JusticeAI Release Summary

Iteration 8 - Release 2

Team Members

Name and Student ID	GitHub ID	Number of story points that member was an **author** on.
Lance Lafontaine 26349188	lancelafontaine	52
Arek Manoukian 21710389	arekmano	46
Sylvain Czyzewski 27066333	vynny	48
Samuel Campbell 26457959	Samuel-Campbell	29
Taimoor Rana 26436110	taimoorrana1	33
Mihai Damaschin 27177895	mihaiqc	42
Zhipeng Cai 21346482	choitwao	54

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

Release 1

Iteration 1 (1 story, 3 points)

Iteration 2 (3 stories, 48 points)

Iteration 3 (5 stories, 51 points)

Iteration 4 (Release 1 end) (4 stories, 29 points)

Release 1 Total: (13 stories, 131 points over 8 weeks)

Release 2

Iteration 5 (5 stories, 32 points)

This iteration we delivered tangible predictions to users of our system. The primary focus of this sprint was to deliver a system which can predict, based on user input, whether or not a lease can be terminated given certain factors.

Iteration 6, (6 stories, 18 points, 3 bugs)

The primary focus was to create a fully-functional beta sign up system that we can advertise during our guerilla marketing campaign. The process collected the user's questions and email addresses, which adds significant value by informing us of our user's needs and demands. This iteration has slightly lower velocity given the change in direction for this sprint (including the creation of a beta page) as well as a reduced time availability during the final exam and holiday seasons.

Iteration 7, (5 stories, 27 points, 1 bug)

The primary focus of Iteration 7 was to complete the beta sign up version of the application. This milestone allows a user to sign up to the website from whom we can gather data about common landlord/tenant concerns from various social media outlets. The beta was given a high priority by the product owner as they wanted to present and advertise the system that we are building.

Iteration 8 (Release 2 End), (7 stories, 35 points, 5 bugs)

The primary focus of Iteration 8 was adding functionality that allows the system to give an estimate of the amount of money a user may receive if they took someone to court over unpaid rent. Furthermore, while not being displayed, we were able to install similarity between precedents' functionality. Additionally, FAQ questions from the LikeHome website were added for extra value to the user, providing terse answers for common questions such as how to sublet an apartment and send a demand letter to a landlord.

Release 2 Total: (23 stories, 112 points, 9 bugs over 8 weeks)

Future Plan

Iteration 9, (8 stories, 57 points)

The primary focus of this iteration is to predict whether a landlord can retake a dwelling. and parsing a user's lease information through OCR. While adding these features are important, refactoring the NLP backend is also a high priority in order to allowed increased scalability when adding additional classification intents. Furthermore, we will be investigating how we can use the new data provided by our P.O. to add more user value in our application.

Iteration 10, (7 stories, 43 points)

The primary focus of this iteration would be to predict if a tenant can be expelled from their dwelling. Another feature to be added will be to autocomplete user information when the user requires a demand letter(This information was parsed from the user's lease in Iteration 9). The remaining focus would be to eliminate high/medium priority bugs in order to prepare for the final release.

Metrics

Machine Learning

Support Vector Machine Classifier

The support vector machine classifier is used to determine if an outcome occurs given a set of facts. So far, the system supports all the outcomes listed in the table below. This technique is only good for predicting boolean type outcomes (e.i.: the lease is terminated == True, the lease is terminated == False). Certain outcomes are numerical types and must therefore use a regression technique in order to create a prediction.

