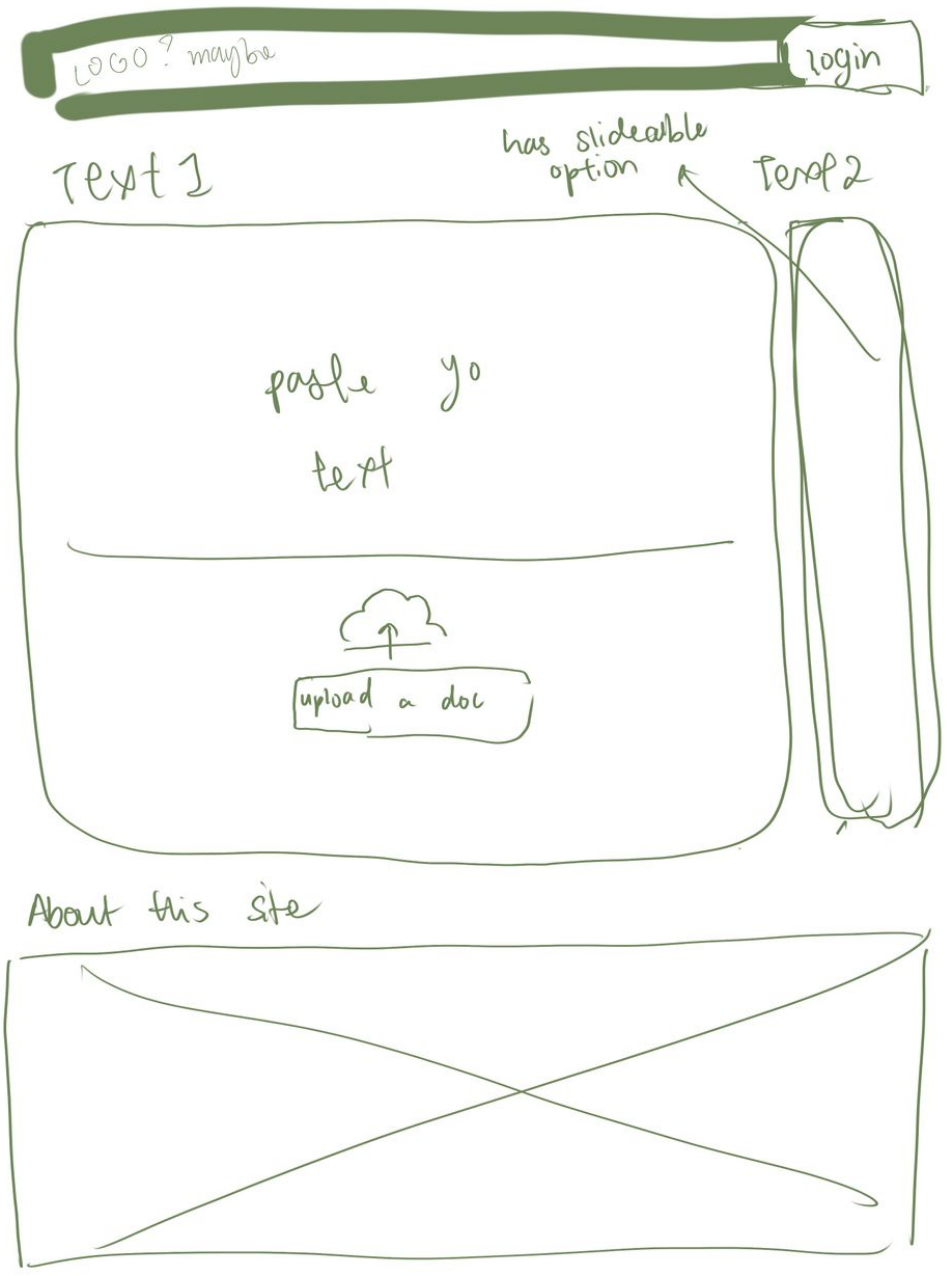
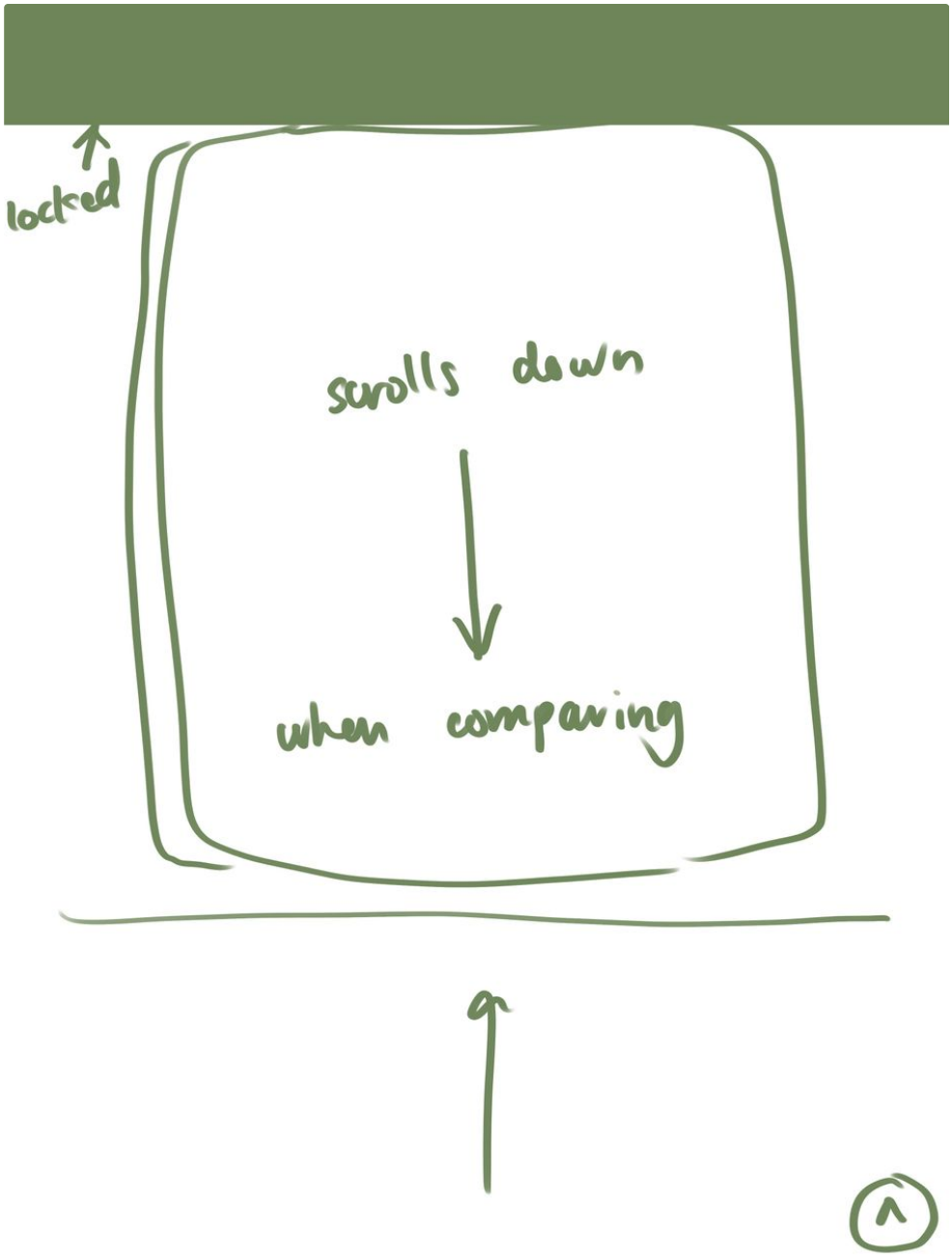
REROLL INC

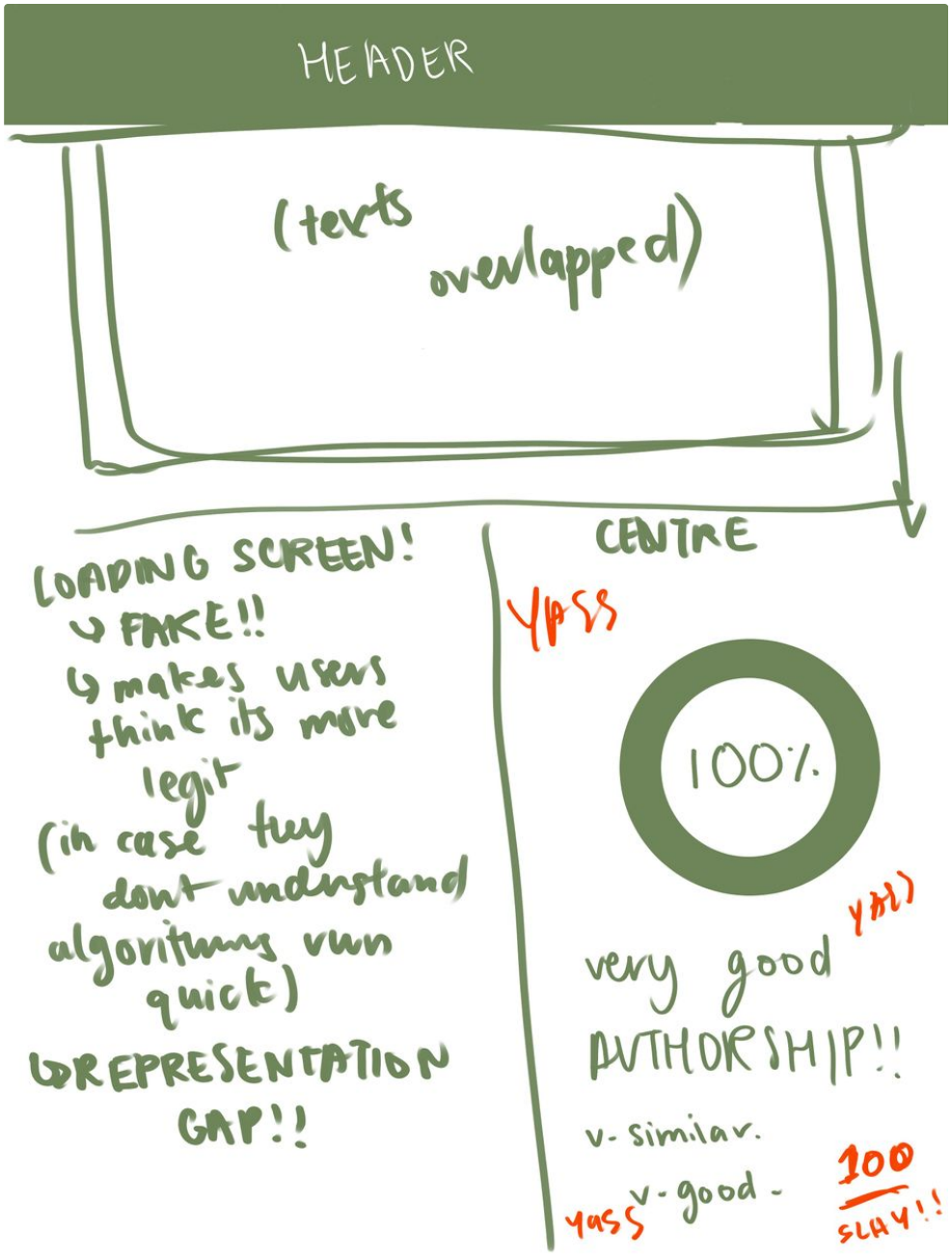
Prototypes

- Rough Prototyping
- Low-Fidelity Prototypes
- High-Fidelity Prototype

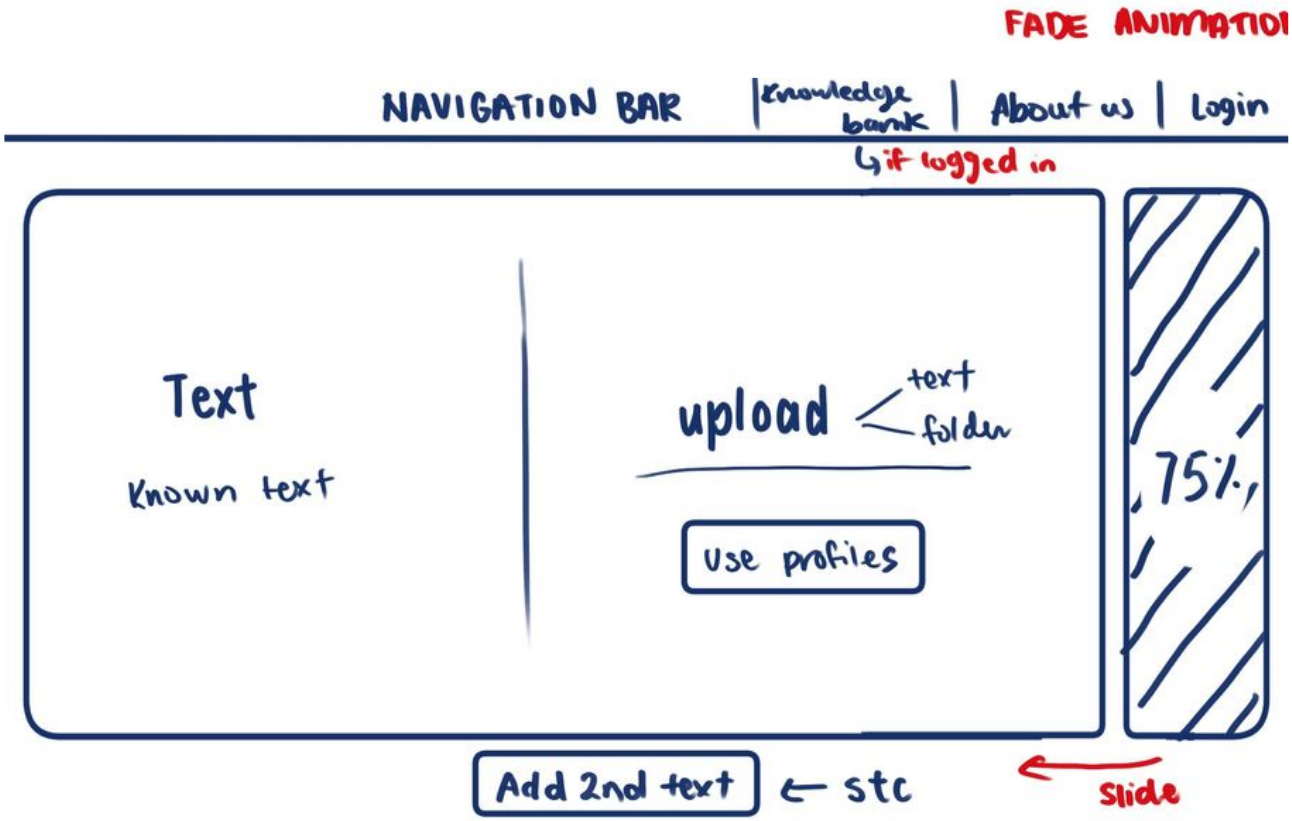
Rough Prototyping




locked
scrolls down
when comparing


HEADER
(texts overlapped)
LOADING SCREEN!
↳ FAKE!!
↳ makes users think its more legit
(in case they dont understand algorithms run quick)
↳REPRESENTATION GAP!!
CENTRE
YASS
100%
very good
AUTHORSHIP!!
v. similar.
yass v. good
100
SLAY!!

