Project Proposal based on Research Paradigm, initial literature review schedule and risk analysis.

Claudinea de Almeida

Academic Paper Management on the Cloud

Abstract

Nowadays, with a technological expansion of cloud computing, there has been a giant expansion of the possibilities of accessing and sharing data in general and a demystification of knowledge.

The best way to share and disseminate knowledge is to create a management system for this data. On the academic side, when you're looking for an idea, or simply getting to know technological innovations, it can become a nightmare. Despite finding a lot on the internet, it is still very restricted to academic proposals and master's and doctoral works.

The idea is based on the development of a web portal, where authors of master's theses and course conclusion works can be published in a community database, where everyone can have free access to research works.

This project will allow users to make their theses public, organizing them by category, even with the different ways of exposing their article.

Currently, systems of this nature are lacking, with few exceptions, and even the existing ones do not have the flexibility to organize the publication by topic, and even free of charge, all based on the most current technologies on the market, which will be discussed later in this document.

Keywords:

Thesis management system, Cloud Computing, API, Publishing Articles

Contents

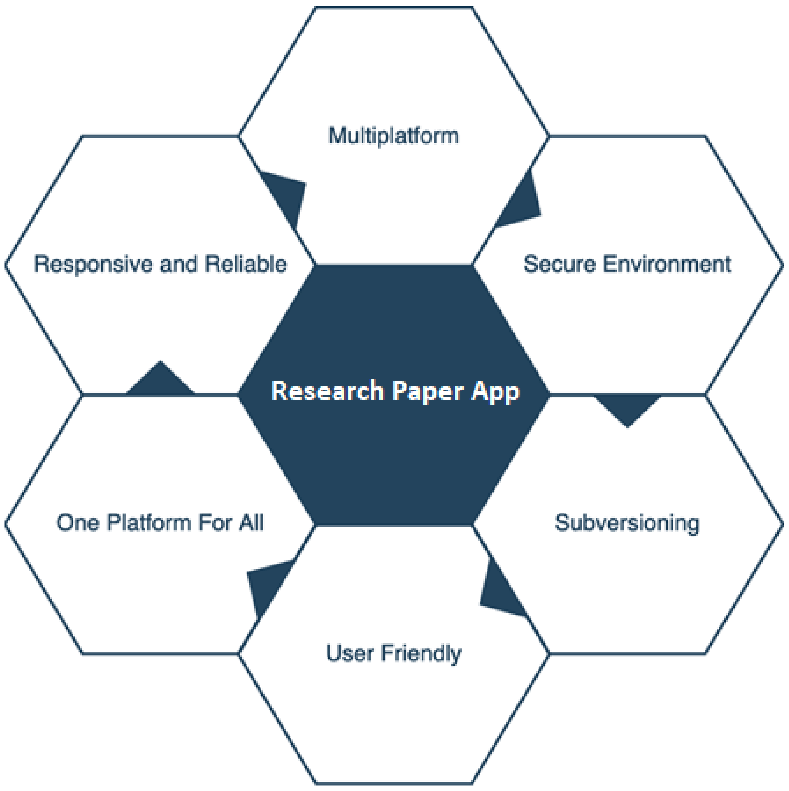
1. Introduction

The main idea of this project is to create a system, based on cloud computing, focused on web page and mobile application, which allows students and professors to publish their articles, theses, final projects of course conclusion automatically and independently.

The system will facilitate the opening of articles to the general public with unrestricted access, organizing publications by categories and making university articles available to everyone. This makes it easier for master's students to have an idea of published articles that will serve as a reference for their future projects.

The System will be developed with the most current technologies and with a simple and understandable interface so that all students, teachers, and readers in general can navigate without complications inherent to information technology.

As a software engineer, my mission is to specify details of how the system works and the best approach so that the result meets users' expectations.



2. Research question and Rational

1. How can I allow unrestricted system access and uploading without affecting software performance?
2. How to categorize the different types of thesis topics and final projects?

The first question can be answered by utilizing the best qualities found in using cloud computing, which allows scalability to handle the changing needs of an application within the confines of the infrastructure by statically adding or removing resources to meet demands. applications, if necessary.

For the second question, the answer will be the use of tags, which allow the merging of related articles.

3. Literature Review

In this session, basic concepts of cloud computing and its particularities that allowed the adequate performance of the proposed system will be discussed. The subject of tags will still be discussed and how we will approach this algorithm to properly categorize published articles.

When it comes to cloud computing, what can be kept in mind, speaking at a high level, is that it makes the issue of physical structure and the location of the servers where the system will be stored flexible, one of the characteristics is that it has elasticity and scalability that allow the server's capacity to be expanded during peak usage times, allowing the system to function normally, without affecting its performance and being invisible to the eyes of the end user.

Regarding the database, NoSQL will be used, which speeds up the inclusion of data and improves the speed in the search for data, without the need for a complex relationship structure used in SQL databases.

In addition, the search will be done with API technology, which simplifies and speeds up the search for data, thus allowing an extra help in terms of performance.

Tags are a way of grouping related subjects, in this way, we will be able to organize the published articles in a more specific way, both from the point of view of who is doing the publication and from the side of the future reader, looking for more consistent related subjects.

3.1 Technologies

This program will be created using Node.js environment, JavaScript as back-end language, MongoDB Atlas as cloud database, Express as server, HTML and Ejs as scripting language and front-end, Bootstrap as framework, GitHub as repository and version control, Postman for API testing.

Node.js

Node.js can be defined as a server-side JavaScript execution environment.

This means that with Node.js it is possible to create JavaScript applications to run as a standalone application on a machine, not depending on a browser to run, as we are used to.

MongoDB Atlas

It is a global cloud document database service for modern applications. Basically, you only care about managing the data that will be there, all the infrastructure and maintenance of the machines, as well as the security of all this is up to them.

NoSQL

NoSql databases use a variety of data models to access and manage data. These database types are specifically optimized for applications that require large data models, low latency, and flexibility. These requirements are met by relaxing some of the other databases' data consistency restrictions.

Express

Having its initial version released in 2010, Express.js (or just Express) is a framework for developing JavaScript applications with Node.js.

HTML and Ejs

HTML (Hypertext Markup Language) is used to design the structure of a web page and its content. HTML is not technically programming languages like C++, JAVA, or python. It is a markup language and ultimately provides declarative instructions to the browser.

EJS simply means inline JavaScript. It is a simple template engine or language. EJS has its own syntax and defined tags. It offers an easier way to generate HTML dynamically.

Bootstrap

Bootstrap is a front-end framework that provides CSS frameworks to create responsive websites and apps quickly and simply. Also, it can handle both desktop websites and mobile pages.

GitHub

GitHub is a cloud-based service that hosts a version control system (VCS) called Git. It allows developers to collaborate and make changes to shared projects while keeping a detailed record of their progress.

Postman

Postman is a tool that supports the documentation of requests made by the API. It has an environment for documentation, execution of API tests and requests in general.

API

APIs are a set of patterns that are part of an interface and that allow the creation of platforms in a simpler and more practical way for developers. From the APIs it is possible to create software, applications, programs, and different platforms.

Car
Web Apps
Enterprise Application
Mobile Device
API
API
Game
Sensors
TV
Wearable Device

Figure 1- API (dzone.com, 2021)

Docker

Docker aims to create, test and implement applications in an environment separate from the original machine, called a container. In this way, the developer can package the software in a standardized manner. This is because the platform provides basic functions for its execution, such as: code, libraries, runtime and system tools.

The great advantage of using the platform is the speed at which the software can be made available.

Docker's entire end to end platform includes UIs, CLIs, APIs and safety that are engineered to work together over the whole application delivery lifecycle. (Docker, 2021)

3.2 Architecture

As far we will using MongoDB as a Database, we will have the architecture like described below:

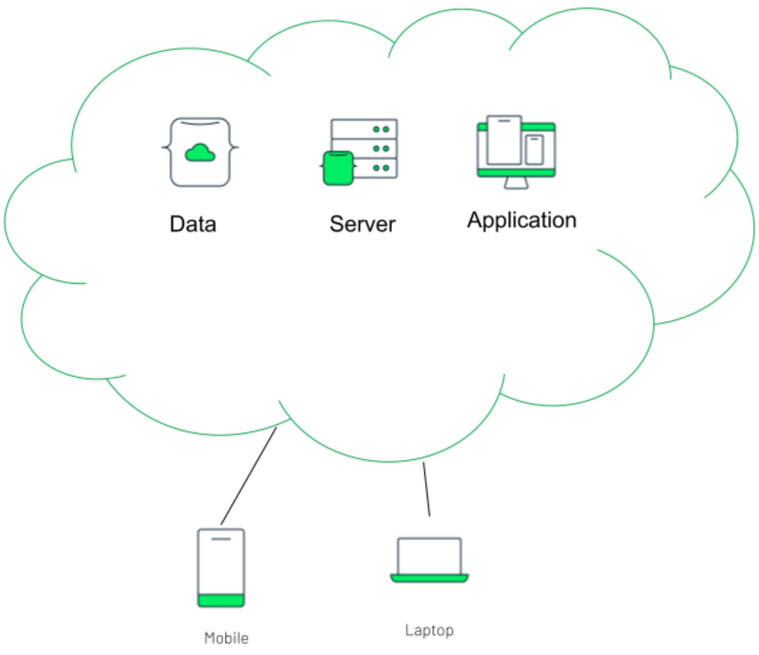


Figure 2 - MongoDB (MongoDB. n.d.)

