JusticeAI Release Summary

Iteration 4 - Release 1

Team members 2

Project summary 2

Velocity 3

Iteration 4 stories 3

Completed 3

Moved to next sprint 3

Iteration Velocities 3

Future iterations 3

Diagrams 4

Architecture Diagram 4

Domain Model Diagram 5

EER Diagram 6

Technologies and Infrastructure 7

Overview 7

Web Frontend 8

Element UI 8

Vue.js 8

Web Backend 8

Flask 8

SQLAlchemy 9

Marshmallow SQLAlchemy 9

NLP Service 9

NLTK 9

RASA 10

Machine Learning Service 10

Keras 10

Tensorflow 10

Scikit Learn 10

Task Worker 11

Celery 11

Database 11

PostgreSQL 11

Coding Name & Style Conventions 12

Code 12

Testing and Continuous Integration 12

Infrastructure, Continuous Integration & Continuous Delivery 14

Travis 14

Docker & Docker Compose 14

Continuous Delivery to a Production Environment 15

CodeCov 16

Iteration 4 Retrospective 17

Appendix I - Analogical case study 18

Team members

Name and Student ID	GitHub ID	Number of story points that member was an **author** on.
Lance Lafontaine	lancelafontaine	45
Arek Manoukian	arekmano	44
Sylvain Czyzewski	vynny	46
Mihai Damaschin	mihaiqc	24
Samuel Campbell	samuel-campbell	36
Taimoor Rana	taimoorrana1	36
Zhipeng Cai	choitwao	42

Project summary

JusticeAI (ProceZeus) is a web chat bot that aims to facilitate access to judicial proceedings involving specific domains of law. Users will have the ability to converse with the chatbot, describing in detail the situation for which they wish to pursue litigation. The system, which will leverage the power of machine learning and natural language processing, will guide the user through a process wherein they'll be prompted with a series of questions relating to their potential case allowing the system to ultimately determine, based on provincial jurisprudence, whether the user has a valid case worth pursuing in the judicial system. Alternatively, the system may also suggest remedies in lieu of legal action if it is deemed unlikely to be in the user's best interest.

Velocity

For the current iteration, we were able to complete 29 points, out of an estimated 43 user story points. At the start of the iteration, we had received feedback on our user story estimates and adjusted our user story scale, to have 8 points be the biggest-sized user story that we would tackle during an iteration. Additionally, this sprint contained a significant amount of research, which was incorrectly estimated in difficulty. As such, less user stories points were completed.

Iteration 4 stories

Completed

- Choose user type before conversation (5 points)
- Outcome clustering (8 points)
- Fact clustering (8 points)
- Better Classification for Tenant/Landlord Problem (8 points)

Moved to next sprint

- Indicate the understood response from the user's answer (8 points)
- Reprompt on ambiguous answer (8 points)

Iteration Velocities

Iteration 1 (1 story, 3 points)
Iteration 2 (3 stories, 48 points)
Iteration 3 (5 stories, 51 points)
Release 1 (4 stories, 29 points)

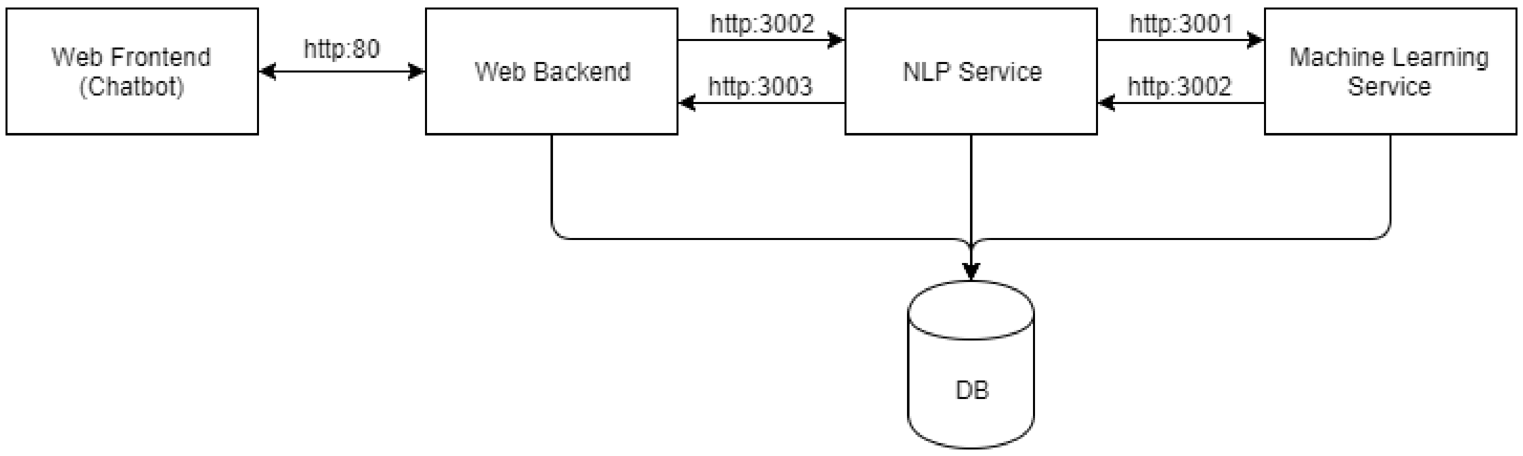
Total: (13 stories, 131 points)

