

Functional Specification Contents

Project Title	Infer-Read	
Student One	Ryan McQuillan	19452414
Student Two	Ethan Hall	19436514
Supervisor	Jennifer Foster	
Completion Date	11/11/2022	

Table of Contents

1. Introduction	3
1.1 Overview	3
1.2 Business Context	3
1.3 Glossary	4
2. General Description	4
2.1 Product / System Functions	4
2.2 User Characteristics and Objectives	5
2.3 Operational Scenarios	5
2.4 Constraints	5
3. Functional Requirements	6
3.1 Document Parsing and Natural Language Processing	6
3.1.1 Description	6
3.1.2 Criticality	6
3.1.3 Technical issues	6
3.1.4 Dependencies	6
3.2 Word Complexity Assessment Algorithm	6
3.2.1 Description	6
3.2.2 Criticality	7
3.2.3 Technical Issues	7
3.2.4 Dependencies	7
3.3 Providing Context Hints & Fetching Synonyms	7
3.3.1 Description	7
3.3.2 Criticality	7
3.3.3 Technical issues	7
3.3.4 Dependencies	7
3.4 Document Complexity Assessment Algorithm	8
3.4.1 Description	8
3.4.2 Criticality	8
3.4.3 Technical issues	8
3.4.4 Dependencies	8
3.5 Modifying & Developing Pipelines and NLP Components for Target Languages	8
3.5.1 Description	8
3.5.2 Criticality	8
3.5.3 Technical issues	8
3.5.4 Dependencies	9
3.6 Data Storage	9
3.6.1 Description	9

3.6.2 Criticality	9
3.6.3 Technical issues	9
3.6.4 Dependencies	9
3.7 User Authentication & Authorization	9
3.7.1 Description	9
3.7.2 Criticality	9
3.7.3 Technical issues	10
3.7.4 Dependencies	10
4. System Architecture	10
4.1 Webapp UI	10
4.2 Backend	11
5. High-Level Design	12
5.1 Web UI Overview	12
5.2 Component Structure	13
5.3 Reading Session Start Sequence Diagram	14
5.4 Reading Session Update Sequence Diagram	15
5.5 Reading Session Quit Sequence Diagram	16
6. Preliminary Schedule	17
7. Appendices	18
7.1 Frontend Mockup	18

1. Introduction

1.1 Overview

This project will make use of language prediction to identify words & phrases within an incoming, user-uploaded text, the purpose of which is to provide assistance in the learning of a language. It will make use of a user profile to store the learner's known words. The web app will perform frequency analysis to rank a text in terms of difficulty in comparison to a learner's word bank, highlighting previously unseen and unknown words for the learner to engage and try to disambiguate.

We want learners to learn via inference similar to how one learns as a language as a native; picking up an intuition of meaning via the context of the word usage by fetching synonyms and other instances of the word's usage provides a similar context, before providing the learner with an explicit translation.

Each learner will have a unique wordbank, initially through testing their awareness of common words and phrases by means of a simple examination. Each instance of a "reading session" will POST to the backend, updating the user's word bank if they declared a novel word as known or indicating difficulty with another word.

1.2 Business Context

We believe this tool could be incorporated as a strong component of an existing language learning tool, catering to a more advanced learner. There is a strong lack of reading tools online for a moderate to advanced learner - we want to capitalize on this lack.

For users of language tools such as Duolingo or Babbel, they could at best, read a short story written for the language tool.

This is dissatisfactory for numerous reasons;

- They are usually extremely basic.
- Not interesting reading material, e.g. story of a woman going to the bus stop.
- Limited by its purpose to cater to an extremely large audience e.g. it would not incorporate cultural elements, phrases, literature style writing, etc.

1.3 Glossary

API	Application programming interface
HTML	HyperText Markup Language
Sass / SassScript	Syntactically Awesome Style Sheets: a markup language interpreted or compiled into Cascading Style Sheets. Adds features such as indented syntax.
OCR	Optical Character Recognition
SPA	Single Page Application
POST	A request to the API that will create a new entry for the table.
PUT	A request to the API that will update an existing entry.
GET	A request to return table data, this data will be user specific. The headers in the request will hold tokens indicating which userID or documentID it will pertain to.
NLP	Natural language processing
Angular / NG	TypeScript-based web application framework
RabbitMQ	Message-broker software built on the Open Telecom Platform framework for clustering and failover

2. General Description

2.1 Product / System Functions

Infer-Read will be a language-learning tool targeted toward intermediate learners of French and/or Irish through English. The learner will be able to upload documents (pdf, txt, epub) that they wish to work through and the application will parse the document, providing the learner with a measure of the difficulty of the text relative to them; classified by occurrence of novel words, word/phrase complexity and frequency.

