



Design & Innovation Project (DIP)

Project Report

Learnify.ai

**School of Electrical and Electronic Engineering
Academic Year 2020/21
Semester 1**

Contents

Chapter 1	Purpose/ Project Objectives	1
1.1	Objectives	1
1.1.1	Machine Learning	2
1.2	Platform Interface	2
1.3	Application Programming Interfaces (API).....	3
1.3.1	User Authentication API.....	3
1.3.2	User Registration API.....	3
1.3.3	File Upload API	3
1.3.4	Summarize API.....	4
Chapter 2	Project Summary.....	4
2.1	Discussion of problems.....	4
2.2	Problems Solved	5
2.3	Unsolved problems	6
Chapter 3	Scope.....	7
3.1	Front-end.....	7
3.2	Back-End.....	15
3.3	Data Processing.....	17
3.3.1	Development Process.....	21
Chapter 4	Schedule.....	23
4.1	Differences Between the Planned and Actual Schedules	23
Chapter 5	Cost	25

Chapter 6	Research.....	25
6.1	Keyword Extraction	26
6.1.1	General Approaches	26
6.1.2	Keyword Extraction Models	27
6.2	Text Summarization	29
6.3	Visualization of Knowledge Graph	30
Chapter 7	Outcomes / Benefits	33
7.1	Outcomes	33
7.2	Benefits	35
7.2.1	A more efficient form of studying	35
7.2.2	Better visualization	36
7.2.3	More Interactive	36
Chapter 8	Reflection	38
8.1	Engineering knowledge	38
8.2	Problem Analysis	39
8.3	Solution development	39
8.4	Individual/Teamwork.....	40
8.5	Future Recommendations	41
Chapter 9	References	44

List of Figures

Figure 1. Original Webpage	7
Figure 2. Login Page	8
Figure 3. Home Page.....	9
Figure 4. Home Page (Front)	9
Figure 5. Courses and Lectures visual display	10
Figure 6. Routes for Platform	10
Figure 7. Home page code	11
Figure 8. Sign In Page Code	11
Figure 9. Log Out Page code	12
Figure 10. Course and Lecture Page Code.....	12
Figure 11. Creating of Course Graph.....	13
Figure 12. Create of Lecture Graph Code.....	13
Figure 13. Rendering of Graph	14
Figure 14. Site Map of Learnify.ai.....	14
Figure 15 Schematic Diagram of Backend	15
Figure 16. Back End API	16
Figure 17. PDF Reader.....	16
Figure 18. Base BERT model performance on a test document	17
Figure 19. Fine-tuned BERT model performance on a test document	18
Figure 20. Fine-tuned BERT model performance based on recommended settings on test document	18

Figure 21. Yake! key term extraction output on same test document.....	19
Figure 22. Keyword Extraction Class Overview	20
Figure 23. Keyword Extraction and Knowledge Graph Input Generation	20
Figure 24. Summarization Function code	21
Figure 25. Example of Keyword extraction by RAKE.....	26
Figure 26. Keywords Extraction with TextRank	28
Figure 27. Example of a Knowledge graph (force graph)	31
Figure 28. Knowledge Graph.....	33
Figure 29 Study Page	34
Figure 30. Problem analysis on why student is studying inefficiently	39

List of Tables

Table 1 BERT fine tune training results	19
Table 2 Milestone	23
Table 3 Cost Table	25

Chapter 1 Purpose/ Project Objectives

Different people have their own way of learning, some of which are memorising content from lecture notes and others might be practicing on tutorials or on past year papers. Moreover, there are students who just blindly reads the entire content, which resulted them losing the idea of linking key topics. To help enrich students learning experience, Learnify.ai was created.

Learnify.ai is a full stack software development project with applied deep learning and hence has a wide set of requirements to be met in terms of implementation. It serves as a web application for students to get a visual representation of the key topics in the reference materials and provides quick search options to look up any chapters in the key topics list along with its summary. Upon reading, it aims to help students by summarizing relevant notes from the database and using keywords to identify the important notes.

1.1 Objectives

The core features of this project platform are topic modelling, archiving, key information extraction and summarization which is to be displayed onto a knowledge graph. These will be briefly elaborated in the following points.

1.1.1 Machine Learning

On the machine learning aspect, there are two tasks involved in the project. First, keyword extraction, which extracts the keywords from the documents uploaded by the user and creates the knowledge graph for the document. Keyword Extraction is a function that automatically detects and pull-out words, phrases, sentences that is important and shows the main gist of what the text document is about.

Second, text summarization, is a methodology which automatically summarizes texts into two-three lines that only contains the main points of the course/lecture which makes it faster to lookup during revision.

Keywords and Summary Extraction displays the meaningful information from the human language text files in pdf format, without requiring domain knowledge and thus present the inputted lecture notes in the form of a summary.

Expected week of completion: Week 7

1.2 Platform Interface

The Platform Interface consists of three key components: Frontend, middleware and backend.

For frontend development, we intend to use ReactJs (in addition to HTML and CSS) and using Flask API as middleware to handle all the data flow. The platform uses MongoDB [7] database (NoSQL database) to store and manage data for each user profile. In addition to the keyword extraction and summarization, the platform needs to have various essential components such as user authentication, user registration and file upload feature for better user interaction.

Expected week of completion: Week 8-11

1.3 Application Programming Interfaces (API)

Data flow is critical for the functioning of entire project. To facilitate the data flow, APIs are used for synchronizing the frontend and the backend for all the functions like user authentication, user registration and requesting of user-profile data. These APIs will have to do cross checking with the database, mongoDB, where these information are stored. As a result, which will help to train the data processing model.

Expected week of completion: Week 9

1.3.1 User Authentication API

The key function of the authentication API is user authentication which takes the user's email and password as parameters. Which will be used to verify the user credentials with the database to validate the login session.

1.3.2 User Registration API

The key function of the user registration API is creating a new profile for the user which takes user's email, username and password as parameters. All these parameters are to be entered at the signs-up interface. After successfully completing the user registration, the user will be instantly be able to login to the platform.

1.3.3 File Upload API

File upload API is critical to the functioning of the project as it handles the file uploaded by the user and stores the data in the mongoDB database. Upon successful upload, the FLASK backend automates the entire keyword extraction and summarization pipeline and updates the database asynchronously.

Chapter 2 Project Summary

2.1 Discussion of problems

One of the 14 grand challenges for engineering in the 21st century is Advanced personalized learning as outlined by an international group of leading technological thinkers ^[1]. This connects to the topic for this project ‘Artificial Learning’ which refers to the use of artificial intelligence and machine learning techniques for the purposes of helping students learn. Therefore, the aim of this project was to improve one aspect of personalized learning.

Firstly, in order to come up with the problem statement, the team collectively discussed and came up with possible problems that students might face. The most common problems faced by students were the difficulty of locating the most important points from within a stack of lecture notes, the difficulty of connecting different concepts together and the lack of engagement with the material.

Revising the key concepts is an important task while studying, especially before an exam, during revision when a student has limited time and requires a quick and easy fix for brushing up on key concepts which they have already learned. As engineering students, while preparing for exams we need to spend more time on solving problems and less on reading through lecture notes, however, currently a lot of time gets wasted and reading through material students already know.

Moreover, since in the current curriculum content is taught on a weekly basis, as students it

becomes difficult for us to connect the different concepts together. For example, in some modules concepts from week 4 are further expanded upon and connected to the material in week 10. Therefore, linking different concepts together was the second problem faced.

The third problem was a lack of engagement with the material. Simply reading from lecture notes or watching Learning Activity Management Systems (LAMS) can become a monotonous process. It reduces the engagement of the user with the material and often students will get distracted while reading. Lecture slides are usually not visually appealing as well which also makes it harder to focus.

2.2 Problems Solved

These problems will be effectively solved by Learnify.ai. It presents the user with a knowledge graph which would provide them with a better visualization of the concepts and as well as linking the relevant topics together. These tools would aid the student during their exam time revision as the knowledge graph uses Natural Language Processing (NLP) to provide an at-one-glance view of the contents of a module as well as how the different concepts are linked. The relationship between the different key terms is also visualized as the distance between the nodes represent how closely related the topics are. The knowledge graph is also interactive as the user can click on a node in the knowledge graph and receive a summary of what the key term means. Users can also choose to delete nodes which they have already studied. Thus, Learnify.ai acts as a study topic checklist while revising for examinations.

2.3 Unsolved problems

Although the two main problems identified during the brainstorming session were solved through this project. However, there will still be problems faced by students and therefore there are different ways to optimize the process of learning. Further improvements were discussed but due to time constraints, they were not implemented. This includes making the knowledge tree more interactive which would further help engage the user in the material they were studying. The graph could also be color-coded such as to highlight the importance of the different concepts of the lecture notes. This would serve to create an even more informative and appealing visual experience for the user. Furthermore, the application is also unable to understand formulae at this stage.

In conclusion, the main goals of the project were successfully met since we created a web application which solved the problems of locating the most important points from within a stack of lecture notes and the problem of connecting different concepts together.

Chapter 3 Scope

Our group was split into 3 smaller sub-groups consisting of front-end team, back-end team and data processing team. Each sub-team will have to interconnect their programs for the user to input their lecture notes/materials and view the key terms of lecture notes in the form of knowledge graph as well as a summary of each key term.

3.1 Front-end

In the front-end team targets on creating webpage designing, the presentation of knowledge graphs, uploading of documents from the user and making the webpage user friendly with introduction videos.