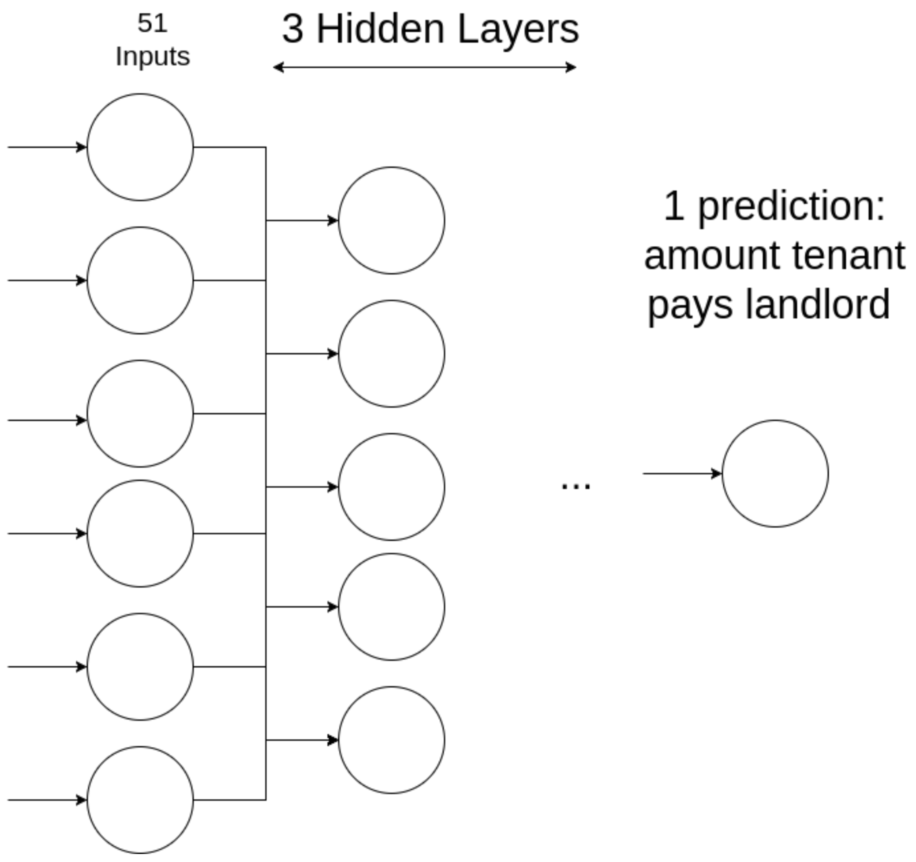
The regression model will only be used for an outcome if the outcome in question returned a True value. Finally, the SVM is used as a filtering layer before the facts can be further used by the regressor.

Values in red indicate bad results as indicated by the recall score.

Outcomes	**Accuracy**	**Precision**	**Recall**	**F1**
additional_indemnity_date	0.88	0.91, 0.84	0.84, 0.92	0.88, 0.88
additional_indemnity_money	0.84	0.90, 0.73	0.87, 0.77	0.88, 0.75
declares_housing_inhabitable	0.99	0.99, 0	1, 0	0.99, 0
declares_resiliation_is_correct	0.92	0.96, 0.69	0.95, 0.73	0.95, 0.71
orders_expulsion	0.96	0.95, 0.97	0.98, 0.93	0.96, 0.95
orders_immediate_execution	0.88	0.93, 0.76	0.88, 0.86	0.91, 0.80
orders_resiliation	0.96	0.95, 0.97	0.98, 0.94	0.97, 0.96
orders_tenant_pay_first_of_month	0.99	0.99, 0	1, 0	0.99, 0
rejects_landlord_demand	0.97	0.97, 0.84	0.99, 0.40	0.98, 0.54
rejects_tenant_demand	0.97	0.97, 0.69	0.99, 0.37	0.98, 0.48
tenant_ordered_to_pay_landlord	0.70	0.82, 0.59	0.64, 0.78	0.72, 0.67
tenant_ordered_to_pay_landlord_legal_fees	0.75	0.88, 0.60	0.73, 0.81	0.80, 0.69

Deep Learning

The deep learning layer of our prediction pipeline is used to create predictions on continuous numerical values. The input data used by this training model is biased and therefore the model needs a specific set of facts in order to create reliable predictions. The SVM discussed above fulfills this role. So far this layer only covers 1 category (tenant_ordered_to_pay_landlord).