3.3 Methodologies

3.3.1 What is Methodology

The methodology is the way to use a coherent and coordinated set of methods to achieve an objective, to avoid, as much as possible, subjectivity in the execution of the work.

3.3.2 Agile

The agile methodology is a set of project management techniques and practices that offers more agility, efficiency, and flexibility.

Continuous collaboration is critical, both between team members and project stakeholders, to make fully informed decisions. The agile manifesto was developed in 2001 and was looking for the best way of developing software. (Agilemanifesto.org, 2001). Its benefits are versatility, based on your client's goals, Transparency.

3.3.3 Scrum

What is Scrum

Scrum is a structure within which people can solve complex adaptive problems while delivering products the highest possible value productively and creatively. (Scrum.org, 2021)
Scrum is one of the most popular Agile frameworks, the reason why we chose this methodology to use in our project. The team needs to be involved with the project to function well, including stakeholders.
Scrum methodology fast explanation at link: https://youtu.be/TRcReyRYIMg.(2015, What is Scrum?)

Benefits: Versatility, based on your client's goals, Transparency

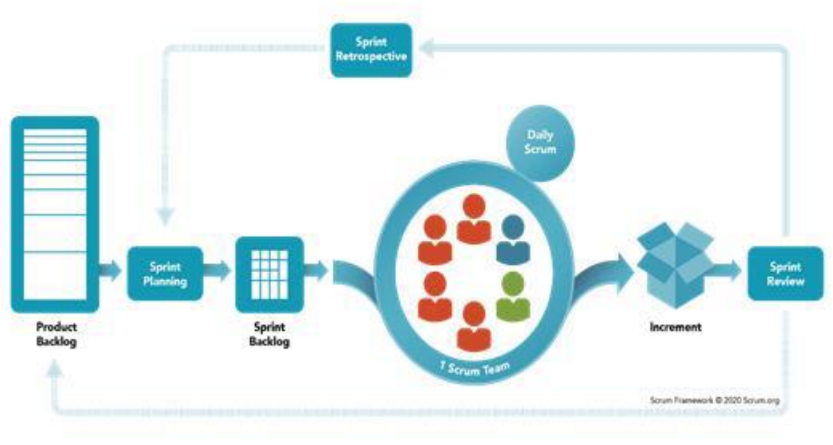


Figure 3 - Scrum Methodology (Scrum.org, 2021)

3.3.4 Summary

Based on the research done on the systems development methodologies available, I decided that Scrum Agile is the best methodology for this application.

4. Related Work

A similar project was found but that focuses on the university's control over the upload of articles and approval of the topic desired by the professor, this project in particular focuses on the search for example articles, not on the approval of article proposals by the professor. (Khalil, Y., 2018)

There is the website http://www.academia.edu, which has something similar, but they don't organize by tags, so the subjects are often mixed and confusing, besides they charge for the full article published, you don't have free access total.

5. Proposed Work

The proposal consists of creating a system for registering student articles in two main points:

1. Uploading articles
2. Creating tags for grouping related topics

5.1 Project requirements

5.1.1 Functional requirements

Input

1. Add paper: user selects the New paper link to upload a new paper in the application.

2. Update paper: user selects the paper that they want to update from the list of papers that will be available on the table.

3. Search paper: users type the title of the paper that he/she would like to search and click on the button Search to start the search paper process.

4. Delete paper: user selects the paper that they want to delete from the list of papers that will be available on the table. For deleting a paper will be required a password.

Output

1. Messages: after insert the paper the user will be able to see the paper in the table of paper. He/she will receive a message confirming if the insert was correct.

2. Messages: after update the paper the user will be able to see the paper in the table of paper. He/she will receive a message confirming if the update was correct.

3. Paper records: user can see on the table the paper's after search based on the search made, following the tags requirement.

4. Paper deleted: users will see that the paper that was deleted on the table and will receive a message confirming if the deletion was correct.

Processes

1. Search type: user type the title of the paper is wanted.

2. Adding paper: user select the paper pdf file and upload it to the application and type the info needed.

3. Tags: when the paper is adding, it will generate a tag to group the papers with similar items.

5.1.2 Non-functional requirements

Control

1. The system should be able to provide a high level of security to user’s data

2. The user interface should be user friendly to provide ease of use.

3. The system should be reliable all the times to gain the trust of users.

4. The system should be easy to use.

5. The system should be able to recover from failure very fast

6. The system should be able to do automatic system backups to ensure that in an event of failure the data lost can be easily retrieved.

Performance

1. Response time should not exceed 4 ms.

2. The system should be in operation all the times.

3. The system should quickly refresh so that the user will not get bored waiting.

4. The system should always give correct paper information.

5. 2 Defining the tasks

5.2.1 Step 1 - Brainstorm and Identifying tasks

1. Understanding concepts and Proposal methodologies
2. Create a database.
3. Create the project
4. Setup the Node.js libraries and dependencies
5. Create the main file
6. Create a list of papers
7. Create the route
8. Connect to the database
9. Create the route
10. Connect to the database
11. Create the first page for the user to see the available options.
12. Create the insert function
13. Create the update page.

14. Create the update function.
15. Create the delete function.
16. Create the search function.
17. Testing the application
18. Update this documentation

5.2.2 Step 2 - Identify the Team

Stakeholders: who will collaborate with the team. Users, Managers, Owner. In this project, the stakeholders are Claudinea de Almeida and Dr. Muhammad Iqbal.
Product Owner: represents the stakeholders and understanding of the marketplace. In this project is Dr. Muhammad Iqbal.
Scrum Master: The servant leader. Claudinea de Almeida is the scrum master.
Development Team: A group dedicated to delivering product at the end of each sprint. It will be Claudinea de Almeida.

5.2.3 Step 3 - Project Initiation (Sprint 0)

1 Meeting date 14/02/2022 for 2 hours to understand the project.

2 Definition of proposal and priorities.

3 Planning next ceremonies.

5.2.4 Product Backlog

1. Create a database.
2. Create the project
3. Setup the Node.js libraries and dependencies
4. Create the main file
5. Create a list of papers
6. Create the route
7. Connect to the database
8. Create the route
9. Create the first page for the user to see the available options.
10. Create the insert function
11. Create the update page.
12. Create the update function.
13. Create the delete function.
14. Create the search function.
15. Testing the application
16. Updating this document

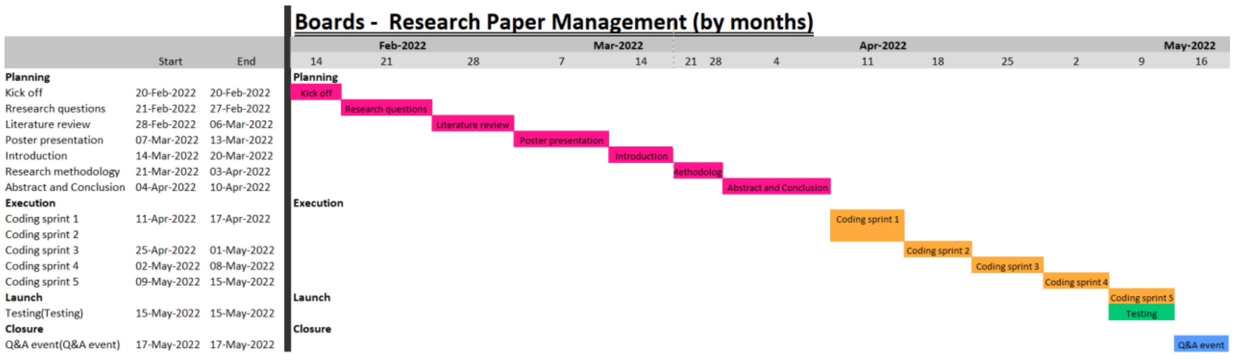
5.2.5 Plan the Sprint Ceremony

Sprint planning – the first day of each sprint Date	Time Start	Time Finish	Subject

14/02/2022	9 am	11:30 am	Understanding the project
21/02/2022	9 am	11:30 am	Understanding the research questions
28/02/2022	9 am	11:30 am	Understanding the literature review
07/03/2022	9 am	11:30 am	Understanding the poster presentation
14/03/2022	9 am	11:30 am	Understanding the introduction
21/03/2022	9 am	11:30 am	Understanding the research methodology
28/03/2022	9 am	11:30 am	Understanding the research methodology
04/04/2022	9 am	11:30 am	Understanding the Abstract and Conclusion
11/04/2022	9 am	11:30 am	Understanding the coding sprint 1
18/04/2022	9 am	11:30 am	Understanding the coding sprint 2
25/04/2022	9 am	11:30 am	Understanding the coding sprint 3
02/05/2022	9 am	11:30 am	Understanding the coding sprint 4
17/05/2022	9 am	11:30 am	**Q&A event**

