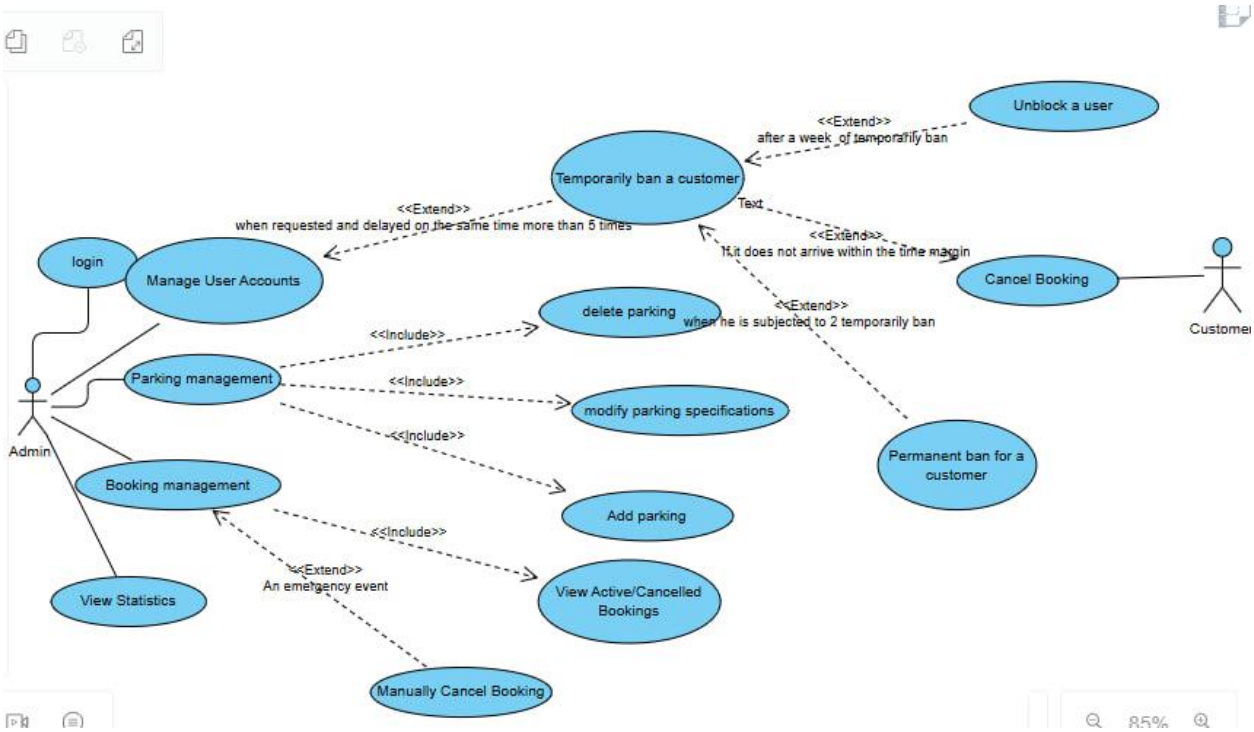
يبدأ المستخدم بفتح التطبيق والبحث عن المواقف المتاحة عبر إرسال طلب GET إلى واجهة برمجة التطبيقات مع الموقع والمدة المطلوبة، فيقوم APIGateway بتمرير الطلب إلى BookingService التي تستعلم عن المواقف المتاحة من قاعدة البيانات وتعيد قائمة النتائج. يتم عرض المواقف المتاحة للمستخدم في التطبيق ليختار الموقف ويدخل بياناته الشخصية. عندها يرسل التطبيق طلب POST بحجز الموقف المختار إلىAPIGateway ، والتي بدورها تمرره إلى BookingService لإنشاء حجز مؤقت في قاعدة البيانات بحالة(pending_payment) ، وترجع معرّف الحجز والمبلغ المستحق ليتم توجيه المستخدم إلى صفحة الدفع.

بعد إدخال تفاصيل البطاقة، يرسل التطبيق طلب POST آخر لإتمام الدفع، فيتم تمريره إلى PaymentService التي تتحقق من معلومات البطاقة وتقوم بعملية الخصم. إذا نجحت عملية الدفع، تقوم PaymentService بإرجاع transaction_id إلى BookingService التي تؤكد الحجز عبر تحديث حالته إلى (confirmed) في قاعدة البيانات، ثم تولد رمز وصول (token)وتعيده مع تأكيد الحجز إلى التطبيق ليظهر للمستخدم. أما إذا فشلت عملية الدفع، يتم إرسال رسالة خطأ للتطبيق لعرضها للمستخدم مع خيارات لإعادة المحاولة.

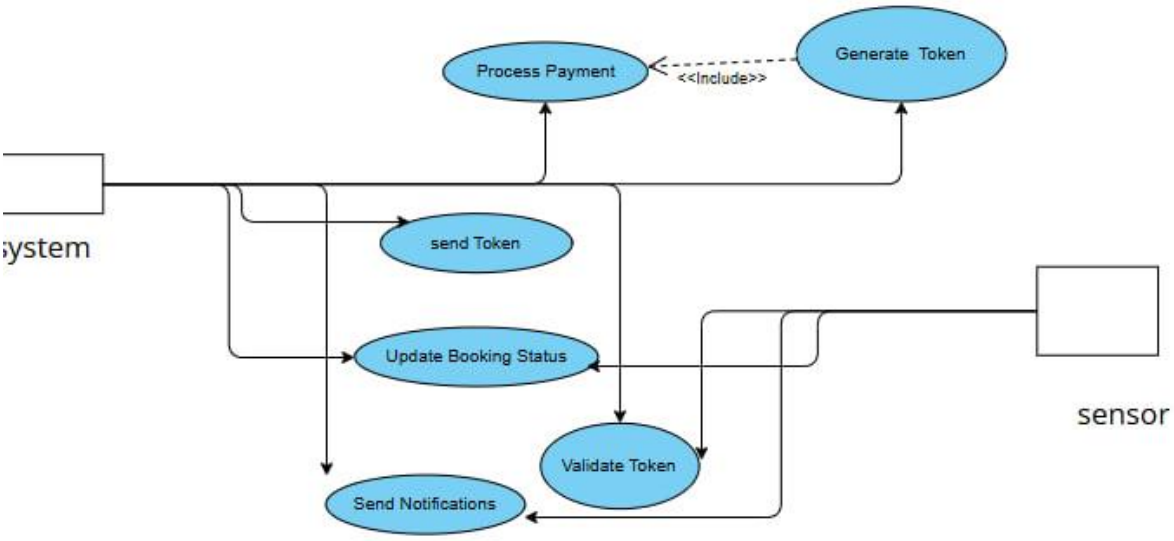
4.2 مخططات حالات الاستخدام



مخطط حالة الاستخدام للزبون



مخطط حالة الاستخدام لمدير النظام



مخطط حالة الاستخدام للنظام

الفصل الثالث

الدراسة النظرية

يقدم هذا الفصل شرح موجز عن الأساليب و المنهجيات المستخدمة لتنجيز المشروع.

1.3البنية المعمارية في ASP.NET

تمثل البنية المعمارية رؤية رفيعة المستوى للنظام، تحدد المكونات الرئيسية، وتوضح العلاقات والتبعيات فيما بينها، وتضع القواعد التي تحكم كيفية تفاعلها. إنها خارطة طريق مفاهيمية توجه جميع قرارات التصميم والتطوير اللاحقة.

تتضمن قرارات التصميم الأساسية في البنية المعمارية:

1-التقسيم إلى مكونات (Componentization): تحديد الوحدات الوظيفية الرئيسية في النظام، مثل الخدمات الدقيقة، الوحدات، أو الطبقات.

2-التفاعل بين المكونات (Component Interaction): تحديد آليات الاتصال بين المكونات، سواء كانت متزامنة (Synchronous) عبر استدعاءات واجهة برمجة التطبيقات (APIs) أو غير متزامنة (Asynchronous) عبر قوائم الانتظار والرسائل.

3-نمط التنظيم (Organizational Pattern): اختيار النمط المعماري الأنسب، مثل المعمارية المعتمدة على الخدمات المصغرة (Microservices)، أو المعمارية الطبقية (Layered Architecture)، أو المعمارية المبنية على الأحداث (Event-Driven Architecture).

4-قابلية الصيانة (Maintainability): تسهل الهندسة المعمارية الواضحة عملية فهم وتعديل النظام. عندما تكون المسؤوليات مفصولة بوضوح، يصبح إصلاح الأخطاء وإضافة الميزات أسهل وأقل عرضة للتسبب في مشاكل جانبية.

5-قابلية التوسع (Scalability): تسمح البنية المعمارية المصممة جيدًا للنظام بالتعامل مع الأحمال المتزايدة من المستخدمين والبيانات بكفاءة، سواء كان ذلك عن طريق التوسع الأفقي (إضافة المزيد من الخوادم) أو التوسع الرأسي (ترقية الخوادم).

6-المرونة (Flexibility): تمكن النظام من التكيف مع التغيرات في متطلبات العمل والتقنيات بمرور الوقت دون الحاجة إلى إعادة بناء شاملة.

-7الموثوقية (Reliability): تساهم في بناء نظام قوي ومقاوم للأخطاء، حيث يمكن لعزل المكونات أن يمنع فشل جزء واحد من النظام من التسبب في فشل النظام بأكمله.