Future iterations

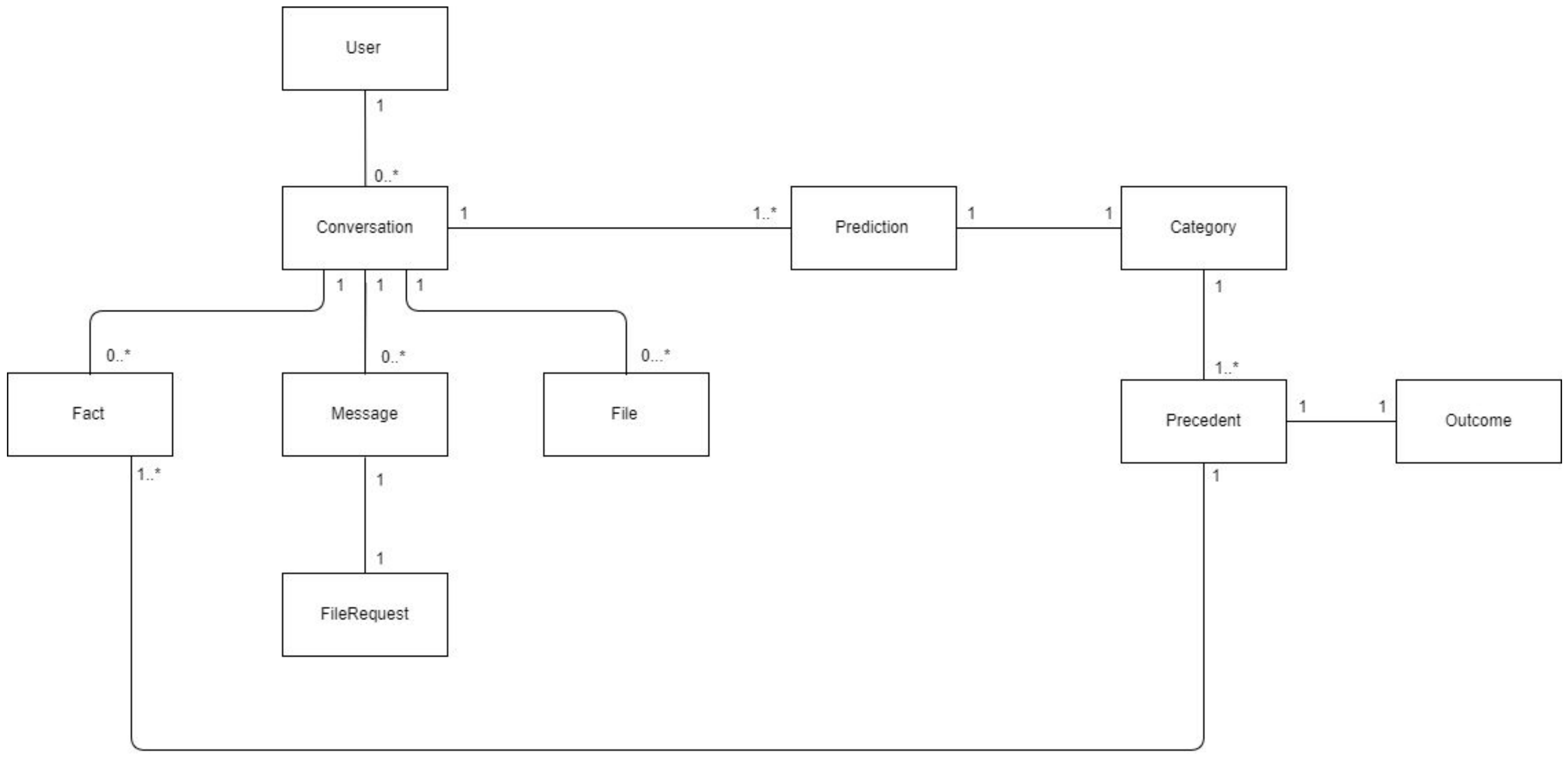
Iteration 5 (6 stories, 40 points)
Iteration 6 (5 stories, 1 Bug, 30 points)