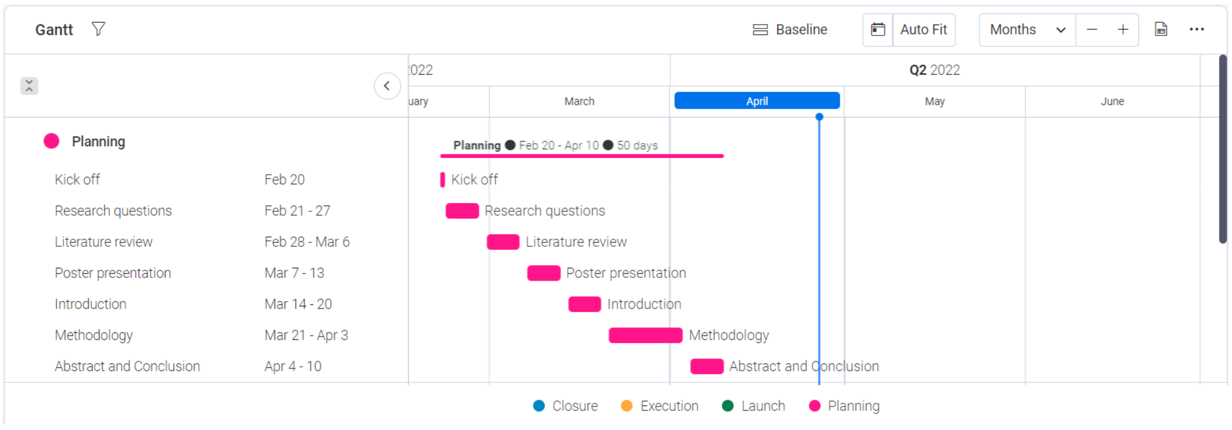
5.2.6 Sprints Cycle

5.2.6.1 Gantt Chart



	Start	End
Planning		
Kick off	20-Feb-2022	20-Feb-2022
Rresearch questions	21-Feb-2022	27-Feb-2022
Literature review	28-Feb-2022	06-Mar-2022
Poster presentation	07-Mar-2022	13-Mar-2022
Introduction	14-Mar-2022	20-Mar-2022
Research methodology	21-Mar-2022	03-Apr-2022
Abstract and Conclusion	04-Apr-2022	10-Apr-2022
Execution		
Coding sprint 1	11-Apr-2022	17-Apr-2022
Coding sprint 2		
Coding sprint 3	25-Apr-2022	01-May-2022
Coding sprint 4	02-May-2022	08-May-2022
Coding sprint 5	09-May-2022	15-May-2022
Launch		
Testing(Testing)	15-May-2022	15-May-2022
Closure		
Q&A event(Q&A event)	17-May-2022	17-May-2022

5.2.6.2 Sprint 0 (from 14/02 to 10/04)


Gantt

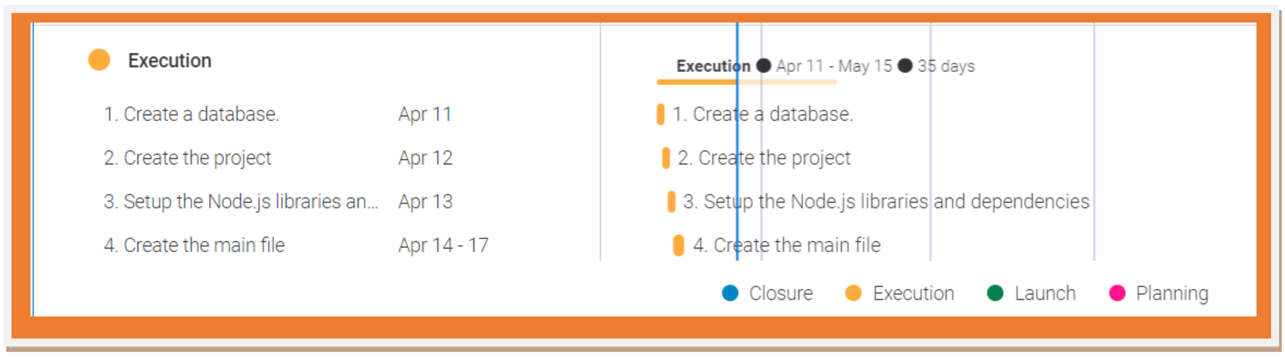
Baseline | Auto Fit | Months

Planning	
Kick off	Feb 20
Research questions	Feb 21 - 27
Literature review	Feb 28 - Mar 6
Poster presentation	Mar 7 - 13
Introduction	Mar 14 - 20
Methodology	Mar 21 - Apr 3
Abstract and Conclusion	Apr 4 - 10

Planning ● Feb 20 - Apr 10 ● 50 days

● Closure ● Execution ● Launch ● Planning

5.2.6.3 Sprint 1 (from 11/04 to 17/04)



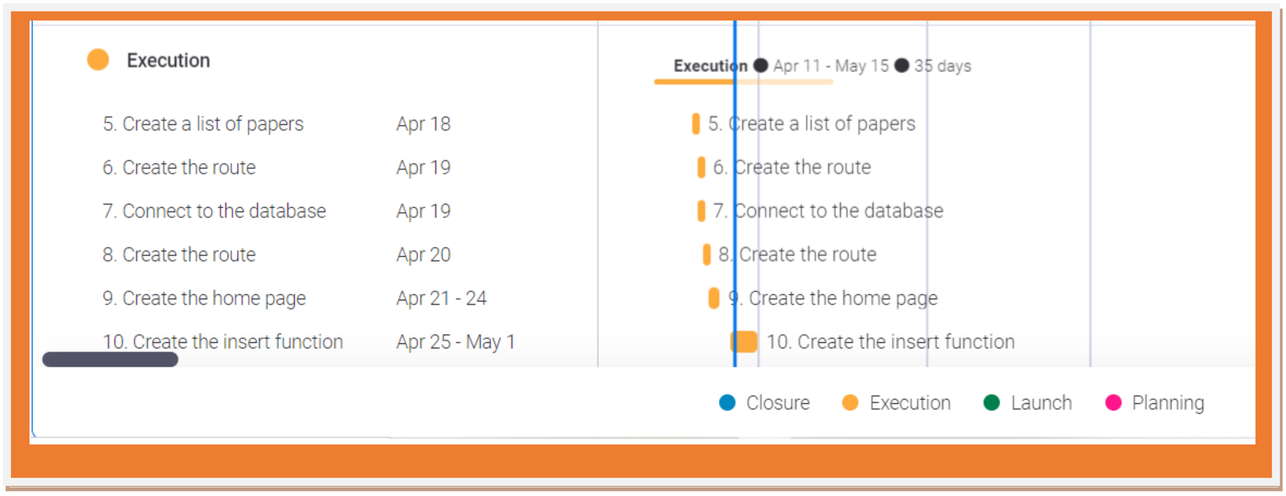
Execution	
1. Create a database.	Apr 11
2. Create the project	Apr 12
3. Setup the Node.js libraries an...	Apr 13
4. Create the main file	Apr 14 - 17

Execution ● Apr 11 - May 15 ● 35 days

3. Setup the Node.js libraries and dependencies

● Closure ● Execution ● Launch ● Planning

5.2.6.4 Sprint 2 (from 18/04 to 24/04)

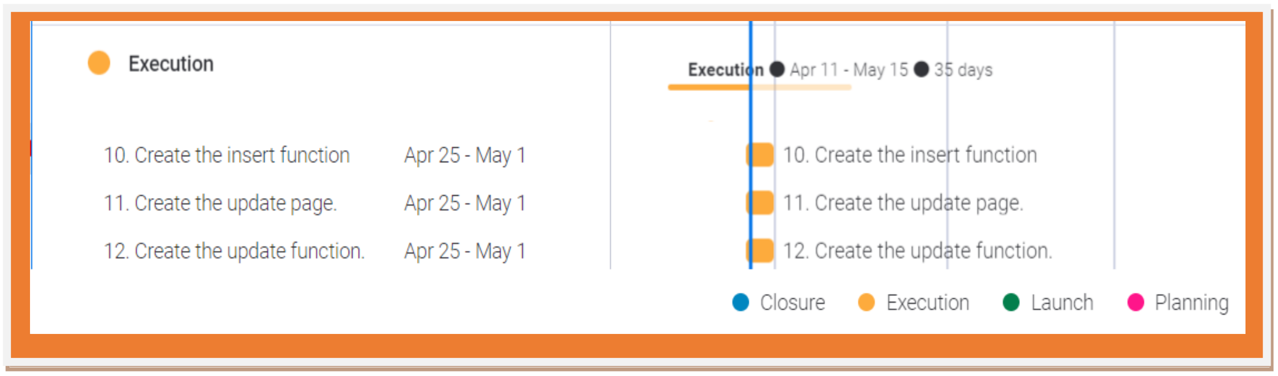


Execution	
5. Create a list of papers	Apr 18
6. Create the route	Apr 19
7. Connect to the database	Apr 19
8. Create the route	Apr 20
9. Create the home page	Apr 21 - 24
10. Create the insert function	Apr 25 - May 1

Execution ● Apr 11 - May 15 ● 35 days

● Closure ● Execution ● Launch ● Planning

5.2.6.5 Sprint 3 (from 25/04 to 01/05)

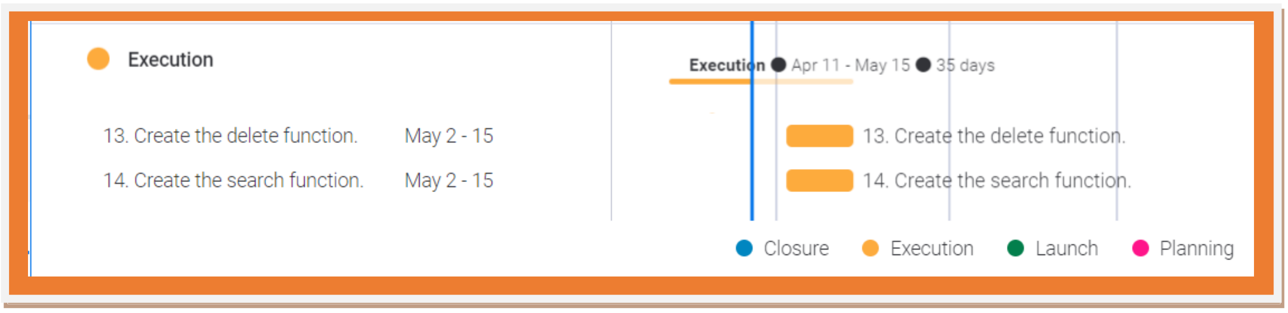


Execution

10. Create the insert function	Apr 25 - May 1
11. Create the update page.	Apr 25 - May 1
12. Create the update function.	Apr 25 - May 1