2.3البنية المعمارية النظيفة Clean Architecture

تم اعتماد هذه المنهجية في التطبيق الخلفي .

1.2.3مفهوم البنية المعمارية النظيفة

البنية المعمارية النظيفة هي أكثر من مجرد نمط تصميمي، بل هي فلسفة هندسية تهدف إلى إنشاء أنظمة برمجية مرنة، قابلة للاختبار، ومستقلة عن التفاصيل الخارجية التي قد تتغير بمرور الوقت. جوهر هذه الفلسفة يكمن في الفصل الصارم بين طبقات النظام، حيث يتم عزل منطق العمل الأساسي (Business Logic) عن أي تبعيات خارجية مثل قواعد البيانات، واجهات المستخدم، أو الأطر البرمجية (Frameworks).

الهدف الرئيسي هو حماية منطق العمل من التغيرات الخارجية. بعبارة أخرى، يجب أن يكون منطق التطبيق قادرًا على العمل بغض النظر عن:

قاعدة البيانات: سواء كانت SQL أو NoSQL.

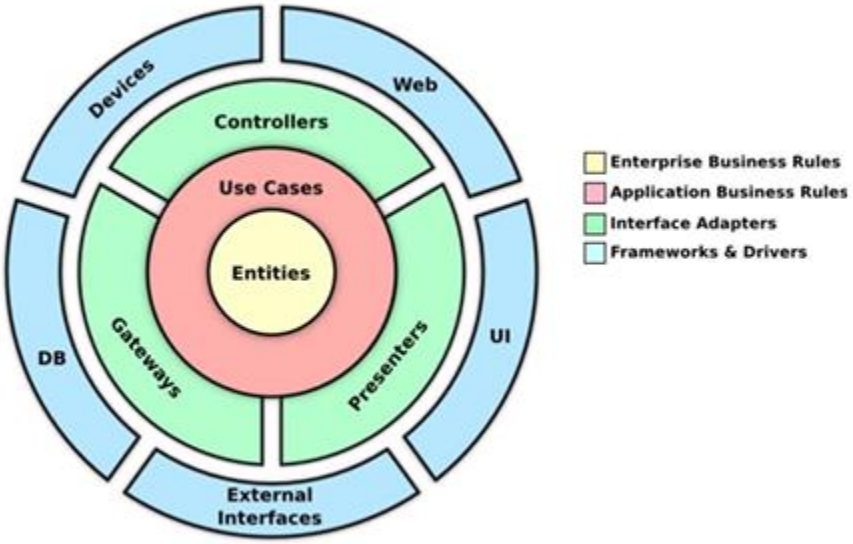
واجهة المستخدم: سواء كانت ويب، سطح مكتب، أو واجهة سطر الأوامر (CLI).

الأطر البرمجية: سواء كان تطبيقًا مبنيًا على Spring Boot، Django، أو Express.

الأدوات الخارجية: مثل أدوات التخزين المؤقت (Caching) أو خدمات المراسلة (Messaging services).

هذا الاستقلال يعزز قابلية النظام للصيانة والتطوير، حيث يمكن تعديل أي من هذه المكونات الخارجية دون التأثير على جوهر التطبيق.

2.2.3هيكل البنية المعمارية النظيفة في التطبيق الخلفي



تُبنى البنية المعمارية النظيفة على مبدأ الدوائر المتداخلة، حيث تكون الدوائر الداخلية هي الأكثر تجريدًا واستقلالية، بينما تكون الدوائر الخارجية هي الأكثر تحديدًا وتعتمد على المكونات الداخلية.

1. دائرة الكيانات (Entities):

تحتوي على قواعد العمليات الخاصة بالمؤسسة (Enterprise-wide Business Rules). هذه الكيانات هي كائنات تمثل جوهر النظام، مثل User، Product، أو Order. يجب أن تكون هذه الكيانات مستقلة تمامًا عن أي طبقة أخرى ولا تحتوي على أي تبعيات خارجي

2. دائرة حالات الاستخدام (Use Cases):

تحتوي على قواعد العمل الخاصة بالتطبيق (Application-specific Business Rules). تُعرف هذه الطبقة العمليات التي يمكن للنظام القيام بها، مثل "إنشاء طلب جديد" أو "تحديث معلومات المستخدم". تقوم حالات الاستخدام بتنسيق تدفق البيانات بين الكيانات وتستخدم بوابات الواجهات (Interface Gateways) للتفاعل مع الطبقات الخارجية.

3. دائرة المحولات/البوابات (Interface Adapters):

تعمل كوسيط بين الطبقات الداخلية والخارجية. تقوم هذه الطبقة بتحويل البيانات من صيغة مناسبة للطبقة الخارجية (مثل صيغة JSON في الويب) إلى صيغة مناسبة للطبقات الداخلية (مثل كائنات الكيانات)، والعكس صحيح.

وحدات التحكم (Controllers) والمقدمات (Presenters): تستقبل الطلبات من واجهة المستخدم (الويب، الموبايل) وتحولها إلى حالات استخدام.

بوابات قواعد البيانات (Database Gateways): تُعرف باسم "المستودعات" (Repositories)، وهي واجهات (Interfaces) تحدد كيفية التفاعل مع قواعد البيانات.

4. دائرة الأطر والواجهة الخارجية (Frameworks & Drivers):

المسؤولية: تحتوي على جميع التفاصيل الخارجية التي قد تتغير، مثل قواعد البيانات، وإطارات الويب (Web Frameworks)، وواجهات المستخدم (UI). هذه الطبقة هي الأكثر اعتمادية على المكونات الداخلية.

3.2.3المبادئ الأساسية للبنية المعمارية النظيفة

1-مبدأ التبعية (Dependency Rule): هذا هو أهم مبدأ في هذه البنية. ينص على أن التبعيات يجب أن تتجه دائمًا إلى الداخل. لا يمكن لدائرة داخلية أن تعرف أي شيء عن دائرة خارجية. على سبيل المثال، لا يمكن لحالة الاستخدام أن تعتمد على وحدة التحكم، ولا يمكن للكيان أن يعتمد على قاعدة البيانات. يتم تحقيق ذلك باستخدام واجهات (Interfaces) في الطبقات الداخلية وتطبيقها في الطبقات الخارجية.

2-الاستقلالية عن الأطر (Framework Independence): يمكن استخدام أي إطار برمجي، ولكن يجب أن يتم "توصيله" بالطبقة الخارجية، بحيث يمكن استبداله بسهولة في أي وقت دون التأثير على منطق العمل.

3-الاستقلالية عن واجهة المستخدم (UI Independence): يمكن تغيير واجهة المستخدم بسهولة دون الحاجة إلى تعديل منطق العمل الأساسي.

4-الاستقلالية عن قواعد البيانات (Database Independence): يمكن تبديل قواعد البيانات دون تعديل حالات الاستخدام أو الكيانات.

5-قابلية الاختبار (Testability): نظرًا لأن منطق العمل معزول تمامًا، يصبح اختباره أسهل بكثير. يمكن اختبار حالات الاستخدام والكيانات دون الحاجة إلى تشغيل خادم ويب أو قاعدة بيانات.:

3.3 بروتوكول الكيربيروس

1.3.3 مقدمة عن بروتوكول الكيربيروس

في عالم اليوم المتزايد ترابطًا وتوزيعًا للأنظمة، أصبحت الحاجة إلى حماية معلومات المستخدم وموارد الخادم أمرًا بالغ الأهمية. المصادقة هي حل حاسم لتحقيق الأمن المطلوب وحماية الخدمات من الحرمان، فدون معرفة هوية العميل الذي يطلب عملية ما، يصعب تحديد ما إذا كان مخولًا أم لا. الأنظمة التقليدية التي تعتمد على كلمات المرور والتي تُرسل عبر الشبكة، قد تكون عرضة للاعتراض من قِبل المهاجمين، مما يجعلها غير مناسبة في بيئات الأنظمة الموزعة. لهذا السبب، تبرز الحاجة إلى أساليب مصادقة قوية لا تسمح للمهاجمين باكتشاف كلمات المرور.

يعد بروتوكول الكيربيروس ،الذي طورتهMIT

نظام مصادقة شبكي مصمم خصيصًا لحماية أمن المستخدمين من خلال التحقق من هويتهم، وحماية أي نظام موزع يسعى لتحقيق أهداف مهمة مثل التكامل والسرية. يُعرف بقدرته على توفير خدمات مصادقة موزعة تسمح للعميل بالتصرف نيابةً عن المستخدم للتحقق من هوية خادم التطبيق دون إرسال بياناته عبر الشبكة، مما يمنع انتحال شخصية المستخدم. الهدف الرئيسي من استخدامه هو المصادقة بين المستخدمين والخدمات.