Diagrams

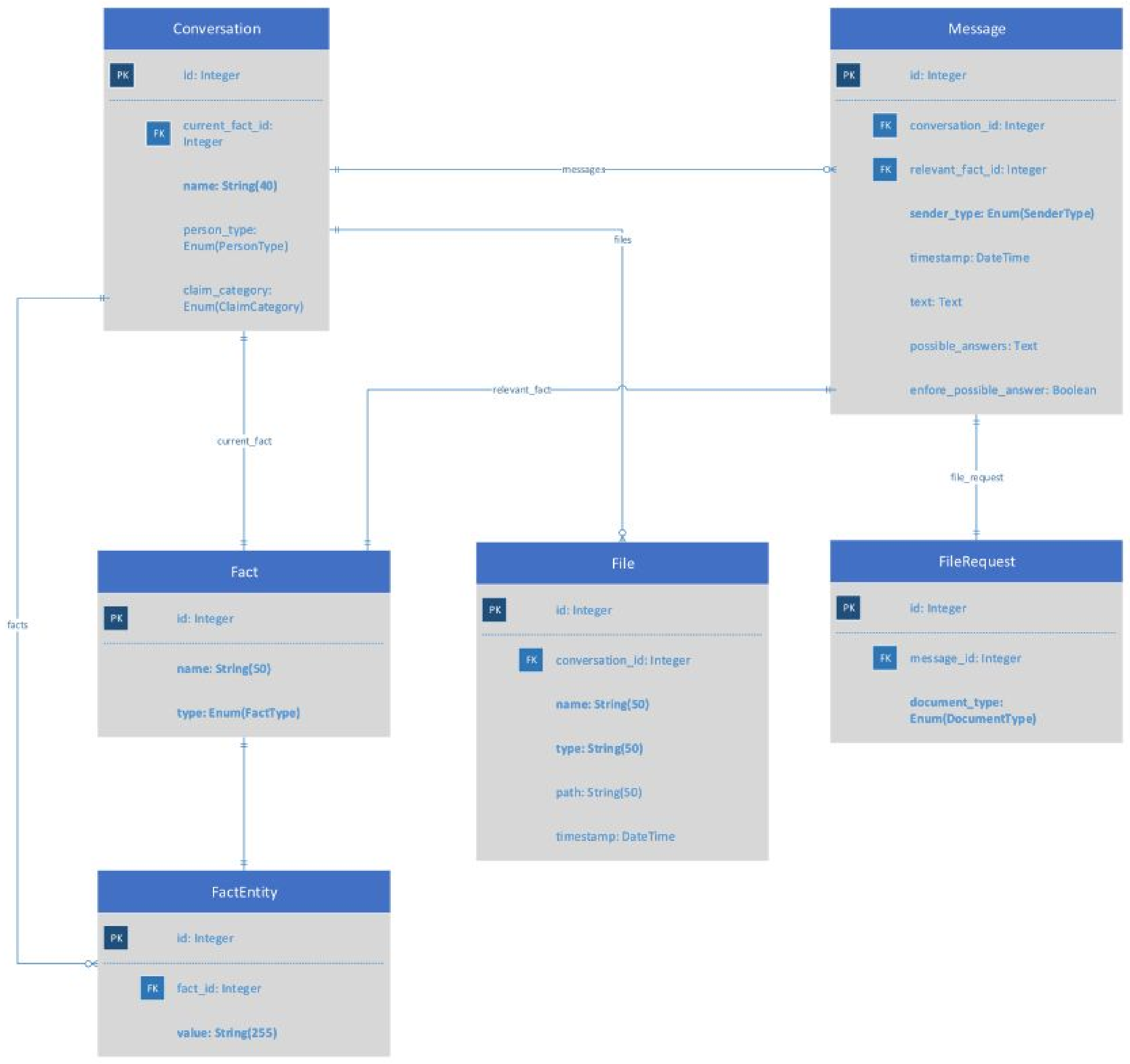
Architecture Diagram



Domain Model Diagram



EER Diagram


Conversation
PK id: Integer
FK current_fact_id: Integer
name: String(40)
person_type: Enum(PersonType)
claim_category: Enum(ClaimCategory)
Message
PK id: Integer
FK conversation_id: Integer
FK relevant_fact_id: Integer
sender_type: Enum(SenderType)
timestamp: DateTime
text: Text
possible_answers: Text
enfore_possible_answer: Boolean
messages
files
relevant_fact
current_fact
file_request
facts
Fact
PK id: Integer
name: String(50)
type: Enum(FactType)
File
PK id: Integer
FK conversation_id: Integer
name: String(50)
type: String(50)
path: String(50)
timestamp: DateTime
FileRequest
PK id: Integer
FK message_id: Integer
document_type: Enum(DocumentType)
FactEntity
PK id: Integer
FK fact_id: Integer
value: String(255)