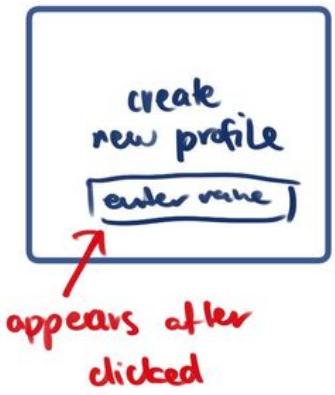

NAVIGATION BAR
WEBSITE TITLE
one line website introduction
Learn more . . .
ENTER AS GUEST
Log in


FADE ANIMATION
NAVIGATION BAR
knowledge bank
About us
Login
if logged in
Text
Known text
upload
text
folder
Use profiles
75%
Add 2nd text
etc
Slide

if user isn't logged in and wants to use profiles

else:
open
Knowledge bank

KNOWLEDGE BANK



appears after
clicked

About

Text

KNOWLEDGE BANK - PROFILE NAME

Instructions

- ~~~
- ~~~
- ~~~
- ~~~

UPLOAD

- .docx or pdf files only

- if already uploaded.

KNOWLEDGE BANK - PROFILE NAME

file name	date added	file type
hello_world.py	01/01/0101	doc

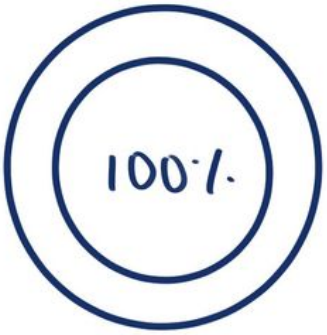
add more

+

Analysing Text...

