

Neoronetix AI (Artificial Intelligence)

by

Md. Rabiussani

ID: CSE1902017077

Md Moniruzzaman

ID: CSE1902017065

Shahin Alom

ID: CSE1902017064

Bijoy Kumar Das

ID: CSE1902017037

Supervised by

Mohammad Naderuzzaman

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science
in Computer Science and Engineering



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SONARGAON UNIVERSITY (SU)**

January 2023

APPROVAL

The Thesis titled “**Neoronetix AI (Artificial Intelligence)**” submitted by Md. Rabiul Sani (CSE1902017077), Md. Moniruzzaman (CSE1902017065), Shahin Alom (CSE1902017037), Bijoy Kumar Das (CSE1902017064) to the Department of Computer Science and Engineering, Sonargaon University (SU), has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and contents.

Board of Examiners

Mohammad Naderuzzaman

Assistant Professor,
Department of Computer Science and Engineering
Sonargaon University (SU)

Supervisor

(Examiner Name & Signature)

Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 1

(Examiner Name & Signature)

Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 2

(Examiner Name & Signature)

Department of Computer Science and Engineering
Sonargaon University (SU)

Examiner 3

DECLARATION

We, hereby, declare that the work presented in this report is the outcome of the investigation performed by us under the supervision of **Mohammad Naderuzzaman**, Lecturer, Department of Computer Science and Engineering, Sonargaon University, Dhaka, Bangladesh. We reaffirm that no part of this Thesis has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

Signature

(**Mohammad Naderuzzaman**)
Assistant Professor

Md. Rabius Sani
ID: CSE1902017077

Md Moniruzzaman
ID: CSE1902017065

Shahin Alom
ID: CSE1902017064

Bijoy Kumar Das
ID: CSE1902017037

ABSTRACT

Neuronetix AI is a powerful artificial intelligence system that utilizes advanced neural network technology to deliver highly accurate predictions and insights. Our system is capable of processing large amounts of data quickly and accurately, making it a valuable tool for businesses in a wide range of industries. With its ability to analyze complex data sets, Neuronetix AI can help businesses improve sales predictions, optimize supply chains, and gain new insights into customer behavior. Additionally, the system can be customized to meet the specific needs of any organization, making it a versatile solution for any business looking to gain a competitive edge in today's data-driven world.

ACKNOWLEDGMENT

At the very beginning, we would like to express our deepest gratitude to the Almighty Allah for giving us the ability and the strength to finish the task successfully within the schedule time.

We would then like to thank our supervisor, **Mohammad Naderuzzaman** for introducing us to the amazingly interesting world of Data Mining Machine Learning and Data Mining. And she is the person who taught us how to perform research work efficiently. Without she and her continuous supervision, guidance and valuable advice, it would have been impossible for us to come at this point and have some output from the thesis. We are especially grateful to Ma'am for allowing us greater freedom in choosing the problems to work on, for his encouragement at times of disappointment.

We would like to convey our gratitude to **Prof. Dr Md Alamgir Hossain**, (Dean, Faculty of Science & Engineering) and special gratitude our honorable departmental head **Bulbul Ahamed**, (Associate Professor & Head, Department of Computer Science and Engineering) for their kind concern, discretion, friendly behavior and precious suggestions.

We would like to express our gratitude to all our teachers. Their motivation and encouragement in addition to the education they provided meant a lot to us.

Last but not the least, we would like to thank our family, our parents, for their encouragement, endless love and for supporting us spiritually throughout our lives.

TABLE OF CONTENTS

Title	Page No.
DECLARATION	iii
ABSTRACT	iv
ACKNOWLEDGEMENT	v
CHAPTER 1	
INTRODUCTION.....	1 – 8
1.1 Introduction.....	1
1.2 What is Neuronetix AI?.....	1-2
1.3 Why AI is Important?.....	3-4
1.4 Real Life Applications of AI.....	4-5
1.5 How Does AI Works?.....	5-6
CHAPTER 2	
MAKING WEBSITE FOR AFFILIATE MARKETING	09-12
2.1 Executive Summary	09
2.2 Introduction.....	09
2.3 Market Analysis	10-11
2.4 Business Model	10-11
2.5 Financial Projections.....	11-12
2.6 Conclusion.....	12
CHAPTER 3	
NEORONETIX AI FRONT END, BACKEND RAW CODE	13-27
3.1 Admin User API.....	13-16
3.2 Admin API.....	16-18
3.3 Section API.....	18-21
3.4 Tag API.....	21-25
3.5 Template API.....	25-28
3.6 Website API.....	28-20
CHAPTER 4	
WRITING SECTION	31-53
4.1 Create AI Section.....	31-32