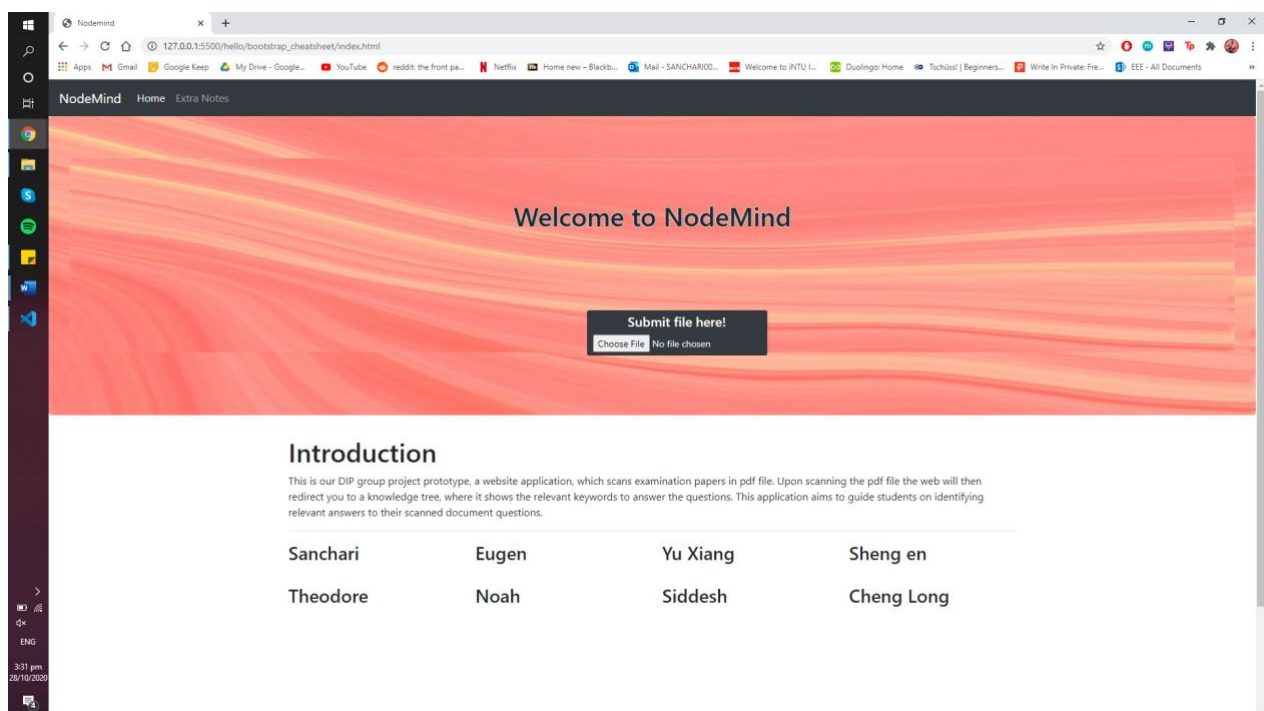


Figure 1. Original Webpage

As shown in Figure 1, the initial idea of the interface is simple and user friendly. But after much consideration about leaking of school notes or materials, we decided to add security functions using Application Programming Interface (API). Due to the changes made, we were unable to use bootstrap, instead now we are using react which is a different library to create the front-end interface. Changes that were made by the front-end were adding of downloaded history features, security function and subscription page.

Learnify.ai

Learnify.ai

Learnify.ai is a web application that scans lecture notes to identify the key terms and uses them to generate a knowledge tree.

By clicking on the knowledge tree the user can access summaries to these key terms.

This would help the user prepare more efficiently during examinations as well as better visualize the content they need to study.

Login

Email address

siddesh@gmail.com

We'll never share your email with anyone else.

Password

.....

Login Sign Up

Figure 2. Login Page

Learnify.ai

Login

Learnify.ai

Sign Up

Username

Email address

Password

Confirm Password

Sign Up

Figure 3. Home Page

Learnify.ai

Home

Courses

Study

Logout

Welcome siddesh !

Add Documents

Upload List

Choose file

Browse

Upload Files

courses ~

Figure 4. Home Page (Front)

```
NodeMind > client > src > JS App.js > ...
1  import React, { Component } from "react";
2  import { BrowserRouter, Switch, Route } from "react-router-dom";
3  import Login from "../pages/login/login";
4  import SignUp from "../pages/signup/signup";
5  import Home from '../pages/home/home';
6  import Courses from '../pages/courses/courses';
7  import Profile from '../pages/profile/profile';
8  import ExamPrep from '../pages/exam/exam';
9
10 class App extends Component {
11   render() {
12     return (
13       <BrowserRouter>
14         <Switch>
15           <Route path="/study" component={ExamPrep} />
16           <Route path="/profile" component={Profile} />
17           <Route path="/courses" component={Courses} />
18           <Route path="/login" component={Login} />
19           <Route path="/signup" component={SignUp} />
20           <Route path="/" component={Home} />
21         </Switch>
22       </BrowserRouter>
23     );
24   }
25 }
26
27 export default App;
```

Page | 10


```

class Home extends Component {
  state = {
    email: '',
    userdata: {},
    debug: '',
    uploads: [],
    uploadfiles: [],
    currentCourse: '',
    error: false,
    addCourse: false,
    userCourses: '',
    uploading: false
  }

  componentWillMount() {
    if (sessionStorage.getItem("debug") !== undefined && sessionStorage.getItem("userdata") !== undefined) {
      this.setState({
        email: sessionStorage.getItem("email"),
        userdata: JSON.parse(sessionStorage.getItem("userdata")),
        debug: sessionStorage.getItem("debug"),
        reload: sessionStorage.getItem("reload")
      })
    }
  }

  onChangeHandler = event => {
    this.state.uploads.push({
      'file': event.target.files[0],
      'course': this.state.currentCourse
    })
    this.setState({uploadfiles: [...this.state.uploadfiles, event.target.files[0].name]})
  }

  handleCourseError = (file) => {
    return file
  }

  handleUpload = (event) => {
    this.state.uploads.map(
      (file) => {
        // console.log(file)
        const data = new FormData();
        data.append('file', file.file);

        if (file.course !== '') {
          let url = 'http://0.0.0.0:10000/extract?course=' + String(file.course) + '&file=' + String(file.name);
          this.setState({
            uploading: true
          });
          fetch(url, {
            method: 'PUT',
            body: data,
          }).then(
            (response) => {
              response.json().then((body) => {
                console.log('API debug code:', body.code, file)
                let url = 'http://0.0.0.0:10000/integrate?email=' + String(this.state.email) + '&password=' + String(this.state.password);
                fetch(url, {
                  method: 'GET',
                  dataType: 'json'
                }).then(res => res.json()).then(data => {
                  sessionStorage.setItem('reload', true)
                  this.setState({uploading: false})
                })
              })
            }
          )
        } else {
          this.setState({
            error: true
          })
          return 0;
        }
      })
  }
}

```

Figure 7. Home page code

```

class Navbar extends Component {
  state = {
    signupPage: false,
    loginPage: false,
    debug: '400'
  }

  handlelogout = () => {
    this.props.logout();
  };

  handlelogin = () => {
    this.setState({
      signupPage: true
    })
  }

  loggedIn = () => {
    return (
      <React.Fragment>
        <div className="container">
          <nav className="navbar navbar-expand-lg navbar-dark bg-dark">
            <a href="/" className="navbar-brand">
              NodeMind
            </a>
            <button
              className="navbar-toggler"
              type="button"
              data-toggle="collapse"
              data-target="#navbarSupportedContent"
              aria-controls="navbarSupportedContent"
              aria-expanded="false"
              aria-label="Toggle navigation"
            >
              <span className="navbar-toggler-icon"></span>
            </button>
            <div className="collapse navbar-collapse" id="navbarSupportedContent">
              <ul className="navbar-nav mr-auto">
                <li className="nav-item active">

```

Figure 8. Sign In Page Code

```

84
85   loggedOut = () => {
86     return (
87       <React.Fragment>
88         <nav className="navbar navbar-expand-lg navbar-dark bg-dark">
89           <a className="navbar-brand" href="/">
90             NodeMind
91           </a>
92           <button
93             className="navbar-toggler"
94             type="button"
95             data-toggle="collapse"
96             data-target="#navbarSupportedContent"
97             aria-controls="navbarSupportedContent"
98             aria-expanded="false"
99             aria-label="Toggle navigation"
100           >
101             <span className="navbar-toggler-icon"></span>
102           </button>
103
104           <div className="collapse navbar-collapse" id="navbarSupportedContent">
105             <ul className="navbar-nav mr-auto">
106               <li className="nav-item active"></li>
107             </ul>
108
109             <form className="form-inline my-2 my-lg-0">
110               {this.state.loginpage ? null : <button
111                 className="btn btn-outline-success my-2 my-sm-0"
112                 type="submit"
113                 onClick={this.handlelogin}
114               >
115                 Login
116               </button>}
117             </form>
118           </div>
119         </nav>
120       </React.Fragment>
121     );
122   };
123
124   componentWillMount(props){
125     if (this.props.loginpage){

```

Figure 9. Log Out Page code

```

12   class Courses extends Component {
13     state = {
14       // ...
15     }
16
17     handleSelection = (event) => {
18       // console.log('event' => s, event.target.id)
19       this.setState({
20         currentCourse: event.target.id,
21       })
22     }
23
24     console.log('course selected')
25     for(var tmp in this.state.userdata.courses){
26       if(this.state.userdata.courses[tmp].course_name === event.target.id){
27         const graph = this.state.userdata.courses[tmp].courseGraph
28         this.setState({
29           currentGraph: graph
30         })
31         console.log('graph set', graph)
32         break;
33       }
34     }
35     // userdata["courses"] -> "currentCourse" -> courseGraph -> set currentGraph
36
37     componentWillMount(){
38       this.setState({
39         email: sessionStorage.getItem('email'),
40         userdata: JSON.parse(sessionStorage.getItem("userdata")),
41         debug: sessionStorage.getItem('debug'),
42         width: window.innerWidth,
43         height: window.innerHeight
44       })
45       var courses = JSON.parse(sessionStorage.getItem("userdata"))["courses"]
46
47       // Structure of the cache
48       // {
49       //   'course_name': {
50       //     '0': <graph>
51       //   }
52       // }
53
54       var cus = {}
55
56       // ...
57
58       render() {
59         if (this.state.debug !== "200"){
60           return <Redirect to={pathname: '/login'} />
61         }
62         var reload = sessionStorage.getItem('reload')
63         if(reload === false){
64           this.handleReload()
65         }
66
67         if (this.state.userdata.courses.length > 0){
68           var courses = this.state.userdata.courses
69           var courseNames = courses.map(
70             (course) => {
71               if (this.state.currentCourse === course.course_name){
72                 return (
73                   <button className="list-group-item list-group-item-action active"
74                     key={course.course_name} type="button" id={course.course_name} onClick={this.handleSelection}
75                     >{course.course_name}</button>
76                 )
77               }
78               return (
79                 <button className="list-group-item list-group-item-action"
80                   key={course.course_name} type="button" id={course.course_name} onClick={this.handleSelection}
81                   >{course.course_name}</button>
82               )
83             }
84         )
85
86         if (this.state.currentCourse !== ""){
87           var query = []
88           for(var tmp in this.state.userdata.courses){
89             if (this.state.userdata.courses[tmp].course_name === this.state.currentCourse){
90               query = this.state.userdata.courses[tmp].lectures
91               break;
92             }
93           }
94
95           // this.setState({currentGraph: query[0].graph})
96           console.log('lecture list', query)
97           var renderLectures = query.map(
98             (lecture) => {
99               return (
100                 <button className="dropdown-item"
101                   key={String(lecture.lecture)+this.state.currentCourse} type="button" id={

