Syrian Arab Republic		الجمهورية العربية السورية
Ministry of Higher Education		وزارة التعليم العالي
Syrian Virtual University		الجامعة الافتراضية السورية

تصميم وتنفيذ تطبيق شبكي آمن متعدد الخدمات ومتعدد المنصات على شبكة الإنترنت

Design & Implementation of a Web-based, Multi-service, cross-platform secure network Application

دراسة مقدمة لنيل درجة ماجستير الدراسات العليا في تقانات الويب

A Thesis submitted to the Syrian Virtual University in Partial Fulfillment of the Requirements for the M.Sc Degree in Web Technologies

إعداد: م. عقبة إسماعيل

Submitted by: Eng. Okbah Ismael
(Okbah_100160)

الدكتورة المشرفة: د.سيرا أستور
Supervised by: Dr. Sira Astour

عنوان الرسالة

تصميم وتنفيذ تطبيق شبكي آمن متعدد الخدمات ومتعدد المنصات على شبكة الإنترنت

Designing & Implementing a Web-based Multi-service cross-platform secure network Application

شكر وتقدير

لا يسعني وأنا أوشك على الاقتراب من موعد مناقشة هذا البحث إلا وأن أشكر كل من كان له دور فعال في وصولي إلى هذه النتيجة ...حيث أتقدم بالشكر إلى الأستاذة الدكتورة م. سيرا أستور مديرة برنامج ماجستير تقانات الويب في الجامعة الافتراضية والمشرفة على هذه الرسالة والتي كان لها دور كبير في وقوفها بجانبي منذ بداية التفكير بهذا البحث، وكانت السند بالتوجيهات والإرشاد والنقد البناء الذي كان مرافق للدراسة منذ بدايتها حتى وصولها للوضع النهائي.

كما أتقدم بالشكر للجامعة الافتراضية التي أتاحت لنا فرصة إكمال دراساتنا العليا في بلدنا سوريا بوجود نخبة من الكفاءات العلمية ذات المستوى العالي.

كما أشكر الأساتذة أعضاء لجنة المناقشة لاهتمامهم ولكل من مد يد المساعدة في هذه الرسالة.

شكراً....

الإهداء

إلى أكرم من في الدنيا و أنبل بني البشر، إلى والدي العميد الشهيد يوسف إسماعيل.

إلى من وضعت الجنة تحت قدميها إلى أمي القديسة الغالية

إلى شريكة حياتي وعمري، إلى أولادي الأحباء

إلى شركائي في الحياة منذ نشأتي إلى أخوتي الأصدقاء الصدوقين

إلى كل أصدقائي وأقربائي

أهدي لكم جميعاً هذا العمل المتواضع

الملخص

أثر الإنترنت منذ ظهوره على البشرية في جميع جوانب الحياة العلمية والاقتصادية والاجتماعية والسياسية وحتى الأمنية. لذلك يعتبر العديد من المهنيين في العالم بأن الإنترنت هي أسلوب حياة.
في ضوء الخدمات العالمية الضخمة التي تقدمها شبكة الويب العالمية، ومع التكلفة المنخفضة جداً لهذه الخدمات، فقد أصبح الاتجاه العالمي هو نحو الاستفادة القصوى من خدمات الإنترنت (التجارة الإلكترونية - الإعلانات التجارية - خدمات الاتصالات بمختلف أنواعها...الخ).
ولكن مع كل هذه المزايا الهائلة، فإن التحدي المتمثل في تأمين وحماية المعلومات المتبادلة عبر الإنترنت، لا يزال قيد الدراسة من أجل اعتماد حلول مختلفة تحقق المطلوب منها.
يشمل تأمين المعلومات المتبادلة وحمايتها، حماية المعلومات الشخصية للمستخدمين، وضمان سريتها هو الهدف الذي يجب تحقيقه. لكن التطبيقات المختلفة المستثمرة على الإنترنت ما زالت تواجه مخاطر سرقة الحسابات الشخصية والمالية والتزوير وغسيل الأموال والأنشطة الإرهابية بكافة أشكالها. وبالتالي يتم بذل الكثير من الجهود لحماية الخصوصية وتشفير المعلومات والتحقق من هوية الأشخاص والكيانات التي تستثمر عبر الإنترنت.
يهدف المشروع المقدم في هذه الأطروحة إلى إنشاء تطبيق دردشة Chat App يمكن من خلاله ارسال واستقبال الملفات النصية بين مشتركين أو أكثر، وذلك من خلال اعتمادنا على الأسلوب المعماري REST APIs متعدد الخدمات، مع التأكيد على أن هذا التطبيق سيعمل على مختلف المنصات (الأجهزة الحاسوبية - الأجهزة اللوحية - الهواتف الذكية) مع مراعاة العديد من المعايير الأمنية التي تضمن تحقق هذه العملية بسرية عالية جداً.

على أن يتم تطوير هذا المشروع مستقبلاً بشكل يضمن أيضاً نقل الملفات المختلفة بين عدد من المستخدمين، بالإضافة إلى إمكانية إجراء الاتصالات الهاتفية المرئية VOIP.

الفهرس

الفصل الأول: مقدمة عامة

1-1 مقدمة

منذ ظهور شبكة الإنترنت العالمية في أواخر القرن الماضي ومع تطور تطبيقات الويب فيها، برزت الحاجة لعمل هذه التطبيقات بالشكل المناسب على مختلف أنظمة التشغيل (MS, macOS, Linux) وكذلك مختلف المتصفحات أيضاً، ومع ظهور الأجهزة اللوحية Tabs والهواتف الذكية Smart Phones أصبح هناك ضرورة ملحة لتشغيل تطبيقات الويب على مختلف هذه المنصات.

وتعتبر البرمجيات متعدد المنصات في مجال الحوسبة عبارة عن برامج مستقلة عن نظام التشغيل الأساسي الخاص بالجهاز المستخدم، حيث يتم تنفيذ هذه البرمجيات على منصات حوسبة متعددة (الحواسب العادية - الحواسب اللوحية - الهواتف الذكية).

ويمكن تقسيم البرامج عبر الأنظمة الأساسية إلى نوعين رئيسيين، يتطلب أحدهما بناءً أو تجميعاً فردياً لكل نظام أساسي يدعمه، أما النوع الثاني فيمكن تشغيله مباشرةً على أي منصة دون إعداد خاص مسبق.

على سبيل المثال البرامج المكتوبة باللغة المفسرة interpreted language أو المكتوبة بإحدى لغات البرمجة المجمعة مسبقاً precompiled bytecode وهذه البرامج يمكن استخدامها مباشرةً أو من خلال مكونات شائعة أو قياسية لجميع المنصات.

2-1 مشكلة البحث

مع كل التطور التقني الهائل ومع كل الوسائل والتسهيلات التي تقدمها شبكة الإنترنت، لا تزال بلدان العالم الثالث لا تستفيد منها إلا بالحدود الدنيا. ومن ضمن هذه الخدمات إمكانية تحقيق التواصل بين الأشخاص والجهات الحكومية والخاصة، الشيء الذي سيوفر بدوره الكثير من التكاليف المالية على الدولة والمجتمع، وكمثال عن ذلك التوفير المادي الكبير من خلال تقليص الاستهلاك الورقي الذي تتكبده الحكومات في تلك البلدان، إضافةً إلى تأمين الاتصال الآمن والسريع بين مؤسساتها المختلفة.

ومن هنا ظهرت الضرورة الملحة لاستثمار كافة الخدمات والتطبيقات والتسهيلات التي تقدمها شبكة الإنترنت، الشيء الذي سيؤدي بدوره لرفع مستوى الأداء في الدولة والمجتمع بكلا القطاعين العام والخاص، وسيوفر الوقت والجهد والمال كذلك، مع ضمان الحفاظ على السرية والخصوصية.

3-1 الهدف من المشروع

إنشاء تطبيق محادثة Chat App نستخدم من خلاله أهم الأفكار المتعلقة بتطبيقات الويب الحديثة متعددة المنصات ومتعددة الخدمات، مع تطبيق أهم وسائل الحماية الأمنية المستعملة في أيامنا الحالية من قبل أهم المواقع والشركات. على أن يكون هذا التطبيق منصة تطوير مستقبلية قابلة للاستخدام العملي في كلا القطاعين العام والخاص من حيث نقل وتبادل المعلومات الآمن عبر شبكة الانترنت.
وقد تم استخدام لغة البرمجة JavaScript في كلا الواجهات الأمامية React والخلفية NodeJS لكامل المشروع، كما اعتمدنا قاعدة البيانات MongoDB

1-4 التطبيقات العملية

يمكن الاستفادة من فكرة المشروع مستقبلاً من خلال إرسال الرسائل والملفات الإلكترونية كبديل عن الوسائط الورقية العادية، الشيء الذي يضمن توفير مالي كبير جداً، ويضمن توفير الوقت أيضاً نظراً للسرعة العالية بنقل البيانات إلكترونياً.
ومع وجود الكثير من التطبيقات العالمية الشهيرة الخاصة بالدردشة وفي مقدمتها تطبيق Messenger وWhatsApp إلا أن الفكرة الرئيسية التي ننشدها من خلال مشروعنا، هي ضرورة وجود تطبيقات محلية سورية تضمن الخصوصية والسرية لتبادل المعلومات دون تدخل أو اختراق من قبل جهات أو دول خارجية.

الفصل الثاني: الدراسة النظرية

2-1 تطبيقات الويب متعددة الخدمات العاملة على المنصات المختلفة

تعدد المنصات مصطلح يستخدم في علم الحوسبة يشير إلى برامج الحاسوب أو أنظمة التشغيل أو لغات الكمبيوتر أو لغات البرمجة وتطبيقاتها التي بإمكانها العمل على عدة منصات حاسوبية، هناك نوعان رئيسيان من البرمجيات متعددة المنصات، الأول يستلزم بناءه لكل منصة على حدا كي يعمل عليها (مثل برنامج مكتوب بلغة مترجمة مثل C) والثاني بإمكانه العمل مباشرةً على أي منصة تدعمه (مثل البرمجيات المكتوبة بلغات مفسرة مثل PERL Java JavaScript Python). ومصطلح المنصة يشير إلى عدة معاني:

- منصات الأجهزة: تشير إلى معمارية ما للحاسوب أو المعالج مثل معمارية x86 وx64 مثل هذه المعمارية يمكن أن يعمل عليها أكثر من نظام تشغيل (MS-DOS / IOS-10).
- المنصات البرمجية: في العادة يشار بهذا المسمى إلى نظم التشغيل التي تعمل على منصات الأجهزة وتتواصل معها، وتعمل بشكل أساسي كوسيط بين تلك المنصات المستخدم، وتقوم بتزويد المستخدم بالعديد من الخدمات وتسهل عليه عمل العديد من المهمات، أيضاً تقوم بتنظيم وإدارة موارد المنصة التي تعمل عليها مثل إدارة المعالج والذاكرة. ويشار أحياناً بالمنصات البرمجية إلى بيئة البرمجة التي توفر الأدوات لتطوير التطبيقات والبرمجيات من أمثلتها جافا. ومن المنصات الأخرى، تلك التي توفر بيئة عمل للتطبيقات ومن أهم أمثلتها منصة فيس بوك وغيرها من المنصات التي تعمل على الإنترنت.

ولا بد لنا أن نذكر بأن البرمجيات متعددة المنصات هي التي تكون قادرة على العمل بكامل وظائفها الأساسية على أكثر من معمارية حاسوبية أو نظام تشغيل.

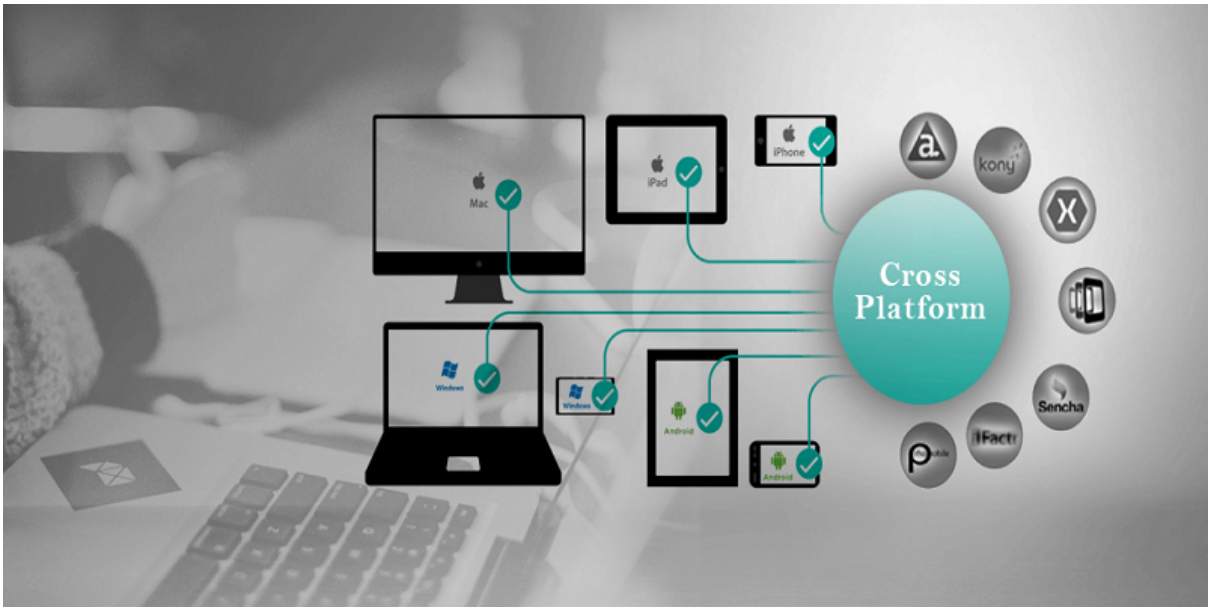
يشار في العادة إلى تطبيقات الويب على أنها متعددة المنصات، بسبب حقيقة أنه يمكن الوصول إليها من عدة أنواع من المتصفحات داخل عدة أنواع من نظم التشغيل، مثل هذه التطبيقات توظف بشكل عام نموذج العميل/الخادم وتختلف التطبيقات من حيث التعقيد والوظائف .

تطبيقات الويب الأساسية تقوم بأداء كل أو معظم العمليات والمعالجات على الخادم server وتمرر النتيجة إلى متصفح الويب الخاص بالعميل. جميع التفاعلات بين المستخدم والتطبيق تتكون من التبادلات البسيطة، من طلبات الحصول على البيانات والردود على هذه الطلبات القادمة من الخادم. ما يجعل هذه التطبيقات تعمل على أكثر من منصة وإن كانت منصة ذات قدرات محدودة، مع التأكيد بأن كل العمليات المعقدة تكون في جانب الخادم.

طرق برمجة المنصات المتعددة:

يوجد عدة طرق مختلفة لبرمجة التطبيقات المتعددة المنصات، أحدها وأبسطها هي تصميم تطبيق من مجموعة مختلفة من الشفرات المصدرية، مثلاً نسخة الويندوز سوف يتم كتابتها بلغة برمجة وتحمل شفرة مصدرية خاصة بها، ونسخة ماكنتوش سوف يتم كتابتها بلغة برمجة وشفرة مصدرية مختلفة عن السابقة، وبهذه الشكل لكل المنصات الأخرى. رغم أن هذه الطريقة تعتبر مباشرة لحل المشكلة، إلا أنها تزيد من تكاليف التطوير المالية والزمنية، لأنه مع شفرة مصدرية مختلفة لكل برنامج تأتي الحاجة لعدد أكثر من المبرمجين.

أيضاً هنالك طريقة اخرى مستخدمة لتطوير هذا النوع من البرامج، تعتمد هذه الطريقة على استخدام برمجيات موجودة مسبقاً تقوم بإخفاء الاختلافات بين المنصات على التطبيق الذي يعمل عليها، ومن أهم الأمثلة على هذه البرمجيات هو آلة جافا الافتراضية ، التي تعمل كوسيط بين نظام التشغيل و البرنامج.



الشكل 2-1: يوضح فكرة لتطبيقات متعددة المنصات

التحديات التي تواجه تطوير التطبيقات متعددة المنصات:

يوجد بعض القضايا المتعلقة بالتطوير المتعدد المنصات التي ينتج عنها عدة تحديات منها:

- عملية اختبار التطبيقات المتعددة المنصات قد تعتبر معقدة، بسبب اختلافها، ينتج عن ذلك سلوك غير متوقع أو بعض المشاكل المتعلقة باستقرار عمل التطبيق. هذه المشاكل تقود المطورين إلى اللجوء إلى أسلوب التطوير الذي ينص على " اكتب مرة واحدة، يشتغل في كل مكان ".

- يكون المطورون غالباً مقيدين بالمميزات المشتركة بين كل المنصات فقط، دون تضمين خواص ومميزات أخرى، مما يحد من أداء النظام ومميزاته بسبب عدم استخدام المميزات المتطورة لكل منصة.
- المنصات المختلفة لها أعراف خاصة بواجهات استخدام مختلفة، وتطبيقات المنصات المتعددة لا تتبع هذه الأعراف في العادة.
- لغات البرمجة النصية والأجهزة الافتراضية يجب أن تتم ترجمتها إلى شفرة تنفيذية بلغة الألة (native executable code) مما يؤدي إلى إضعاف الأداء بسبب الوقت المستغرق في هذه العملية.
- البيئات التنفيذية للتطبيقات متعددة المنصات قد تعاني من ثغرات وعيوب أمنية، مما يخلق بيئة خصبة للبرمجيات الخبيثة.

خدمات الويب Web Services:

وهي طريقة معيارية لدمج التطبيقات المستندة إلى الويب باستخدام المعايير المفتوحة (XML SOAP WSDL UDDI) عبر العمود الفقري لبروتوكول الإنترنت.

XML هو تنسيق البيانات المستخدم لاحتواء البيانات وتوفير البيانات الوصفية حولها، ويستخدم SOAP لنقل البيانات، أما WSDL فيستخدم لوصف الخدمات المتاحة، ويسرد UDDI الخدمات المتاحة.

خدمة الويب هي طريقة اتصال بين جهازين إلكترونيين عبر الشبكة، وهي وظيفة برمجية يتم توفيرها على عنوان شبكة عبر الويب مع تشغيل الخدمة دائمًا كما هو الحال في مفهوم الحوسبة المساعدة.

من الناحية العملية توفر خدمة الويب عادةً واجهة ويب غرضية التوجه لمخدم قاعدة البيانات، يتم استخدامها أحياناً بواسطة مخدم ويب آخر أو بواسطة تطبيق جوال يوفر واجهة استخدام للمستثمر النهائي (end-user UI).

العديد من المؤسسات التي تقدم البيانات في صفحات HTML المنسقة، تزود أيضاً على الخادم الخاص بها تلك البيانات كملفات XML أو JSON وغالبًا يتم ذلك من خلال خدمة ويب للسماح بالمشاركة. قد يكون هناك تطبيق آخر يتم تقديمه للمستخدم النهائي وهو mashup حيث يستهلك مخدم الويب العديد من خدمات الويب على أجهزة مختلفة ويقوم بتجميع المحتوى في واجهة مستخدم واحدة. ومن أهم خدمات الويب العامة:

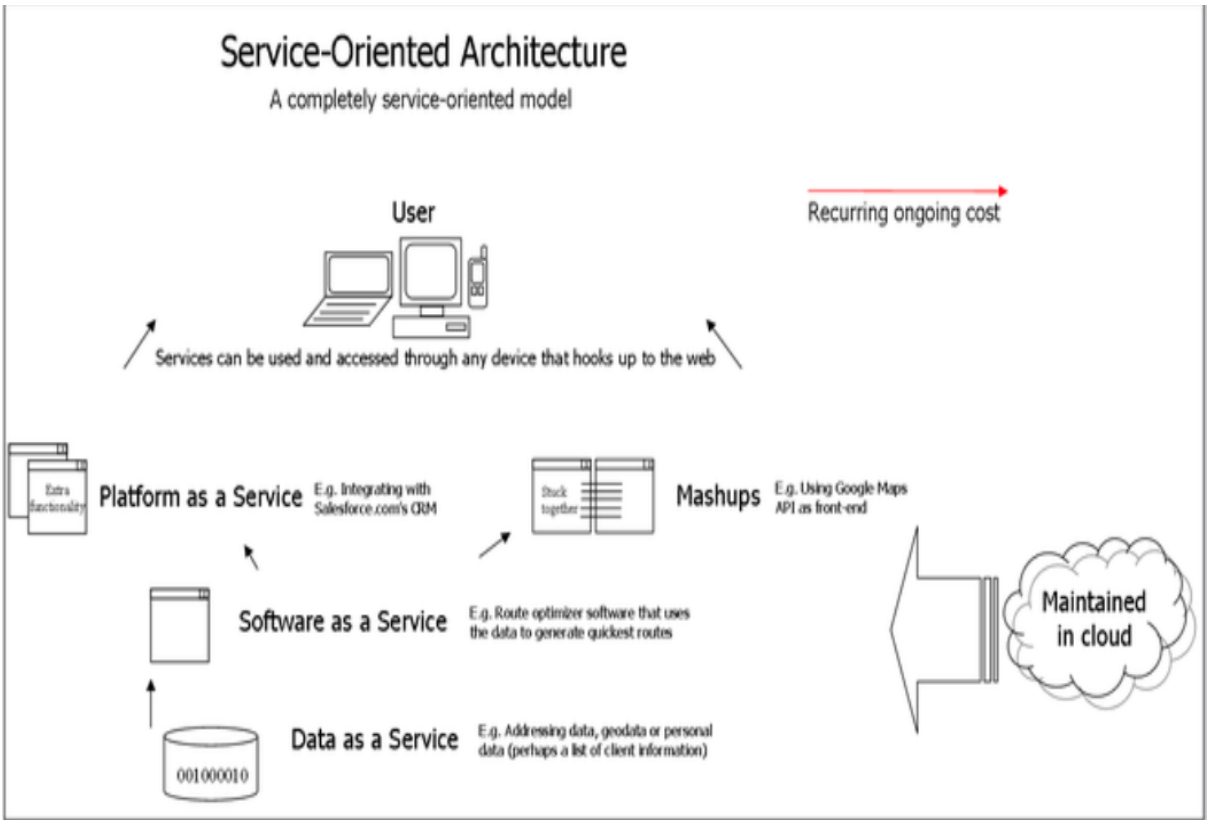
- JSON-RPC
- JSON-WSP
- Representational state transfer (REST) versus remote procedure call (RPC)
- Web Services Conversation Language (WSCL)

- Web Services Description Language (WSDL), developed by the W3C
- Web Services Flow Language (WSFL), superseded by BPEL
- Web template
- WS-MetadataExchange
- XML Interface for Network Services (XINS), provides a POX-style web service specification format

يجب تحديد قواعد الاتصال بالأنظمة المختلفة وفق الآتي:

- كيف يمكن لنظام ما أن يطلب البيانات من نظام آخر.
- ما هي المعلمات المحددة المطلوبة في طلب البيانات.
- ماذا سيكون هيكل البيانات المنتجة.
- ما هي رسائل الخطأ التي يتم عرضها عند عدم مراعاة قاعدة معينة للاتصال، وذلك لتسهيل استكشاف الأخطاء وإصلاحها.

يتم تعريف كل قواعد الاتصال هذه في ملف يسمى WSDL (لغة وصف خدمات الويب). يحدد دليل يسمى UDDI (الوصف العالمي والاكتشاف والتكامل) أي نظام برمجي يجب الاتصال به لأي نوع من البيانات. لذلك عندما يحتاج نظام برمجي واحد إلى تقرير أو بيانات معينة، فإنه ينتقل إلى UDDI ويكتشف الأنظمة الأخرى التي يمكنه الاتصال بها لتلقي تلك البيانات. بمجرد أن يكتشف نظام البرنامج الأنظمة الأخرى التي يجب عليه الاتصال بها فإنه سيتصل بعد ذلك بهذا النظام باستخدام بروتوكول خاص يسمى SOAP (بروتوكول الوصول إلى الكائنات البسيط). عندها يقوم نظام مزود الخدمة أولاً بالتحقق من صحة طلب البيانات من خلال الرجوع إلى ملف WSDL ثم معالجة الطلب وإرسال البيانات بموجب بروتوكول SOAP.



الشكل 2-2: يوضح بنية الخدمات الموجهة

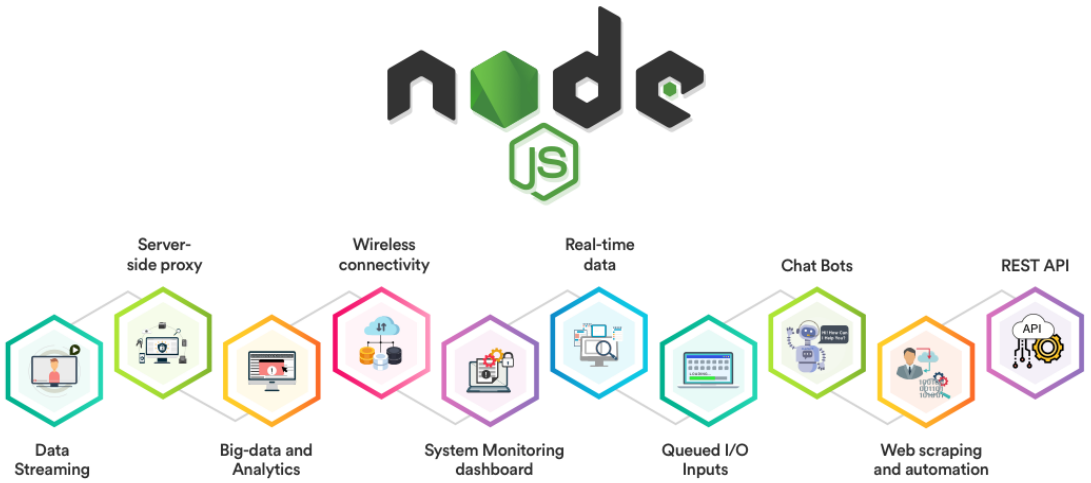
2-2 مفهوم الـ MERN Stack

تسمى أي مجموعة من التقانات المتعددة التي تستخدم لبناء تطبيق ويب ما، بالحزمة Stack ومع التطور الهائل لتطبيقات الويب متعددة المنصات ظهرت العديد من المكتبات الخاصة بالمطورين خاصةً ما يتعلق بلغة البرمجة JavaScript التي تعتبر أهم لغات البرمجة في مجال التطبيقات البرمجية الخاصة بشبكة الانترنت، التي أصبحت من أقوى وأهم اللغات البرمجية على الويب خاصةً مع ظهور NodeJS الشيء الذي مكن المطورين من البرمجة بكلا الواجهتين الأمامية والخلفية بواسطة JS.
ومصطلح MERN مأخوذ من الأحرف الأولى لكل من MongoDB و Express و React و NodeJS.

2-2-1 NodeJS

هي بيئة تشغيل خاصة بلغة البرمجة JavaScript مفتوحة المصدر ومتعددة المنصات وخلفية تعمل على محرك V8 وتنفذ كود JavaScript خارج متصفح الويب. حيث تتيح للمطورين استخدام JavaScript لكتابة أدوات سطر الأوامر والبرمجة النصية من جانب المخدم، لإنتاج محتوى صفحة ويب ديناميكي قبل إرسال الصفحة إلى متصفح الويب الخاص بالمستخدم. وبالتالي فإن Node.js يمثل

نموذج "JavaScript في كل مكان". الشيء الذي يؤدي إلى تطوير تطبيقات الويب باستخدام لغة برمجة واحدة بدلاً من لغات مختلفة للنصوص البرمجية من جانب الخادم والعميل.
تهدف اختيارات التصميم هذه إلى تحسين الإنتاجية وقابلية التوسع في تطبيقات الويب مع العديد من عمليات الإدخال/الإخراج، بالإضافة إلى تطبيقات الويب في الوقت الفعلي (مثل برامج الاتصال في الوقت الفعلي والألعاب عبر متصفح الإنترنت).



الشكل 2-3: يوضح الإمكانيات الهائلة لـ NodeJS

تم كتابة NodeJS في البداية بواسطة Ryan Dahl في عام 2009 الذي انتقد الإمكانات المحدودة لخادم الويب الأكثر شيوعًا آنذاك Apache HTTP Server خاصةً ما يتعلق بالتعامل مع الكثير من الاتصالات المتزامنة (حتى 10000 وأكثر) والطريقة الأكثر شيوعًا لإنشاء الكود (البرمجة التسلسلية) عندما يقوم الكود إما بحظر معالجة العملية بأكملها، أو في حالة الاتصالات المتزامنة فإنه يقوم باستخدام المكدسات stacks.
في يناير 2010 تم تقديم مدير الحزم لبيئة NodeJS التي تسمى npm. الشيء الذي سهل على المبرمجين نشر ومشاركة الكود المصدري للحزم وهو مصمم لتبسيط تثبيت وتحديث وإلغاء تثبيت هذه الحزم.
يسمح NodeJS بإنشاء خوادم الويب وأدوات الشبكات باستخدام لغة البرمجة JavaScript ومجموعة من الوحدات النمطية modules التي تتعامل مع العديد من الوظائف الأساسية. ويتم توفير الوحدات النمطية لنظام الملفات I/O والشبكات (DNS HTTP TCP TLS/SSL UDP) والبيانات الثنائية (المخازن المؤقتة) ووظائف التشفير وتدفق البيانات والوظائف الأساسية الأخرى وتستخدم وحدات واجهة برمجة تطبيقات مصممة لتقليل تعقيد كتابة تطبيقات الخادم.

بنية المنصة:

تعتمد NodeJS على مفهوم البرمجة المقادة بالأحداث إلى خوادم الويب أو ما يعرف بـ event-driven programing مما يتيح تطوير خوادم الويب السريعة في JavaScript. حيث يمكن للمطورين إنشاء خوادم قابلة للتطوير بدون استخدام مؤشرات الترابط، وذلك باستخدام نموذج مبسط من البرمجة التي تعتمد على الأحداث والتي تستخدم عمليات رد النداء للإشارة إلى اكتمال المهمة.

دعم الصناعة:

هناك الآلاف من المكتبات مفتوحة المصدر لـ NodeJS ومعظمها مستضاف على موقع npm وهناك العديد من المؤتمرات وأحداث المطورين التي تدعم مجتمع NodeJS بما في ذلك NodeConf وNode Interactive وNode Summit بالإضافة إلى عدد من الأحداث الإقليمية.