4.2	Create Loop Section.....	32-35
4.3	Format Content.....	35-38
4.4	Generate Article.....	38-44
4.5	Get Google Image.....	44-48
4.6	Get Youtube Videos.....	48-52
4.7	Key Manager.....	52-57
4.8	Process Queue.....	57-60
4.9	Publish Post on WP.....	60-62
4.1	Utility Function.....	62-63
CHAPTER 5		
SERVER		64-72
5.1	App Js File.....	64-68
5.2	Router.....	68-69
5.2	Server.....	69-72
CHAPTER 6		
ADMIN PANEL		74-78
6.1	Site List.....	74
6.2	Site Add.....	74
6.3	Section List.....	75
6.4	Section Add.....	75
6.5	Template List.....	76
6.6	Template Add.....	76
6.7	Generate Article.....	77
6.8	Article List.....	77
6.9	Setting (Manage Token).....	78
CHAPTER 7		
OUTPUT		79
7.1	Website Post.....	79

LIST OF FIGURES

<u>Figure No.</u>	<u>Title</u>	<u>Page No.</u>
Fig.1.2	Artificial Intelligence	2
Fig.1.3	Why AI is Important	4
Fig.1.4	Applications of AI	6
Fig.1.6	Benefits of AI	7
Fig.6.1	Site List	74
Fig.6.2	Site Add	74
Fig.6.3	Section List	75
Fig.6.4	Section Add	75
Fig.6.5	Template List	76
Fig.6.6	Template Add	76
Fig.6.7	Generate Article	77
Fig.6.8	Article List	77
Fig.6.9	Setting (Manage Token)	78
Fig.7.1	Website Post	79

CHAPTER 1

INTRODUCTION

1.1 Introduction

Artificial Intelligence (AI) is a rapidly growing field that has the potential to revolutionize the way we live and work. One of the most promising areas of AI is the use of neural networks, which are systems of algorithms that are designed to recognize patterns in data. Neuronetix AI is an advanced AI system that utilizes cutting-edge neural network technology to deliver highly accurate predictions and insights. This system is designed to handle large amounts of data quickly and accurately and can be applied to a wide range of industries, including finance, healthcare, retail, and many more.

Neuronetix AI is a powerful tool for businesses looking to gain a competitive edge in today's data-driven world. With its ability to analyze complex data sets, Neuronetix AI can help businesses improve their sales predictions, optimize their supply chains, and gain new insights into customer behavior. Furthermore, the system can be customized to meet the specific needs of any organization, making it a versatile solution for any business looking to gain a competitive edge in today's data-driven world.

The system can help to improve decision making by providing accurate predictions and insights, it can also help to automate certain processes, and ultimately increase efficiency and reduce costs. With the help of Neuronetix AI, businesses can stay ahead of the curve and make better decisions faster.

1.2 What is Neuronetix AI?

AI, or Artificial Intelligence, is the simulation of human intelligence processes by machines, especially computer systems. These processes include learning (the ability to improve performance based on experience), reasoning (using rules to reach approximate or definite conclusions), and self-correction.

Neuronetix AI is an advanced Artificial Intelligence system that utilizes neural network technology to deliver highly accurate predictions and insights. It is designed to handle complex data sets and can be applied to a wide range of industries, such as finance, healthcare, retail, and more. The system uses neural networks, which are systems of algorithms that are designed to recognize patterns in data, to analyze this data and make predictions or insights from it.

In summary, AI is the broad field of study that focuses on creating machines that can perform tasks that would typically require human intelligence. Neuronetix AI is a specific application of AI that uses neural network technology to analyze complex data sets and make predictions or insights.

Neuronetix AI is a powerful tool that can help businesses make better decisions by providing accurate predictions and insights, which can then be used to optimize operations and improve performance. For example, in the finance industry, Neuronetix AI can be used

to analyze market data and make predictions about stock prices. In healthcare, the system can be used to analyze patient data and make predictions about treatment outcomes. In retail, Neuronetix AI can be used to analyze sales data and make predictions about consumer behavior.

The system can also be used to automate certain processes, such as data analysis and decision making. This can help to increase efficiency and reduce costs for businesses. Additionally, Neuronetix AI can be used to create personalized experiences for customers by analyzing their data and behavior patterns.

The system is highly customizable, it can be tailored to specific needs of the user, this means that it can be used for a wide range of applications and industries, including finance, healthcare, retail, manufacturing, energy, and more. The system also constantly updates itself based on data and feedback, which allows it to continuously improve its predictions and insights.

In conclusion, Neuronetix AI is a cutting-edge technology that can help businesses gain a competitive edge in today's data-driven world. By providing accurate predictions and insights, automating processes, and creating personalized experiences, Neuronetix AI can help businesses improve their operations, increase efficiency, and ultimately drive growth.