Execution ● Apr 11 - May 15 ● 35 days
10. Create the insert function
11. Create the update page.
12. Create the update function.
● Closure ● Execution ● Launch ● Planning

5.2.6.6 Sprint 4 (from 02/05 to 15/05)

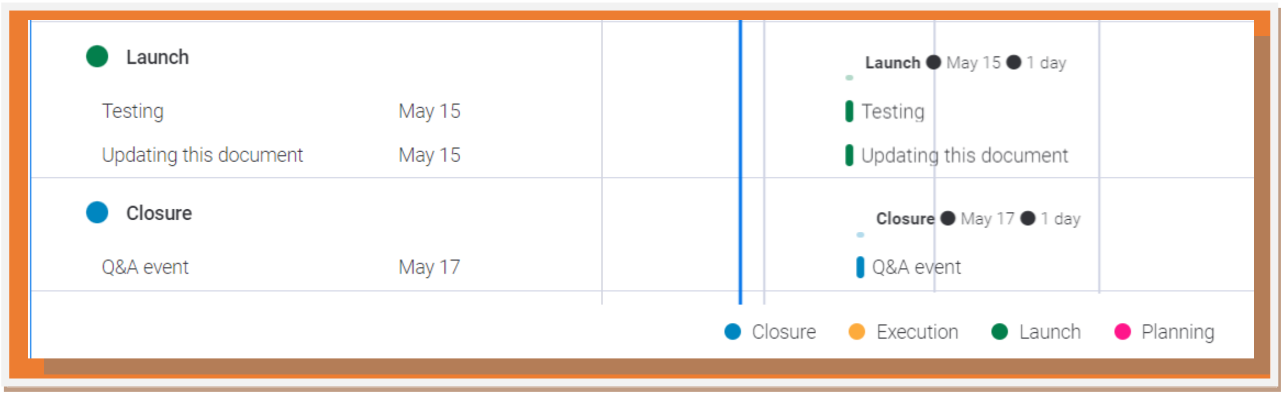


Execution

13. Create the delete function.	May 2 - 15
14. Create the search function.	May 2 - 15

Execution ● Apr 11 - May 15 ● 35 days
13. Create the delete function.
14. Create the search function.
● Closure ● Execution ● Launch ● Planning

5.2.6.6 Testing and Deployment (from 16/05 to 17/05)



Launch

Testing	May 15
Updating this document	May 15

Closure

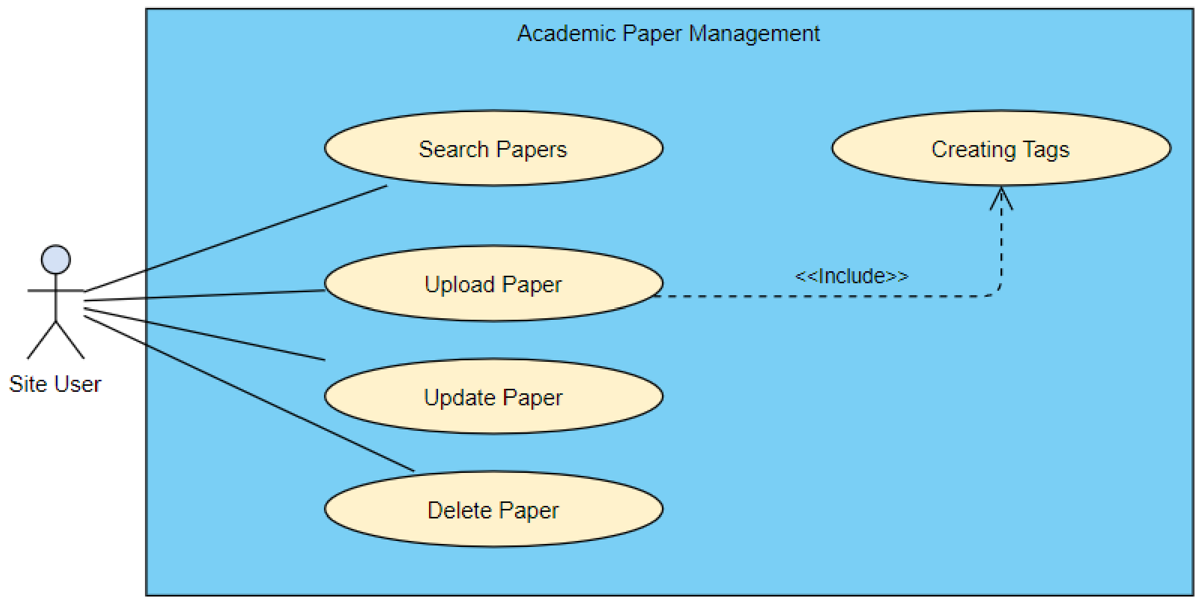
Q&A event	May 17

5.2.7 Summary

In this chapter it is possible to see the schedule for the development of the app split into Sprints. Each Sprint represents the cycle of development, testing and deployment to the QA (Quality Assurance) environment. Once a Sprint is deployed to the QA environment it is possible to test the system with actual users. After all the Sprints are set as done, there is the system test and finally the deployment in production.

5.3 System Analysis & Design

General Use case



5.3.1 Use case Uploading articles

Item	Description
Use Case ID	UCAD01
Use Case Name	Uploading articles
Actor:	Student/Professor
Trigger	Before Start the upload of the article
Business Role	Title cannot be empty The file must be *.pdf
Description	The user case uploads the article
Normal Flow	The user inserts the data, upload the pdf article, and save
Alternate Flow	User cancel the action Fail when uploading or saving the article and show the error
Cross reference	N/A

Main flow – Upload Article.

The use case starts when the user clicks on the link to add user. The user wants to insert an article or academic paper.

1. The user clicks on the button Upload article
2. The system opens the window to locate the file
3. User locate the file
4. User click on the button open
5. User fill up the other information about the article/paper
6. User click on the button save
7. The system will move the paper to a specified folder used in this project
8. The system will show the message “Paper Upload successfully”
9. Use case finished

Alternative flow

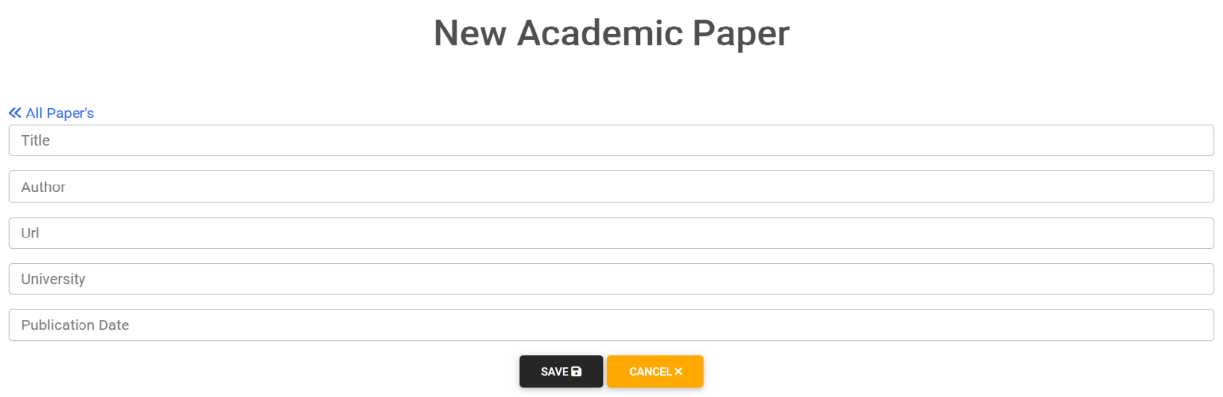
a) Cancel

1. The user clicks on a cancel button
2. The system cancels and clean any previews typed information.
3. The system returns to the main webpage.

a) Fail

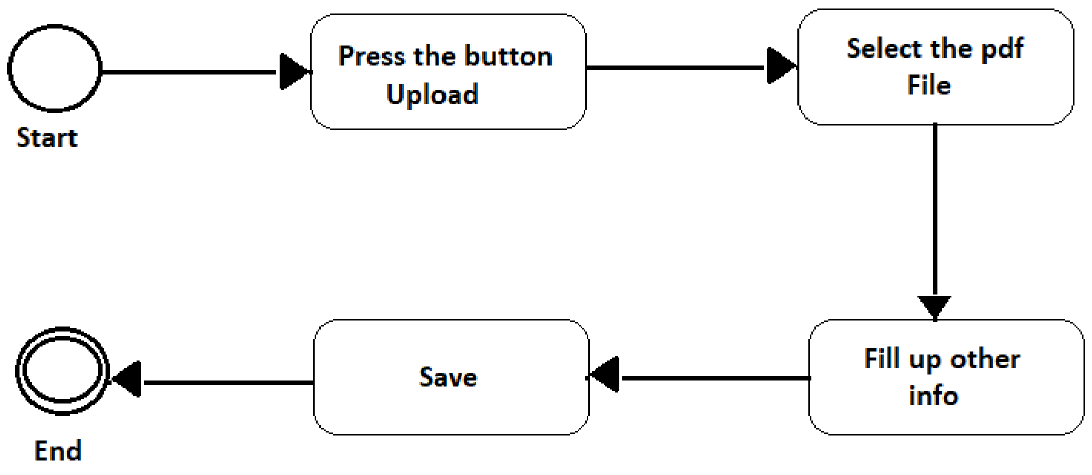
1. The Academic Paper Management system fails
2. The system shows the error message.
3. The system shows the main webpage.