تعريف بروتوكول كيربيروس

كيربيروس (Kerberos) هو بروتوكول مصادقة شبكة حاسوب قوي يستخدم التشفير بمفتاح سري لتوفير مصادقة قوية للعميل والخادم في الشبكات غير الآمنة. تم تطويره في معهد ماساتشوستس للتكنولوجيا (MIT) بهدف تمكين الأجهزة على الشبكة من إثبات هويتها لبعضها البعض بطريقة آمنة، مع منع التنصت وهجمات إعادة الإرسال.(Replay Attacks)

يتألف بروتوكول كيربيروس بشكل أساسي من ثلاث جهات رئيسية:

1 .مركز توزيع المفاتيح(Key Distribution Center - KDC)

يُعتبر KDC هو القلب النابض لنظام كيربيروس. هو خادم موثوق به مركزيًا، مسؤول عن إصدار التذاكر (Tickets) التي تُستخدم للمصادقة. يتكون KDC من جزأين رئيسيين:

خادم المصادقة:(Authentication Server - AS)

مسؤول عن مصادقة المستخدم. عند تسجيل الدخول، يتواصل العميل مع خادم المصادقة.

يتحقق من هوية العميل باستخدام كلمة مرور أو بيانات اعتماد أخرى. إذا كانت المعلومات صحيحة، فإنه يُصدر للعميل تذكرة تُعرف باسم تذكرة منح التذاكر .(Ticket-Granting Ticket - TGT) هذه التذكرة مشفرة باستخدام مفتاح سري مشترك بين العميل و.KDC

خادم منح التذاكر:(Ticket-Granting Server - TGS)

مسؤول عن إصدار التذاكر للوصول إلى خدمات محددة.

عندما يريد العميل الوصول إلى خدمة معينة (مثل خادم ملفات أو خادم بريد)، فإنه يرسل TGT إلى .TGS يقوم TGS بالتحقق من صحة TGT ثم يُصدر تذكرة خدمة (Service Ticket) للعميل. هذه التذكرة هي التي سيستخدمها العميل للتواصل مع الخادم الذي يقدم الخدمة.

4.3 البنية المعمارية النظيفة في فلاتر

1.4.3 شرح البنية

تتكون هذه البنية من ثلاث طبقات هي :

- **طبقة العرض (Presentation Layer)**:

مسؤولة عن واجهات المستخدم وتفاعل المستخدم. تشمل شاشات تسجيل الدخول، عرض المواقف على الخريطة، واختيار المواقف أو إتمام الدفع، مع التركيز على تجربة مستخدم سلسة وبسيطة.

تتألف هذه الطبقة من :

- **منطق الأعمال BLOC**

يُستخدم نمط Bloc لإدارة الحالة في طبقة العرض، خاصة مع إطار .Flutter يعمل كوسيط بين واجهة المستخدم وطبقة المجال: يستقبل أحداث المستخدم (مثل النقر على "حجز مكان وقوف")، ويتفاعل مع طبقة المجال لتنفيذ المنطق، ثم يحدِّث حالة واجهة المستخدم.

مثال: عند اختيار مكان وقوف على الخريطة، يُرسل Bloc حدثًا (مثل `SearchAvilableGarages`)لطبقة المجال، ويُحدِّث الواجهة حسب النتيجة (تأكيد/خطأ)

الفائدة :فصل منطق العرض عن الواجهة، مما يجعل إدارة الحالة قابلة للاختبار وإعادة الاستخدام.

- **العرض:(Presentation)**

يشير إلى المكونات المرئية للمستخدم التي تُعرض البيانات ديناميكيًا بالاعتماد على حالة.Bloc

الفائدة :عزل تصميم الواجهة عن المنطق الأساسي، مما يسمح بتعديل الواجهة دون التأثير على المنطق.

- **طبقة المجال (Domain Layer):**

تحتوي على منطق الأعمال الأساسي وقواعد التطبيق، مثل التحقق من صحة بيانات الحجز أو إدارة Token. تكون مستقلة عن أي تقنيات خارجية، مما يتيح اختبارها بسهولة وإعادة استخدامها دون تعديل.

تتألف هذه الطبقة من :

الكيانات (Entity)

هي كائنات البيانات الأساسية التي تمثل معلومات التطبيق الجوهرية، مستقلة عن أي تقنية أو تصميم.

حالات الاستخدام(Use Cases)

تُعرّف الإجراءات التي ينفّذها النظام، معزّزةً المنطق التجاري. كل حالة تركّز على مهمة واحدة وتتفاعل مع الكيانات.

واجهة المستودع(Repository Interface)

تحدّد العمليات على البيانات (مثل استرجاع/حفظ بيانات الحجز) دون تفصيل آلية التنفيذ. تعمل كجسر بين طبقتَي المجال والبيانات.

- طبقة البيانات (Data Layer):

تتولى التواصل مع مصادر البيانات الخارجية، مثل الخادم (عبر واجهات برمجية) أو الأجهزة (مثل حساسات بلوتوث وNFC). تعمل كوسيط لجلب البيانات أو إرسالها، مثل بيانات الحجز أو Token، مع ضمان الاتصال الآمن والفعال.

تتألف هذه الطبقة من :

النماذج:(Models)

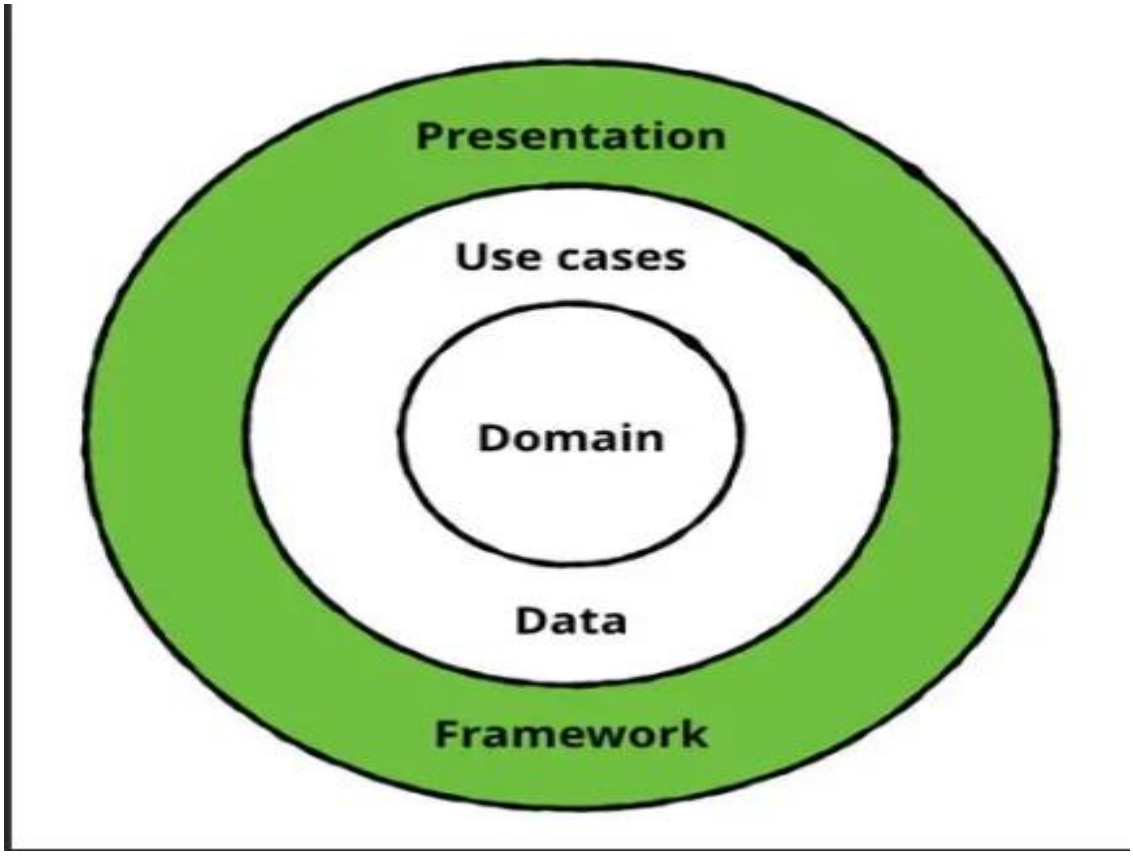
تمثل البيانات في طبقة البيانات، غالبًا مشابهة للكيانات لكنها تحتوي تفاصيل تقنية مرتبطة بمصادر البيانات) مثل تنسيقات JSON .

تنفيذ المستودع(Repository Implementation)

يوفر التنفيذ الفعلي لواجهات المستودع (المحددة في طبقة المجال) عبر التفاعل مع مصادر البيانات.

مصادر البيانات (DataSources)

مكونات تتفاعل مباشرة مع مصادر البيانات الخارجية(قواعد بيانات، واجهات برمجةAPIs ، أجهزة مثل بلوتوث/ NFC)



الشكل توضح البنية المعمارية النظيفة

2.4.3- مقارنة البنية المعمارية النظيفة مع بعض البنى الأخرى وسبب واختيارها

البنية	المزايا	العيوب
Clean Architecture	فصل واضح بين منطق الأعمال، واجهة المستخدم، ومصادر البيانات. سهولة اختبار الوحدات (Unit Testing)لطبقة المجال. مرونة في تغيير مصادر البيانات) مثل استبدال API أو مكتبة اتصال (دون التأثير على الكود. دعم التوسع لتطبيقات معقدة.	زيادة التعقيد الأولي بسبب إنشاء طبقات متعددة . وقت تطوير أطول للمشاريع الصغيرة.
MVC(Model-View-Controller)	بساطة في التنفيذ للتطبيقات الصغيرة . سهولة الفهم للمطورين المبتدئين.	قد تصبح فوضوية مع نمو التطبيق بسبب الاقتران الضيق بين المكونات صعوبة في اختبار الوحدات بسبب التداخل بين المنطق و واجهة المستخدم
BLoC(BusinessLogic Component)	فصل جيد بين منطق الأعمال وواجهة المستخدم. دعم التفاعل في الوقت الفعلي (مثل إشعارات الحجز).	تعقيد إضافي بسبب إدارة Streams و.Events قد يتطلب وقتًا أطول لفهمها وتنفيذها مقارنة بـ.MVC
MVVM(Model-View-ViewModel)	مناسبة للتطبيقات التي تعتمد على ربط البيانات .	قد تؤدي إلى اقتران بين ViewModel وواجهة المستخدم .