→ Loading animation

RESULTS

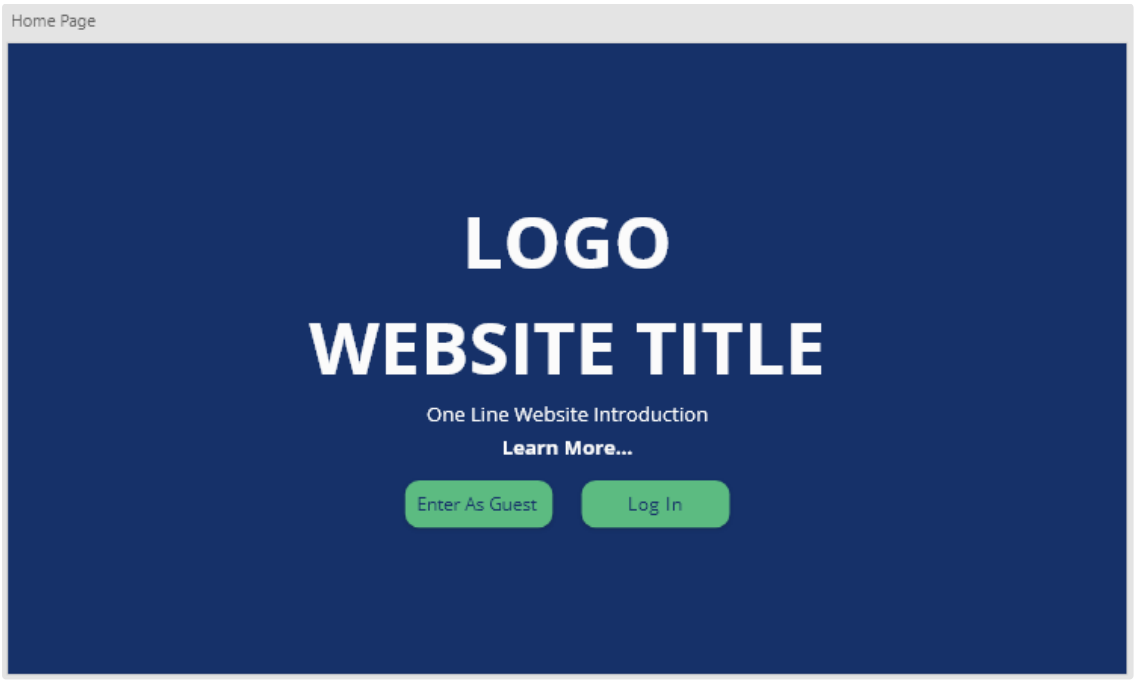


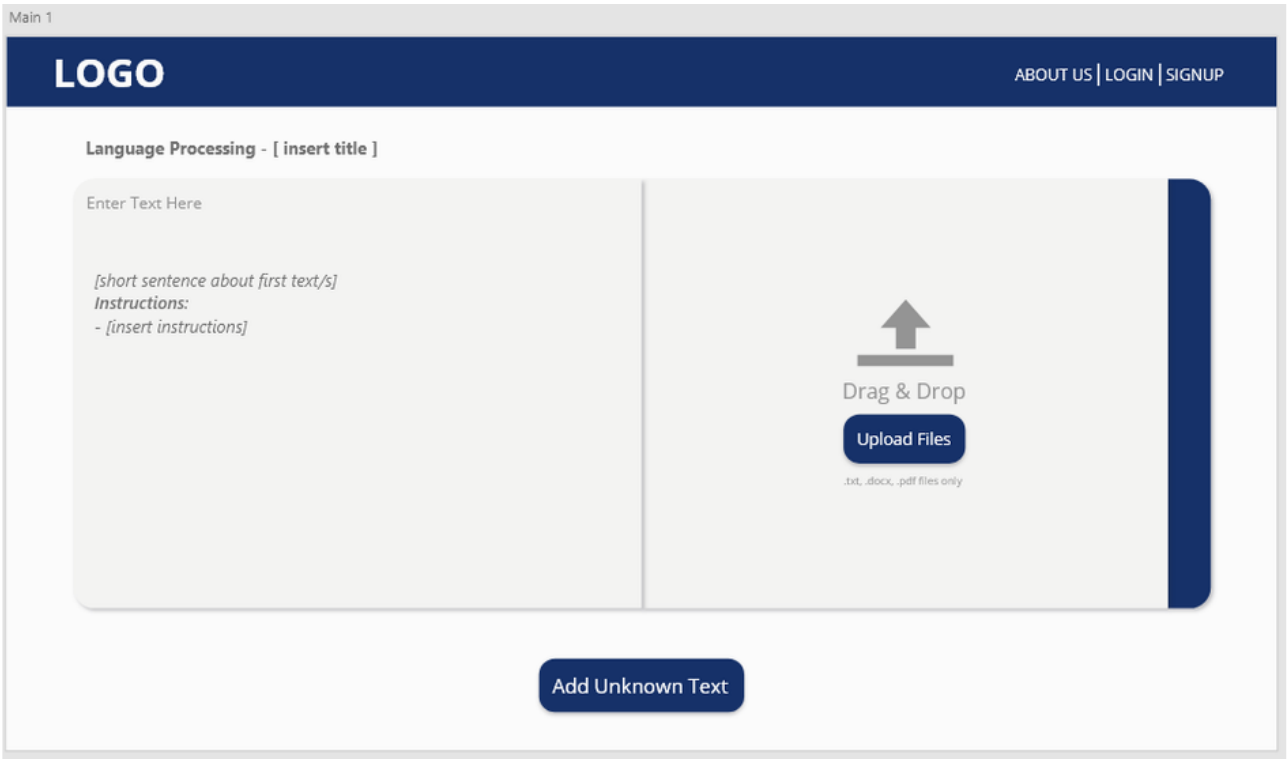
Basic Information

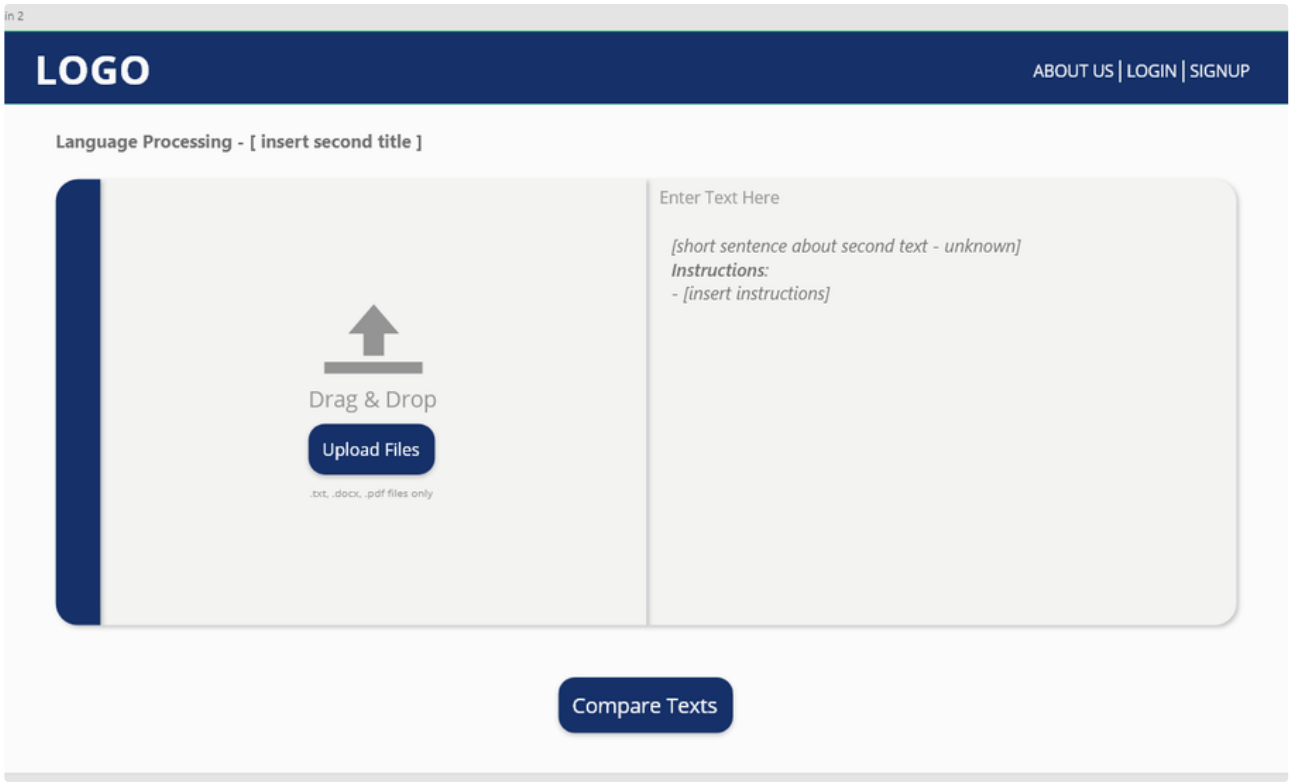
See detailed

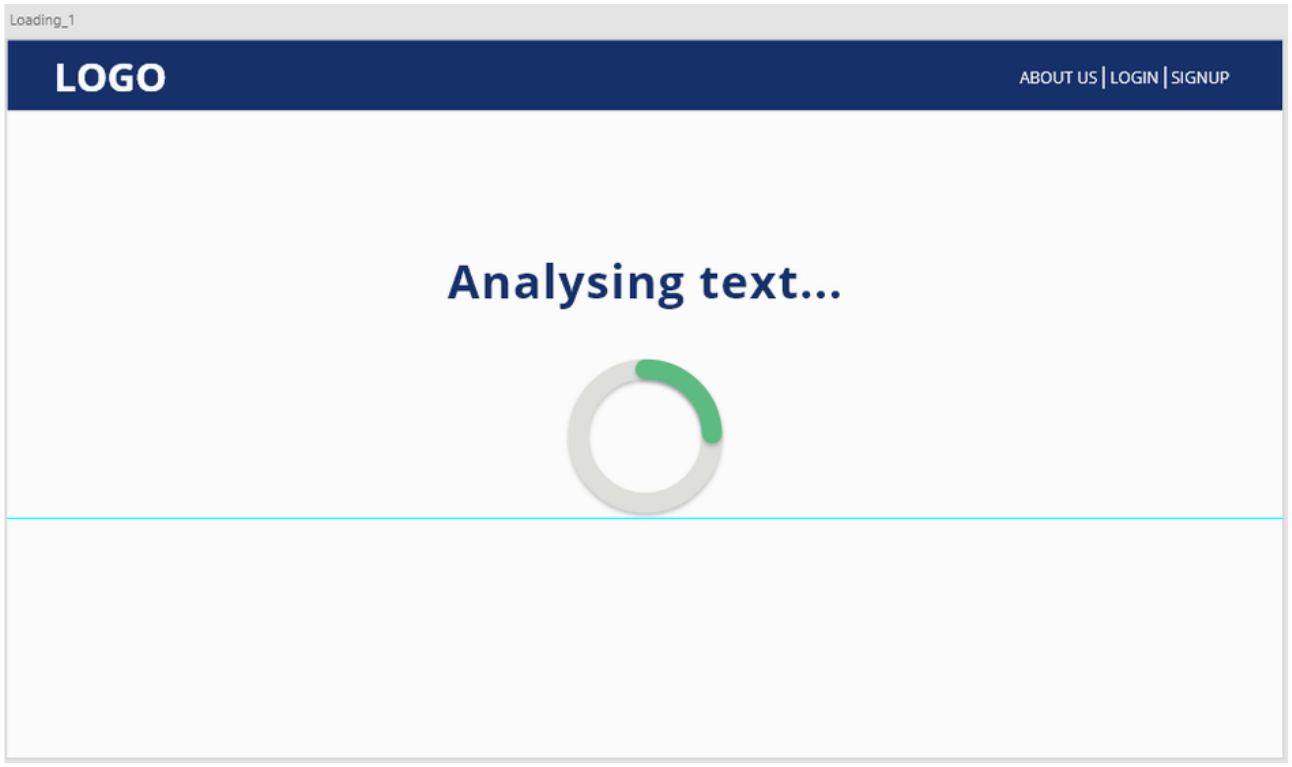
High-Fidelity Prototype

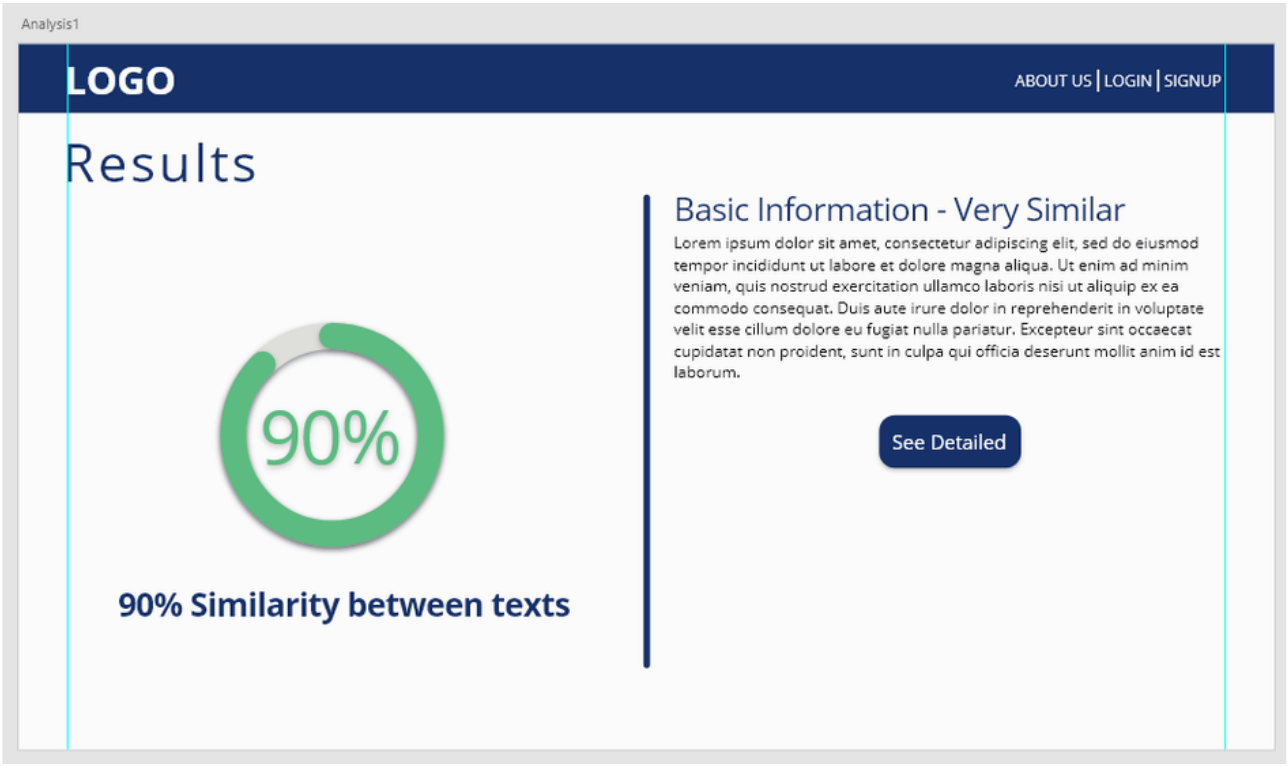
Changes from Eduardo's feedback have been implemented



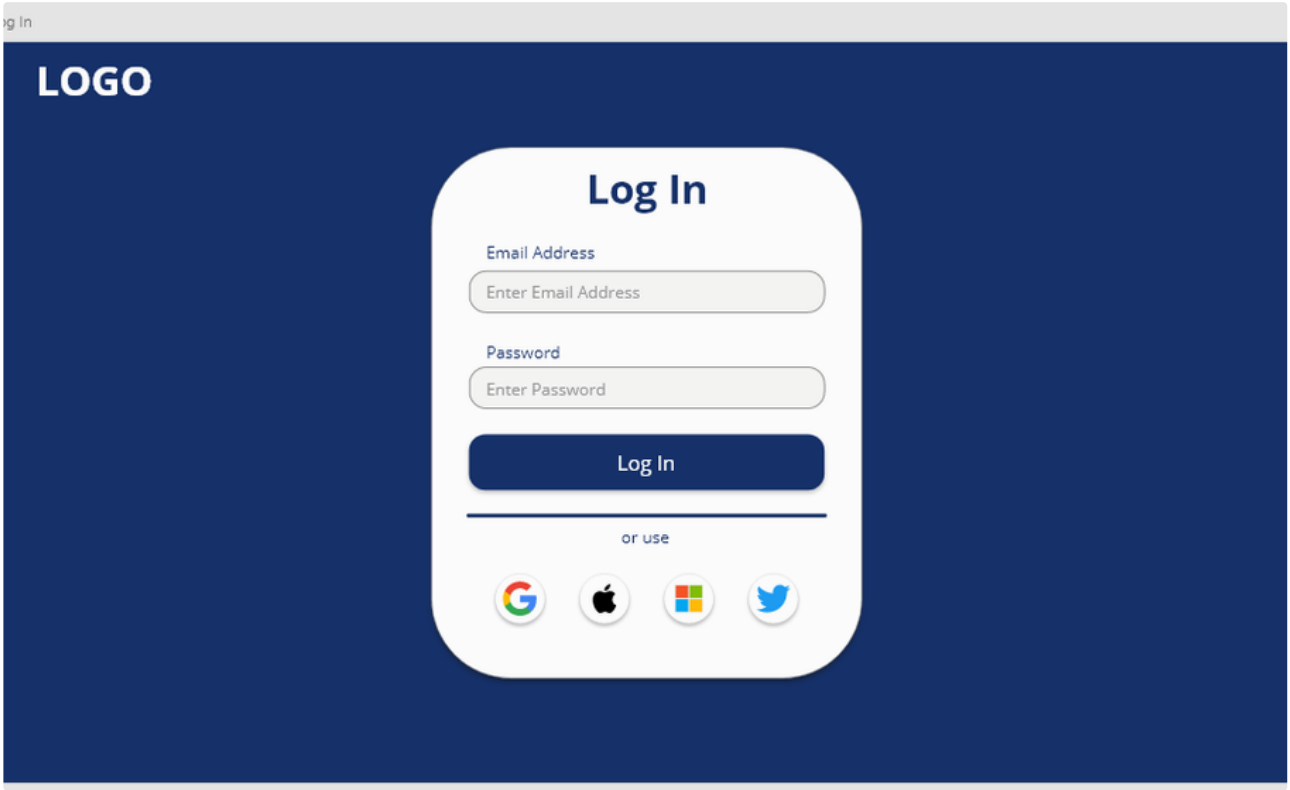



in 2
LOGO
ABOUT US | LOGIN | SIGNUP
Language Processing - [insert second title]
Drag & Drop
Upload Files
.txt, .docx, .pdf files only
Enter Text Here
[short sentence about second text - unknown]
Instructions:
- [insert instructions]
Compare Texts


Loading_1
LOGO
ABOUT US | LOGIN | SIGNUP
Analysing text...



Extensions for future version:



1 Sprint 1

Sprint **Planning**: Sprint 1 - Planning

Sprint **Review**: Sprint 1 - Review

Sprint **Retrospective**: Sprint 1 - Retrospective

Team **Structure**: Roles

Sprint Tasks:

- Visual Design Decisions
- Site Structure and Navigation
- User Stories
- Motivational Modelling
- User Persona
- Architectural Design

Visual Design Decisions

Theme: Smart/Elegant/Academic-core

Typography: Open Sans - Google Fonts

Imagery: Potentially create our own infographics - Lin

Features:

Theme - can toggle between light or dark

- update and add along the way

Colour Scheme: Coolors.co

Main colour - Navy Blue #163169

- inspiration from Unimelb

Accent colour - Green #5cbb81

Logo Designs

Site Structure and Navigation

Sitemap: Home (contains the function), Login, Profile,

Navigation Menu: Home, Login, Profile

Header: Title, Logo (on top left), User Profile (on top right)

Footer: About (what the website does), creditation

=> In the middle: big textbox (maybe 3/4 of the page) with options “paste your text” and “upload a doc” below it. The textboxes stack on top of each other like cards.

=> When you compare the texts, the site should automatically scroll down. Add a “back to top” button.

Website Decision

- Fake loading screen - depending on whether or not the model runs within a time limit or not
  - Representation gap between user and developers
    - A quicker load time can give off the impression for the less technologically inclined to believe that the website and model did not run clearly and decisively
  - In the case that the texts that are being analysed do require time to run, the loading screen will serve a purpose to let the user understand that its running
    - (!!) idea to have a message pop up if the texts are long, that the model is taking longer to run

👥 User Stories

AIM: To compare a set of texts that we know come from an individual and compare that with another given text to see the likelihood of the second text being written by the same individual that wrote the first set

---

EPIC: Compare the authorship of texts in a simple, elegant and intuitive manner.

As a **Teacher** User**:**

- I want to be able to compare my student's texts to see if they were written similarly
  - Medium
  - To be able to check for plagiarism between student works
- I want to be able to create a profile for my students to make it easier to identify their work/authorship
  - Medium
  - To ensure that their work was done by themselves
- *(Optional)* I want to be able to upload a folder/create a directory to create/manage my student's profiles

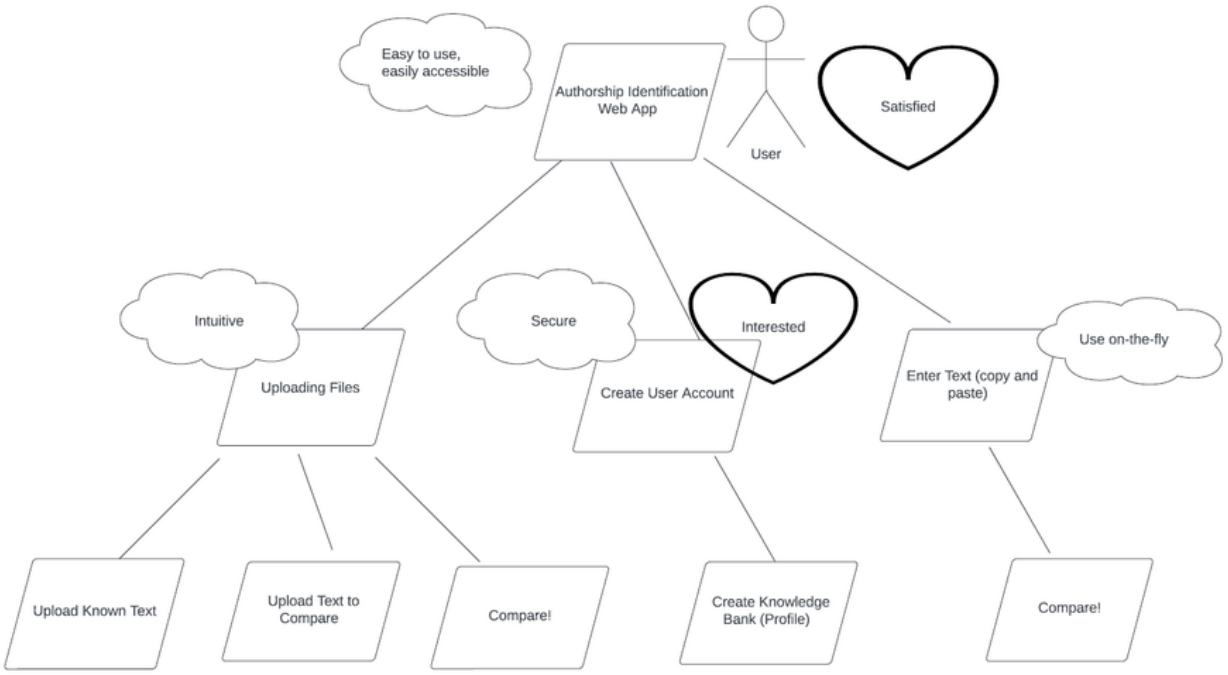
---

As an **Author/Student** User**:**

- I want to be able to compare one of my texts with another one of my texts to see if my authorship match
  - High
  - Must: main function of website, can help ensure consistency across works
- I want to be able to get a more accurate reading of my authorship similarity by adding more texts
  - Medium
  - Should: I should be able to get a more accurate score for a comparison by allowing the website too to look at more of my work
- (!!: Extensions)
  - I want to be able to see which parts of my texts have a higher similarity
    - (function: create a pdf and highlight similarities)
- I want to be able to add more of my original work to create my authorship profile
- I want to be able to see the history of what I have compared before
- I want to be able to see if my work has a high similarity to a text/author I have referenced from

---

Motivational Modelling



User Persona

Gerard Cain

"I need a tool that can identify potential instances of plagiarism without false positives."

Age: **45**
Work: **Professor of English Literature**
Family: **Married**
Location: **Melbourne, Victoria**

Personality

Introvert	Extrovert
Thinking	Feeling
Sensing	Intuition
Analytical	Creative

Organised | Hard Working | Empathetic

Goals

- Verify the originality of his students' assignments to prevent plagiarism
- Provide constructive feedback to students while maintaining trust.

Challenges

- Increasing instances of suspected plagiarism among his students.
- Balancing the need for strict plagiarism detection with fostering a positive learning environment.
- Problems with the currently available solutions.

Bio

Dr. Gerard Cain is a professor of english literature. He has a Ph.D in English Literature from the University of Melbourne. He has been teaching for 15 years and is well known for his expertise in Australian literature. Dr. Cain is dedicated to maintaining academic integrity in his classroom and ensuring that his students submit original work. He is proficient with common academic software tools and is open to using new technology to improve the educational experience for his students.

Motivation

Price
Convenience
Speed
Accuracy

Behavioural Insights

- Regularly assigns essays and research papers
- Values educational technology that assists in maintaining academic integrity
- Wants a user-friendly tool that can me integrated into his learning management system (LMS)

Emma Rodriguez

"Sometimes, I worry that my writing might appear inconsistent, especially when I'm exploring different topics "

Age: **21**
Work: **Student**
Location: **Melbourne, Victoria**
Gender: **Female**

Personality

Introvert	Extrovert
Thinking	Feeling
Sensing	Intuition
Analytical	Creative

Diligent | Motivated | Inquisitive

Goals

- Compare own texts to see if authorship matches
- Get accurate information about their own writing style
- Ensure consistency across works of writing

Challenges

- Struggles with maintaining a consistent writing style across assignments
- Concerned about accidental plagiarism when writing essays

Bio

Emma is a motivated student with a passion for literature. She is in her final year of her bachelor's degree studying English literature at the University of Melbourne. She often works on various writing projects such as essays and creative writing assignments as part of her studies. She also has a strong desire to improve her writing skills.