Technologies and Infrastructure

Overview

Component		
Tool	Language	Purpose
Web Frontend		
Element UI	HTML/CSS/JS	Front end styling and layout
Vue.js	Javascript	Minimal web application framework
Web Backend		
Flask	Python 3	Backend HTTP server
SQLAlchemy	Python 3	SQL ORM
Marshmallow SQLAlchemy	Python 3	SQLAlchemy model to JSON conversion
NLP Service		
NTLK	Python 3	Speech tagging, word tokenizer, text sentiments...
RASA	Python 3	Intent, entity extraction, classification, etc.
Machine Learning Service		
Keras	Python 3	Supervised Learning framework
Tensorflow	Python 3	Neural Network Machine Learning Framework
Scikit Learn	Python 3	Unsupervised Learning Framework
Task Worker		
Celery	Python 3	Performing background tasks

Database		
PostgreSQL	SQL	Data persistence

Web Frontend

Element UI

- Element UI is an open source front end component framework. It contains styling and built-in javascript functionality for various common web components, such as forms, inputs and date pickers, as well as providing a convenient grid system that greatly facilitates web page front end styling and layout.
    - Alternatives: Bootstrap, Foundation Framework, pure.css, skeleton
    - Reason Chosen:
        - Compatible with node8+
        - Works well with Vue.js
        - Well documented and widely used

Vue.js

- Vue.js is an open source front end framework for building single page applications. It leverages component based architecture that allows for the creation of an interactive website. Its primary purpose will be to power the visible portion of the chatbot, displaying messages, sending messages to the server, and prompting the user for various interactions such as answering questions or providing files to use as evidence.
    - Alternatives: AngularJS, Angular 4, React
    - Reason Chosen:
        - Low learning curve
        - High performance
        - Small footprint and minimal API

Web Backend

Flask

- Flask is an open source HTTP backend web framework. It is a light framework, dubbed a 'micro-framework', that only supplies the most essential backend functionalities such as HTTP endpoint handling, form validation, and session handling amongst others. It is easily extendible and has supports a variety of middleware.
    - Alternatives: Django, web2py
    - Reason Chosen:
        - Our backend only requires JSON responses and requires no template generation. The alternatives contain too many features that would never be used.

            - Low learning curve and minimal API

SQLAlchemy

- SQLAlchemy is a python based database ORM that allows for the creation of database tables and relations defined in python classes. It greatly simplifies the task of development as it eliminates the need to write and maintain raw SQL queries. In addition, it supports schema migrations which are necessary for a project where models may change with as requirements are added or removed.
    - Alternatives: PeeWee, Django ORM
    - Reason Chosen:
        - Integrates well with flask
        - Established community and a great deal of well written documentation.

Marshmallow SQLAlchemy

- Marshmallow SQLAlchemy is a python library that allows SQLAlchemy models to be serialized into a JSON format. It serves as an extension of the marshmallow library, which allows for python objects to be serialized into JSON. Marshmallow SQLAlchemy removes the need to write custom serializer classes and instead allows a SQLAlchemy model to be retrieved directly from the database,and automatically serialized into a format that can be used by the front end (JSON).
    - Alternatives: Handwritten serialization classes
    - Reason Chosen:
        - Major convenience over handwritten serializers
        - Support for SQLAlchemy
        - Good documentation

NLP Service

NLTK

- NLTK (Natural Language Toolkit) is a state-of-the-art suite of libraries providing comprehensive documentation with hands-on guides for Natural Language Processing with Python.
    - Alternatives: TextBlob, Stanford CoreNLP, spaCy, Gensim, Pattern, TreeTagger
    - Reason Chosen:
        - Apache license
        - Active community with abundant support
        - Customizable (ie. Allows new tokenizer models & compatible with machine learning libraries.)
        - Capable of supporting French tagging

        - Can be upgraded with add-ons (ie. Stanford libraries written in Java)