Figure 1.2: Artificial Intelligence

1.3 Why AI is Important?

AI is important because it has the potential to revolutionize the way we live and work. Here are a few key reasons why:

Automation: AI can automate repetitive or mundane tasks, which can help to increase efficiency and reduce costs for businesses. This can also free up human workers to focus on more complex or higher-level tasks.

Improved decision-making: AI can analyze large amounts of data quickly and accurately, which can help businesses make better decisions. For example, AI can be used to analyze market data and make predictions about stock prices, or to analyze patient data and make predictions about treatment outcomes.

Personalization: AI can create personalized experiences for customers by analyzing their data and behavior patterns. This can help businesses to improve customer satisfaction and increase sales.

Advancements in various fields: AI can also be used to improve various industries and fields such as healthcare, transportation, education, finance, and many more.

Efficiency and Speed: AI can process and analyze large amount of data quickly and accurately, which can help businesses to make decisions faster.

Cost-Effective: AI can automate tasks, process data and make predictions, which can help to reduce costs for businesses.

In summary, AI has the potential to bring significant benefits to businesses and society as a whole by increasing efficiency, improving decision-making, creating personalized experiences, and advancing various fields.

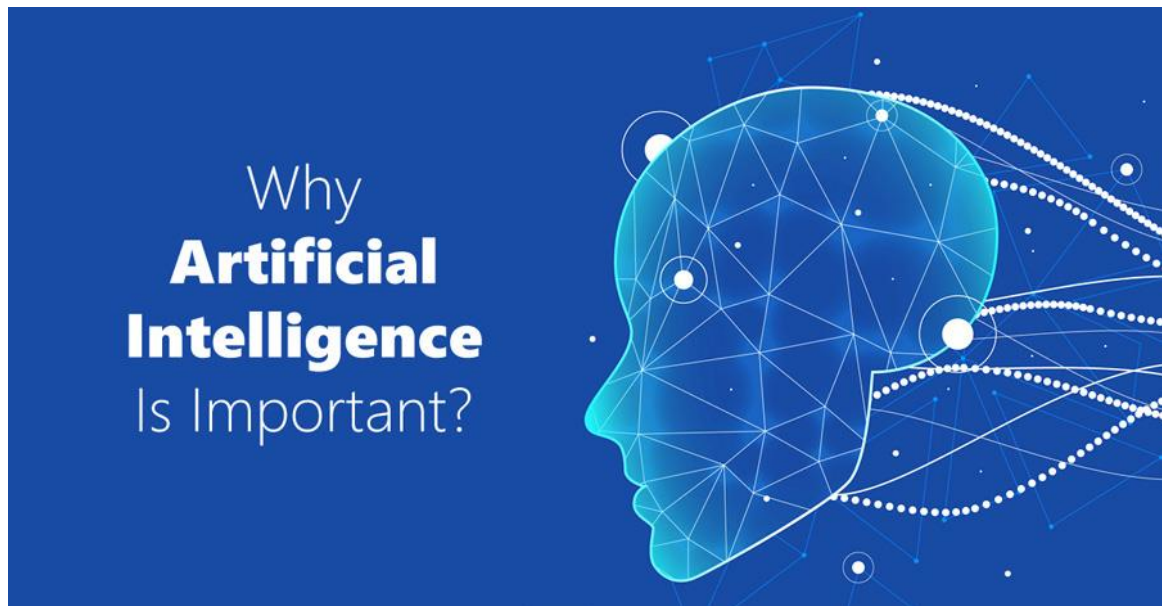


Figure: Why AI is Important

1.4 Real Life Applications of AI

There are many real-life applications of AI in various industries, some of the most significant examples include:

Healthcare: AI can be used to analyze medical images and make predictions about disease diagnosis and treatment outcomes. It can also be used to analyze patient data and monitor vital signs, which can help to improve patient care.

Finance: AI can be used to analyze market data and make predictions about stock prices, detect fraud, and automate trading decisions.

Retail: AI can be used to analyze sales data and make predictions about consumer behavior, which can help businesses to optimize inventory and improve marketing strategies.

Manufacturing: AI can be used to optimize supply chain operations, improve production efficiency, and predict equipment failures.

Transportation: AI can be used in self-driving cars, traffic prediction and optimization, logistics, and fleet management.

Education: AI can be used to personalize learning, to provide automated grading, and to facilitate virtual tutoring.

Robotics: AI is being used in manufacturing, cleaning, and farming robots.

Language Processing: AI is being used for natural language processing, for example, in chatbots, language translation, and speech recognition.

Gaming: AI is being used in game design, game-playing and game-training.

Agriculture: AI is being used to improve crop yields, monitor crop health, and optimize irrigation systems.

These are just a few examples of how AI is being used in the real world. The field is constantly evolving, and new applications are being developed all the time.