Motivation

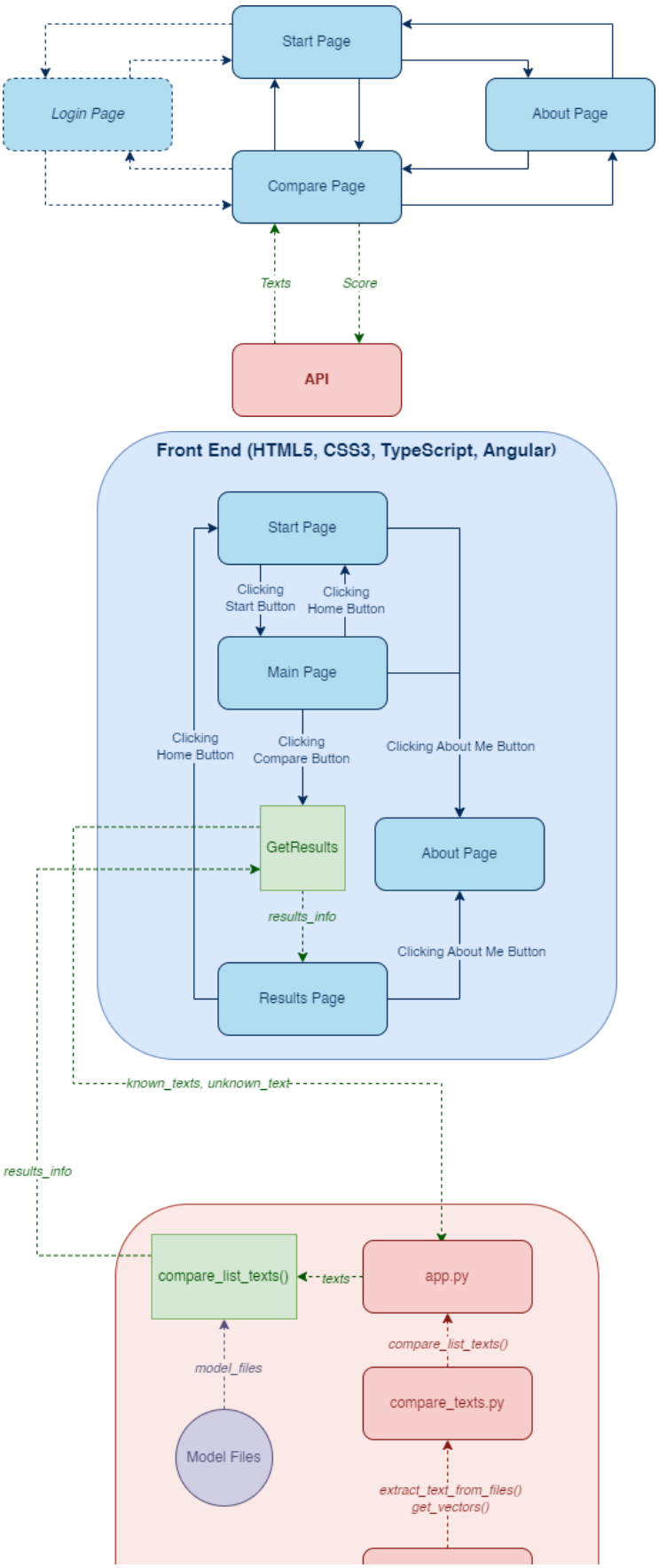
Price
Convenience
Speed
Accuracy

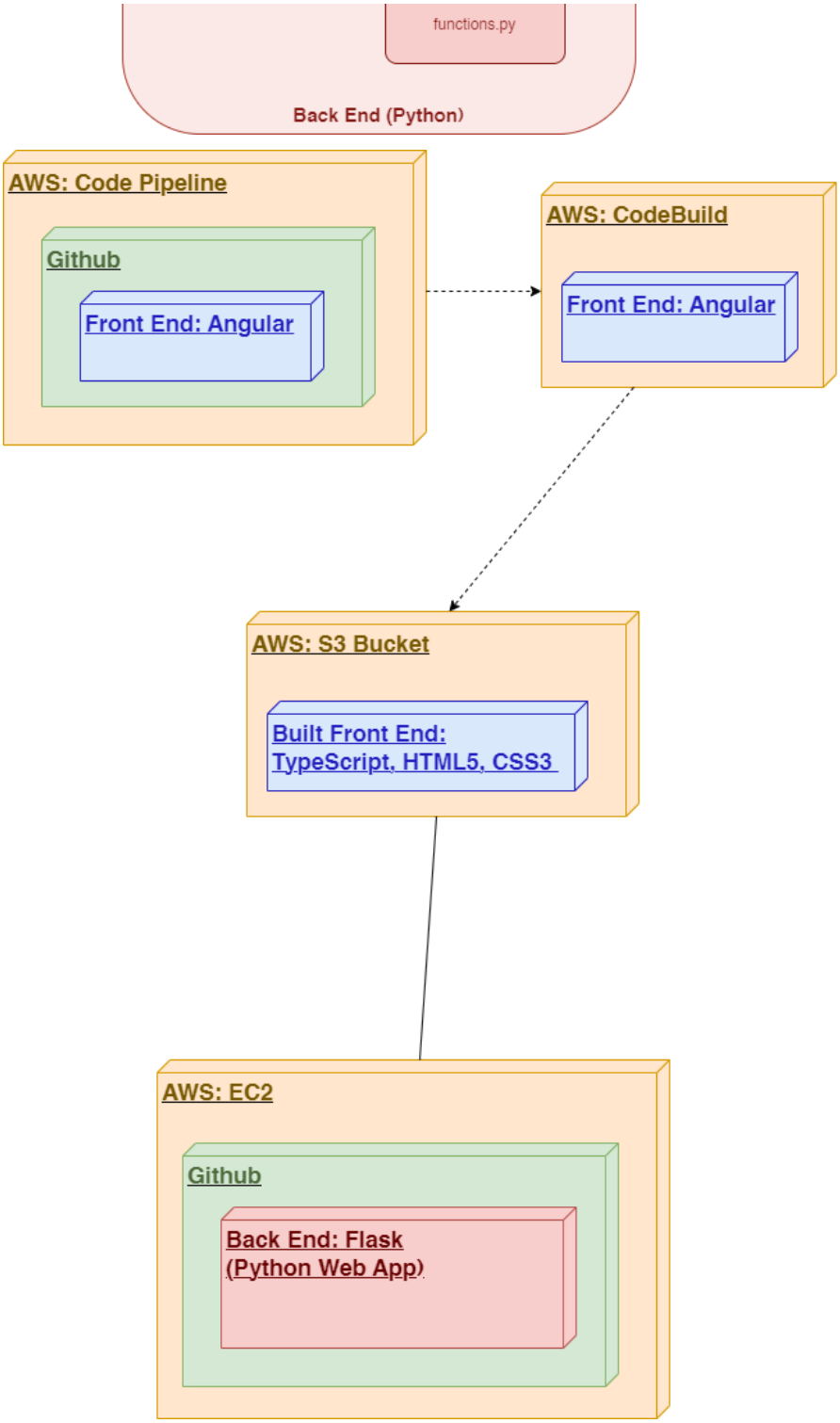
Behavioural Insights

- Regularly writes essays or creative pieces
- Values a user-friendly and reliable tool to help assess her own work
- Actively seeks out resources and tools that can help improve her writing skills

Architectural Design

High-Level Structure & System Behaviour





Where `compare_list_text()` takes in the texts inputted from the front end and the `model_files` from the algorithm and outputs `results_info` which is the authorship percentage.

Performance Optimization Considerations

- **Image optimization:** File formats, compression, resolution
- **Minification**: HTML, CSS, JavaScript files
- **Caching**: Browser caching, CDN usage
- Page load speed considerations

Database

Dependencies

- SQLAlchemy (ORM – object relational mapper) - used to access database

Working with Engines and Connections — SQLAlchemy 2.0 Documentation

- Database Driver/ Dialect

Dialects — SQLAlchemy 2.0 Documentation

Possible database

1. PostgreSQL
2. MySQL and MariaDB
3. SQLite
4. Oracle
5. Microsoft SQL Server

SQLAlchemy packages used

- create_engine, ForeignKey, Column, String, Integer, DateTime
- sqlalchemy.ext.declarative import declarative_base
- sqlalchemy.orm import sessionmaker

declarative_base = base class where all entities class inherited from

sessionmaker = allow us do stuff to database.

Helpful video - SQLAlchemy Turns Python Objects Into Database Entries

Architecture Resources

Resources for Architectural Plan

Part 1 - Set up backend, Database, frontend, flask http request

Using Python, Flask, and Angular to Build Modern Web Apps - Part 1

Part 2 - Set up Auth0 for both frontend and backend

Using Python, Flask, and Angular to Build Modern Web Apps - Part 2

Part 3 - Angular, set up roles in Auth0

Using Python, Flask, and Angular to Build Modern Web Apps - Part 3

pipenv documentation

pipenv

API research

Dependencies

- Flask
- Marshmallow

Flask package used

- Flask
- Jsonify
- request

Marshmallow package used

- Schema
- Fields

Algorithm Research

Algorithm Analysis

The algorithm has a `data` folder that's split into `train_data` and `test_data` where each folder is filled with folders of any name that contains `.txt` files where in which there are 1+ `knownX.txt` files and 1 `unknown.txt` file. *The* `test_data` *and* `train_data` *folders also contain* `contents.json` *and* `truth.json` *files which act as a sort of directory on what folders are to be used and what the answer is for each folder's authorship.*

All the `.txt` files are then preprocessed so as to be more space efficient and only contain the useful word data, this process as stated in the `preprocess_text()` function, "Preprocess a given text by tokenizing, removing punctuation and numbers, removing stop words, and lemmatizing".

Using a combination of all the training data set of preprocessed text, we train a **Word2Vector Model** which uses the trained dataset to learn relationships between words.

We then utilise the **W2V Model** alongside a **Style Vector** (Vector of punctuation, sentence lengths, words used and word count created using the function `calculate_style_vector()`) to create the final vector data which will be used by the model.

The main model itself is then built using a **SiameseNet** which sort of combines the `base_network` and `clf_network` (2 unique neural network models with different layers amounts and activation functions) so that both models can be trained together.

The model then validates the `clf_network` by using both a train and validation set and reducing loss, thus increasing the final model performance.

Finally the models are tested using `test_data` which has been transformed into word vectors, then transformed again using the `base_network` to create `siamese_vectors` that the `clf_network` model uses to get the final percentage score on the known texts vs unknown text authorship score.

Final Algorithm For The Project

The main 3 models that this entire process uses are the **W2V Model**, **Base Network Model** and **CLF Network Model**. So after using the `PAN14_data_demo.ipynb` file to train and validate the models, I saved the model files into a `model_files` folder so that the models can be reused without needing to be retrained, thus drastically speeding up the processing.

After the algorithm analysis I found the main process to get an authorship score for a folder of known and unknown texts was using the function `extract_text_from_files()` to get the processed texts and transform them into a `numpy` array word vector using the **W2V Model**. I then used the **Base Network Model** to obtain predictions on the known and unknown word vectors which were then transformed into representations by getting the `numpy` means. A combined array of the known and unknown representations was then used by the **CLF Network Model** to obtain the final score for the texts authorship.

A summary of the process would be:

1. Load models
2. Extract known and unknown texts from file
3. Transform texts into word vectors
4. Use **Base Network Model** to create unknown and known representations
5. Use the combined array of representations with the **CLF Network Model** to obtain the final authorship score

2 Sprint 2

Sprint **Planning**: Sprint 2 - Planning

Sprint **Review**: Sprint 2 - Review

Sprint **Retrospective**: Sprint 2 - Retrospective

Team **Structure**: Team Roles

Sprint Tasks:

- Coding Guidelines
- Documentation
- Sprint 2 - Team Decisions
- Team Roles
- Git Notes/Guidelines
- Prototype Changes
- Security
- Motivational Modelling [UPDATED]
- Current Website

Coding Guidelines

Naming Conventions

- Choose simple, meaningful and descriptive names for variables, functions, classes and files.

Code Organization

- Organize and separate code into respective sections and modules depending on its functionality
- Build and follow a file structure and directory hierarchy

Code Reusability

- Write reusable functions and modules if necessary
- Avoid duplicating code

Code Readibility

- Keep lines of code short and neat
- Make sure that the code is not overly complex or have a nested code structure

Comments and Documentation

- Write brief and simple comments to explain what the code is about
- Provide comments for complex functions, classes and modules describing the its purpose

Version Control

- Use github as the version control system
- Commit regularly and provide clear and concise commit messages

Dependency management

- Manage external dependencies required carefully using a package manager
- list all dependencies and packages used for the app in text file
- Keep dependencies up to date and secure

Testing and Quality Assurance

- Ensure that any code changes produce desirable result during testing before pushing it to the main branch

Error Handling

- use try-catch blocks if neccesary for proper error handling
- provide meaningful error messages

Maintaining Coding Guidelines

Through the use of GitHub Actions and code linting we are able to keep our coding guidelines maintained and thoroughly checked. We use the standard NodeJS linter and python linter (pylint using PEP8 conventions) as the standards used. Both of these linters adhere to the guideline standards we are looking for and as such were our chosen linters.

Documentation

To set up in Visual Studio Code:

1. Install Node.js here: Download | Node.js
2. Open up a terminal in VSC, and type: `npm install -g @angular/cli`
3. When done, navigate to the folder containing the pages of the website
4. You can run the website by typing: `ng serve`
5. You can view the website by opening up a browser and going to `localhost:4200`

Note: this only works if your terminal is `cmd`, not `powershell`.

Python needs to be installed for the compare page to work:

1. Do `Ctrl + Shift + P` and choose Python: Create Environment
2. Choose `Venv`
3. Choose the Python interpreter that you prefer
4. Select `requirements.txt`, then click "OK"

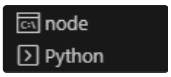
Note: if this is your first time running the website, you need to install `NLTK`. The steps are as follows:

1. Go to the `back-end` folder and select `functions.py`
2. Uncomment line 8, and save
3. You can discard the changes made to `functions.py` so it does not get pushed along with your commit

To run the algorithm along with the website:

1. Select `app.py` in the `back-end` folder
2. Click the “run” button located on the top-right of your screen (in VSC)
3. If this is your first time running it should install `NLTK`, then run the server

Note: you should have both of these under your terminal to use the website and algorithm together.



Sprint 2 - Team Decisions

Monday 28th August

- Decided on work delegation
- Decided on dates for meetings / task deadlines
- Created logo samples in preparation for logo decisions

Thursday 31st August

- Decided on backend Python packages
- Decided to use `input` for frontend
  - plan use of buttons linked to text area
- Flask and backend extensions
- User stories rehaul to fit version 1.0

Thursday 7th September

- Fixed utilizing `textarea` instead of `input` for a better visualization of inputting text into the website

Monday 11th September

- minor styling decisions for the front end of the website
- decided on the first iteration of upload (.txt focus first, other file types will come after everything is linked)

Wednesday 13th September

- upload files to include .docx and .pdf
- major switch on frontend side to shift to flex-box to ensure scaling of all components
- major restructuring of git files for readability and easy understanding

Monday 18th September

- continuous upload of of known files and ability to delete specific files from the already uploaded ones

Tuesday 19th September

- functionality and visual changes made under group decision
- final completed stamp on back end!

