جامعة تشرين

كلية الهندسة المعلوماتية

قسم البرمجيات ونظم المعلومات

السنة الخامسة

منصة أعمال حرة

(مشروع تخرج)

إعـــــــداد الطلاب

الحسن صالح

أحمد زحلوط

اسم المشرف:

د. باسل حسن

2021-2020

المحتويات

1. مقدمة

يهدف مشروعنا الى تسهيل ممارسة الأعمال الحرة في سوريا خصوصاً. تكمن أهمية ذلك في قلة الشركات البرمجية حالياً (مع أن عددها بدأ بالتزايد) ومتطلبات تلك الشركات من خبرات سنين من العمل.

إن الاعتماد على الأعمال الحرة يتيح فرصة عمل لأي شخص يملك خبرة في مجاله، كما يتيح لصاحب الأعمال خيارات عديدة في اختيار الشخص المناسب وسهولة التواصل.

فمثلاً نسبة كبيرة من الطلاب المتخرجين ما تزال تبحث عن عمل نهدف الى إزالة ذلك عن طريقة هذه المنصة كما ستمكن هذه المنصة من إعطاء خبرة للشخص تمكنه من العمل في الشركات إن أراد ذلك.

في الواقع حتى بعض الشركات البرمجية قد تلجأ الى عامل حر في بعض الأحيان لأداء مشروع لعدة أسباب مثلاً في بعض الأحيان قد تكون أجرة تكليف شخص مستقل أقل لمشروع معين من موارد الشركة أو قد يكون الشخص ذو خبرة جيدة في مجال معين.

توجد منصات كثيرة للأعمال الحرة خارجاً مثل منصة *Upwork Inc.* لكن للأسف معظم هذه المواقع تكون محظورة في سوريا أو يوجد صعوبة في عملية الدفع لعدم امتلاك معظم السكان (خصوصاً الطلاب) للبطاقات المصرفية وغيرها.

الشكل 1 منصة Upwork

مصدر الصورة: https://www.upwork.com/

2. المتطلبات

1.1. وصف النظام

إن منصة *F.A.I.T.H (Freelancers Association & Information Technology Hub)* هي موقع الكتروني يتيح للمستخدمين إنشاء حسابات بنوعين إما عامل حر *Freelancer* أو صاحب عمل *Stakeholder*، ولكن يجب على مدير الموقع الموافقة على أي حساب عند إنشاءه وذلك لضرورة التأكد من المعلومات.

يجب أن يكون صاحب العمل قادر على إنشاء المشاريع وجعلها قابلة للظهور للعلن أم لا بالإضافة لرؤية أي عملية مزايدة عليها وتقديم طلب عمل للعامل الحر. يمكنه أيضاً أن يكتب تعليقاً في أي مزايدة من أجل النقاش.

يمكن للعامل الحر أن يعدل معلوماته وأن يرى المشاريع العلنية والمزايدة عليها، كما بإمكانه أن يرفض أو يقبل طلبات العمل التي يتم تقديمها له.

يمكن أيضاً لصاحب العمل تقييم العاملين عن طريق خدمة مستقلة.

1.2. المتطلبات الوظيفية

لقد قمنا بإنشاء مستند مستقل من أجل المتطلبات الوظيفية ويمكن رؤيته على الرابط التالي (مع الأخذ بالعلم أن المستند غير محدث لأخر المتطلبات حالياً):

https://github.com/TheKiddos/faith/blob/main/Faith-SRS.pdf

ولكننا سنقوم بشرح بعد المتطلبات الوظيفية هنا.

مثلاً من الوصف السابق يمكننا استخراج متطلب **إدارة المشروع *Project Management*** ويكون وصفها كما يلي:

يجب أن يملك صاحب العمل واجهة تمكنه من إدارة المشروع حيث يمكنه ارسال طلبات توظيف، أن يقوم بالتعديل على خصائص المشروع وأن يجعل المشروع علني أو خاص. إضافة الى رؤية بعض الاحصائيات حول

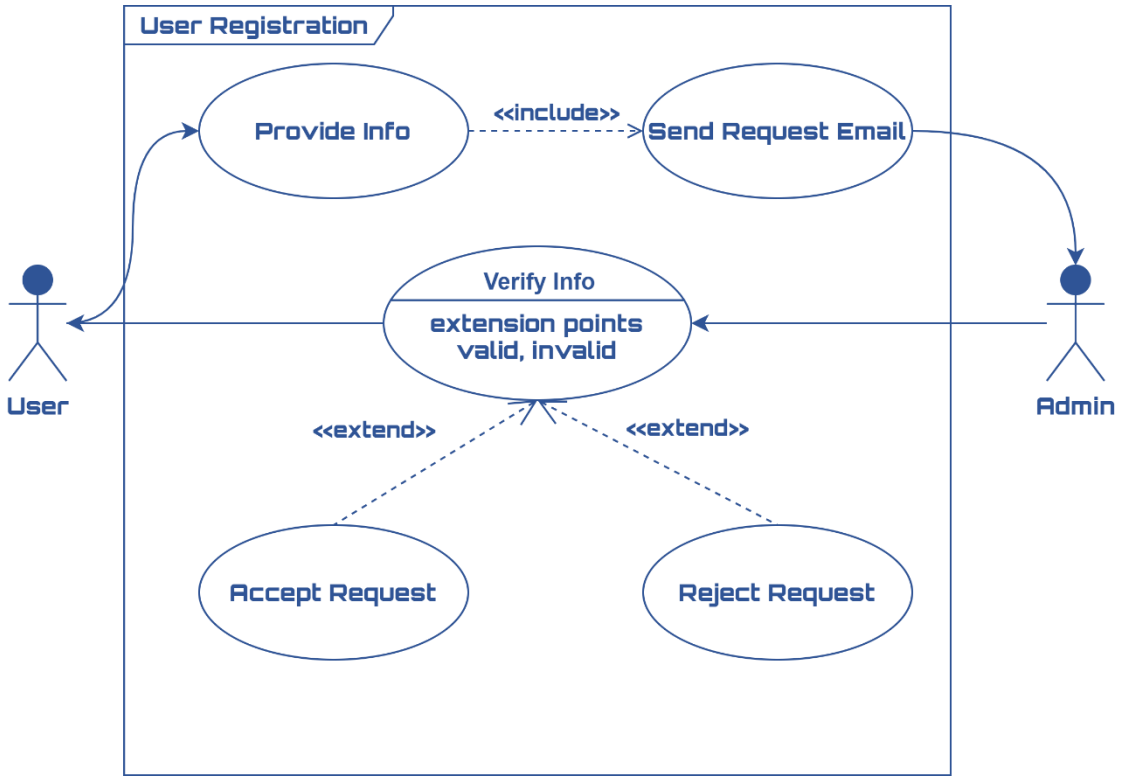
هذا المشروع. أيضاً إن كان المشروع قد أصبح في طور العمل (تم تعيين عامل) يجب أن يرى صاحب العمل ذلك ويجب أن يكون قادراً على التواصل أو إنشاء مهمات للعامل الحر.

3. حالات الاستخدام

توجد أيضاً حالات الاستخدام في المستند السابق.

يجب علينا أن نعلم أننا قمنا بكتابة وصف هذه الحالات بطريقة ال *BDD* والتي سنتطرق لها لاحقاً.