RASA

- RASA is a powerful natural language tool divided into two essential components handling natural language understanding (Rasa NLU) and multi-turn dialogues with machine learning (Rasa Core). Rasa NLU combines natural language processing and machine learning to perform intent classification and entity extraction from input text utilizing preconfigured processing pipelines composed of prewritten classifiers. Rasa core provides dialogue management (DM) and natural language generation (NLG), simplifying the task of determining what stage the conversation is at (DM) and being able to generate responses (NLG).
    - Alternatives: wit.ai, dialogflow.com
    - Reason Chosen:
        - Open source (Apache license)
        - Not a SaaS solution
        - Fits well with chatbot use case

Machine Learning Service

Keras

- Keras is a machine learning library that simplifies the use of other more complex machine learning frameworks
    - Reason Chosen:
        - Reduces the maintenance cost associated with our machine learning service
        - Makes machine learning code understandable

Tensorflow

- Tensorflow is a library that allows you to build neural networks, to be used for supervised machine learning
    - Reason Chosen:
        - Extensive documentation
        - Community support available for answering our questions

Scikit Learn

- Scikit learn is a library that simplifies vector computation. It simplifies the use of clustering and unsupervised learning algorithms
    - Reason chosen:
        - Simplifies vector manipulation
        - Standard tool for computation in the python language
        - Extensive documentation available

Task Worker

Celery

- Celery is an open source Python-based task queue. Its purpose is to perform potentially long tasks/jobs that are better done outside of the HTTP request/response cycle, in order to prevent delays from the user's point of view. It uses a message broker, such as RabbitMQ or Redis, to receive messages from components requiring a long task to performed, allowing it to continue execution while the task is performed by Celery.
    - Alternatives: Redis Queue, Taskmaster, Huey
    - Reason Chosen:
        - Most widely used python task queue.
        - Supports RabbitMQ as message broker.

Database

PostgreSQL

- PostgreSQL is an open source database management system that allows data persistence and retrieval via SQL queries. It is supported by SQLAlchemy, allowing it to be used in conjunction with an ORM in a Python project. In addition, it is known to be very efficient when working with complex data sets providing very quick join operations.
    - Alternatives: MySQL/MariaDB, MongoDB, Cassandra
    - Reason Chosen:
        - Relational database
        - Liberal open source license
        - Contains the bigserial data type, ensuring complex objects can be stored in the relational database. This will be useful for our natural language and machine learning components.
        - Traditional SQL syntax (Conforms to SQL standard)

Coding Name & Style Conventions

All Python code throughout the project is checked against PEP 8, the official style guide for Python code. All Javascript code throughout the project is checked against the default ESLint style guide.

These style guides were chosen primarily due to their standardization and ubiquity in their respective communities, which would allow for developers familiar with these language to easily understood our conventions.

Code

File path with clickable GitHub link	Purpose (1 line description)
PrecedentParser	Parses legal precedents and vectorizes facts/outcomes and preprocesses them, to prepare for clustering
HDBSCANWrapper	Wrapper around HDBSCAN clustering algorithm. 1 of 3 tested within the project
RasaClassifier	Class contains intent classifier which, classifies the answer given by the user, in natural language
BackendService/App	API route file that contains all RESTful API endpoints that are needed in Vue.js front-end web client in order to communicate to the system
Synonym Word Fetcher	Word2Vec did not contain all the words found in our precedents and therefore we fetched the word's synonym in order to find a vector for it

Testing and Continuous Integration

List the **5** most important test with links below.

Test File path with clickable GitHub link	What is it testing (1 line description)
Precedent Parser	Tests if facts and decisions are getting extracted from precedents files
RelatedWordFetcher	Test if similar words are properly being parsed from HTML

BackendService/App	Tests that the file upload functionality works correctly
Chat.spec	Tests that the chat user interface correctly communicates with the back-end
Vector	Tests that the precedent fact vectorizer and tokenizer work correctly

Our team has chosen to use the CodeCov service as a code coverage monitoring and reporting tool. As is explained in further detail later in this report, the tool report that 60% of our code is covered by unit tests. We feel as though this is an acceptable level of confidence in our testing strategies given current industry standards.

The remaining 40% of the code is not unit tested either because it would offer little benefit to do so (i.e. the code is trivial), particularly difficult to stub or mock (i.e. some ORM database models) or the code is being extensively tested through other means (i.e. detailed reports and comparisons of data clustering techniques for machine learning).

Infrastructure, Continuous Integration & Continuous Delivery

Travis

Travis is the continuous integration service that is being used to build our Docker images and run our tests with our service containers on every commit. The page that shows the status of every build on all branches can be found here.