	سهلة الاستخدام مع أدوات إدارة الحالة مثل.Provider	

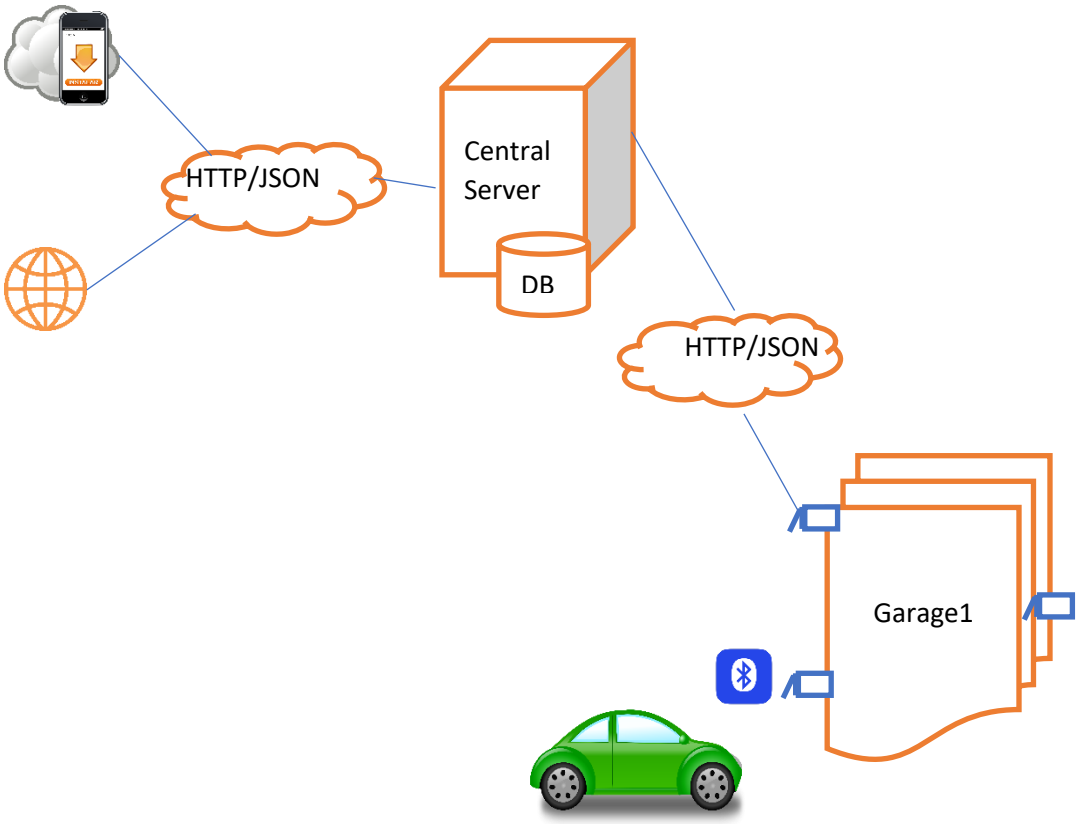
تم اختيار العمارة النظيفة لتطوير برنامج خلية حجز مواقف السيارات للأسباب التالية:

- تعقيد المشروع المعتدل : يتطلب النظام التكامل مع تقنيات متعددة (بلوتوث، NFC)، مما يجعل فصل المكونات أمراً حاسماً لتقليل التعقيد.
- قابلية الاختبار: عزل طبقة المجال يسمح باختبار منطق الأعمال (مثل إدارة الرموز والحجوزات) بشكل مستقل، مما يضمن جودة عالية للكود.
- القدرة على التوسع : يدعم النظام إضافة ميزات مستقبلية (مثل طرق دفع جديدة أو فلترة متقدمة) دون الحاجة لإعادة هيكلة الكود.
- سهولة الصيانة : يسمح بتحديث أو تغيير مكونات مثل واجهات البرمجة (APIs) أو مكتبات الاتصال دون التأثير على واجهة المستخدم أو منطق الأعمال.
- الأداء : فصل الأوامر عن الاستعلامات يعزز أداء معالجة البيانات.

على عكس أنماط التصميم MVC و MVVM، تقدم العمارة النظيفة استقلالية أكبر بين الطبقات، مما يجعلها مثالية لتطبيق معقد مثل نظام حجز مواقف السيارات. بينما يعد نمط BLoC قوياً للتطبيقات التفاعلية، إلا أنه قد يقدم تعقيدات غير ضرورية مقارنة بالهيكل المنظم للعمارة النظيفة.

تصميم النظام

يعرض هذا الفصل القرارات التصميمية التي بني من خلالها النظام.



الفصل الرابع

تنفيذ النظام

يَعرض هذا الفصل القرارات التصميمية التي بني من خلالها النظام

1.4- مقدمة

يكشف هذا القسم تفاصيل تنفيذ نظام حجز مواقف السيارات، يركز على شرح وظائف المكونات الأساسية للنظام، بما في ذلك التطبيق المحمول، ومنصة الإدارة الإلكترونية، والخادم المركزي، وأجهزة استشعار الدخول/الخروج، مع توضيح تفاعلاتها ومبادئ التصميم التي توجه تطويرها. يستند التصميم إلى مبادئ الأنظمة الموزعة لضمان قابلية التوسع والمرونة والأداء العالي.

تشمل مبادئ التصميم الرئيسية فصل الاهتمامات، حيث يتم تقسيم وظائف النظام إلى خدمات مستقلة مثل إدارة الحجوزات والتحقق من الهوية ومعالجة الدفعات، مما يقلل التعقيد ويسهل الصيانة. بالإضافة إلى ذلك، تم اعتماد فصل الأوامر عن الاستعلامات لتعزيز كفاءة معالجة البيانات واسترجاعها بشكل مستقل. يتم دعم التحقق من الهوية بتقنيات البلوتوث والاتصال قريب المدى(NFC) ، مع توليد رموز تشفير (Tokens) عبر بروتوكول كيربيروس لضمان أمن وسلامة العمليات .

2.4- الأدوات المستخدمة

Flutter: لتطوير تطبيق الموبايل المتوافق مع Android(+10) وIOS(+13) ، يوفر واجهة مستخدم سلسة لعرض المواقف، الحجز، والدفع.

ASP.NET: لتطوير الخادم المركزي وموقع الإدارة، يدعم معالجة الطلبات عبر HTTP/JSON وإدارة الحجوزات والمستخدمين.

SQLSERVER: قاعدة بيانات لتخزين بيانات المواقف، الحجوزات، والمستخدمين.

Kerberos: لتوليد Token مشفر للتحقق من الحجوزات.

HTML: لتطوير واجهة أساسية لمحاكاة حساسات الدخول/الخروج، تدعم التحقق من Token عبر بلوتوث وNFC.

3.4- تنفيذ النظام

1.3.4 تطبيق الفلاتر

تم تطوير تطبيق الموبايل باستخدام Flutter بناءً على Clean Architecture لضمان فصل الاهتمامات، تسهيل الصيانة، وقابلية التوسع والاختبار

1.1.3.4تنفيذ التقنيات داخل المشروع

تم تقسيم النظام إلى الوحدات التالية لتنفيذ وظائفه بكفاءة:

المدفوعات :

- تمكين المستخدمين من الدفع عبر محفظة رقمية مدمجة بالتطبيق.
- عرض تكاليف الحجز قبل التأكيد، مع خيارات استرداد للإلغاء خارج الفترة الحرجة (±15 دقيقة).

الحجز ومواقف السيارات :

- عرض المواقف المتاحة .
- تمكين المستخدمين من تحديد أوقات الوصول والمغادرة، مع خيار الإلغاء.
- تأمين عمليات الحجز باستخدام بروتوكول (Kerberos) لتوليد رمز تشفير (Token) لكل حجز.
- تخصيص 3 أماكن احتياطية لمنع فترات الانتظار.

الإشعارات :

- إرسال تنبيهات فورية للمستخدمين عبر لتزكير بموعد الحجز قبل ربع ساعة .

المصادقة (Auth) :

- اشتراط تسجيل الدخول و انشاء حساب عبر البريد الإلكتروني وكلمة المرور.

بوابة البلوتوث والـ NFC :

- التحقق من صلاحية الحجز عند مدخل المواقف عبر البلوتوث والـ NFC.
- تفعيل الاتصال تلقائياً عند الموقف .

2.3.4-التقنيات المستخدمة للاتصال بين الحساس والتطبيق

1.2.2.4– البلوتوث

تقنية Bluetooth هي معيار اتصال لاسلكي قصير المدى يستخدم موجات الراديو لنقل البيانات بين الأجهزة بكفاءة. تعتمد تقنية Bluetooth Low Energy (BLE) في النظام لتقليل استهلاك الطاقة، مما يتيح التواصل السريع بين تطبيق المستخدم وحساسات المواقف للتحقق من الحجز وفتح الباب .