Wireframe

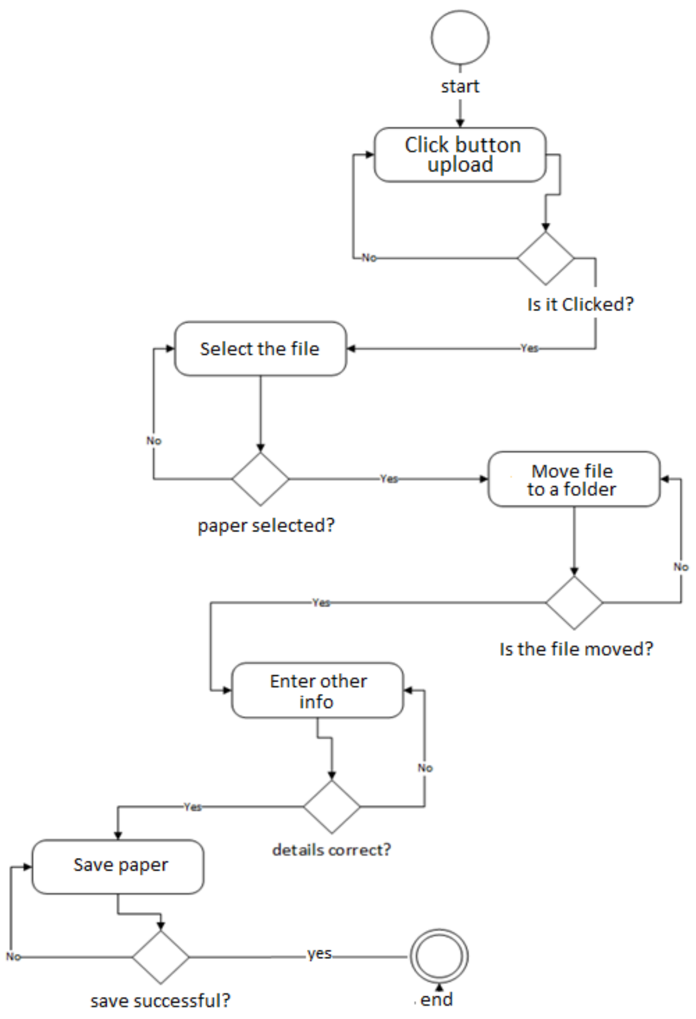


5.3.2 Diagrams

5.3.2.1 State Diagram



5.3.2.2 Activity Diagram



5.3.2 Use case Creating tags for grouping related topics

Item	**Description**
Use Case ID	UCAD02
Use Case Name	Creating tags
Actor:	Student/Professor
Trigger	While uploading the article
Business Role	Tag info cannot be empty
Description	The user chooses the tag or type new one before uploads the article
Normal Flow	The user selects or type the tag and save
Alternate Flow	N/A
Cross reference	UCAD01

Main flow – Creating Tags.

The use case starts when the user clicks on the button Save. The user wants to insert an article or academic paper.

1. The system generates the Tag information according with the data typed by the user
2. The system saves the tag on the database
3. Use case finished

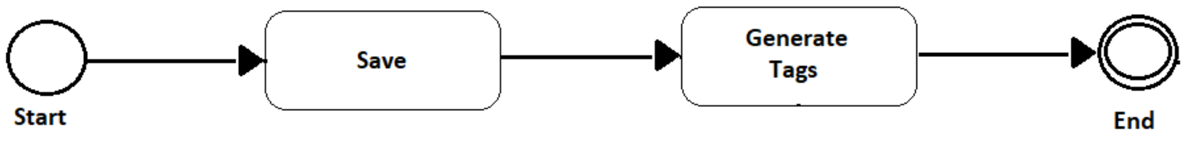
Alternative flow

a) Fail

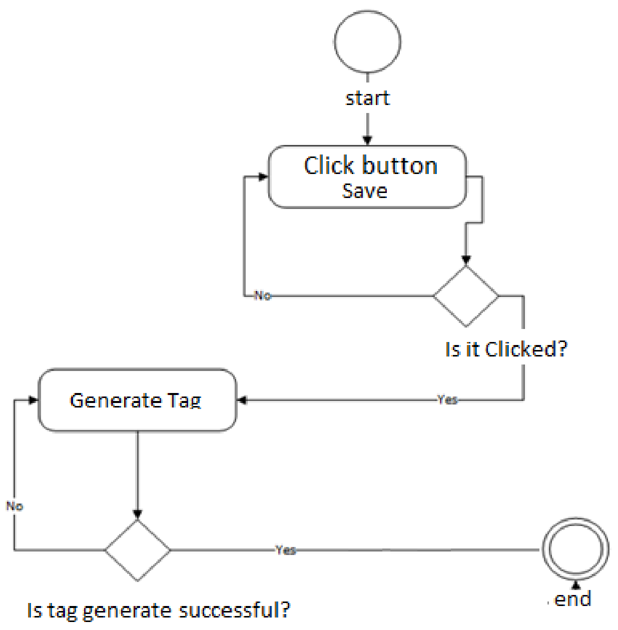
1. The Academic Paper Management system fails
2. The system shows the error message.
3. The system shows the main webpage.

5.3.3 Diagrams

5.3.3.1 State Diagram



5.3.3.2 Activity Diagram



5.3.3 Use case Search articles

Item	**Description**
Use Case ID	UCAD03
Use Case Name	Searching articles
Actor:	Student/Professor
Trigger	After upload of the article
Business Role	Title cannot be empty The table cannot be empty
Description	The user case searches the article
Normal Flow	The user searches the paper or article
Alternate Flow	Fail when uploading or saving the article and show the error
Cross reference	UCAD01 UCAD02

Main flow – Search Article.

The use case starts when the user types the title of the article and clicks on the Search button. The user wants to find an article or academic paper.

1. The system search for the in the database
2. The system prompts the list of similia paper in the table

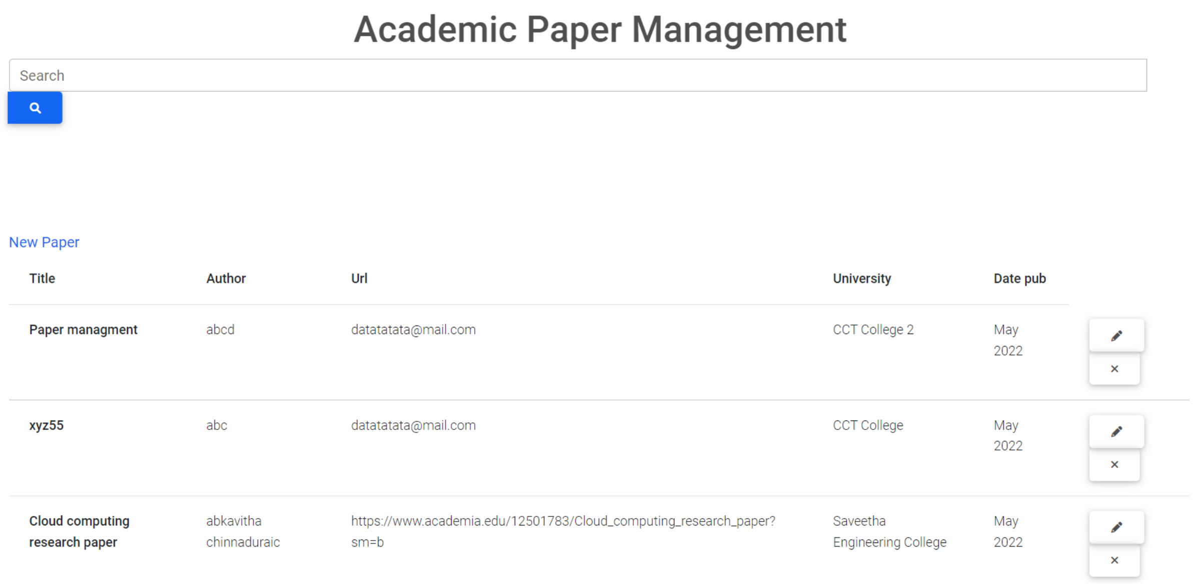
3. Use case finished

Alternative flow

a) Fail

4. The Academic Paper Management system fails
5. The system shows the error message.
6. The system refreshes the main webpage.