Thursday 21st September

- decided to add more security on API
- final wrap up decisions on sprint 2, largely just confirmation checks
  - confluence page alterations and documentation that we felt was needed

Team Roles

Development Team

Scrum Master - Lin

- Planning, organizing and managing of confluence, progress and meetings
- Cooperating with the front end team on the website design
- Cooperating with the back end team on making sure all intended features of the version are added correctly and efficiently

Frontend Developer - Jia Jie, Adrian

- Coding, executing and adjusting of all matters related to front-end
- Cooperating with the back end team on making sure the correct data and their corresponding data types are pipelined into the front-end

Backend Developer - Saaiq, Hanizam

- Coding, executing and adjusting of all matters related to back-end
- Cooperating with the front end team in making sure the connection between the front-end and back-end is seamless with errors from client and server side being handled correctly

Git Notes/Guidelines

- how we merge, what guidelines we have, wh en we have merge conflicts what do we do

Branching:

When branching, follow the general guidelines below:

Branch name:

> [name]/[task/item]

An example of this would be:

> Lin/website header

Commits

Commits within individual branches align to the following guidelines:

> [main work]: [short but descriptive line about commit]

Current main work tasks:

- Feat
  - Feature currently added
  - Code largely creates and implements feature being stated
- Fix
  - a bug fix or code fix
  - done after the commit of a feature

Merging Guidelines

Current:

- When merging to main, send a message to Slack to confirm with other team members, or
- Confirming with team members whilst in a meeting on Zoom

New:

- Implement branching
- Adhere to commit guidelines
- Continue to notify team members through Slack

Handling Merge Conflicts

- Contact specific team member/s to resolve merge conflict
- Organise a small meeting to handle merge conflict
- In the case of accidental overwrites, revert back to previous version/s, and notify team members

Prototype Changes

Changes made since last iteration of prototype:

- Animation of text input area has now been *removed*
  - Switched to two text input areas side by side for ease of use and switch between known and unknown text
- inclusion of an add text input area to file button in the known text area
  - to allow for more texts to be included via the text input area
- Removal of a loading screen while texts process
  - due to changes made to backend, text analysis was significantly sped up, reducing the need for a log in screen

Changes to be added

Analysis Low fidelity Prototype

- a breakdown of all available information provided by the model on the texts being analyzed

Security

API Security

Input Security

1. Using a `compare_model` which the API expects to make sure the correct data is sent and used by the function
    a. In this case the `compare_model` expects **form data** with the arguments `known_texts` , `unknown_text` , `known_files` , `unknown_file`
    b. Makes sure to include the specific data types of each argument, its requirement and a description for the docs for potential users to understand
2. Error checking the inputs sent with a `400: "No known texts provided"` and `401: "No unknown text provided"` post error that is sent through, which aborts the function as intended

Function Security

1. Error checking the function with a `500: "Error calculating score"` to make sure there is improper score being outputted
2. Removing duplicate and white space texts as well as ignoring files that couldn't be parsed to insure there is no API crashing

Output Security

1. Using a `score_model` which the API marshals to make sure only the required API data is sent through
    a. Is in the form of a JSON response with a `score` header as well as headers for other text statistics
    b. Includes the exact data typing's, requirements and description of each header in the model

Login Security

In the event that a sign up and login system is implemented

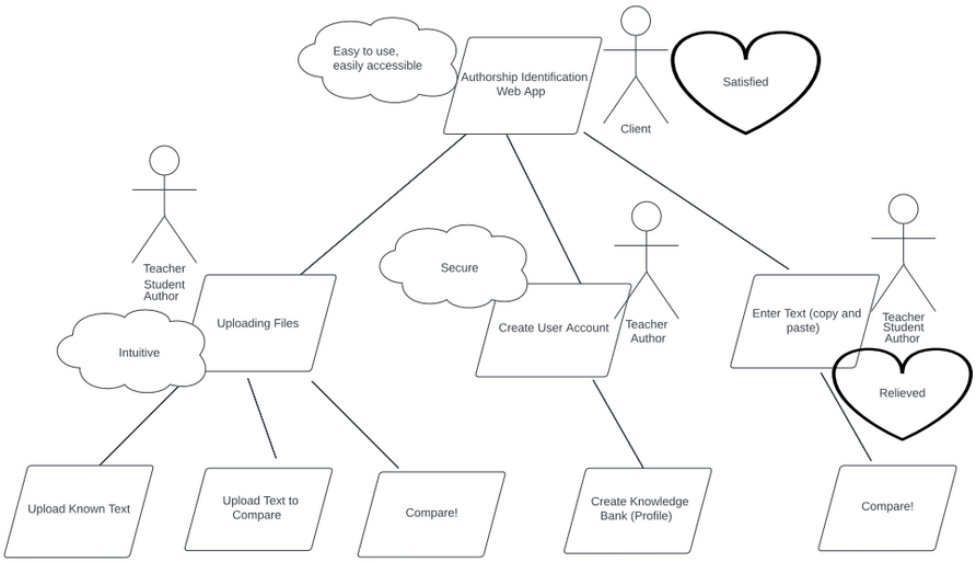
0Auth

- The use of the 0Auth service wouldn't just allow for a simplified implementation of an advanced login system but also would provide security in the handling of account data
- Would provide secure storage and accessing of sensitive password information

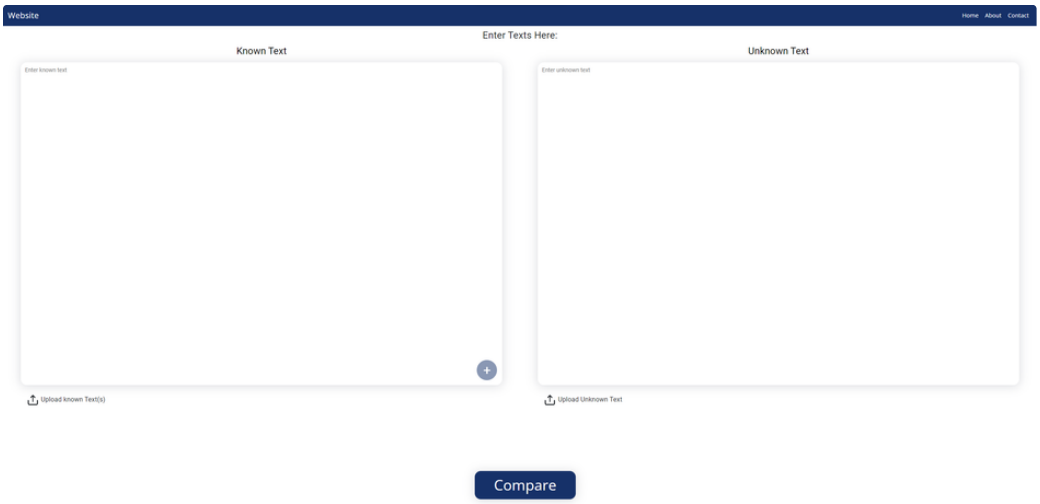
Profiles

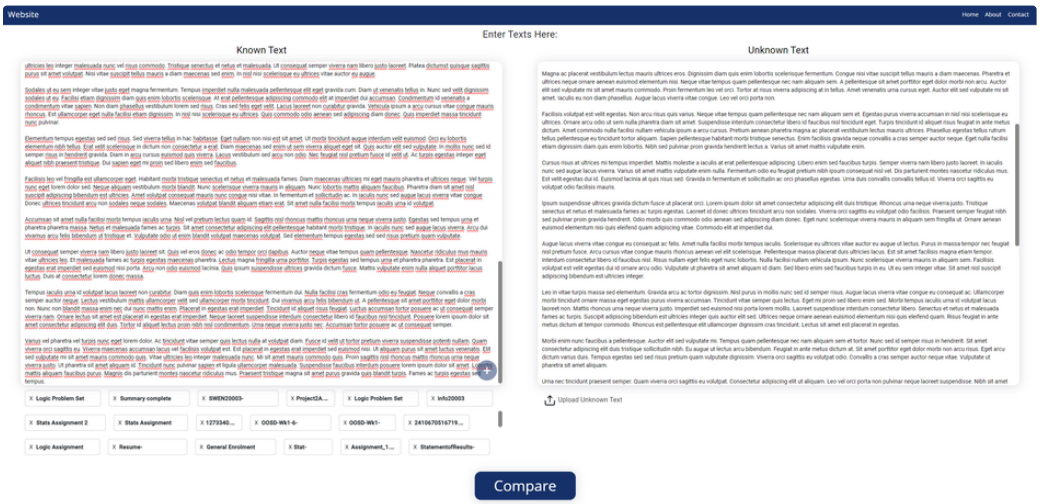
- Would need to implement authentication (potentially alongside the 0Auth authentication) with the API to access profile data
    - Help keep profile data private so only the correct user has access to view profiles

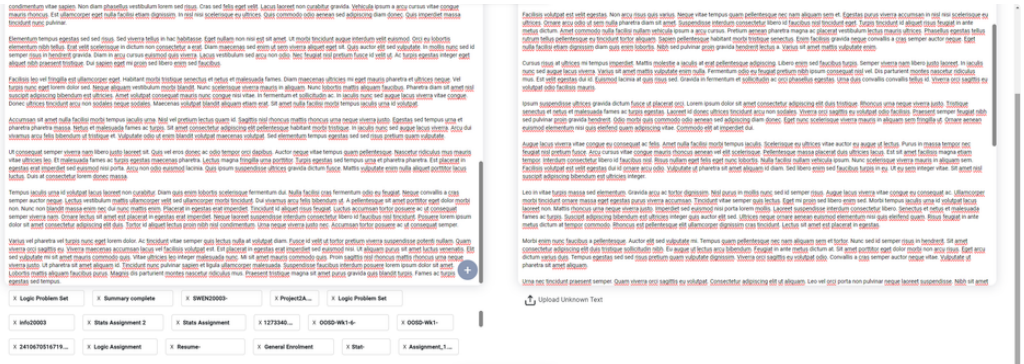
Motivational Modelling [UPDATED]



Current Website





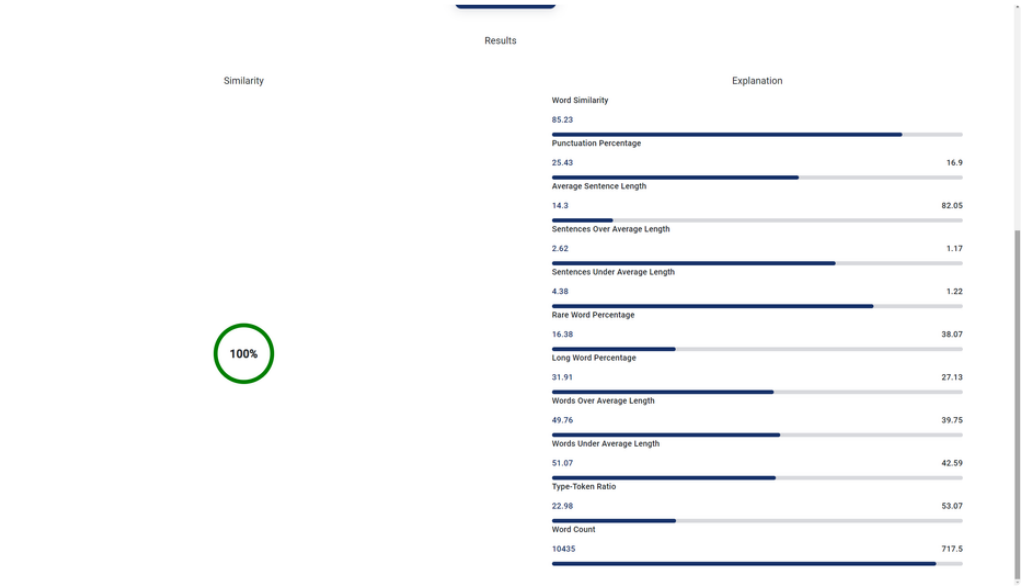


Results

Similarity

100%

Explanation



3 Sprint 3

- Updated User Stories
- Testing
- Deployment Process & CI/CD
- Website Updates
- Changes Made Since
- Model Performance Testing
- Tutorial Images

Updated User Stories

AIM: To compare a set of texts that we know come from an individual and compare that with another given text to see the likelihood of the second text being written by the same individual that wrote the first set

---

EPIC: Compare the authorship of texts in a simple, elegant and intuitive manner.

As a **Teacher** User**:**

- As a teacher, I want to be able to compare the works of my students to their own past work that I know of, and/or to potentially compare to another students work, and see the authorship similarity between them. This would allow me to better understand my student's current work, and better assist me in ensuring the authenticity of their work and make decisions related to it.
- I want to be able to get a more accurate reading of the authorship similarity by adding more texts that I know belong to the student based off past submitted work.
- I want to be able to upload documents of different file types for comparison as student submissions may vary in type.

Testing

- Acceptance Criteria
- Acceptance Test
- Test Cases
- Structural Testing
- Bug Testing
- Non-Functional Testing

Acceptance Criteria

Created based on updated user story.

ID	Feature	**Acceptance Criteria** *(Given/When/Then)*
1	Input Interface	Given the user is looking for a website to compare two different texts, when the user accesses the website, they then get greeted with a *user-friendly interface* where they can input a set of known texts and an unknown text for comparison
2	Security	Given the user is uploading documents that may be sensitive or personal, when the user uploads the documents to our website, our website processes the information then returns the result through the already pre-trained model without storing any of the documents
3	Multi-file type input	Given the user has documents of multiple types to upload, when the user uploads documents, they have the flexibility to upload .txt, .docx and .pdf, alongside textbox input. Then the user can start the comparison model
4	Results Explanation	Given that the user has uploaded and clicked compare, when the model has finished processing the texts, the results are then presented as a large visual circle with colour indicating similar percentage. The results is given alongside a set of data on the similarity information such as word similarity, punctuation percentage and more.
5	Instructions	Given that the user is not familiar with the website or has forgotten how to use it, when the user clicks at the bottom right of the website (the ? mark button), the instructions then pop up to help assist the user

Adequacy criterion = set of test obligations. E.g., cover all statements and all branches in the program under test

A test suite satisfies an adequacy criterion if

✔ all the tests succeed (pass)

✔ each test obligation is satisfied by >=1 test case(s)

Acceptance Criteria: set of predefined requirements that must be met in order to mark a user story complete.