Figure 1.4: Applications of AI

1.5 How does AI Works?

AI, or artificial intelligence, is a broad field that encompasses a range of techniques and technologies. However, at a high level, AI can be broken down into two main categories: rule-based systems and machine learning.

Rule-based systems: This type of AI uses a set of predefined rules to make decisions or perform tasks. For example, a rule-based system might be programmed to identify an object in an image by looking for specific features, such as shape or color. Rule-based systems can be simple to set up and understand, but they can be inflexible and may not be able to handle exceptions or new situations.

Machine learning: This type of AI uses algorithms to learn from data and make predictions or decisions. There are several different types of machine learning, including:

Supervised learning: This type of machine learning is used to train a model to make predictions based on labeled data, for example, data that has been labeled as "dog" or "cat."

Unsupervised learning: This type of machine learning is used to find patterns in unlabeled data, for example, grouping similar images together.

Reinforcement learning: This type of machine learning is used to train models to make decisions based on rewards or penalties, for example, training a game-playing AI to make the best move based on the current game state.

Deep Learning: Is a subset of Machine Learning, which uses neural networks to learn from data. It is used to recognize patterns in large data sets, and are used in image recognition, speech recognition, natural language processing, and many other applications.

In summary, AI systems can use a combination of rule-based systems and machine learning techniques to make decisions or perform tasks. The specific techniques used will depend on the application and the type of data available.

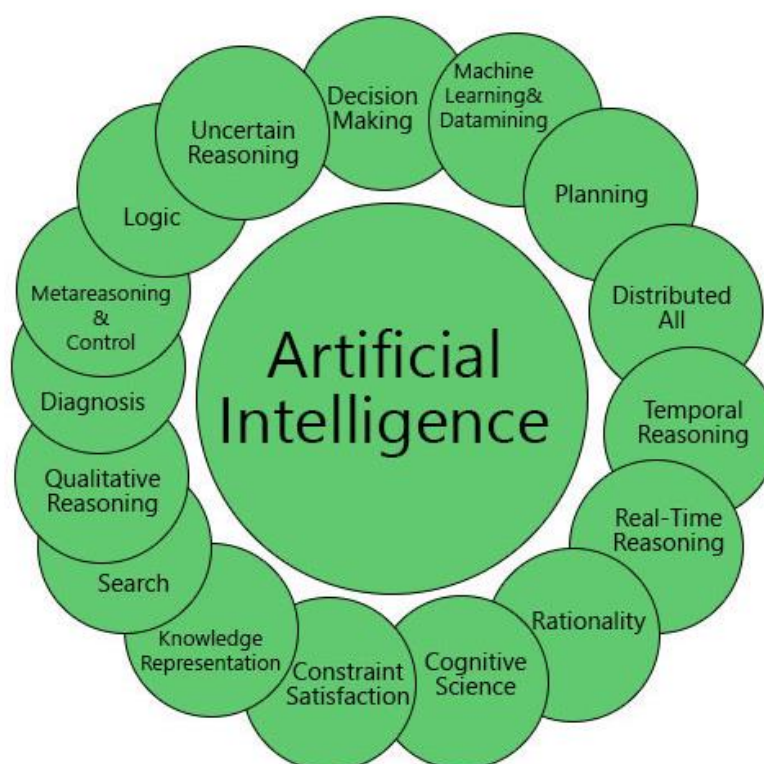


Figure 1.5: How Does AI Works

1.6 The Benefits of AI

There are many benefits of AI, which can improve various industries and fields, here are some of the most significant examples:

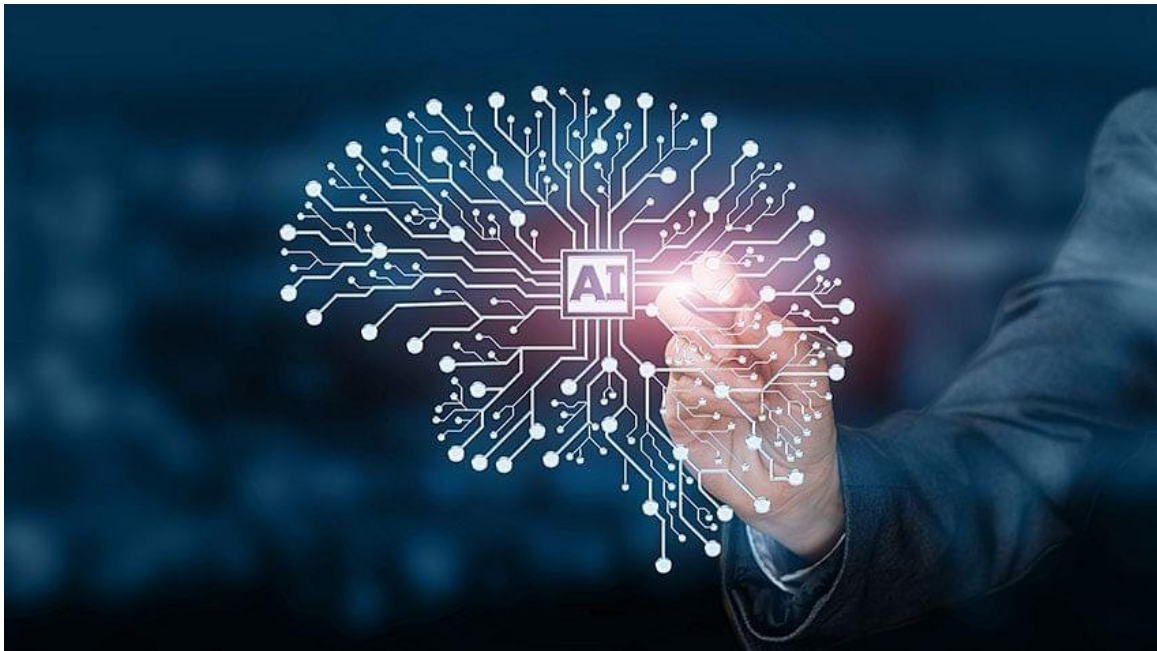
Automation: AI can automate repetitive or mundane tasks, which can help to increase efficiency and reduce costs for businesses.

Improved decision-making: AI can analyze large amounts of data quickly and accurately, which can help businesses make better decisions.

Personalization: AI can create personalized experiences for customers by analyzing their data and behavior patterns.

Advancements in healthcare: AI can be used to analyze medical images, make predictions about disease diagnosis, monitor vital signs, and improve patient care.

Advancements in finance: AI can be used to analyze market data and make predictions about stock prices, detect fraud, and automate trading decisions.



Figures: Benefits Of AI

Advancements in transportation: AI can be used in self-driving cars, traffic prediction and optimization, logistics, and fleet management.

Advancements in education: AI can be used to personalize learning, to provide automated grading, and to facilitate virtual tutoring.

Advancements in Robotics: AI is being used in manufacturing, cleaning, and farming robots.

Advancements in Language Processing: AI is being used for natural language processing, for example, in chatbots, language translation, and speech recognition.

Advancements in Agriculture: AI is being used to improve crop yields, monitor crop health, and optimize irrigation systems.

These are just some of the many benefits of AI. The field is constantly evolving, and new applications are being developed all the time, bringing more benefits to different fields and industries.

Chapter -2

MAKING WEBSITE FOR AFFILIATE MARKETING

2.1 Executive Summary

The Executive Summary is a brief overview of the key points of the Amazon Affiliate project. It should provide a clear and concise summary of the project's purpose, goals, market analysis, business model, marketing and sales strategy, financial projections, and implementation plan.

In the Executive Summary, it should be highlighted the main aspects of the project such as the target market, the products or services offered, the competition, the revenue streams, the main risks, and the expected return on investment.

Additionally, the Executive Summary should present the main objectives of the project, including short-term and long-term goals, the expected outcome, and the benefits of the project.

Overall, the Executive Summary is a crucial section that gives a quick snapshot of the entire project and is often the first thing that potential investors or partners will read. It should be written in a clear and concise manner, highlighting the key points and selling points of the project.

2.2 Introduction

The Executive Summary is a brief overview of the key points of the Amazon Affiliate project. It should provide a clear and concise summary of the project's purpose, goals, market analysis, business model, marketing and sales strategy, financial projections, and implementation plan.

In the Executive Summary, it should be highlighted the main aspects of the project such as the target market, the products or services offered, the competition, the revenue streams, the main risks, and the expected return on investment.

Additionally, the Executive Summary should present the main objectives of the project, including short-term and long-term goals, the expected outcome, and the benefits of the project.

Overall, the Executive Summary is a crucial section that gives a quick snapshot of the entire project and is often the first thing that potential investors or partners will read. It should be written in a clear and concise manner, highlighting the key points and selling points of the project.

2.3 Market Analysis

The Market Analysis section of an Amazon Affiliate project is an in-depth examination of the target market, the industry, and the competition. It should provide a detailed understanding of the market size, growth, and trends, as well as the key drivers and challenges of the industry.

In this section, it is important to identify the target market, including demographics, psychographics, and geographic location. It should also include an analysis of the buying habits, needs, and preferences of the target market.

Additionally, the Market Analysis should also include a competitive analysis, which examines the strengths and weaknesses of the main competitors in the market. This information can be used to identify opportunities and threats and to develop a competitive advantage.

Furthermore, it should also include a SWOT analysis (Strengths, Weaknesses, Opportunities and Threats) that provides a comprehensive overview of the project's internal and external environment.