قمنا بإنشاء عدة مخططات لحالات الاستخدام بدلاً من واحد حتى نجمع الحالات المتشابه ضمن إطار واحد بدلاً من وضع كل الحالات في مخطط واحد كبير. مثلاً إن حالات الاستخدام الخاصة بالحسابات يمكن جمعا في المخطط التالي:



الشكل 2 حالات الاستخدام للحسابات

شرح حالات الاستخدام:

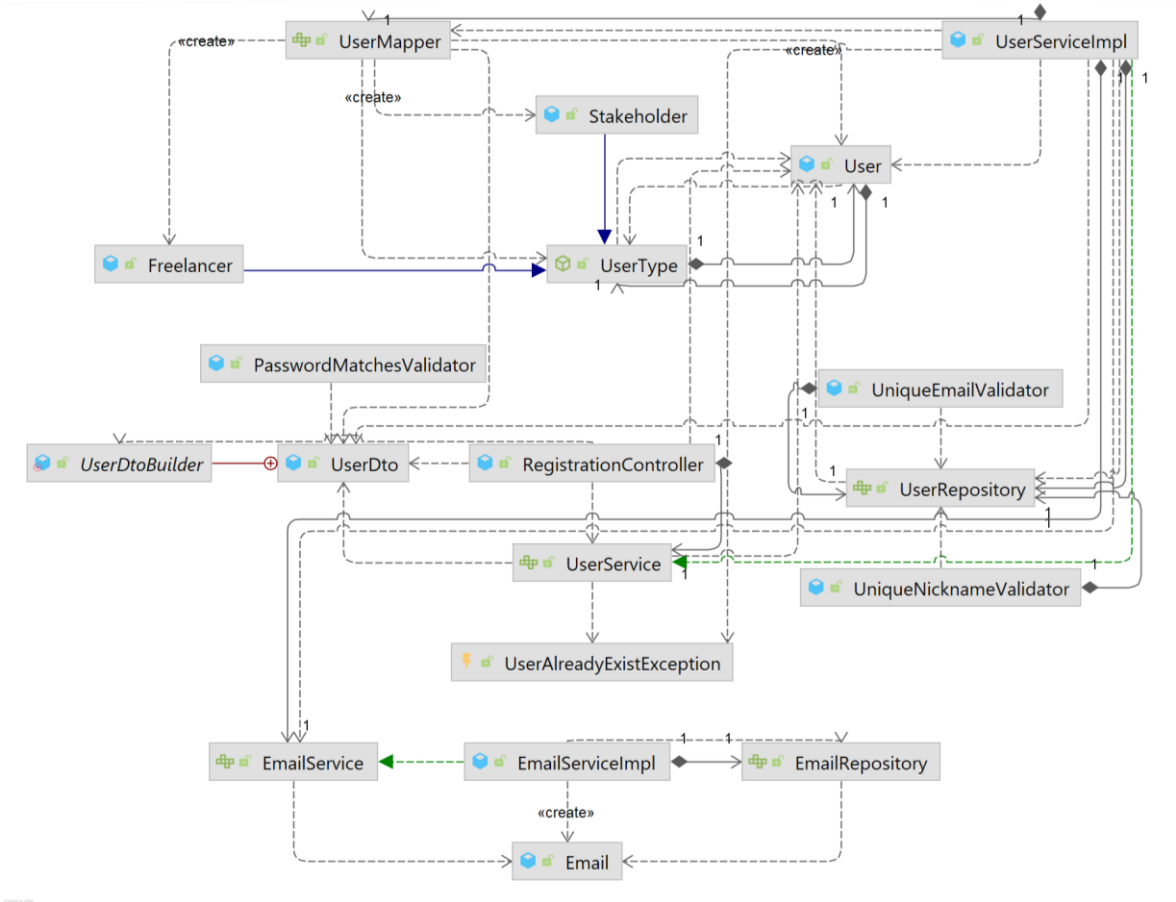
يمكن لأي مستخدم *User* من تزويد النظام بمعلوماته وعند إتمام هذه الحالة يتم بشكل فوري إرسال بريد الكتروني لمدير الموقع.

يمكن لمدير الموقع من التحقق من هذه المعلومات وعند ذلك يمكنه إما رفض أو قبول تلك المعلومات.

4. مخطط الصفوف

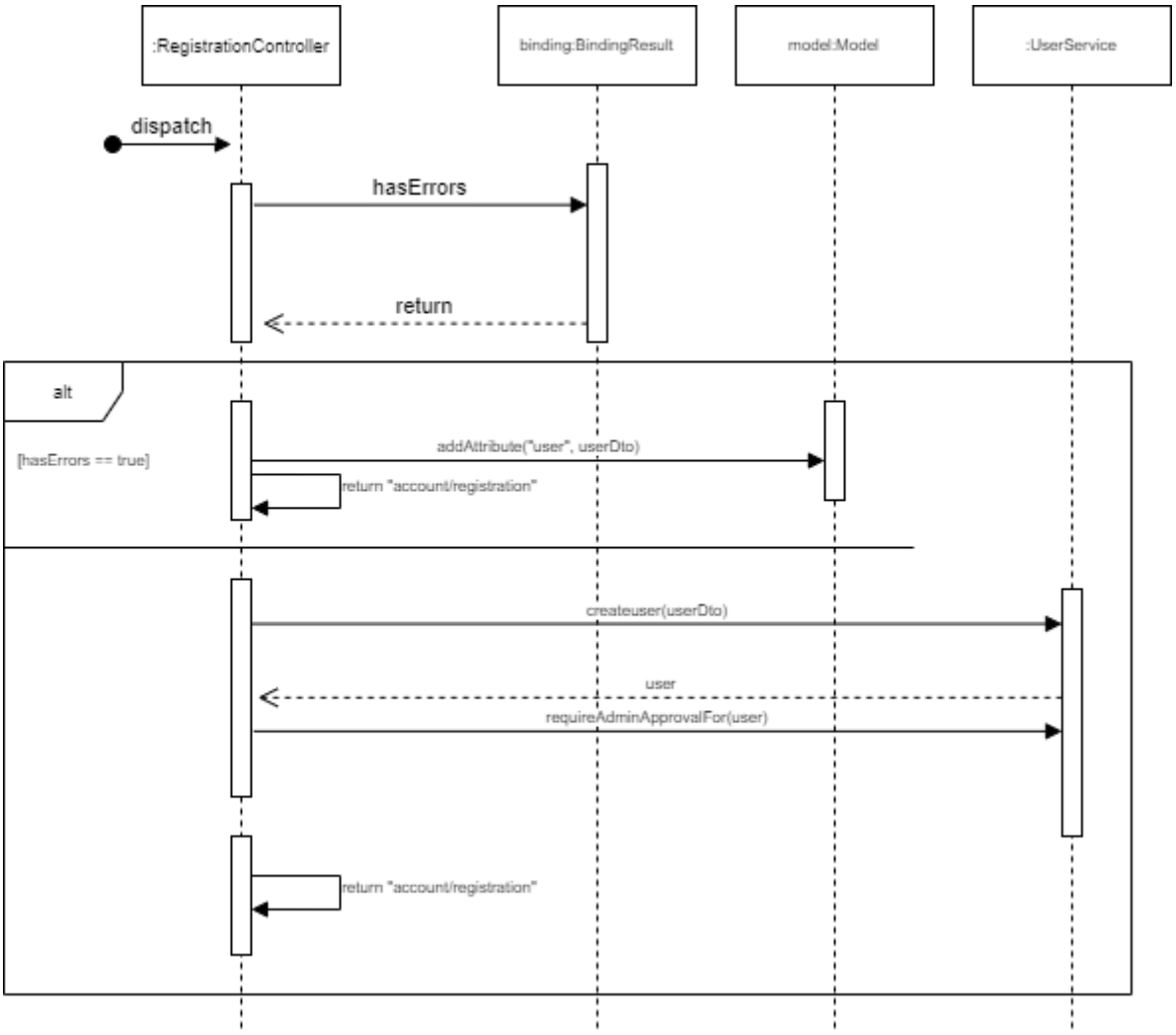
لن نقوم بوضع مخطط الصفوف بأكمله بسبب حجمه بل سنضع مخطط صفوف يعبر عن ميزة إنشاء الحساب السابقة.

ملاحظة: المخطط التالي تم توليده باستخدام برنامج *Intillij* لهذا قد تظهر علاقات أكثر من المعتاد. وقمنا بإخفاء الحقول والطرائق والواصفات ليتسع المخطط.