- given [condition], when [something happens], then [result]

Acceptance Test

AC ID	AT ID	Acceptance Test	TC	Success?
1	1.1	User is able to see the home/welcome screen upon accessing website	**TC-5**	✓
	1.2	User is able to see the main features of website after successful load out of website	**TC-5**	✓
	1.3	User is able see all the navigation and buttons clearly	**TC-5**	✓
3	3.1	User is able to upload .pdf, .docx, .txt, or a combination of them	**TC-2**	✓
	3.2	User is able to use the text input box multiple times through converting them into files with the '+' button	**TC-3**	✓
4	4.1	User is able to successfully upload files/utilise the text boxes	**TC-2, TC-3**	✓
	4.2	User is able to successfully press the 'Compare' button	**TC-1**	✓
	4.3	User is able to see the animation and display of authorship comparison results.	**TC-1**	✓
	4.4	User is able to see the visual explanation bars on the details extracted from the comparison	**TC-1**	✓
5	5.1	User is able to click the '?' button at the bottom right of the website	**TC-4**	✓

AC - Acceptance Criteria, AT - Acceptance Test, TC - Test Case

Test Cases

Test Case [TC-1]

Test Type: Functional **Execution Type:** Manual

Objective: Test if results screen shows proper results after pressing compare button given correct input

Setup: Must have at least one known text and one unknown text

Pre-Conditions: Dynamic results bar shows resulting similarity percentage with colour depending on percentage

Notes:

1. Add known text(s) and unknown text through textbox or uploading files
2. Press compare button
   - results section should appear
   - results bar should have fill animation with colour based on percentage
   - explanation section should show evaluation metrics with numbers and similarity bars

Time constraint:

Minimum: 5 min

Maximum: 10 min

Test Case [TC-2]

Test Type: Functional **Execution Type:** Manual

Objective: Uploading files on compare page

Setup: Be on compare page with files ready to upload

Pre-Conditions: Any files uploaded to either the known or unknown texts sections should show up as file cards under the text boxes while can be removed by clicking the 'x' button

Notes: Types of files accepted: PDF, docx and txt

Time constraint:

Minimum: 5 min

Maximum: 10 min

Test Case [TC-3]

Test Type: Functional **Execution Type:** Manual

Objective: Utilize the '+' button in the text area to save text box text into a file - multiple inputs

Setup: Be on compare page with multiple texts ready to upload

Pre-Conditions: multiple texts of medium length separated for multiple input attempts

Notes: N/A

Time constraint:

Minimum: 5 min

Maximum: 10 min

Test Case [TC-4]

Test Type: Functional **Execution Type:** Manual

Objective: Seeing if tutorial pop-up button works

Setup: Be on compare page

Pre-Conditions: After clicking the '?' button in the bottom right of the page, a pop-up should appear with images containing instructions on how to use the website

Notes: Images should be easily readable and instructions should be clear for a new user

Time constraint:

Minimum: 5 min

Maximum: 10 min

Test Case [TC-5]

Test Type: Functional **Execution Type:** Manual

Objective: Fully functional load out of deployed product

Setup: Access to link - AuthoWrite

Pre-Conditions: Website is fully deployed, no crashes occurred and latests changes have been updated and passed through test cases

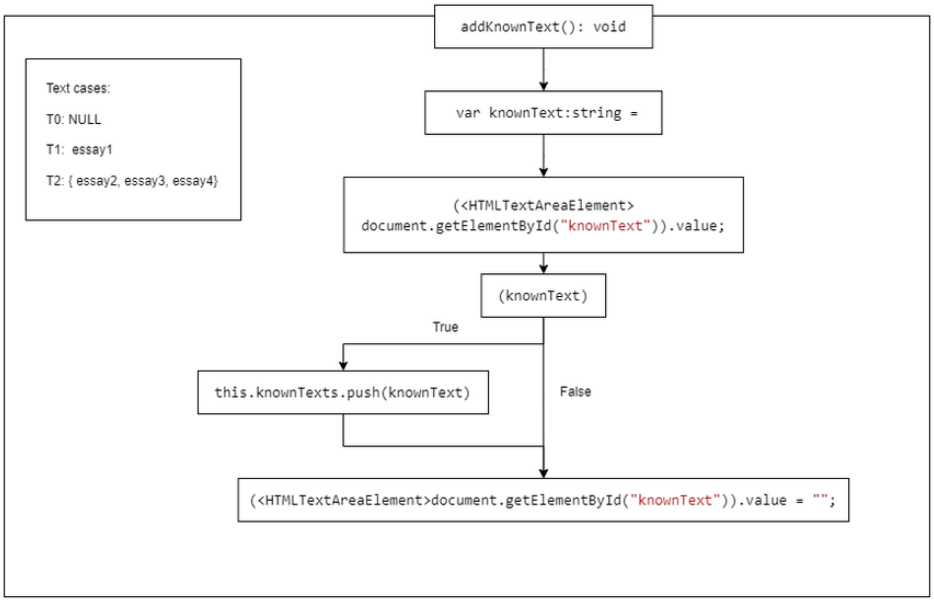
Notes: Check through that all things are loaded, buttons are all present

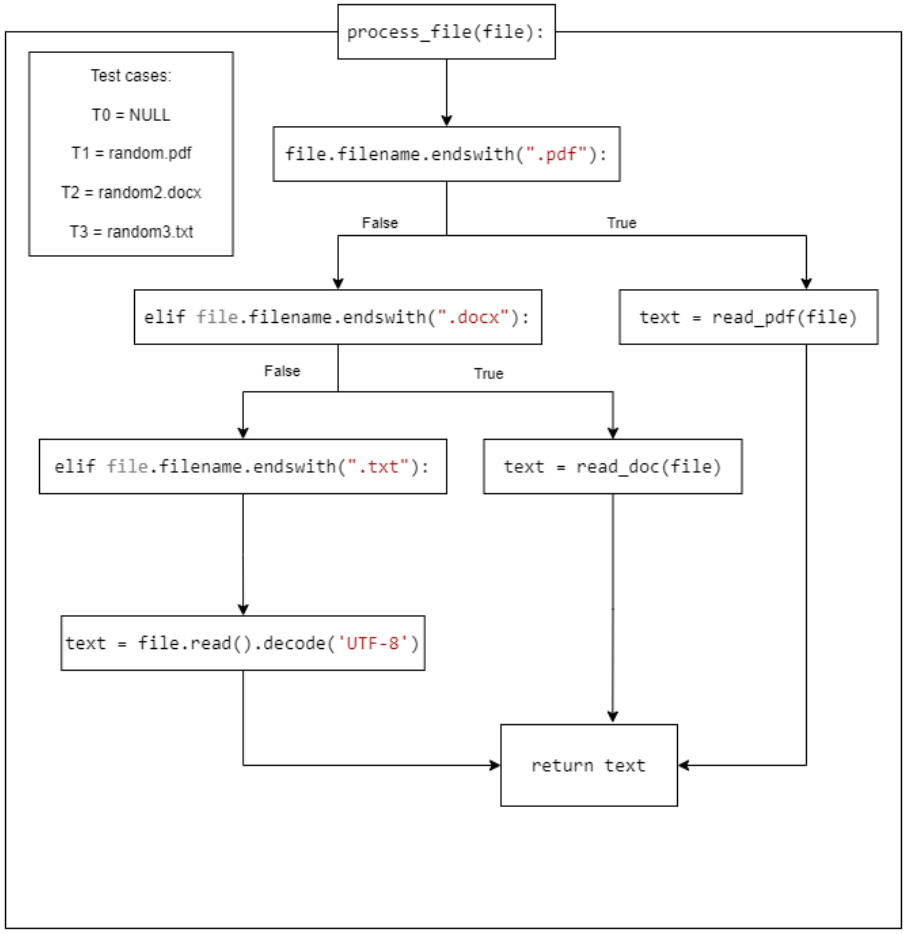
Time constraint:

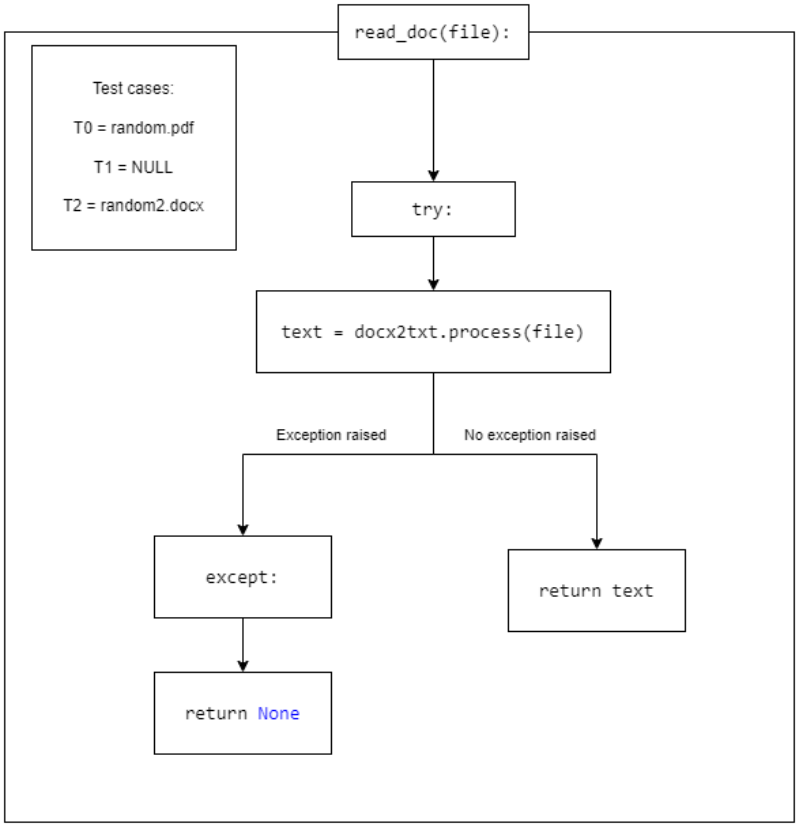
Minimum: 2 min

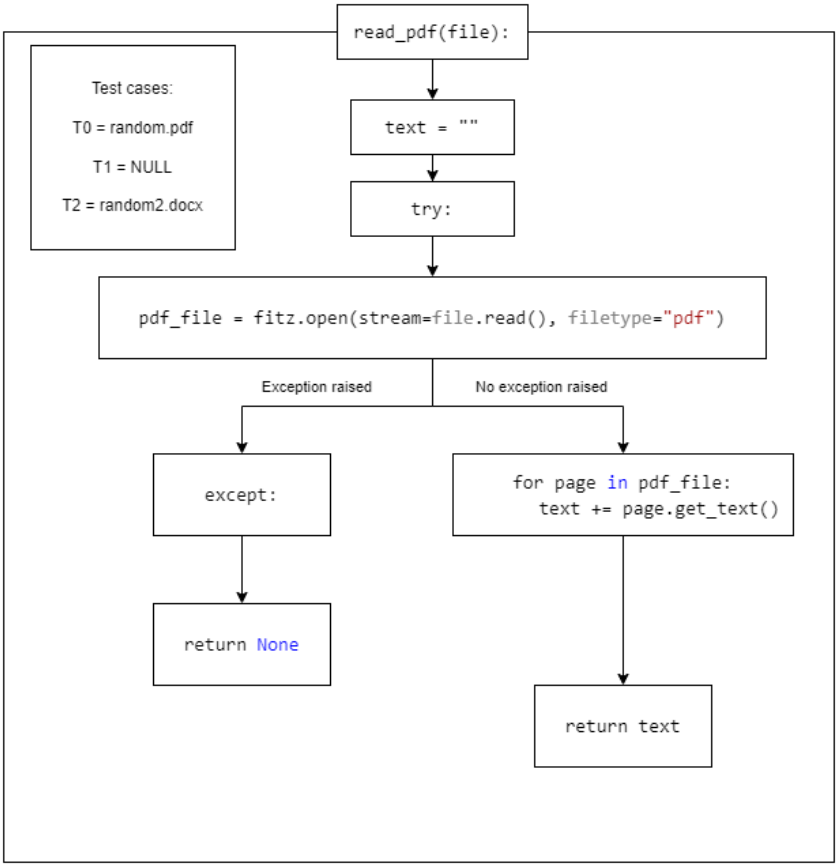
Maximum: 10 min

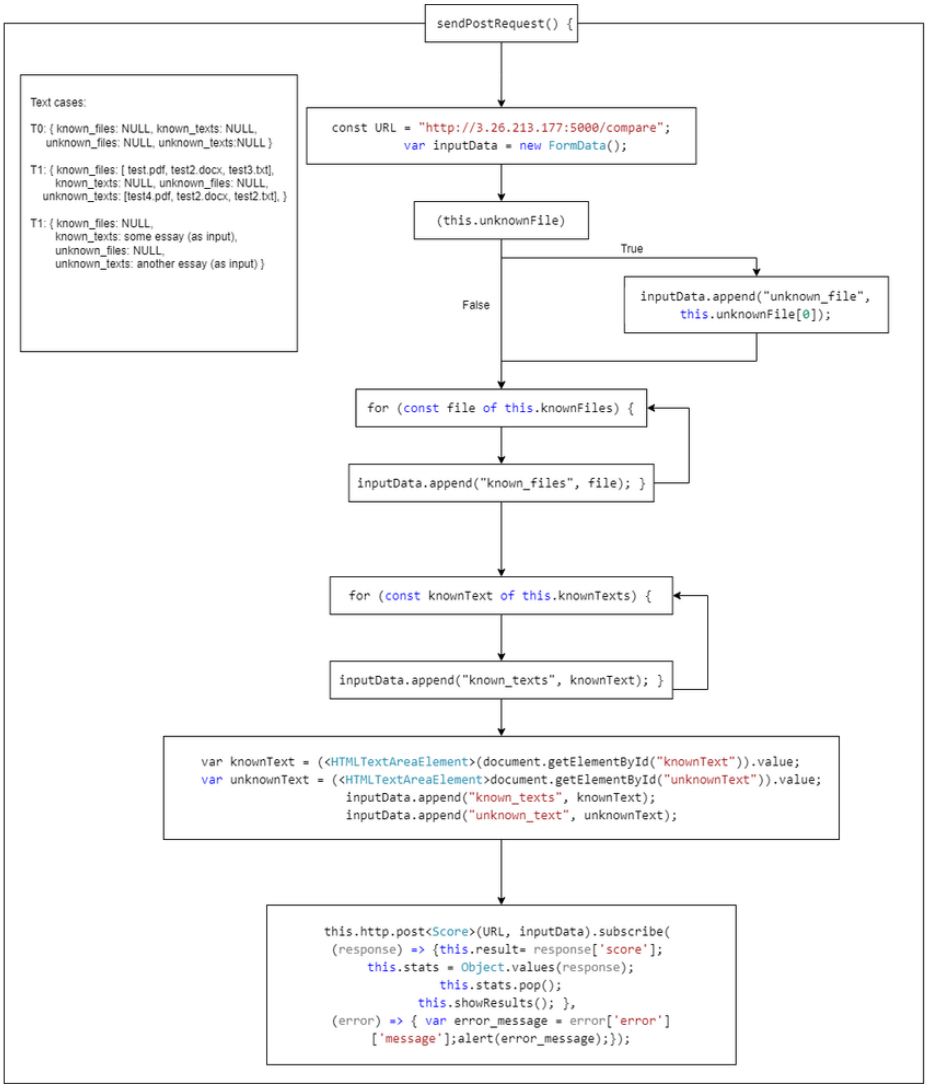
Structural Testing

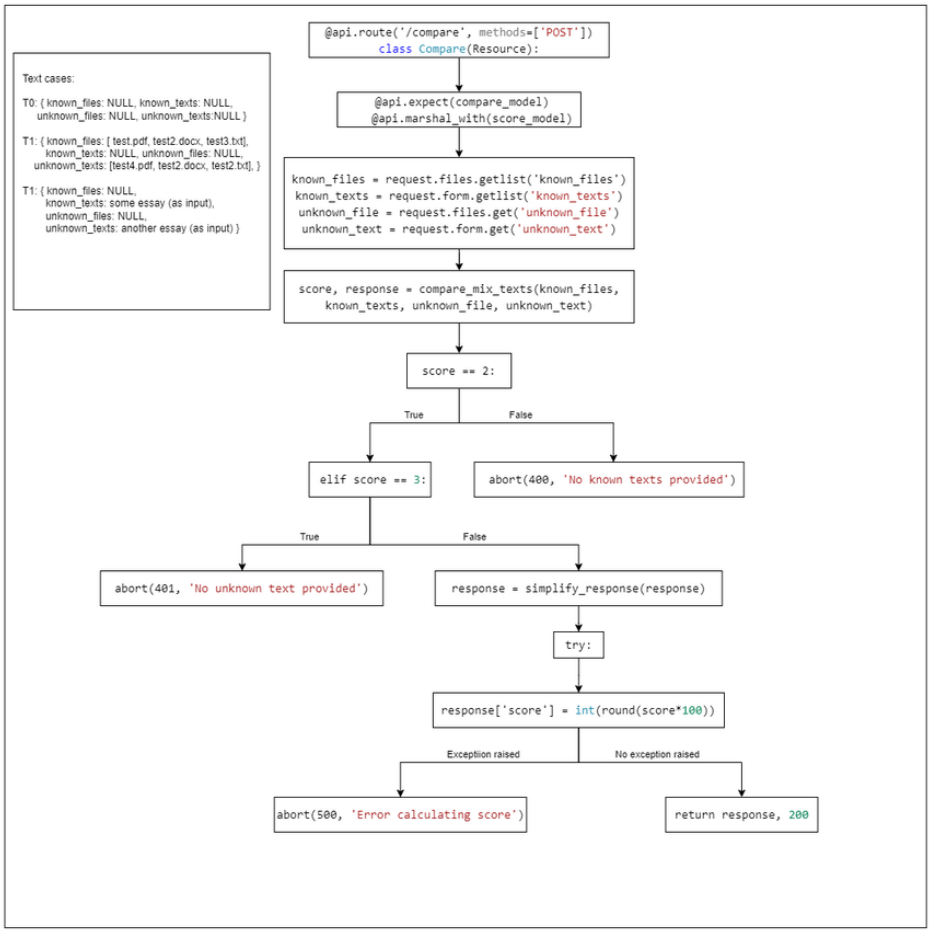



process_file(file):
Test cases:
T0 = NULL
T1 = random.pdf
T2 = random2.docx
T3 = random3.txt
file.filename.endswith(".pdf"):
False
True
elif file.filename.endswith(".docx"):
text = read_pdf(file)
False
True
elif file.filename.endswith(".txt"):
text = read_doc(file)
text = file.read().decode('UTF-8')
return text


read_doc(file):
Test cases:
T0 = random.pdf
T1 = NULL
T2 = random2.docx
try:
text = docx2txt.process(file)
Exception raised
No exception raised
except:
return text
return None


read_pdf(file):
Test cases:
T0 = random.pdf
T1 = NULL
T2 = random2.docx
text = ""
try:
pdf_file = fitz.open(stream=file.read(), filetype="pdf")
Exception raised
No exception raised
except:
for page in pdf_file:
text += page.get_text()
return None
return text


sendPostRequest() {
Text cases:
T0: { known_files: NULL, known_texts: NULL, unknown_files: NULL, unknown_texts:NULL }
T1: { known_files: [test.pdf, test2.docx, test3.txt], known_texts: NULL, unknown_files: NULL, unknown_texts: [test4.pdf, test2.docx, test2.txt], }
T1: { known_files: NULL, known_texts: some essay (as input), unknown_files: NULL, unknown_texts: another essay (as input) }
const URL = "http://3.26.213.177:5000/compare";
var inputData = new FormData();
(this.unknownFile)
True
False
inputData.append("unknown_file", this.unknownFile[0]);
for (const file of this.knownFiles) {
inputData.append("known_files", file); }
for (const knownText of this.knownTexts) {
inputData.append("known_texts", knownText); }
var knownText = (<HTMLTextAreaElement>(document.getElementById("knownText")).value;
var unknownText = (<HTMLTextAreaElement>document.getElementById("unknownText")).value;
inputData.append("known_texts", knownText);
inputData.append("unknown_text", unknownText);
this.http.post<Score>(URL, inputData).subscribe(
(response) => {this.result= response['score'];
this.stats = Object.values(response);
this.stats.pop();
this.showResults(); },
(error) => { var error_message = error['error']
['message'];alert(error_message);});


Text cases:
T0: { known_files: NULL, known_texts: NULL,
unknown_files: NULL, unknown_texts:NULL }
T1: { known_files: [test.pdf, test2.docx, test3.txt],
known_texts: NULL, unknown_files: NULL,
unknown_texts: [test4.pdf, test2.docx, test2.txt], }
T1: { known_files: NULL,
known_texts: some essay (as input),
unknown_files: NULL,
unknown_texts: another essay (as input) }
@api.route('/compare', methods=['POST'])
class Compare(Resource):
@api.expect(compare_model)
@api.marshal_with(score_model)
known_files = request.files.getlist('known_files')
known_texts = request.form.getlist('known_texts')
unknown_file = request.files.get('unknown_file')
unknown_text = request.form.get('unknown_text')
score, response = compare_mix_texts(known_files,
known_texts, unknown_file, unknown_text)
score == 2:
True
False
elif score == 3:
abort(400, 'No known texts provided')
True
False
abort(401, 'No unknown text provided')
response = simplify_response(response)
try:
response['score'] = int(round(score*100))
Exceptiion raised
No exception raised
abort(500, 'Error calculating score')
return response, 200

Bug Testing

Functional testing:

- best for missing logic faults
- A common problem: Some program logic was simply forgotten
- applies at all levels
  - unit - form module interface spec
  - integration - from API or subsystem spec
  - system - from system requirements spec

- tested corrupted files
- tested scroll bars - still need to change
  - include edge
- tested files which do not contain any text (eg. only images)

Mobile Testing

- Found an initial issue of a white background when zooming out of the page
  - Solution: needs to be changed to blue
- New issue after deployment of a failure in the responsive design
  - Solution: fix issues with flex and flex-grid

Non-Functional Testing

- **Visibility**
  - When we hit “compare”, the page automatically scrolls down to the results section. We implemented this feature because it was not intuitive as to where the results were.
  - We added a tutorial page that shows clearly how to use the website.
  - We have file icons whenever a user adds a file, and users can also press the “X” to delete them. We implemented this feature because using only alert boxes to notify the addition of a file was not very clear, and also this allows users to remove files easily.
- **Match**
  - Also for adding files, we use the universal “+” symbol to show users that they can add files. Furthermore, “X” is easily recognisable as removing files.
  - The tutorial symbol is “?” which is also a commonly used symbol for help.
  - We have basic instructions as placeholders in the textboxes to guide users.
- **Control**
  - The adding and removing of files allows users to freely add and remove files, so the users do not need to refresh the page to clear unwanted files.
- **Consistency**
  - We clearly label our textboxes with “known” and “unknown” to indicate to our users what exactly they should put into each textbox and we also show results consistently in this way.
- **Error Prevention**
  - If there are no files added or no text inputted, there will be an error message notifying users of this.
- **Recognition**
  - Most symbols such as “+”, “X”, “?” are easily recognisable, the compare button is a button with the “compare” text on it.
- **Flexibility**
  - Flexibility between adding text input or adding files. Can also mix and match between text / files. Furthermore, .txt, .docx and .pdf file types are all supported.
- **Minimalism**
  - Not much text or elements on screen before users click the “compare” button, only relevant information is needed. Tutorial is only shown if user decides to click “?”.
- **Recover**
  - Our website does not have many errors, the only one being the error message for no file/text input, written in plain English.
- **Help and Documentation**
  - The error message for no file/text input is written in plain language.

UX: how people feel about a product and their impression and how it feels to them and how they feel afterwards

Heuristic Evaluation

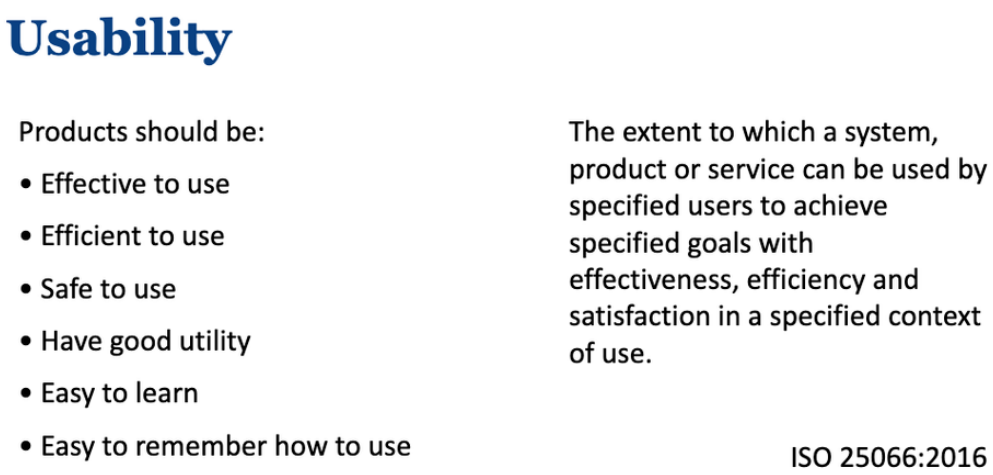
- **Visibility** - should always keep users informed about what is going on (ie appropriate feedback)
  - we should probably add a popup if their analysis runs over a certain amount of time
- **Match** - between system and real world (speak users' language, simple common language, follow real world conventions)
- **Control** - user control and freedom, support undo and redo

- Users often choose system functions by mistake and will need a clearly marked “emergency exit” to leave the unwanted state without having to go through an extended dialogue.

- **Consistency**
  - Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.
- Error Prevention
  - Even better than good error messages is a careful design which prevents a problem from occurring in the first place.
- Recognition - Make objects, actions, and options visible.
  - (!!) add an ℹ️ button to pull up instructions again when users need it?
  - “Instructions for use of the system should be visible or easily retrievable whenever appropriate.”
- Flexibility - Accelerators - unseen by the novice user - may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users.
- Minimalism - Dialogues should not contain information which is irrelevant or rarely needed
- Recover - Help Users Recognise, Diagnose, and Recover from Errors
  - error messages should include plain language expression, precisely indicate the problem, construct/suggest a solution
- Help and Documentation
  - Troubleshooting: Make sure your system indicates errors and messages in plain language

In line with user stories, ask questions:

- Will the user know what to do?
- Will the user see how to do it?
- Will the user understand from the feedback whether the action was correct or not?



UX: how people feel about a product and their impression and how it feels to them and how they feel afterwards

Deployment Process & CI/CD

The entire deployment of both the front end and back end was done on AWS (Amazon Web Service).

To ensure a successful deployment, the repository needed to be split into 2 repositories for the front and back end separately as they are built under different architectures and have different requirements for running.

To also ensure a successful CI/CD of maintaining both our code structure and to unit test our code, we implemented a linting and test case check for both repositories using **GitHub Actions** to validate our repositories before pushes or merges to the main branch.

Front End Deployment

The front end deployment was done through using **CodePipeline** to take the github repository, download the repository and then build the Angular app using **CodeBuild** which is then sent through to the **S3 Bucket** that is publicly shown.

This ensures a successful CI/CD as any changes to the main branch is automatically detected by CodePipeline and automatically pulls the new version and rebuilds the program and updates all the files in the S3 bucket. Any issues caused with an unsuccessful pull or build will just not update the current website and still have the old working version running so no issues are noticed for the user.

For the code structure checking, we are using the standard NodeJS linter that is included with NodeJS and is easily implementable with GitHub Actions.

For the code testing, thanks to our use of Angular, we are able to use **Angular**'s already supplied unit tester which can be done locally using the command `ng test`. We can simply include testing with our GitHub Actions pipeline by creating a testing build and running the `ng test` command to unit test and check the functionality of all components in the project.

Back End Deployment

Since the back-end deployment is a **Flask** web app that utilises the python package **tensorflow**, The usual method of *AWS ElasticBeanstalk* cannot be used for free as there isn't sufficient memory requirements for the app to be functional. As such the alternative method was to run the application on an **EC2 instance**. This allowed for more control at the expense of ease of use.

The process then was to manually set up the application at first by connecting to the EC2 instance and installing both git and python to the server and cloning the private github repository using the SSH cloning method. Afterwards the commands to setup running the server were:

```
git pull  # pull latest github repository

pip install -r requirements.txt  # installing dependencies

python -c 'import nltk; nltk. download(["punkt", "stopwords","wordnet"])'  # install nltk packages (run once)

export FLASK_APP=application.py  # export flask web app

nohup flask run --host=0.0.0.0
```

Which allowed for continuous running of the server.

To make sure the application would always automatically update with our latest repository, another timing script was created to check for any github repository changes every 10min and in the event of a change, for the commands to be rerun.

For the code structure we used pythons standard linter `pylint` to keep check of our coding.

For the process of unit testing, the built in `unittest` package was used as it integrates seamlessly with Flask allowing for local checking without needing to completely run the web app. This allowed for us to do a large number of tests on every potential file type making sure the correct response was outputted. This unittest process was then implemented in the GitHub Actions pipeline so that all changes or merges to the repository are checked by the linter and tester.