Overall, the Market Analysis section should provide a thorough understanding of the target market, the industry, and the competition, and should identify the key opportunities and challenges for the project. This information can be used to develop a marketing and sales strategy that is tailored to the specific needs of the target market.

2.4 Business Model

The Business Model section of an Amazon Affiliate project is where the project's revenue streams and operating costs are outlined. It should describe how the project will generate revenue and make a profit.

In this section, it is important to describe the main components of the business model, such as the value proposition, the target customer, the revenue streams, and the operating costs.

The value proposition describes the unique benefit that the project will provide to the target market. It should clearly explain how the project will solve a problem or meet a need for the target customer.

The target customer should be described in detail, including demographics, psychographics, and geographic location. The revenue streams should be described, including any partnerships or collaborations that will be used to generate revenue.

The operating costs should be outlined, including the cost of goods sold, marketing and advertising expenses, and overhead costs.

Additionally, the Business Model section should also include a break-even analysis, which estimates the point at which the project will begin to generate a profit. This information can be used to determine the project's financial viability and to identify potential risks.

Overall, the Business Model section should provide a clear and concise overview of the project's revenue streams and operating costs, and should demonstrate how the project will

generate a profit. This information can be used to develop a financial projection and to determine the project's financial viability.

2.5 Marketing and Sales Strategy

The Marketing and Sales Strategy section of an Amazon Affiliate project outlines the plan for promoting and selling the project. It should provide a detailed understanding of how the project will reach and engage the target market and generate revenue.

In this section, it is important to describe the marketing and sales tactics that will be used to reach the target market, such as online and offline advertising, content marketing, public relations, and events.

Additionally, the Marketing and Sales Strategy should also include information on how the project will be priced and distributed, as well as any partnerships or collaborations that will be used to generate revenue.

In order to develop a marketing and sales strategy, it is important to conduct a thorough analysis of the target market and the competition. This information can be used to identify the key opportunities and challenges, and to develop a strategy that is tailored to the specific needs of the target market.

It should also include a detailed budget for the marketing and sales activities, and should include measurable objectives and performance indicators that will be used to track the success of the strategy.

Overall, the Marketing and Sales Strategy section should provide a clear and comprehensive plan for promoting and selling the project, and should demonstrate how the project will reach and engage the target market and generate revenue. This information can be used to develop a detailed budget and to track the success of the marketing and sales efforts.

2.6 Financial Projections

The Financial Projections section of an Amazon Affiliate project provides an estimate of the project's financial performance over a specific period of time, typically 3-5 years. It should include a projected income statement, balance sheet, and cash flow statement.

The Income statement, also known as the Profit and Loss (P&L) statement, should include projected revenue, cost of goods sold, operating expenses and projected net profit or loss.

The Balance Sheet, should include projected assets, liabilities and equity at the end of each financial year.

The Cash Flow statement, should include projected cash inflows and outflows, as well as the projected net cash position at the end of each financial year.

In order to develop accurate financial projections, it is important to have a clear understanding of the project's revenue streams and operating costs, as well as the market

and industry trends. A detailed budget for the marketing and sales activities should also be included.

It is important to note that financial projections are estimates and not guarantees. It is always a good idea to also include a sensitivity analysis, which will show how the financial projections are affected by changes in key assumptions such as revenue growth rates and operating costs.

Overall, the Financial Projections section should provide a clear and concise overview of the project's projected financial performance and should demonstrate the project's financial viability. This information can be used to secure funding and to make informed business decisions.

2.7 Conclusion

The Conclusion section of an Amazon Affiliate project is a summary of the main findings and recommendations from the project. It should provide a clear and concise overview of the project's key takeaways and should include any final thoughts or observations.

In this section, it is important to summarize the main findings from the project, including the target market, the revenue streams, the marketing and sales strategy, and the financial projections. It should also include any recommendations for future work or improvements.

Additionally, the Conclusion section should also discuss the project's overall impact and potential for success, including any benefits or opportunities that the project may provide.

It is important to note that the Conclusion section should be written in a clear and concise manner, avoiding unnecessary detail or technical jargon. It should be easy to understand for both technical and non-technical audiences.

Overall, the Conclusion section should provide a clear and concise summary of the project's main findings and recommendations, and should demonstrate the project's potential for success. This information can be used to inform future work and to make informed business decisions.

CHAPTER 3

NEORONETIX AI API RAW CODE

3.1 Admin User API

Note: The purpose of an Admin User API in Node.js is to provide a set of endpoints for administrator users to manage and maintain user accounts on a web application or system. This can include tasks such as creating new user accounts, updating user information, and deleting user accounts. These endpoints are typically protected by authentication and authorization mechanisms to ensure that only authorized administrator users have access to them.

Create User: An endpoint to create a new user account. This can include fields such as name, email, password, and any other relevant user information.