2.2.2.4– بروتوكول NFC

1.2.2.2.4– مقدمة

تقنية الاتصال قريب المد (NFC) تتيح التواصل اللاسلكي بين الأجهزة على مسافات قصيرة جدًا (بضعة سنتيمترات). في النظام، تُستخدم NFC للتحقق السريع من الـ Token عبر تقريب الهاتف من قارئ في الموقف، مما يدعم الوصول الآمن والدفع الإلكتروني .

2.2.2.2.4– آلية عمل NFC و التردد الخاص به

تعمل تقنية NFC على تردد 13.56 ميجاهرتز (MHz) المعياري عالمياً للاتصالات قصيرة المدى، حيث يتشكل مجال كهرومغناطيسي عند تقارب جهازين مدعومين (عادةً ضمن 4 سم) يمكن من خلاله نقل الطاقة والبيانات بشكل متزامن دون حاجة لإقران معقد كالبلوتوث، إذ يبدأ الاتصال فور تقارب الأجهزة. توفر هذه التقنية اتصالاً فورياً دون ضبط يدوي، واستهلاكاً منخفضاً للطاقة، وتبادلاً آمناً للبيانات المصمم خصيصاً للمعاملات القريبة، مع وظيفة مزدوجة تمكّنها من نقل البيانات وتغذية

الأجهزة السلبية كبطاقات NFC. تجعل آلية "اللمس للاتصال" هذه التقنية مثالية للمدفوعات اللاتلامسية، أنظمة التحكم بالوصول، وإقران أجهزة إنترنت الأشياء.

3.2.2.2.4– أوضاع تشغيل NFC

تعمل تقنية NFC بثلاثة أوضاع أساسية تختلف حسب نوع الأجهزة ووظيفة الاتصال:

- الوضع النشط (Active Mode)

يقوم كلا الجهازين في هذا الوضع بتوليد موجات كهرومغناطيسية بشكل متبادل لإرسال واستقبال البيانات.

يُستخدم هذا النمط عادةً لتبادل البيانات بين أجهزة متكافئة مثل الهواتف الذكية.

- الوضع السلبي (Passive Mode)

في هذا الوضع، يبث أحد الأجهزة (مثل الهاتف) موجات كهرومغناطيسية، بينما يعمل الجهاز الآخر (مثل بطاقة NFC أو وسوم RFID) كمستقبل فقط دون توليد موجات.

يُطبق في أنظمة الدخول الذكي، المدفوعات الإلكترونية، وبطاقات المواصلات حيث يكون القارئ نشطاً والبطاقة سلبية.

- الوضع التبادلي (Peer-to-Peer Mode)

يُمكّن هذا الوضع تبادل البيانات بين جهازين متساويين (مثل مشاركة جهات الاتصال أو الملفات بين الهواتف).

تتبادل أدوار الإرسال والاستقبال تلقائياً بين الطرفين، مما يجعله الأكثر تطوراً في التطبيقات اليومية.

3.2.2.2.4– استخدام تقنية NFC في المشروع

صُمم نظام آلي في هذا المشروع لفتح بوابة الكراج تلقائياً عند اقتراب الزبائن من الحساس بأجهزتهم المحمولة المدعومة بتقنية NFC. يعمل النظام كالتالي:

- يعمل الحساس في وضع القارئ السلبي ، منتظراً اقتراب جهاز المستخدم.

- عند الاقتراب، يقرأ الحساس بيانات الحجز (التوكن الخاص بالحجز) من الهاتف.
- ينقل الحساس هذه البيانات إلى الخادم .
- يتحقق الحساس من وجود التوكن الخاص بالحجز من الخادم
- إذا تم التحقق، يرسل الخادم إشارة فتح البوابة للحساس .

لماذا تم اختيار الوضع السلبي في المشروع

تم اختيار الوضع السلبي (Passive Mode) لمشروع "نظام حجز مواقف السيارات" للأسباب التالية:

- ملاءمة الدور الوظيفي :

يعمل الحساس المثبت عند البوابة كـ"قارئ" يصدر الموجات ويستقبل البيانات من الهاتف دون تفاعل ثنائي.

- تعزيز الأمان :

تبقى السيطرة على عملية الاتصال حصراً مع الحساس، مما يمنع التبادل العشوائي للبيانات من الهاتف ويقلل مخاطر الاختراق.

- ترشيد استهلاك طاقة الهاتف :

ينقل الهاتف البيانات فقط عند الاقتراب من الحساس **دون إصدار موجات**، مما يحافظ على بطارية المستخدم.

لماذا تم استبعاد الأوضاع الأخرى:

- الوضع النشط (Active Mode) :

يتطلب طاقة من كلا الجهازين (الحساس والهاتف) مما لا يتناسب مع طبيعة الحساسات الثابتة.

- الوضع التبادلي (Peer-to-Peer) :

يحتاج تدفق بيانات ثنائي الاتجاه، بينما النظام يتطلب تدفقاً أحادي الاتجاه (من الهاتف إلى القارئ فقط).

3.2.2.4– مقارنة بين NFC و BLE

المعيار	BLE	NFC
المدى	متوسط (0–10متر)	قصير جداً (1–4 سنتيمتر)
سرعة النقل	متوسطة (1–2 ميجابيت /ثانية)	بطيئة (424 كيلوبت / ثانية)
وقت الاتصال	فوري (0.1 ثانية)	يحتاج اقتران (2–5 ثانية)
الأمان	عالي (بسبب القرب المادي)	متوسط
التوافق	محدود (الأجهزة الحديثة فقط)	واسع (معظم الأجهزة منذ 2012)

4.2.2.4– المحاكاة في التطبيق

خلال تنفيذ التطبيق، واجهنا صعوبة بسبب عدم توافر حساسات فعلية لاختبار تقنيتي Bluetooth وNFC . للتغلب على ذلك، تم إنشاء صفحة ويب بسيطة لمحاكاة دور الحساس، حيث تتيح اختبار عمليات التحقق من الـ Token وفتح البوابات افتراضيًا. ومع ذلك، ظهرت مشكلة إضافية تتعلق بدعم NFC في الأجهزة المستخدمة، حيث اكتُشف أن الجوال يدعم NFC ككاتب (Writer) فقط وليس كقارئ (Reader) ، مما حد من إمكانية التحقق المباشر من الـ Token عبر NFC .

2.3.4 الخادم المركزي

الخادم الخلفي (Backend _ASP.NET Core Web API): يمثل السيرفر المركزي للنظام، يستضيف منطق العمل، ويدير قاعدة البيانات، ويوفر واجهات برمجية (APIs) لجميع العملاء.

إطار عمل .NET Core.

هو إطار عمل (Framework) مجاني، مفتوح المصدر(Open-source) ، ومتعدد المنصات (Cross-platform) تم تطويره بواسطة .Microsoft يُمكّن المطورين من بناء أنواع مختلفة من التطبيقات، بما في ذلك تطبيقات الويب، والخدمات الصغيرة (Microservices)، وتطبيقات سطح المكتب، وتطبيقات الموبايل، وإنترنت الأشياء.

إطار عمل ASP.NET WEP API

ASP.NET Web APIهو جزء من إطار عملASP.NET ، وهو مُصمم خصيصاً لبناء خدمات HTTP (واجهات برمجية) تصل البيانات والوظائف عبر الشبكة. يُستخدم بشكل أساسي لإنشاء خدماتRESTful ، والتي تُعتبر العمود الفقري لتطبيقات الموبايل والويب الحديثة التي تتصل بخوادم البيانات.

و يقدم ميزات :

1-بناء خدمات RESTful بسهولة:

هو مُصمم خصيصاً لدعم مبادئ REST (Representational State Transfer)، مما يتيح لك بناء APIs نظيفة ومنظمة باستخدام أفعال HTTP (GET, POST, PUT, DELETE) للتعامل مع الموارد.

2-دعم كامل لبروتوكول HTTP:

3-تفاوض المحتوى (Content Negotiation):

يستطيع ASP.NET Web API التفاوض تلقائياً مع العميل لتحديد أفضل تنسيق للبيانات (مثل JSON أو XML) لإرجاع الاستجابة، بناءً على رأس Accept في طلب العميل. JSON هو التنسيق الافتراضي والشائع.

4-نظام التوجيه المرن (Flexible Routing):

يُمكنك من تحديد مسارات (URLs) مخصصة لنقاط النهاية (Endpoints) الخاصة بك بسهولة، باستخدام التوجيه المستند إلى السمات ([Route]) أو التوجيه التقليدي.

5-ربط النموذج والتحقق من الصحة (Model Binding & Validation):

يقوم تلقائياً بربط بيانات الطلب الواردة (سواء من جسم الطلب JSON، Query String، أو مسار الـURL) بكائنات C# (DTOs).

6-المصادقة والتفويض (Authentication & Authorization):

يتكامل بسلاسة مع نظام المصادقة والتفويض القوي في ASP.NET Core (مثل ASP.NET Core Identity، JWT Bearer Authentication، وAuthorization Policies) لحماية الAPIs الخاصة بك.