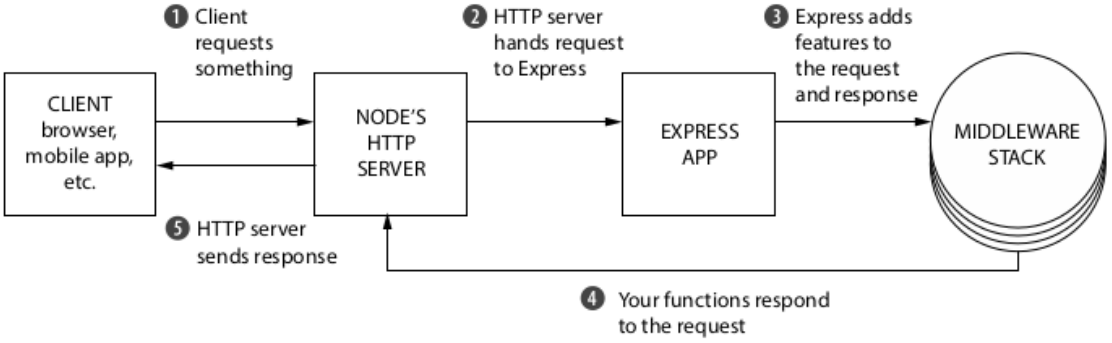
التفاصيل التقنية:

- تستخدم libuv للتعامل مع الأحداث غير المتزامنة، وهي عبارة عن طبقة تجريدية لوظائف الشبكة ونظام الملفات على كل من أنظمة Windows وPOSIX مثل Linux وmacOS وOSS على NonStop وUnix.
- تعمل على حلقة حدث ذات مؤشر ترابط واحد single-thread event loop باستخدام اتصالات الإدخال/الإخراج غير المحظورة، مما يسمح لها بدعم عشرات الآلاف من الاتصالات المتزامنة دون تكبد تكلفة تبديل سياق مؤشر الترابط.
- V8 هو محرك تنفيذ لغة البرمجة JavaScript الذي تم إنشاؤه في البداية لـ Google Chrome. ثم أصبح مفتوح المصدر في عام 2008. وقد تمت كتابته بلغة C++ ويقوم V8 بترجمة شفرة المصدر إلى لغة الآلة في وقت التشغيل. وفي عام 2016 أصبح يتضمن أيضاً مترجم بايت كود bytecode interpreter.
- npm هو مدير الحزم المثبت مسبقًا لمنصة خادم Node.js بحيث يقوم بتثبيت برامج NodeJS من سجل npm وتنظيم تثبيت وإدارة برامج الطرف الثالث أيضاً. يمكن أن تتراوح الحزم في سجل npm من مكتبات مساعدة بسيطة مثل Lodash إلى مقدمي المهام مثل Grunt.
- يمكن دمج NodeJS مع المتصفح وقاعدة بيانات تدعم بيانات JSON.
- تسمح بإعادة استخدام نفس النموذج وواجهة الخدمة، بين العميل والمخدم.
- تستخدم NodeJS حلقة الحدث event-driven لتحقيق قابلية التوسع، بدلاً من سلاسل العمليات.
- تدعم WebAssembly.
- توفر طريقة لإنشاء الإضافات addons عبر واجهة برمجة تطبيقات تستند إلى C تسمى N-API والتي يمكن استخدامها لإنتاج وحدات عقدة قابلة للتحميل (قابلة للاستيراد) من

التعليمات البرمجية المصدرية المكتوبة بلغة C++ ويمكن تحميل الوحدات مباشرةً في الذاكرة وتنفيذها من داخل بيئة JS كوحدات CommonJS البسيطة.

Express 2-2-2

هو إطار عمل ويب سريع وحازم وأساسي ومتوسط لـ NodeJS حيث يمكننا أن نفترض التعبير السريع كطبقة مبنية في الجزء العلوي من NodeJS تساعد في إدارة المخدم والمسارات. وهو يوفر مجموعة قوية من الميزات لتطوير تطبيقات الويب والجوال.



الشكل 2-4: يوضح آلية استخدام Express

يمكن تلخيص بعض الميزات الأساسية لـ Express framework من خلال الآتي:

- يمكن استخدامه لتصميم تطبيقات ويب أحادية الصفحة، ومتعددة الصفحات، ومختلطة.
- يسمح بإعداد البرامج الوسيطة للرد على طلبات HTTP.
- يحدد جدول التوجيه الذي يتم استخدامه لأداء إجراءات مختلفة بناءً على طريقة HTTP وعنوان URL.
- يسمح بعرض صفحات HTML ديناميكيًا استنادًا إلى تمرير الوسائط إلى القوالب.
- فائق السرعة في عمليات I/O.
- غير متزامن ووحيد الترابط Async & Single Threaded.
- واجهة برمجة التطبيقات القوية تجعل عملية التوجيه أمراً سهلاً.

يعتمد Express بشكل كبير على البرمجيات الوسيطة Middlewares وهي عبارة عن وظائف تصل إلى كائن الطلب والاستجابة (req, res) وتستدعى خلال دورة حياة الطلب والاستجابة. حيث يمكن أن تؤدي وظيفة البرامج الوسيطة المهام التالية:

- يمكنها تنفيذ أي كود برمجي.
- يمكنها إجراء تغييرات على الكائن (req, res).
- يمكنها إنهاء دورة الطلب والاستجابة (req, res).
- يمكنها استدعاء تابع البرمجيات الوسيطة التالية في المكدس.

React 2-2-3

(المعروفة أيضًا باسم React.js أو ReactJS) هي مكتبة جافا سكريبت مفتوحة المصدر أو واجهة أمامية لبناء واجهات المستخدم أو مكونات واجهة المستخدم. تم إنشاؤها من قبل شركة Facebook تستخدم هذه المنصة كأساس في تطوير التطبيقات أحادية الصفحة SPA أو تطبيقات الهاتف المحمول. إن React معنية فقط بإدارة الحالة وتقديم هذه الحالة إلى DOM لذلك يتطلب إنشاء تطبيقاتها عادةً إلى استخدام مكتبات إضافية للتوجيه، بالإضافة إلى وظائف معينة من جانب العميل.

<u>الميزات البارزة:</u>

- يتكون كود React من كيانات تسمى المكونات Components يمكن تقديم هذه المكونات لعنصر معين في DOM باستخدام مكتبة React DOM. عند عرض أحد المكونات، يمكن للمرء أن يمرر قيمًا تُعرف باسم props.
- الطريقتان الأساسيتان للتصريح عن المكونات في React هما عبر المكونات الوظيفية functional components والمكونات المبنية على الصفوف class-based components.
- يتم التصريح عن المكونات الوظيفية بواسطة تابع يقوم بعد ذلك بإرجاع عنصر من نوع JSX.

```
const Greeting = (props) => <div>Hello, {props.name}</div>
```

- يتم الإعلان عن المكونات المبنية على الصفوف باستخدام ES6.

```
class ParentComponent extends React.Component {
  state = { color: 'green' };
  render() {
    return (
      <ChildComponent color={this.state.color} />
    );
  }
}
```

- ميزة أخرى ملحوظة هي استخدام نموذج كائن المستند الافتراضي أو Virtual DOM حيث تقوم React بإنشاء ذاكرة تخزين مؤقت لهيكل البيانات في الذاكرة وتحسب الاختلافات الناتجة، ثم تقوم بتحديث DOM المعروض في المتصفح بكفاءة، يسمح هذا للمبرمج بكتابة التعليمات البرمجية كما لو تم عرض الصفحة بأكملها عند كل تغيير، بينما تعرض مكتبات React المكونات الفرعية التي تتغير بالفعل فقط. يوفر هذا العرض الانتقائي تعزيزًا كبيرًا

للأداء، ويوفر الجهد المبذول لإعادة حساب نمط CSS وتخطيط الصفحة وعرض الصفحة بأكملها.

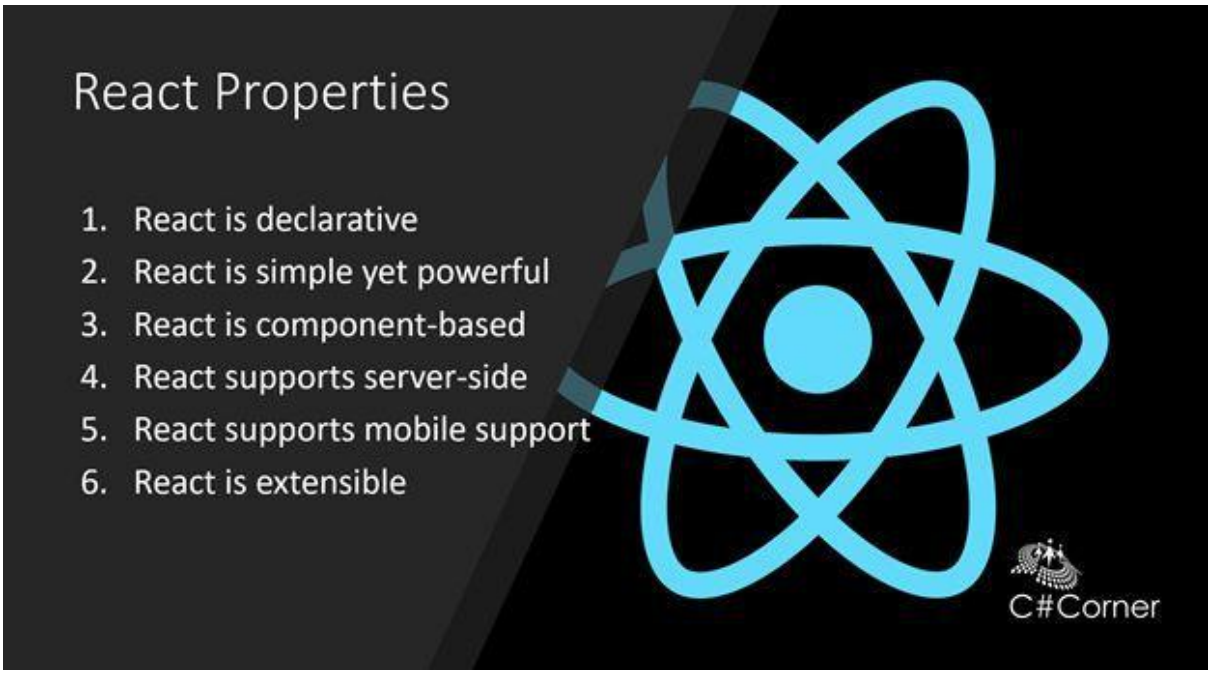
- تستخدم طرق دورة الحياة الخاصة بالمكونات components شكلاً من أشكال الربط الذي يسمح بتنفيذ التعليمات البرمجية في نقاط محددة أثناء عمر المكون ضمن زمن تشغيل الكود.
- من أهم الأمور التي تتفوق فيها منصة React هي استخدام ما يعرف بـ JSX وهي اختصار لـ JavaScript XML الشيء الذي يسمح للمطور دمج لغة البرمجة هذه مع التعابير الخاصة بـ HTML وبشكل مباشر وسلس.

```
class App extends React.Component {
 render() {
  return (
    <div>
      <p>Header</p>
      <p>Content</p>
      <p>Footer</p>
    </div>
  );
 }
}
```

- يجب تغليف العناصر المتعددة على نفس المستوى في عنصر React واحد مثل عنصر الموضح أعلاه، أو جزء محدد بواسطة <Fragment> أو في شكله المختصر <> أو إعادته كمصفوفة.
- يوفر JSX مجموعة من سمات العناصر Attributes المصممة لإسقاط العناصر التي توفرها HTML ويمكن أيضًا تمرير السمات المخصصة إلى المكون، حيث يتم استلام جميع السمات بواسطة المكون كـ props.
- تنطبق البنية الأساسية لـ React على ما هو أبعد من عرض HTML في المتصفح، على سبيل المثال يحتوي Facebook على مخططات ديناميكية تُعرض لعلامات <canvas> ويستخدم Netflix وPayPal التحميل العام لعرض HTML متطابق على كل من المخدم والعميل.

React Native

- هو إطار عمل خاص بتطبيقات الهاتف الجوال مفتوح المصدر تم إنشاؤه بواسطة شركة Facebook يتم استخدامه لتطوير تطبيقات (Android, Android-TV, iOS, macOS, tvOS, Windows, UWP) من خلال تمكين المطورين من استخدام منصة React جنبًا إلى جنب مع إمكانات النظام الأساسي الأصلي.
- تتطابق مبادئ عملها تقريبًا مع React باستثناء أنها لا تتعامل مع Virtual DOM لكن يتم التعامل مع DOM من خلال عملية خلفية (التي تفسر JavaScript المكتوبة من قبل المطورين) مباشرةً على الجهاز النهائي، ويتم الاتصال مع النظام الأساسي الأصلي عن طريق البيانات المتسلسلة عبر جسر غير متزامن ومجمع.
- تقوم مكونات React بتغليف الكود الأصلي الحالي والتفاعل مع واجهات برمجة التطبيقات الأصلية عبر نموذج واجهة المستخدم التعريفي لـ React وجافا سكريبت. ويتيح ذلك تطوير التطبيقات الأصلية لفرق جديدة كاملة من المطورين، ويمكن أن يتيح للفرق المحلية الحالية العمل بشكل أسرع.
- React Native لا تستخدم HTML أو CSS وبدلاً من ذلك يتم استخدام الرسائل من سلسلة جافا سكريبت لمعالجة طرق العرض الأصلية. حيث تتيح React Native أيضًا للمطورين كتابة التعليمات البرمجية الأصلية بلغات مثل Java أو Kotlin لنظام Android و Objective-C أو Swift لنظام iOS مما يجعل ذلك أكثر مرونة.



الشكل 2-5: يوضح المميزات الرئيسية لـ ReactJS

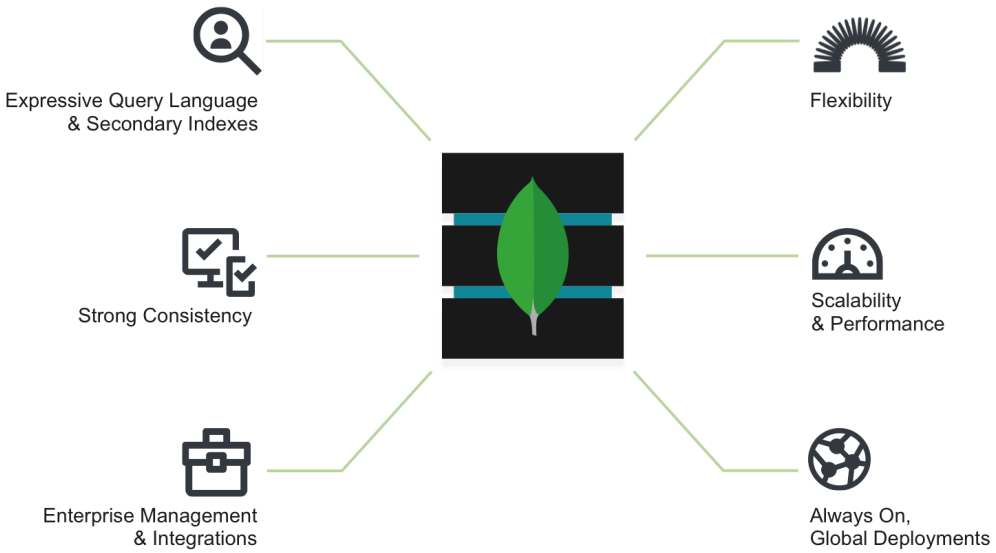
MongoDB 2-2-4

عبارة عن برنامج قاعدة بيانات موجه للمستندات متعدد المنصات متاح المصدر. يستخدم مستندات تشبه JSON مع مخططات اختيارية schemas.

الخصائص الرئيسية:

- تدعم عمليات البحث بشكل فعال جداً، ويمكن أن ترجع الاستعلامات حقولًا محددة من المستندات وتتضمن أيضًا وظائف JavaScript معرفة من قبل المستخدم، ويمكن تكوين الاستعلامات لإرجاع عينة عشوائية من النتائج ذات الحجم المحدد.
- يمكن فهرسة الحقول الموجودة في مستند MongoDB بالمؤشرات الأولية والثانوية.
- توفر MongoDB مجموعات من النسخ المتماثلة، تتكون مجموعة النسخ المتماثلة من نسختين أو أكثر من البيانات، وقد يعمل عضو مجموعة النسخ المتماثلة في دور النسخة المتماثلة الأساسية أو الثانوية في أي وقت. عند فشل نسخة متماثلة أولية تقوم مجموعة النسخ المتماثلة تلقائيًا بإجراء عملية اختيار لتحديد أي نسخة ثانوية يجب أن تصبح أساسية. يمكن للنسخ الثانوية أن تخدم عمليات القراءة اختياريًا، لكن هذه البيانات تكون متسقة في النهاية بشكل افتراضي فقط.
- يمكن تشغيل MongoDB على خوادم متعددة، وموازنة التحميل أو تكرار البيانات للحفاظ على النظام وتشغيله في حالة فشل الأجهزة.
- يمكن استخدام MongoDB كنظام ملفات يسمى GridFS مع موازنة التحميل ومميزات النسخ المتماثل للبيانات عبر أجهزة متعددة لتخزين الملفات.
- توفر ثلاث طرق لأداء التجميع وهي خط أنابيب التجميع aggregation pipeline ووظيفة التقليل map-reduce function وطرق التجميع أحادية الغرض single-purpose aggregation.
- يمكن استخدام JavaScript في الاستعلامات ووظائف التجميع (مثل MapReduce) وإرسالها مباشرة إلى قاعدة البيانات ليتم تنفيذها.
- تدعم المجموعات ذات الحجم الثابت، والتي تسمى المجموعات ذات الحد الأقصى. حيث يحافظ هذا النوع من المجموعات على ترتيب الإدراج، وبمجرد الوصول إلى الحجم المحدد يتصرف مثل قائمة انتظار دائرية.
- لها عدة اصدارات Community Server وEnterprise Server وMongoDB Atlas.
- لديها محركات رسمية للغات البرمجة الرئيسية وبيئات التطوير. ويوجد أيضًا عدد كبير من برامج التشغيل غير الرسمية للغات وأطر البرمجة الأخرى.
- مع خاصية Stitch يمكن الوصول إلى MongoDB والخدمات الأخرى بدون الحاجة لمخدم serverless بالإضافة إلى توفر مكتبات خاصة بالعملاء لكل من JS وAndroid وiOS.

- تم تقديم MongoDB Compass باعتباره واجهة المستخدم الرسومية الأصلية، وهناك منتجات ومشاريع تابعة لجهات مستقلة توفر واجهات مستخدم للإدارة وعرض البيانات.
- اعتباراً من إصدار MongoDB 2.6 فصاعدًا ترتبط الثنائيات من حزم MongoDB RPM وDEB الرسمية بالمضيف المحلي localhost افتراضيًا. وتم توسيع هذا السلوك الافتراضي ليشمل جميع حزم MongoDB عبر جميع الأنظمة الأساسية. نتيجة لذلك سيتم رفض كافة الاتصالات الشبكية بقاعدة البيانات ما لم يتم تكوينها بشكل صريح من قبل المسؤول system administrator.



الشكل 2-6: يوضح المميزات الرئيسية لـ MongoDB

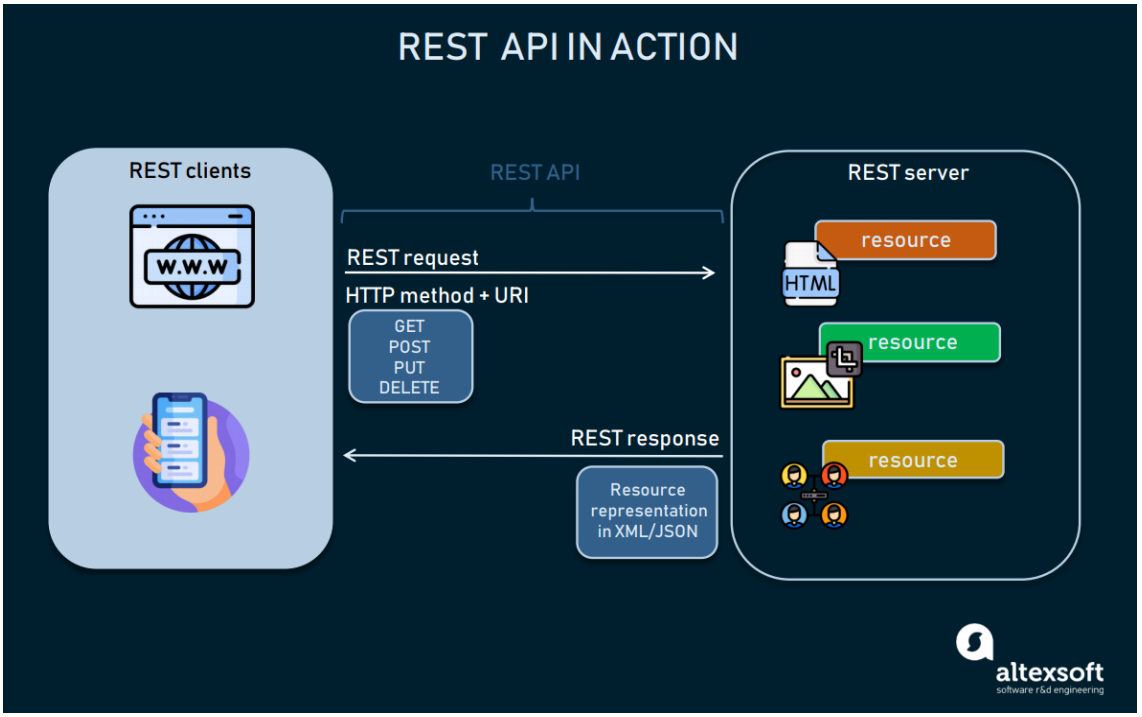
2-3 REST API إيجابيات وسلبيات

هو نمط معماري برمجي يستخدم مجموعة فرعية من HTTP يتم استخدامه بشكل شائع لإنشاء تطبيقات تفاعلية تستخدم خدمات الويب. تسمى خدمة الويب التي تتبع هذه الإرشادات بـ RESTful يتيح هذا الأسلوب إمكانية التشغيل البيني بين أنظمة الكمبيوتر على الإنترنت التي توفر هذه الخدمات.

يجب أن توفر خدمة الويب موارد الويب الخاصة بها في تمثيل نصي، وتسمح بقراءتها وتعديلها باستخدام بروتوكول عديم الحالة stateless ومجموعة محددة مسبقًا من العمليات.

يتيح هذا الأسلوب إمكانية التشغيل البيني بين أنظمة الكمبيوتر على الإنترنت التي توفر هذه الخدمات. ويعتبر بديل عن SOAP كطريقة للوصول إلى خدمة ويب. في خدمة الويب من نوع RESTful تؤدي الطلبات التي يتم إجراؤها إلى URI للمورد، إلى استجابة بحمولة منسقة بتنسيق HTML أو XML أو JSON أو أي تنسيق آخر. البروتوكول الأكثر شيوعًا لهذه الطلبات والاستجابات هو HTTP الذي يوفر الطرق الأكثر شيوعاً للاستخدام مثل GET وPOST وPUT وDELETE.

الهدف من REST هو زيادة الأداء وقابلية التوسع والبساطة وقابلية التعديل والرؤية وإمكانية النقل والموثوقية. يتم تحقيق ذلك من خلال اتباع المبادئ الأساسية فيه مثل بنية المخدم/العميل، وانعدام الحالة، وإمكانية التخزين المؤقت، واستخدام نظام متعدد الطبقات، ودعم التعليمات البرمجية عند الطلب، واستخدام واجهة موحدة. ويجب اتباع هذه المبادئ حتى يتم تصنيف النظام على أنه REST.



الشكل 2-7: يوضح آلية عمل REST API

الخصائص المعمارية:

تؤثر قيود النمط المعماري REST على الخصائص المعمارية التالية:

- الأداء في تفاعلات المكونات، والتي يمكن أن تكون العامل المهيمن في الأداء الذي يدركه المستخدم وكفاءة الشبكة.
- قابلية التوسع التي تسمح بدعم أعداد كبيرة من المكونات والتفاعل فيما بينها.
- بساطة الواجهة الموحدة.
- قابلية تعديل المكونات لتلبية الاحتياجات المتغيرة (حتى أثناء تشغيل التطبيق).
- وضوح الاتصال بين المكونات من قبل وكلاء الخدمة.
- إمكانية نقل المكونات عن طريق نقل كود البرنامج مع البيانات.
- الموثوقية في مقاومة الفشل على مستوى النظام في حالة وجود أعطال داخل المكونات أو الموصلات أو البيانات.

القيود المعمارية:

توجد ستة قيود تحدد نظام RESTful وهذه القيود تحدد الطرق التي من خلالها يمكن للمخدم أن يستجيب ويعالج الطلبات الواردة من العملاء، ومن خلال التشغيل ضمن هذه القيود يكتسب النظام خصائص غير وظيفية مرغوبة، مثل الأداء وقابلية التوسع والبساطة وقابلية التعديل وقابلية النقل والموثوقية.

قيود REST الرسمية هي كما يلي:

- بنية المخدم/العميل Client/Server.
- انعدام الحالة Statelessness.
- القابلية للتخزين المؤقت Cacheability.
- نظام الطبقات Layered system.
- الكود عند الطلب (اختياري) Code on demand.
- الواجهة الموحدة Uniform interface.

تطبيق REST على خدمات الويب:

تسمى واجهات برمجة تطبيقات خدمة الويب التي تلتزم بقيود REST المعمارية بـ RESTful APIs التي تتميز بالجوانب التالية:

- معرّف URI أساسي كمثال http://api.example.com.
- طرق HTTP القياسية (GET وPOST وPUT وDELETE).
- نوع وسائط يحدد عناصر بيانات انتقال الحالة.

على عكس خدمات الويب المستندة إلى SOAP لا يوجد معيار قياسي لواجهات برمجة تطبيقات الويب RESTful. هذا لأن REST هو أسلوب معماري بينما SOAP هو بروتوكول. كما أن REST ليس معيارًا في حد ذاته ولكن تطبيقات RESTful تستخدم معايير مثل HTTP وURI وJSON

وXML. يصف العديد من المطورين واجهات برمجة التطبيقات الخاصة بهم بأنها RESTful على الرغم من أن واجهات برمجة التطبيقات هذه لا تفي بجميع القيود المعمارية الموضحة أعلاه (خاصة قيود الواجهة الموحدة).

مزايا REST API:

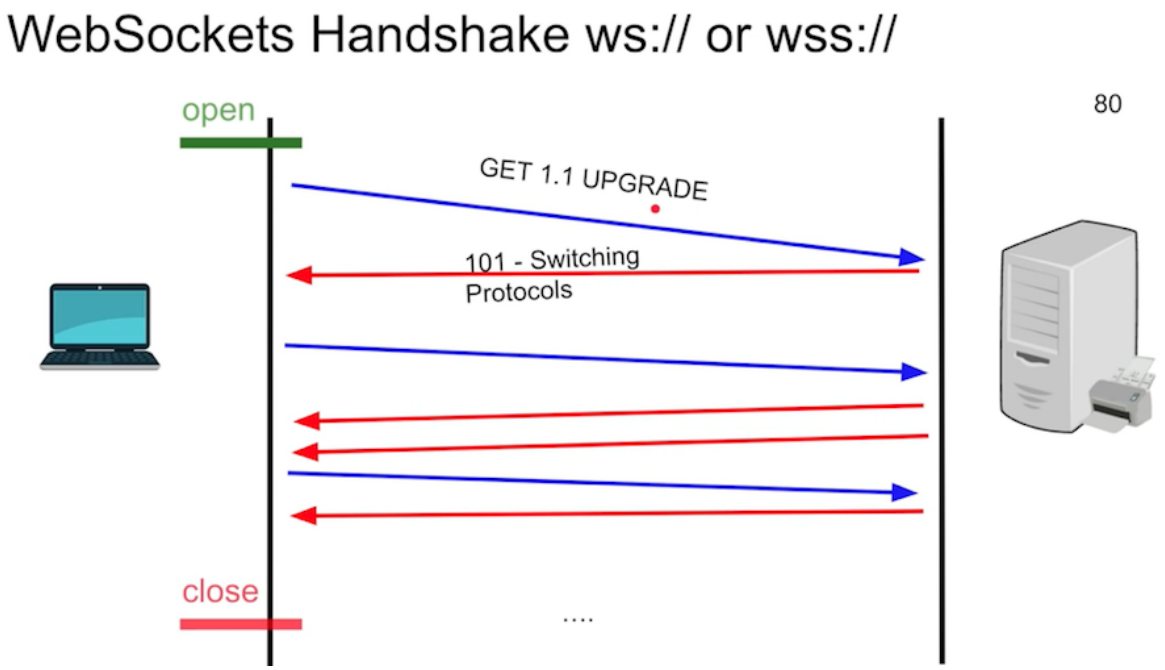
- واجهة برمجة تطبيقات REST سهلة الفهم والتعلم نظرًا لبساطتها.
- القدرة على تنظيم التطبيقات المعقدة وتسهيل استخدام الموارد.
- يمكن إدارة الحمل العالي بمساعدة من خادم وكيل HTTP وذاكرة التخزين المؤقت.
- سهلة الفحص والاستكشاف.
- إنه يجعل من السهل على العملاء الجدد العمل على تطبيقات أخرى، سواء كانت مصممة خصيصًا لغرض أم لا.
- استخدام طرق HTTP القياسية لاسترداد البيانات والطلبات.
- تعتمد REST API على الرموز، ويمكن استخدامها لمزامنة البيانات مع موقع ويب دون أي تعقيدات.
- يمكن للمستخدمين الاستفادة من الوصول إلى نفس الكائنات القياسية ونموذج البيانات عند مقارنتها بخدمات الويب المستندة إلى SOAP.
- لديها المرونة الكبيرة عن طريق تسلسل البيانات بتنسيق XML أو JSON.
- تسمح بالحماية المستندة إلى المعايير باستخدام بروتوكولات OAuth للتحقق من طلبات REST.

عيوب REST API:

- الافتقار إلى الحالة state، حيث تتطلب معظم تطبيقات الويب آليات ذات حالة. لنفترض أنك اشتريت موقع ويب يمكن الزبون من استخدام عربة التسوق الإلكترونية. يلزم معرفة عدد العناصر الموجودة في هذه العربة قبل إجراء الشراء الفعلي. حينها يقع عبء الحفاظ على حالة البيانات على عاتق العميل نفسه، مما يجعل تطبيق العميل ثقيلًا ويصعب صيانته.
- الأمان والحماية في REST أقل منها في SOAP على سبيل المثال، لذلك نجد بأن استخدام REST جيد وملائم على التطبيقات العامة، لكن ذلك غير ملائم لتمرير البيانات ذات درجة الوثوقية العالية بين المخدم والعميل.