```

Figure 10. Course and Lecture Page Code

```

11 sp = spacy.load('en_core_web_sm')
12 all_stopwords = sp.Defaults.stop_words
13 pp = pprint.PrettyPrinter(indent=4)
14
15 from transformers import BartTokenizer, BartForConditionalGeneration, BartConfig
16 model = BartForConditionalGeneration.from_pretrained('facebook/bart-large-cnn')
17 tokenizer = BartTokenizer.from_pretrained('facebook/bart-large-cnn')
18
19 > def summarise(text): ...
29
30 > def summarizeCourse(lectureDocs): ...
63
64 > def top_3(keywords): ...
86
87 > def isSubstring(str1:str, dictionary:list) -> bool: ...
104
105 > def createGraph(keywords): ...
125
126 > def getGraphs(lectures): ...
143
144 > def clean(text:str): ...
151
152 > def mergeGraphs(graphs): ...
180
181 > def similarity(kw1, kw2, nlp): ...
189
190 > def parseGraph(keywordList, keywordDict, sparseGraph): ...

```

Figure 11. Creating of Course Graph

```

192 def parseGraph(keywordList, keywordDict, sparseGraph):
193     parseData = {
194         "nodes": [],
195         "links": []
196     }
197     nodes, links = [], []
198     for kw in keywordList:
199         # print(kw)
200         sample = {
201             "id": kw,
202             "group": keywordDict[kw]["group"]
203         }
204         nodes.append(sample)
205     parseData["nodes"] = nodes
206     # print(parseData)
207     # print(sparseGraph)
208     nlp = spacy.load("en_core_web_lg")
209     print('\nCreating Sparse Graphs')
210     for i, nn in tqdm(enumerate(sparseGraph)):
211         for j, indicator in enumerate(nn):
212             if keywordList[j] == keywordList[i]:
213                 continue
214             elif indicator == 1:
215                 # 10 is arbitrary value
216                 value = similarity(keywordList[i], keywordList[j], nlp) * 5
217                 sample = {
218                     "source": keywordList[i],
219                     "target": keywordList[j],
220                     "value": int(value)
221                 }
222                 # print(sample)
223                 links.append(sample)
224     parseData["links"] = links
225     return parseData
226
227

```

Figure 12. Create of Lecture Graph Code

```

6  class Graph extends Component {
7
8      state = { width: this.props.width, height: this.props.height };
9
10 >  updateDimensions = () => {--
12      };
13 >  componentDidMount() {--
15      }
16 >  componentWillUnmount() {--
18      }
19
20 >  handleClick = node => {--
31      };
32
33      fgRef = createRef();
34
35      render() {
36          if (this.props.data){
37              return (
38                  <React.Fragment>
39
40                      <ForceGraph3D
41                          ref={this.fgRef}
42                          onClick={this.handleClick}
43                          height="100vh"
44                          width={0.667*this.state.width}
45                          graphData={this.props.data}
46                          nodeAutoColorBy="group"
47                          nodeThreeObject={node => {
48                              const sprite = new SpriteText(node.id);
49                              sprite.color = node.color;
50                              sprite.textHeight = 8;
51                              return sprite;
52                          }}
53                          // linkWidth={1}
54                      />
55                  </React.Fragment>
56              );
57          }
58          else{
59              return null
60          }
61      }

```

Figure 13. Rendering of Graph

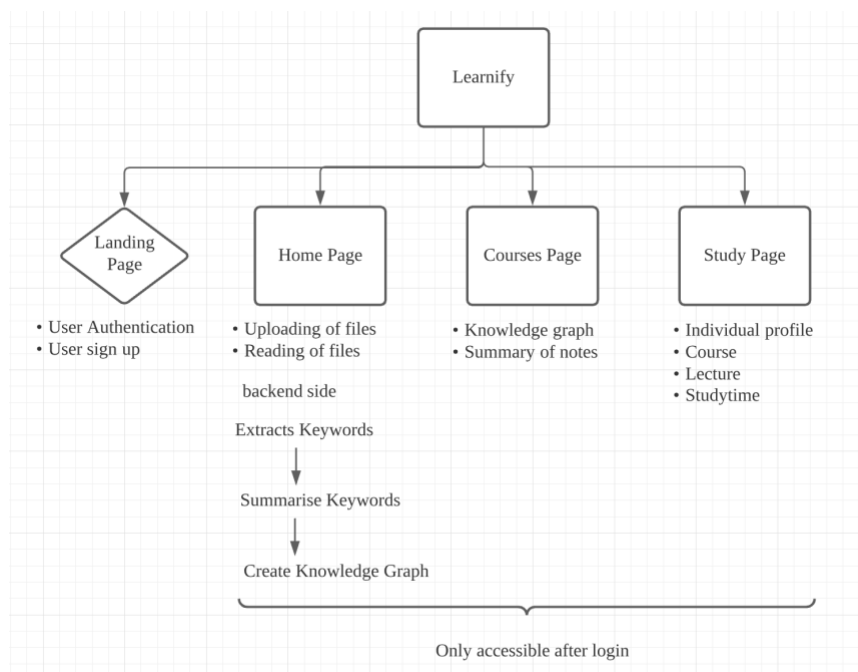


Figure 14. Site Map of Learnify.ai

3.2 Backend

For the user to use the web application, the user must be able to access the stored notes in the platform. In order to do so, all these notes must be stored into a storage space. MongoDB being a documented-oriented database program was used as a storage database for this web application. Where the user's notes and personal information will be stored at.

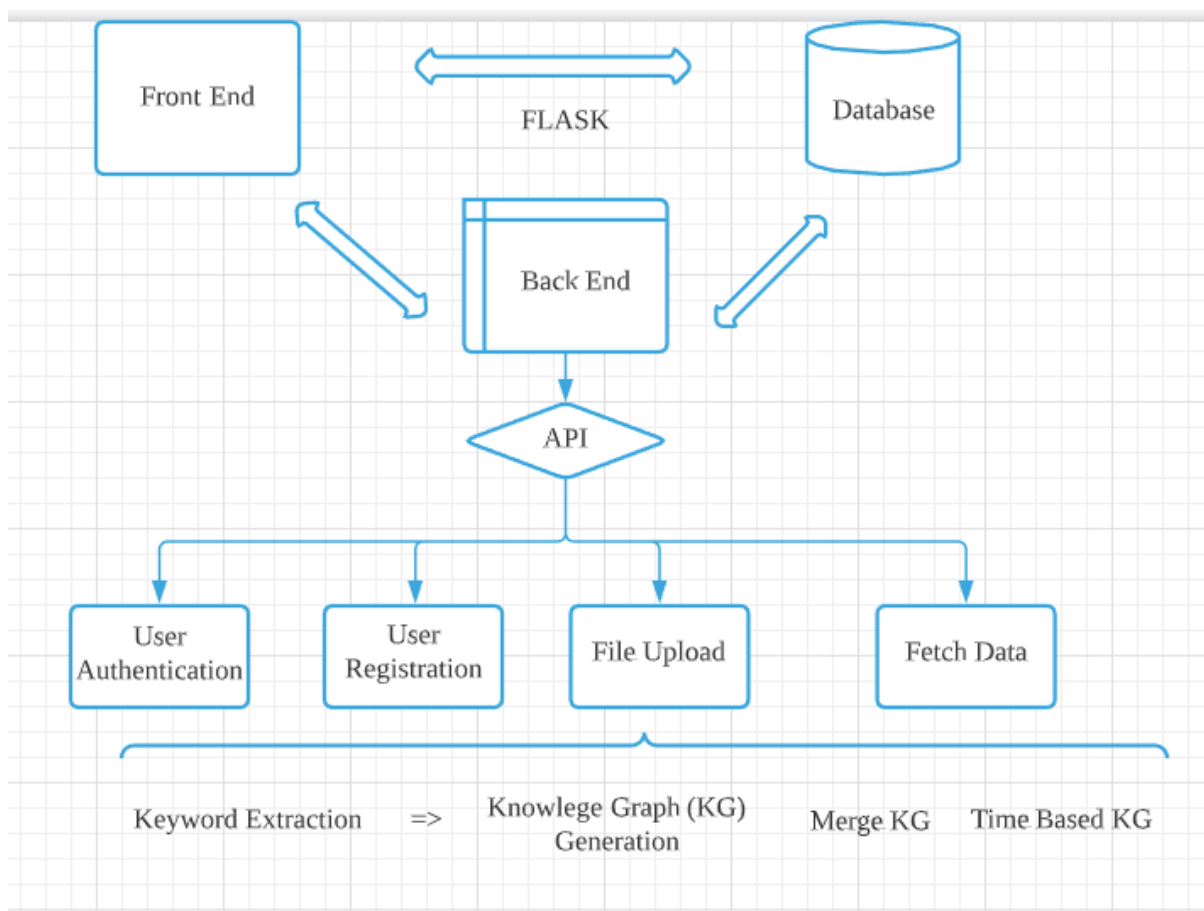


Figure 15 Schematic Diagram of Backend

```

37 # Set the secret key to sign the JWTs with
38 app.config['JWT_SECRET_KEY'] = 'e4aa2564fec8a421232c895233838f3f66ab305ef508e47597b27ff744b6131'
39
40 > class keywordExtractor:--
197
198 @app.route('/summarise', methods=['GET'])
199 > def summariser():--
209
210 @app.route('/addcourse', methods=['PUT'])
211 > def addCourse():--
232
233 @app.route('/deleteUser', methods=['PUT'])
234 > def deleteUser():--
236
237 @app.route('/getData', methods=['GET'])
238 > def getData():--
250
251 @app.route('/auth', methods=['GET'])
252 > def authenticate():--
286
287 @app.route('/integrate', methods=['GET'])
288 > def integrateGraphs():--
331
332 @app.route('/extract', methods=['PUT'])
333 > def upload_files():--
400
401 @app.route('/signup', methods=['PUT'])
402 > def signup():--
449
450 port = int(os.environ.get("PORT", 10000))