7-حقن الاعتماديات (Dependency Injection - DI):

يحتوي على حاوية DI مدمجة، مما يسهل إدارة تبعيات الخدمات ويجعل الكود أكثر قابلية للاختبار والصيانة.

8-مرونة الاستضافة:

يمكن استضافة تطبيقات ASP.NET Web API في بيئات مختلفة، بما في ذلك IIS، Kestrel (خادم الويب المدمج)، أو كخدمات مصغرة (Microservices) في حاويات (Containers) مثل Docker.

9-تكامل Swagger/OpenAPI:

يتكامل بسهولة مع مكتبات مثل Swashbuckle لتوليد توثيق API تفاعلي تلقائياً (Swagger UI)، مما يسهل على المطورين فهم واختبار الAPIs.

ASP.NET Core MVC

هو إطار عمل (Framework) يُستخدم لبناء تطبيقات الويب التقليدية (Traditional Web Applications) التي تقوم بتقديم صفحات HTML (Views) من جانب الخادم (Server-Side Rendering) إلى المتصفح. إنه جزء أساسي من إطار عمل ASP.NET Core.

و هو تطبيق لنمط تصميم Model-View-Controller (MVC) ضمن بيئة ASP.NET Core، ويتميز بالنقاط التالية:

نمط Model-View-Controller (MVC):

Model (النموذج): يمثل البيانات ومنطق العمل الخاص بالتطبيق. هو الطبقة التي تتفاعل مع قاعدة البيانات (عبر Entity Framework Core في مشروعك) وتُطبق قواعد العمل.

View (العرض): يمثل واجهة المستخدم (User Interface) التي يراها المستخدم في المتصفح. في ASP.NET Core MVC، يتم بناء الViews باستخدام Razor syntax، الذي يمزج كود C# مع HTML لإنشاء صفحات ديناميكية.

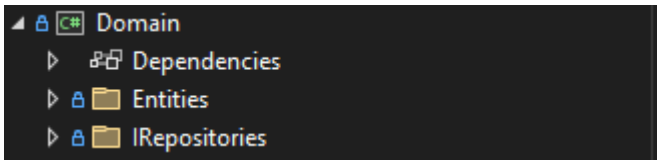
Controller (المتحكم): يعالج طلبات المستخدمين (طلبات HTTP). هو يستقبل المدخلات، يتفاعل مع الModel لتنفيذ منطق العمل، ثم يحدد الView الذي سيتم عرضه للمستخدم.

1.2.3.4 – تقسيم الطبقات ضمن الخادم المركزي :

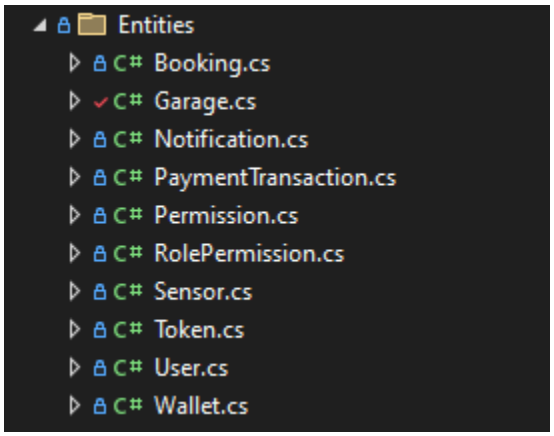
يوضح الشكل السابق طبقات عمل التطبيق، حيث تم احترام منهجية البنية المعمارية النظيفة (clean architecture)و التصميم المقاد بالمجال (domain driven design) في تنجز هذه الطبقات .

1.1.2.3.4– طبقة المجال Domain layer :

هي تمثل المنطق الأساسي للعمل، والقواعد، والكيانات التي تحدد طبيعة نظامك. هذه الطبقة هي الأكثر استقراراً، ولا يجب أن تعتمد على أي طبقات اخرى dependency.



الكيانات هي حجر الزاوية في طبقة Domain، فهي تمثل المفاهيم الأساسية، البيانات، والقواعد التي تحكم مجال العمل.



الواجهات العامة للمستودعات (IRepositories):

تمثل عقوداً (Contracts) تُحدد العمليات الأساسية التي يمكن إجراؤها على البيانات المتعلقة بكيانات المجال. و هي لا تحتوي على أي منطق تطبيقي أو تفاصيل تنفيذية،

و لها دور أساسي في

1-تجريد الوصول الى البيانات ،حيث تخفي هذه الواجهات التفاصيل المعقدة حول كيفية تخزين البيانات واسترجاعها. بالنسبة للخدمات في طبقة Application (التي تستخدم هذه الواجهات)، لا يهم إذا كانت البيانات تُخزن في SQL Server، MongoDB، ملفات، أو حتى في الذاكرة. يجعل منطق عمل حجز المواقف (مثل إنشاء حجز، تعديل كراج) مستقلاً عن تكنولوجيا قاعدة البيانات. في حال مستقبلاً تم تغيير SQL Server إلى قاعدة بيانات أخرى، ستحتاج فقط لتعديل طبقة Infrastructure (حيث توجد تطبيقات المستودعات الفعلية) دون المساس بطبقتي Application وDomain.

2-استقلالية طبقة المجال مما يجعلها الطبقة الداخلية التي لا تعتمد على أي طبقة خارجية. هذا يحافظ على نظافة طبقة المجال وتجريدها من تفاصيل التنفيذ.

3- فصل الارتباطات

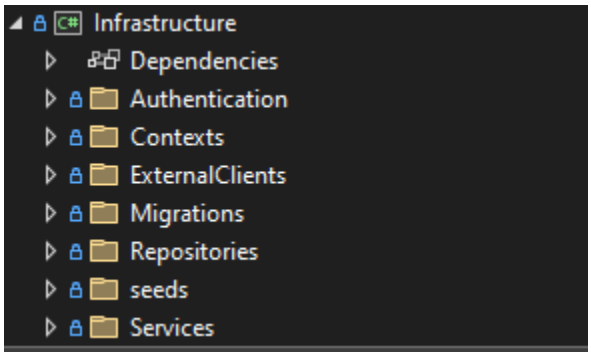
يقلل هذا التجريد من الارتباط بين طبقة Application(الخدمات) وطبقة Infrastructure(تنفيذ قاعدة البيانات). طبقة Applicationتعتمد فقط على الواجهات (IRepository<T>) وليس على الكلاسات الفعلية التي تُنفذ الوصول إلى قاعدة البيانات.

2.1.2.3.4- طبقة البنية التحتية Infrastructure layer :

هذه الطبقة حيوية جداً، فهي التي تُحوّل التصميمات المجردة في طبقات Domain وApplication إلى واقع عملي وقابل للتشغيل.

في بنية Clean Architecture، تُعد طبقة Infrastructure هي الطبقة الخارجية (Outermost Layer). مهمتها الأساسية هي التعامل مع جميع التفاصيل التقنية الخارجية، مثل الوصول إلى قاعدة البيانات.

تمثل الجزء العملي والتنفيذي من الBackend ، حيث تحتوي على جميع الكلاسات والتكوينات التي تربط منطق العمل الخاص بك بالتقنيات والأنظمة الخارجية.،حيث تملك هذه الطبقة التبعية لطبقة التطبيق



توضح الفقرات التالية ما تم تنجيزه ضمن طبقة البنية االتحتية

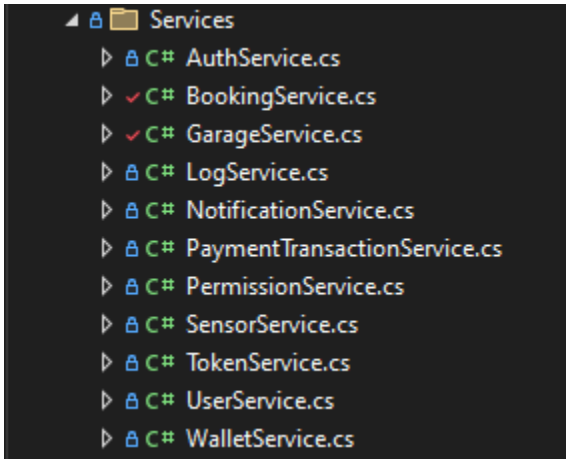
هي المسؤولة عن التواصل المباشر مع قاعدة البيانات (SQL Server) باستخدام Entity Framework Core (EF Core).
هي تُنفذ استعلامات CRUD، وتُحوّل بيانات الكيانات إلى/من سجلات قاعدة البيانات.

- **تنفيذ المستودعات Repositories**

هي المسؤولة عن التواصل المباشر مع قاعدة البيانات (SQL Server) باستخدام Entity Framework Core (EF Core).
هي تُنفذ استعلامات CRUD، وتُحوّل بيانات الكيانات إلى/من سجلات قاعدة البيانات.

- **تنفيذ الخدمات**

تحتوي على التطبيقات الفعلية لواجهات الخدمات التي تُعرف في طبقة Application



هي النقطة التي تُطبق فيها قواعد العمل المعقدة. هذه الخدمات تستخدم المستودعات (Repositories) لإجراء عمليات على البيانات، وقد تستدعي خدمات أخرى لتنسيق العمليات المعقدة.