WebSocket 2-4

هو بروتوكول اتصالات كمبيوتر يوفر قنوات اتصال ثنائية الاتجاه عبر اتصال TCP واحد، تم جعل هذا البروتوكول معيارياً من قبل IETF في عام 2011 وتم توحيد WebSocket API بواسطة W3C. وهو يختلف عن HTTP ويقع كلا البروتوكولين في الطبقة 7 في نموذج OSI ويعتمدان على TCP في الطبقة 4. وعلى الرغم من اختلاف البروتوكولين لكن تم تصميم WebSocket كي يدعم بروتوكول HTTP عبر المنافذ (80-443) ولتحقيق التوافق بين البروتوكولين أثناء الاتصال بين الأجهزة، يتم استخدام مصطلح المصافحة handshake عند بدء عملية التخاطب وذلك من أجل التحول من بروتوكول HTTP إلى بروتوكول WebSocket.



الشكل 2-8: يوضح عملية المصافحة في WebSocket

يتيح بروتوكول WebSocket التفاعل بين مستعرض الويب ومخدم الويب مع حمل أقل مما هو موجود لدى HTTP الوحيد الاتجاه، مما يسهل نقل البيانات في الوقت الفعلي من الخادم وإليه.
وبهذه الآلية أصبح ممكنًا للمخدم إرسال المحتوى إلى العميل دون أن يرسل العميل الطلب أولاً، والسماح بتمرير الرسائل ذهابًا وإيابًا مع إبقاء الاتصال مفتوحًا. يتم تنفيذ إصدار آمن من هذا البروتوكول في معظم المتصفحات المشهورة عالمياً.

لتأسيس اتصال WebSocket يرسل العميل طلب مصافحة إلى المخدم، ثم يقوم المخدم بإرجاع استجابة مصافحة إلى العميل، كما هو موضح في المثال أدناه:

طلب العميل

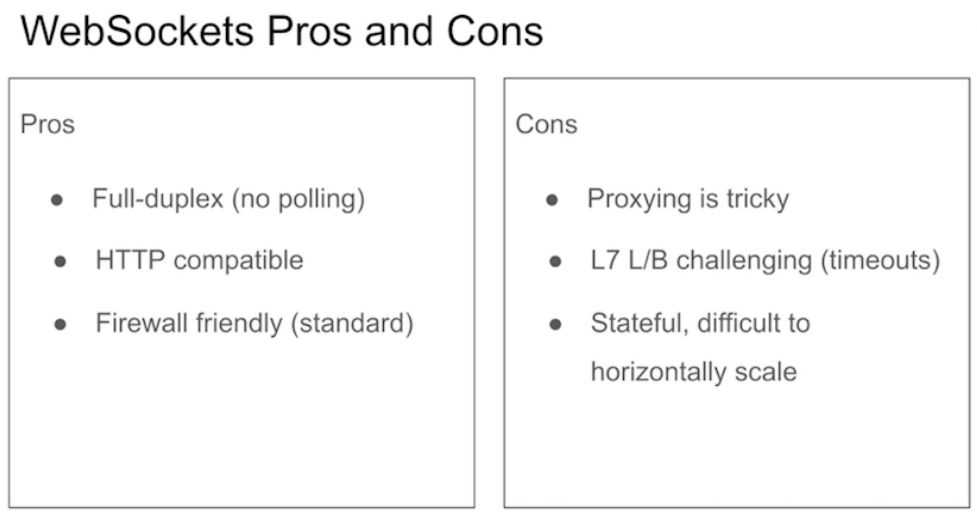
```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

استجابة المخدم

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

الاعتبارات الأمنية:

على عكس طلبات HTTP العادية، لا يتم تقييد طلبات WebSocket بواسطة سياسة نفس الأصل same-origin policy لذلك يجب أن تتحقق المخدمات من صحة مصدر عنوان "Origin" مقابل الأصول المتوقعة أثناء إنشاء الاتصال، وذلك لتجنب هجمات اختراق WebSocket عبر الموقع (على غرار تزوير طلب Cross-site) والذي قد يكون ممكنًا عند مصادقة الاتصال باستخدام ملفات تعريف الارتباط أو مصادقة HTTP. ومن الأفضل استخدام الرموز المميزة أو آليات الحماية المماثلة لمصادقة اتصال WebSocket عند نقل بيانات حساسة.



الشكل 2-9: يوضح إيجابيات وسيئات WebSocket

NGINX 2-5

هو عبارة عن مخدم ويب مفتوح المصدر، خفيف وذو أداء عالي جداً، يمكن أن يستخدم من أجل التعامل مع الملفات وتبادلها.

الشكل 2-10: يوضح ماهية NGINX

يمكن تحديد أهم مميزات هذا المخدم من خلال النقاط التالية:

- هو مخدم ويب مفتوح المصدر، سريع وخفيف الحمل وعالي الأداء، يمكن استخدامه لخدمة الملفات الثابتة.
- يعتبر الأشهر خلف كل من Apache وMicrosoft IIS.
- يعمل على تحسين تسليم المحتوى والتطبيقات، وتحسين الأمان، وتسهيل قابلية التوسع والتوافر لمواقع الويب الأكثر ازدحامًا على الإنترنت.
- بنسخته الأولى عمل مخدم NGINX على تقديم خدمة الويب من خلال بروتوكول HTTP لكن حالياً يستخدم كذلك كمخدم وكيل عكسي لكل من البروتوكولات (HTTP, HTTPS,

(SMTP, IMAP, POP3) إضافةً لاستخدامه كموازن حمل Load Balancing و Cache أيضاً.

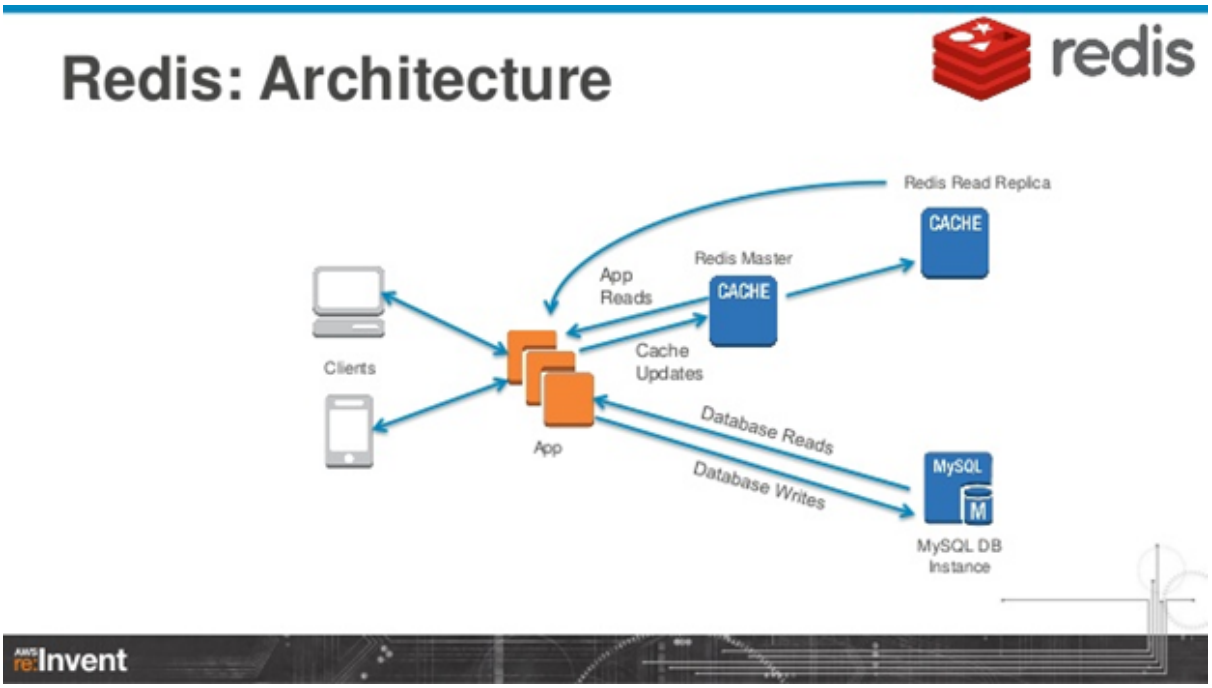
- يمكننا القول إن NGINX هو مجرد نوع من البرامج المستخدمة في خوادم الويب لخدمة الطلبات المتزامنة.
- يقدم nginx بنية قائمة على الأحداث وغير متزامنة، حيث تجعل منه هذه الميزة أكثر الخوادم موثوقية من حيث قابلية التوسع والسرعة.
- من أهم الشركات التي تستخدم هذه التقنية IBM, Google, Microsoft, VMware, LinkedIn, facebook.

الشكل 2-11: يوضح حالات استخدام NGINX

Redis 2-6

- وهي عبارة عن مخزن المعطيات في الذاكرة مفتوح المصدر، والتي يمكن استخدامها كقاعدة بيانات أو كمخزن مؤقت Cache أو كوسيط لإرسال الرسائل.
- تعتمد Redis على بنية الزوج key/value في عملية التخزين.
- تعتبر قاعدة معطيات من طراز NoSQL

- تدعم بنية المعطيات المتعددة.
- تم بناؤها باستخدام النسخ المتماثلة.
- تدعم مختلف أنواع المعطيات (Strings, Lists, Sets, Sorted Sets, Hashes) (Bitmaps, Hyperlogs, Geospatial indexes

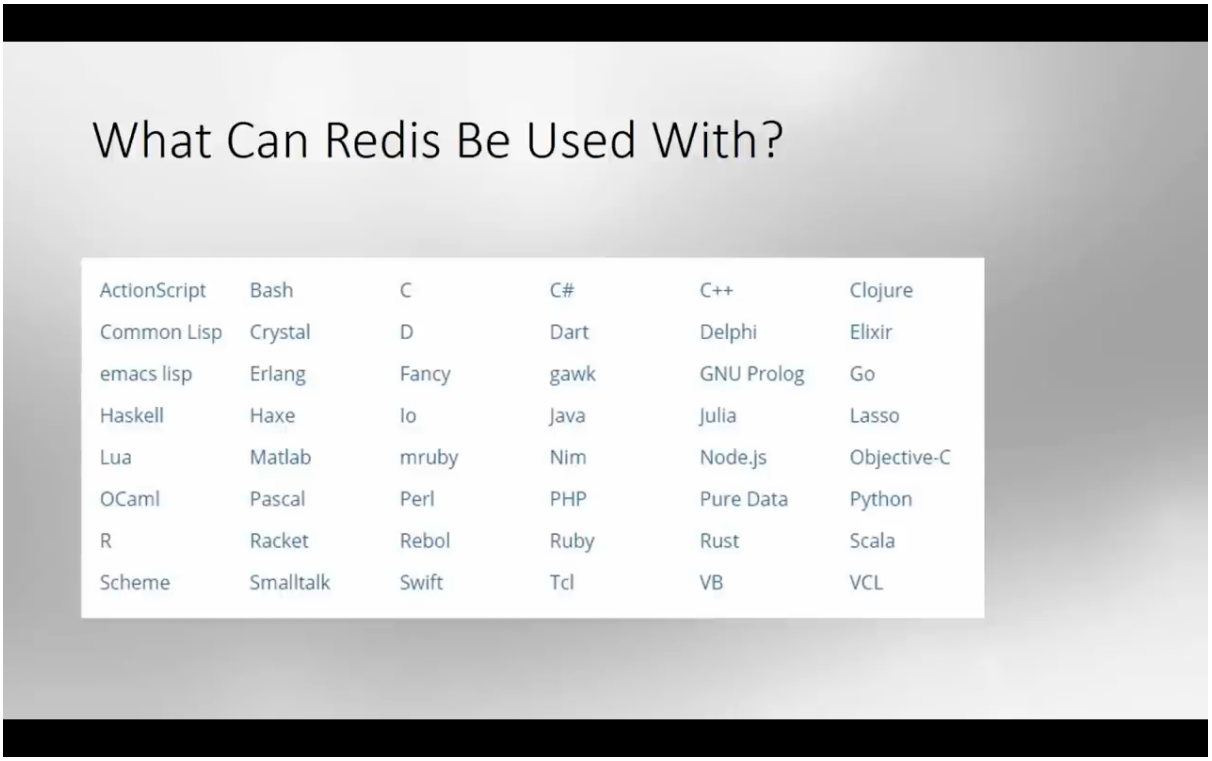


الشكل 2-12: يوضح بنية Redis

مزايا Redis:

- مرنة للغاية
- ليس لها مخطط ثابت Schema وليس لها أسماء للأعمدة Column Names
- سريعة جداً في تنفيذ العمليات (حوالي 110,000 عملية SET في الثانية، و حوالي 81,000 عملية GET في الثانية)
- غنية جداً من ناحية دعمها لأنماط المعطيات
- تدعم التخزين المؤقت Caching والتخزين الدائم Disk Persistence
- مصممة للوصول من قبل الشخص أو العميل الموثوق به Trusted Client وهي لا تسمح بالوصول الخارجي المباشر External Access / Internet Exposure
- يمكن إعداد عملية المصادقة البسيطة
- يمكن جعلها مقيدة لواجهات معينة
- لا تدعم تشفير المعطيات

وأخيراً يبين الجدول التالي أهم حالات الاستخدام الخاصة بـ Redis:



الشكل 2-13: يبين حالات استخدام Redis

JWT 2-7

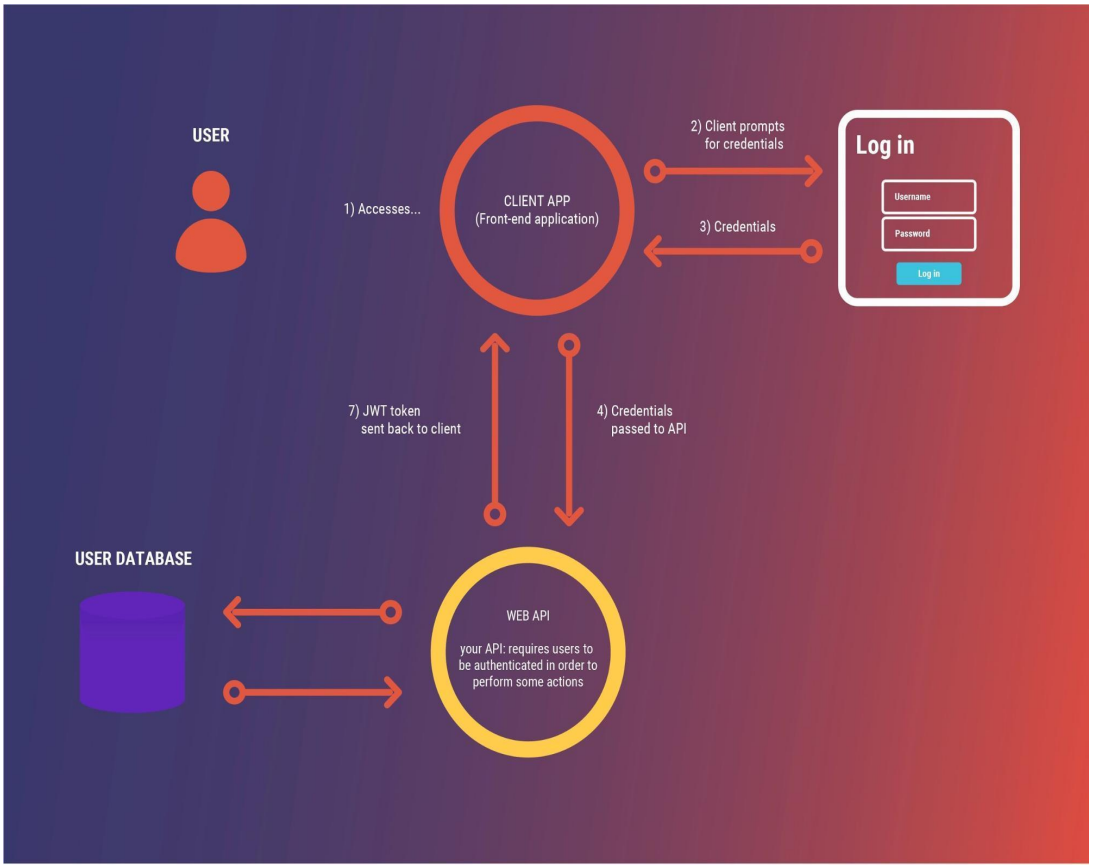
JSON Web Token هي طريقة قياسية مفتوحة آمنة، والتي تساعد في نقل جميع المعلومات بين طرفين معنيين بشكل آمن. يمكن توقيع JWT بمساعدة أي مفتاح سري بخوارزمية مناسبة. عندما تتبادل الأنظمة البيانات السرية، يتم استخدام التطبيق الآمن JSON مما يساعد في تحديد هوية المستخدم دون أي بيانات اعتماد خاصة، حيث تعد JWT حاليًا أحدث التقنيات التي تستخدمها خدمات تطوير التطبيقات، مما يساعد في تأمين واجهات برمجة التطبيقات.

فوائد تأمين الحماية في API:

تعمل رموز الويب JSON على إنشاء تطبيق يقوم بترميز البيانات السرية لتوفير الأمان. الشيء الذي يضمن الأمور التالية:

- إنشاء خوادم تفويض مخصصة.
- يتم إنشاء النطاقات والمطالبات المخصصة.
- يمكننا الالتزام بالامتثال Compliance.
- لا يتم تمرير أوراق الاعتماد الشخصية Credentials أو تبادلها.
- يمكننا إدارة الوصول إلى الـ API مع القواعد والامتثال المناسب.

- يتم إجراء التغييرات القياسية والتحديثات التلقائية على واجهة برمجة التطبيقات الخاصة بنا باستخدام منصة JWT API.
- تمت مصادقة جميع المستخدمين في التطبيق وهوياتهم صحيحة.
- ستكون الرموز المميزة المسجلة بواسطة JWT على الويب رمزًا فريدًا.
- عندما يتم تأمين التطبيق بمساعدة رمز فريد، يتم تقييد الوصول على أي شخص يثبت أنه ضار للتطبيق.



الشكل 2-14: يبين حالة استخدام JWT

الطرق التي تساعد فيها JWT على تأمين الحماية للواجهة البرمجية API:

- توفر آلية تشارك المعلومات المؤمنة عبر مجالات أمان مختلفة في تطبيقات الوقت الفعلي.
- تقوي الاتصال والعلاقة بين الطرفين اللذين يتبادلان البيانات من خلال واجهة برمجة التطبيقات.
- تؤكد الهوية المرتبطة بالثقة بين الطرفين المتصلين.
- تساعد على إنشاء واستخدام الرموز المميزة.

آلية تطبيق JWT:

- يقوم مستخدم التطبيق أولاً بتسجيل الدخول. نحتاج إلى بدء استخدام التطبيق بإدخال بيانات اعتماد تسجيل الدخول الخاصة بنا.
- بمجرد التحقق من بيانات تسجيل الدخول الخاصة بمستخدم التطبيق، تقوم API الخاصة بالتطبيق بإنشاء رمز JWT ثم تسجيل الدخول باستخدام المفتاح السري لواجهة برمجة التطبيقات، حيث يعد تطبيق JWT للتطبيقات أمرًا إلزاميًا، لأنه يوفر اتصالاً آمنًا وتبادل آمن للبيانات.
- ستقوم واجهة برمجة التطبيقات بعد ذلك بإعادة الرمز المميز إلى متصفح المستثمر.
- بعد أن يتلقى تطبيق العميل رمز JWT يتحقق من صحته، ثم يستخدمه لاحقًا في كل مرة دون أن يرسل المستخدمون بيانات اعتمادهم الشخصية مرة أخرى.
- تتكون بنية JWT من ثلاثة أجزاء رئيسية يفصل بينها نقطة، كما يلي:

[Header].[Payload].[Signature]

2-8 المخاطر الأمنية لتطبيقات الويب

تنتشر مخاطر أمان التطبيقات ويمكن أن تشكل تهديداً مباشراً لتوافر الأعمال. على الرغم من أنه ليس مطلباً أمنياً مستقلاً، إلا أن مخاطره المتزايدة للتسبب في هجمات رفض الخدمة تجعله أمراً بالغ الأهمية.
يقوم العالم باستخدام التطبيقات والبرامج المستندة إلى الويب، ونظراً لانتشارها الضخم ضمن شبكة الإنترنت، أصبحت الثغرات هي ناقل الهجوم الجديد. وقد ينتج عن الهجوم على أحد تطبيقات الويب معلومات لا ينبغي أن تكون متاحة، أو التجسس على المتصفح، أو سرقة الهوية، أو سرقة الخدمة أو المحتوى، أو إتلاف صورة الشركة أو التطبيق نفسه، وكذلك رفض الخدمة المخيف. وتعد تطبيقات الويب عرضة للخطر نظراً لطبيعة HTTP ذاتها لأنها نص واضح، حيث يجد المهاجمون أنه من السهل جداً تعديل المحددات وتنفيذ الوظائف التي لم يكن الغرض منها أن يتم تنفيذها كوظيفة للتطبيق.
لكن ولحسن الحظ، حتى لو لم تكن المؤسسة أو الشركة على دراية كاملة بنقاط ضعفها، يمكن للمطور إحداث فرق كبير لتجنب أكبر 10 نقاط ضعف في تطبيقات الويب. وتعد الإشارة إلى مشروع أمان تطبيق الويب المفتوح (OWASP) بداية رائعة لتقليل هذه المخاطر، حيث تتواصل مع المطورين والمؤسسات لمساعدتهم على إدارة مخاطر تطبيقات الويب بشكل أفضل. فيما يلي شرح موجز لأهم عشرة مخاطر أمنية في OWASP:

أولاً: الحقن SQL Injection

يتكون هجوم حقن SQL من إدخال أو حقن استعلام SQL عبر بيانات الإدخال من العميل إلى التطبيق. يمكن لاستغلال حقن SQL الناجح قراءة البيانات الحساسة من قاعدة البيانات، وتعديل بيانات قاعدة البيانات (إدراج / تحديث / حذف) وتنفيذ عمليات الإدارة على قاعدة البيانات (مثل إيقاف تشغيل DBMS) واستعادة محتوى ملف معين موجود في ملف DBMS النظام وفي بعض الحالات إصدار أوامر لنظام التشغيل. ويعد حقن SQL شائعاً جداً مع تطبيقات PHP و ASP نظراً لانتشار الواجهات الوظيفية القديمة. كما أن شدة هجمات حقن SQL محدودة بمهارة المهاجم وخياله، وبدرجة أقل الدفاع في الإجراءات المضادة العميقة مثل اتصالات الامتيازات المنخفضة بخادم قاعدة البيانات وما إلى ذلك.
ومن أهم الجوانب السلبية لهذا الهجوم أنه سوف يؤثر بشكل فعال على كل من السرية والنزاهة والمصادقة والتفويض العائدة للموقع المستهدف.
مثال برمجي عن هذا الهجوم:

```
SELECT * FROM items
WHERE owner = 'hacker'
AND itemname = 'name';
DELETE FROM items;
SELECT * FROM items WHERE 'a'='a';
```

تتمثل إحدى الطرق التقليدية لمنع هجمات حقن SQL في استخدام التحقق من صحة الإدخال، وإما قبول الأحرف فقط من قائمة القيم الآمنة المسموح بها، أو تحديد قائمة رفض القيم التي يُحتمل أن تكون ضارة. الحل الآخر المقترح بشكل شائع للتعامل مع هجمات حقن SQL هو استخدام الإجراءات المخزنة stored procedures وعلى الرغم من أن الإجراءات المخزنة تمنع بعض أنواع هجمات حقن SQL إلا أنها تفشل في الحماية من العديد من الهجمات الأخرى، لكنها تساعد عادةً في منع هجمات الحقن من خلال تقييد أنواع العبارات التي يمكن تمريرها.

آلية الحماية:

أحد أفضل الممارسات لتحديد هجمات حقن SQL **هو وجود جدار حماية لتطبيق الويب** WAF الذي يعمل أمام مخدمات الويب على مراقبة حركة المرور التي تدخل وتخرج من هذه المخدمات وتحدد الأنماط التي تشكل تهديدًا، فهو بالأساس حاجز بين تطبيق الويب والإنترنت.

لكننا استخدمنا في مشروعنا قاعدة البيانات MongoDB وهي غير علائقية أي أنها من نمط NoSQL DB لذلك فإننا نعتبر محميين من هذا النوع من الهجوم.

ثانياً: البرمجة النصية عبر الموقع XSS-Cross Site Scripting

هي نوع من الحقن، حيث يتم حقن البرامج النصية الخبيثة في مواقع الويب الموثوقة. وتحدث هذه الهجمات عندما يستخدم المهاجم تطبيق ويب لإرسال تعليمات برمجية ضارة على شكل نص برمجي من جانب المستعرض، إلى مستخدم نهائي مختلف. تنتشر العيوب التي تسمح لهذه الهجمات بالنجاح وتحدث في أي مكان يستخدم فيه تطبيق الويب مدخلات من مستخدم داخل المخرجات التي يولدها دون التحقق من صحتها أو ترميزها. تحدث هجمات البرمجة النصية عبر المواقع (XSS) عندما:

- تدخل البيانات إلى تطبيق ويب من خلال مصدر غير موثوق به، وغالباً ما يكون طلب ويب.
- يتم تضمين البيانات في المحتوى الديناميكي الذي يتم إرساله إلى مستخدم الويب دون التحقق من صحته بحثاً عن محتوى ضار.

غالباً ما يتخذ المحتوى الضار المرسل إلى متصفح الويب نصاً من لغة البرمجة JavaScript ولكنه قد يتضمن أيضاً HTML أو Flash أو أي نوع آخر من التعليمات البرمجية التي قد ينفذها المستعرض. تعد مجموعة الهجمات التي تعتمد على XSS غير محدودة تقريباً، ولكنها تشمل عادةً نقل البيانات الخاصة مثل ملفات تعريف الارتباط أو معلومات الجلسة الأخرى إلى المهاجم، أو إعادة توجيه الضحية إلى محتوى ويب يتحكم فيه المهاجم، أو تنفيذ عمليات ضارة أخرى على جهاز المستخدم تحت ستار الموقع المعرض للخطر. لكن أخطر هذه الهجمات تكمن بالكشف عن ملف تعريف ارتباط جلسة المستخدم، مما يسمح للمهاجم باختطاف جلسة المستخدم والاستيلاء على الحساب الشخصي له. ولذلك من الضروري جداً إيقاف تشغيل دعم HTTP TRACE على جميع خوادم الويب.

أمثلة برمجية:

يقرأ مقطع كود JSP التالي معرّف الموظف eid من طلب HTTP ويعرضه للمستخدم

```
<% String eid = request.getParameter("eid"); %>
...
Employee ID: <%= eid %>
```

يستعلم مقطع كود JSP التالي عن قاعدة بيانات لموظف بمعرف معين ويطبع اسم الموظف المقابل

```
<%...
 Statement stmt = conn.createStatement();
 ResultSet rs = stmt.executeQuery("select * from emp  where id="+eid);
 if (rs != null) {
  rs.next();
  String name = rs.getString("name");
%>

Employee Name: <%= name %>
```

آلية الحماية:

بشكل عام من المرجح أن يتضمن منع ثغرات XSS بشكل فعال مجموعة من الإجراءات التالية:

- تصفية المدخلات عند الوصول input.
- تشفير البيانات على الإخراج output.
- استخدام ترويسات الاستجابة المناسبة.
- تأمين سياسة أمن المحتوى.

لكن وبما أننا نستخدم في مشروعنا البنية REST التي تؤمن تبادل المعطيات بين متصفح المستثمر ومخدم التطبيق عبر JSON دون أن تقوم بإرسال صفحات HTML، فإننا نعتبر محميين من هذا الخطر.

ثالثاً: المصادقة المكسورة وإدارة الجلسة
Broken Authentication & Session Management

يمكن أن تسمح هذه الأنواع من نقاط الضعف للمهاجم بالتقاط أو تجاوز طرق المصادقة التي يستخدمها تطبيق الويب.

- بيانات اعتماد مصادقة المستخدم غير محمية عند تخزينها.
- بيانات اعتماد تسجيل الدخول التي يمكن التنبؤ بها.
- يتم عرض معرفات الجلسات في عنوان URL (على سبيل المثال: إعادة كتابة عنوان URL).

- معرفات الجلسات عرضة لهجمات تثبيت الجلسة.
- لا تنتهي قيمة الجلسة أو لا يتم إبطالها بعد تسجيل الخروج.
- لا يتم تدوير معرفات الجلسات بعد تسجيل الدخول بنجاح.
- يتم إرسال كلمات المرور ومعرفات الجلسة وبيانات الاعتماد الأخرى عبر اتصالات غير مشفرة.

الهدف من الهجوم هو الاستيلاء على حساب أو أكثر، وأن يحصل المهاجم على نفس الامتيازات التي تتمتع بها الضحية. ومن الأمثلة عن هذا الهجوم عندما يتم وضع معرفات الجلسة session ID's ضمن العنوان URL.

http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii

يريد المستخدم في المثال أعلاه من خلال الموقع الإلكتروني السماح لأصدقائه بمعرفة سجل المبيعات حيث يرسل المستخدم الرابط أعلاه عبر البريد الإلكتروني دون أن يدرك أنه يتخلى أيضاً عن معرف الجلسة. عندما يستخدم الأصدقاء الرابط فإنهم يستخدمون جلسة المستخدم وبطاقة الائتمان الخاصة به أيضاً.

ويمكن منع هذا الهجوم عن طريق كل من:

إدارة الجلسة:

- يجب حماية بيانات اعتماد مصادقة المستخدم عند تخزينها باستخدام التجزئة أو التشفير.
- لا تكشف معرّف الجلسة في عنوان URL.
- يجب أن تنتهي مهلة معرفات الجلسة بشكل صحيح أثناء تسجيل الخروج.
- إعادة إنشاء معرّفات الجلسة بعد تسجيل الدخول بنجاح.
- لا ترسل بيانات اعتماد عبر اتصالات غير مشفرة (كلمات المرور- معرفات الجلسة ...الخ).

المصادقة المكسورة:

- طول كلمة المرور يجب ألا يقل عن ثمانية خانات، حيث يؤدي الجمع بين هذا الطول والتعقيد إلى صعوبة تخمين كلمة المرور باستخدام هجوم القوة الغاشمة.
- تعقيد كلمة المرور، يجب أن تتكون كلمات المرور من مزيج من الأحرف الأبجدية الرقمية التي تضم كل من الأحرف والأرقام وعلامات الترقيم والرموز الرياضية وغيرها من الرموز التقليدية.
- ضبط عملية تكرار اسم المستخدم/كلمة المرور، بحيث لا تشير استجابات فشل المصادقة إلى الجزء غير الصحيح من بيانات المصادقة.
- الحماية ضد تسجيل الدخول باستخدام مفهوم القوة الغاشمة، بغرض تعطيل الحساب بعد عدد محدد من محاولات تسجيل الدخول غير الصالحة ويجب تعطيل الحساب لفترة زمنية كافية.

رابعاً: تزوير الطلب عبر الموقع CSRF-Cross Site Request Forgery

هو هجوم يجبر المستخدم النهائي على تنفيذ إجراءات غير مرغوب فيها على تطبيق ويب تمت مصادقته سابقاً، حيث ينتحل المهاجم هوية وامتيازات الضحية لأداء وظيفة غير مرغوب فيها، ومع القليل من المساعدة باستعمال الهندسة الاجتماعية (مثل إرسال رابط عبر البريد الإلكتروني أو الدردشة) فقد يخدع المهاجم مستخدمي تطبيق الويب لتنفيذ الإجراءات التي يختارها. إذا كانت الضحية مستخدماً عادياً فقد يجبر هجوم CSRF الناجح المستخدم على تنفيذ طلبات تغيير الحالة مثل تحويل الأموال وتغيير عنوان بريده الإلكتروني وما إلى ذلك. أما إذا كانت الضحية حساباً إدارياً فيمكن لـ CSRF اختراق تطبيق الويب بالكامل. يجب اتباع المبادئ التالية للدفاع ضد CSRF:

- تحقق مما إذا كان إطار العمل الخاص بك يحتوي على حماية CSRF مضمنة واستخدمها.
- استخدم دائماً سمة SameSite Cookie لملفات تعريف الارتباط للجلسة.
- تنفيذ تخفيف واحد على الأقل من قسم الدفاع في تخفيف العمق Defense in Depth Mitigations.
    - استخدم ترويسات الطلبات المخصصة request headers.
    - تحقق من الأصل باستخدام الترويسات القياسية standard headers.
    - استخدم ملفات تعريف الارتباط المزدوجة double submit cookies.
- ضع في اعتبارك تنفيذ الحماية القائمة على تفاعل المستخدم للعمليات شديدة الحساسية.
- تذكر أنه يمكن استخدام أي برمجة نصية عبر المواقع (XSS) للتغلب على جميع تقنيات التخفيف من CSRF.
- لا تستخدم طلبات GET لعمليات تغيير الحالة، إذا قمت بذلك لأي سبب من الأسباب، فيجب عليك أيضًا حماية هذه الموارد من CSRF

أخيراً فإنه يمكن إضافة رمز CSRF المميز من خلال الحقول المخفية والعناوين، ويمكن استخدامه مع النماذج ومكالمات AJAX. ويجب التأكد من عدم تسريب الرمز المميز في سجلات الخادم أو في عنوان URL. ومن المحتمل أن يتم تسريب رموز CSRF في طلبات GET في عدة مواقع، مثل محفوظات المتصفح وملفات السجل وأجهزة الشبكة التي تسجل السطر الأول من طلب HTTP وترويسات الإحالة إذا كان الموقع المحمي يرتبط بموقع خارجي. وفي مشروعنا استخدمنا الرمز JWT الذي يضمن التحقق من هوية مستثمر التطبيق عند كل عملية إدخال أو إخراج.

أمثلة برمجية:

```
<"form action="/transfer.do" method="post>
input type="hidden" name="CSRFToken">
value="OWY4NmQwODE4ODRjN2Q2NTlhMmZlYWEwYzU1YWQwMT
<"==VhM2JmNGYxYjJiMGI4MjJjZDE1ZDZMGYwMGEwOA
[...]
<form/>
```

خامساً: التكوين الأمني الخاطئ Security Misconfiguration

يتم تعريف التهيئة الخاطئة للأمان ببساطة على أنها الفشل في تنفيذ جميع ضوابط الأمان لخادم أو لتطبيق ويب، أو تنفيذ ضوابط الأمان ولكن مع وجود أخطاء. وهذه مشكلة واسعة الانتشار ويمكن أن تحدث على أي مستوى من مكدس التطبيق App Stack. يتزايد هذا التحدي في مراكز البيانات المختلطة والبيئات السحابية اليوم، ومع تعقيد التطبيقات وأنظمة التشغيل وأطر العمل وأعباء العمل.

هذه البيئات متنوعة من الناحية التكنولوجية وتتغير بسرعة، مما يجعل من الصعب فهم وتقديم عناصر التحكم الصحيحة للتكوين الآمن، بدون المستوى الصحيح من الرؤية، يفتح خطأ التكوين الأمني مخاطر جديدة للبيئات غير المتجانسة، وتشمل هذه:

- منافذ إدارة غير ضرورية ومفتوحة لتطبيق ما، وهذه تعرض التطبيق للهجمات عن بعد.
- الاتصالات الصادرة لمختلف خدمات الإنترنت، ويمكن أن تكشف هذه سلوكيات غير مرغوب فيها للتطبيق في بيئة حرجة.
- التطبيقات القديمة التي تحاول الاتصال بالتطبيقات التي لم تعد موجودة، ويمكن للمهاجمين تقليد هذه التطبيقات لإنشاء اتصال.

ويمكن الحد من خطورة هذا التهديد بشكل كبير من خلال الآتي:

- تقييد الوصول إلى واجهات المسؤول Administrator.
- تعطيل التصحيح Debugging.
- تعطيل استخدام الحسابات الافتراضية وكلمات المرور.
- تعطيل قائمة الدليل Directory Listing.
- تصحيح البرامج وتحديثها بانتظام.
- إزالة الميزات غير المستخدمة.
- استخدم الأتمتة لصالحك.

سادساً: التخزين المشفر غير الآمن Insecure Cryptographic Storage

هو عبارة عن ثغرة أمنية شائعة، تحدث عندما لا يتم تخزين البيانات الحساسة بشكل آمن. لا يعد التخزين المشفر غير الآمن ثغرة أمنية واحدة، ولكنه مجموعة من الثغرات الأمنية. ترتبط نقاط الضعف في المجموعة بالتأكد من تشفير البيانات الأكثر أهمية عند الحاجة إلى ذلك. هذا يتضمن:

- التأكد من القيام بتشفير البيانات الصحيحة.
- التأكد من توافر التخزين والإدارة المناسبين للمفاتيح.
- التأكد من عدم استخدام خوارزميات سيئة معروفة.
- التأكد من عدم القيام بتطبيق التشفير الخاص بك، والذي قد يكون آمنًا وقد لا يكون كذلك.

غالباً ما يفترض المطورون أن تخزين البيانات لن يتم فحصه بواسطة مستخدم عشوائي. لكن العديد من مستخدمي التطبيق أو البرنامج لديهم حق الوصول إلى السجل والملفات المؤقتة وقواعد البيانات. يمكن لهؤلاء المستخدمين الوصول إلى البيانات الحساسة بتنسيقها غير المشفر باستخدام ملفات التسجيل المؤقتة والمخفية. الشيء الذي قد يستغله المهاجم بصورة فعالة جداً، ولضمان التخزين الآمن للبيانات الحساسة اتبع الخطوات التالية:

- تحديد جميع البيانات الحساسة والقيام بتشفيرها حتى عند تخزينها على القرص الصلب.
- التأكد من عدم إمكانية الكتابة فوق البيانات الحساسة.
- الكتابة فوق مواقع الذاكرة الحساسة مباشرة بعد عدم الحاجة إلى البيانات في الذاكرة.
- تحديد الأشخاص الذين لديهم الأحقية بمعرفة الأسرار، والأشخاص الذين ليس لديهم ذلك.
- احتفظ بالأسرار مثل الخوارزميات الخاصة ومفاتيح التشفير وإدارة الحقوق الرقمية بعيدة عن الأشخاص الغير مخولين بها وحتى من المسؤول.
- تحديد البيانات الحساسة المقروءة في الذاكرة، واستبدلها ببيانات عشوائية واستخدم تشفيراً قوياً لحمايتها.

سابعاً: فشل في تقييد الوصول إلى العنوان Failure to Restrict URL Access

هي إحدى الثغرات الأمنية الشائعة المدرجة في مشروع أمان تطبيق الويب المفتوح (OWASP) فإذا فشل تطبيقك في تقييد الوصول إلى عنوان URL بشكل مناسب، يمكن اختراق الأمان من خلال تقنية تسمى التصفح الإجباري أو القسري. يمكن أن يكون التصفح القسري مشكلة خطيرة للغاية إذا حاول المهاجم جمع بيانات حساسة من خلال مستعرض ويب عن طريق طلب صفحات معينة أو ملفات بيانات. باستخدام هذه التقنية يمكن للمهاجم تجاوز أمان موقع الويب عن طريق الوصول إلى الملفات مباشرة بدلاً من الروابط اللاحقة. ببساطة يحدث الفشل في تقييد الوصول إلى عنوان URL عندما يؤدي

خطأ ما في إعدادات التحكم في الوصول، إلى قدرة المستخدمين على الوصول إلى الصفحات التي من المفترض أن تكون مقيدة أو مخفية.
إن أفضل طريقة لضمان حماية الموقع هي استخدام تحليل التعليمات البرمجية جنباً إلى جنب مع اختبار الأمان عبر الموقع. من خلال هذه العملية يمكن للمطورين ومختبري الأمان التأكد من أن سياسة التحكم في الوصول الخاصة بهم فعالة وتمتد إلى كل صفحة من صفحات الموقع. أخيراً من المهم إجراء الاختبار قبل الإطلاق للتحقق بشكل استباقي من حماية الصفحات.

ثامناً: الحماية غير الكافية لطبقة النقل Inefficient Transport Layer Security

هي نقطة ضعف أمنية ناتجة عن عدم اتخاذ التطبيقات أي تدابير لحماية حركة مرور الشبكة. أثناء المصادقة قد تستخدم التطبيقات SSL-TLS لكنها غالباً ما تفشل في استخدامها في مكان آخر ضمن التطبيق، مما يترك البيانات ومعرفات الجلسة مكشوفة. كما تنص OWASP كثيراً ما تفشل التطبيقات في المصادقة والتشفير وحماية سرية وسلامة حركة مرور الشبكة الحساسة، وعندما تفعل ذلك فإنها تدعم أحياناً خوارزميات ضعيفة أو تستخدم شهادات منتهية الصلاحية أو غير صالحة أو لا تستخدمها بشكل صحيح.

تاسعاً: هجمات رفض الخدمة (DOS) ورفض الخدمة المنعكس (RDOS):

هجوم رفض الخدمة (DoS) هو محاولة لجعل النظام غير متاح مثل منع الوصول إلى موقع ويب، حيث يستهلك هجوم DoS الناجح جميع موارد الشبكة أو النظام المتاحة، مما يؤدي عادةً إلى تباطؤ أو تعطل الخادم. وعندما تقوم مصادر متعددة بالتنسيق في هجوم DoS فإن ذلك يُعرف باسم هجوم DDoS. يلاحظ المعيار MS-ISAC نوعين رئيسيين لهذا الهجوم وهما النوع القياسي Standard والنوع المنعكس Reflection.

يحدث هجوم DDoS القياسي عندما يرسل المهاجمون قدراً كبيراً من حركة مرور الشبكة المشوهة مباشرةً إلى المخدم المستهدف. إحدى الطرق التي يمكن للمهاجم من خلالها تحقيق ذلك هي استخدام الروبوتات botnets لإرسال حركة المرور مع العرض بأن الروبوتات عبارة عن عدد كبير من أجهزة الكمبيوتر الضحية أو ما يعرف بالزومبي المتصلة عبر الإنترنت، والتي تتواصل مع بعضها البعض ويمكن التحكم فيها من مكان واحد. عندما يستخدم المهاجم شبكة الروبوتات لتنفيذ هجوم DDoS فإنهم يرسلون التعليمات إلى بعض أو كل أجهزة الزومبي المتصلة بها، وبالتالي تكبير حجم هجومهم، مما يجعله ينشأ من شبكات متعددة وربما من بلدان متعددة.

بينما يحدث هجوم DDoS الانعكاسي عندما ينتحل المهاجمون عنوان IP الخاص بالهدف ليقوموا بدور الضحية المقصودة، ثم يرسلون طلبات مشروعة إلى الخوادم الشرعية التي تواجه الجمهور. يتم إرسال الردود على هذه الطلبات إلى الضحية المقصودة وتنشأ من خوادم شرعية.

بالإضافة إلى هذه الأساليب، فإن الأسلوب الذي يستخدمه المهاجمون لزيادة فعالية هجومهم يسمى التضخيم، ويحدث التضخيم عندما تكون الاستجابة المرسلة إلى الضحية أكبر من الطلب الذي تم إرساله

من المهاجم. بالإضافة إلى استخدام شبكات الروبوت (botnet) تتوفر بعض الأدوات مجاناً عبر الإنترنت، والتي يمكن لممثلي التهديدات الإلكترونية استخدامها لتنفيذ هجمات DDoS. تم تصميم معظم هذه الأدوات في الأصل لتكون بمثابة أدوات اختبار ضغط ومنذ ذلك الحين أصبحت أدوات مفتوحة المصدر تُستخدم لشن هجمات DDoS من قِبل هواة تهديدات إلكترونية.

أنواع هجمات DDoS القياسية:

- الفيضان المتزامن SYN Flood.
- فيضان الـ UDP.
- الهجمات من نوع SMBLoris وهو عبارة عن هجوم DDoS على مستوى التطبيق.
- فيضان ICMP الذي يحدث عندما يستخدم المهاجم شبكة الروبوتات لإرسال عدد كبير من حزم ICMP إلى الخادم الهدف في محاولة لاستهلاك عرض الحزمة المتاح، ورفض وصول المستخدمين الشرعيين.
- فيضان HTTP GET.

أنواع هجمات DDoS المنعكسة:

- هجوم انعكاس NTP مع التضخيم.
- هجوم انعكاس DNS مع التضخيم.
- هجوم انعكاس CLDAP مع التضخيم.
- هجوم انعكاس Pingback Reflection مع تضخيم.
- هجوم انعكاس SSDP مع التضخيم.
- هجوم انعكاس Microsoft SQL مع التضخيم.

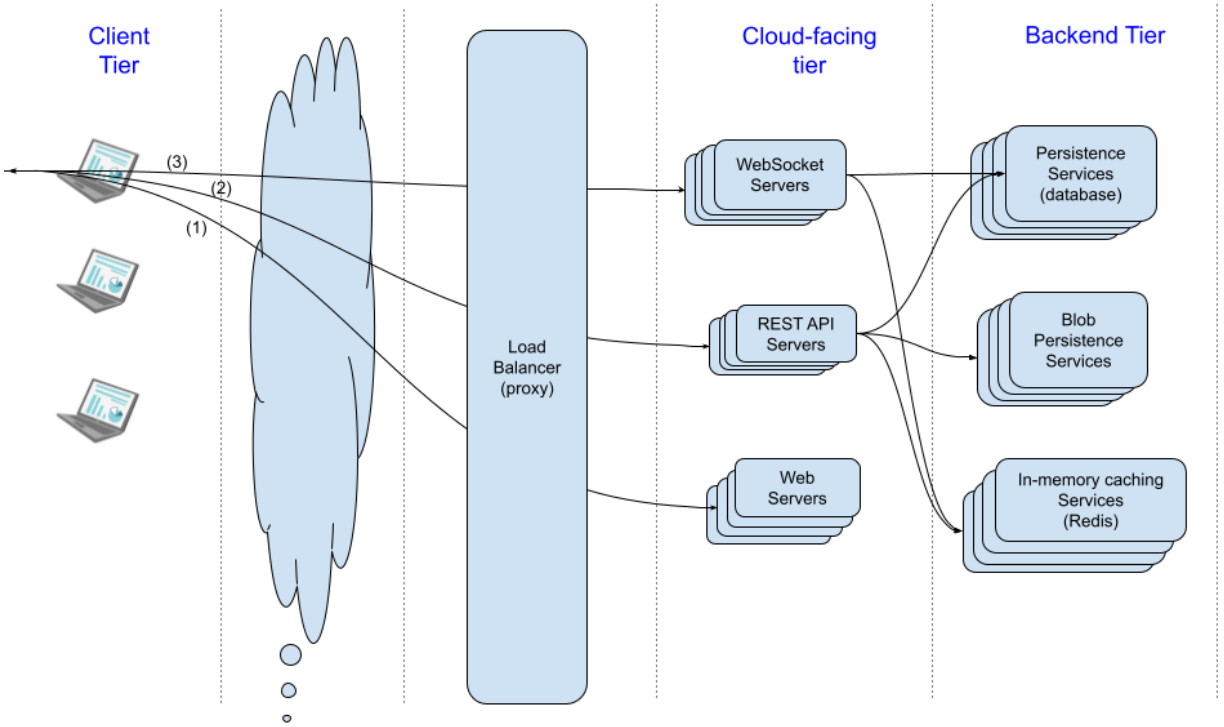
عاشراً: هجوم الرجل في المنتصف MITM

عندما ينجح المتسلل في اقحام نفسه بين طرفي اتصال، يُسمى هذا النوع من الهجوم (هجوم الرجل في الوسط - MITM) وبهذه الطريقة يمكن للمتسلل اعتراض البيانات بين مضيف المصدر والوجهة، ويمكنه تعديل البيانات وإعادة إرسالها إلى المضيف الوجهة وأيضاً إدخال أي نوع من البيانات الخاطئة. يمكن أن تؤثر هجمات MITM على توافر البيانات وسريتها وسلامتها ومصداقيتها. لكن يمكن للتشفير القوي أن يخفف من هذا النوع من الهجوم، حيث يوفر SSL وSSH واستخدام IPSec أيضاً أماناً خاصاً من طرف إلى طرف end to end حيث يتم تشفير الاتصال بالكامل.

الفصل الثالث: وصف المشروع

3-1 بنية النظام System Architecture:

يمكننا شرح بنية النظام المستخدم في مشروعنا من خلال المخطط التالي:



الشكل 3-1: يوضح بنية النظام للمشروع

بالنسبة لطبقة الزبون Client-Tier فقد تم استخدام مكتبة React من أجل بناء الواجهة الأمامية front-end لكامل المشروع كونها تعتبر المكتبة الأولى لإنشاء تطبيقات ويب كبيرة وسريعة باستخدام JavaScript حيث تم إنشاء ما يعرف بالـ SPA وتم استخدام المكونات الأساسية Components من أجل بناء الأجزاء الرئيسية والصفحات الفرعية الخاصة بمستثمر التطبيق للوصول إلى واجهات المستخدم المطلوبة UI بالشكل الأمثل. ومن ناحية أخرى فقد تم استخدام Expo كونه إطار عمل

ومنصة لتطبيقات React العالمية، وهو عبارة عن مجموعة من الأدوات والخدمات المبنية على React Native والمنصات الأصلية التي تساعدنا على تطوير وبناء ونشر وتكرار التطبيقات على iOS وAndroid وتطبيقات الويب من نفس كود، وبالتالي تمكنا من بناء تطبيق متعدد المنصات.

أما بالنسبة للطبقة الخلفية Backend-Tier فقد تم استخدام NodeJS وهي عبارة عن بيئة تشغيل JavaScript مفتوحة المصدر ومتعددة المنصات وخلفية تعمل على محرك V8 وتنفذ كود JavaScript خارج متصفح الويب. حيث تتيح NodeJS للمطورين استخدام JavaScript لكتابة أدوات سطر الأوامر والبرمجة النصية من جانب المخدم، أي تشغيل البرامج النصية من جانب المخدم لإنتاج محتوى صفحة ويب ديناميكي قبل إرسال الصفحة إلى متصفح الويب الخاص بالمستخدم. مع العرض بأننا قد استخدمنا في تطبيقنا العديد من المكتبات المرتبطة مع NodeJS حيث نذكر من أهمها:

- Express هو إطار عمل تطبيق ويب خاص بـ NodeJS مبسط ومرن يوفر مجموعة قوية من الميزات لتطوير تطبيقات الويب والجوال. وفيما يلي بعض الميزات الأساسية له:
  - يسمح بإعداد البرامج الوسيطة للرد على طلبات HTTP.
  - يحدد جدول التوجيه الذي يتم استخدامه لأداء إجراءات مختلفة بناءً على طريقة HTTP وعنوان URL.
  - يسمح بعرض صفحات HTML ديناميكيًا بناءً على تمرير الوسائط إلى القوالب.
  - وأهم شيء فإننا استخدمنا Express من أجل إنشاء تطبيق يحقق المعيارية REST.
- JWT اختصار لـ JSON Web Token وهو معيار إنترنت مفتوح (RFC 7519) لنقل المعلومات الموثوقة بشكل آمن بين الأطراف بطريقة مضغوطة. تحتوي الرموز على مطالبات تم ترميزها ككائن JSON وموقعة رقميًا باستخدام سر خاص أو باستخدام المفتاح العام ضمن معيار PKI حيث تتحقق الرموز المميزة الموقعة من سلامة المطالبات الموجودة في الرمز المميز، بينما تخفي الرموز المشفرة المطالبات من الأطراف الأخرى. ويمكن استخدامه أيضًا لتبادل المعلومات على الرغم من استخدامه بشكل أكثر شيوعًا للترخيص، لأنه يوفر الكثير من المزايا على إدارة الجلسة باستخدام الرموز المميزة العشوائية في الذاكرة. أكبرها هو تمكين تفويض منطق المصادقة لمخدم جهة خارجية مثل AuthO وما إلى ذلك. مع العرض بأن JWT يتألف من ثلاثة أجزاء رئيسية أولها Header الذي يحتوي على قائمة عمليات التشفير التي يتم تطبيقها على JWT، وثانيها Payload الذي يحتوي على البيانات الفعلية التي سيتم نقلها باستخدام الرمز المميز، ويُعرف هذا الجزء أيضًا باسم جزء "المطالبات" من رمز JWT المميز، ويمكن أن تكون المطالبات من ثلاثة أنواع (مسجلة - عامة - خاصة) أما الجزء الثالث فهو Signature الذي يُستخدم للتحقق من أن الرسالة لم تتغير على طول الطريق، وإذا تم توقيع الرموز المميزة باستخدام مفتاح خاص، فإنه يتحقق أيضًا من أن المرسل هو من يدعي ذلك، يتم إنشاؤه باستخدام الرأس المشفر والحمولة المشفرة والسر والخوارزمية المحددة في الرأس، وهذه الأجزاء الرئيسية تكون وفق الصيغة [Header].[Payload].[Signature]
- passport هو برنامج وسيط للمصادقة متوافق مع Express إن الغرض الوحيد منه هو مصادقة الطلبات، وهو ما يفعله من خلال مجموعة قابلة للتوسيع من المكونات الإضافية

المعروفة باسم الاستراتيجيات، لا يقوم Passport بتثبيت المسارات أو يفترض أي مخطط قاعدة بيانات معين، مما يزيد من المرونة ويسمح للمطور باتخاذ قرارات على مستوى التطبيق وفيه واجهة برمجة التطبيقات بسيطة حيث تقدم إلى Passport طلبًا للمصادقة الذي يوفر بدوره روابط للتحكم في ما يحدث عند نجاح المصادقة أو فشلها.

- passport-jwt وهي استراتيجية Passport للمصادقة باستخدام JWT وتتيح هذه الوحدة مصادقة المسارات النهائية endpoints باستخدام رمز ويب JSON مع العرض بأن الغرض منه هو استخدامه لتأمين المسارات النهائية لتطبيقات الـ RESTful بدون جلسات.
- mongodb تُستخدم هذه الطريقة لتوصيل مخدم MongoDB بتطبيق NodeJS الخاص بنا، وهي طريقة غير متزامنة من وحدة MongoDB.
- websocket هذا تطبيق JavaScript خالص لإصدارات بروتوكول WebSocket ولـ NodeJS .

وسوف نقوم بعد ذلك بشرح المشروع وبشكل مفصل.

طبقة الواجهة الخلفية Backend Tier Persistence Services

- خدمات قاعدة المعطيات database
- خدمات التخزين المؤقت في الذاكرة Redis
- خدمات تخزين الأغراض كبيرة الحجم BLOB

أولاً: خدمات قاعدة المعطيات

في علم الكمبيوتر النظري تنص نظرية CAP على أنه من المستحيل أن يوفر مخزن البيانات الموزعة في وقت واحد أكثر من اثنين من الضمانات الثلاثة التالية:

- ➢ التناسق Consistency: تتلقى قراءة كل حدث كتابة أو خطأ.
- ➢ التوفر Availability: يتلقى كل طلب استجابة (بدون خطأ) دون ضمان احتوائه على أحدث كتابة.
- ➢ التسامح مع التقسيم Partition Tolerance: يستمر النظام في العمل على الرغم من إسقاط (أو تأخير) عدد عشوائي من الرسائل من قبل الشبكة بين العقد.

عندما يحدث فشل في قسم الشبكة، يجب أن نقرر إما إلغاء العملية وبالتالي تقليل التوافر availability ولكن مع ضمان التناسق consistency أو المضي قدماً في العملية وبالتالي تأمين التوافر، ولكن المخاطرة بمفهوم التناسق. حيث تشير نظرية CAP إلى أنه في وجود قسم الشبكة، يتعين على المرء الاختيار بين التناسق والتوافر.

من الناحية العملية لا يوجد نظام موزع آمن من فشل الشبكة، وبالتالي يجب التسامح بشكل عام مع تقسيم الشبكة. في حالة وجود التقسيم يتم الخيار ما بين التناسق أو التوفر.

عند اختيار التناسق على التوافر سيعيد النظام خطأ أو انقضاء المهلة، إذا تعذر ضمان تحديث معلومات معينة بسبب تقسيم الشبكة. وعند اختيار التوافر بدلاً من التناسق سيعمل النظام دائمًا على معالجة

الاستعلام ومحاولة إرجاع أحدث إصدار متاح من المعلومات، حتى لو لم يكن يضمن تحديثها بسبب تقسيم الشبكة. أما في حالة عدم وجود فشل في الشبكة أي عندما يعمل النظام الموزع بشكل طبيعي يمكن تلبية كل من التوافر التناسق معاً. وفي حال استخدام قواعد المعطيات من نوع NoSQL فإنه يتم اختيار التوافر على حساب التناسق، كما هو الحال مع قواعد المعطيات MongoDB.

مع التأكيد على أننا قد استخدمنا قاعدة المعطيات من نوع NoSQL في مشروعنا لتأمين ضمان قابلية التوسع بشكل ديناميكي وفعال جداً، كما أننا استخدمنا MongoDB تحديداً كونها الأكثر شيوعاً للاستخدام مع NodeJS. وأخيراً فقد تم التفاعل بين قاعدة المعطيات MongoDB المستخدمة وخدمات واجهات التطبيق البرمجية API Services في الحالات التالية:

- عند الاشتراك بالموقع لأول مرة sign up.
- عند تسجيل الدخول sign in.
- عند إنشاء لائحة دردشة جديدة create chat list.
- عند زيارة لائحة دردشة قديمة revisit old chat list.
- عند كتابة رسالة ضمن لائحة الدردشة create new message.
- عند تسجيل الخروج logout.

ثانياً: خدمات التخزين المؤقت في الذاكرة Redis

Remote Dictionary Server وهو عبارة عن مخزن لهيكل البيانات في الذاكرة، يستخدم كمخزن لقاعدة بيانات من نوع (مفتاح/قيمة) في الذاكرة، وكذلك ذاكرة تخزين مؤقت ووسيط رسائل مع استمرارية اختيارية. يدعم Redis أنواعًا مختلفة من هياكل البيانات المجردة مثل السلاسل والقوائم والخرائط والمجموعات والمجموعات المصنفة و HyperLogs والصور النقطية والتدفقات والمؤشرات المكانية.

روجت Redis لفكرة نظام يمكن اعتباره في نفس الوقت متجرًا وذاكرة تخزين مؤقت، باستخدام تصميم يتم فيه دائمًا تعديل البيانات وقراءتها من ذاكرة الكمبيوتر الرئيسية، ولكن يتم تخزينها أيضًا على قرص بتنسيق غير مناسب للعشوائية، وذلك فقط لإعادة بناء البيانات مرة أخرى في الذاكرة بمجرد إعادة تشغيل النظام. يوفر Redis نموذج بيانات غير معتاد مقارنة بنظام إدارة قواعد البيانات العلائقية RDBMS حيث لا تصف أوامر المستخدم استعلامًا ليتم تنفيذه بواسطة محرك قاعدة البيانات، بل تصف عمليات محددة يتم إجراؤها على أنواع بيانات مجردة معينة. ومن ثم يجب تخزين البيانات بطريقة مناسبة لاحقًا للاسترجاع السريع، دون مساعدة من نظام قاعدة البيانات في شكل فهارس ثانوية أو تجميعات أو ميزات أخرى مشتركة لنظام RDBMS التقليدي.

يستخدم تطبيق Redis بشكل مكثف استدعاء نظام fork لتكرار العملية التي تحتفظ بالبيانات، بحيث تستمر العملية الرئيسية في خدمة العملاء، بينما تنشئ العملية الفرعية نسخة من البيانات على القرص.

ووفقًا لتصنيفات DB-Engines الشهرية فإنه غالبًا ما يكون Redis قاعدة بيانات القيمة الرئيسية الأكثر شيوعًا. عادةً ما يحتفظ Redis بمجموعة البيانات الكاملة في الذاكرة، ويمكن تحقيق الثبات في Redis من خلال طريقتين مختلفتين، إما عن طريق الالتقاط snapshot حيث يتم نقل مجموعة البيانات بشكل غير متزامن من الذاكرة إلى القرص على فترات منتظمة كملف ثنائي باستخدام تنسيق

ملف تفريغ Redis RDB، أو عن طريق التسجيل journaling حيث تتم إضافة سجل لكل عملية تعديل مجموعة البيانات إلى ملف الإلحاق فقط (AOF) في المعالجة الخلفية، ويمكن لـ Redis إعادة كتابة ملف الإلحاق فقط في الخلفية لتجنب نمو غير محدد للمجلة.

ثالثاً: خدمات BLOB-Binary Large Object

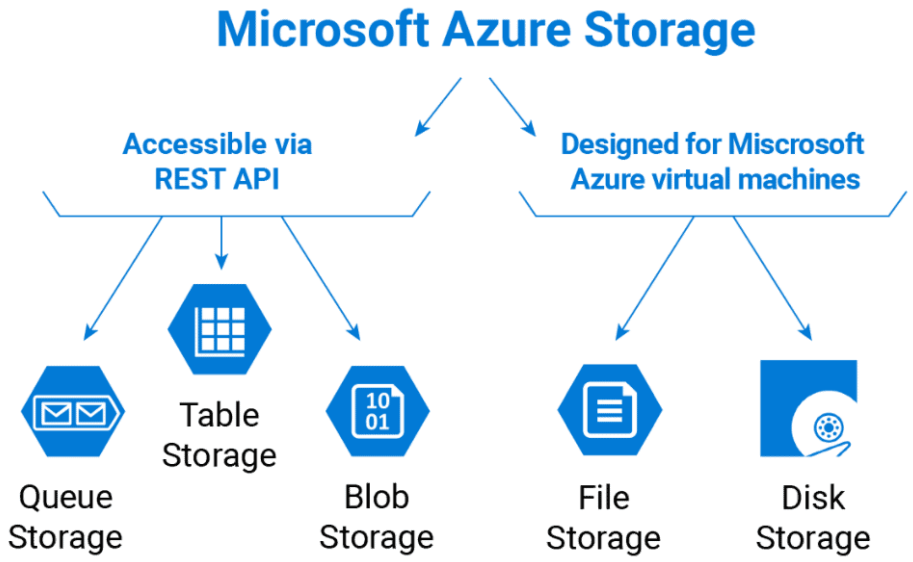
الكائن الثنائي الكبير BLOB هو مجموعة من البيانات المركزة المخزنة في ملف في قاعدة بيانات أو على برنامج معين. على الرغم من أن الاسم نفسه يمكن أن يبدو معقدًا، إلا أنه يتعلق بالبيانات الثنائية المضغوطة. هنا نتطلع إلى تحديد معنى BLOB في مساحة البيانات، وكيف تُدار من قبل الشركات.

ويعتبر ملف خام يمكن أن يحتوي على وحدة تخزين رقمية يبلغ حجمها عدة غيغابايت. يتم ضغطه في ملف واحد يتم تخزينه بعد ذلك داخل قاعدة البيانات. نظرًا لأن البيانات الثنائية لا يمكن قراءتها إلا بواسطة الكمبيوتر وتتكون من الأرقام 0 و 1 فغالبًا ما يتطلب فتحه برنامج ذو صلة. ومن أهم ملفات وأنواع BLOB النموذجية نذكر:

- Video .avi
- Audio .mp3
- Image .jpeg
- Graphics .giff