5. مخطط التسلسل

يبدو مخطط ال *Sequence* لعملية إنشاء حساب من منظور ال *Controller* كما يلي:



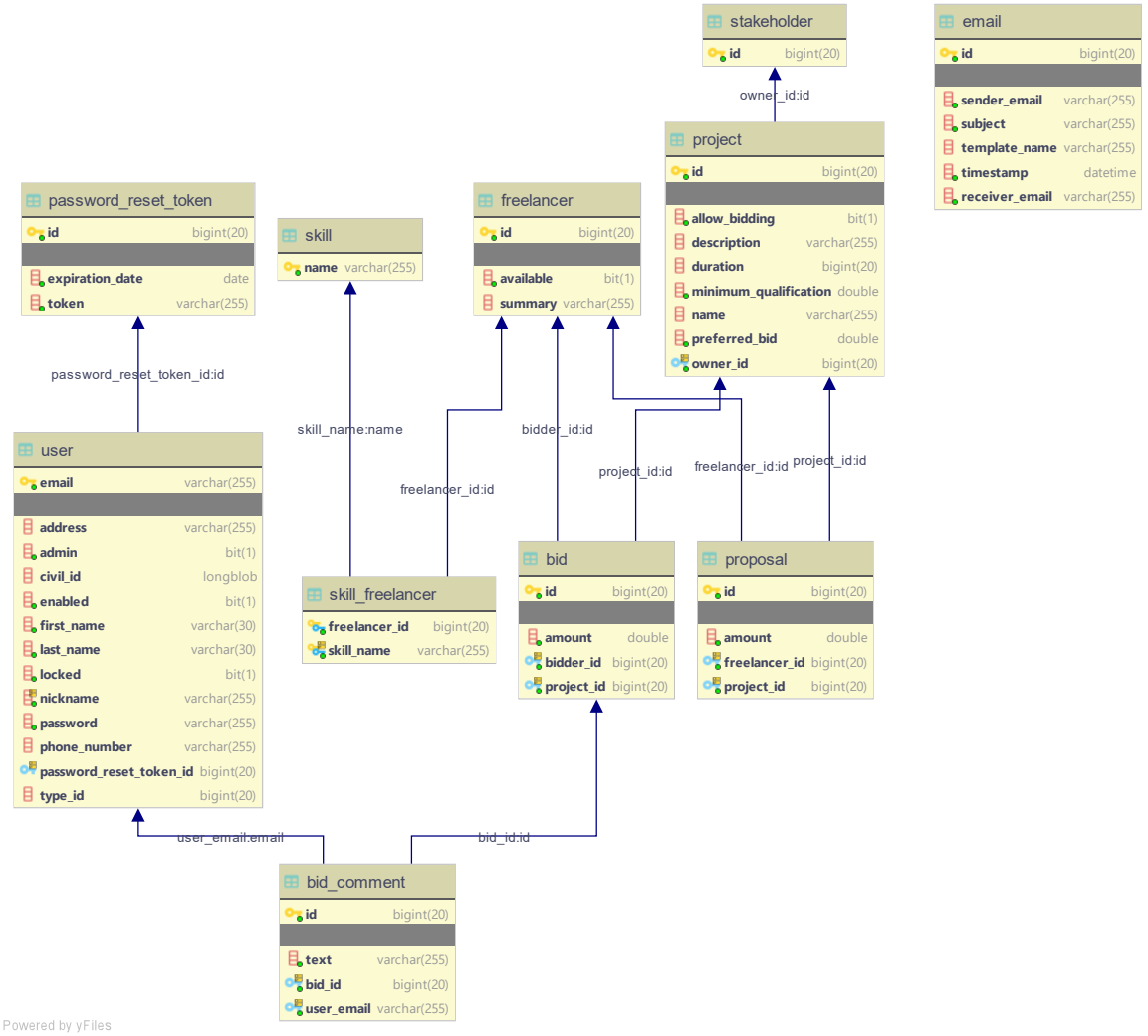
6. مخطط قاعدة البيانات

لم نقوم بأي عملية تصميم للقاعدة وذلك بفضل وجود ال *ORM (Object-Relational Mapper)* والذي يقوم بالتحويل بين الكائنات والعلاقات بشكل سلس دون حاجة الى التدخل وذلك يزيل عبأ إضافة كود مستقل

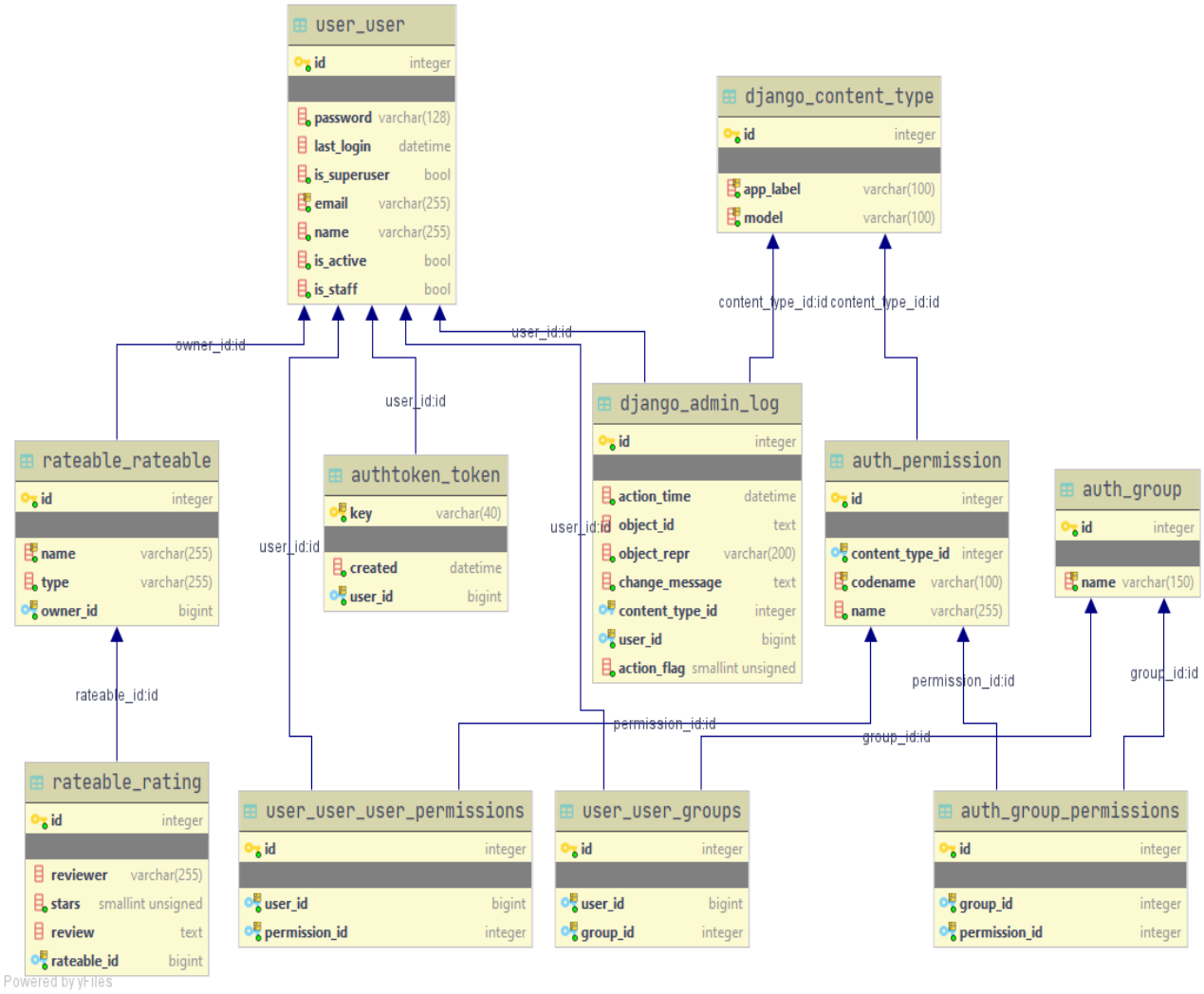
لل *SQL* (أو أياً كانت اللغة المستخدمة) حيث أي تغيير يطرأ نقوم فقط بتغيير الكود ليتولد الكود الخاص بال *SQL* المقابل تلقائياً.

قمنا باستخدام *MySQL* كمحرك قواعد بيانات في مشروع المنصة بينما قمنا باستخدام *PostgreSQL* لمنصة التقييم.

الشكلين يوضحان المخططين الخاصين بقاعدة البيانات لكل منهما حيث تم إنشاءه باستخدام *Intellij*.