Wireframe


Academic Paper Management

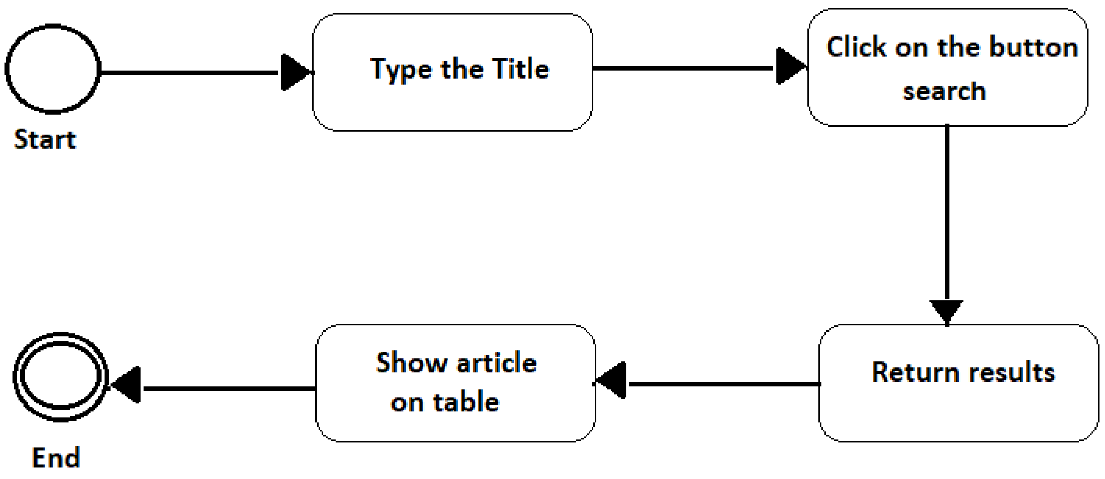
Search

New Paper

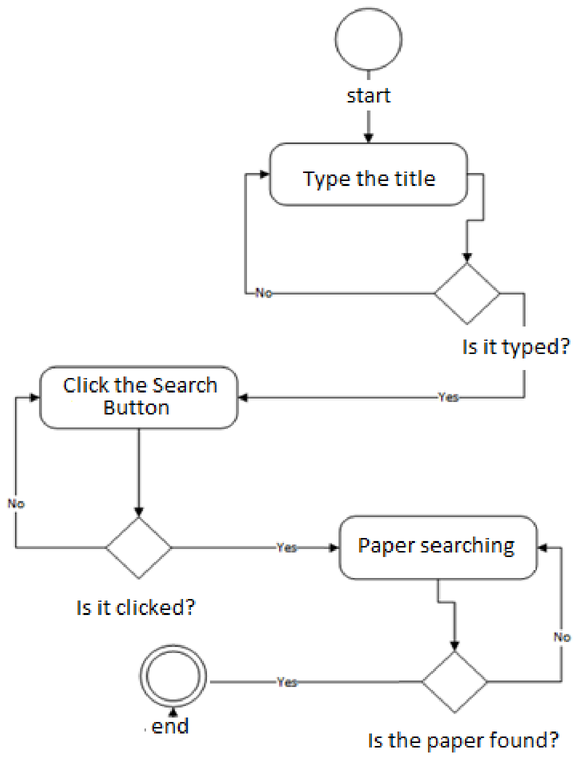
Title	Author	Url	University	Date pub
Paper managment	abcd	datatatata@mail.com	CCT College 2	May 2022
xyz55	abc	datatatata@mail.com	CCT College	May 2022
Cloud computing research paper	abkavitha chinnaduraic	https://www.academia.edu/12501783/Cloud_computing_research_paper?sm=b	Saveetha Engineering College	May 2022

5.3.4 Diagrams

5.3.4.1 State Diagram



5.3.4.2 Activity Diagram



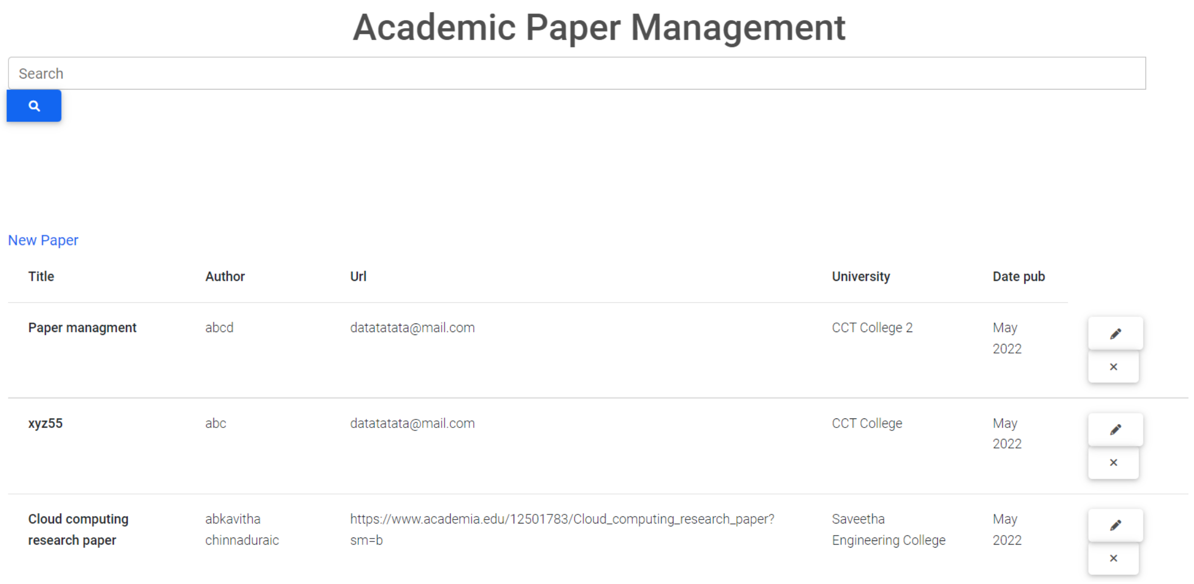
Proposed Implementation

An online system was created based on a simple and friendly screen that allows searching, viewing the data already registered, adding a button, updating, and deleting data.

Project available at GitHub link:

https://github.com/designerclau/academicresearch.git

Home page



Academic Paper Management

Search

New Paper

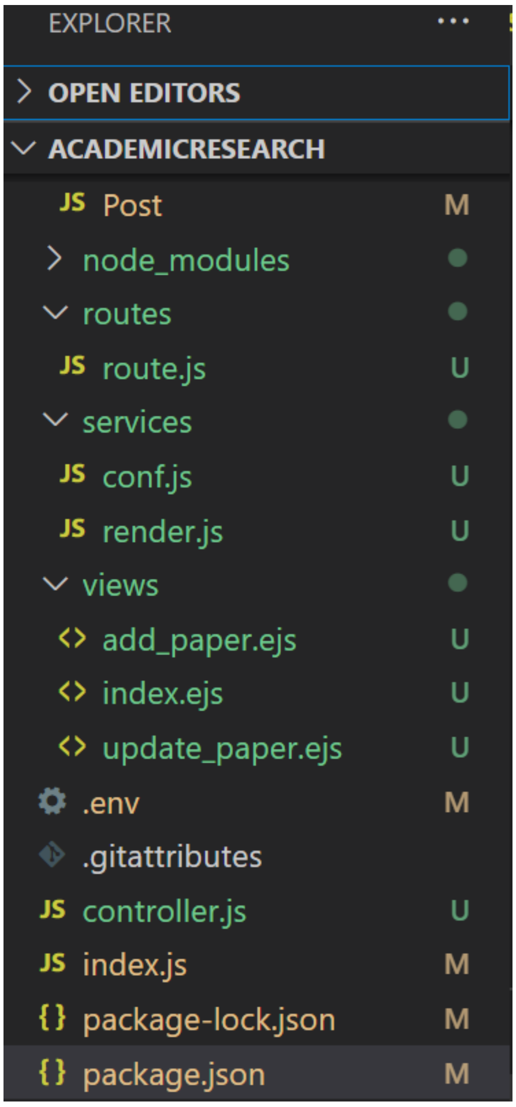
Title	Author	Url	University	Date pub
Paper managment	abcd	datatatata@mail.com	CCT College 2	May 2022
xyz55	abc	datatatata@mail.com	CCT College	May 2022
Cloud computing research paper	abkavitha chinnaduraic	https://www.academia.edu/12501783/Cloud_computing_research_paper?sm=b	Saveetha Engineering College	May 2022

Figure 4 - Home Page

1.2 Coding

1.2.1 Project Explorer

This is the structure of the system, with the files, and folders.



EXPLORER
OPEN EDITORS
ACADEMICRESEARCH
Post M
node_modules
routes
route.js U
services
conf.js U
render.js U
views
add_paper.ejs U
index.ejs U
update_paper.ejs U
.env M
.gitattributes
controller.js U
index.js M
package-lock.json M
package.json M

Figure 5 – Project Explorer

1.2.2 Package.json

In this file are listed the dependencies and libraries used in the system.

```
{
  "name": "academicresearch",
  "version": "1.0.0",
  "description": "Node.js, Express, MongoDB Altas web app",
  "main": "index.js",
  "scripts": {
    "devStart": "nodemon index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "academicresearch"
  },
  "author": "Claudinea de Almeida",
  "license": "ISC",
  "dependencies": {
    "axios": "^0.26.1",
    "body-parser": "^1.20.0",
    "dotenv": "^16.0.0",
    "ejs": "^3.1.6",
    "express": "^4.17.3",
    "express-session": "^1.17.2",
    "github": "^14.0.0",
    "mdb-ui-kit": "^3.11.0",
    "mongoose": "^6.2.10",
    "morgan": "^1.10.0",
    "path": "^0.12.7"
  },
  "devDependencies": {
    "nodemon": "^2.0.15"
  }
}
```

1.2.3 Index.js

This is the main file of this project, where the main functions and settings are found. On this screen I added the necessary libraries and created my routines.