التحدي الذي تواجهه العديد من الشركات هو إدارة الكائنات الثنائية الكبيرة، غالبًا ما تكون قاعدة البيانات غير قادرة على قراءة جميع ملفات BLOB نظرًا لطبيعتها الثنائية. هذا لأن قواعد البيانات غير قادرة على تحديد مثل هذه الكميات الكبيرة من المعلومات بوضوح. أحد الخيارات عند اختيار حل تخزين لكائن تخزين البيانات الثنائية الكبيرة هو إدارة الأصول الرقمية DAM وهذا يلغي الحاجة إلى خوادم إضافية ويساعد على تحسين الكفاءة من خلال التخزين السحابي المركزي.

إن إمكانية استرداد الملفات الثنائية الكبيرة ومشاركتها أمر مهم جداً، وإذا لم يكن لدينا قواعد بيانات مجهزة للتعامل مع البيانات غير المهيكلة فقد نفقد المعلومات، ويمكن أن تتباطأ الإنتاجية أيضًا عندما نكافح للوصول إلى الملفات أو فهم ما هو موجود عليها. لذلك لا بد من إيجاد حل تخزين لا يمكنه استيعاب BLOB فحسب، بل يتأكد أيضًا من إمكانية تحديد موقع الملفات وفهمها بواسطة النظام الذي نختاره.



الشكل 3-2: يوضح استخدام Blob في شركة MS

طبقة الواجهة السحابية Cloud Facing Tier

- مخدمات الويب Web Servers
- مخدمات REST
- مخدمات WebSocket

أولاً: مخدمات الويب Web Servers

تنحصر الخدمة الرئيسية التي تقدمها هذه المخدمات بتقديم كامل حزمة الملفات الخاصة بعرض التطبيق على المتصفح والتي تحوي ملف HTML الرئيسي (كوننا نستخدم بنية SPA) بالإضافة إلى كافة ملفات JavaScript حيث يقوم المتصفح بتقديم المحتوى بالشكل الأمثل وذلك بالاستعانة بمنصة React Expo المستخدمة التي أمنت لنا عرض التطبيق على مختلف المنصات بالطريقة الأنسب.

ثانياً: مخدمات REST

REST عبارة عن مجموعة من القيود المعمارية التي تمكن من تبادل المعطيات بين التطبيقات الموجودة في الأنظمة الحاسوبية المختلفة، حيث يمكن لمطوري واجهة برمجة التطبيقات تنفيذ REST بعدة طرق. وتعتبر الترويسات Headers والمعلمات Parameters مهمة أيضًا في طرق HTTP لواجهة برمجة تطبيقات RESTful لأنها تحتوي على معلومات مهمة لبيانات تعريف الطلب والتفويض

ومعرّف الموارد الموحد URI والتخزين المؤقت وملفات تعريف الارتباط وأكثر من ذلك. وهناك ترويسات طلبات واستجابة ولكل منها معلومات اتصال HTTP ورموز الحالة الخاصة بها.

وقد استخدمنا في مشروعنا خدمات *REST* لكل من الحالات التالية:

- تقوم بمقام الوسيط بين المتصفح وقاعدة البيانات، بما يخص عمليات I/O بواسطة JSON خاصةً في كل من الأحداث التالية:

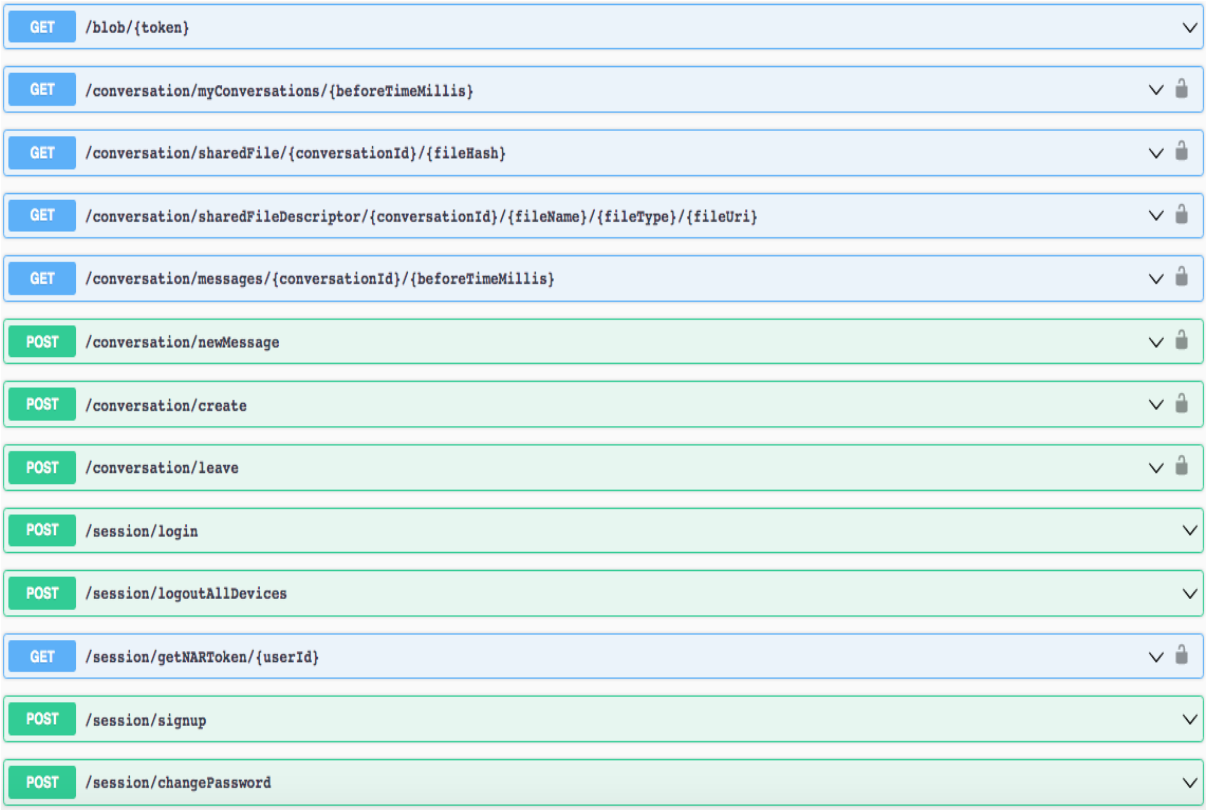
(Login, Logout, Create Conversation, Signup, Visit Old Conversation, Send Message, Leave Conversation)

- الاشتراك مع خدمة WebSocket في عدة وظائف ضمن التطبيق.
- الاتصال مع قاعدة البيانات Redis في كل من الحالات:

(Send Message, Logout, Send Message)

- تخزين واسترجاع الملفات كبيرة الحجم الموجودة ضمن BLOB.

ويبين المخطط التالي واجهات API التي استخدمناها في مشروعنا من خلال استخدامنا لمكتبة swagger:



الشكل 3-3: يوضح المسارات المستخدمة في REST API

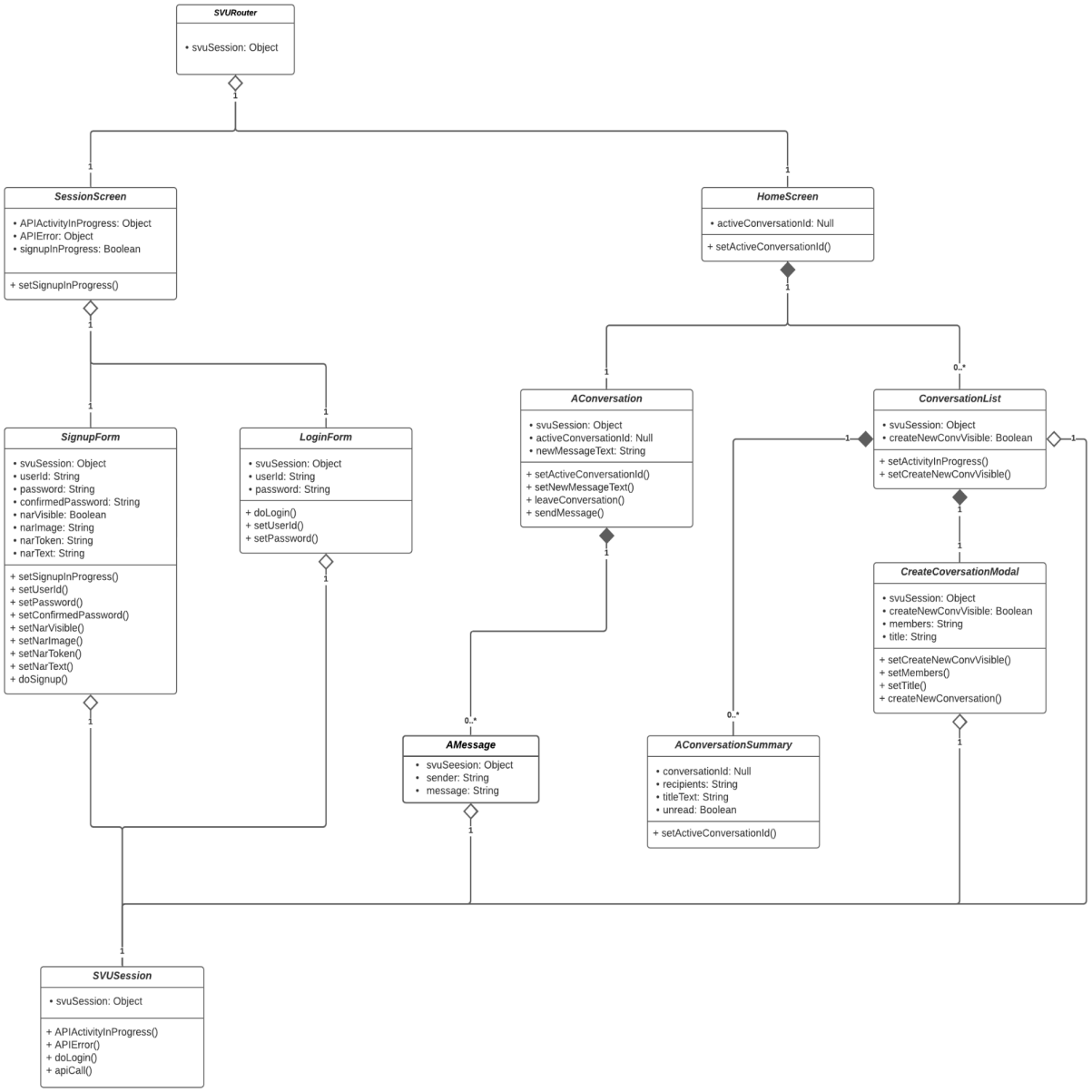
ثالثاً: مخدمات WebSocket

يتم تعريفها على أنها اتصال ثنائي الاتجاه بين الخوادم والعملاء، مما يعني أن كلا الطرفين يتواصلان ويتبادلان البيانات في نفس الوقت، يحدد هذا البروتوكول اتصال مزدوج كامل من الألف إلى الياء، حيث تأخذ خطوة إلى الأمام في توفير وظائف سطح المكتب الثرية لمتصفحات الويب حيث تمثل تطوراً طال انتظاره في تكنولوجيا الويب الخاصة بالعميل/الخادم. وقد تم استخدام هذه الخدمات في مشروعنا في كل من الأحداث:

(Login, Send Message, New Conversation, Leave Conversation)

طبقة الزبون Client Tier

هذه الطبقة معنية بالمتصفح وهنا استخدمنا React كونها المنصة البرمجية الأقوى والخاصة بلغة JS والأكثر شيوعاً عالمياً، وهي من إنتاج شركة facebook وكذلك فقد استخدمنا أيضاً React Expo التي تؤمن لنا خاصية تعدد المنصات Cross-Platforms والتي تضمن لمستخدم التطبيق إمكانية الاستثمار من قبل الأجهزة الإلكترونية المختلفة (الحواسب العادية - الحواسب اللوحية - أجهزة الهاتف الذكية) بشكل سلس وفعّال. حيث يبين الشكل التالي مخطط الصفوف الخاص بالواجهة الأمامية مع العلاقة فيما بينها:



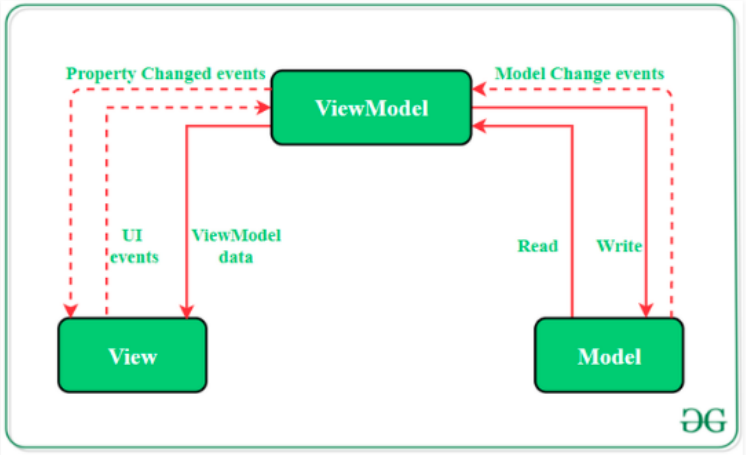
الشكل 3-4: يوضح مخطط مكونات الواجهة الأمامية ضمن React

3-2 تصميم النظام System design

تم الاعتماد على النموذج MVVM الذي يقترح فصل منطق عرض البيانات (واجهة المستخدم) عن منطق الأعمال الأساسي في التطبيق، طبقات الكود المنفصلة لـ MVVM هي:

- النموذج Model هذه الطبقة مسؤولة عن تجريد مصادر البيانات Data حيث يعمل الطراز و ViewModel معًا للحصول على البيانات وحفظها.
- طريقة العرض View الغرض من هذه الطبقة هو إعلام ViewModel بإجراء المستخدم حيث تراقب هذه الطبقة ViewModel ولا تحتوي على أي نوع من منطق التطبيق.
- ViewModel الذي يعرض تدفقات البيانات ذات الصلة بالعرض، علاوة على ذلك، فهي تعمل كحلقة وصل بين النموذج والعرض.

كما هو موضح بالشكل التالي:



الشكل 3-5: يوضح مخطط MVVM المستخدم في التطبيق

وقد تم تمثيل كل مكون من نموذج MVVM في مشروعنا وفق الآتي:

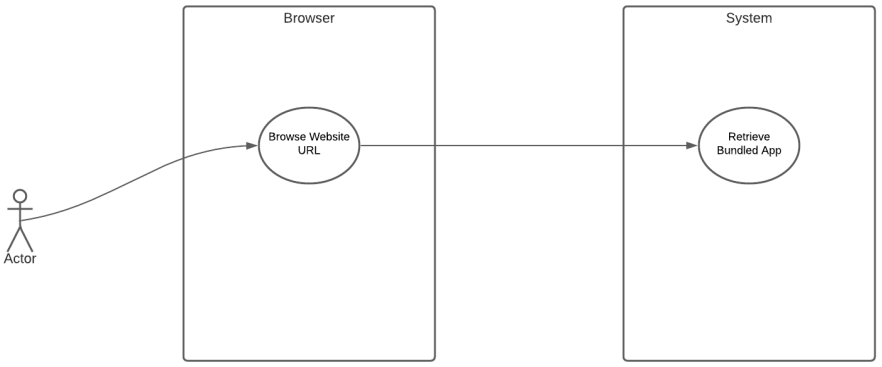
- تمثيل الـ View من خلال المكونات
  HomeScreen/ SessionScreen/ LoginForm/ SignupForm
- تمثيل المكون ViewModel من خلال المكونات
  AMessage/ ConversationList/ AConversation/ SVUSession
- تمثيل المكون Model من خلال محتوى قاعدة البيانات المستخدمة، بالإضافة إلى الحمل payload الذي يتم إرساله مع كل طلب request مرتبط بالمسار النهائي له endpoint.

وسيتم التطرق لأهم الأمثلة المستخدمة بشكل عملي في مشروعنا ضمن فصل التنفيذ والنتائج.

مخططات حالات الاستخدام والتسلسل في التطبيق

تحميل التطبيق على المتصفح Loading App:

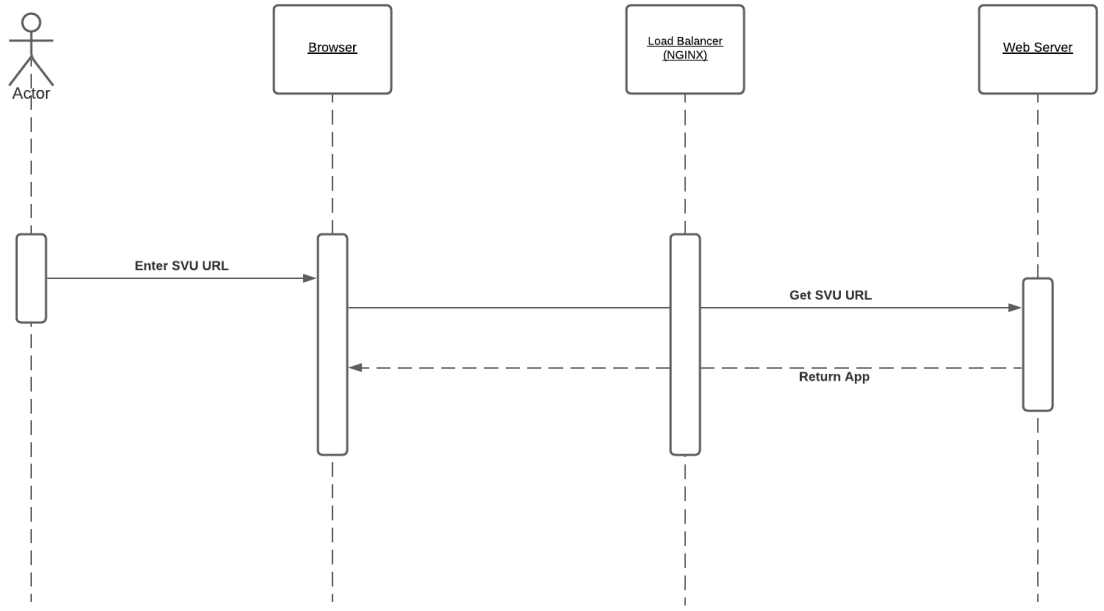
- مخطط حالة الاستخدام Use case diagram:



الشكل 3-6: يوضح مخطط حالة الاستخدام لتحميل التطبيق

يبين المخطط قيام مستثمر التطبيق بإدخال عنوانه عبر متصفح الإنترنت حيث يقوم النظام من خلال خدمة الويب Web Service بتحميل كامل حزمة التطبيق (ملفات HTML وReactJS) على المتصفح الذي يقوم بدوره بعرض المحتوى بالشكل الأمثل من خلال مفهوم SPA.

- مخطط التسلسل Sequence diagram:

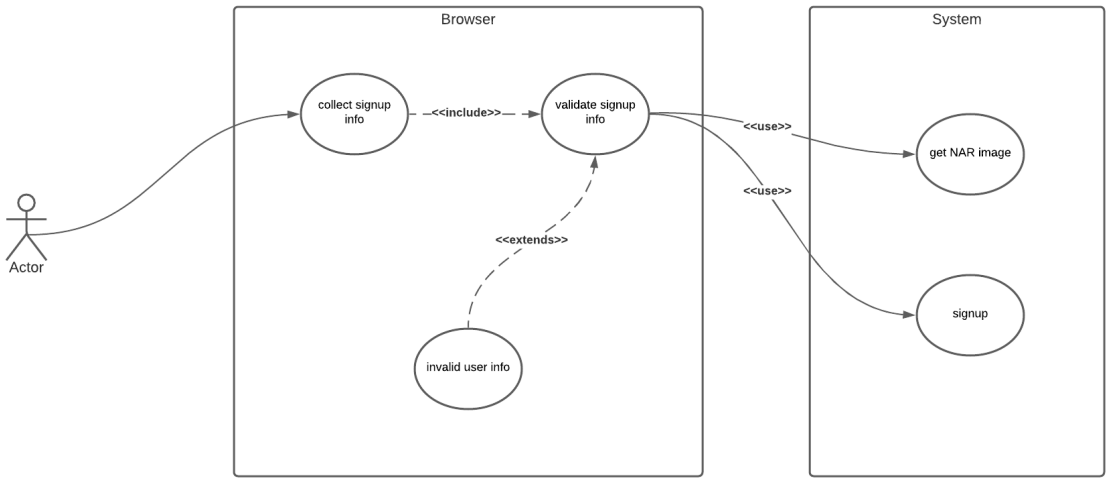


الشكل 3-7: يوضح مخطط التسلسل لتحميل التطبيق

- يقوم المستثمر بإدخال عنوان التطبيق على المتصفح.
- يقوم المتصفح بطلب محتوى التطبيق من مخدم الويب Web Server.
- يعيد مخدم الويب محتوى التطبيق كاملاً مع كافة البرمجيات المرتبطة به HTML & JS.

الاشتراك للحصول على حساب ضمن التطبيق Signup:

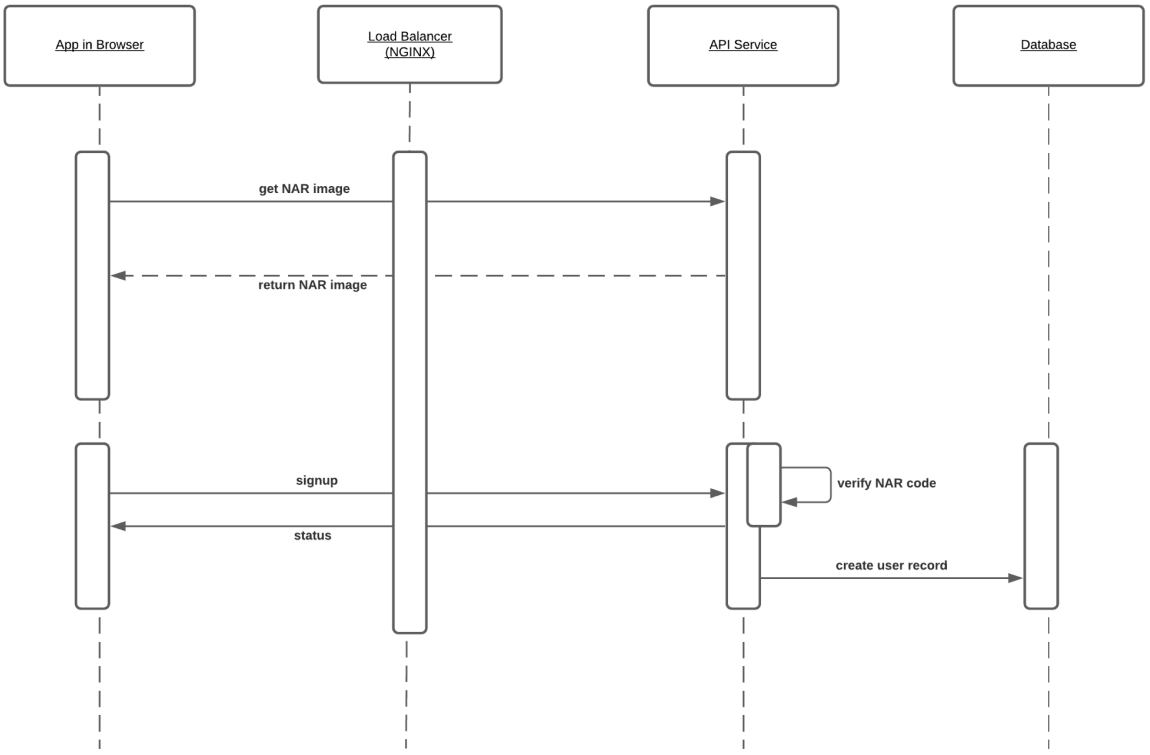
- مخطط حالة الاستخدام Use case diagram:



الشكل 3-8: يوضح مخطط حالة الاستخدام للاشتراك بالتطبيق

يقوم المستثمر عند دخوله إلى موقع التطبيق بإدخال البيانات المطلوبة (البريد الإلكتروني - كلمة السر - التأكيد على كلمة السر) وبعد التحقق من صحة وتكامل البيانات من قبل المتصفح، يقوم المتصفح بطلب الصورة الخاصة بالـ NAR من النظام وذلك من خلال API Service وبعد ذلك يقوم المستثمر بإدخال الرمز الذي يظهر ضمن الـ NAR وبنهاية العملية ينتقل المتصفح من صفحة التسجيل signup إلى صفحة الدخول login.

- مخطط التسلسل Sequence diagram:

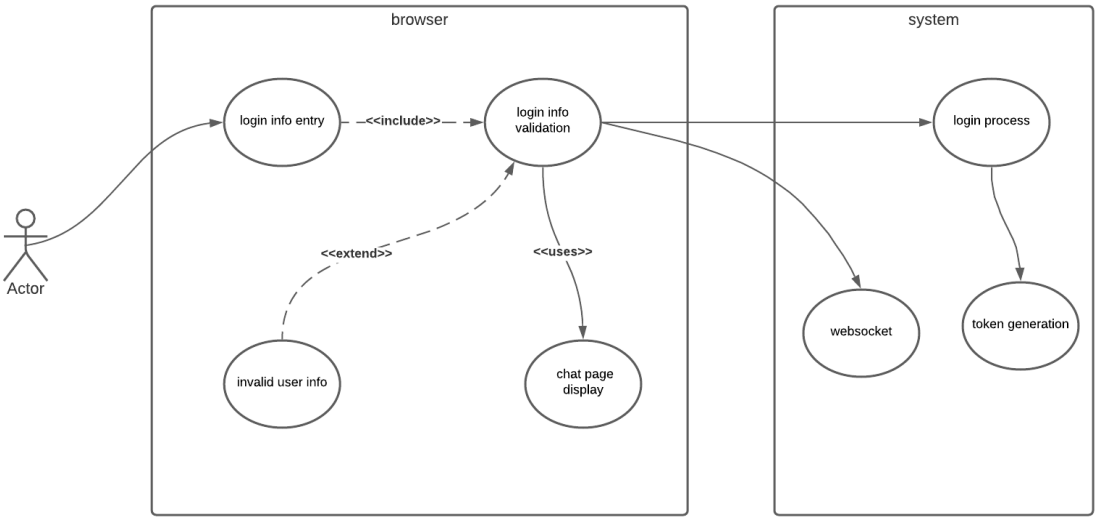


الشكل 3-9: يوضح مخطط التسلسل للاشتراك بالتطبيق

- بعد إدخال البيانات الخاصة بطلب الاشتراك يقوم المتصفح بطلب صورة NAR من قبل API services.
- تعيد API services الصورة المطلوبة إلى المتصفح.
- يقوم المستثمر بإدخال النص الموجود ضمن صورة NAR ويتم بعدها التأكد من صحة النص المدخل من قبل API services.
- بعد ذلك يتم التواصل بين API services وقاعدة المعطيات DB من أجل إنشاء المشترك الجديد، واستحضار معلومات هذا المشترك من أجل الانتقال إلى مرحلة تسجيل الدخول Login.

تسجيل الدخول إلى الحساب Login:

- مخطط حالة الاستخدام Use case diagram:

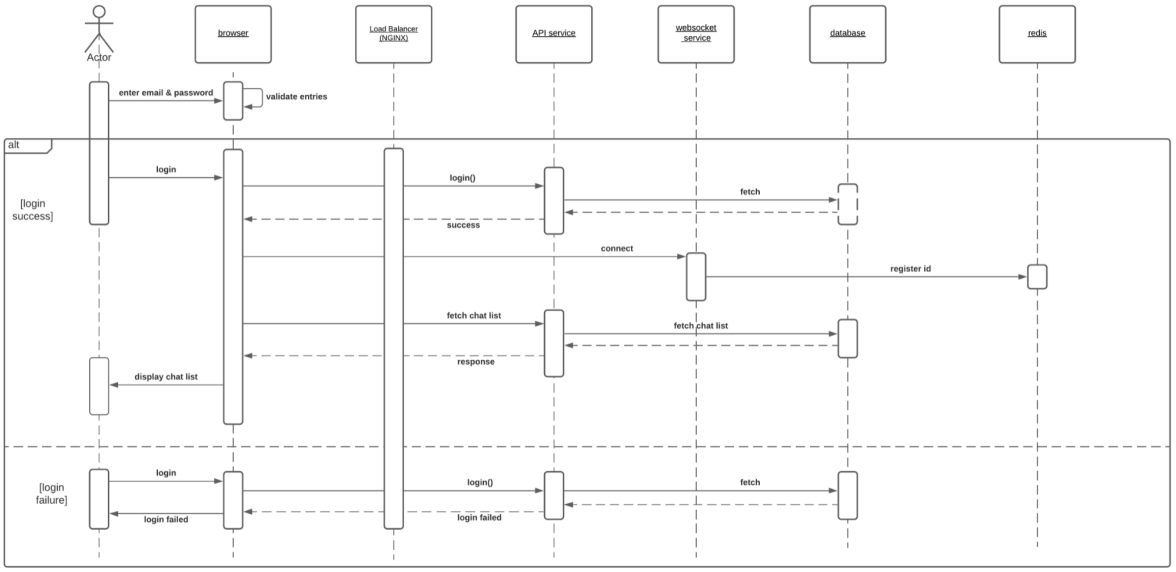


الشكل 3-10: يوضح مخطط حالة الاستخدام لتسجيل الدخول

يقوم مستثمر التطبيق بإدخال البريد الإلكتروني العائد له مع كلمة السر الخاصة به، ثم يقوم المتصفح بالتأكد من المعلومات المدخلة، وعند ذلك يقوم المتصفح بواسطة React بالانتقال إلى صفحة المحادثة Chat Page.

وبنفس الوقت يتم التواصل بين المتصفح والنظام من أجل معالجة تسجيل الدخول والحصول على الرمز Token من جهة، وفتح قناة اتصال WebSocket من جهة أخرى، وعندها يصبح مستثمر التطبيق جاهزاً لاستثمار تطبيق المحادثة بشكل فعال.

- مخطط التسلسل Sequence diagram:

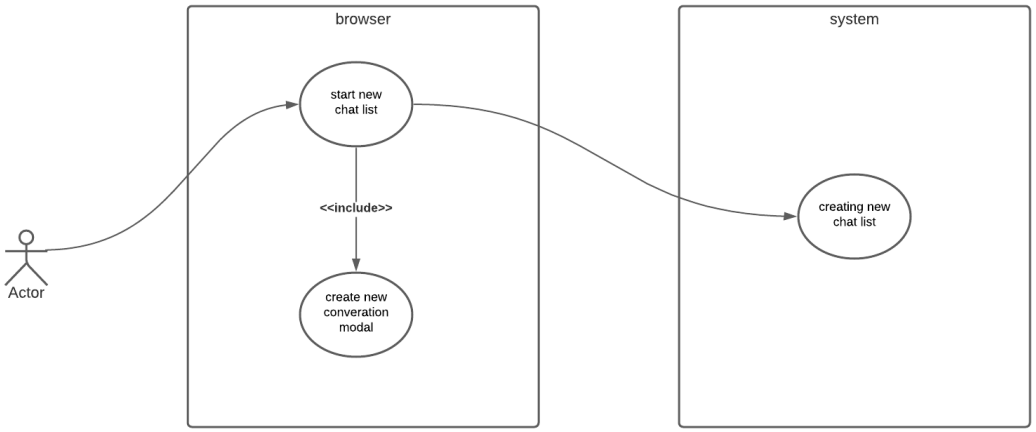


الشكل 3-11: يوضح مخطط التسلسل لتسجيل الدخول

- بعد إدخال كلمة البريد الإلكتروني وكلمة السر من قبل مستثمر التطبيق، يقوم المتصفح بالتحقق من صحة البيانات المدخلة (التحقق من صيغة البريد الإلكتروني).
- يقوم المتصفح بإرسال البريد الإلكتروني وكلمة المرور المجزأة hashed password إلى قاعدة البيانات للتأكد من معلومات المستخدم.
- بعد ذلك يتصل المتصفح مع مخدم WebSocket مرسلاً له JWT الخاصة بالمستثمر، وبعد ذلك تقوم WebSocket بتسجيل هذا المستثمر ضمن خدمات Redis.
- عند ذلك يظهر المتصفح الصفحة الخاصة بالمحادثة والتي تتضمن لائحة المحادثة الخاصة بالمستثمر.
- طبعاً كل ما ذكر أعلاه سيتم في حالة كانت البيانات المدخلة من قبل المستخدم صحيحة، أما إذا كانت غير صحيحة (البريد الإلكتروني أو كلمة السر) عند ذلك سيحدث فشل في الدخول أصولاً.

إنشاء محادثة جديدة Create New Chat List:

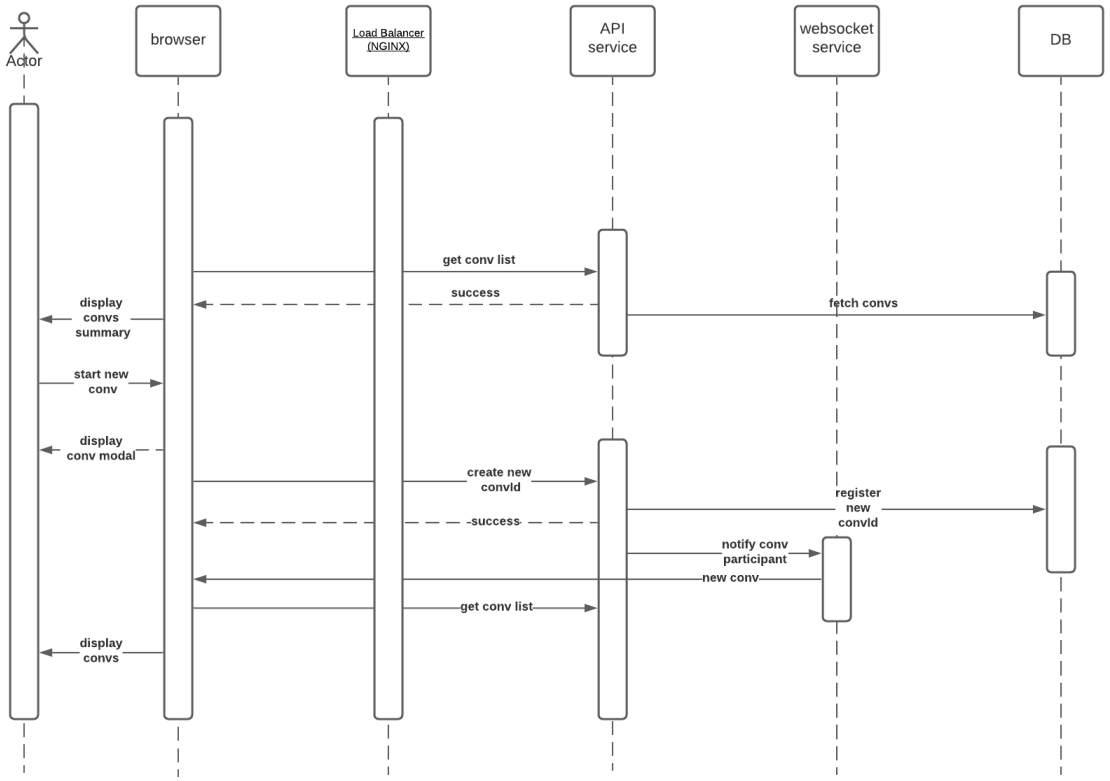
- مخطط حالة الاستخدام Use case diagram



الشكل 3-12: يوضح مخطط حالة الاستخدام لإنشاء محادثة جديدة

يقوم مستثمر التطبيق بالضغط على الزر الخاص بـإنشاء محادثة جديدة ضمن واجهة التطبيق في المتصفح، عندها تقوم React بإظهار الـ Modal الخاص بإدخال المعلومات الخاصة بالمحادثة الجديدة (عنوان البريد الإلكتروني الخاص بمستقبلي رسائل المحادثة، وعنوان المحادثة) وبعد ذلك يقوم النظام بإنشاء المحادثة الجديدة المرغوبة.

- مخطط التسلسل Sequence diagram

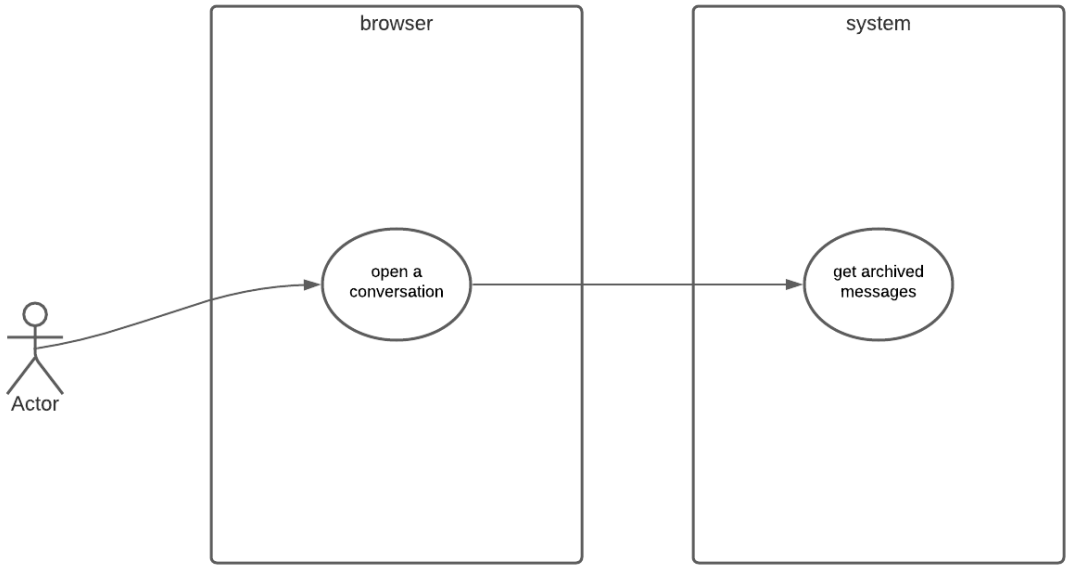


الشكل 3-13: يوضح مخطط التسلسل لإنشاء محادثة جديدة

- بعد تسجيل الدخول من قبل مستثمر التطبيق، تقوم React بعرض الصفحة الخاصة بـ ConversationList التي تظهر كافة المحادثات الخاصة بالمستثمر من خلال المكون AConversationSummary.
- عند الضغط على الزر الخاص بإنشاء محادثة جديدة، يقوم المتصفح بطلب معرف محادثة من قبل API service الذي يقوم بدوره بتسجيل المعرف الجديد ضمن قاعدة البيانات.
- يقوم المتصفح بالطلب من WebSocket service بإرسال إشعار بإنشاء المحادثة الجديدة المطلوبة إلى متصفح المستثمر.
- بعد ذلك يطلب المتصفح الحصول على لائحة المحادثات من API service.
- وأخيراً يقوم المتصفح بعرض لائحة المحادثات على مستثمر التطبيق.

مشاهدة محتوى محادثة قديمة Visit a Conversation:

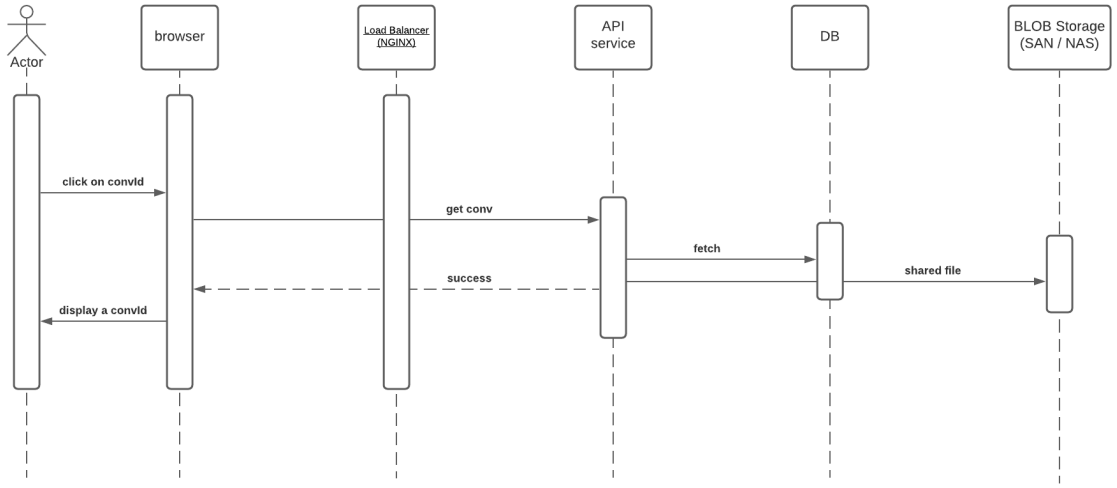
- مخطط حالة الاستخدام Use case diagram



الشكل 3-14: يوضح مخطط حالة الاستخدام لمشاهدة محتوى محادثة

يقوم مستثمر التطبيق بالضغط على المحادثة المرغوبة التي تحمل العنوان الخاص بها، والبريد الإلكتروني للمشاركين فيها، بعد ذلك تقوم React بإظهار محتوى المحادثة.

- مخطط التسلسل Sequence diagram

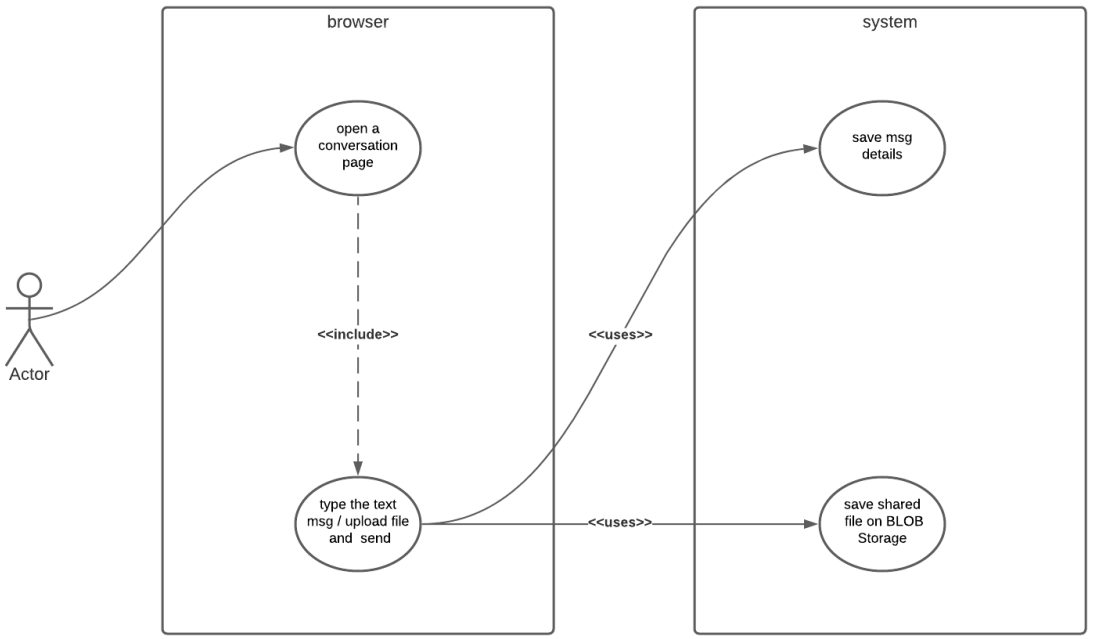


الشكل 3-15: يوضح مخطط التسلسل لمشاهدة محتوى محادثة

- عند الضغط على عنوان المحادثة المرغوب بمشاهدة محتواها، يطلب المتصفح من خدمة API الحصول على محتوى المحادثة.
- تقوم خدمة API باستحضار المحتوى من قاعدة المعطيات بالإضافة إلى الحصول على الملف المشارك من خلال BLOB، ومن ثم ترسل النتيجة إلى المتصفح.
- أخيراً يقوم المتصفح بعرض محتوى المحادثة على مستثمر التطبيق.

إرسال رسالة جديدة Send New Message:

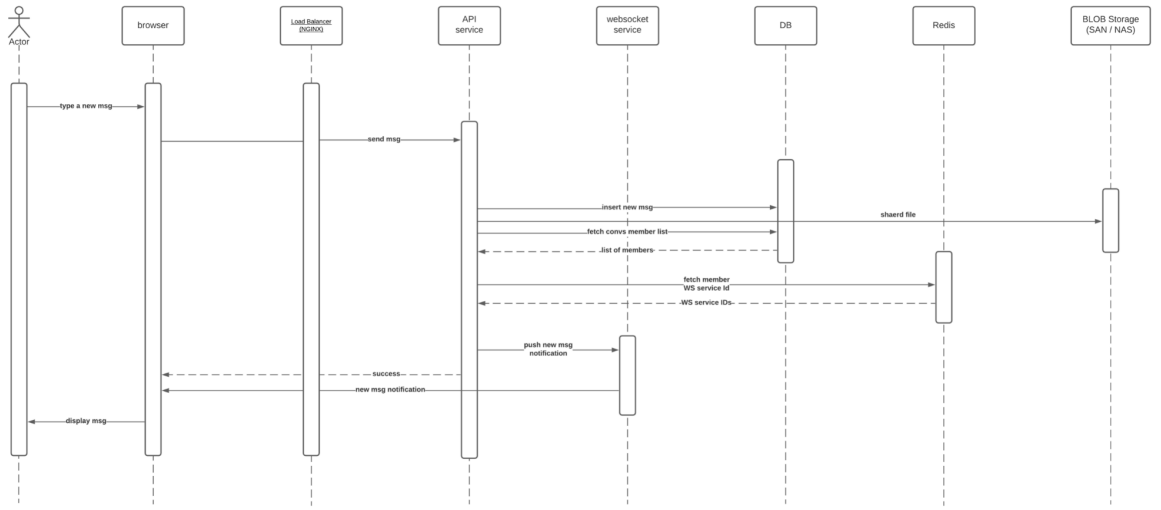
- مخطط حالة الاستخدام Use case diagram



الشكل 3-16: يوضح مخطط حالة الاستخدام لإرسال رسالة جديدة

يقوم مستثمر التطبيق بزيارة المحادثة التي يود إرسال رسالة ضمنها إلى المشتركين فيها، وبعد ذلك يقوم إما بكتابة رسالته و/أو اختيار الملف الذي يود إرساله، وعند ذلك يقوم النظام بحفظ الرسالة النصية ضمن المحادثة، وحفظ الملف المشارك ضمن BLOB Storage.

- مخطط التسلسل Sequence diagram

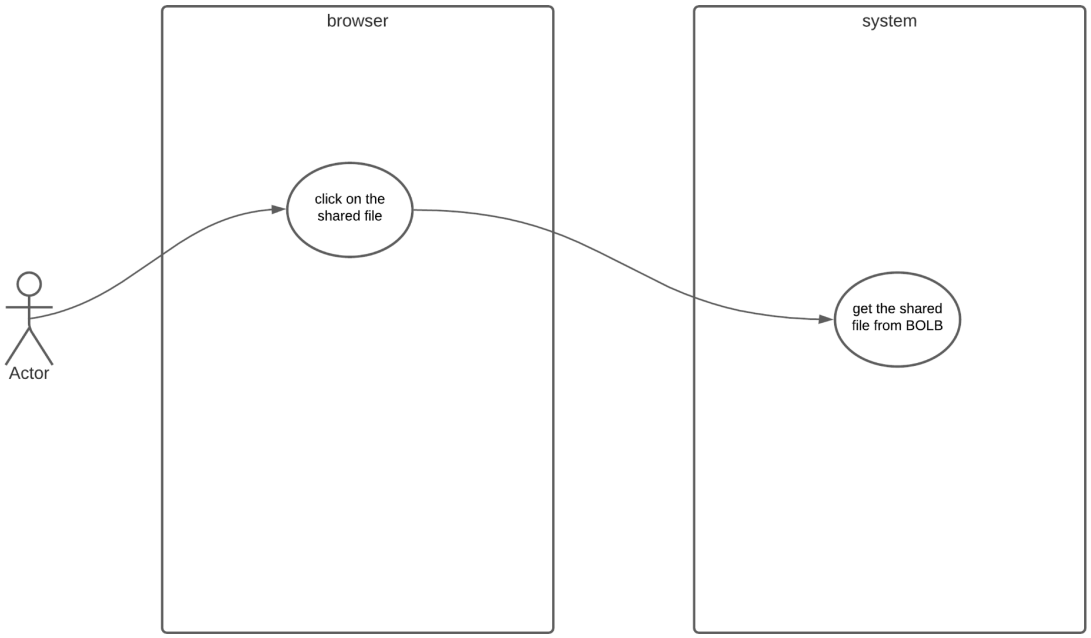


الشكل 3-17: يوضح مخطط التسلسل لإرسال رسالة جديدة

- يقوم مستثمر التطبيق بكتابة الرسالة الجديدة و/أو اختيار الملف المراد المشاركة به، ثم يقوم المتصفح بإرسال محتوى النص و/أو الملف المختار إلى خدمة API.
- تقوم خدمة API بالتواصل مع قاعدة المعطيات DB من أجل إضافة الرسالة الجديدة ضمن المحادثة المرغوبة من جهة، ومن أجل الحصول على عناوين البريد الإلكتروني الخاص بمشتركي المحادثة من جهة أخرى.
- تقوم خدمة API بالحصول على المعرفات IDs الخاصة بالمشاركين بالمحادثة ضمن خدمة WebSocket وذلك من خلال التواصل مع خدمة Redis.
- ثم تقوم API بالطلب من خدمة WebSocket إرسال إشعار بالرسالة الجديدة إلى المتصفح الخاص بمشتركي المحادثة.
- أخيراً يقوم المتصفح بعرض الرسالة الجديدة.

تحميل ملف Downloading a File:

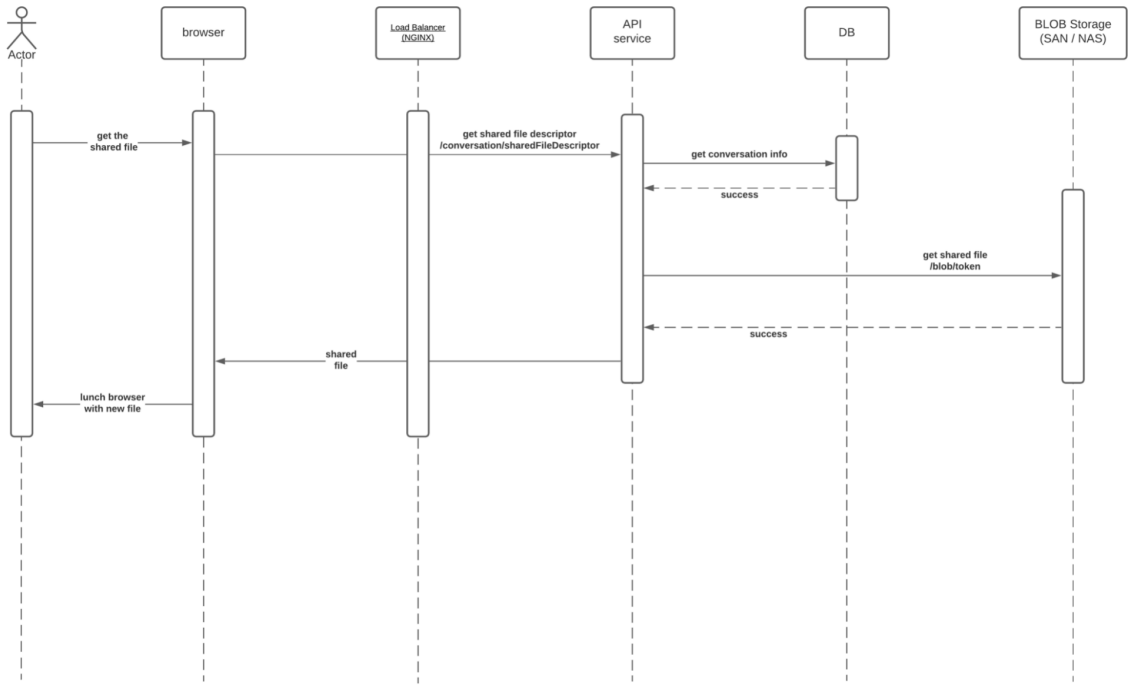
- مخطط حالة الاستخدام Use case diagram



الشكل 3-18: يوضح مخطط حالة الاستخدام لتحميل ملف

عندما يقوم مستثمر التطبيق بالنقر على الأيقونة الخاصة بالملف الذي أرسل إليه، يتم عرض هذا الملف ضمن نافذة جديدة صغيرة، بحيث يمكن للمستثمر أن يعرض الملف أو أن يحمله على جهازه.

- مخطط التسلسل Sequence diagram

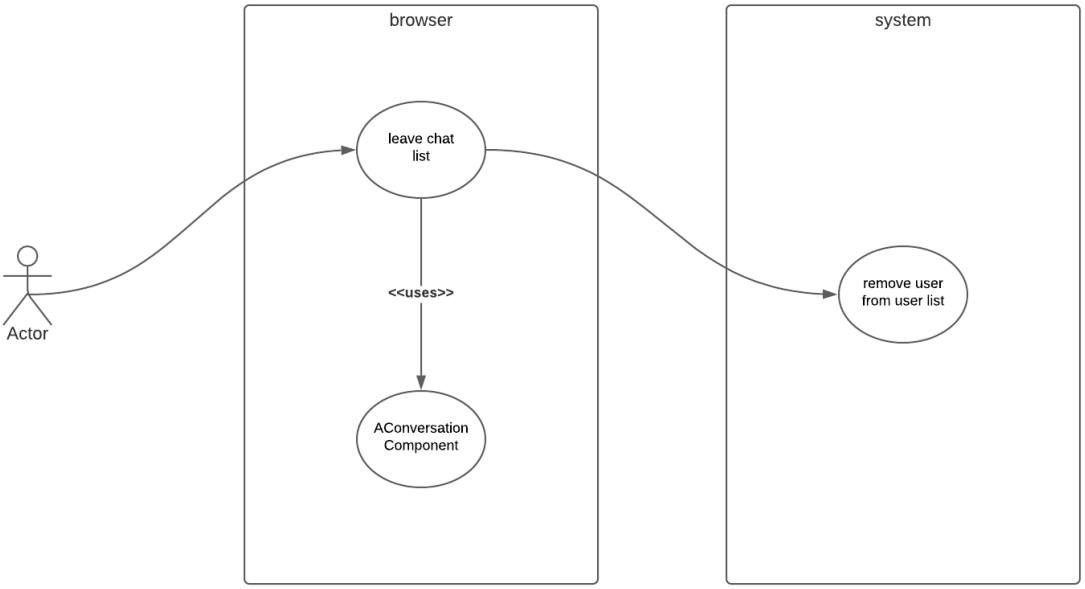


الشكل 3-19: يوضح مخطط التسلسل الخاص بتحميل ملف

- عند قيام مستثمر التطبيق بالنقر على أيقونة الملف المشارك معه، يتم طلب هذا الملف من قبل المتصفح، الذي يستعين بخدمة REST API من أجل تحميل على الملف المطلوب.
- تقوم REST API بالحصول على موصف الملف المشارك File Descriptor من قبل قاعدة المعطيات DB.
- بعد نجاح الخطوة السابقة تقوم REST API بطلب الملف المشارك من مخزن الملفات كبيرة الحجم BLOB.
- بعد ذلك يتم إرسال الملف المشارك إلى متصفح مستثمر التطبيق، الذي يقوم بدوره بتحميل هذا الملف ضمن نافذة جديدة صغيرة على المتصفح.

مغادرة المحادثة Leave Conversation List:

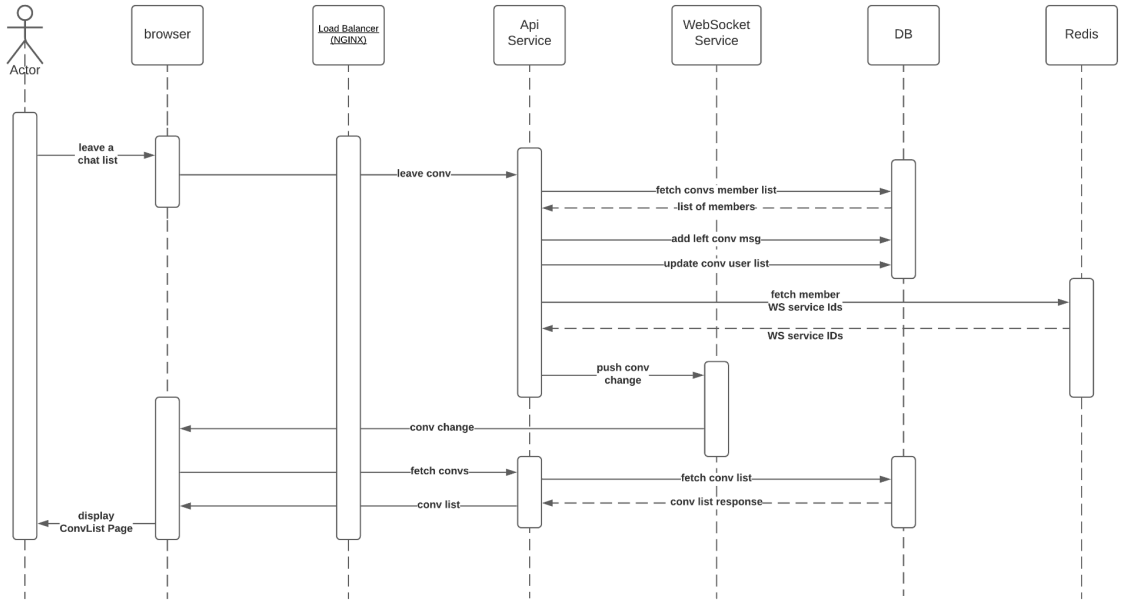
- مخطط حالة الاستخدام Use case diagram



الشكل 3-20: يوضح مخطط حالة الاستخدام لمغادرة المحادثة

يقوم مستثمر التطبيق بالضغط على الزر leave من أجل مغادرة المحادثة، وذلك بالاستعانة بالمكون AConversation ضمن تطبيق React الخاص بنا، وأخيراً يقوم النظام بإزالة هوية المستثمر ID من المحادثة المستهدفة.

- مخطط التسلسل Sequence diagram

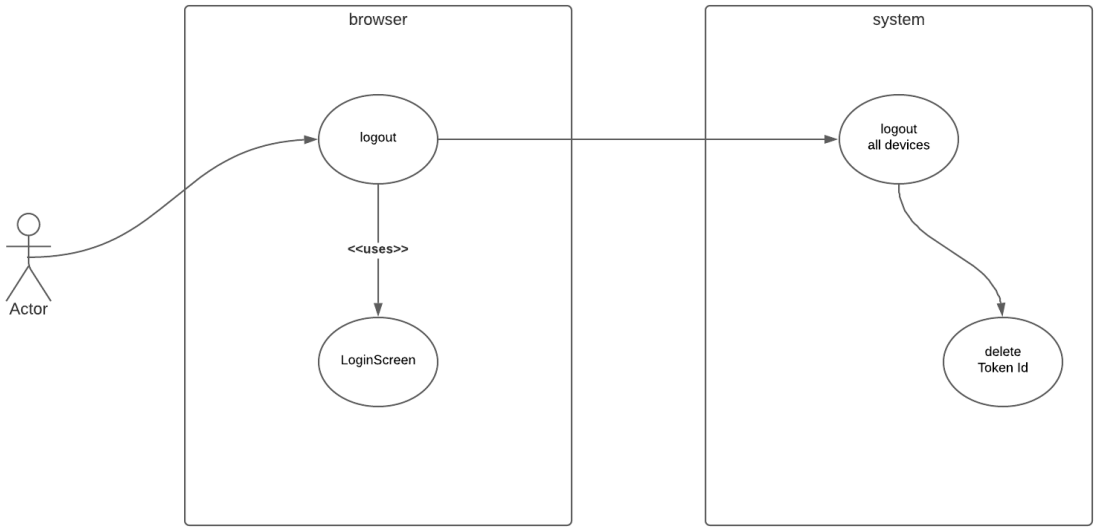


الشكل 3-21: يوضح مخطط التسلسل لمغادرة المحادثة

- يقوم مستثمر التطبيق بالنقر على زر مغادرة المحادثة leave الشيء الذي يجعل المتصفح يرسل هذا الأمر إلى خدمة API.
- تتصل خدمة API مع قاعدة المعطيات DB من أجل تحديث مشتركي المحادثة بعد مغادرة مستثمر التطبيق لها.
- تتصل خدمة API مع خدمة Redis من أجل الحصول على معرفات الـ WebSocket العائدة لمشتركي المحادثة، وبعد ذلك تقوم بإرسال التحديث إلى خدمة WebSocket التي تقوم بدورها بإرسال التحديث إلى المتصفح.
- يقوم المتصفح بعد ذلك بجلب لائحة المحادثات العائدة لمستثمر التطبيق من قاعدة المعطيات بالاعتماد على خدمة API من جديد.
- وأخيراً يعرض المتصفح لائحة المحادثات المتبقية والعائدة لمستثمر التطبيق نفسه.

الخروج من التطبيق على كافة الأجهزة Logout:

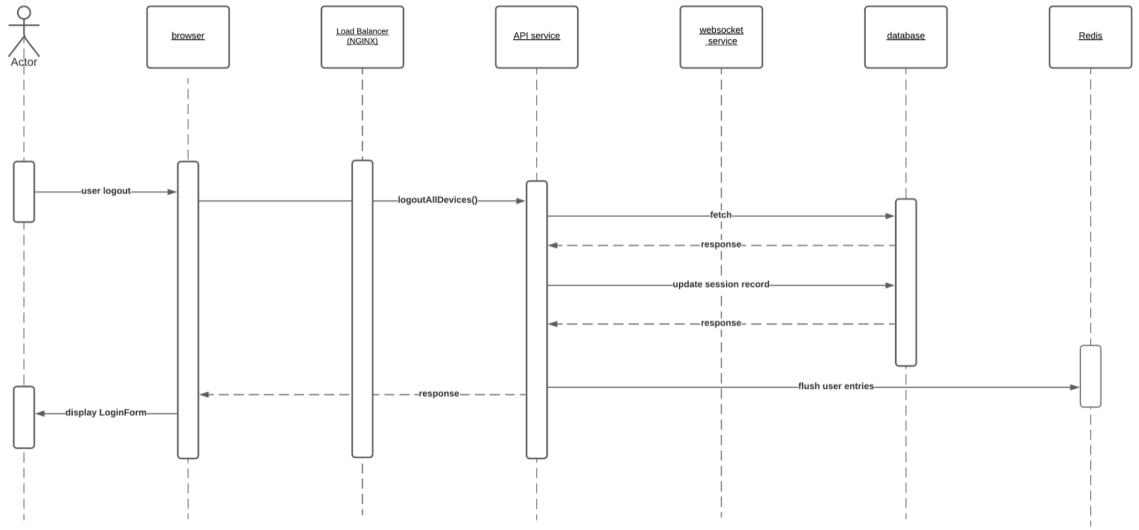
- مخطط حالة الاستخدام Use case diagram



الشكل 3-22: يوضح مخطط حالة الاستخدام للخروج من التطبيق

يقوم مستثمر التطبيق بالضغط على الزر logout من أجل تسجيل الخروج من التطبيق، وعند ذلك يقوم النظام بحذف معرف الـ token الخاص بالمستثمر، وتقوم React بالإنتقال إلى الواجهة LoginScreen.

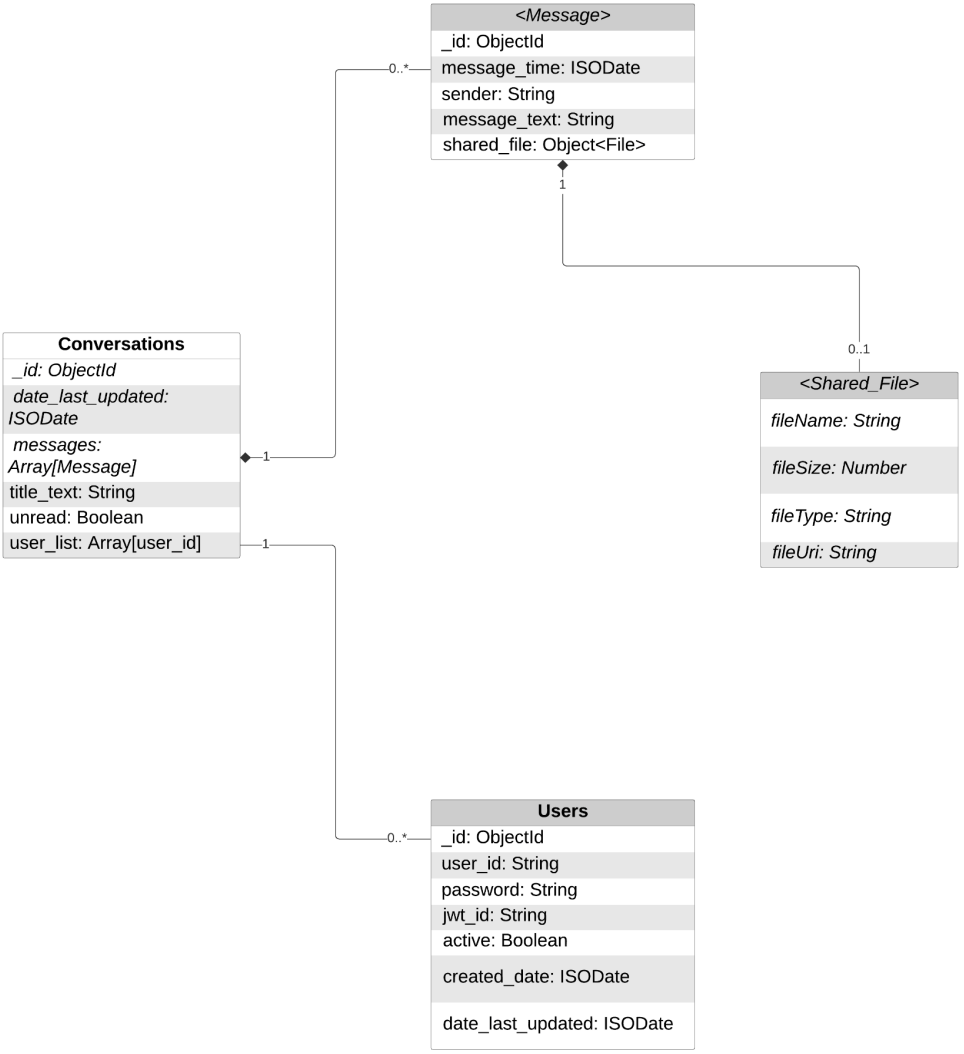
- مخطط التسلسل Sequence diagram



الشكل 3-23: يوضح مخطط التسلسل للخروج من التطبيق

- عند قيام المستخدم بالضغط على زر logout يقوم المتصفح بالاتصال مع خدمة API من أجل ذلك.
- تقوم خدمة API بالتواصل مع قاعدة المعطيات من أجل تحديث الجلسة session ومن ثم تقوم API بالاتصال مع خدمة redis من أجل تحديث العناوين الخاصة بخدمة WebSocket.
- أخيراً تقوم API بإرسال النتيجة إلى المتصفح الذي يقوم بدوره وبواسطة React بعرض المكون LoginForm.

مخطط قاعدة البيانات:



الشكل 3-24: يوضح مخطط قاعدة البيانات

الفصل الرابع: التنفيذ والنتائج

4-1 مقدمة:

يهدف هذا الفصل إلى التعريف بأهم التقانات المستخدمة، وإعدادات بيئة العمل، مع توثيق لبعض المهام التي يقدمها التطبيق وأهم البرمجيات المستخدمة، مع عرض حول الإجراءات الأمنية المنفذة ضمنه.

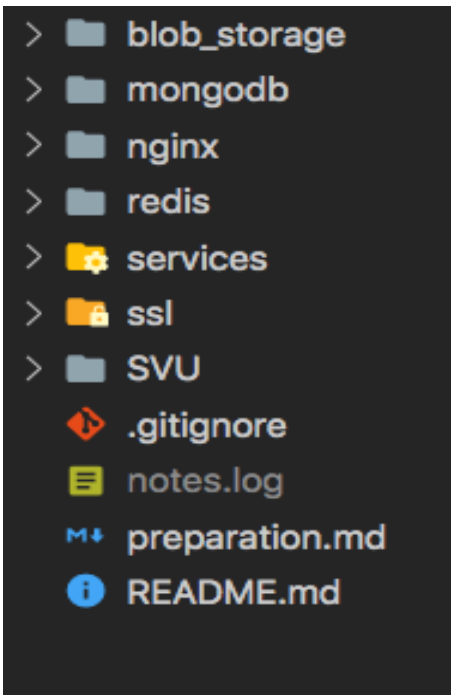
4-2 التقانات المستخدمة:

تم الاعتماد على النموذج MVVM الذي يقترح فصل منطق عرض البيانات (واجهة المستخدم) عن منطق الأعمال الأساسي في التطبيق، وقد تم عرض أهم التقانات المستخدمة في التطبيق ضمن الفصل الثالث، ونذكر هنا أهم هذه التقانات:

- فيما يخص الواجهة الأمامية فقد استخدمنا كل من (React , Expo).
- فيما يخص الواجهة الخلفية فقد استخدمنا كل من (NodeJS , Express , JWT , Passport , Passport-JWT , WebSocket, NGINX).
- أما ما يخص قاعدة المعطيات فقد استخدمنا MongoDB وRedis.

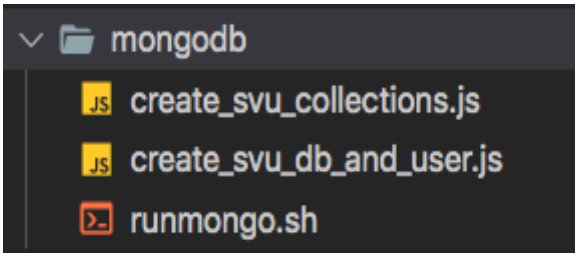
4-3 إعداد بيئة العمل:

نعرض من خلال الشكل التالي أهم المجلدات المستخدمة في التطبيق، في كلا الطرفين الأمامي والخلفي backend & frontend بالإضافة إلى قاعدة البيانات، ومخدم NGINX



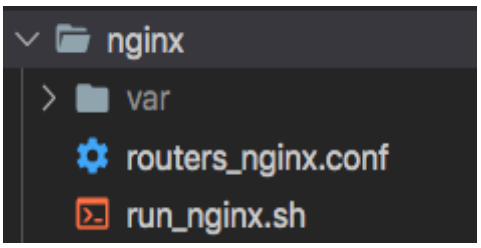
الشكل 4-1: يوضح بنية التطبيق الشاملة

- المجلد blob_storage يحتوي على الملفات التي يتبادلها مستثمري التطبيق خلال محادثاتهم، مع العرض بأن عملية تسمية هذه الملفات تتم بشكل آلي ووفق قواعد محددة لذلك.
- المجلد mongodb يحتوي على الملفات الأساسية لقاعدة البيانات الخاصة بإنشاء المجموعات create_svu_collections.js و create_svu_db_and_user.js و runmongo.sh



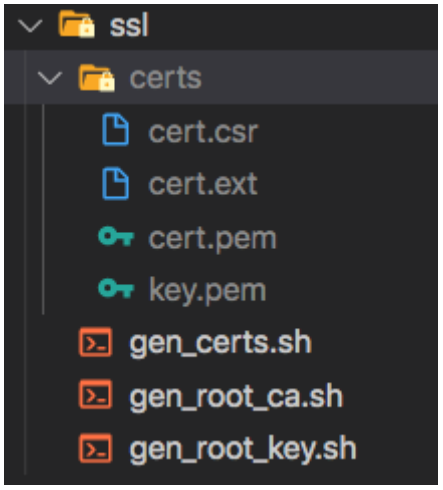
الشكل 4-2: يوضح الملفات الأساسية في mongodb

- المجلد nginx يحتوي على أهم الملفات الخاصة بمخدم NGINX وأهمها routers_nginx.conf الخاص بتهيئة هذا المخدم، والملف run_nginx.sh الخاص بإقلاع المخدم.



الشكل 4-3: يوضح الملفات الأساسية في nginx

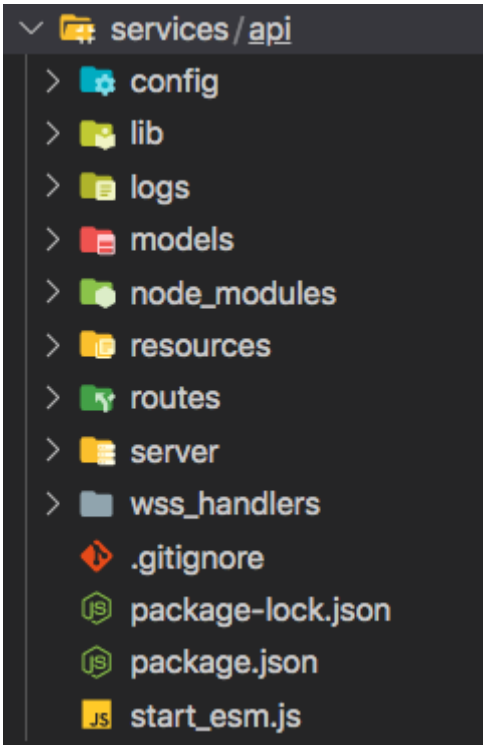
- المجلد redis الخاص بقاعدة البيانات Redis.
- المجلد ssl يحتوي على الملفات الخاصة بالتشفير المعياري SSL.



الشكل 4-4: يوضح الملفات الأساسية في ssl

- المجلد services وهو المجلد الخاص بالواجهة الخلفية، وسوف نتكلم عنه بالتفصيل لاحقاً.
- المجلد SVU وهو المجلد الخاص بالواجهة الأمامية، وسوف نتكلم عنه بالتفصيل لاحقاً.

4-3-1 أهم المجلدات والملفات في الواجهة الخلفية Backend



الشكل 4-5: يوضح الملفات الأساسية في الواجهة الخلفية

يوضح الشكل 4-5 أهم المجلدات والملفات التي تم استخدامها ضمن الواجهة الخلفية للتطبيق:

- config الذي يحتوي على ملف التهيئة الرئيسي للتطبيق config.json الذي يتضمن كافة المعلومات الخاصة بإعدادات البرنامج، حيث يوضح الشكل 4-6 هذه المعلومات:

```
{
  "port": 18000,
  "ws_port": 18001,
  "app_root": "/svu/api",
  "app_title": "SVU Application",
  "app_version": "1.0.0",
  "app_url_domain": "localhost",
  "external_server_url": "https://localhost",
  "pid_file_path": "/var/run/svu_app.pid",
  "login_url": "https://localhost/",
```

```
   "wss_server": {
       "ssl": {
           "cert_file": "../../ssl/certs/cert.pem",
           "key_file": "../../ssl/certs/key.pem"
       },
       "wss_port": 18001,
       "protocols": ["svu-protocol"],
       "wss_handlers_root": "wss_handlers",
       "allow_origin": ["https://localhost",
"https://localhost:18001"]
   },
   "app": {
       "ssl": {
           "cert_file": "../../ssl/certs/cert.pem",
           "key_file": "../../ssl/certs/key.pem"
       },
       "security": {
           "jwt_header_name": "",
           "session_duration": 3600,
           "allow_origin": ["https://localhost"]
       },
       "third_party_oauth": {},
       "blob_storage": {
           "path": "../../blob_storage"
       },
       "database": {
           "host_seed": "localhost:27017",
           "login_id": "svu_db_user",
           "login_password": "svu_db_pwd",
           "connect_retry_millies": 5000,
           "database": "svudb"
       }
   }
}
```

الشكل 4-6: يوضح محتوى الملف config.json

- lib/utils يحتوي هذا المجلد على ملفات الخدمات الرئيسية وهي:
    - blobHelper.js
    - cryptoHelper.js
    - imageHelper.js
    - ws_backend_client.js

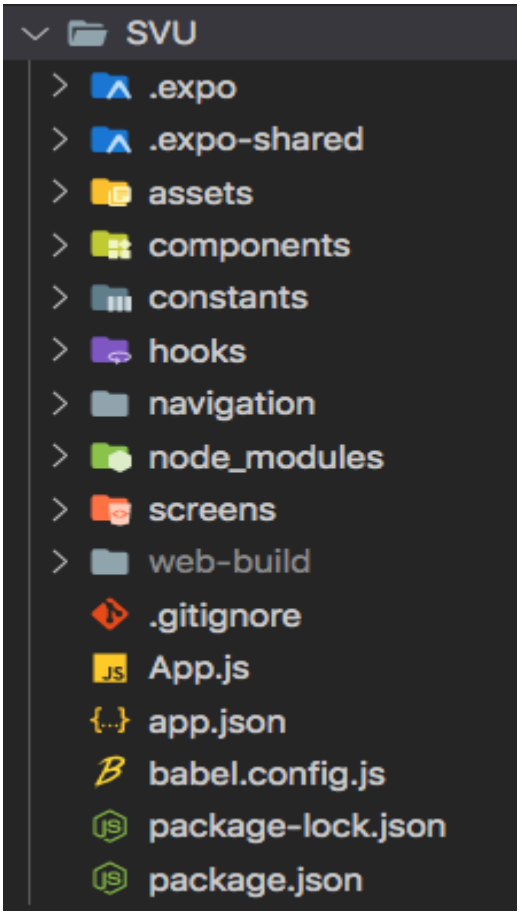
- Logs يحتوي على الملف الخاص بحفظ كامل السجل النصي الخاص بالتطبيق.
- Models يحتوي هذا المجلد على ملفات المجموعات collections الموجودة ضمن قاعدة البيانات الخاصة بالتطبيق، وهي:
    - conversation.js
    - file.js
    - session.js
    - user.js

- node_modules يحتوي هذا المجلد على مكتبة ملفات Node الرئيسية الخاصة بتطبيقنا والتي تتعلق بملف package.json الموجود ضمن الواجهة الخلفية للتطبيق.
- resources/nar_images يحتوي على ملفات الصور التي نستخدمها كـ NAR.
- routes يعتبر هذا المجلد من أهم المجلدات في الواجهة الخلفية، حيث يستخدم من أجل تحديد مسارات REST التي استخدمت من أجل خدمات API وهي:
    - blob.js
    - conversation.js
    - session.js
    - user.js

وسوف نتطرق بالتفصيل لأهم هذه المسارات في فصل توثيق بعض المهام التي يقدمها التطبيق.

- server وهو المجلد الرئيسي في الواجهة الخلفية، حيث يحتوي على كل ما يتعلق بكل من NodeJS و Express مع كل ما يتعلق بهما بكل من الموجه وقاعدة المعطيات والويب سوكيت، حيث يحوي هذا المجلد على كل من الملفات التالية:
    - server.js
    - router.js
    - ws_server.js
    - database.js
    - config.js
    - app_logger.js

4-3-2 أهم المجلدات والملفات في الواجهة الأمامية Frontend



الشكل 4-7: يوضح الملفات الأساسية في الواجهة الأمامية

يوضح الشكل 4-7 أهم المجلدات والملفات التي تم استخدامها ضمن الواجهة الأمامية للتطبيق:

- expo يحتوي هذا المجلد على كافة الملفات المتعلقة بـ React Expo التي تؤمن لنا كتابة البرنامج لمرة واحدة، والعمل على مختلف المنصات.
- assets يحتوي على ملفات الصور والصوت والخطوط المستخدمة ضمن تطبيقنا.
- components يعتبر هذا المجلد من الأهم ضمن تطبيقات React لأنه يحتوي على المكونات الرئيسية Components التي نستخدمها ضمن برنامج الواجهة الأمامية في التطبيق.
- hooks يحتوي هذا المجلد على ملفين أساسيين تم إنشاؤها من نوع Custom Hook وأهمها على الإطلاق هو الملف useSVUSessionContext.js الذي يعتبر الملف الأساسي في عملية التواصل بين كلا الواجهتين الأمامية والخلفية للتطبيق.

- node_modules يحتوي على كافة الملفات الخاصة بـ NodeJS والخاصة بتطبيقنا وفق الملف package.json

- screens يحتوي على ملفات الواجهات الأساسية للتطبيق وأهمها HomeScreen.js و SessionScreen.js
- App.js يعتبر الملف الرئيسي في مكتبة React والذي يتم من خلاله تسليم الصفحة الأساسية للتطبيقات من طراز SPA

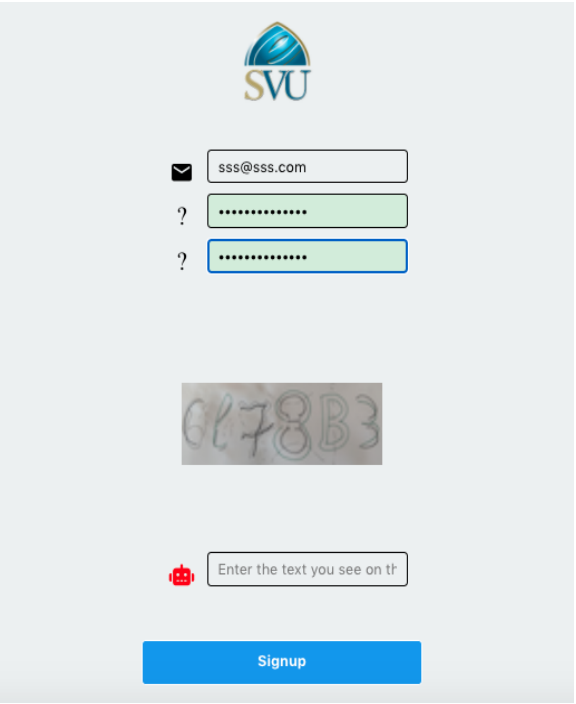
4-4 توثيق بعض المهام التي يقدمها النظام وأهم البرمجيات المستخدمة:

سنعرض هنا أهم المهام مع البرمجيات المستخدمة في كلا الواجهتين الأمامية والخلفية للنظام، معتمدين في نظامنا على نموذج MERN Stack الذي يتضمن كل من :

- MongoDB
- Express
- React
- NodeJS

4-4-1 الاشتراك للحصول على حساب ضمن التطبيق Signup:

بدايةً يقوم مستثمر التطبيق بإدخال البريد الإلكتروني الخاص به مع كلمة السر والتأكيد على كلمة السر وبعد التأكد من صحة الإدخال تظهر صورة NAR التي تحوي نصاً ما من أجل أن يقوم المستثمر بإدخاله، والشكل التالي يبين النافذة التي تظهر للمستثمر عند الدخول على صفحة الموقع:



الشكل 4-8: يوضح النافذة الخاصة بالاشتراك Signup

وبعد ذلك وبعد التأكد من صحة البيانات المدخلة، يتم إرسالها إلى الواجهة الخلفية بصيغة JSON حيث يقوم التابع doSignup() بالمعالجة المطلوبة وفق الشكل التالي:

```
const doSignup = async () => {
  let payload = {
    "userId": userId,
    "password": password,
    "narToken": narToken,
    "narText": narText
  };

  let response = await apiCall(`/session/signup`, payload);

  if (response.success) {
    setSignupInProgress(false);
  }
  console.log(response);
}
```

الشكل 4-9: يوضح معالجة البيانات المدخلة للاشتراك بالتطبيق

وعند الواجهة الخلفية يقوم Express من خلال الموجه الخاص به Router وبعد تحديد المسار المطلوب بمعالجة العملية الخاصة بـ signup من أجل الحصول على حساب جديد، وذلك كما هو مبين وفق الكود المرفق والموجود ضمن الملف session.js:

```
router.post("/signup", async (req, res) => {
  let loginUrl = getConfig().get("login_url");
  let userId = req.body["userId"];
  let password = req.body["password"];
  let narToken = req.body["narToken"];
  let narText = req.body["narText"];

  if (!CryptoHelper.validateEmail(userId)) {
      return res.status(400).send("Invalid email sent as userId.");
  }
```

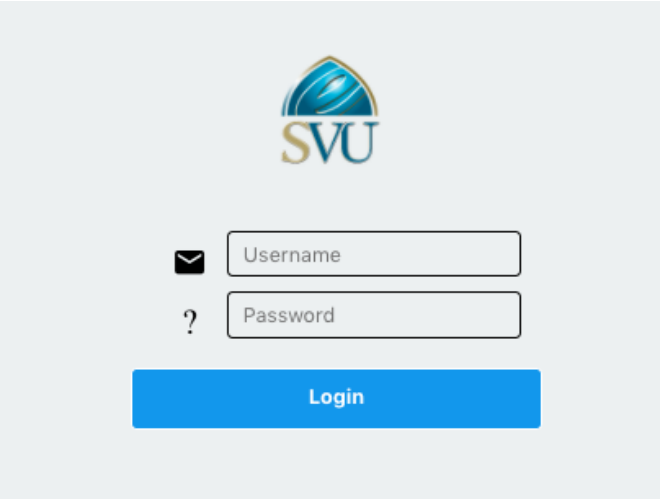
```
  if (!password || (password.length < 12)) {
      return res.status(400).send("password length must be >= 12 characters");
  }

  let narClear = CryptoHelper.decryptNaRToken(narToken);
  if (userId.concat(":", narText) != narClear) {
      return res.status(403).send("Robots are forbidden");
  }

  let responsePayload = {
      success: true,
      message: `Signup successful, should be redirected to login.`,
  }

  let existingAccount = await findOne("users", { user_id: userId });
  if (existingAccount) {
      return res.json(responsePayload);
  }

  // at this point, everything is ok to create the account and activate it:
  let newAcount = {
      user_id: userId,
      password: CryptoHelper.genPasswordHash(password),
      jwt_id: "",
      active: true
  }
  await insertOne("users", newAcount);

  return res.json(responsePayload);
})
```

الشكل 4-10: يوضح معالجة البيانات المدخلة للاشتراك بالتطبيق من جهة الواجهة الخلفية

وفي النهاية تقوم React بالانتقال إلى النافذة الخاصة بتسجيل الدخول Login.

4-4-2 تسجيل الدخول إلى الحساب Login:

بعد الانتهاء من عملية الاشتراك من قبل مستثمر التطبيق، يتم إدخال البريد الإلكتروني وكلمة المرور ضمن المكون الخاص بتسجيل الدخول، كما هو موضح بالشكل التالي:



الشكل 4-11: يوضح نافذة تسجيل الدخول Login

وبعد ذلك وبعد التأكد من صحة البيانات المدخلة، يتم إرسالها إلى الواجهة الخلفية بصيغة JSON حيث يقوم التابع doLogin() بالمعالجة المطلوبة وفق الشكل التالي:

```
const doLogin = async (userId, password) => {
  if (!(userId && password)) {
    return;
  }

  let payload = {
    userId: userId,
    password: password,
  }

  try {
    let loginResponse = await apiCall("/session/login", payload);
```

```
    let sessionUpdateAction = {
      type: "API_CALL",
      newState: {
        userId: userId,
        expireTimeMillis: loginResponse.expireTimeMillis,
      }
    }

    dispatchSessionUpdate(sessionUpdateAction);

    playSound();