```

Figure 16. Back End API

```

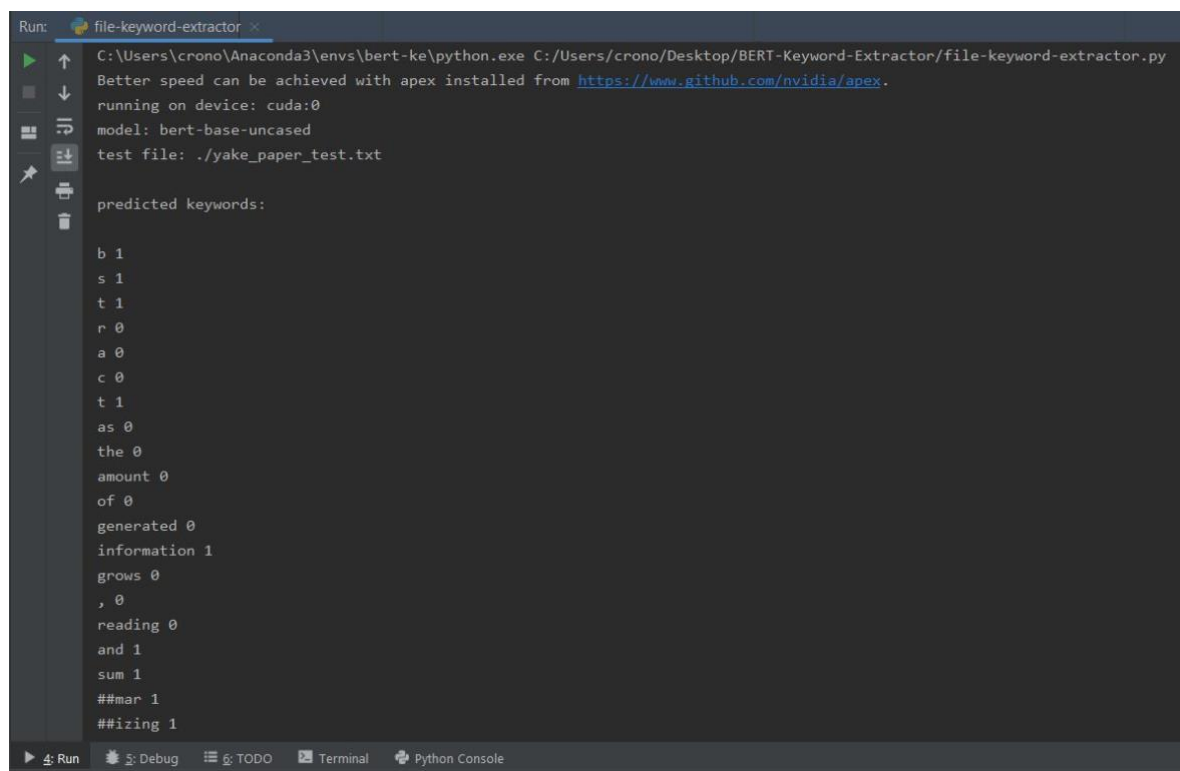
6
7 class pdfReader:
8
9     def __init__(self, path:str):
10         ''' Initiates the pdf reader object for the given file'''
11         assert len(path) != 0
12         self.pdfFileObj = open(path, 'rb')
13         self.pdfReader = PyPDF2.PdfFileReader(self.pdfFileObj)
14         self.pages = self.pdfReader.numPages
15
16     def getText(self, start:int, end:int=None):
17         '''
18         Args:
19             start -> Starting page number (index starts from 1)
20             end -> Ending page number
21         Return:
22             Dict with page number as key and the text content as value.
23         '''
24         end = self.pages
25         pages = dict()
26         for i in range(start-1, end):
27             pageObj = self.pdfReader.getPage(i)
28             pageContent = pageObj.extractText()
29             pageContent = ' '.join(pageContent.split('\n'))
30             pages[i] = pageContent
31
32         return pages

```

Figure 17. PDF Reader

3.3 Data Processing

The data processing team worked on sourcing, developing and integrating data processing methods for generating keywords, summarizing and identifying relational links in keywords in the lecture materials uploaded into the back-end database. For our keyword/key-term extraction, to ensure the extraction accuracy, we experimented with various models, statistical, graph-based and machine learning before settling on a statistical unsupervised method. Using the keyword extracted from the materials, relevant sentences relating to a keyword are extracted from the lecture notes and fed into the summarizer for a summary of the keyword topic, providing a brief explanation on each topic. The generated keywords, summary and keyword relational links are then passed to the front-end force graph (knowledge-graph) to be displayed in the interface.



```

Run: file-keyword-extractor
C:\Users\crono\Anaconda3\envs\bert-ke\python.exe C:/Users/crono/Desktop/BERT-Keyword-Extractor/file-keyword-extractor.py
Better speed can be achieved with apex installed from https://www.github.com/nvidia/apex.
running on device: cuda:0
model: bert-base-uncased
test file: ./yake_paper_test.txt

predicted keywords:

b 1
s 1
t 1
r 0
a 0
c 0
t 1
as 0
the 0
amount 0
of 0
generated 0
information 1
grows 0
, 0
reading 0
and 1
sum 1
##mar 1
##izing 1

```

Figure 18. Base BERT model performance on a test document

```

Run: file-keyword-extractor x
C:\Users\crono\Anaconda3\envs\bert-ke\python.exe C:/Users/crono/Desktop/BERT-Keyword-Extractor/file-keyword-extractor.py
Better speed can be achieved with apex installed from https://www.github.com/nvidia/apex.
running on device: cuda:0
model: ./model_3e-5_4ep.pt
test file: ./yake_paper_test.txt

predicted keywords:

light 0
- 1
weight 1
un 1

Process finished with exit code 0

```

Figure 19. Fine-tuned BERT model performance on a test document

lr=3e-5, 4 epochs

Dataset: SemEval 2010 dataset (scientific publications)

```

Run: file-keyword-extractor x
C:\Users\crono\Anaconda3\envs\bert-ke\python.exe C:/Users/crono/Desktop/BERT-Keyword-Extractor/file-keyword-extractor.py
Better speed can be achieved with apex installed from https://www.github.com/nvidia/apex.
running on device: cuda:0
model: ./model_5e-5_4ep.pt
test file: ./yake_paper_test.txt

predicted keywords:

of 0
generated 1
a 0
light 0

Process finished with exit code 0

```

Figure 20. Fine-tuned BERT model performance, recommended settings on test document

lr=5e-5, 4 epochs

Dataset: SemEval 2010 dataset (scientific publications)

Learning rate = 3e-5

Epoch = 4

Validation loss: 0.04686414202054342

Validation Accuracy: 0.9845987654320988

	Validation F1-Score: 0.5079191698525396
Learning rate = 5e-5	Validation loss: 0.04605963922761105
Epoch = 4	Validation Accuracy: 0.985796753543667
	Validation F1-Score: 0.5419082762256194

Table 1. BERT fine tune training results

```

Anaconda Prompt (Anaconda3)
(yake) C:\Users\crono\Desktop\yake>python main.py
score          keyword
-----
0.0004885630921598115  terms
0.0004994039937721685  keywords
0.0005353872469425369  candidate term
0.0006310081783928718  information sciences
0.0007567217818415125  term
0.0007915577869370038  keyword extraction
0.0009532286383719708  gold keywords
0.0011240140056544782  yake
0.0012806280734158114  candidate keyword score
0.0017068000962101738  relevant keywords
0.001839158840831863  information
0.0021135280965713  relevant terms
0.002160312812771908  document
0.002568103096375974  keyword extraction method
0.002890928382338881  automatic keyword extraction
0.00294906294035247  term frequency
0.0032422902644645077  text
0.003434932742564172  results
0.003489681685549752  extraction
0.003690349062292603  term gold keywords
0.003719109875338473  keyphrase extraction
0.003726770774767964  candidate term identification
0.0038256564536375466  score
0.003957121704531865  sciences
0.004017382246097735  relevant
0.004075936712692306  term score
0.004232009154296841  pasquali
0.00424084867971466  language
0.00427122084350529  irrelevant terms
0.00434021172509808  unsupervised keyword extraction
0.004356736665949733  mangaravite
0.00469993242949074  computer science
0.004756934082103318  keyword extraction algorithm
0.004872413546667808  feature
0.0048804501022559664  science
0.004935653975834673  information retrieval
0.004956558852628715  methods
0.005068667543571819  single terms
0.005214780379557325  section
0.005223124314896487  similar candidate keywords
0.005390697730125633  scores
0.00550906811524551  campos
0.0055353296427756585  keyword extraction systems
0.005640951099062552  table
0.0062481335257275105  gold
0.006252961898658712  keyword score