الشكل 3 مخطط ال ER للمنصة



الشكل 4 مخطط ER لخدمة التقييم

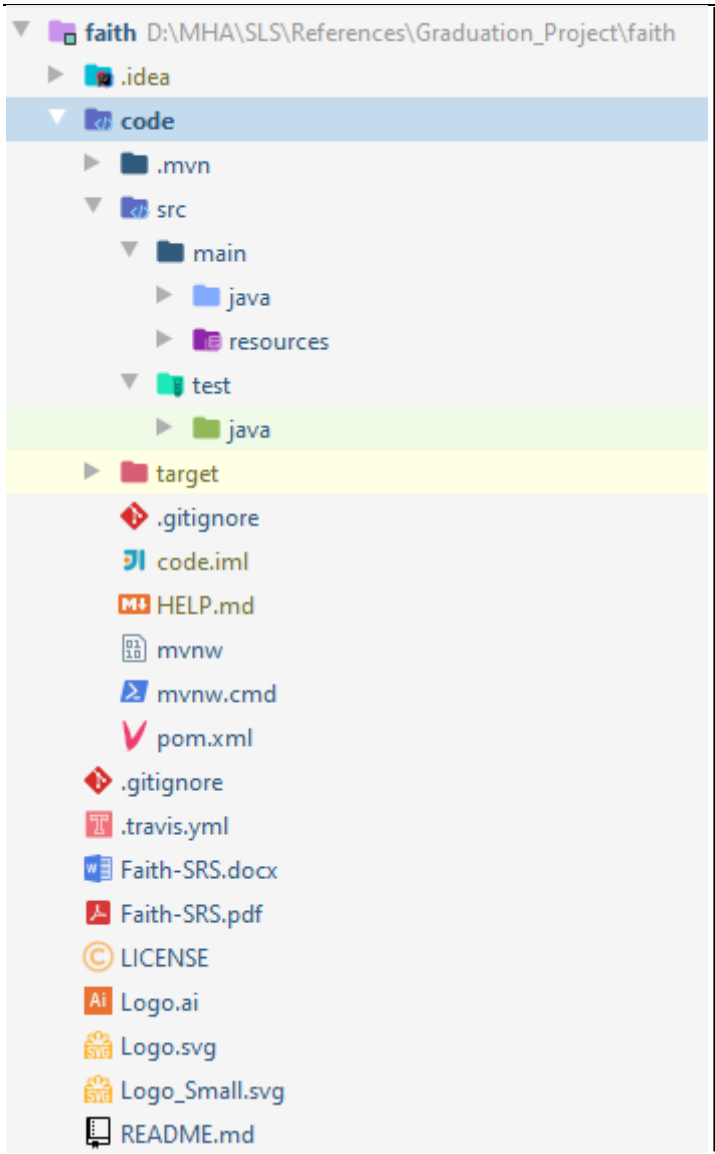
طبعاً في خدمة التقييم بعد الجداول لم نقم بإنشائها بل تتبع ل *Django* بشكل افتراضي مثل *auth_group*.

توفر أيضاً *Django* ميزة ال *Migrations* وهي ميزة تتبع التغييرات الحاصلة على القاعدة للقيام بها أو التراجع عنها ويتم انشائها بلغة وسيطيه.

يمكن تحقيق شيء مماثل باستخدام *Spring* ولكن بلغة ال *SQL* وليست بفعالية *Django*.

7. بنية المشروع

1.3. بنية F.A.I.T.H



تتألف بنية المنصة من بنية *Maven* و*Spring Boot* قياسية *[4]*.

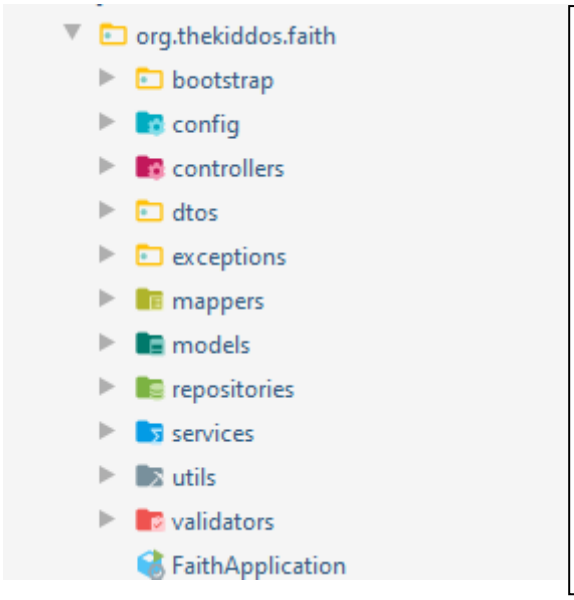
سنذكر شرح بعض هذه الملفات في أقسامها الخاصة ونشرح البقية هنا:

لدينا الكود المصدري مكتوب بلغة الجافا كله داخل مجلد *Code*.

حيث *src* يحوي على الكود الفعلي والمصادر مثل الصور والصفحات وغيرها بينما *test* يحوي على أكواد الاختبارات والتي سنتحدث عنها لاحقًا.

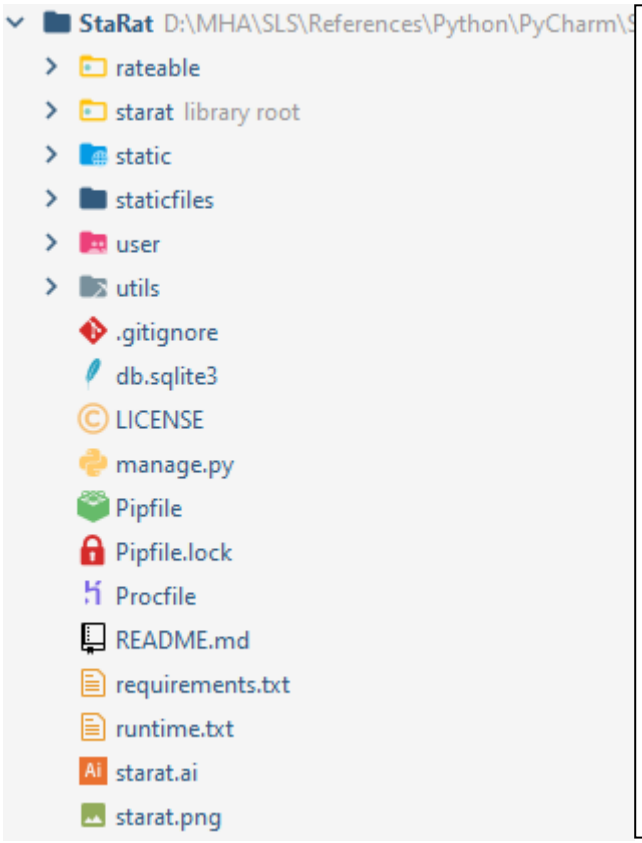
Pom.xml هو ملف اعدادات خاص بأداة *maven* والتي تعتبر أداة *Build* وإدارة مشاريع.

الشكل 5 بنية FAITH



كما نرى الحزمة مرتبة بشكل واضح وتتبع بنية *Spring Boot* مثلاً، *Bootstrap* تحوي الأكواد التي تنفذ عند تشغيل المنصة. *Config*: تحوي الاعدادات. وكما نرى نحن نعمل ضمن *MVC* حيث لدينا *Models* و*Controllers*. *Dtos*: تحوي تمثيلات للموديلات وسيطية، *mappers*: تحوي صفوف للتحويل بين *dtos* والمودلات، *repositories* تحوي ما يلزم للتعامل مع قاعدة البيانات.

الشكل 6 حزمة المنصة



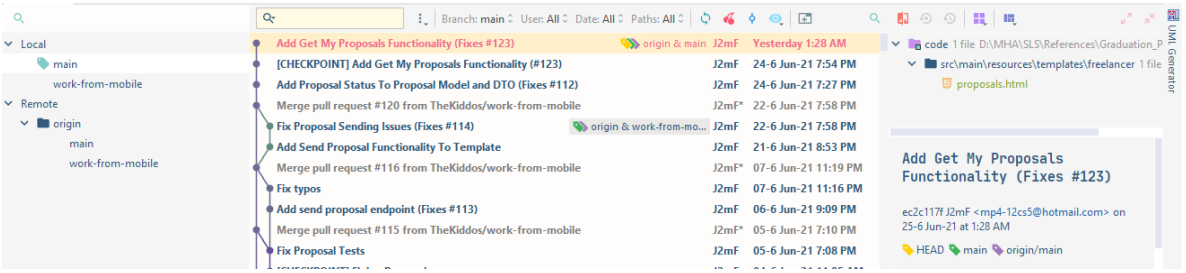
تحوي خدمة التقييم بنية مشروع *Django* قياسية لن نقوم بشرحها. لكن يوجد بعض الملفات مثل:

Procfile ويحوي التعليمات اللازمة لتعريف الخدمات التي نريدها في *Heroku* وهي خدمة استضافة.