Learners will log-in in order to upload their documents. Each user will have a unique word bank containing all of the vocabulary that they're currently familiar with. This bank will be initialised via a simple introductory test to evaluate the user's current level and populate the bank.

The learner will be able to read through passages of their chosen text in a simple dynamic “page-like” UI, where any occurrences of new vocabulary are highlighted, prompting the user to hover over and select the word; allowing them to rate their understanding and add it to the word bank if known.

If they assess their understanding of the new word as poor or too unfamiliar, Infer-Read will provide instances of the word in other contexts and known synonyms of the new word allowing the learner to consider the meaning via inference.

2.2 User Characteristics and Objectives

Infer-Read is primarily aimed at intermediate language learners looking to improve their literacy in either French and/or Irish. The learner would be someone who enjoys reading and aims to learn via reading, improving their vocabulary and intuitive understanding of grammar.

Infer-Read assumes enough technical understanding to be able to operate a simple web-based application, upload documents in a suitable format and have some experience in their language of choice, although this is not a hard requirement.

2.3 Operational Scenarios

2.3.1 Educational Institutions(second-level, third-level)

Infer-Read would allow students to upload documents they may find challenging and more actively engage with the material in their learning efforts and make noticeable progress through a text. A student studying either language in college will come across texts that they struggle with, or may want to further their studies through other material they would enjoy but find too daunting.

2.4 Constraints

2.4.1 Knowledge & Previous Experience

Both team members have little to no background in the realm of natural language processing and machine learning, which are both significant components of the project. Infer-Read will also be built with a technology stack that we aren’t familiar with, which will require learning time ahead of development.

2.4.2 Hardware & Computing Power

The architecture of our backend design requires a decent amount of computing power to host if we want to deploy multiple containers to distribute and process requests and provide fair response times.

3. Functional Requirements

3.1 Document Parsing and Natural Language Processing

3.1.1 Description

Infer-Read must be able to parse a document and perform several different text normalisation and natural language processing tasks. These include tokenization, lemmatization and parts-of-speech tagging.

3.1.2 Criticality

These features are critical to even the very base functions of the application; having legible document pages, assessing word and document complexity etc.

3.1.3 Technical issues

Given how fundamental this is to the application, document parsing will need to be very efficient. There will also be unique cases to handle depending on how texts are laid out and what formats given documents are.

3.1.4 Dependencies

This is the first requirement of the application, so it has no dependencies.

3.2 Word Complexity Assessment Algorithm

3.2.1 Description

The application has to be able to discern the complexity of a word that is new to a learner, relative to their current level of ability, the frequency of the word and likelihood of occurrence in the given context.

3.2.2 Criticality

This feature is also essential to the functionality of the application, without it we cannot assess the complexity of documents and provide meaningful information and feedback to learners.

3.2.3 Technical Issues

We need to design an algorithm that can weight the complexity of a word in a way that is relative to multiple variables unique to both a user and a text. The potential solution could also involve the use of a language model to assess word probability through word masking (replacing the word with a [MASK] token and predicting what word would be likely to occur).

3.2.4 Dependencies

Document Parsing

3.3 Providing Context Hints & Fetching Synonyms

3.3.1 Description

Infer-Read needs to provide instances of novel words in different sentences and fetch synonyms of the word if the different instances are not enough for the user to form an understanding.

3.3.2 Criticality

This is the functionality required for the main concept of the application and so is extremely important for development.

3.3.3 Technical issues

We need to be mindful of the efficiency of fetching the information for each word; how long it takes to query this information, how we handle these calls and provide the instances (from where, storage, caching etc).

3.3.4 Dependencies

Document Parsing

3.4 Document Complexity Assessment Algorithm

3.4.1 Description

The application has to determine the complexity of an uploaded document based on its content, depending on the user's vocabulary.

3.4.2 Criticality

This feature is important to Infer-Read's implementation, but not as absolutely essential as the previous functionalities outlined.

3.4.3 Technical issues

This assessment needs to be performed quickly and presented to the learner before they approach the text. We need to decide on the criteria that compose a document's complexity and how we can judge that on a case by case basis.

3.4.4 Dependencies

Document Parsing, Word Complexity Assessment Algorithm, User Profiles

3.5 Modifying & Developing Pipelines and NLP Components for Target Languages

3.5.1 Description

As we go along in development, we will have to customise and modify third-party NLP libraries and machine learning pipelines to suit our needs in French and especially Irish. There are significantly less solutions available for Irish and so we will have to make modifications and develop our own solutions for different NLP tasks. For example, this may involve POS taggers and lemmatizers for Irish using existing datasets obtained from third parties.

3.5.2 Criticality

This requirement is ad-hoc and will depend on what gaps we need to fill to perform different tasks. For Irish, it will be more critical to the application's functionality.

3.5.3 Technical issues

This will depend on what specific components need to be developed when we encounter them.