```

Figure 21. Yake! key term extraction output on same test document

(Lower score represents higher confidence)

```

NodeMind > backend > test > keyword.py > ...
103
104 @staticmethod
105 def intersect(keywords1, keywords2):
106     """
107     :param keywords1: list of kw from model 1
108     :param keywords2: list of kw from model 2
109     :return: common keywords list
110     """
111     intersect = set(keywords1) & set(keywords2)
112     keywords = list(intersect)
113     return keywords
114
115
116 @staticmethod
117 def similarity(kw1, kw2):
118     """
119     multi-gram word vector = average vector of words
120     :return: normalised similarity between 2 key terms score
121     """
122     doc1 = nlp(kw1)
123     doc2 = nlp(kw2)
124     return doc1.similarity(doc2)
125
126 def node_link(keywords, lect_num, sparse_graph, file):
127     """
128     format force graph dataset to .json file

```

```

NodeMind > backend > test > keyword.py > ...
4 from utils import top_3
3 from typing import List
4 from sklearn.neighbors import NearestNeighbors
5 import numpy as np
6 import spacy
7 import os
8
9 class keywordExtractor:
10
11 > def __init__(-
38
39 @staticmethod
40 > def isSubstring(str1:str, dictionary:List) -> bool:-
57
58 > def createGraph(self):-
78
79 @staticmethod
80 > def top_3(keywords):-
103
104 @staticmethod
105 > def intersect(keywords1, keywords2):-
115
116 @staticmethod
117 > def similarity(kw1, kw2):-
125
126 > def node_link(keywords, lect_num, sparse_graph, file):-

```

Figure 22. Keyword Extraction Class Overview

```

NodeMind > backend > test > keyword.py > ...
11
12 class keywordExtractor:
13
14     def __init__(
15         self,
16         path:str,
17         language:str = "en",
18         max_ngram_size:int = 3,
19         numOfKeywords:int = 50,
20         deduplication_threshold:float = 0.9,
21         deduplication_algo:str = 'seqm',
22         windowSize = 1,
23         k=3):
24
25         self.language = language
26         self.max_ngram_size = max_ngram_size
27         self.numOfKeywords = numOfKeywords
28         self.deduplication_threshold = deduplication_threshold
29         self.deduplication_algo = deduplication_algo
30         self.windowSize = windowSize
31
32         self.custom_kw_extractor = yake.KeywordExtractor(
33             lan=language, n=max_ngram_size, dedupLim=deduplication_threshold,
34             dedupFunc=deduplication_algo, windowsSize=windowSize,
35             top=numOfKeywords, features=None)
36
37         self.alg = spacy.load(f"{os.getcwd()}/nlp")

```

```

NodeMind > backend > test > keyword.py > ...
60
61 def createGraph(self):
62     keywords = self.custom_kw_extractor.extract_keywords(self.te
63     list1 = []
64     list2 = []
65     for i, kw in enumerate(keywords):
66         print(kw)
67
68         list1.append((i, kw[1])) # list w index number
69         if not keywordExtractor.isSubstring(kw[1], list2):
70             list2.append(kw[1]) # list w just keywords
71
72     graph = top_3(list2)
73
74     for i, nn in enumerate(graph):
75         print('\n---', list2[i], '---')
76         for j, indicator in enumerate(nn):
77             if list2[j] == list2[i]:
78                 continue
79             elif indicator == 1:
80                 print(list2[j])
81
82 @staticmethod
83 def top_3(keywords):
84     """
85     unnormalised vector used to calculated knn.

```

Figure 23. Keyword Extraction and Knowledge Graph Input Generation

```

10
11 sp = spacy.load('en_core_web_sm')
12 all_stopwords = sp.Defaults.stop_words
13 pp = pprint.PrettyPrinter(indent=4)
14
15 from transformers import BartTokenizer, BartForConditionalGeneration, BartConfig
16 model = BartForConditionalGeneration.from_pretrained('facebook/bart-large-cnn')
17 tokenizer = BartTokenizer.from_pretrained('facebook/bart-large-cnn')
18
19 def summarise(text):
20     print('[!] Tokenizing texts')
21     inputs = tokenizer([text], max_length=1024, return_tensors='pt')
22     print('[!] Summarising texts')
23     summary_ids = model.generate(
24         inputs['input_ids'],
25         num_beams=6, early_stopping=True,
26         max_length=10000, min_length=1000
27     )
28     return [tokenizer.decode(g, skip_special_tokens=True, clean_up_tokenization_spaces=False) for g in summary_ids][0]
29

```

Figure 24. Summarization Function code

3.3.1 Development Process

During the development, a machine learning transformer model (BERT [1]) was pretrained and tested. However, the open-sourced model seemed to perform poorly as seen in Figure 17 – 19. Hence, statistical, graphics methods were explored. TF-IDF [2] was tested as a baseline statistical approach, however, TF-IDF require more than one document to extract keywords from, due to its inherent need for inverse document frequency. Our objective requires keyword extraction based on single document hence was not a viable method. We ultimately finalized on statistical method named YAKE! [3].

Next, for every keyword, sentences containing the keyword is retrieved, producing a collection of sentences relating to the keyword. To produce a summary of this collection, we require an abstractive summarization model. We have chosen BART [4], a transformer based neural network for machine translation. The larger version (12 transformer layers) is used in our summarization task. BART specializes in natural language generation based on

input text, this is excellent for summary in our case, making sense of seemingly disjointed collection of sentences and producing meaningful summary, hence it was chosen in our system.

Finally, in order to construct the knowledge graph, the data obtained so far from the keyword extractor needs to be processed. Keywords are vectorised using spaCy's pre-trained word vectors in order to obtain cosine similarity scores for the knowledge graph edge's lengths. These lengths represent how related these keywords are to each other. The scores are also used to compute the k-nearest-neighbours of each keyword. These processed data serve as inputs for our knowledge graph.

Chapter 4 Schedule

PHASE	Planned Milestone Date	Actual Milestone Date
Initiating Phase	Week 2	Week 2
Planning Phase	Week 4	Week 5
Execution Phase	Week 11	Week 12
Closing Phase	Week 12	Week 12
Project End Date	9 th Nov 2020	12 th Nov 2020

Table 2. Milestone

4.1 Differences Between the Planned and Actual Schedules

During the entire DIP (Design & Innovation Project) journey, there are bound to be challenges in between, which ended up causing the project timeline to deviate from what was planned at the start. The following are the roots of the challenges that hindered the progression of the project. Firstly, since some of the group members did not have much experience with programming many of them needed to spend a lot of time on research and learning new programming frameworks. Every member was divided into the following group, frontend, backend and data processing. Front-end members were tasked to pick up basic knowledge of HTML, CSS and JavaScript and Bootstrap. These were the initial skills which were required. Further delay was added when the main framework was changed to React.js since it was easier

to work with. This change required us to pick up React.js as well. The backend was tasked to find out on how the collected data is to be stored. The data processing team had to find a suitable machine learning model that extracts and summarizes the relevant content. Apart from learning new programming languages, research was conducted to understand the connections between different languages that are needed for various parts of the scripts.

Secondly, in the execution phase, there were modifications made to further improve the interface of the web application. To enhance the frontend security system and to protect the user's personal information, we also added authentication application programming interface (API), login API and signup API. These were not considered when the initial timeline was created. Furthermore, to improve the accuracy of the data processing, a trial-and-error process was required to achieve a decent result which also added to the time delay.

Lastly, integrating the codes together was another challenge. Reason is that some codes are not in the same programming language as the others. To integrate these different branches together, we are to find out the types of extensions that can be used to combine them together. Managing the different codes sourced requires expert understanding as many of the codes and API are often catered to vendors own purpose.

Chapter 5 Cost

PHASE	Planned Costs	Actual Costs
Initiating Phase	\$0	\$0
Planning Phase	\$0	\$0
Execution Phase	\$0	\$0
Closing Phase	\$0	\$0
Project Total Costs	\$0	\$0

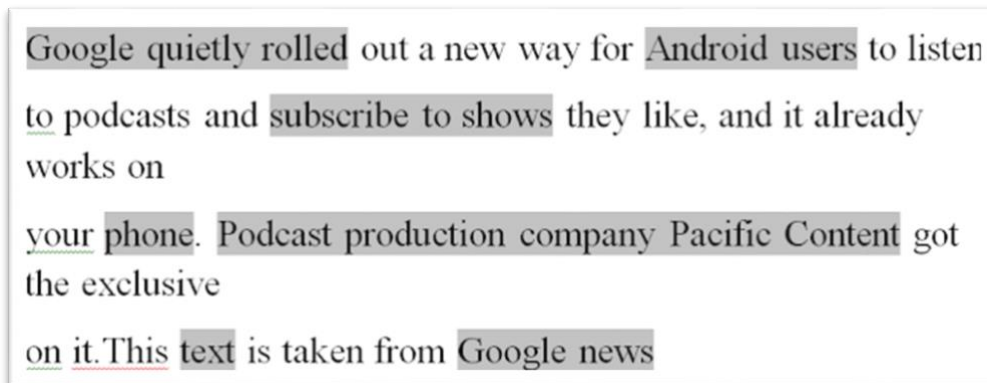
Table 3. Cost Table

Initially, we planned to purchase a cloud server account called Firebase to use it for storage of the PDF files, but it was not needed as the server is not overloaded. Most of the resources that we needed is available for free online thus there was no need for additional costs.

Chapter 6 Research

There are 3 main processes in Learnify.ai project in order to improve student learning experience through the implementation of summarization of lecture node in a knowledge graph. Firstly, we do a keyword extraction process from the content. Secondly, the extracted keywords will be used to find relevant sentences, which are combined and summarized. Finally, a visualization of the summarized content will be implemented in a form of knowledge graph.

6.1 Keyword Extraction



Google quietly rolled out a new way for Android users to listen to podcasts and subscribe to shows they like, and it already works on your phone. Podcast production company Pacific Content got the exclusive on it. This text is taken from Google news

Figure 25. Example of Keyword extraction by RAKE

6.1.1 General Approaches

6.1.1.1 Statistical Approach

Identifying main keywords in documents use of word frequency, word collocations, co-occurrences (N-gram statistics, words that frequently go together), TF-IDF (Term Frequency-Inverse Document Frequency) that measures the importance a word is to a document in a collection of documents and RAKE (Rapid Automatic Keyword Extraction) which uses a list of stopwords and phrase delimiters to detect the related words in a piece of text. [10]

6.1.1.2 Linguistic Approach

A method of using linguistic information to extract the keywords. An example will be using part-of-speech of words or the relations between words in a dependency grammar representation of sentences. Like generally nouns and noun phrases are given higher scores as they are objects that has information as compared to verbs, etc.