- **سياق قاعدة البيانات context**

AppDbContextهو كيان مركزي في المشروع ، يُعد بمثابة الجسر الحيوي الذي يربط التطبيق عبر Entity Framework Core)بقاعدة بيانات .SQL Server هو يمثل "جلسة (Session) "مع قاعدة البيانات أو "وحدة عمل (Unit of Work) "

و هو المسؤول عن تحويل الكيانات الى جداول علائقية و يعرف مخطط قاعدة البيانات ، و يقوم تلقائيا بتتبع أي تعديلات على الكيانات.

هو الأساس الذي تعتمد عليه ميزة الMigrations في EF Core. عندما تُنشئ Migration جديدة، فإن EF Core يستخدم AppDbContext لفهم نموذجك الحالي ومقارنته بالنموذج السابق لإنشاء أوامر SQL اللازمة لتعديل مخطط قاعدة البيانات.

حيث تم اعتماد منهجية Code First لتخزين المعطيات في قاعدة المعطيات.

- **ادارة الأدوار**

ضمن النظام المقترح يوجد ثلاثة أدوار أساسية :

الزبون : يقوم بحجز موقف السيارات و عملية الدفع

مدير النظام :يقوم باضافة الكراجات وادارتها و و ادارة المستخدمين و ادارة المحافظ للمستخدمين

مدير موقف السيارات :يعدل على مواصفات الكراج الخاص فيه و ادارة المحافظ للمستخدمين في نفس الموقف

كل دور يمثل مجموعة محددة مسبقاً من المسؤوليات والصلاحيات

5.2.1.5 المصادقة و التفويض

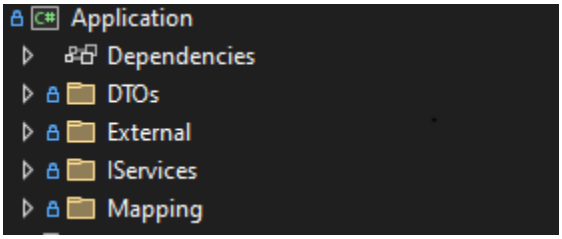
تم تنجيز المصادقة من خلال رموز (JWT), حيث ان لكل مستخدم دور خاص فيه ,و لكل دور سماحية (permission)
واحدة او أكثر.

يتم التفويض بعد ذلك من خلال التأكد من ان هذا المستخدم يمتلك د الدور لطلب المعلومات من النظام ، حيث أن جميع الطلبات تمر عبر middleware .

3.1.2.3.4– طبقة التطبيق :

طبقة Application تُعد الطبقة الثانية في بنية Clean Architecture (بعد طبقة Domain). هي مسؤولة عن تنفيذ قواعد العمل الخاصة بالتطبيق (Application-Specific Business Rules)، وتنسيق تدفق البيانات، وتعريف حالات الاستخدام (Use Cases) التي يقدمها النظام. هذه الطبقة تعتمد على طبقة Domain، ولكنها مستقلة عن طبقات Infrastructure وPresentation.

تُشكل طبقة Application الواجهة البرمجية (API) لمنطق عمل تطبيقك، حيث تُعرّف العمليات التي يمكن أن يقوم بها النظام وتُحدد البيانات التي يتم تبادلها.



كائنات نقل البيانات (DTOs - Data Transfer Objects):

تحتوي على كيانات DTO التي تُستخدم لنقل البيانات بين طبقات النظام، وتحديداً بين الBackend والFrontend (أو أي عميل آخر). تُستخدم هذه الDTOs لـ:

1. تقليل حجم البيانات: إرسال فقط البيانات الضرورية لعملية معينة.
2. تخصيص البيانات: تهيئة البيانات لتناسب متطلبات واجهة المستخدم أو متطلبات API محددة.
3. فصل الكيانات عن العرض/الاستقبال: منع الكيانات من التعرض مباشرة للعملاء، مما يحمي تصميم طبقة Domain.
4. التحقق من صحة المدخلات

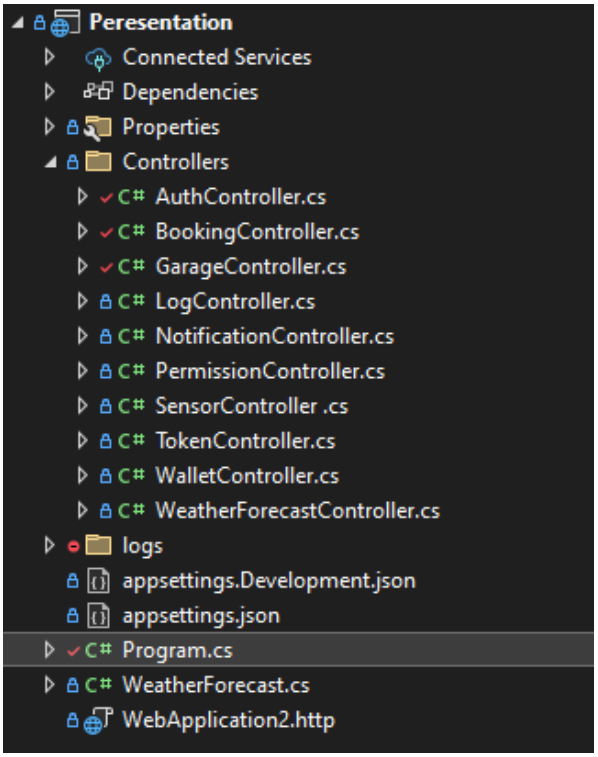
AutoMapper :

المحتوى: يحتوي هذا المجلد على كلاسات Profile التي تُحدد قواعد التحويل بين الكيانات (Entities)

والDTOs (Data Transfer Objects).

4.1.2.3.4– طبقة Presentation :

هي الطبقة الخارجية (Outermost Layer) في بنية Clean Architecture. إنها النقطة التي يتفاعل معها المستخدمون النهائيون (عبر تطبيق الموبايل أو متصفح الويب) والباب الذي تدخل منه الطلبات إلى النظام. مسؤوليتها الرئيسية هي التعامل مع تفاصيل واجهة المستخدم (UI) والبروتوكولات (مثل HTTP).



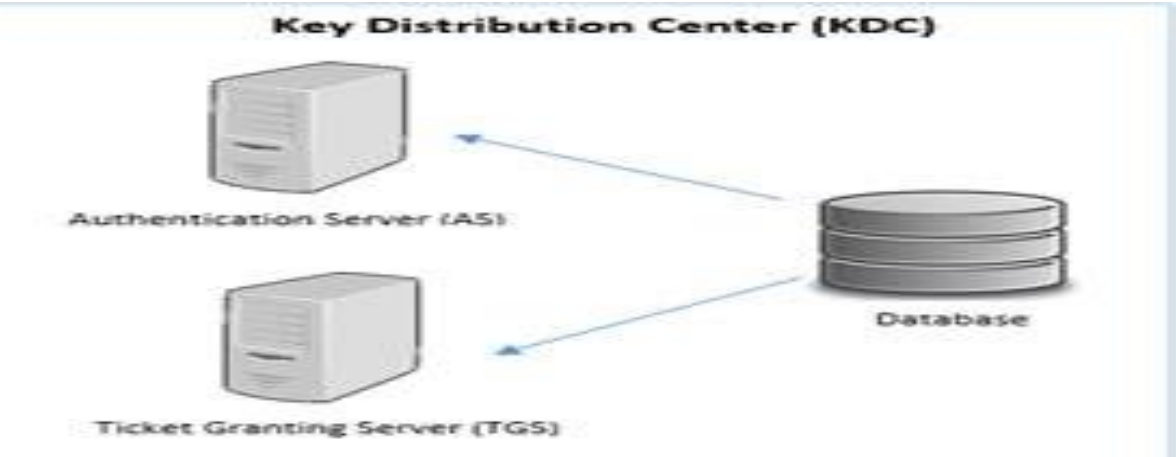
المتحكمات (Controllers):

الدور: هي نقطة الدخول الرئيسية لطلبات HTTP. تستقبل هذه الطلبات، تُفسر مدخلات المستخدم، وتستدعي الدوال المناسبة من طبقة Application (الخدمات). ثم تُنسق الردود لإرسالها إلى العميل.

و هي من نوع متحكمات Web API: (ترث من ControllerBase، وتُوضع عليها [ApiController] و[Route("api/[controller]")]).

المسؤولية: تُعالج طلبات HTTP التي ترجع بيانات (غالباً JSON) لتطبيق الموبايل (Flutter App) أو أي عميل آخر يستهلك API (مثلاً Postman، Swagger UI).

3.3.4 تحقيق مفهوم بروتوكول Kerberos على مستوى الخادم



لمكونات الأساسية في الBackend لتحقيق Kerberos-like:

AuthService: يلعب دور الAS لإصدار الJWT (TGT).

TokenService: يلعب دور الTGS (إصدار تذاكر الوصول للحساس) ودور الموثّق المركزي لتلك التذاكر.

Token Entity: هيكل بيانات "التذاكر" نفسها.

SensorService: يستقبل تقارير الحساسات التي تُقدم معلومات بعد استخدام التذاكر.

قاعدة البيانات (SQL Server): تخزن جميع المفاتيح السرية (كلمات المرور المشفرة)، التذاكر الصادرة، وClaims الصلاحيات.