3.5.4 Dependencies

This requirement does not depend on any others.

3.6 Data Storage

3.6.1 Description

The application needs to store data specific to each user. This includes a user's word bank, their uploaded documents, a stored state of a user's progress through a document as well as profile information and passwords. For documents, we need to keep their parsed representations and measures of complexity per word to avoid having to reprocess documents per visit

3.6.2 Criticality

Data storage is essential to the application for data persistence for users.

3.6.3 Technical issues

We need to design the solution for our container-based architecture to have persistent volumes and availability

3.6.4 Dependencies

None

3.7 User Authentication & Authorization

3.7.1 Description

Users need to have accounts that persist across sessions with a log-in process that can authenticate them appropriately. We also need to ensure that users only have access to their respective data and cannot access data from the database they're not supposed to.

3.7.2 Criticality

This functionality is necessary to users to store their vocabularies and documents for future use and record progress.

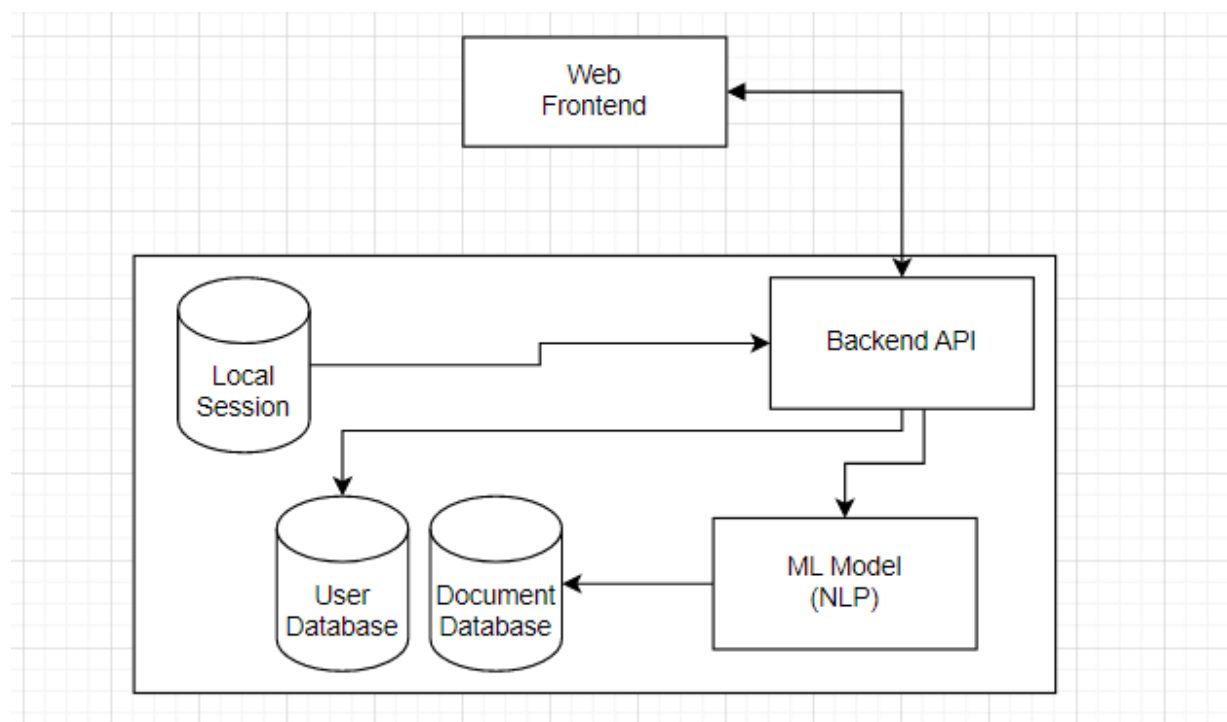
3.7.3 Technical issues

We need to ensure our method of authentication is secure and persists for the length of the user's session. We also need to ensure that our API endpoints can only be accessed by those with the authorization granted to them for that endpoint's functionality

3.7.4 Dependencies

Data Storage

4. System Architecture



4.1 Webapp UI

The web app is developed using Angular. This will provide the visuals for the user & will communicate with the API to communicate & make changes to any DB. Angular has the main principles of being extremely lightweight with the majority of business logic within the API, within the application however the majority of functions are in services or shared locations such as libraries. As such, the components themselves are designed to be extremely lightweight & slim. Only containing code & html that could only ever be used by the sole component.

Communication is done “on action completion” for InferRead, during which no heavy action (POST or PUT) is taken without user consent, e.g. Document Upload, Session Completion, Wordbank update, etc.