Every build in CI consists of building the Docker image for each of our service, and running the tests for that service within a Docker container. Unless every unit test passes, the build fails, which prevents the merging of a pull request into our master branch. This harsh requirement is used to maintain a high code quality in our trunk branch at all times. A healthy trunk branch is of utmost importance in an environment where continuous integration and delivery is being used, as the code that enters the master branch is a direct representation of what's currently in use in our production environment.

Additionally, our build validated that all coding style conventions had been applied. Up until the end of Iteration 4, any style or linting error would cause the build to fail. However, due to a decision reached as an outcome of the Iteration 3 retrospective, code style and linting errors now only cause warnings in the log output of the build. This allowed our team to iterate more quickly given that our build time had reached 20 minutes towards the middle of Iteration 4.

In an effort to reduce times before we are able to merge to master, our Travis configuration was optimized to use a build matrix. This allowed us to parallelize our build on a per-service basis, reducing build times from 20 minutes to approximately 10 minutes which greatly enhanced developer productivity. However, while developing on our local machines, Docker uses layers in order to cache build steps as efficiently as possible, but even with these optimizations, every change to the source code could take 5 to 10 minutes to build. This is a serious time sink and intends to be addressed in issue #156 in Iteration 5.

Docker & Docker Compose

Docker is a container platform that has gained a wide industry adoption in the past several years. We have used Docker as a means to provide a portable, performant and reliable infrastructure and platform for our application. Our team develops all code in a microservice architecture where each service is defined as a separate container. These are intended to operate as self-sufficient applications that communicate with one another through REST APIs via HTTP requests.

The Docker Compose tool was used as a means of coordinating the operation of each service container. It allows for the configuration of specific services in different environments, and provides a means for linking them all in a virtual network.

Containers are used in every stage of the project, spanning development, builds, tests, and deployment. In order to make all of this convenient, we've developed a small script to provide utility commands for development, testing, linting, and more.

Continuous Delivery to a Production Environment

Our team has decided to practice continuous delivery, where every new commit on the master branch is auto-deployed to a production environment. This commitment was taken on with the goal of supplying a live environment for our product owner and stakeholders to assess the progress of our iterations and deliver feedback in a timely manner. We've found that these benefits have been valuable enough to outweigh the slight additional complexity.

Deploying the latest commits to our production environment is possible due to Ansible, a lightweight configuration management and server automation tool. If a build passes on the master build, Ansible is installed and configured with our server credentials on the Travis virtual machine. The build machine then able to issue commands to server to ensure that it pulls the latest changes, spins down the running instance, rebuilds all Docker containers and spins up all instances in a virtual network again.

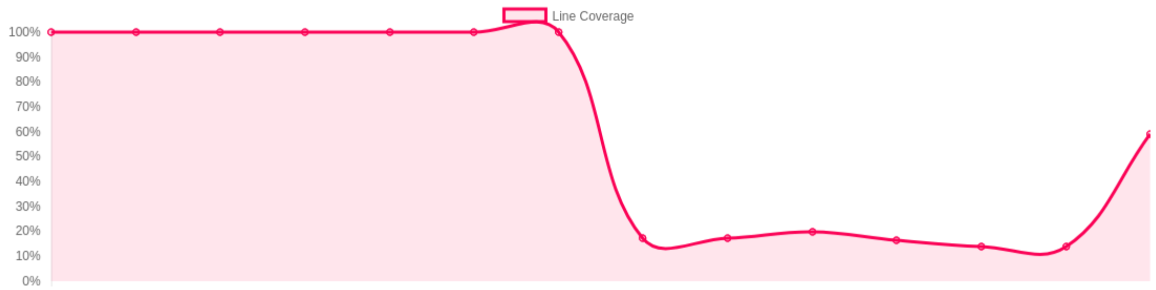
Our production environment is continually accessible at https://capstone.cyberjustice.ca. However, we've password-protected it for the time being, given that it is currently not intended to be released to the public.

Our team has contribution to a significant portion of the infrastructure, including the configuration of a web server and proxy over SSL, password management and configuration, direct binary downloads for applications dependencies, daily automated PostgreSQL database backups, and much more.

CodeCov

To keep track of our Code coverage, we use the free CodeCov service. Every time a new build is completed successfully on Travis CI, the resulting line coverage is uploaded. At the time of this writing, our line coverage is at 60%, with a total of 56 unit tests. The test breakdown per service is shown below.

Service	Number of unit tests	Line coverage %
ML Service	22	67 %
NLP Service	7	39.47%
Web Client	21	100 %
Backend Service	6	46.96 %
Total	56	60%



Line coverage % over the time of JusticeAI.