The observed problem with this current implementation is the multiplicative error cause by the SVM layer. Further review on our data points collection must be conducted in order to resolve the problem.

tenant_ordered_to_pay_landlord:

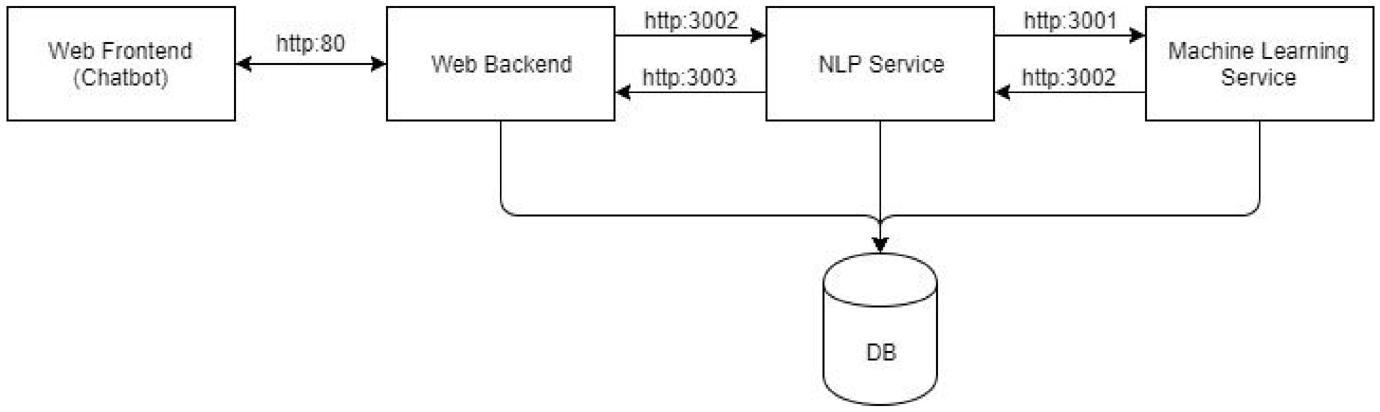
Standardized: -26.38 (3.94) MSE, model: intermed_model, epochs: 100
Standardized: -20.34 (2.31) MSE, model: adv_model, epochs: 100
Standardized: -19.19 (2.00) MSE, model: vadv_model, epochs: 100