The full index.js code is listed below.

```
//main file

const express = require('express');
const morgan = require('morgan');
const mongoose = require('mongoose');
const bodyparser = require('body-parser');
const papers = require('./model/Post');
const { request } = require('express');
const path = require('path');
const res = require('express/lib/response');
require('dotenv/config');
const session = require('express-session');


let responseObj = {
    "status": "",
    "msg": "",
    "body": {

    }
}
const app = express();
//requests timing
app.use(morgan('tiny'));
app.use(express.json());

//parser request to bodyparser
//app.use(bodyparser.json());
app.use(bodyparser.urlencoded({extended:true}));

//set view engine
app.set('view engine','ejs');
app.use('/', require('./routes/route'));

app.use("/css", express.static(path.join(__dirname,"node_modules/mdb-ui-
kit/css")));
app.use("/js", express.static(path.join(__dirname,"node_modules/mdb-ui-
kit/js")));

//connect to database
mongoose.connect( process.env.DB_connection, () =>
                  console.log('Connected to Db')
 );


const port = process.env.PORT || 3001;
app.listen(port, () => console.log(`Listening on port ${port} `));
```

1.2.4 Post.js

In this file you have the schema database, in MongoDB standards, using Mongoose to access this library.

```
//Database schema
const mongoose = require('mongoose');
const PostSchema = new mongoose.Schema({

    title: {
        type: String,
        required:false
    },
    author:{
        type: String,
        required:false
    },
    urlstring:{
        type: String,
        required:false
    },
    university:{
        type: String,
        required:false
    },
    datepub:{
        type: String,
        required:false
    }

});

module.exports = mongoose.model('academics',PostSchema);
```

1.2.5 Route.js

A router is a JavaScript object that maps URLs to functions. The router calls a function based on the URL.

```
const express = require('express');
const route = express.Router();

const services = require('../services/render');
const controller = require('../controller');

route.get('/', services.homeRoutes);

route.get('/add_paper', services.add_paper);

route.get('/update_paper', services.update_paper);

//API to create users
route.post('/api/papers', controller.create);

//retrieve and return all papers or a single one
route.get('/api/papers', controller.find);

//update paper by id
route.put('/api/papers/:id', controller.update);

//delete paper by id
route.delete('/api/papers/:id', controller.delete);

module.exports = route
```

1.2.6 Conf.js

A conf is a JavaScript file that execute functions while submitting the buttons.

```
$("#add_paper").submit(function(event){
  alert("Data inserted successfuly")
})

$("#update_paper").submit(function(event){
    event.preventDefault();
    var unindexed_array = $(this).serializeArray();
    var data ={};

    $.map(unindexed_array,function(n,i){
        data[n['title']]= n['value']
    })
    console.log(data);
    var request = {
        "url":'http://localhost:3001/api/papers/${data.id}',
        "method":"PUT",
        "data":data
    }

    $.ajax(request).done(function(response){
        alert("Data updated successfuly")
    })


  })
```

1.2.7 Render.js

JavaScript uses the Document Object Model (DOM) to manipulate DOM elements. Rendering refers to showing the output in the browser. The DOM establishes parent-child relationships and adjacent sibling relationships between the various elements in the Ejs file.

```
const axios = require('axios');
const { param } = require('../routes/route');

exports.homeRoutes=(req,res) =>{
    //make a get request to the api/papers
    axios.get('http://localhost:3001/api/papers')
```

```
      .then(function(response){
        res.render("index",{papers:response.data});
      })
      .catch(err =>{
          res.send(err);
      })
}


exports.add_paper=(req,res) =>{
    res.render("add_paper")
}

exports.update_paper=(req,res) =>{
    axios.get('http://localhost:3001/api/papers', {params:{id:req.query.id}})
    .then(function(paperdata){
        res.render("update_paper",{papers:paperdata.data});
    })
    .catch(err =>{
          res.send(err);
    })

}
```

1.2.7 Views

View engines are useful for rendering web pages.

EJS simply means inline JavaScript. It is a simple modelling language/engine that allows the user to generate HTML with simple JavaScript.

a) Index.ejs

This is the main webpage, the home page of the website.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-
to-fit=no" />
    <meta http-equiv="x-ua-compatible" content="ie=edge" />
    <title>Academic Paper Management</title>
    <!-- MDB icon -->
    <link rel="icon" href="img/mdb-favicon.ico" type="image/x-icon" />
    <!-- Font Awesome -->
    <link
      rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css"
    />
    <!-- Google Fonts Roboto -->
    <link
      rel="stylesheet"

href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700;900&
display=swap"
    />
    <!-- MDB -->
    <link rel="stylesheet" href="./css/mdb.min.css" />
  </head>
  <body>

    <!-- Start your project here-->
    <div class="container">
      <div class="d-flex justify-content-center align-items-center"
style="height: 50vh">
        <div class="text-center">

            <h1>  Academic Paper Management  </h1>

            <div class="input-group" >
              <div class="form-outline" >

                <input type="search" id="form1" class="form-control"
size="150px" "/>
                <label class="form-label" for="form1">Search</label>
              </div>

                <button type="button" class="btn btn-primary">
                 <i class="fas fa-search"></i>
              </button>


            </div>
          </div>
        </div>
      </div>
      <div class="container">
            <div class="box-nav d-flex-justify-between">
                <a href="/add_paper" class="border-shadow">
                    <span class="text-gradient"> New Paper <i class="fa-solid
fa-memo"></i></i></span>
                </a>
            </div>
              <table class="table">
                <thead>
                  <tr>
```

```
                    <th scope="col">Title</th>
                    <th scope="col">Author</th>
                    <th scope="col">Url</th>
                    <th scope="col">University</th>
                    <th scope="col">Date pub</th>
                  </tr>
                </thead>
                <tbody>
                  <% for(var i = 0; i < papers.length; i++){ %>

                  <tr>
                    <th scope="row"><%= papers[i].title %></th>
                    <td><%= papers[i].author %></td>
                    <td><%= papers[i].urlstring %></td>
                    <td><%= papers[i].university %></td>
                    <td><%= papers[i].datepub %></td>
                    <td>
                      <a href="/update_paper?id=<%= papers[i]._id %>"
class="btn border-shadow update">
                        <span class="text-gradient"> <i class="fas fa-pencil-
alt"></i></span>
                      </a>
                      <a class="btn border-shadow delete">
                        <span class="text-gradient"> <i class="fas fa-
times"></i></span>
                      </a>
                    </td>
                  </tr>
                  <% } %>
                </tbody>
              </table>

            </div>




    <!-- End your project here-->

    <!-- MDB -->
    <script type="text/javascript" src="./js/mdb.min.js"></script>


    <!-- Custom scripts -->
    <script type="text/javascript"></script>
  </body>
</html>
```

b) Add_paper.ejs

Webpage to add new paper.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-
to-fit=no" />
    <meta http-equiv="x-ua-compatible" content="ie=edge" />
    <title>Academic Paper Management</title>
    <!-- MDB icon -->
    <link rel="icon" href="img/mdb-favicon.ico" type="image/x-icon" />
    <!-- Font Awesome -->
    <link
      rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css"
    />
    <!-- Google Fonts Roboto -->
    <link
      rel="stylesheet"

href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700;900&
display=swap"
    />
    <!-- MDB -->
    <link rel="stylesheet" href="./css/mdb.min.css" />
  </head>
  <body>

    <!-- Start your project here-->

    <div class="container">
       <div class="d-flex justify-content-center align-items-center"
style="height: 20vh">
          <div class="text-center">

             <h1>  New Academic Paper  </h1>
          </div>
       </div>
        <div class="box-nav d-flex-justify-between style="height: 10vh">
             <a href="/" class="border-shadow">
                <span class="text-gradient"> <i class="fas fa-angle-double-
left"></i> All Paper's </span>
             </a>
        </div>
        <form action="/api/papers" method="POST" id="add_paper">
```

```html
            <div class="input-group mb-3">
               <input type="text" class="form-control" name="title"
placeholder="Title" aria-label="Title" aria-describedby="title">
            </div>

            <div class="input-group mb-3">
               <input type="text" class="form-control" name="author"
placeholder="Author" aria-label="Author" aria-describedby="title">

            </div>

            <div class="input-group mb-3">
               <input type="text" class="form-control" id="urlstring"
placeholder="URL" aria-describedby="urlstring">
            </div>

            <div class="input-group mb-3">
              <input type="text" class="form-control" name="university"
placeholder="University" aria-label="University" aria-describedby="university">
           </div>

           <div class="input-group mb-3">
              <input type="text" class="form-control" name="datepub"
placeholder="Publication Date" aria-label="datepub" aria-describedby="datepub">

           </div>

          <div class="col-md-12 text-center">
            <button type="submit" class="btn btn-dark">Save <i class="fa-solid
fa-floppy-disk"></i></button>
            <button type="button" class="btn btn-warning">
               <span class="text-gradient"> Cancel <i class="fas fa-
times"></i></span>
            </button>
          </div>

        </form>


            </div>





    <!-- End your project here-->

    <!-- MDB -->
    <script type="text/javascript" src="./js/mdb.min.js"></script>
    <script src="../services/conf.js"></script>
    <!-- Custom scripts -->
    <script type="text/javascript"></script>
  </body>
</html>
```

c) Update_paper.ejs

Webpage to update a paper already in the application.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-
to-fit=no" />
    <meta http-equiv="x-ua-compatible" content="ie=edge" />
    <title>Academic Paper Management</title>
    <!-- MDB icon -->
    <link rel="icon" href="img/mdb-favicon.ico" type="image/x-icon" />
    <!-- Font Awesome -->
    <link
      rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0/css/all.min.css"
    />
    <!-- Google Fonts Roboto -->
    <link
      rel="stylesheet"

href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700;900&
display=swap"
    />
    <!-- MDB -->
    <link rel="stylesheet" href="./css/mdb.min.css" />
  </head>
  <body>