  } catch (error) {
    console.log(`loginResponse Error: ${error}`);
  }
};
```

الشكل 4-12: يوضح معالجة البيانات المدخلة الخاصة بتسجيل الدخول Login

وعند الواجهة الخلفية يقوم Express من خلال الموجه الخاص به Router وبعد تحديد المسار المطلوب بمعالجة العملية الخاصة بـ login من أجل الدخول إلى التطبيق، وذلك كما هو مبين وفق الكود المرفق والموجود ضمن الملف session.js:

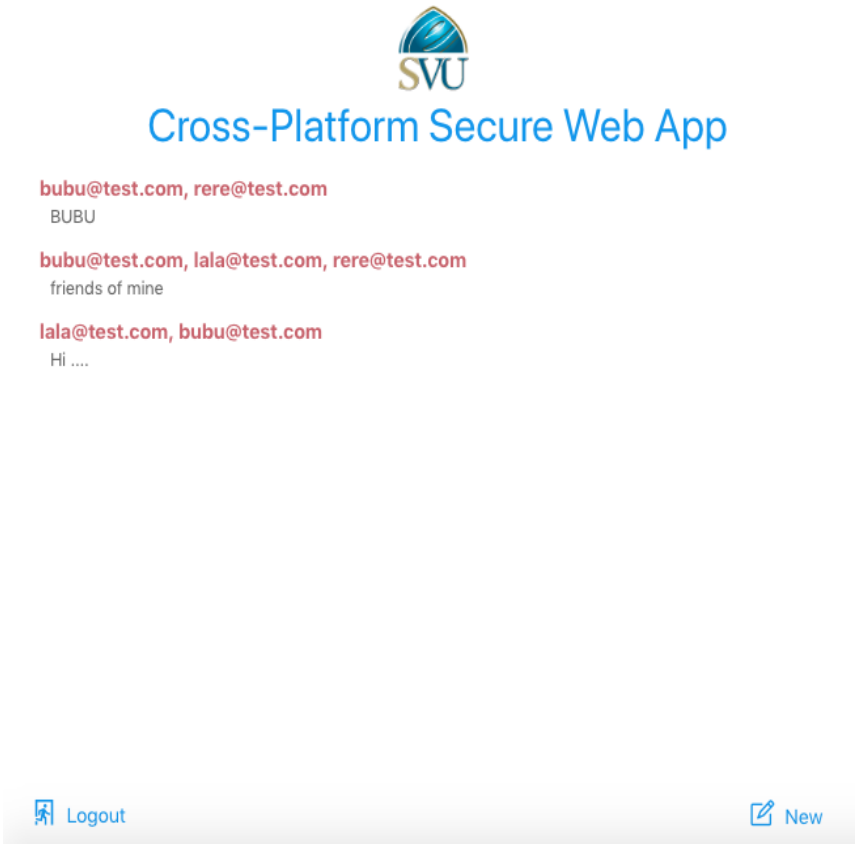
```
router.post("/login", async (req, res) => {
  // let user = new User(req.body.email, req.body.password);
  //  1- lookup user and hashed password from mongo db, if user does not exist, return 401
  //  2- set Bearer JWT header if successful, and return success

  let userId = req.body["userId"];
  let password = req.body["password"];

  if (!CryptoHelper.validateEmail(userId)) {
     return res.status(400).send("Invalid user ID");
  }

  let existingAccount = await findOne("users", { user_id: userId });
  if (!existingAccount || !existingAccount.active) {
     return res.status(401).send("Unauthorized");
  }

  if (!CryptoHelper.validatePassword(password, existingAccount.password)) {
```

```
    return res.status(401).send("Unauthorized");
  }

  let jwtId = existingAccount.jwt_id;
  if (!jwtId) {
    jwtId = CryptoHelper.genRandomString(16);
    await updateOne("users", { user_id: userId }, { $set: { jwt_id: jwtId } });
  }

  let expireTimeMillis = (new Date()).getTime() + 3600000;
  let responsePayload = {
    scope: "SVU",
    token_type: "JWT",
    expireTimeMillis: expireTimeMillis,
  };

  const secretKey = fs.readFileSync(path.resolve("./config", appConfig.get("app:ssl:key_file")));
  const domainUrl = new URL(appConfig.get("external_server_url"));
  const opts = {
    expiresIn: 3600,
    audience: domainUrl.hostname,
    issuer: domainUrl.hostname,
    subject: userId,
    jwtid: jwtId,
  };

  const token = sign(responsePayload, secretKey, opts);

  res.set("authorization", "Bearer " + token);
  res.json(responsePayload);
});
```

الشكل 4-13: يوضح معالجة البيانات الخاصة بتسجيل الدخول إلى التطبيق من جهة الواجهة الخلفية

4-4-3 إنشاء محادثة جديدة Create New Chat List:

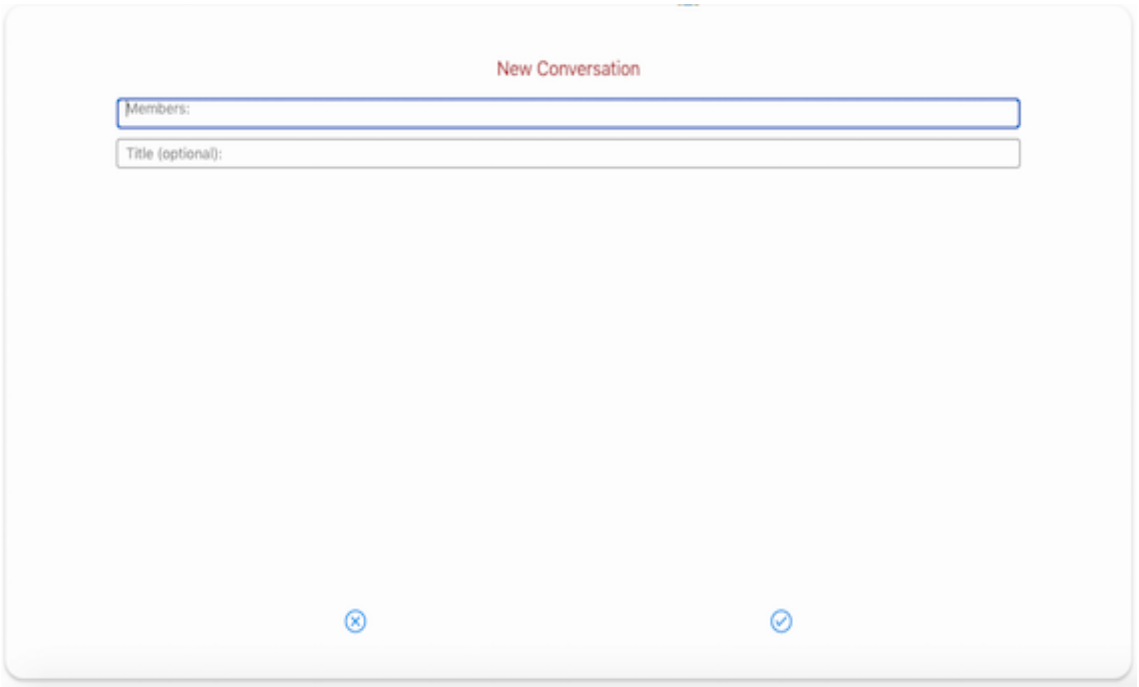
بعد النجاح بالدخول إلى التطبيق، تظهر النافذة الرئيسية فيه والتي تتضمن لائحة بالمحادثات القديمة إضافةً إلى وجود زرين أحدهما للخروج من التطبيق، والثاني لإنشاء محادثة جديدة، كما هو موضح بالشكل التالي:



الشكل 4-14: يوضح النافذة الرئيسية في التطبيق

وعند النقر على الزر New تظهر النافذة الخاصة بإنشاء المحادثة الجديدة، التي تتضمن نموذج form يحتوي على عنصري إدخال input الأول خاص بإدخال عناوين البريد الإلكتروني للأشخاص المراد المحادثة معهم، والثاني خاص بعنوان المحادثة، وبعد إدخال عناوين البريد الإلكتروني للأشخاص المراد بدء المحادثة معهم، وإدخال عنوان المحادثة يتم النقر على الزر ذي الرمز صح ليصار إلى بدء المحادثة الجديدة المرغوبة، أو يتم النقر على الزر ذي الرمز ضرب ليصار إلى إلغاء العملية.

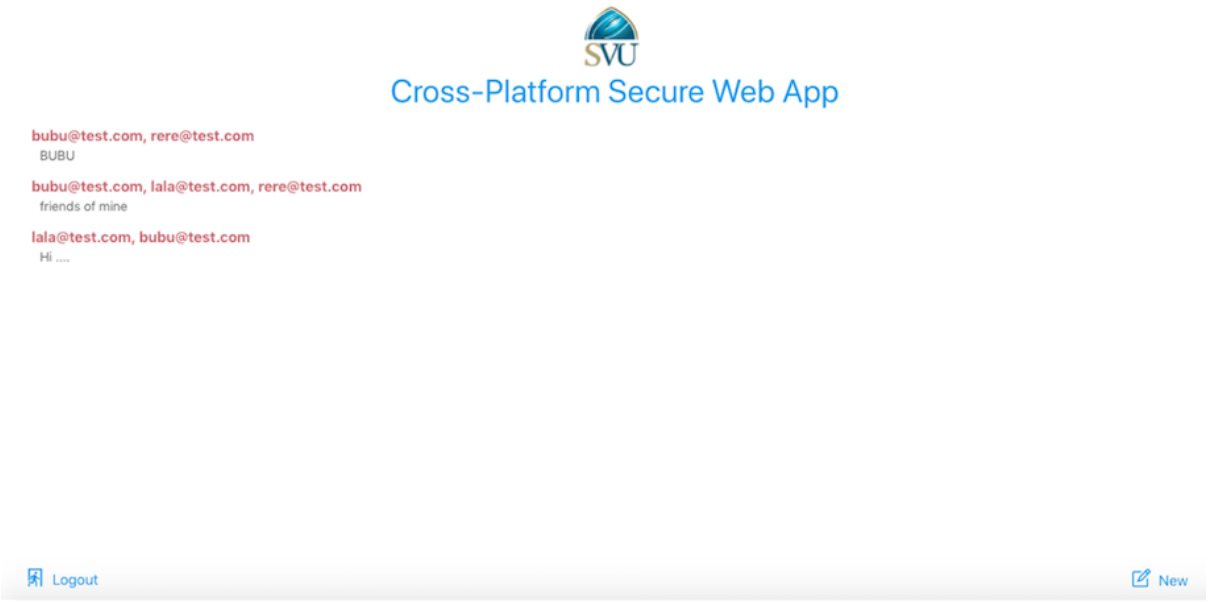
وفي النهاية وعند البدء بمحادثة جديدة، تقوم React بالانتقال إلى الواجهة الرئيسية الخاصة بالمحادثة والتي من خلالها يمكن للمشتركين بهذه المحادثة تبادل الرسائل النصية وتبادل مختلف أنواع الملفات، على ألا يتجاوز حجم الملف قيمة 10 ميغابايت.
حيث يبين الشكل التالي صورة النافذة الخاصة بإنشاء محادثة جديدة:



الشكل 4-15: يوضح النافذة الخاصة بإنشاء محادثة جديدة

4-4-4 إرسال رسالة جديدة Send New Message:

بعد إنشاء المحادثة الجديدة، تنتقل React إلى الواجهة الخاصة بمحادثات المشترك التي تتضمن لائحة بالمحادثات القديمة إضافةً لآخر محادثة تم إنشاؤها، كما هو موضح بالشكل التالي:



الشكل 4-16: يوضح النافذة الخاصة بلائحة المحادثات

وعند النقر على أي من هذه المحادثات يتم الانتقال مباشرةً إلى النافذة الخاصة بالمحادثة المعنية، والتي من خلالها يمكن للمستثمر تبادل الرسائل النصية والملفات مع المشتركين معه ضمن هذه المحادثة، حيث يبن الشكل التالي ذلك:

الشكل 4-17: يوضح النافذة الخاصة بإحدى المحادثات

من خلال الشكل السابق نبين بأن النصوص المكتوبة والملفات المشاركة في الجهة اليمنى تخص المرسل، أما في الجهة اليسرى فيوجد فيها اسم الشخص المستقبل مع الرسائل النصية والملفات التي شارك بها ضمن هذه المحادثة، مع الملاحظة بإمكانية وجود أكثر من مستقبل.
أخيراً نبين من خلال الشكل 4-18 الدالة الخاصة بإرسال الرسالة ضمن الواجهة الأمامية للتطبيق:

```
const sendMessage = async () => {
   let payload = {
     "conversationId": activeConversationId,
     "message": {
       "messageText": newMessageText,
       "sharedFile": null,
     }
   };
   if (!newMessageText.trim() | newMessageText.trim().length <= 0) {
     setNewMessageText("");
     return;
   }
   if ((doc != null) && (newMessageText.indexOf(`\uD83D\uDCCE ${ doc.name }`) >=
0)) {
     // console.log("Doc: " + JSON.stringify(doc.uri, 4, null));
     // console.log("Doc: " + doc.uri);
     payload.message.sharedFile = {};
     payload.message.sharedFile.fileName = doc.name;
     payload.message.sharedFile.fileSize = doc.size;
     payload.message.sharedFile.fileType = doc.type;
     payload.message.sharedFile.fileUri = JSON.stringify(doc.uri);
   }

   try {
     let apiResponse = await apiCall("/conversation/newMessage", payload);
     setNewMessageText("");
     setDoc(null);
   } catch (error) {
     console.log(`apiCall Error: ${ error }`);
   }
 };
```

الشكل 4-18: يوضح إجرائية إرسال رسالة ضمن الواجهة الأمامية

وتم في الواجهة الخلفية للتطبيق معالجة إجرائية إرسال رسالة جديدة (نص و/أو ملف) من خلال استخدام REST API مع المسار 'newMessage/' كما هو موضح بالشكل التالي:

```
router.post("/newMessage", async (req, res) => {
 const id = uuidv4();
 const messageTime = new Date();

 let conversationId = req.body["conversationId"];
 let message = req.body["message"];
 // console.log(`\n\nreq.body: ${JSON.stringify(req.body, 4, null)}\n\n`)

 let existingConv = await findOne("conversations",
   {
     _id: conversationId,
     user_list: req.user.userId,  // this is very important for security, only user
conversations are are allowed to include the new message.
   },
   { projection: { _id: 0, title_text: 1, user_list: 1 } });

 if (!existingConv | isEmpty(existingConv)) {
   return res.json({});
 }

 if (message.sharedFile) {
   let tempFileURI = message.sharedFile.fileUri;
   let blobFileUri = await BlobHelper.writeFile(tempFileURI);

    message.sharedFile.fileUri = blobFileUri;
 }

 let insOp = await updateOne("conversations",
   { _id: conversationId },
   {
     $push: {
       messages: {
         _id: id,
         message_time: messageTime,
         sender: req.user.userId,
         message_text: message.messageText,
         shared_file: message.sharedFile,
```

```
      }
    }
  })
if (insOp.result.nModified == 1) {
  const conversation = {
    conversationId: existingConv._id,
    userList: existingConv.user_list,
    titleText: existingConv.title_text,
  };

  const wsMessage = {
    endPoint: "conversation",
    conversation: conversation,
  }
  try {
    await sendMessage("wss://localhost:18001", wsMessage);
  } catch (err) {
    console.log(err);
  }

  return res.json({});
 } else {
  return res.status(400).json({});
 }
});
```

الشكل 4-19: يوضح التعامل مع إجرائية إرسال رسالة جديدة في الواجهة الخلفية

4-4-5 تحميل ملف ضمن المحادثة Download Shared File:

يبين الشكل التالي التابع الخاص بتحميل الملف المشارك به ضمن المحادثة على الحاسب الشخصي للمستثمر، والوارد من قبل أحد المشاركين ضمن نفس المحادثة:

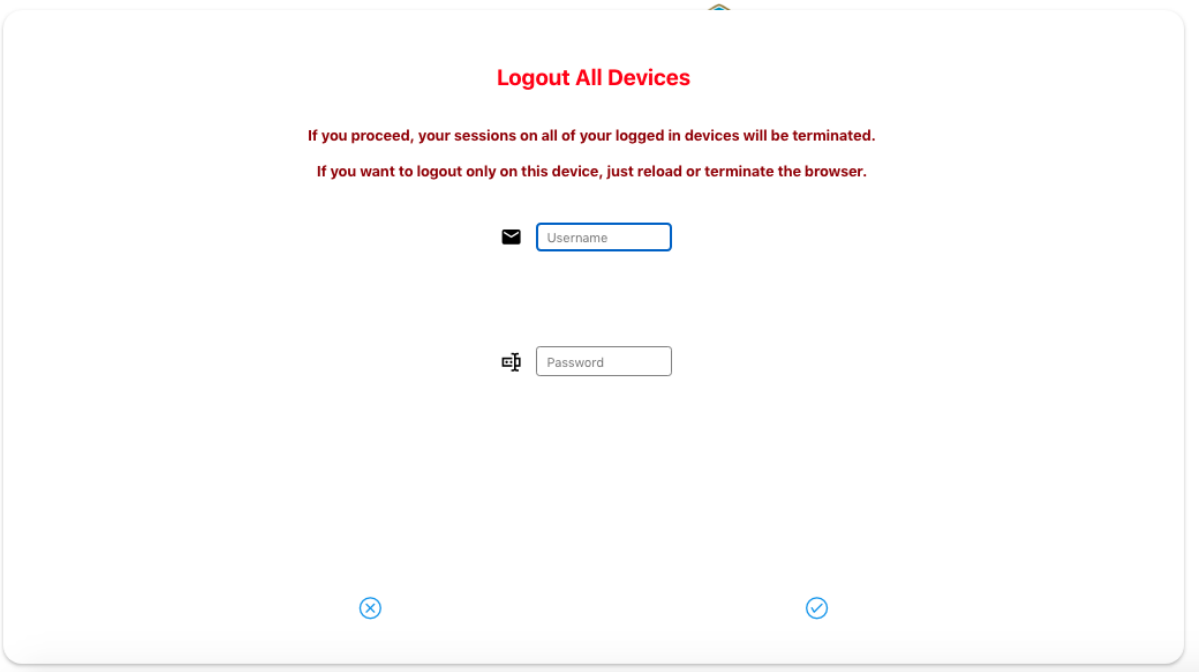
```
async function downloadFile () {
  console.log(` ${ sharedFile.fileName }`);
  console.log(`FileSystem.documentDirectory = ${ FileSystem.documentDirectory }`);

  let dnldReqTokenUri = `/conversation/sharedFileDescriptor/${ conversationId }/${
encodeURIComponent(sharedFile.fileName) }/${
encodeURIComponent(sharedFile.fileType) }/${ sharedFile.fileUri }`;
```

```
  let reqDownloadToken = await apiCall(dnldReqTokenUri);

  console.log(reqDownloadToken);

  let dnldReqUri = `${ svuSession.apiUrl }/blob/${ reqDownloadToken.fileDescrStr
}`;
  let result = await WebBrowser.openBrowserAsync(dnldReqUri);
  setDownloadResult(result);

  console.log(JSON.stringify(result, null, 4));
 }
```

الشكل 4-20: يوضح إجرائية تحميل الملف المشارك محلياً

4-4-6 الخروج من التطبيق على كافة الأجهزة Logout of All Devices:

عندما يقوم مستثمر التطبيق بالضغط على زر Logout تظهر النافذة الخاصة بذلك، وهذه العملية تتطلب من المستثمر إدخال عنوان البريد الإلكتروني الخاص به وكلمة المرور، وبعد ذلك تتم عملية الخروج من التطبيق على كافة الأجهزة الخاصة به (الحاسب الشخصي والمحمول، الحاسب اللوحي، الهاتف الذكي) كما هو موضح بالشكل التالي:



الشكل 4-21: يوضح واجهة الخروج من التطبيق على كافة أجهزة المستثمر

نبين من خلال الكود التالي آلية الخروج من التطبيق على الواجهة الأمامية:

```
const doLogout = async (userId, password) => {
  if (!(userId && password)) {
    return;
  }

  let payload = {
    userId: userId,
    password: password,
  }

  try {
    let loginResponse = await apiCall("/session/logoutAllDevices", payload);
    let sessionUpdateAction = {
      type: "API_CALL",
      newState: initSVUSessionContext,
    }

    dispatchSessionUpdate(sessionUpdateAction);

    playSound();

  } catch (error) {
    console.log(`loginResponse Error: ${error}`);
  }

}
```

الشكل 4-22: يوضح الخروج من التطبيق على كافة أجهزة المستثمر على الواجهة الأمامية

ونبين من خلال الكود التالي آلية الخروج من التطبيق على الواجهة الخلفية:

```
router.post("/logoutAllDevices", async (req, res) => {
  let loginUrl = getConfig().get("login_url");

  let userId = req.body["userId"];
  let password = req.body["password"];

  let responsePayload = {
```

```
        success: true,
        message: "All sessions on all devices have been logged out.",
    }
    if (!CryptoHelper.validateEmail(userId)) {
        return res.status(400).send("Invalid user ID");
    }
    let existingAccount = await findOne("users", { user_id: userId });
    if (!existingAccount || !existingAccount.active) {
        return res.status(401).send("Unauthorized");
    }
    if (!CryptoHelper.validatePassword(password, existingAccount.password)) {
        return res.status(401).send("Unauthorized");
    }
    await updateOne("users", { user_id: userId }, { $set: { jwt_id: "" } });
    console.log(`logged out on all devices for ${userId} `)
    return res.json(responsePayload);
})
```

الشكل 4-23: يوضح الخروج من التطبيق على الواجهة الخلفية

4-5 الأمن والحماية في التطبيق:

لقد استخدمنا في مشروعنا كل من الحلول SSL وJWT من أجل الأمن والحماية اللازمين لنا، حيث نبين من خلال الآتي الإجراءات التي قمنا بها:

SSL:

تم في مشروعنا وعلى مستوى التطوير Dev القيام بالتالي:

- إنشاء CA خاص بنا، من خلال الملف gen_root_ca.sh الموجود ضمن المجلد ssl.

```
bin/bash/!#
#
SSL_ROOT_CA_HOME=~/.ca_certs/root_ca

if [[ ! -e $SSL_ROOT_CA_HOME/root_ca_key.pem ]]; then
echo "root_ca_key.pem does not exist!\npelase generate one before running
"this script\n
exit 1
fi

if [[ -e $SSL_ROOT_CA_HOME/root_ca.pem ]]; then
```

```
echo "this will override existing ca certificate, are you sure you want
"?to proceed
read -p " please answer YES to continue: " user_answer
echo $user_answer
if [[ $user_answer != "YES" ]]; then
".... echo "aborting
exit 1
fi
fi
######################
Become a Certificate Authority #
######################
Generate root certificate #
openssl req -x509 -new -nodes -key ${SSL_ROOT_CA_HOME}/root_ca_key.pem
-sha256 -days 3600 -out ${SSL_ROOT_CA_HOME}/root_ca.pem
chmod 400 ${SSL_ROOT_CA_HOME}/root_ca.pem #
######################
######################
```

الشكل 4-24: يوضح الكود الخاص بالملف gen_root_ca.sh

- تم توليد المفتاح الخاص بمخدم التطبيق ومن ثم توقيعه من قبل الـ CA الخاصة بنا، وذلك من خلال الملف gen_root_key.sh الموجود ضمن المجلد ssl.

```
bin/bash/!#
#
SSL_ROOT_CA_HOME=~/.ca_certs/root_ca

if [[ -e $SSL_ROOT_CA_HOME/root_ca_key.pem ]]; then
"?echo "this will override existing key, are you sure you want to proceed
read -p " please answer YES to continue: " user_answer
{,,user_answer=${$user_answer #
echo $user_answer
if [[ $user_answer != "YES" ]]; then
".... echo "aborting
exit 1
fi
fi

mkdir -p $SSL_ROOT_CA_HOME
```

```
######################
Become a Certificate Authority #
######################
Generate private key #
openssl genrsa -des3 -out ${SSL_ROOT_CA_HOME}/root_ca_key.pem 4096
chmod 400 ${SSL_ROOT_CA_HOME}/root_ca_key.pem

######################
######################
```

الشكل 4-25: يوضح الكود الخاص بالملف gen_root_key.sh

- توليد الشهادة الموقعة من خلال الملف gen_certs.sh الموجود ضمن المجلد ssl.

```
bin/bash/!#
#
SSL_ROOT_CA_HOME=~/.ca_certs/root_ca
SVU_SSL_HOME=../config/ssl
######################
Create CA-signed certs #
######################
Generate a private key #
if [[ ! -e ${SVU_SSL_HOME}/key.pem ]]; then
openssl genrsa -out ${SVU_SSL_HOME}/key.pem 4096
fi
Create a certificate-signing request #
openssl req -new -key ${SVU_SSL_HOME}/key.pem -out ${SVU_SSL_HOME}/cert.csr
Create a config file for the extensions #
SVU_SSL_HOME}/cert.ext cat <<-EOF}$<
authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment,
dataEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = localhost # Be sure to include the domain name here because Common
Name is not so commonly honoured by itself
```

```
DNS.2 = bar.$SVU_DOMAIN_NAME # Optionally, add additional domains (I've#
(added a subdomain here
IP.1 = 127.0.0.1 # Optionally, add an IP address (if the connection which
(you have planned requires it
EOF
Create the signed certificate #
openssl x509 -req -in ${SVU_SSL_HOME}/cert.csr -CA
\ $SSL_ROOT_CA_HOME/root_ca.pem
\ CAkey $SSL_ROOT_CA_HOME/root_ca_key.pem -CAcreateserial-
out ${SVU_SSL_HOME}/cert.pem -days 825 -sha256 -extfile-
${SVU_SSL_HOME}/cert.ext
```

الشكل 4-26: يوضح الكود الخاص بالملف gen_certs.sh

- بعد ذلك قمنا بإدخال الشهادة الموقعة والمفتاح الخاص بالتطبيق ضمن موقع Chrome يدوياً، كي نضمن تشغيل التطبيق الخاص بمشروعنا بالشكل الأمثل من الناحية الأمنية.

JWT:

تم في مشروعنا استخدام الكائن JSON Web Token وذلك لزيادة السرية والأمان في التطبيق، وذلك وفق الآلية التالية:

- يتم التأكد من هوية مستثمر التطبيق بعد إدخاله المعلومات الأساسية الخاصة به (البريد الإلكتروني وكلمة المرور).
- بعد التحقق هوية مستثمر التطبيق، يتم في الواجهة الخلفية توليد JWT خاص وتوقيعه باستخدام المفتاح المعلن Public Key العائد للمخدم، ثم يتم إرساله إلى مستثمر التطبيق المخول Client.
- يتم إرسال الـ JWT الموقع مع كل طلب مرسل من قبل المستثمر request حيث يقوم المخدم بالتأكد من صحته من خلال فك التشفير بواسطة مفتاحه الخاص.

ومن خلال استخدامنا للكائن JWT يمكننا حل الكثير من المشاكل المتعلقة بالمصادقة والترخيص.

الحلول الأمنية:

نعرض من خلال الآتي أهم الحلول والخطوات التي تمكنا من تطبيقها ضمن مشروعنا:

- تشفير كافة البيانات المتبادلة بين المستثمر والتطبيق الخاص بنا، من خلال استثمارنا لتقنية SSL. الشيء الذي أمن تلك البيانات من الكشف من جهة، وأمن لنا التصدي لتهديد الرجل في المنتصف Man in the middle attack من جهة أخرى.
- استخدامنا لـ MongoDB يمنع هجوم SQL Injection كون قاعدة المعطيات هذه ليست علائقية NoSQL DB.
- كلمة السر طويلة بشكل كافي 12 خانة.
- استخدامنا لـ JWT أمن لنا الآتي:
  - إجبار كل الـ REST endpoints في التطبيق على وضع origin على كافة الترويسات Headers.
  - لا يتم حفظ أي معلومات خاصة credentials على كل من واجهتي التطبيق الأمامية والخلفية، الشيء الذي يؤمن الحماية من هجوم XSS.
  - عدم الحاجة نهائياً لاستخدام الـ cookies وبالتالي لا يتم حفظ أي معلومات خاصة بالجلسة Session الشيء الذي يؤمن الحماية ضد هجوم انتحال الشخصية.
  - لا يتم حفظ أي معلومات خاصة بالرمز JWT في كلا واجهتي التطبيق الأمامي والخلفي وليست ذات قيمة دائمة.
  - يقوم التطبيق تلقائياً بعد ساعة من الخمول بإخراج المستثمر والعودة به إلى واجهة الدخول الأساسية Login.
- تشفير كافة المعطيات المخزنة في الواجهة الخلفية بشكل كامل.
- استخدامنا لمخدم NGINX الذي يقوم في مشروعنا بدور موازنة الحمل Load Balancing والتوجيه Router وأخيراً بدور الخانق Throttling الشيء الذي أمن لنا الحماية اللازمة ضد أهم التهديدات، فهو لا يسمح للتدفق بأن يستمر بشكل دائم، وكذلك لا يسمح لاتصالات HTTP لأنه يسمح فقط باتصالات HTTPS فقط لا غير.

الفصل الخامس: الخاتمة والتطوير المستقبلي

- ملخص

قدمنا من خلال التطبيق نموذج مصغر عن تطبيق محادثة متكامل، يمكن المستثمرين من إجراء المحادثة وتبادل الملفات فيما بينهم، وبشكل آمن بمستوى عالي جداً، مع تطبيق أهم الحلول التقنية بكلا واجهتي التطبيق الأمامية والخلفية، بكلا الاتجاهين H/W و S/W.

- التطوير المستقبلي Future Enhancement

مع كل ما قدمه المشروع من إمكانيات خاصة بمجال المحادثة ونقل الملفات، فإننا نرغب في المستقبل بتطوير التطبيق بحيث يقدم خدمات أكثر ضمن بيئة عمل أوسع، وذلك وفق الآتي:

- تطوير المحادثة بحيث تشمل مستقبلاً عملية التراسل الصوتي والمرئي بما يخص الـ Media Chat.
- تطوير التطبيق بحيث نجعله يعمل على الهواتف الذكية مستقلاً عن المتصفح Mobile App أي أن يكون من طراز Native App.
- التطوير العملي للتطبيق من ناحية الاستخدام الفعال للتجهيزات الأساسية، من ناحية المخدمات وقواعد المعطيات، بحيث نتمكن من استخدام هذا التطبيق ضمن المؤسسات الحكومية والخاصة بالشكل الأمثل وضمن الحفاظ على استمرارية العمل حتى مع التوسع الكبير بعدد المشتركين فيه، مع إمكانية استخدام مفهوم المجموعات الخاصة Grouping ضمن العمل.

المراجع

- "SDD Technology blog: Definition of cross platform". SDD Technology. Retrieved 2020-10-18.
- "Design Guidelines: Glossary". java.sun.com. Retrieved 2011-10-19.
- "12 benefits of Xamarin Cross-platform app development". HeadWorks. 15 Mar 2019.
- https://www.sdd-technology.com/news/definition-of-cross-platform
- https://www.w3.org/TR/ws-arch/
- https://www.w3.org/2002/ws/
- https://en.wikipedia.org/wiki/Web_service
- https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/
- Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)"
- Fielding, Roy (June 2014). "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, Section 4"
- "Fielding discusses the development of the REST style"
- https://exyte.com/blog/how-to-consume-rest-api-in-javascript
- https://en.wikipedia.org/wiki/Node.js
- https://nodejs.org/en/docs/
- https://netflixtechblog.com/node-js-in-flames-ddd073803aa4
- https://aws.amazon.com/getting-started/hands-on/deploy-nodejs-web-app/
- https://www.npmjs.com
- https://core.ac.uk/download/pdf/77240211.pdf (Cross-Platforms)
- https://www.researchgate.net/publication/331429981_React_Native_Application_Development
- https://reactjs.org
- https://journals.sagepub.com/doi/full/10.1155/2013/867693 (WebSocket)

- https://www.ijert.org/a-review-on-various-aspects-of-mongodb-databases
- https://www.researchgate.net/publication/327120267_MongoDB_-_a_comparison_with_NoSQL_databases
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5191106/ (MongoDB)
- https://en.wikipedia.org/wiki/MongoDB
- http://www.websocket.org/quantum.html
- https://blog.chromium.org/2009/12/web-sockets-now-available-in-google.html
- https://journals.sagepub.com/doi/full/10.1155/2013/867693 (WebSocket)
- https://www.javatpoint.com/nginx-tutorial
- https://www.javatpoint.com/redis-tutorial
- https://www.canto.com/blog/binary-large-object/
- https://www.appknox.com/blog/how-jwt-helps-in-securing-your-api
- https://crashtest-security.com/broken-authentication-and-session-management/
-