الشكل 7 بنية خدمة التقييم

8. التحكم بالإصدار

قمنا باستخدام *Git* منذ بداية العمل على المشروع وقمنا باستخدام أفضل الممارسات بجعل كل *Commit* تحتوي ميزة أو مجموعة ملفات مترابطة ويمكن رؤيتها في الشكل التالي:



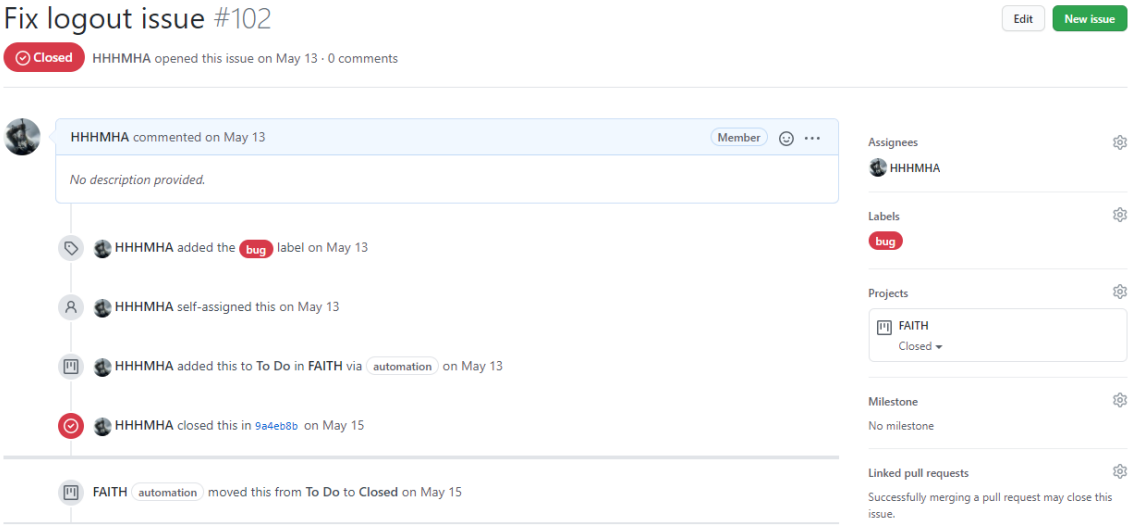
الشكل 8 ال Commits

أيضاً قمنا برفع كل من المشروعين على *Github* ويمكن الاطلاع عليهما من خلال الرابطين:

TheKiddos/faith: F.A.I.T.H. is a freelancers website for Syria created as a graduation project for Tishreen University (github.com)

TheKiddos/StaRat: StaRat is a simple django rest api to rate businesses created as part of Tishreen Graduation Project (github.com)

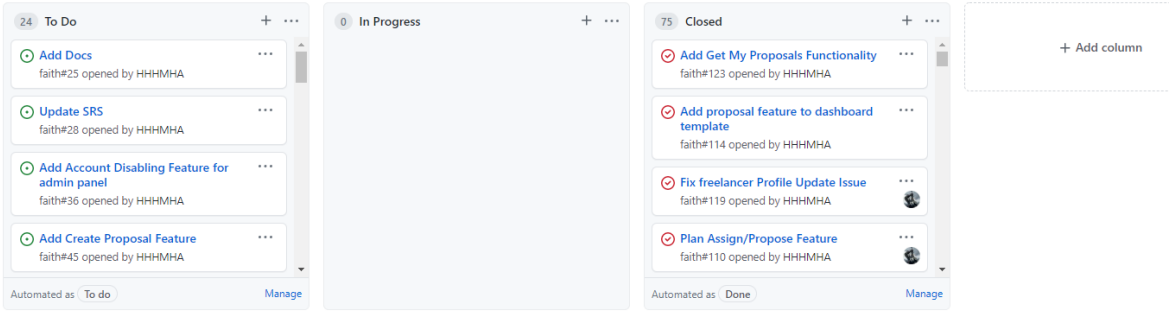
قمنا باستخدام ميزة ال *Issue Tracker* حيث قبل العمل على أي ميزة نقوم بإضافتها كما يوضح الشكل:



الشكل 9 ال Issues

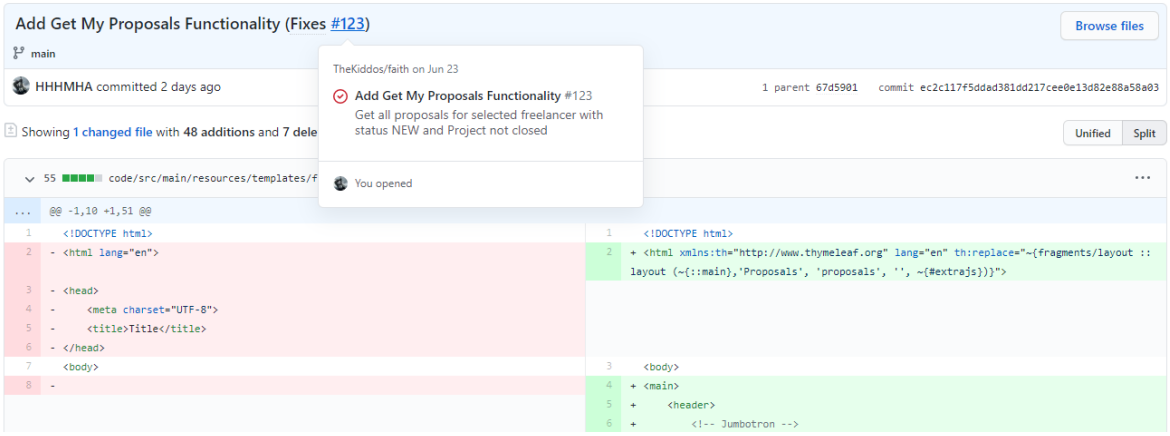
حيث عند الانتهاء من *Issue* يمكن اغلاقها.

وقمنا أيضاً بربط هذه ال *Issues* بميزة المشاريع حيث تقدم واجهة سهلة للأعضاء لمعرفة سير المشروع:



الشكل 10 إدارة المشروع

أحد المزايا التي قمنا باستخدامها هي ربط ال *Issue* بال *Commit* حيث يمكن ذكرها وحتى إغلاقها من الرسالة مباشرةً عن طريق ذكر رقمها:



الشكل 11 ذكر رقم ال Issue في ال Commit

9. البيئة والأدوات

تم استخدام لغة الجافا مع *Spring Boot* لإنشاء المنصة حيث قمنا بالعمل على *Intillij*.

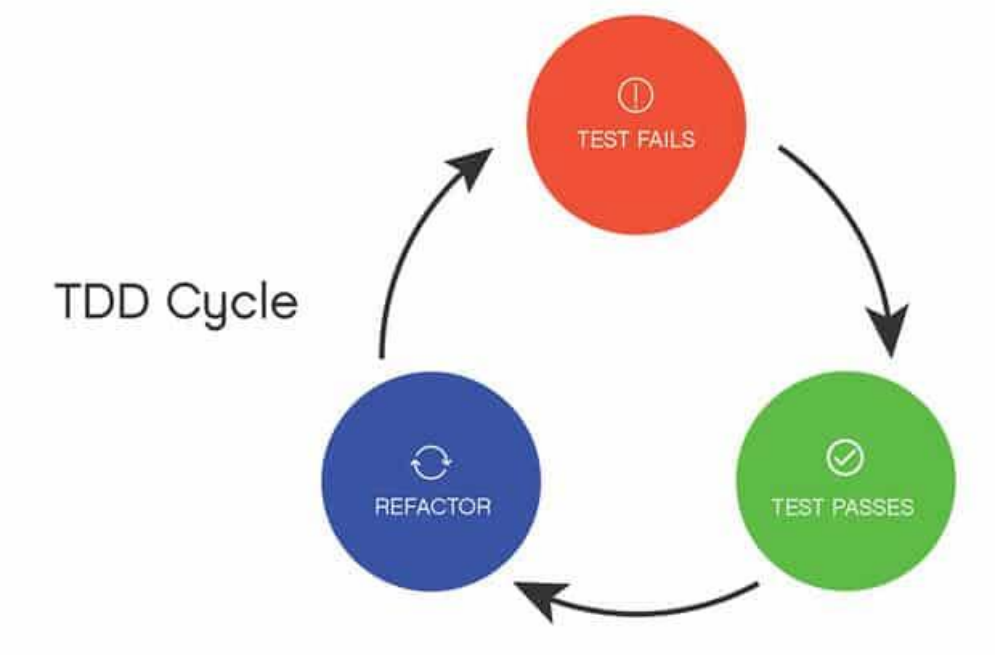
لإنشاء وتشغيل المشروع وتحميل المكاتب تم استعمال *Maven*.