    <!-- Start your project here-->

    <div class="container">
       <div class="d-flex justify-content-center align-items-center"
style="height: 20vh">
         <div class="text-center">

            <h1>  Update Academic Paper  </h1>
         </div>
       </div>
        <div class="box-nav d-flex-justify-between style="height: 10vh">
```

```
             <a href="/" class="border-shadow">
                <span class="text-gradient"> <i class="fas fa-angle-double-
left"></i> All Paper's </span>
             </a>
        </div>
        <form method="Post" id="update_paper">

            <div class="input-group mb-3">
              <input type="hidden" class="form-control" name="id" value="<%=
papers._id %>">
               <input type="text" class="form-control" name="title" value="<%=
papers.title %>" placeholder="Title" aria-label="Title" aria-
describedby="title">
            </div>

            <div class="input-group mb-3">
               <input type="text" class="form-control" name="author" value="<%=
papers.author %>" placeholder="Author" aria-label="Author" aria-
describedby="title">

            </div>

            <div class="input-group mb-3">
                 <input type="text" class="form-control" id="urlstring"
value="<%= papers.urlstring %>" placeholder="urlstring" aria-
describedby="urlstring">
            </div>

            <div class="input-group mb-3">
              <input type="text" class="form-control" name="university"
value="<%= papers.university %>" placeholder="University" aria-
label="University" aria-describedby="university">
           </div>

           <div class="input-group mb-3">
              <input type="text" class="form-control" name="datepub" value="<%=
papers.datepub %>" placeholder="Publication Date" aria-label="datepub" aria-
describedby="datepub">

           </div>

          <div class="col-md-12 text-center">
            <button type="submit" class="btn btn-dark">Save <i class="fa-solid
fa-floppy-disk"></i></button>
            <button type="button" class="btn btn-warning">
               <span class="text-gradient"> Cancel <i class="fas fa-
times"></i></span>
            </button>
          </div>
        </form>

            </div>

    <!-- End your project here-->

    <!-- MDB -->
    <script type="text/javascript" src="./js/mdb.min.js"></script>
    <script type="submit">
      $("#update_paper").submit(function(event){
        event.preventDefault();
        var unindexed_array = $(this).serializeArray();
        var data ={};

        $.map(unindexed_array,function(n,i){
           data[n['title']]= n['value']
         })
         console.log(data);
         var request = {
           "url":'http://localhost:3001/api/papers/${data.id}',
           "method":"PUT",
           "data":data
          }

        $.ajax(request).done(function(response){
        alert("Data updated successfuly")
         })


      })
    </script>

    <!-- Custom scripts -->
    <script type="text/javascript"></script>
  </body>
</html>
```

1.2.8 Controller.ejs

It contains functions that separate the code to route requests from the code that processes requests.

```
var paperDB = require('./model/Post');

//create and save new paper
exports.create = (req,res)=>{
    //validate the request
    if(!req.body){
        res.status(400).send({message:"Content cannot be empty"})
        return;
    }

    //new paper

      const newPaper = new paperDB({
        title:req.body.title,
        author:req.body.author,
        urlstring:req.body.urlstring,
        university:req.body.university,
        datepub:req.body.datepub
       });

       //save paper

       newPaper
         .save(newPaper)
         .then(data =>{
            // res.send(data)
             res.send({message: "Paper inserted successfuly"});
             res.redirect('/');
         })
         .catch(err =>{
             res.status(500).send({
                 message: err.message||"Some error occured"
             });
         });
}

//find papers or a single paper
exports.find = (req,res)=>{
    if(req.query.id){

        const id = req.query.id;

        paperDB.findById(id)
            .then(data =>{
                if(!data){
                   res.status(404).send({message: "Cannot find a paper with id
"+id+". Maybe paper not found!"})
                }else{
                   res.send(data);
                }
            })
            .catch(err =>{
                res.status(500).send({
                    message: err.message||"Some error occured while finding the
paper"
                });
            });
    }else{
        paperDB.find()
        .then(data =>{
          res.send(data)
      })
      .catch(err =>{
          res.status(500).send({
              message: err.message||"Some error occured"
          });
      });
    }

}

//update papers by id
exports.update = (req,res)=>{
   //validate the request
   if(!req.body){
      res.status(400).send({message:"Content cannot be empty"})
      return;
   }

  //update paper
  const id = req.params.id;
  paperDB.findByIdAndUpdate(id,req.body,{useFindAndModify:false})
     .then(data =>{
         if(!data){
            res.status(404).send({message: "Cannot update paper with id ${id}.
Maybe paper not found!"})
         }else{
            res.send(data);
         }

     })
     .catch(err =>{
         res.status(500).send({
             message: err.message||"Some error occured"
         });
     });
}

//delete papers by id
exports.delete = (req,res)=>{
    const id = req.params.id;
    paperDB.findByIdAndDelete(id)
     .then(data =>{
         if(!data){
            res.status(404).send({message: "Cannot delete paper with id ${id}.
Maybe id is wrong!"})
         }else{
            res.send({message: "Paper deleted successfuly"});
         }

     })
     .catch(err =>{
         res.status(500).send({
             message: err.message||"Could not delete the paper with id ${id}"
         });
     });
}
```

1.2.9 Other files

a) Env

In this file, put a configuration to access the MongoDB database, in order to protect this information.

DB_connection =
mongodb+srv://admin:Academic12@academicsearch.6msru.mongodb.net/paper?retryWrites=true&w=majority

b) Dockerfile

After completing my project, I created this file to set up the Docker container for my project.

```
FROM node:carbon

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 3001

CMD ["npm" , "start"]
```

After that, I created the docker image with the command:

```
docker build -t claudinea/academicresearch
```

And I drove the docker. I used a different port than the original one.

```
docker run -p 49160:3001 -d claudinea/academicresearch
```

Proposed Evaluation

Research Timeline

The Gantt chart shows the basic timeline of the project

Task	Feb	Mar	April	May
Proposal initiation	🟩			
Learning concepts	🟩	🟩		
Dataset preparation		🟩		
Study of Program Language, tools, and libraries		🟩	🟩	
Developing Uploading Article			🟩	
Developing Tag			🟩	
Developing search			🟩	
Evaluation of results				

Figure 4 – Project Gantt chart

Conclusion

References

Docs.mongodb.com. 2021. [online] Available at: <https://docs.mongodb.com/manual/text-search/> [Accessed 27 December 2021].

W3schools.com. 2021. *Node.js MongoDB Find*. [online] Available at: <https://www.w3schools.com/nodejs/nodejs_mongodb_find.asp> [Accessed 27 December 2021].

Youtu.be. 2021. *Build A REST API With Node.js, Express, & MongoDB - Quick*. [online] Available at: <https://youtu.be/fgTGADljAeg> [Accessed 27 December 2021].

Youtu.be. 2021. *Build A Restful Api With Node.js Express & MongoDB | Rest Api Tutorial*. [online] Available at: <https://youtu.be/vjf774RKrLc> [Accessed 27 December 2021].

Youtu.be. 2021. *Como criar um mecanismo de busca em Node.js e MongoDB*. [online] Available at: <https://youtu.be/qLO8Q870fmc> [Accessed 27 December 2021].

Khalil, Y., 2018. *Development of a web-based system for thesis and project proposal management*. [online] Academia.edu. Available at: <https://www.academia.edu/70051130/Development_of_a_web_based_system_for_thesis_and_project_proposal_management> [Accessed 10 April 2022].

Agyepong, S., 2011. *MSEARCH: A Mobile Search Service*. [online] Academia.edu. Available at: <https://www.academia.edu/70362064/MSEARCH_A_Mobile_Search_Service> [Accessed 5 March 2022].

Dasic, P. and Crvenkovic, B., 2016. *Applications of the Search as a Service (SaaS)*. [online] Academia.edu. Available at: <https://www.academia.edu/29837306/Applications_of_the_Search_as_a_Service_SaaS_> [Accessed 5 March 2022].

Dašić, P., Dašić, J. and Crvenković, B., 2016. *Service Models for Cloud Computing: Search as a Service (SaaS)*. [online] Academia.edu. Available at: <https://www.academia.edu/29576005/Service_Models_for_Cloud_Computing_Search_as_a_Service_SaaS_?pop_sutd=false> [Accessed 5 March 2022].

Kanade, S. and Manza, R., 2019. *A Comprehensive Study on Multi Tenancy in SAAS Applications*. [online] Academia.edu. Available at: <https://www.academia.edu/61390511/A_Comprehensive_Study_on_Multi_Tenancy_in_SAAS_Applications> [Accessed 5 March 2022].

s, s., 2012. *CLOUD COMPUTING : SAAS*. [online] Academia.edu. Available at: <https://www.academia.edu/21491005/CLOUD_COMPUTING_SAAS> [Accessed 5 March 2022].

Samir, A., n.d. *An Aspect-Oriented Approach for SaaS Application Customization*. [online] Academia.edu. Available at: <https://www.academia.edu/65168311/An_Aspect_Oriented_Approach_for_SaaS_Application_Customization> [Accessed 5 March 2022].

MongoDB. n.d. *What Is SaaS? Software-as-a-Service Explained | MongoDB*. [online] Available at: <https://www.mongodb.com/cloud-explained/saas-software-as-a-service> [Accessed 1 April 2022].