The main design follows SPA common practises,

- As a single page application, responsiveness is a huge focus. Making everything lightweight & not obstructing that is a main focus.
- Reusability, code within Angular can be heavily reused, via microservices, libraries & domains.
- A high user experience, by merit of being a single page application, the user is saved a heavy amount of cognitive load that involves navigating a multi-page application,
 - In addition, we will be taking steps to avoid common SPA pitfalls, we will keep navigation the app simple & have a constant list of the left hand side of the screen so the user is always aware of how to traverse the page.

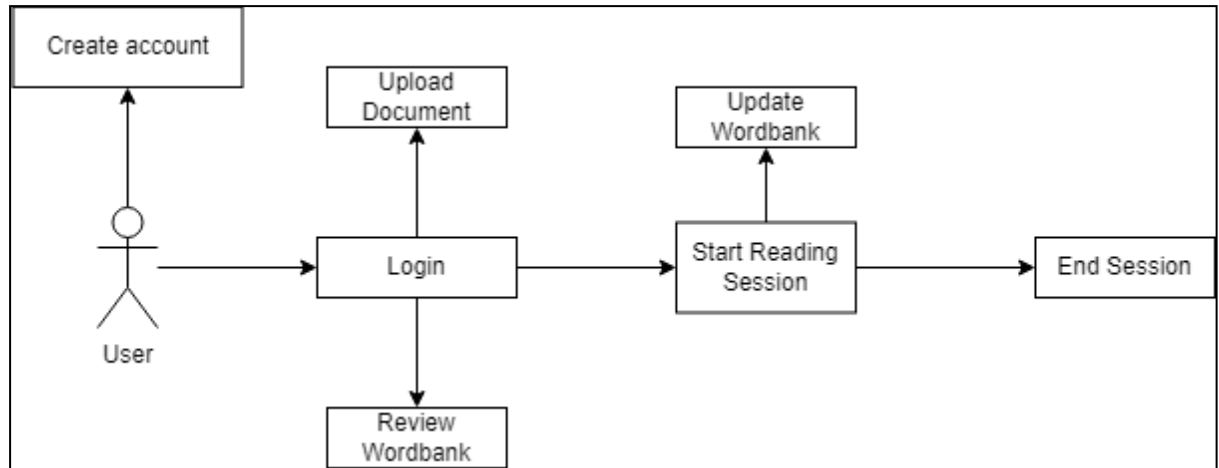
4.2 Backend

The backend will handle all of the logic of the application. It will be built using Node.js & Express.js. The frontend will send requests via a message queue(RabbitMQ) to a modified Python container where all of our NLP and ML tasks will be carried out depending on the endpoint. The backend will also be linked to a database container with persistent volumes.

The idea behind the backend architecture is that our application requests can be distributed evenly among the containers of our modified image through our message queue. This also allows for high-availability and fast response times.