بينما لخدمة التقييم قمنا باستعمال *Python* و*Django* وقمنا باستخدام *Pipenv* كبيئة وهمية. ولل *api* تم استخدام مكتبة ال *Django Rest Framework* الشهيرة.

10. التطوير المقاد بالاختبارات

يعتبر التطوير المقاد بالاختبارات *Test-Driven Development* او *TDD* اختصارا من الممارسات المهمة التي قمنا باستخدامها. ببساطة تعتمد هذه الطريقة على كتابة الاختبارات قبل كتابة الكود. هناك عدة أهداف من ذلك أولا هنا نحن سوف نفكر ببساطة بواجهة الكود القابلة للاستخدام بدلا من تفاصيل التحقيق مما ينتج عنه تصميم ابسط غالبا، أيضاً نحن نكتب فقط الكود الضروري ونسبة التغطية ستكون كاملة. *[1]* أيضا هذا يجبر العمل على دفعات ووحدات صغيرة بالإضافة للميزات العامة للاختبارات مثلا عند وجود اختبار يمكننا القيام بعميلة ال *Refactor* وبعدها يمكننا تشغيل الاختبارات للتأكد من أننا لم نقم بأي خطأ أثناء العملية مما يسرع الوقت أيضا هذه الاختبارات بمثابة توثيق للنظام ويمكن لشخص آخر من قراءتها معرفة الهدف من اي كود او وحدة معينة.

تعتمد ال *TDD* على دورة الحياة التالية*[2]*



تبدأ هذه الدورة بكتابة اختبار ما

هذا الاختبار سيفشل (لم نقم بكتابة كود بعد)

الان نقوم بكتابة ابسط كود يجعل هذا الاختبار ينجح

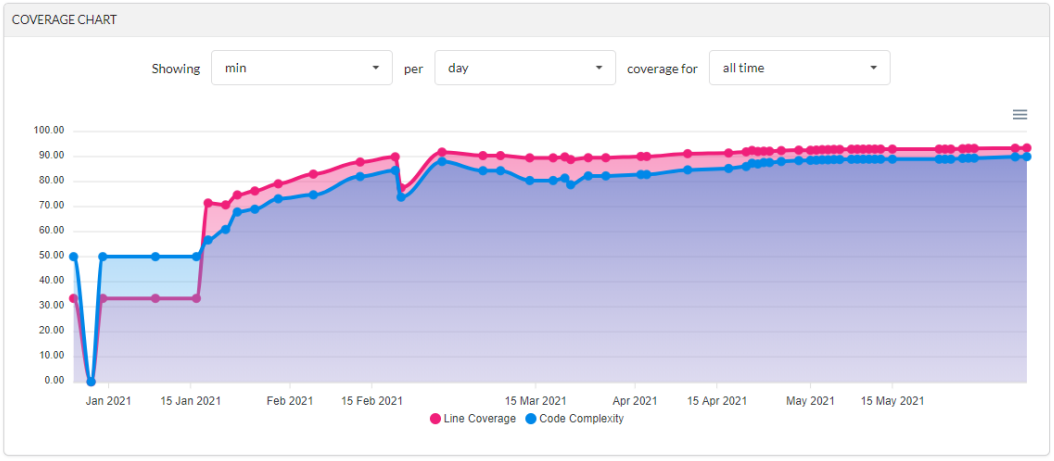
ثم بعد التأكد من نجاح الاختبار نقوم بعملية ال *Refactoring* والتعميم للكود (يمكن بدلا من ذلك كتابة اختبار اخر يجبر على التعميم وتدعى هذه الطريقة بال *Triangulation*) ثم تشغيل الاختبارات للتأكد من انها ما زالت تنجح واعادة الحلقة.

يمكن رؤية جميع الاختبارات التي قمنا بكتابتها في مجلد ال *tests* ضمن مشروع المنصة.

حيث قمنا باستخدام مكتبة *Junit* بالإضافة الى استخدام تقنية ال *Mocking* في عدة حالات وهي تقنية تسمح بمحاكاة الصفوف المعقدة أو الخارجية والاعتماديات.

حيث قمنا بإنشاء *Unit Tests* و *Integration Tests* للأسف تأخذ هذه الاختبارات فترة تشغيل طويلة نسبياً بسبب وجود *Spring Boot Context*.

يمكن رؤية تقرير عن تغطية المشروع على *Github* حيث قمنا بتضمين خدمة خارجية تدعى ب *Code Cov* ويمكن رؤيتها على الرابط التالي *Codecov* ويمثل الشكل التالي حالة المشروع:



الشكل 12 تغطية الكود

حيث قمنا حالياً بالوصول لنسبة تغطية *93%* وفي الواقع النسبة أعلى من ذلك لكن بسبب استخدام مكاتب خارجية لا يتم احتساب بعض الأجزاء من الأكواد.

11. التكامل المستمر

ان التكامل المستمر *Continues Integration* او اختصارا *CI* يعد من الأدوات التي يجب اضافتها لأي مشروع لما يتيحه من إمكانيات

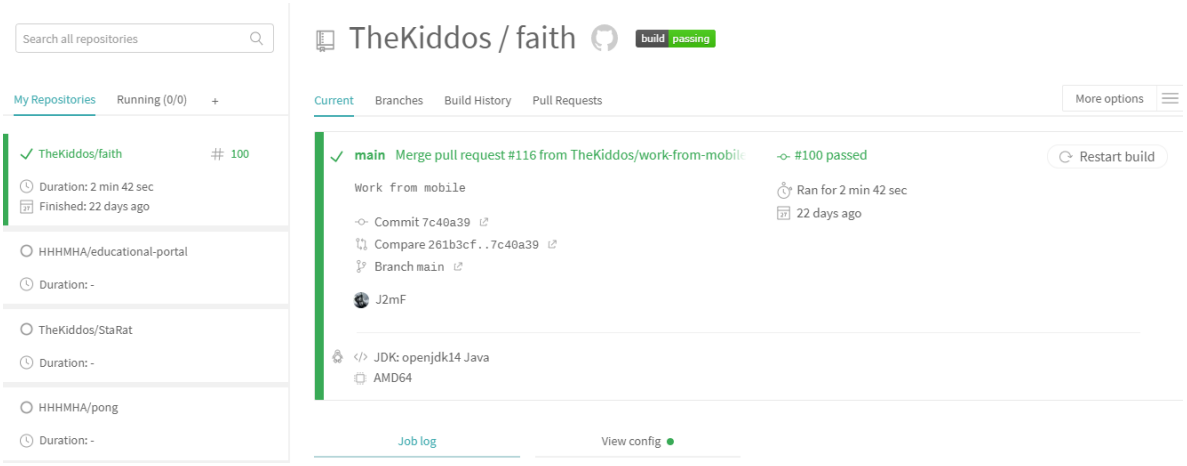
بأبسط أشكاله يتيح لنا ال *CI* بتنفيذ عملية ما بعد رفعنا لكود ومشاركته

في مقال الخبير في هندسة البرمجيات الشهير *Martin Fowler* قام بوصف التكامل المستمر كالتالي:

"التكامل المستر هو ممارسة في تطوير البرمجيات حيث يقوم أعضاء الفريق بدمج/اكمال عملهم بشكل متكرر، عادة كل شخص يقوم بذلك مرة في اليوم على الأقل مما يؤدي الى عدة تكاملات في اليوم. كل تكامل يتم التحقق منه باستخدام عملية بناء مؤتمتة (تضمن الاختبارات) للتحقق من أخطاء التكامل بأسرع وقت ممكن." *[3]*