تم تحقيق المفهوم من خلال تكييف أدوار الخادم المركزي (Kerberos (KDC لكي تتناسب مع بنية تطبيقك ASP.NET Core Web API.الBackend الخاص بك يلعب دور مركز توزيع المفاتيح(KDC) ، حيث يتولى مسؤولية إصدار "التذاكر" والتحقق منها. و ذلك عن طريق

1. المصادقة الأولية (AS - Authentication Server Role)

خادم المصادقة (AS) يقوم بمصادقة المستخدمين وإصدار تذاكر منح التذاكر (TGTs).

و ذلك عن عند قيام المستخدم بتسجيل الدخول عبر تطبيق الموبايل (POST /api/Auth/login)، يقوم بالتحقق من هوية المستخدم باستخدام UserManager. إذا كانت المصادقة ناجحة، يقوم AuthService بتوليد JWT Token وإرجاعه. هذا الJWT Token يعمل كTGT في نظامنا، حيث يسمح للمستخدم بالوصول إلى APIs الBackend الأخرى.

2. طلب وإصدار "تذكرة الحجز " (TGS - Ticket-Granting Service Role)

: خادم منح التذاكر (TGS) يقوم بإصدار تذاكر خدمة محددة لخدمات معينة.

يتم ذلك بأن تطبيق الموبايل (باستخدام الJWT الذي حصل عليه) يطلب من الBackend "تذكرة الحجز" عبر نقطة نهاية POST /api/Token/create.

: يتم انشاء تذكرة حجز و حفظ هذه التذكرة في قاعدة البيانات . و يتم ارجاع هذه التذكرة الى تطبيق الموبايل.

3. التحقق من "تذكرة الوصول للحساس" (Service Validation Role)

خادم الخدمة (Application Server) يقوم بالتحقق من صحة تذكرة الخدمة.

الحساس الفيزيائي يستقبل "تذكرة خدمة الحجز" من تطبيق الموبايل عبر Bluetooth.

يقوم الحساس بالاتصال بالBackend (نقطة نهاية POST /api/Token/validate-sensor-access)

يتم البحث عن التوكن في قاعدة البيانات. و االتحقق من صلاحيته الزمنية (ValidTo > DateTime.UtcNow).

و ضمان استحدامها لمرة واحدة.

الناتج: الBackend يؤكد للحساس ما إذا كانت التذكرة صالحة أم لا.

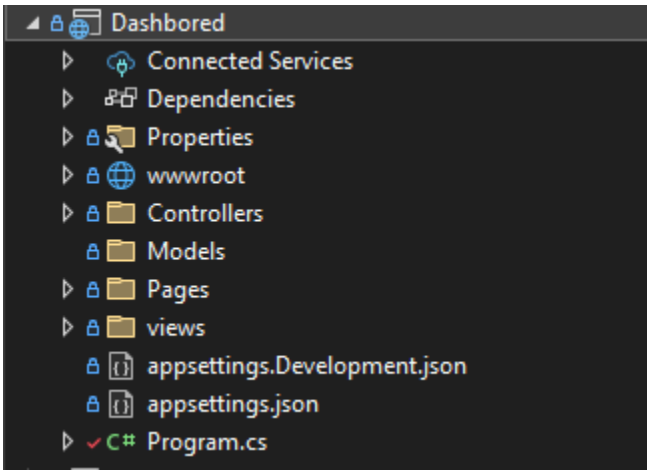
4. تحديثات الحالة بعد منح الوصول (Reporting Role)

: خادم الخدمة (الحساس) يمنح الوصول ثم قد يبلغ عن الحالة.

بعد أن يؤكد الBackend صلاحية التذكرة للحساس، ويقوم الحساس بعمل معين (مثل فتح بوابة)، فإن الحساس يرسل تقريراً بحدوث هذا الحدث إلى الBackend عبر نقطة نهاية POST /api/Sensor/report-status.

بهذه الطريقة، يُحقق الـ Backendالخاص بك مفهومKerberos-like ، حيث يُصبح هو مركز الثقة الذي يُصدر ويتحقق من "التذاكر" للوصول الآمن إلى الموارد المادية.

4.3.4 تنجيز الواجهة الأمامية .



يمثل مشروع Dashbored تطبيق ASP.NET Core الموحد الذي يجمع بين وظائف الخادم الخلفي (Backend) وواجهة المستخدم الأمامية (Frontend) للوحة تحكم الإدارة. هذا التصميم يتيح إدارة مركزية للخدمات وتقديم صفحات الويب في نفس التطبيق ل مدير النظام .

4.4- مشاكل أثناء التنفيذ

1.4.4 عدم القدرة على ربط تطبيق الفلاتر مع الخادم المركزي

اثناء التنفيذ تم اعتراض العمل بعدم القدرة على ربط التطبيق مع الخادم ما دفعنا إلى استخدام أداة ngrok

و هي أداة سطر أوامر (Command-line tool) مفتوحة المصدر وشائعة جداً تُستخدم لإنشاء أنفاق آمنة (Secure Tunnels)

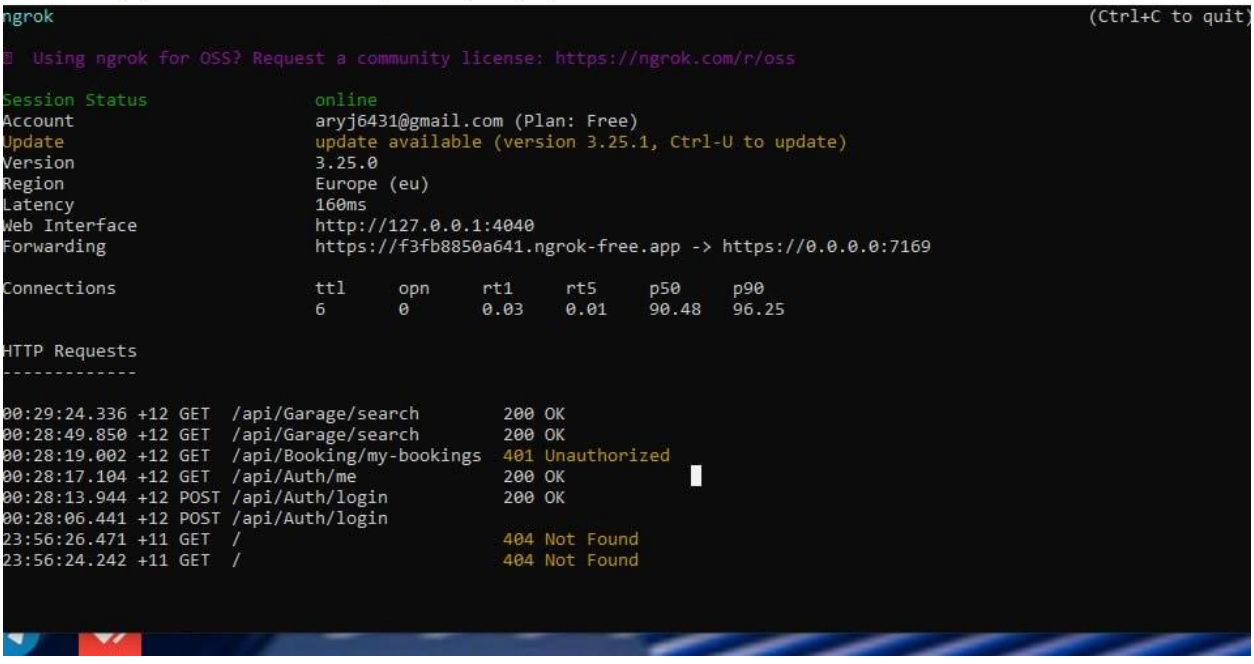
هي خدمة تُمكّننا من كشف (Expose) خادم ويب يعمل محلياً على جهازك إلى الإنترنت العام، حتى لو كان الجهاز خلف جدار حماية (Firewall) أو NAT (Network Address Translation)، ودون الحاجة لتكوين إعدادات معقدة للشبكة أو المنافذ (Ports).
تفيد في :

1-الوصول الخارجي للخادم المحلي:. بما أن تطبيق Flutter يعمل على جهاز موبايل أو محاكي منفصل، ويحتاج للاتصال بالBackend الذي يعمل على جهاز الكمبيوتر اخر ، فإن ngrok يوفر هذا الاتصال بسهولة عبر الإنترنت.

2-اختبار تطبيقات الموبايل والWebhooks: مثالي لاختبار تطبيقات الموبايل، أو الخدمات التي تستخدم الWebhooks (مثل بوابات الدفع التي تحتاج لإرسال إشعارات لBackend محلي).

3-مشاركة العروض التوضيحية: يمكنك مشاركة مشروعك المحلي مع الآخرين بسهولة دون الحاجة لنشره على خادم عام.

4-التشفير (HTTPS): يوفر ngrok عناوين HTTPS عامة، مما يعني أن الاتصال بين تطبيق Flutter والBackend (عبر ngrok) سيكون مشفراً حتى لو كان الBackend المحلي يعمل على HTTP فقط (على الرغم من أن الBackend الخاص بك يعمل على HTTPS أيضاً).



تعرض هذه الصورة عمل ال ngrok الأداة المستخدمة لتوصيل المعلومات بين التطبيق والخادم

ملحق

الاختبارات

يَعرض هذا الفصل بعض الاختبارات التي تمت لحد الان

ذ

الخطا في الصورة ذلك لاننا نستدعي تابع جلب ولم يتم اكماله بعد