6.1.1.3 Graph Base Approach

Considering words as vertices connected by a directed edge and represent them in a graph. The edges can be labeled with the relation that words have in a dependency graph. Importance of word as keyword extraction is based on the measure of vertex and edge.

6.1.1.4 Machine Learning Approach

This approach is the use of artificial intelligence that builds algorithms that can learn from example and make their own prediction in keywords extraction. Hybrid Approach is the approach that combines 2 of more of the above approaches for a better and accurate results for keyword extraction.

Notable keyword extraction algorithms that were tried are the TF-IDF, TextRank algorithm, BERT and YAKE.

6.1.2 Keyword Extraction Models

TextRank algorithm [5] is based on PageRank which is a tool often used in keyword extraction and text summarization. TextRank follows closely to the implementation of PageRank. PageRank calculate, in a bunch of websites of interest, the probability of user visiting another website from webpage of interest. In TextRank, sentences/words are used in place of webpage. Similarity between all sentences/words are used in place of webpage transition probability and

are stored in a square matrix. This square matrix is then converted to a graph, sentences/words as nodes and similarity scores as edges. An arbitrary number of top ranked sentences forms a summary of the text, while top ranked words form the main keywords of the text provided. [12]

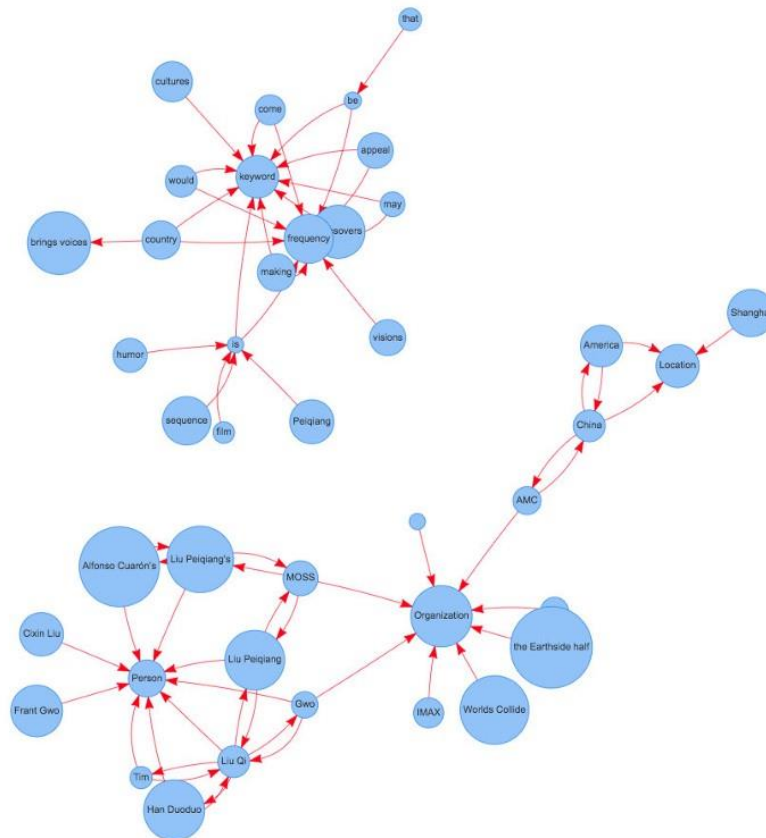


Figure 26. Keywords Extraction with TextRank

BERT or Bidirectional Encoder Representations from Transformers [1] is a state-of-the-art NLP language model. Its model architecture is a multi-layer bidirectional transformer encoder excluding the transformer decoder of the original transformer architecture in [6]. Bidirectionality allows the model to learn the meaning of a word based on all its surrounding instead of the sequence of how words appear. Conventional NLP models are single direction,

IE left to right like how we would read in English. An example is the RNN. There exist also bidirectional RNNs (i.e., Bidirectional LSTM) however RNNs generally, suffer from semantic meaning losses as sentences gets progressively longer w.r.t to word of interest. Transformers overcomes this issue through the use of self-attention mechanism. Hence transformers are still known to be the top performer for language modelling.

YAKE! is a light-weight unsupervised keyword extraction method using statistical text features from a single document to filter the most relevant keywords of the document texts [3]. As a statistical method, YAKE! does not require training, dictionaries, external corpora and is domain independent. Unlike other methods, YAKE! is able to identify varying n-gram keywords and is only restricted to the upper limit 'n' set in the code. I.e., keywords with word length of 1 to 3 are returned in a process of extraction when 'n' is specified to 3. This difference outperforms BERT algorithm we tested which can only output a single n-gram keyword (i.e., specified $n = 3$, all keywords returned are 3-word in length). Experiments reported in [3] showed YAKE! outperforms other unsupervised methods that we have tested, namely TF-IDF, RAKE and TextRank.

6.2 Text Summarization

Summarization of text is to shorten documents and reduce the text while maintaining the key information elements and what the document is talking about. To do so, various Natural Language Processing (NLP) tools have been used to help in summarizing text. There are 2

approaches to text summarization, with one being extractive summarization which is by copying and rearranging passages from original document, and the other which is abstractive summarization which produces new phrases, rephrase words or long text into phrases not found in the original document.

SMRRY (SUMMARY) is a popular text summarization software, with API that can be integrated to existing projects. How it works is it rank sentences by importance using their core algorithm and remove anything unnecessary to shorten the text. The core algorithm works in the way of associating words with their related grammatical parts like “NTU” to “Singapore University”. It will then record the times each word appears in the text and rank them up based on popularity. And finally return X of the most highly ranked sentences in chronological order.

Fairseq (fair sequence) [14] is another sequence modeling toolkit that can help in translation, summarization, language modeling and other text generation task. And the text summarization model that we have decided to use is BART [4], which is a pretrained model derived from fairseq which helps in doing text summarization.

6.3 Visualization of Knowledge Graph

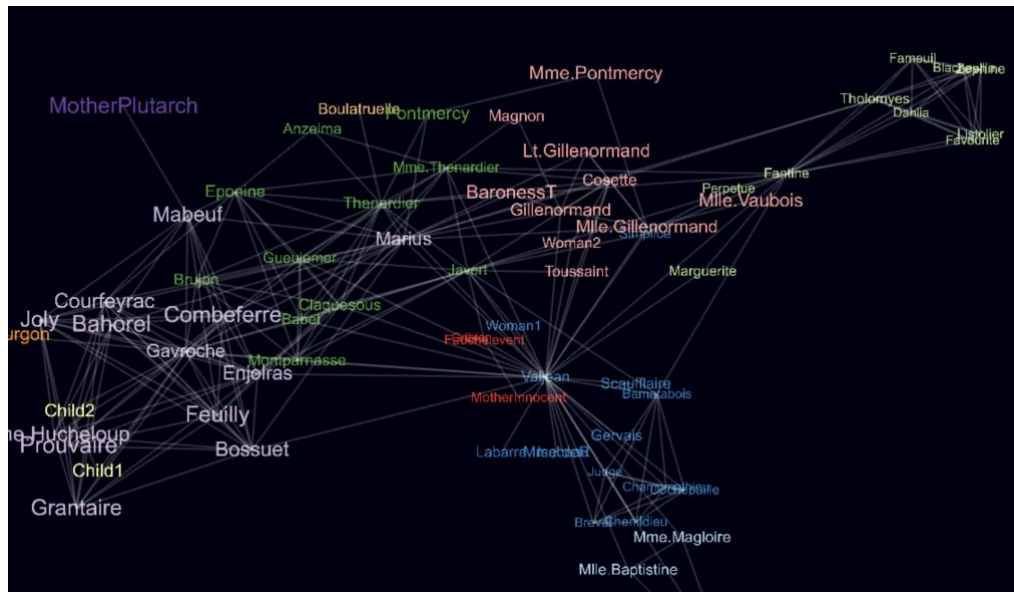


Figure 27. Example of a Knowledge graph (force graph)

Knowledge graph has been used to simplify searches, making more condensed information that is easier to absorb and concatenate with relevant topics. In summary, it acts as a mind map that is formed through Deep Learning, webbing all the topics together. Clicking the nodes will link users to their respective data. There will be challenges such as integration of information together and accuracy of visualization of graph. [8]

LinkedIn's Knowledge Graph. Similarly, LinkedIn's Knowledge graph is built based on profiles created on LinkedIn, with jobs, titles, skillsets, experience and geographical location which links them into a force graph, As for Learnify.ai, it acts as a mind map to link academic topics and notes based on their relevant in keyword.

Implementation:

Relationship between keywords are calculated by cosine similarity. However, since our keyword extractor uses unsupervised methods, it is not possible to return embeddings of each word. Hence, we vectorized words using SpaCy's pre-defined English large word vectors. Each word is represented by a vector of dimension 300. Key-terms with multiple words are vectorized via average of the terms. Next, top 3 similar terms to each key-term are found by comparing cosine similarity scores and assigned as key-terms to the key-term of interest.

Chapter 7 Outcomes / Benefits

7.1 Outcomes

The initial intended outcome for the project was to develop a web-based application to aid student's in studies and revisions. Hence, our objective was to build a key knowledge extraction and relation function to automatically provide students with the intuitive overview of an entire course as an interactive graph while also providing brief summaries of each topic extracted. Revisions to the app's features were made and additional features identified to be beneficial are added in the course of the project. Hence our finalized app will be featuring 1. key knowledge summary and relation-based knowledge graph, 2. Key knowledge importance ranking, 3. manual input key term – course term relation finder, 4. time constraint, individual study session dynamic study topics recommendation.