كيف نستفيد من ذلك مثلا في مشروعنا نحن نقوم بكتابة اختبارات كما قلنا يمكننا جعل ال *CI* يقوم بتشغيل هذه الاختبارت على اي كود يتم رفعه لدينا والحصول على نتيجة هذه الاختبارت عن طريق البريد الاكتروني. هذا يضمن اننا دائما سنحصل على كود يعمل او سنعلم ان حصل خطأ ما

كما يوضح الشكل نتيجة احد ال *Builds* التي قام بها *TRAVIS-CI*



الشكل 13 التكامل المستمر

كما يوفر وضع حالة لل *Build* على *github* مباشرة

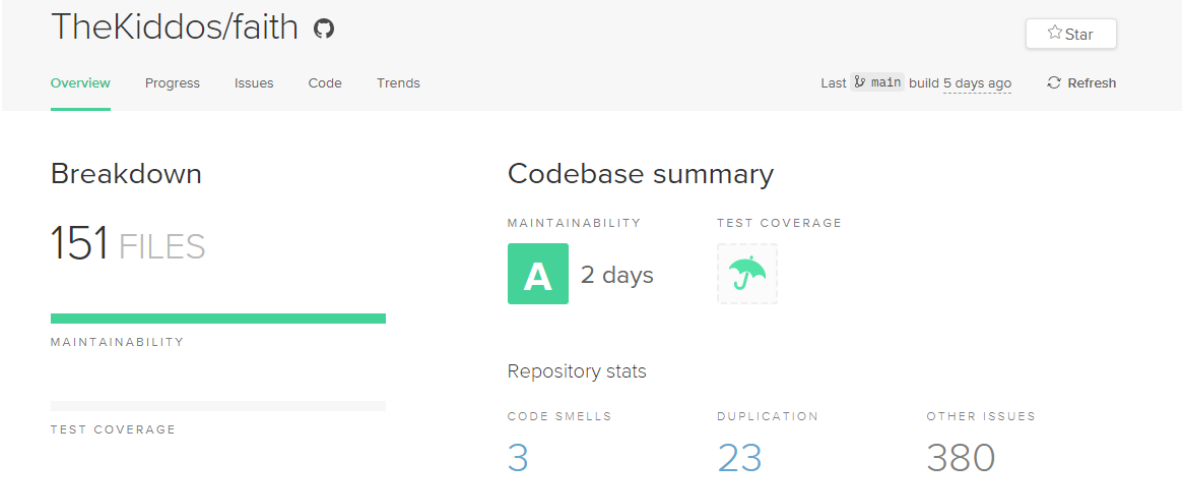
12. جودة الكود

قمنا بتضمين خدمة للتحقق من نظافة وجودة الكود بشكل عام (طبعاً لا يمكن معرفة ذلك بتلك البساطة لكن يمكن للأداة التحقق من التكرار، وأفضل الممارسات ...الخ).

حيث قمنا بكتابة كود نظيف وبعملية ال *Refactoring* عند الحاجة لضمان سهولة فهم الكود.

ومعظم الملفات لا تتجاوز ال 50 سطراً باستثناء الاختبارات.

حيث قمنا باستخدام *Code Climate* لتحقيق ذلك وحصلنا على درجة *A* لتقييم الكود كما يوضح الشكل:



الشكل 14 جودة الكود

13. التطوير المقاد بالسلوك والاختبارات الوظيفية

يعتبر التطوير المقاد بالسلوك *Behavior-Driven Development (BDD)* تطويراً لل *TDD* وهي تعتمد على كتابة اختبارات مقروءة يمكن للزبون فهمها. حيث يعبر كل اختبار عن ميزة ما أو حالة استخدام فهذه الاختبارات تعد هامة جداً فهي دليل واضح على ما تم إنجازه.

الاختبارات الوظيفية *Functional Testing* قد يتم عادةً كتابتها من قبل الزبون أو فريق ال *Quality Assurance* ولكننا قمنا بكتابتها باستخدام ال *BDD*.

حيث قمنا باستخدام لغة *Gherkin* وهي لغة تعتمد على وصف الميزة/قصة المستخدم/حالة الاستخدام بالاعتماد على سيناريوهات كل سيناريو يتألف من *Given-When-Then*.

ولتشغيلها قمنا باستخدام *Cucumber* حيث لكل جملة يتم كتابة التعليمات المقابلة لها وتشغيلها وراء الستار.

مثال عن ذلك ما يلي:

```
Feature: User Registration

  Scenario: Successful Request Submission
    Given A new user visits registration page
    And User fills required info
    When User clicks submit button
    Then Account is created and deactivated
    And Admin receives an email
    And User is redirected to thank you page
```

حيث تشرح إنشاء حساب في منصتنا.

ولكننا لم نقف عند ذلك فقد قمنا بتضمين *Selenium* وهو يجعل الاختبار قابلاً للتنفيذ على المتصفح بشكل مباشر. ويتم إعطاء تعليمات للمتصفح وكأن مستخدماً يقوم باختبار الموقع.

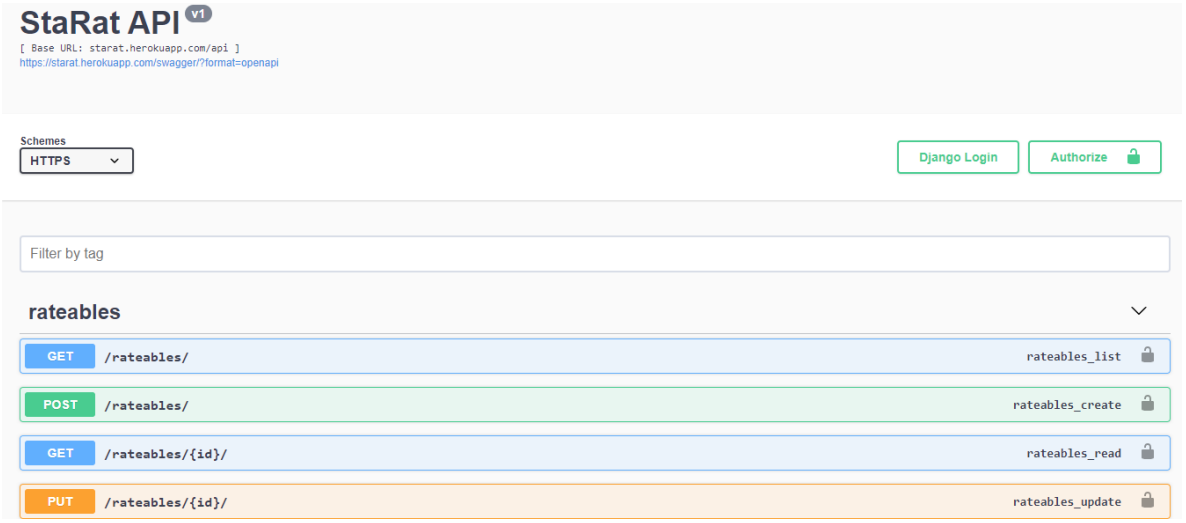
14. توثيق ال *API*

قمنا بإعداد خدمة تقييم خارجية وتصميها ك *Restful Webservice* وقمنا باستخدام *Swagger* لتوثقيها حيث يقوم بعرض جميع ال *endpoints* المتوفرة مع طريقة لتجربتها.