5. High-Level Design

5.1 Web UI Overview

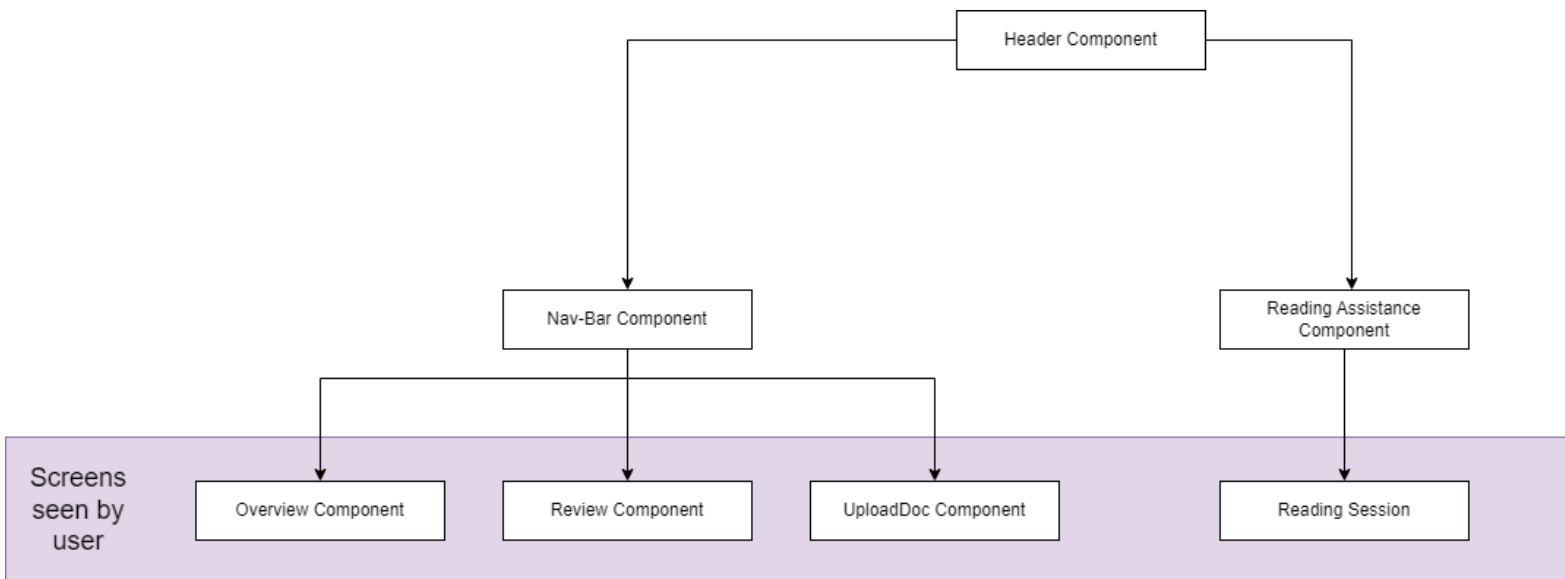


This diagram displays the possible user interaction, as the application is account based, the user must register or login before being able to interact with the reading capabilities. Our form of tracking progress is user-based, so it is required.

Then, once logged in the user has three options.

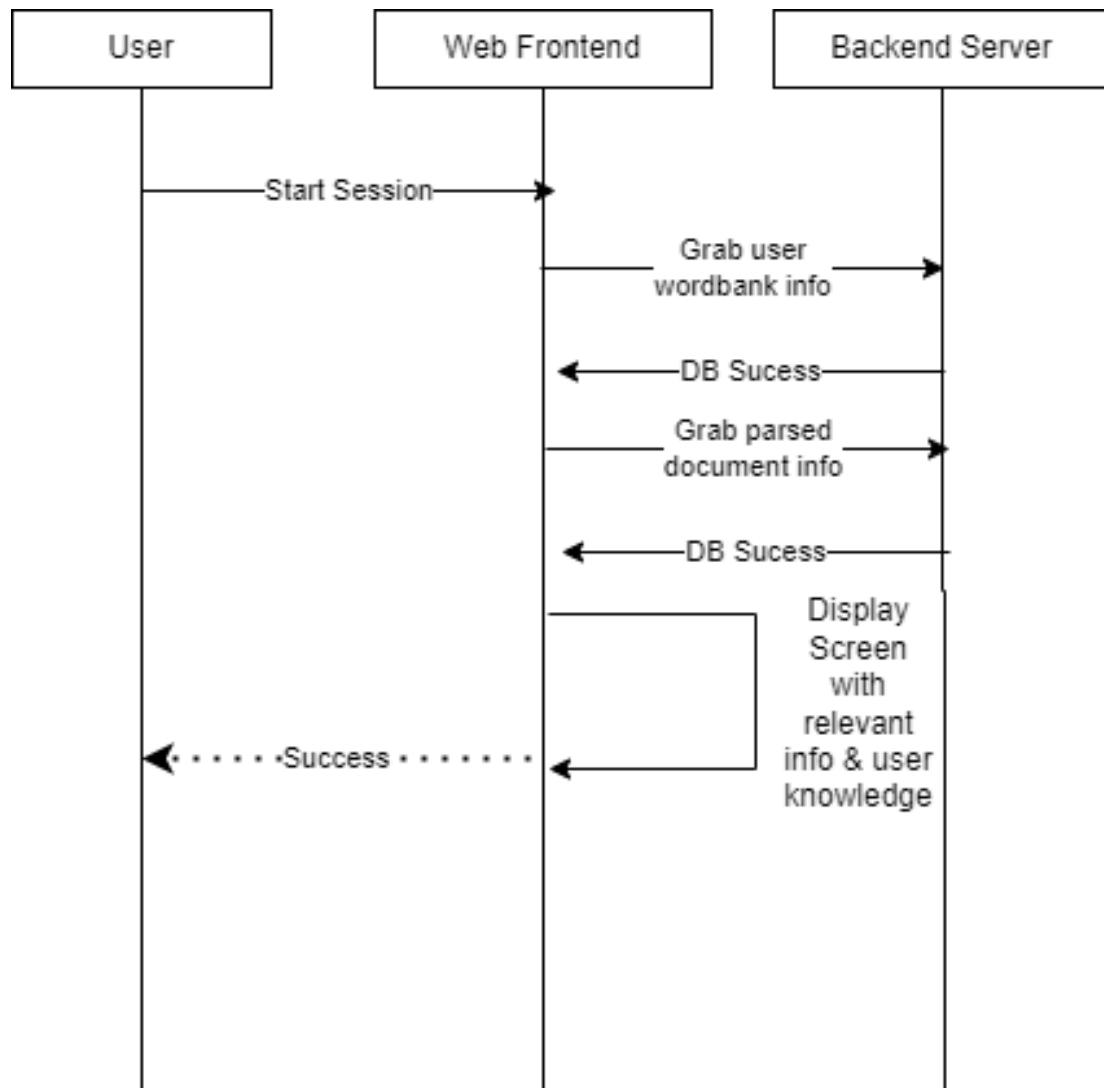
1. Review Wordbank: This displays words encountered & the user's level of knowledge for each word. Words can be Known (the user has knowledge of this word in most contexts) , Encountered (Has been seen before) , Low Knowledge (the user has knowledge of this word in some contexts), Troubled (the user has repeatedly asked for help with the word)
2. Upload Document: The user uploads a document that they intend to read, this is where the system mines the document, parses the words & phrases and puts the document in a format readable.
3. Start Reading Session: This is where the user begins reading, the words are able to be clicked for additional information, phrases & colloquialisms are singled out. Within this screen the user can make two choices.
 - 3.1. End Session: Intuitive, finish reading & go back to the previous screen.
 - 3.2. Update Wordbank: Through user input the system makes judgments on the users knowledge of each word, for instance if a user is consistently asking for help on one word, that word is marked as 'Troubled'.

