JusticeAI Iteration Summary

Iteration 2

Team Members

Name	Student ID	GitHub ID
Lance Lafontaine	26349188	lancelafontaine
Arek Manoukian	21710389	arekmano
Sylvain Czyzewski	27066333	vynny
Mihai Damaschin	27177895	mihaiqc
Samuel Campbell	26457959	samuel-campbell
Taimoor Rana	26436110	taimoorrana1
Zhipeng Cai	21346482	choitwao

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

Iteration 2 was the first iteration where the team started working on higher value user stories. While work on the infrastructure was still being performed, user stories were being completed. During this sprint, we were able to complete **48/61** story points, giving us a velocity of **48 points**.

The following is a list of user stories that were completed in Iteration 2:

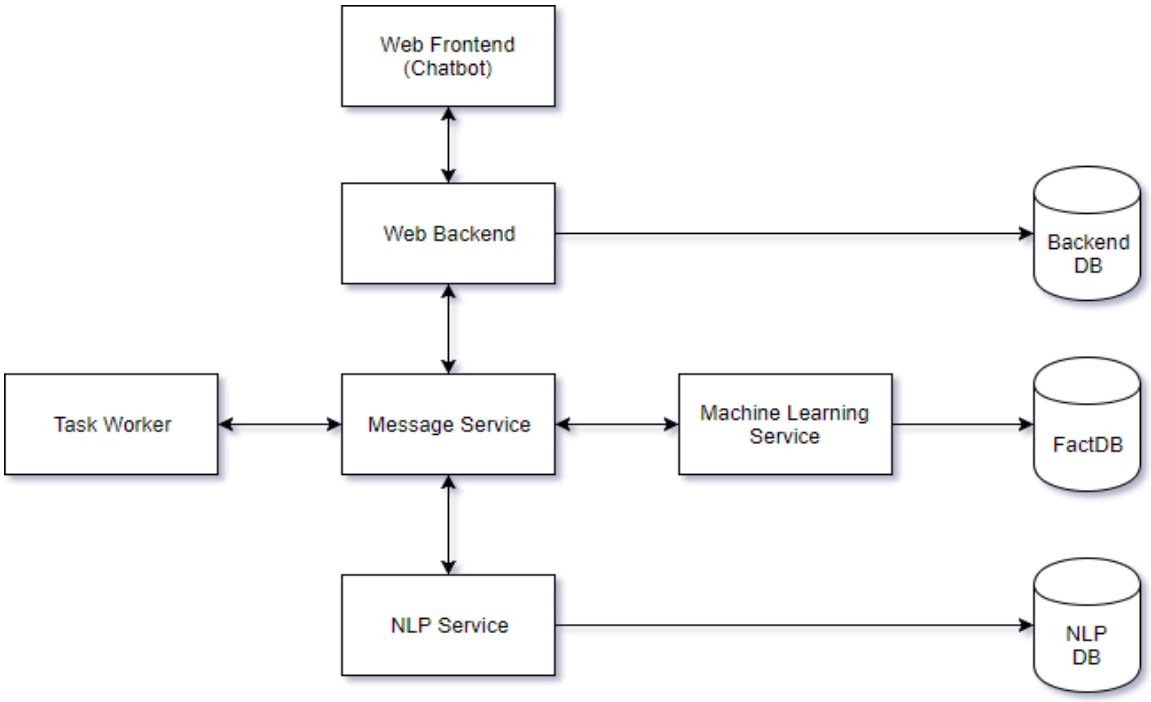
- #20 Claim information gathering (20)
- #8 Chat with System (NLP) (20)
- #4 Access system online (8)

The following is a list of user stories that partially complete and are still awaiting product owner signoff:

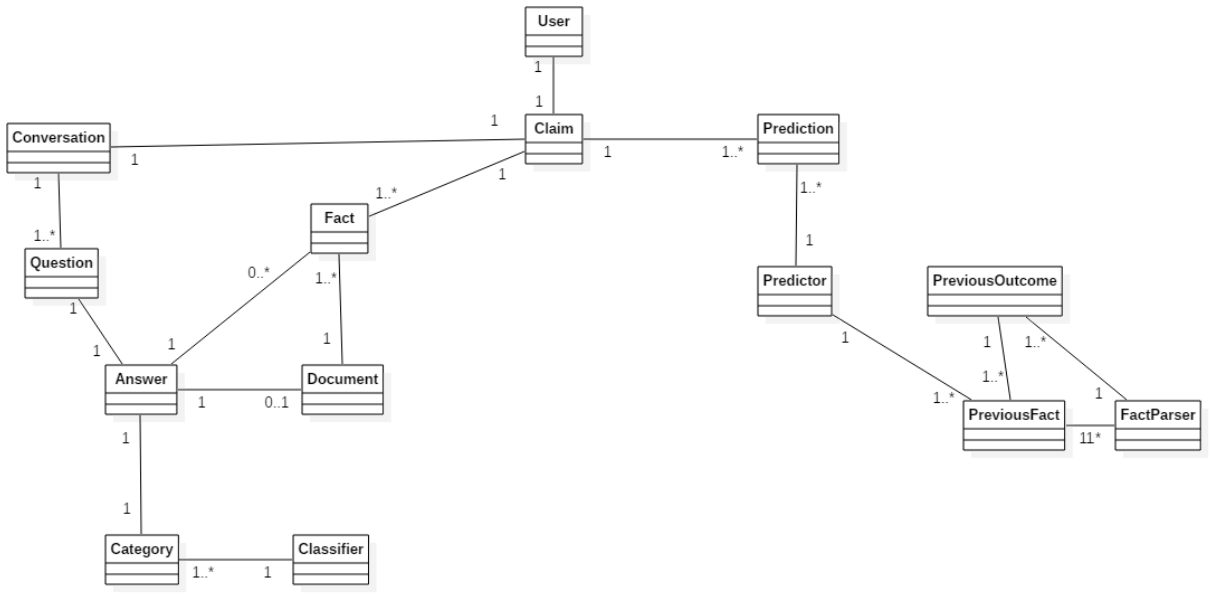
- #15 Claim evidence gathering (13)

Overall Architecture and Class Diagram

Architecture Diagram



Domain Model



Plan for Next Two Iterations

Iteration 3 (43 Points)

Claim evidence gathering (13 points)

Be able to make the distinction between original case and successful appeals (20)

Persist conversation text (5)

End User License Agreement (5)

Iteration 4 (140 Points. Stories will require breaking down and clearer requirements)

Categorical eviction case analysis (40)

Pause conversation (5)

Resume conversation (5)

View previous conversations (5)

Technologies and Infrastructure

Overview

Component		
Tool	Language	Purpose
Web Frontend		
Bootstrap	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
Message Service		
RabbitMQ	Python 3	Distributing messages between components
NLP Service		
NTLK	Python 3	Speech tagging, word tokenizer, text sentiments...
Machine Learning Service		
Caffe2	Python 2.7	Machine Learning framework
Task Worker		
Celery	Python 3	Performing background tasks
Database		
PostgreSQL	SQL	Data persistence

Web Frontend

Bootstrap

- Bootstrap is an open source front end framework developed by Twitter. It contains styling for various common web components, such as forms and inputs, as well as providing a convenient grid system that greatly facilitates web page styling and layout.
    - Alternatives: Foundation Framework, pure.css, skeleton
    - Reason Chosen:
        - Team member's past experiences
        - Industry standard

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.

Message Service

RabbitMQ

- RabbitMQ is a flexible, erlang-based, open source message broker with client implementations available in many popular languages (Python, Java, PHP…). It provides a message queue system that allows separate applications to publish messages and subscribe to numerous queues allowing for message consumption.
    - Alternatives: Celery, Amazon SQS, ZeroMQ
    - Reason Chosen:
        - Open Source
        - Multiple client implementations allow for flexibility in choosing programming languages for other system components

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)

Machine Learning Service

Caffe2

- Caffe2 is an open source machine learning framework, primarily used to construct deep learning networks.
    - Alternatives: Tensorflow, Theano
    - Reason Chosen:
        - Apache 2.0 License

        - Has a collection of pretrained deep learning models, which can then be trained on specific data (Caffe2 Model Zoo).
        - Offers a relatively simple way of manipulating deep learning topologies compared to alternatives.