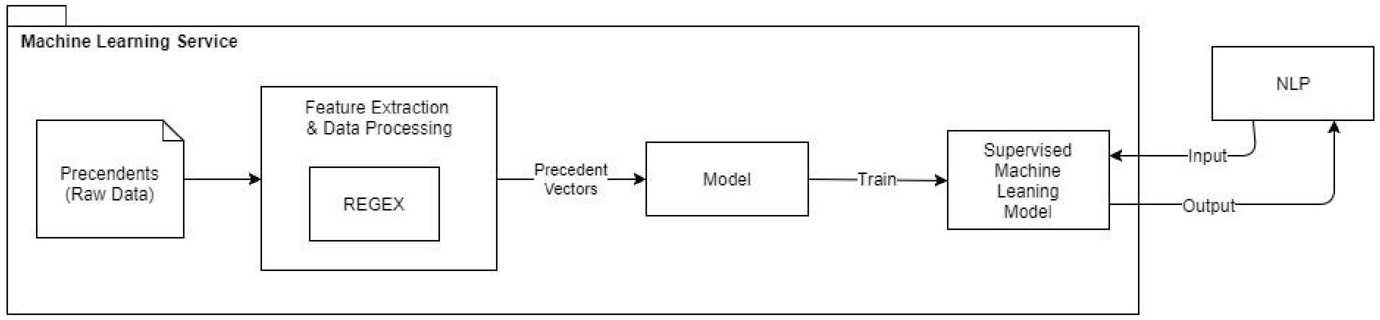
Overall Architecture

Tooling and Infrastructure

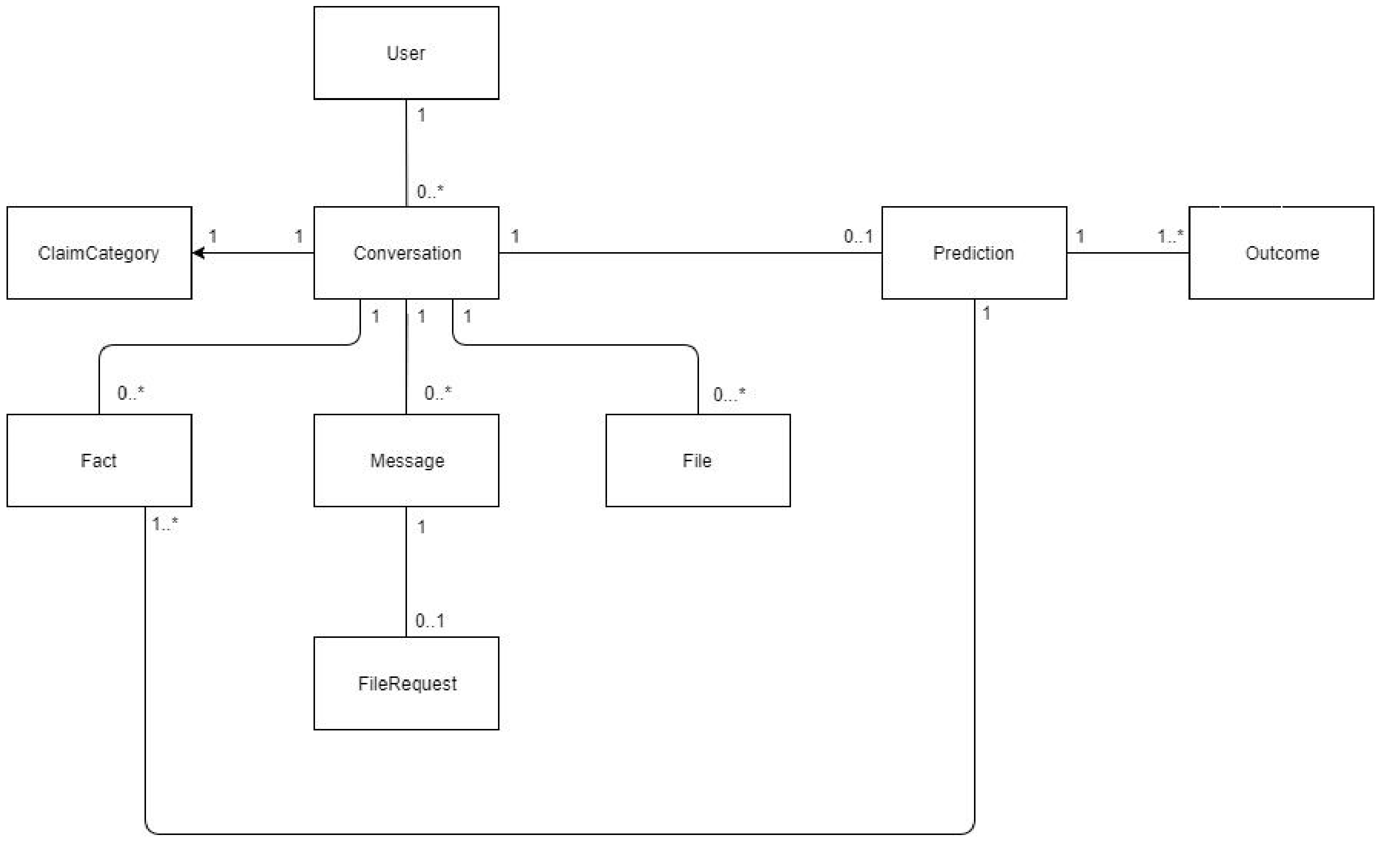
Component		
Tool	Language	Purpose
Web Frontend		
Element UI	HTML/CSS/JS	Front end styling and layout
Vue.js	Javascript	Web application framework
Web Backend		
Flask	Python 3	Backend HTTP server
Flask SQLAlchemy	Python 3	SQL ORM
Flask Marshmallow	Python 3	SQLAlchemy model to JSON conversion
NLP Service		
RASA	Python 3	Intent, entity extraction, classification, etc.
SciKit-Learn	Python 3	Text vectorization, covariance estimation for outlier detection
Machine Learning Service		
Keras	Python 3	Supervised Learning
Tensorflow	Python 3	Neural Network Machine Learning
SciKit-Learn	Python 3	Supervised Learning
Database		
PostgreSQL	SQL	Data persistence