It can be noticed that there is a sharp drop in our coverage at the beginning of this sprint; this is due to a defect that was fixed, which reported our code coverage as 100% on every build.

Iteration 4 Retrospective

What went well:

- ML cooperation went well
- Information-sharing on research-heavy tasks was frictionless (Issue #105)
- Cross-team collaboration was well-executed

What went less well:

- Not enough physical meetings
- Linting issues slow down development speed
- Meeting note-taking is only done by 1 team member
- We have accumulated significant technical debt, since there was a push for a significant amount of features these 2 sprints. It is slowing down our development speed
- Not enough unit tests are being written per change
- Init scripts are not written for each service

What we will do to fix issues:

- Sub-team meetups on Mondays and Wednesday
- Demote Linting from causing builds to fail to giving warnings only.
- 2 people will be in charge of note-taking going forward
- Have a developer story to remove technical debt (Make builds faster, clean up dependencies, create some form of a caching solution, remove unused files)
- We will enforce unit tests for every PR
- Write setup steps on the README of each respective module

Appendix I - Analogical case study

ProceZeus is a web chatbot that aims to facilitate access to judicial proceedings involving the tenant/landlord domain of law. Users will have the ability to converse with the chatbot, describing in detail the situation for which they wish to pursue litigation. The system, which will leverage the power of machine learning and natural language processing, will guide the user through a process wherein they'll be prompted with a series of questions relating to their potential case allowing the system to ultimately determine, based on provincial jurisprudence, whether the user has a valid case worth pursuing in the judicial system. Alternatively, the system may also suggest remedies in lieu of legal action if it is deemed unlikely to be in the user's best interest.

There exists a system that is similar ProceZeus, founded in Montreal, called Botler.AI[1]. It, however, does not target the same legal domain. The system is a chatbot which specialises in helping users immigrate to Canada under a specific immigration act. The system allows users to communicate with it using natural language and responds in a conversational format.

Unlike BotlerAI, which only assists with acquiring specific resources concerning immigration and determining whether or not a person is eligible for immigration, ProcesZeus attempts to make predictions about the outcome of legal proceedings using previous hearings from its domain. Furthermore, ProcesZeus concentrates on a precise analytical understanding of the English language whereas BotlerAI's current offering is a form generator running through a predetermined dialogue.

After numerous field tests of the BotlerAI platform, we have observed that the product is in its nascent stages as we've been unable to produce a scenario where the application genuinely helps accomplish the goal of aiding in immigration application. While being backed by experts such as Dr. Yoshua Bengio, a key researcher in the field of deep learning, BotlerAI's current offering leaves much to be desired, and insofar has shown to be a series of input form elements bearing the adornment of a dialogue driven, intelligent chat bot. Due to various comments Mr. Amir Moravej's, creator of BotlerAI, has made concerning Donald Trump, its timing in the current political climate and its complete lack of advancement in the past months in combination with its press coverage [2], one might argue that the product resembles more of a political statement than a rigorous engineering challenge.

[1] Botler.AI. (2017). [online]. Available at:https://botler.ai/login [Accessed 9 May 2017].
[2] Lohr, S. (2017). *A Trump Dividend for Canada? Maybe in Its A.I. Industry*. [online] Nytimes.com. Available at: https://www.nytimes.com/2017/05/09/technology/a-trump-dividend-for-canada-maybe-in-its-ai-industry.html [Accessed 9 May 2017].

A competitor more comparable in purpose and implementation to ProceZeus is CaseCrunch. CaseCrunch advertises itself as an AI application that focuses on the prediction of legal decisions with high accuracy. The company summarizes their mission statement as “Use data science to investigate the law, Find truth in law, Build accurate legal prediction systems, Move from niche systems to general legal intelligence, Make systems widely available” [3].

CaseCrunch recently hosted a “Man vs. Machine” challenge, during which their AI was put to the test against 100 lawyers based out of London on 750 cases over a weekend. In this instance, the startup’s prediction accuracy of 86.6% beat the group of layers’ rate of 62.3%. CaseCrunch attributes their win to how their AI can understand “non-legal factors”, although they explicitly mention their machine may not be generally better than humans in other instances[4]. When asked whether lawyers should fear their jobs, the CaseCrunch team responded that the bot may have responded better or worse in a variety of situations, but that it’s possible that this technology could be replace any of the tasks that a junior lawyer would typically accomplish[5].