5.2 Component Structure



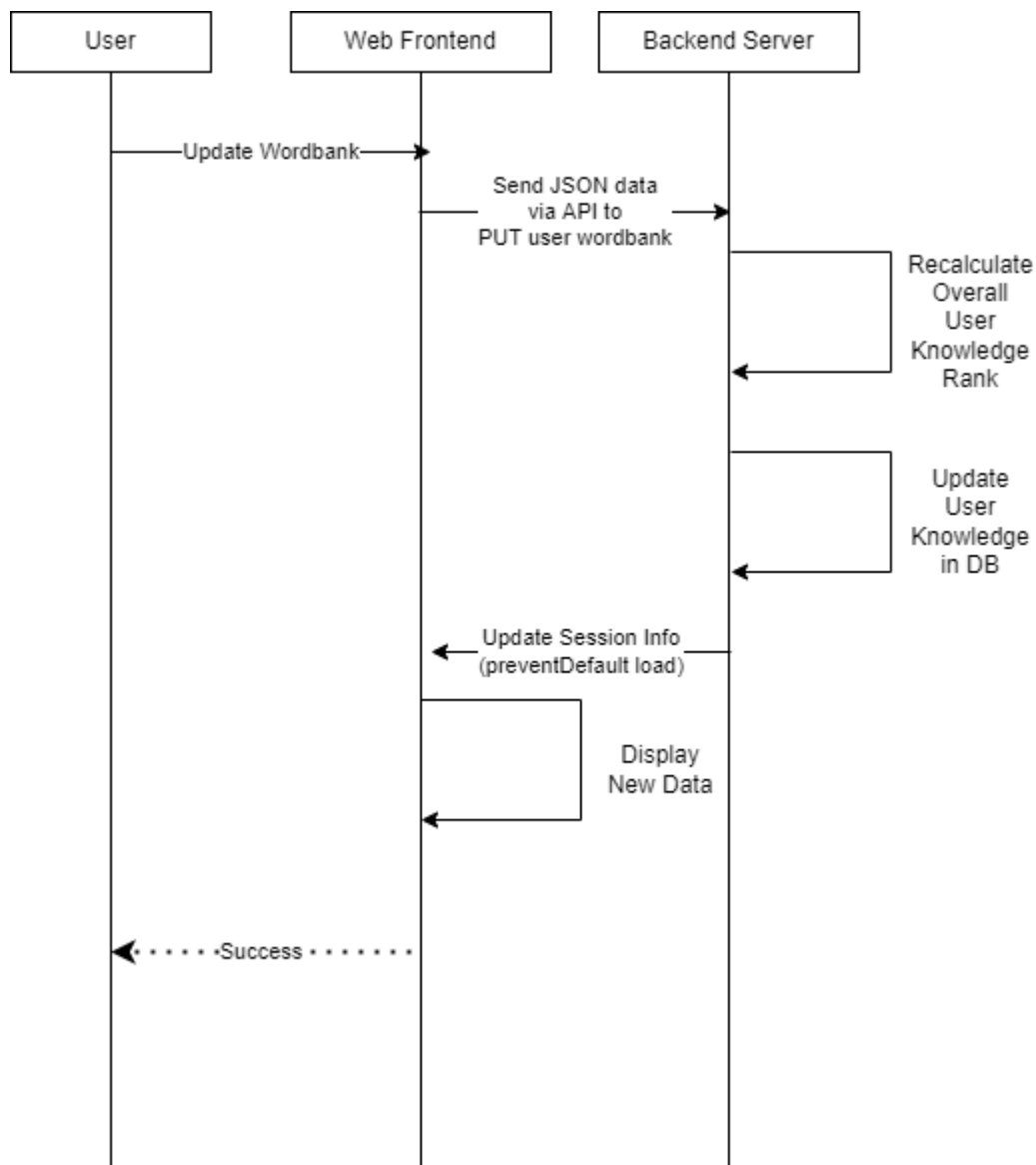
Within Angular, the components are divided as displayed above, as progress continues services will be made on an ad-hoc basis.