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)

Continuous Integration Processes and Naming/Coding Conventions

In order to maintain a high standard of code and release quality, our team has established strict development processes that are enforced by Travis CI, a continuous integration service which run on every commit push, pull request and merge.

Pull Request Process

Pull requests can be created after branching from the master branch or by forking the repository.

If you create any branch, the branch name must be in this format, or our continuous integration service will fail the build and you will not be able to merge:

<issue-number>/<short-description-of-issue>

Example: 87/store-user-session-data

In order for a pull request to be merged, it must have at least 2 approvals by reviewers that can validate your changes. Additionally, all continuous integration services must pass.

Git Commit Style

We will be following a similar approach to the Git commit guidelines.
Every commit in our repository must have a well-formatted message, or our continuous integration service will fail your build and you will not be able to merge:

[#<issue-number>] <very-short-summary-of-changes>
- Detailed bullet of important change 1
- Detailed bullet of important change n

Example:
[#12] Added user login feature
- Created user model
- Added OAuth library to our dependencies for authentication
- Created a login page

Naming and Coding Conventions

Our project will use strict continuous integration services to lint our code on every commit. We have chosen to abide by Google's style guide for all languages. This is enforced with the linters listed below during continuous integration.

- Python
  - Style Guide
    - Google Python Style Guide
  - Tools
    - pylint
    - flake8
- HTML/CSS
  - Style Guide
    - Google HTML/CSS Style Guide
  - Tools
    - csslint
    - HTMLBeautify
- Javascript
  - Style Guide
    - Google JavaScript Style Guide
  - Tools
    - jshint
    - eslint

Unit Tests and Code Coverage

We expect every pull request to have unit tests. In order to ensure that unit tests are always being written, we are using CodeCov as a service which monitors for unit test code coverage. If the total code coverage falls below a certain preset percentage, or if that pull request introduces a negative diff greater than 1% in our code coverage, our CI build will fail and you will not be able to merge.

Sprint 2 Retrospective

What went well

- Good in person communication
- UI feedback meeting
- Communication on Slack
- Good learning pace for ML, NLP, and legal domain
- Standup channel on slack helped us communicate daily
- MVP work was well-done

What went less well

- Communication with PO was inconsistent (Unsure what Product owner wants)
- Communication between teams/user stories/components of the system could be improved (API between microservices should be well-defined)
- Didn't run Docker before merging, building dependencies
- Velocity way too high
- Pushes were too late and last minute
- Everything broke at the end of the iteration
- CI/CD process was undefined and undocumented
- Need more people to groom our backlog
    - Story splitting needs to occur
    - Defining user stories with more details
- Presentation was not prepared in advance
- Legal Domain experts weren't receptive to our calls for help

Measures to try out in Iteration 3 (Addressing pitfalls of Iteration 2)

- Improve communication between teams/user stories
    - Define API before everything else in task and push to master
    - Create a micro-service API doc
- Groom the Backlog
    - Break down the stories into 13 point stories (below feature-level threshold)
    - Meetings on Thursday to refine backlog
- Refine CI/CD
    - Write a README about Travis and its imposed rules
    - Make docker cross platform, to prevent odd Docker bugs
    - Docker build should be tested in Travis
- Refine Agile Process
    - Merge to master often (Once a day or every 2 days) with complete modules
    - Smaller pull requests, to prevent last minute midnight code review

    - Scope our work to solving current iteration's problems
    - Add User Interface Mockups and Designs as user story comments
    - Comment on GitHub issues, not in Slack
- Prepare for Sprint Review
    - Meet at 3 PM. Have product owner come in for the Sprint review at 5 PM
    - Code Freeze Thursday at 6PM. Prevents last minute failures
- Obtain legal resources
    - Product Owner will set up an Advisory board of legal experts to assist us