Architecture

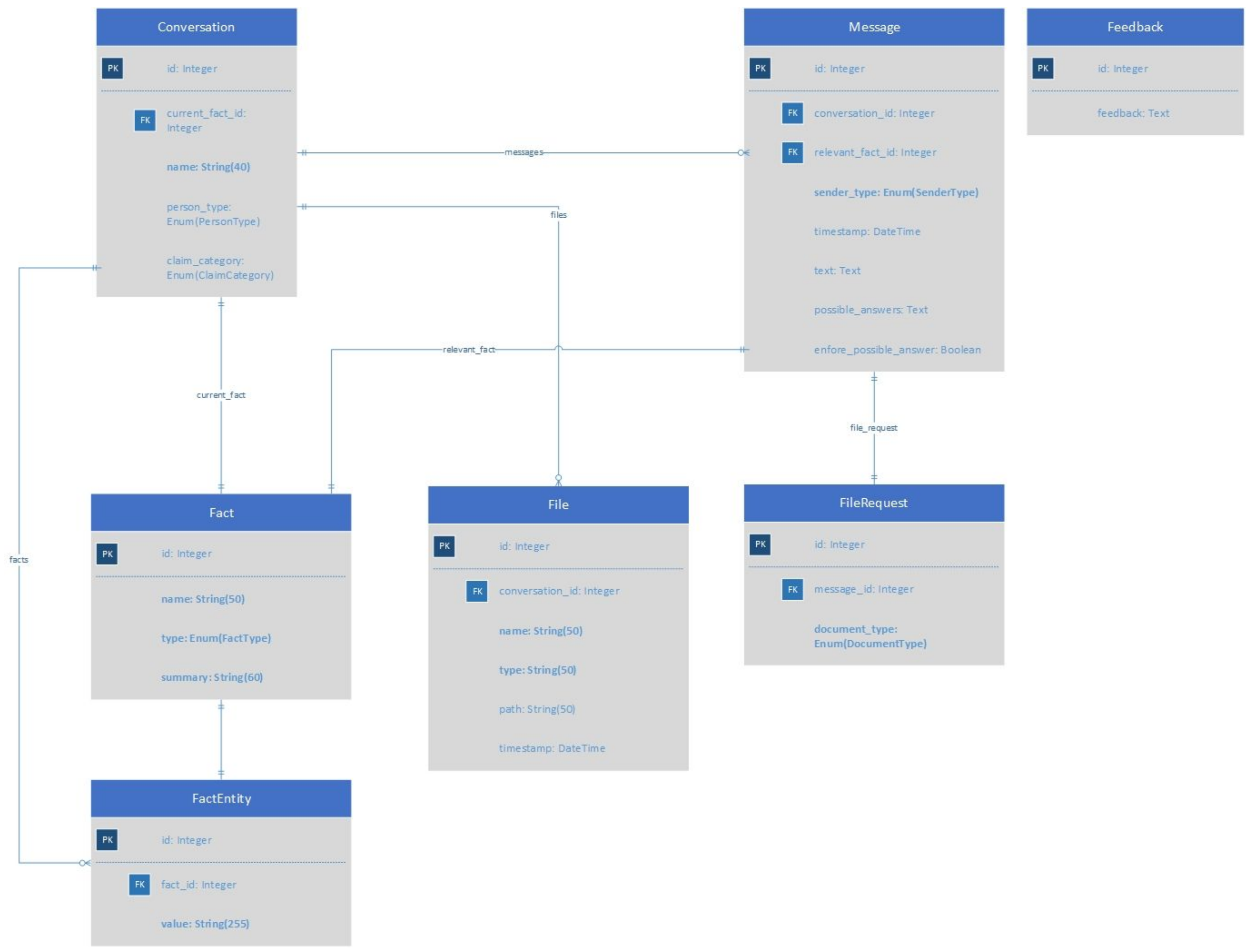




Domain Model



ER Diagram



Naming and Code 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

Important Source Code

File Path	Purpose
src/ml_service/model_training/classifier/multi_class_svm.py	Create a multi output classifier model for machine learning predictions.
src/ml_service/feature_extraction/post_processing/regex/regex_entity_extraction.py	Entity extractor from raw data
src/nlp_service/rasa/rasa_classifier.py	Core of the RASA utilization methods
src/nlp_service/rasa/text	Directory containing the data RASA is trained with
src/nlp_service/controllers/nlp_controller.py	Central point of interaction in the nlp_service with RASA classifier and other services

Testing and Continuous Integration

Important Tests

Test File Path	What is it testing
src/ml_service/feature_extraction/post_processing/regex/regex_entity_extraction_test.py	Tests the functionality of the entity extraction from the unstructured data.
src/ml_service/web/ml_controller_test.py	Tests the ML endpoint
src/nlp_service/rasa/nlp_integration_test.py	Tests core functionality of intent classification with its data
src/nlp_service/rasa/rasa_test.py	Tests whether the nlp intent classification by rasa is functioning
src/web_client/test/unit/specs/chat.spec.js	Tests chat interaction