5.3 Reading Session Start Sequence Diagram



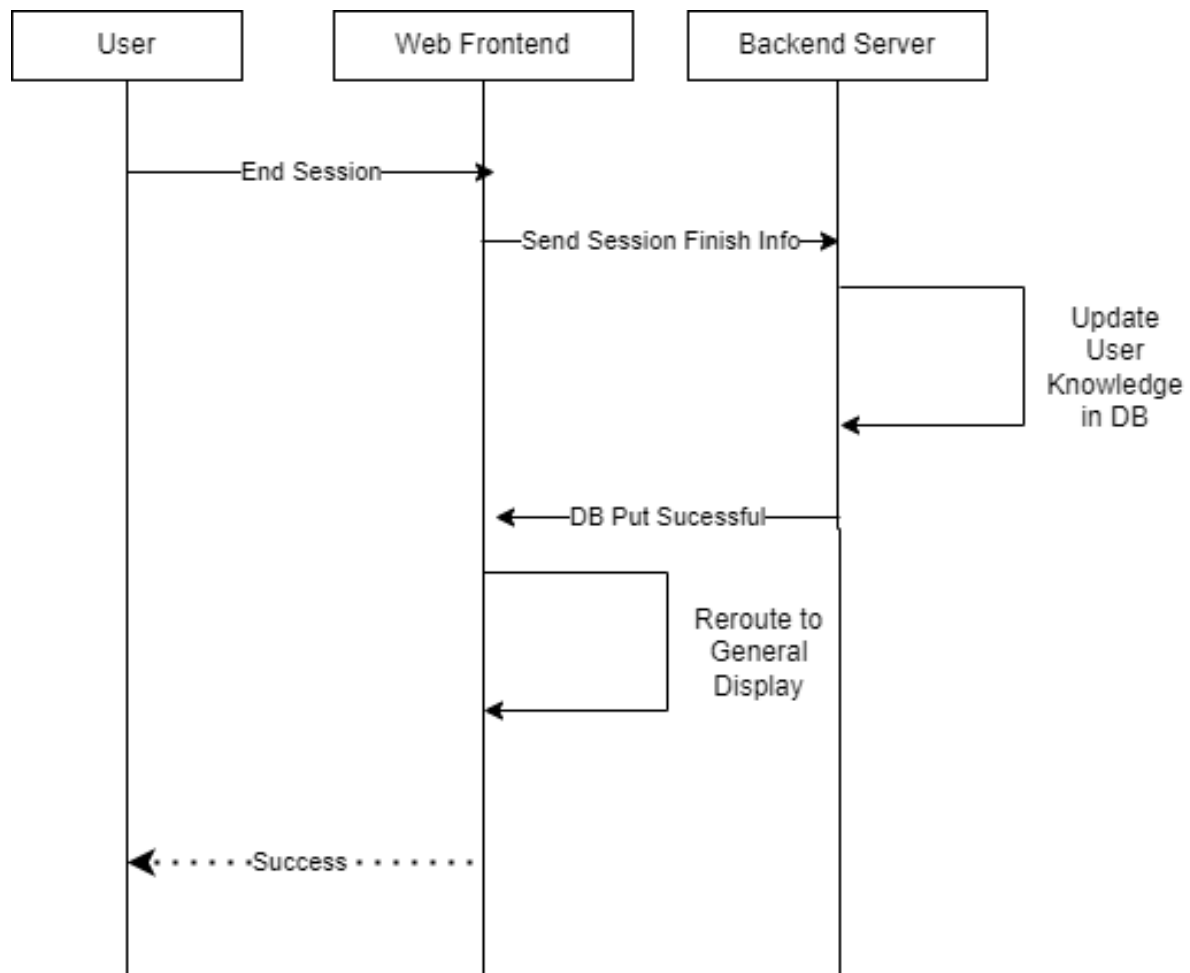
This diagram shows the communication between user input, frontend & the backend when 'Read' is selected. It is from the perspective of the visuals, as in how does the screen manage to change while remaining responsive. Essentially, the functions related to it are within the 'Reading Session' component, on route change calls to the API are sprung. A success would mean that the frontend successfully loaded & displayed all the info related in the document related to the user & to the document.

5.4 Reading Session Update Sequence Diagram



Within a reading session, the user may inform the system of their word knowledge. Either this be through private study or the system made an inaccurate estimation. We have a tool for the user to outright make the request. When the user states that for a word, *W*, is a knowledge state of *K*, (either 'Known', 'Encountered', 'Low Knowledge' or 'Troubled'), the frontend makes a PUT request for the user concerning *W* with a knowledge of *K*. There does not exist validation on choices as language knowledge is fluid.