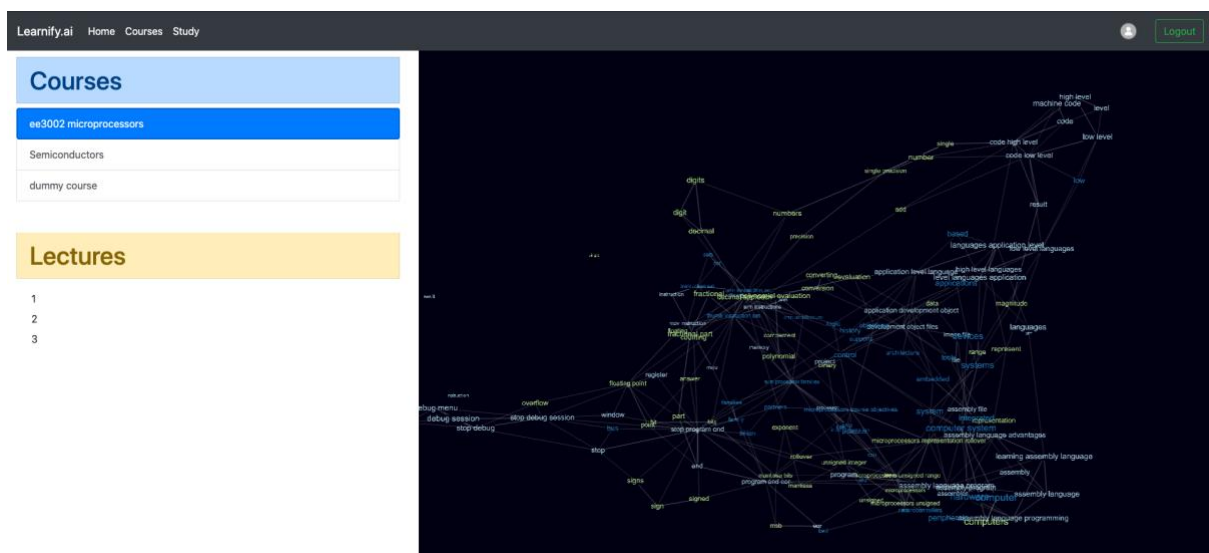


Figure 28. Knowledge Graph

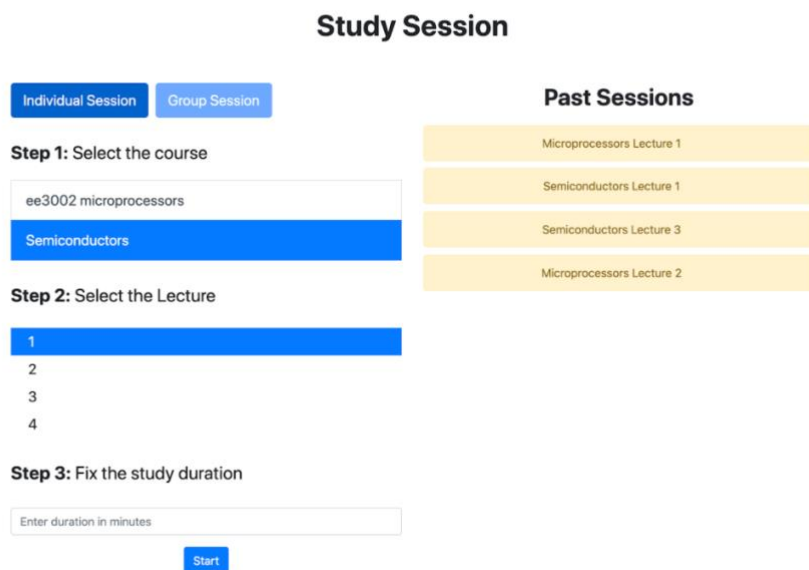
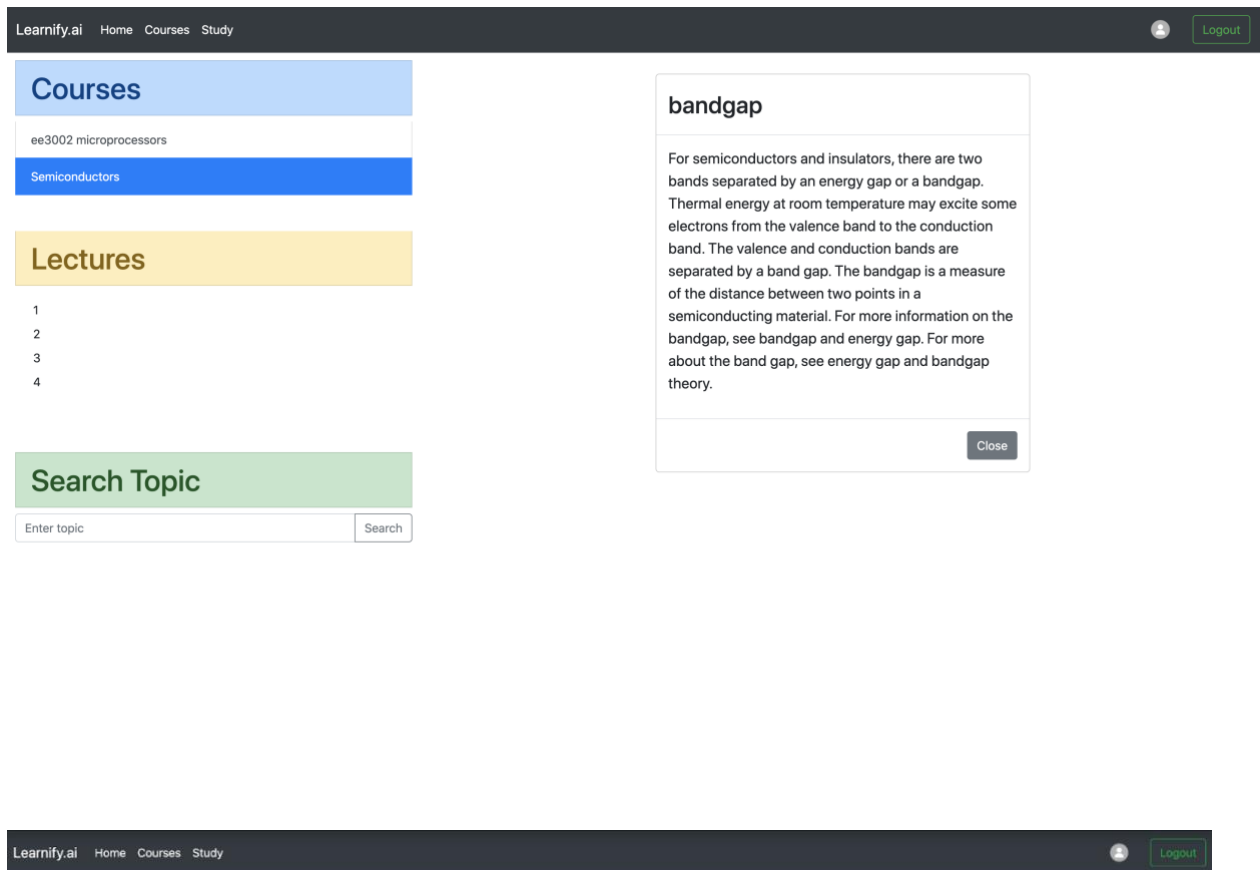


Figure 29 Study Page

These features are achieved through 3 core functions of the app. Keyword extraction, extractive summarization and knowledge graph. These features are then made available to the user

through supporting key elements such as the user interface, database and APIs which supports 1. input, output and storing of personal data, 2. management of user accounts and 3. displaying of visuals. Even though some features included during the planning phase was replaced, we believe our app still brings appreciable benefits to students.

7.2 Benefits

Studying strategies can differ among students but one of the most common techniques is repeated reading. Although this is one of the most frequent form of exam revision, it is also one of the most ineffective.^[1] It is time-intensive and often not as useful as other study techniques such as practicing questions or even making outlines and review sheets. Therefore, to minimize the time spent on re-reading notes students can instead use Learnify.ai which collects the most important terms from lecture notes and presents them in the form of an easy-to-read knowledge graph. This provides the user a brief overview of their subject.

7.2.1 A more efficient form of studying

The knowledge graph can present the key terms collected from a particular set of chapters or even a whole module, since Learnify.ai allows the user to upload multiple files. This offers the

student a more efficient use of their time since the time taken to re-read notes will be reduced. There is also a search feature which allows the user to type in the key term they are looking for. Therefore, students can focus more time on practicing questions which is considered a better method of learning since it involves memory retrieval.

7.2.2 Better visualization

The knowledge graph also links the relevant concepts together which helps the student visualize the subject they are studying. The distance between the nodes represents to what extent the individual nodes are related. This helps in chunking closely connected nodes together and makes the content easier to memorize which aids in learning. The information is also organized in a top-down structure. This approach is conducive to subjects which explain concepts using deductive reasoning which involves starting with the bigger picture and breaking it down into smaller segments to derive a theory.

7.2.3 More Interactive

Moreover, not only is the knowledge graph more time efficient and easier to learn, it is also a more interactive form of learning than simply reading lecture notes again and again. The users can use the knowledge graph to test if they know what the key terms mean and then check if they are right by clicking on the node so that they can view the summary. Users can even delete the terms they are confident with so that they focus on the unfamiliar terms.

[1] Karpicke, Jeffrey D., Butler, Andrew C. and Roediger III, Henry L.(2009)'Metacognitive strategies in student learning: Do students practise retrieval when they study on their own?',*Memory*,17:4,471 — 479

Chapter 8 Reflection

8.1 Engineering knowledge

Most members were not very experienced with programming when the project began but since the project was primarily focused on coding all members had to research and begin self-learning the necessary skills. The programming languages all members were familiar with was limited to Python and C/C++. Due to this, the programming language primarily used was python. The general interfaces which needed to be learned were Visual Studio Code and Github. Group members also had to learn specific skills depending on the tasks they were assigned. Therefore, front-end, back-end and data processing teams were all required to pick up different skills.