Website Updates

Website link:

AuthoWrite

New Github Repositories:

Back-end: https://github.com/Re-Roll/AuthoWrite-back-end Connect your Github account

Front-end: https://github.com/Re-Roll/AuthoWrite-front-end Connect your Github account

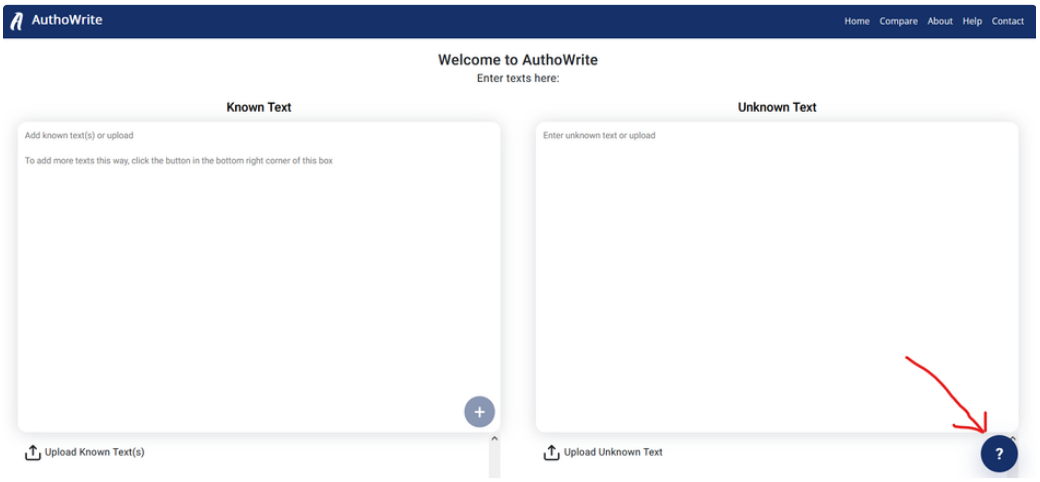
Changes Made Since

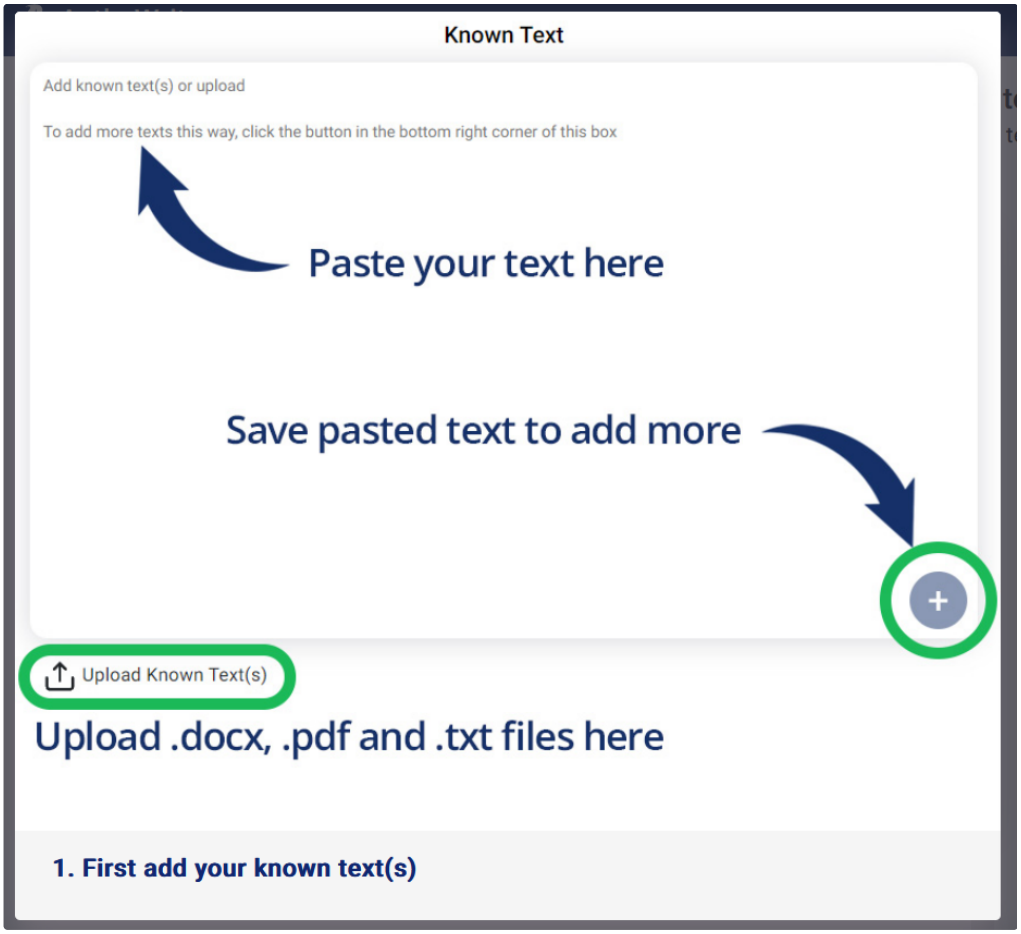
As we're nearing the end of the last sprint, here are some of the changes that have been made

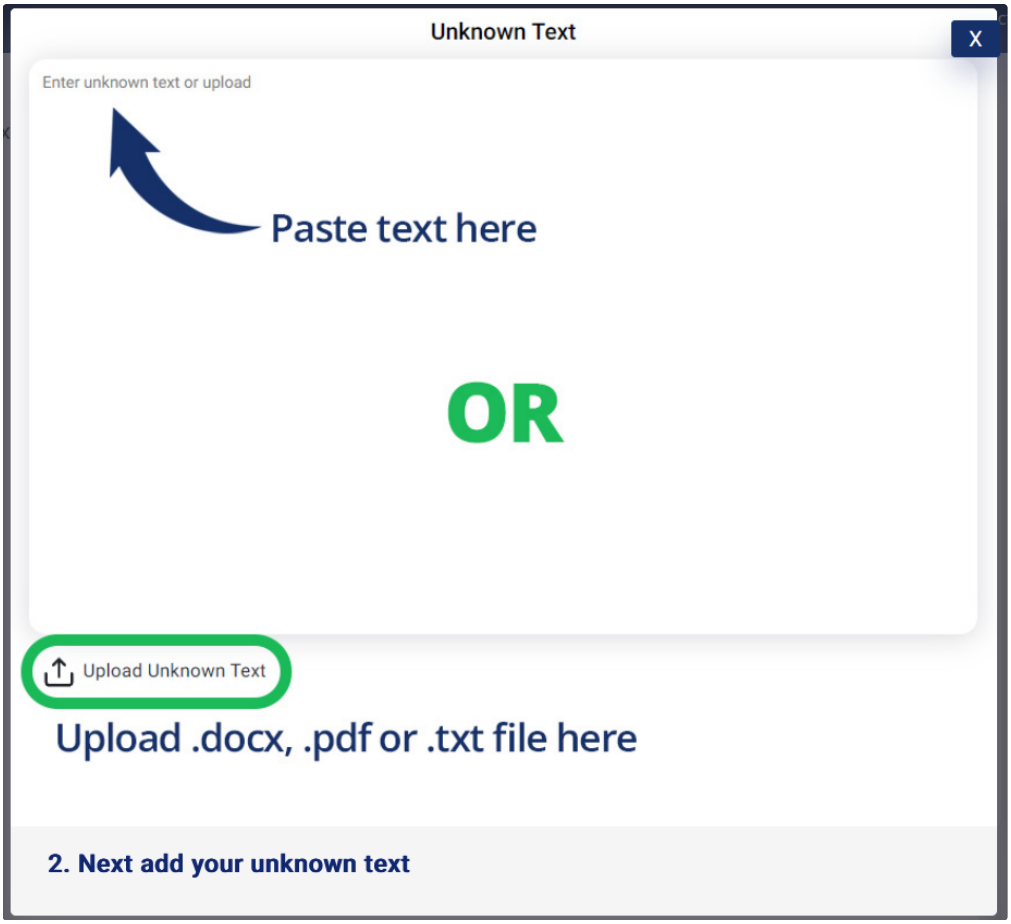
- **Features we have moved to future extension possibility/potential**
  - These features include the signup/login feature to create profiles. This is to maintain our focus on the functionality and improve upon the model that was given to us in order to meet the clients requirements
- **What has changed since sprint 2**
  - Additions:
    - Restyled website homepage with interactable background

- Created new git repositories for deployment (separated into front-end and back-end)
- Website deployment has now been complete and deployed with AWS
- A tutorial button has now been added which leads to a popup with visual instructions for aid
- Testing changes are now being implemented for the website

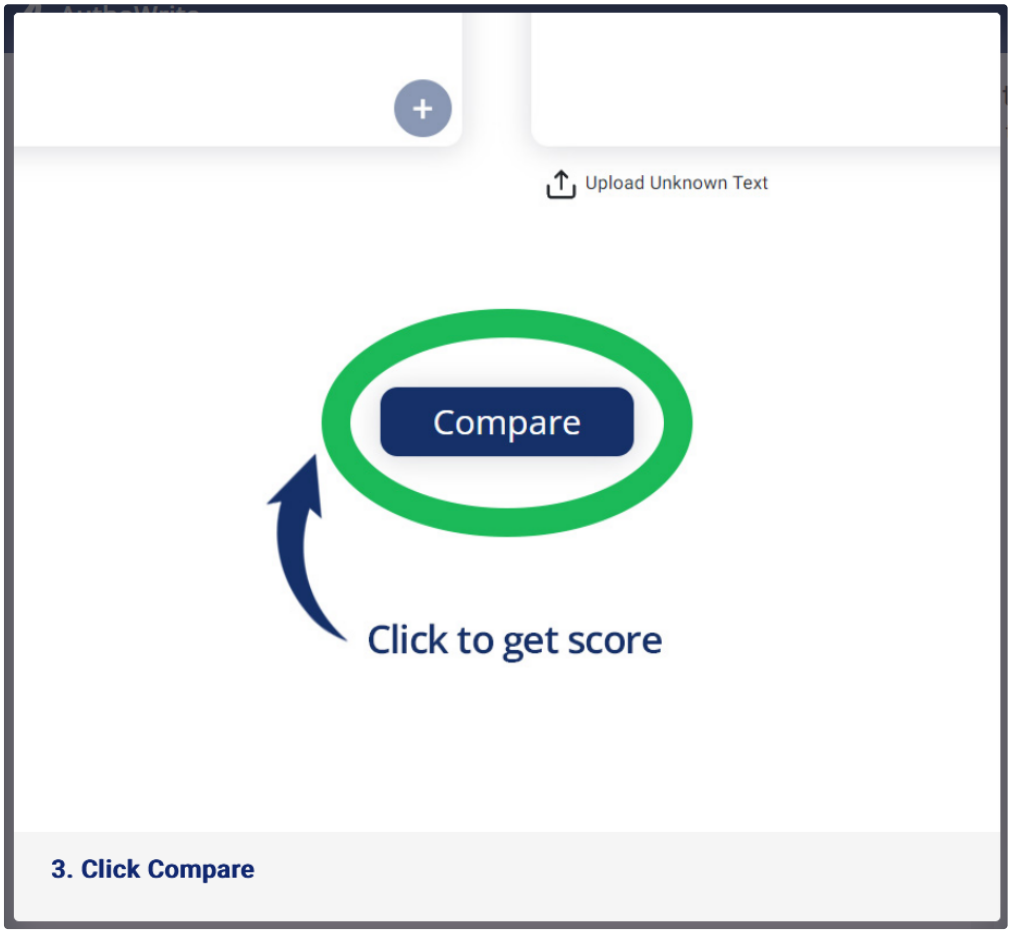
Tutorial Images



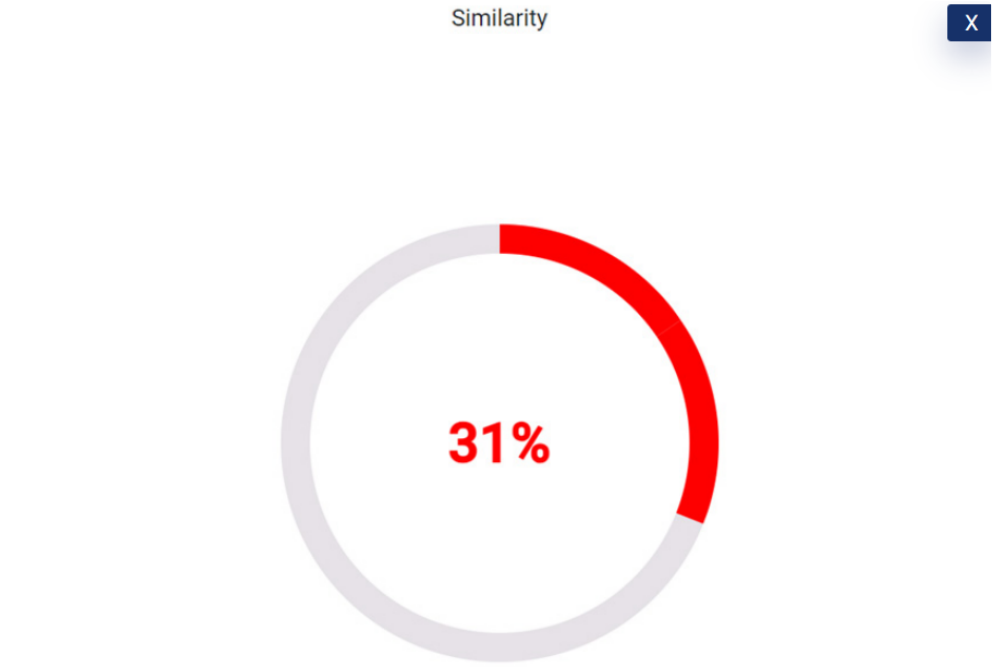




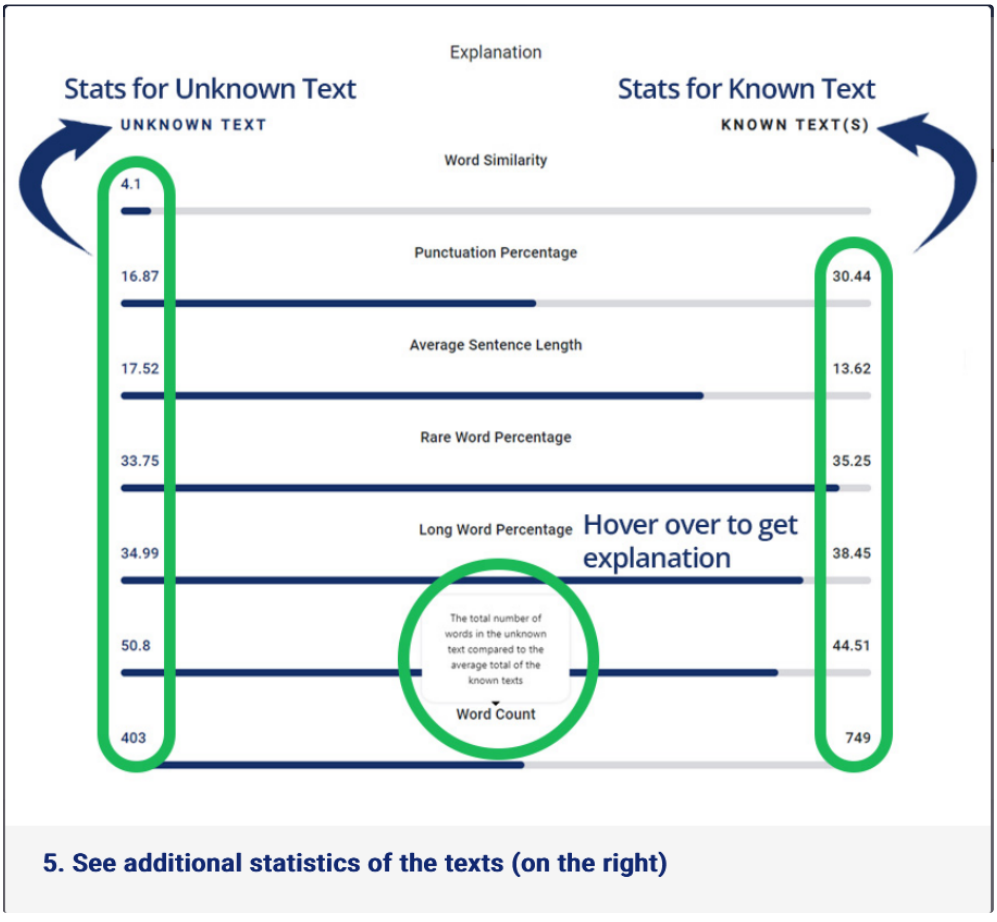
2. Next add your unknown text



3. Click Compare



4. See your result (on the left)



5. See additional statistics of the texts (on the right)

Potential Extensions

Given the time:

- Profile pictures
- Animation loading
- Users + Profile
- Profile Sharing