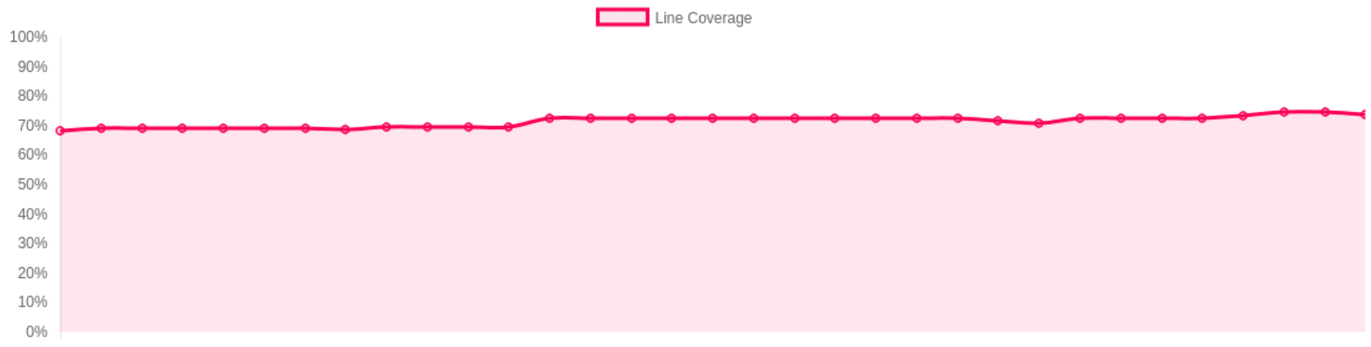
Continuous Integration and Deployment

We use TravisCI for continuous integration, which can be viewed here. No major architecture or technology changes related to continuous integration or deployment have occurred since Release 1. However, we have migrated our production platform to use an Azure VPS as opposed to a VM provided by our stakeholder. This simple change in hosting has reduced production HTTP request times from approximately 1 seconds to 250 milliseconds. Our public production site is now available at https://procezeus.ca. We've also introduced healthcheck endpoints to our services which are being monitored with UptimeRobot every 5 minutes, resulting in a notification in our team channel if ever any service is unresponsive.

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 76.63%, with a total of 124 unit tests. The test breakdown per service is shown below.

Service	Number of unit tests [Rel 1 #]	Line coverage % [Release 1 %]
ML Service	51 [22]	79.76% [67.00%]
NLP Service	30 [7]	66.38% [39.47%]
Web Client	27 [21]	97.39% [100.00%]
Backend Service	16 [6]	65.80% [46.96%]
Total	124 [56]	76.63% [60%]



Line coverage % over the time of JusticeAI during Release 2 development.

Sprint Retrospective (Iteration 8)

What went well

- Lots of development went on (very productive)
- RASA json tool worked properly and greatly speed up development for a low time investment
- NLP worked really efficiently and teamwork is going well
- The main application looks beautiful
- We ended up finishing the stories despite being constrained on time (NLP was quick)
- ML pair programming went well

What went less well

- ML was pushed to the last minute; should have been split up beforehand which forced NLP to work extremely fast and under a lot of stress
- Predictions weren't great for ML using SVR
- Still no fact meeting, everyone needs to understand what each fact means to properly use it within the code
- Interaction between groups (NLP and ML) is minimal
- ML isn't working as much as a team as NLP is
- People don't describe what they're doing and what's been done in their standups in detail

Measures to implement in iteration 9

- If you create an ML binary, upload it yourself via FTP instead of relying on Lance to do it
- Update individual service README.md as technical docs to help team understand big picture
- Adds facts to wiki/README to help team understand them better as well as a short description and an example
- No more UI stories, push back to final sprint, they are slowing down the development of the core functionality of the project
- More support for ML meetings and more ML documentation
- Ask for help when help is needed (or when it's too much) from other individuals
- Crash course on what happens from both main teams (NLP and ML)
- Organize in person ML meetings via Doodle
- Be more descriptive of changes and big picture in PRs