Read User: An endpoint to retrieve information about a specific user. This can include fields such as name, email, and any other relevant user information.

Update User: An endpoint to update the information for a specific user. This can include fields such as name, email, password, and any other relevant user information.

Delete User: An endpoint to delete a specific user account.

List Users: An endpoint to retrieve a list of all users in the system, along with relevant user information.

Search Users: An endpoint to search for users based on certain criteria, such as name or email.

Additionally to these endpoints, the API can expose functionalities such as pagination, sorting, filtering, and others that can help the administrator to have a better control over the users.

All these endpoints are typically protected by authentication and authorization mechanisms, such as JSON Web Tokens (JWT) or OAuth, to ensure that only authorized administrator users have access to them.

```
const adminDetailFactory = require("../operation/admin/adminDetail");
const adminListFactory = require("../operation/admin/adminList");
const createAdminFactory = require("../operation/admin/createAdmin");
const removeAdminFactory = require("../operation/admin/deleteAdmin");
const updateAdminFactory = require("../operation/admin/updateAdmin");

function adminUserApiFactory(options = {}) {
```

```
const { formParser, queryParser, parameterParser, userInfoParser } = options;
```

```
const createAdmin = createAdminFactory();
```

```
const adminList = adminListFactory();
```

```
const removeAdmin = removeAdminFactory();
```

```
const updateAdmin = updateAdminFactory()
```

```
const adminDetail = adminDetailFactory()
```

```
return {
```

```
  /**
```

```
   * POST /
```

```
   * @param {object} req
```

```
   */
```

```
  create: async function (req) {
```

```
    const { fields } = formParser(req);
```

```
    const writeResult = await createAdmin(fields);
```

```
    return writeResult
```

```
  },
```

```
  /**
```

```
   * GET /
```

```
   * @param {object} req
```

```
   */
```

```
  list: async function (req) {
```

```
    const query = queryParser(req)
```

```
    const filter = {
```

```
      search: query.search && query.search.trim()
```

```
    };
```

```
    const resolve = { }
    const listResult = await adminList(filter, resolve, query)
    return listResult;
  },

  /**
   * GET /:id
   * @param {object} req
   */
  detail: async function (req) {
    const { id } = parameterParser(req)
    const query = queryParser(req);
    const resolve = { }
    const detail = await adminDetail(id, resolve);
    return detail;
  },

  /**
   * DELETE /:id
   * @param {object} req
   */
  remove: async function (req) {
    let { id } = parameterParser(req)
    id = id.split(",").map(i => i.trim()).filter(i => i);
    const deleteResult = await removeAdmin(id);
    return deleteResult;
  },

  /**
   * PUT /
```

```

    * @param {object} req
    *
    */
    update: async function (req) {
      const { fields } = formParser(req)
      const results = await updateAdmin(fields);
      return results;
    }
  }
}

module.exports = adminUserApiFactory;

```

3.2 Admin API

Note: An Admin API, also known as an Administrator API, is a set of programming interfaces that allow developers to access and control certain functionalities within a web application or system. These functionalities are typically restricted to administrator users and are not available to regular users.

The Admin API is designed to provide a set of endpoints that can be consumed by a web or mobile application, allowing administrator users to manage and maintain different aspects of the application, such as user accounts, settings, and data. These endpoints are typically protected by authentication and authorization mechanisms to ensure that only authorized administrator users have access to them.

User management: Creating, reading, updating, and deleting user accounts.

Data management: Creating, reading, updating, and deleting data within the application.

System management: Configuring settings, monitoring system performance, and managing resources such as servers and databases.

Auditing: Keeping track of user activity and changes to the system.

Reports: Generating reports on user activity, system performance, and other metrics.

The Admin API is typically built using a web framework such as Node.js and can be accessed via HTTP requests. The API can return data in various formats, such as JSON,

XML, and HTML. Additionally, the API can be designed to be modular and extensible, allowing developers to add new functionalities over time.

It's important to note that, Admin API can be integrated with various other tools such as:

Authentication and Authorization: To ensure that only authorized administrator users have access to the API.

Versioning: To handle breaking changes and multiple versions of the API.

Logging and Monitoring: To track and debug issues that may arise during the usage of the API.