5.5 Reading Session Quit Sequence Diagram



When finished, the user will request to end the session. In the frontend this appears to be a switch in visuals, but the frontend will send update the users knowledge on words encountered & mark the document as read. This could have a complete reevaluation of the users overall knowledge.

6. Preliminary Schedule

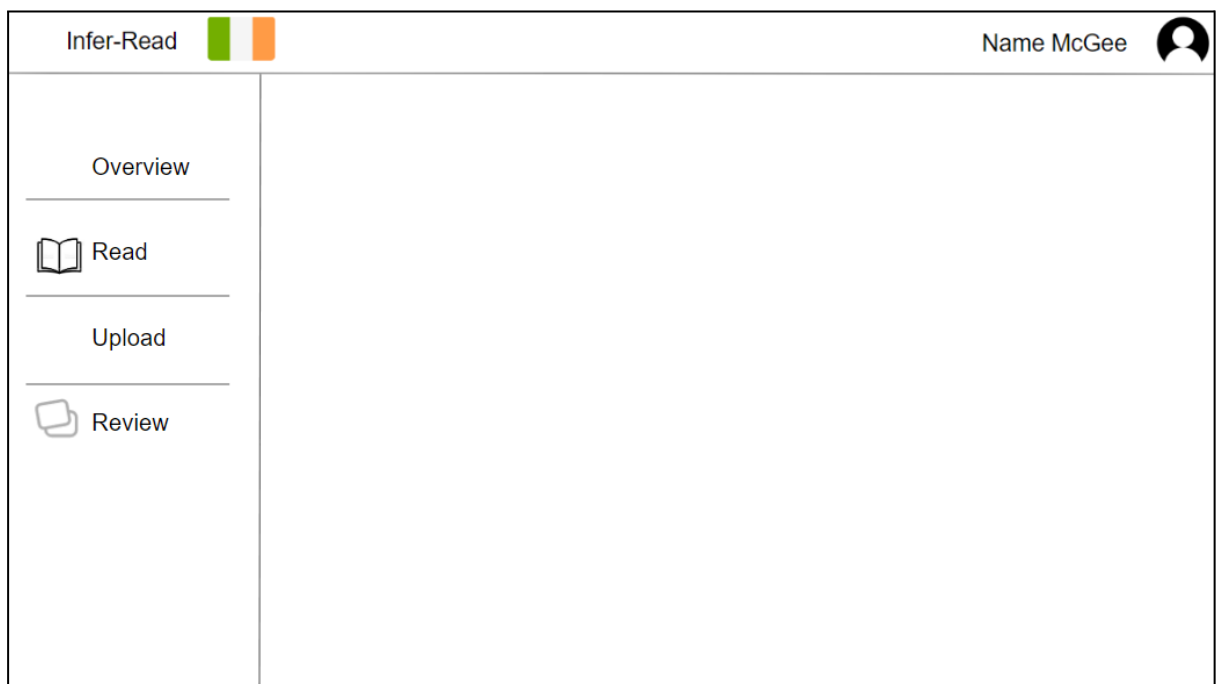
Item	Start Date	End Date	Assignee
OCR prototype	2022-10-20	2022-10-23	Ethan
Angular FrontEnd Prototype	2022-10-20	2022-11-10	Ryan
Functional Spec	2022-11-01	2022-11-16	Both
Flask Research + Dev	2022-11-17	2022-11-25	Ethan
RabbitMQ Research + Dev	2022-11-23	2022-11-30	Both
RabbitMQ Implementation with fake API	2022-11-29	2022-12-05	Ryan
Database Design & Implementation, User Authentication & Authorization	2022-12-06	2022-12-16	Both
Deployment of image containers, API testing	2022-12-20	2022-12-30	Ethan
NLP Research	2023-01-10	2023-02-10	Both
Development & Testing of NLP pipelines through SpaCy	2023-01-20	2023-02-10	Both
Word & Document Complexity Algorithm Development & Testing	2023-01-25	2023-02-20	Both
UI Testing	2023-02-21	2023-02-28	Ryan
Deployment	2023-02-21	2023-02-28	Ethan
User Testing	2023-03-01	2023-03-31	Both
Bug Fixes & Testing	2023-03-01	2023-03-31	Both
Documentation	2023-04-01	2023-04-27	Both
Final date for all project materials	2023-04-28		




7. Appendices


Specifies other useful information for understanding the requirements.

7.1 Frontend Mockup



Screen design after login, used by Overview, Upload 7 Review

Infer-Read

Name McGee

What is Lorem Ipsum?

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Why do we use it?

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like).

Where does it come from?

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going

Dummy

Synonyms

- Idiot
- Fool
- Jester

Need More ?

Screen design for the reading session