أيضاً قمنا برفع الخدمة على منصة *Heroku* ويمكن الولوج الى *swagger* عن طريق الرابط:

StaRat API

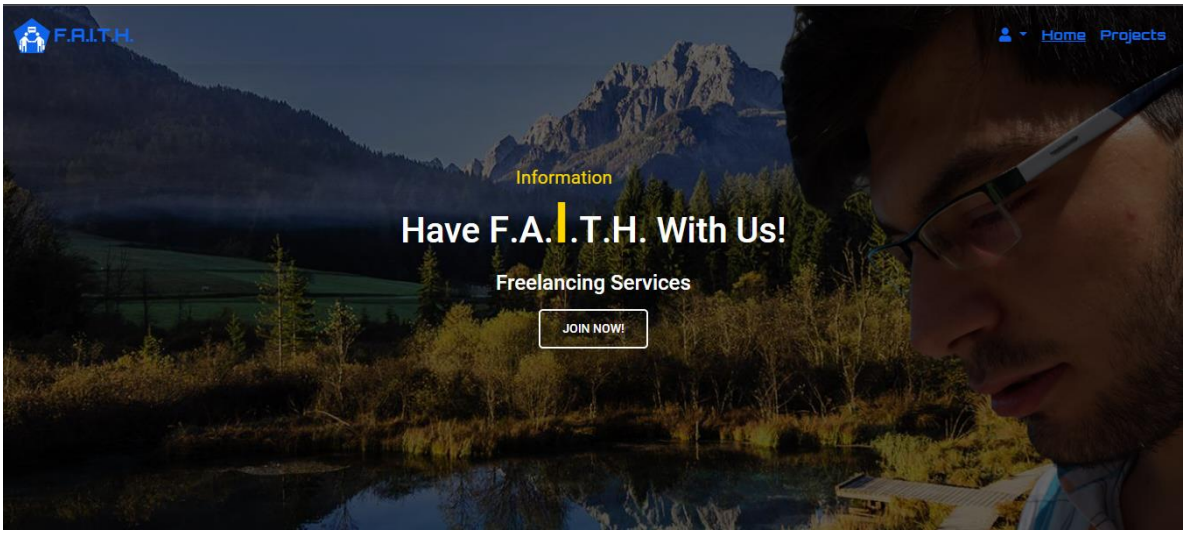
حيث تمثل الصورة التالية الواجهة لها:



الشكل 15 واجهة Swagger

15. المنصة

فيما يلي يمكن رؤية بعض واجهات المنصة.

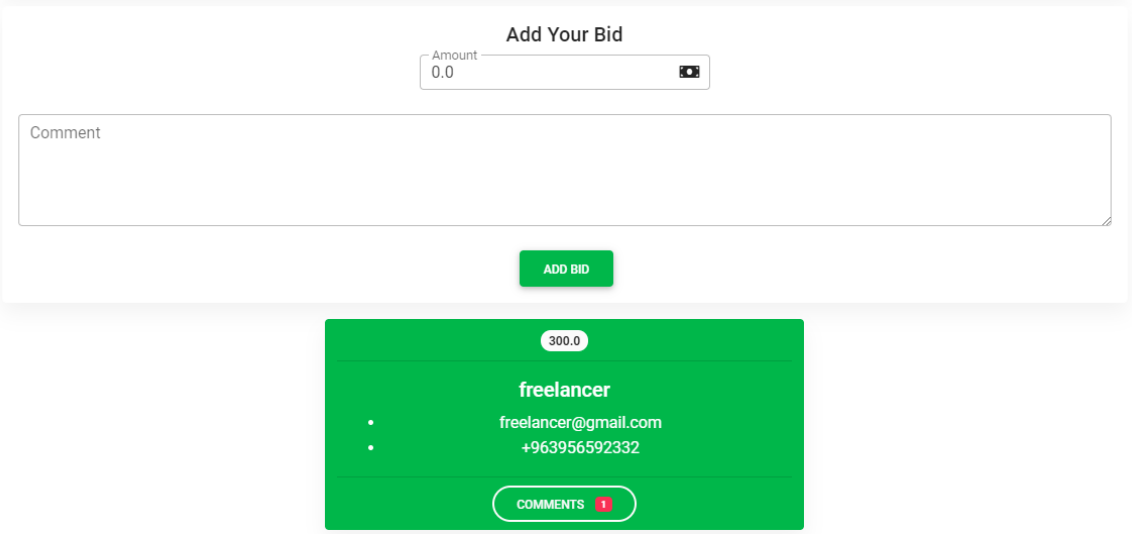


الشكل 16 الواجهة الرئيسية

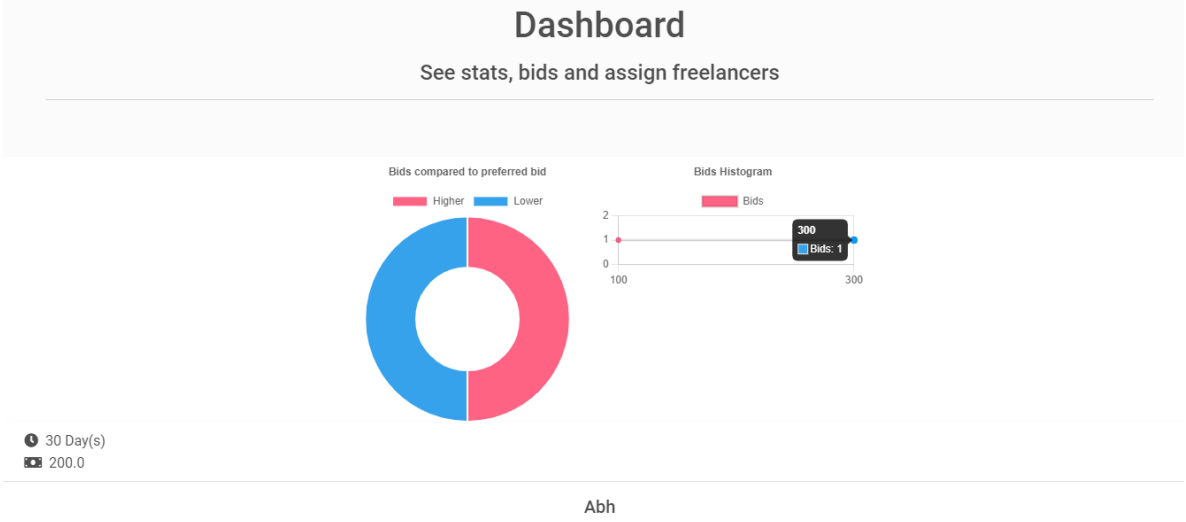
List of users

#	Nickname	Email	First Name	Last Name	Phone Number	Delete	Activate
1	admin	admin@faith.com	Admin	Faith		×	
2	freelancer2	freelancer2@gmail.com	free	lancer	+963956592332	×	✓
3	freelancer	freelancer@gmail.com	free	lancer	+963956592332	×	✓
4	stakeholder	stakeholder@gmail.com	stake	holder	+963956592332	×	✓

الشكل 17 واجهة التحكم بالمستخدمين



الشكل 18 المزايدة



الشكل19 إدارة المشروع

16. خاتمة

نود في نهاية هذا المشروع شكر كل من ساهم في تحقيقه. لقد كانت تجربة رائعة وتعلمنا منها الكثير.

صحيح أن الغاية من المشروع هي تسهيل إيجاد عمل ولكن أحد الغايات الأساسية أيضاً لهذا المشروع هو إيضاح العديد من الممارسات الجيد في هندسة البرمجيات فتعتبر كتابة الاختبارات والتطوير المقاد بالاختبارات مهارات هامة لأي مهندس برمجيات. واستخدام التكامل المستمر يجب أن يكون جزء من أي مشروع في هذا الوقت.

أيضاً الاختبار باستخدام *Selinuim* ليس بالأمر السهل دائماً بسبب طبيعة الواجهات من مؤثرات ومكاتب خارجية لذلك قد نراه يفشل في بعض الأحيان.

ما زال يوجد الكثير من المشاكل والتحسينات التي يمكننا القيام بها لكن الوقت غير كاف وسنحاول الاستمرار في هذا المشروع بعد انتهاءه.

17. مراجع:

[1] pulkitagarwal03pulkit ،"Advantages and disadvantages of Test Driven Development (TDD)،" 2020 .[متصل] .Available: https://www.geeksforgeeks.org/advantages-and-disadvantages-of-test-driven-development-tdd./

[2] M. Warcholinski ،"Test-Driven Development (TDD) cycle .[متصل] "،Available: https://brainhub.eu/blog/test-driven-development-tdd./

[3] M. Fowler ،"Continuous Integration،" 2006 .[متصل] .Available: https://www.martinfowler.com/articles/continuousIntegration.html.

[4] A. Khandelwal ،"Spring Boot 2.0 — Project Structure and Best Practices (Part 2)،" 2019 .[متصل] .Available: https://medium.com/the-resonant-web/spring-boot-2-0-project-structure-and-best-practices-part-2-7137bdcba7d3.