The front-end team had to take up HTML, GITHUB, JAVA, CSS and REACTJS. While the back-end team learned MONGODB and improved their PYTHON skills. The Data Processing team had to research, evaluate algorithms / models and script model integration in between their own models and in between the frontend and backend teams.

8.2 Problem Analysis

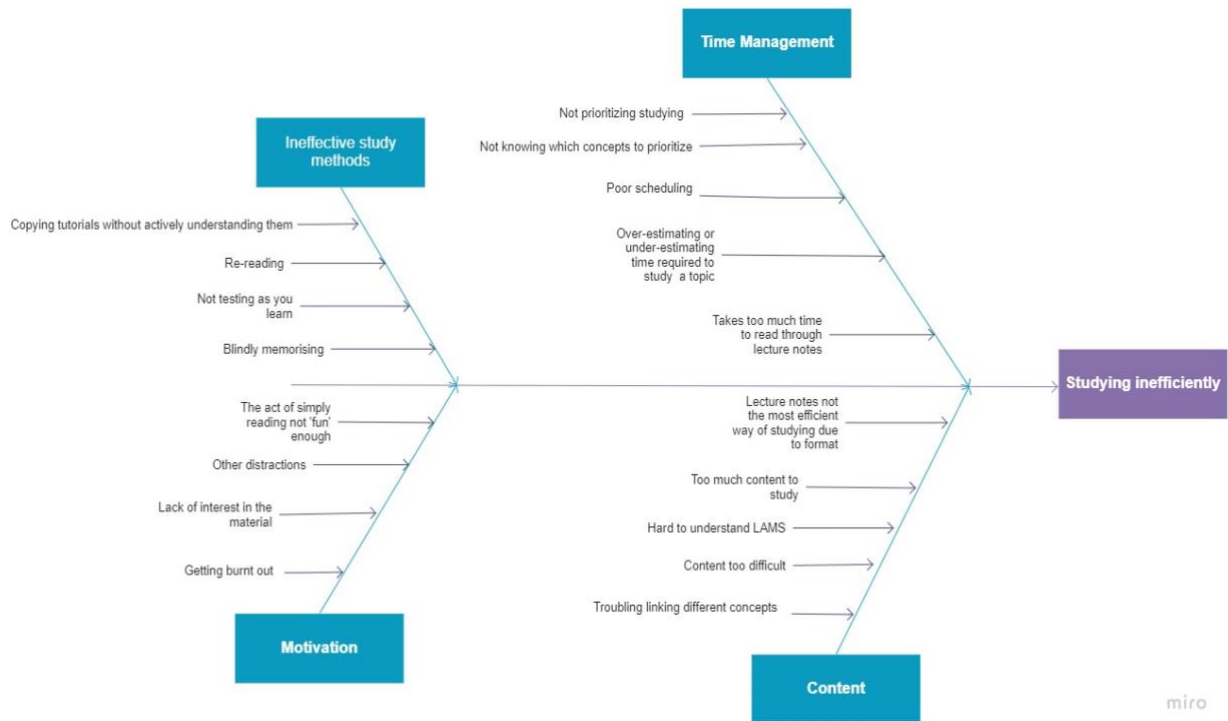


Figure 30. Problem analysis on why student is studying inefficiently

8.3 Solution development

Although we couldn't target all the causes for the chosen problem, we tried to design a solution which would target as many causes as possible. Through the problem analysis we discovered, simply reading lecture notes was not an effective method of studying but students often resort to it since it is the most straight-forward approach. Secondary problems identified were not finding the act of studying sufficiently engaging as well as not testing yourself periodically which is essential for studying effectively. Learnify.ai targets these issues since it offers an

alternative to lecture notes in the form of the knowledge graph which provides links between different concepts as well a faster way to remember different concepts. It also provides the user with a summary of the key terms when a user clicks on a node. This helps improve the efficiency of studying which in turn helps the student with the problem of time management. Learnify.ai also has a search function which can be used to quickly search through the knowledge graph for key terms further improving efficiency and allowing for a more targeted form of studying. This knowledge graph is also more interactive than the walls of text available in textbooks or lecture notes and therefore using Learnify.ai it is possible to further gamify the learning experience.

8.4 Individual/Teamwork

In phase 1, the planning stage of the project, we had to come up with a project idea which related to our main topic 'Artificial Learning'. To do this successfully we conducted several group brainstorming sessions to come up with problems commonly faced by students and potential solutions to those problems. This was done with the full group and after a discussion with our supervisor we were able to narrow down our topic to Learnify.ai by Week 2. Next to further hone the project requirements and steps for implementation we decided to split the team into three groups. Siddesh, Seng En and Cheng Long were assigned as the lead programmers since they had the greatest amount of experience. Their task was to find out the necessary steps required to implement the project for example, the programming languages and frameworks which needed to be used. Yu Xiang, Theodore and Noah were formed the Research group who

oversaw checking online for ideas similar to Learnify.ai to ensure that the project was sufficiently unique and so that we could improve upon the pre-existing ideas. Sanchari and Eugen were placed in the writing department which ensured updated the blogs and oversaw the writing of the project charter. During this process, members of the team more experienced with programming offered resources and support to the less experienced members in order to better facilitate their learning.

In phase 2, the team was split again for the implementation phase. The team was divided into three groups: Front-end, Back-end and Data processing. This would allow for progress to be done on all three fronts simultaneously and would also give the front-end and back-end teams ample time to become comfortable with their respective tools. Sanchari, Eugen and Yu Xiang were in the front-end team, Cheng Long, Theodore and Noah were in the back-end team and Siddesh and Seng En were in the data processing team.

In phase 3, the team was divided again into three groups. Seng En and Siddesh worked on integrating the different components of the project and putting the final touches in order to prepare it for the demo. Cheng Long, Noah and Theodore focused on researching and writing the background knowledge and research section of the report while Sanchari, Eugen and Yu Xiang focused on writing different sections of the report.

8.5 Future Recommendations

Further improvements which can be made include making the knowledge graph more interactive. This could be done by adding a feature which allows users to delete nodes which they are done studying. This would be further improved by including a mode called group study session where users can invite their friends and compete on who can complete or 'destroy' the knowledge graph the fastest.

To further improve the effectiveness of this feature before a node can get destroyed the user would be required to complete a short set of questions based on that topic. The interface for this could be further enhanced by adding score cards and completing each node would give users a certain number of points.

Additionally, another mode called 'rebuild mode' can also be added which would allow the users to rebuild parts of the knowledge graph from memory. More mini-games could also be added such as one which presents the users with a list of summaries and a list of the key terms and requires them to pair them up. This would help the user revise the content provided on the web application and also make it more interesting and user-friendly for younger audiences. Incorporating these mini-games would emphasize on memory retrieval which is a more effective method of learning than simply reading the content.

Other improvements such as making the user interface more visually appealing could be done as well. Image data could be included as well wherever possible since using different types of data would improve information retention. Nodes in the knowledge graph could also be color coded as per the user's understanding. When the user has freshly learned the concept and logs it in to the knowledge graph accordingly the node would turn green. Then as time passes this color would slowly shift to red indicating that the user has likely forgotten this concept and

should revise it again. This change in color would be modelled using the forgetting curve [1]

(reference: Article Source: [Replication and Analysis of Ebbinghaus' Forgetting Curve](#)

Murre JMJ, Dros J (2015) Replication and Analysis of Ebbinghaus' Forgetting Curve. PLOS ONE 10(7): e0120644. <https://doi.org/10.1371/journal.pone.0120644>) which maps the decline of memory retention over time. Once the user revises the concept and logs it into the knowledge graph the node would turn green again. Unread topics can be shown in white.

Chapter 9 References

- [1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018.
- [2] J. Ramos, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, 2003, vol. 242: New Jersey, USA, pp. 133-142.
- [3] R. Campos, V. Mangaravite, A. Pasquali, A. Jorge, C. Nunes, and A. Jatowt, "YAKE! Keyword extraction from single documents using multiple local features," *Information Sciences*, vol. 509, pp. 257-289, 2020/01/01/ 2020, doi: <https://doi.org/10.1016/j.ins.2019.09.013>.
- [4] M. Lewis *et al.*, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.
- [5] R. Mihalcea, Rada, and P. Tarau, *TextRank: Bringing Order into Texts*. 2004.
- [6] A. Vaswani *et al.*, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998-6008.
- [7] "MongoDB Atlas," MongoDB.com. [Online]. Available: https://www.mongodb.com/cloud/atlas/lp/try2?utm_source=google. [Accessed: 12-Nov-2020].
- [8] V. Asturiano, "vasturiano/force-graph," 2020. [Online]. Available: <https://github.com/vasturiano/force-graph>.
- [9] S. Dery, "Challenges of Knowledge Graphs," 2 Dec 2016. [Online]. Available: <https://medium.com/@sderymail/challenges-of-knowledge-graph-part-1-d9ffe9e35214>.
- [10] L. Gonçalves, "Automatic Text Summarization with Machine Learning — An overview," April 2012. [Online]. Available: <https://medium.com/luisfredgs/automatic-text-summarization-with-machine-learning-an-overview-68ded5717a25>.
- [11] "Keyword Extraction," MonkeyLearn Inc. , [Online]. Available: <https://monkeylearn.com/keyword-extraction/>.
- [12] X. Liang, "Understand TextRank for Keyword Extraction by Python," 19 Feb 2019. [Online]. Available: <https://towardsdatascience.com/textrank-for-keyword-extraction-by-python-c0bae21bcec0#:~:text=TextRank%20is%20an%20algorithm%20based,Extraction%20with%20TextRank%2C%20NER%2C%20etc>.
- [13] "Automatic Keyword Extraction using RAKE," [Online]. Available: <https://thinkinfi.com/automatic-keyword-extraction-using-rake-inpython/>.
- [14] M. O. a. S. E. a. A. B. a. A. F. a. S. G. a. N. N. a. D. G. a. M. Auli, "fairseq: A Fast, Extensible Toolkit for Sequence Modeling," [Online]. Available: <https://github.com/pytorch/fairseq>.