Although Botler.AI and CaseCrunch share many similarities with ProceZeus, neither tool has been released to the public at large to date. It is clear that the role of artificial intelligence in the law domain is still being defined. There are many questions and ethical dilemmas in this space that have simply not been explored by any company or startup to date. However, this does not preclude us from drawing conclusions for possible sociological and economical issues that may arise within this ever growing sector.

As engineers, it is our responsibility, according to section 2, which treats with the duties and obligations towards the public, an engineer, in article 1, “must respect his obligations towards man and take into account the consequences of the performance of his work on the environment and on the life, health and property of every person”[2](Code of ethics of engineers, 2006) as well as consider, according to article 3, “that certain works are a danger to public safety, he must notify the Ordre des ingénieurs du Québec (Order) or the persons responsible for such work”[6](Code of ethics of engineers, 2006) according to article 3 section 2. As such, we have debated the repercussions of our project and feel strongly that although no official standards for the application of AI to law exists, it will contribute to the greater good as the benefit to society will be quite sizeable due to the monumental strides in legal accessibility the application will provide.

---

[3] CaseCrunch. (2017). [online] Available at: http://www.case-crunch.com/ [Accessed 9 May 2017].
[4] CaseCrunch. (2017). [online] Available at: http://www.case-crunch.com/ [Accessed 9 May 2017].
[5] Cellan-Jones, R. (2017). *The robot lawyers are here - and they’re winning.* [online] bbc.com. Available at: http://www.bbc.com/news/technology-41829534 [Accessed 9 May 2017].
[6] Code of ethics of engineers. (2006). 3rd ed. [ebook] Quebec, p.1. Available at: http://oiq.qc.ca/Documents/DAJ/Lois/TheCodeofEthicsofEngineers.pdf [Accessed 14 Nov. 2017].

Perhaps the most resonating issue common across all projects involving rapidly improving artificial intelligence and its wide scale adoption is the potential long term decrease in available job positions. Specifically pertaining to our scenario, individuals working in the domain of law. As our application (and other programs using similar techniques) develop, so will the interest of investors and companies wishing to cut costs by replacing personnel with computerized solutions. Automation, technology and AI is admittedly an extensive and unavoidable threat to job security and future employment prospects.

It is however important to emphasize that our application is not a lawyer and therefore its judgment may not under any circumstances be considered as legal advice. The team, with the aid of experienced lawyers, will ensure its user's understand this important distinction. We do not intend to replace lawyers, but to act as a facilitator for citizens to inform themselves about the laws that govern their daily lives. We also do not intend to provide a service that is 100% foolproof. Our system provides a disclaimer to explicitly seek the request of a legal professional in order to confirm the information being presented.

Another sociological issue that affects all projects using precedents as data to train an artificial intelligence model is the possible bias that is further propagated by the tool. Given that historical legal cases are shaped not only by the law but by a variety of implicit socioeconomic factors. If ProceZeus were to only be trained on previous legal cases, it would be reinforcing the same biases on its current and future users. This is troublesome primarily because our artificial intelligence should serve all of its users with the same level of service and legal information regardless of previous conditions that may have led to skewed results. This issue will be specifically avoided in our project by incorporating a hardcoded decision tree where if a base set of facts are inferred, the application is able to provide rigid outcomes based on actual laws, as opposed to a simply behaving as a predictive model.

One last ethical concern that we have considered was our data privacy model. BotlerAI does not seem to provide any privacy policy or end user license agreement. Likewise, CaseCrunch is not a publically-accessible service, thereby making our project one of the first applications that we are aware of to request information about a user's existing legal situation. Given that there is the possibility of the user reveal personally-identifiable data to our service, we must ensure that we are handling their user with the utmost discretion. We've already begun the process of drafting privacy policies and an end user license agreement that is presented to the user as a disclaimer once they first begin using the system. Additionally, we have taken measures to implement modern application security practices, limit data access as much as possible, as well as architect our application in order to be able to erase user data after a set period of time. These conclusions were drawn in order to both provide a valuable service to the user as well as protect their integrity and identity.

This analogical case study attempts to draw parallels with the ethical concerns of the Botler.AI and CaseCrunch projects with our own. Although the application of artificial intelligence to the domain of law is a relatively new endeavor, this provides us with the opportunity to explore these contemporary issues in a way that impacts society positively. Some of the corollaries explored were our responsibility to provide accurate legal information without negatively impacting the law profession, our willingness to not propagate any biases that may impact specific individuals negatively, and our commitment to providing a useful service that does not infringe on the user's right to privacy. We feel as though these values are ones that embody the duties of an practicing engineer and influence our decision in the creation of a product that will serve to provide accessible legal information to everyone.