Overall, an Admin API provides a powerful tool for administrators to manage and maintain web applications and systems, allowing them to perform a wide range of tasks with ease and efficiency.

```
const ArticleOperations = require("../operation/article");

function articleApiFactory(options = {}) {

  const { formParser, queryParser, parameterParser } = options;

  const sectionOperations = new ArticleOperations();

  return {

    /**
     * POST /
     * @param {object} req
     */
    create: async function (req) {
      const { fields } = formParser(req);
      const writeResult = await sectionOperations.create(fields);
      return writeResult
    },

    /**
```

```

* GET /
* @param {object} req
*/
list: async function (req) {
  const query = queryParser(req)
  const filter = {
    search: query.search && query.search.trim(),
    id: query.id && query.id.split(",").map(i => i.trim()).filter(i => i)
  };
  const resolve = {
    cover: query.resolveCover == 1, //default false
    site: query.resolveSite == 1, //default false
    template: query.resolveTemplate == 1, //default false
    tokenUsage: query.tokenUsage == 1
  }
  const listResult = await sectionOperations.list(filter, resolve, query)
  return listResult;
},
}

}

module.exports = articleApiFactory;

```

3.3 Section API

Note: A Section API, also known as a Segmentation API, is a set of programming interfaces that allow developers to access and control certain functionalities within a web application or system related to segmentation. These functionalities are typically restricted to certain users who have the right access and are not available to all users.

The Section API is designed to provide a set of endpoints that can be consumed by a web or mobile application, allowing users to segment the data within the application, such as

different categories, groups, or subsets of data. These endpoints can be protected by authentication and authorization mechanisms to ensure that only authorized users have access to them.

The main purpose of the Section API is to allow users to separate and organize their data in a way that makes sense to them. It can be used for various purposes such as:

- Grouping similar data together
- Creating different views or perspectives of the data
- Filtering data based on certain criteria
- Creating subsets of data for specific tasks or analyses

The Section API typically returns data in various formats such as JSON or XML, and can be accessed via HTTP requests. Additionally, the API can be designed to be modular and extensible, allowing developers to add new functionalities over time.

In summary, a Section API is an important tool for organizing and managing data within a web application or system, allowing users to segment and analyze their data in a meaningful way.

```
const SectionOperations = require("../operation/section");
```

```
function sectionApiFactory(options = {}) {

  const { formParser, queryParser, parameterParser } = options;

  const sectionOperations = new SectionOperations();

  return {

    /**
     * POST /
     * @param {object} req
     */
    create: async function (req) {
      const { fields } = formParser(req);
      const writeResult = await sectionOperations.create(fields);
      return writeResult
    }
  }
}
```

```

},

/**
 * GET /
 * @param {object} req
 */
list: async function (req) {
  const query = queryParser(req)
  const filter = {
    search: query.search && query.search.trim(),
    id: query.id && query.id.split(",").map(i => i.trim()).filter(i => i)
  };
  const resolve = {
    cover: query.resolveCover == 1, //default false
  }
  const listResult = await sectionOperations.list(filter, resolve, query)
  return listResult;
},

/**
 * DELETE /:id
 * @param {object} req
 */
remove: async function (req) {
  let { id } = parameterParser(req)
  id = id.split(",").map(i => i.trim()).filter(i => i);
  const deleteResult = await sectionOperations.remove(id);
  return deleteResult;
},

```

```

/**
 * PUT /
 * @param {object} req
 *
 */
update: async function (req) {
  const { fields } = formParser(req)
  const results = await sectionOperations.update(fields);
  return results;
}
}

}

module.exports = sectionApiFactory;

```

3.4 Tag API

Note: A Tag API, also known as a Tagging API, is a set of programming interfaces that allow developers to access and control certain functionalities within a web application or system related to tagging.

Tagging is a method of organizing and categorizing data by adding labels or tags to it. A Tag API allows developers to create, read, update, and delete tags, as well as to associate tags with specific data.

The Tag API can provide a set of endpoints that can be consumed by a web or mobile application, allowing users to:

- Create new tags
- Retrieve information about existing tags
- Update or delete existing tags
- Associate tags with specific data
- Search for data based on specific tags

The Tag API can be protected by authentication and authorization mechanisms to ensure that only authorized users have access to it. The API typically returns data in various

formats such as JSON or XML, and can be accessed via HTTP requests. Additionally, the API can be designed to be modular and extensible, allowing developers to add new functionalities over time.

In summary, a Tag API is a useful tool for organizing and managing data within a web application or system by allowing users to add labels or tags to their data, making it more easily searchable and categorizable.

```
const TagOperations = require('../operation/tag')

// const createTagFactory = require("../operation/tag/createTag");
// const removeTagFactory = require("../operation/tag/deleteTag");
// const tagDetailFactory = require("../operation/tag/tagDetail");
// const tagListFactory = require("../operation/tag/tagList");
// const updateTagFactory = require("../operation/tag/updateTag");

/**
 * locale creation api factory
 * @param {object} options
 * @param {Function} options.formParser
 * @param {Function} options.queryParser
 * @param {Function} options.parameterParser
 */
function tagApiFactory(options = {}) {

  const { formParser, queryParser, parameterParser } = options;

  // const createTag = createTagFactory({});
  // const tagList = tagListFactory({});
  // const tagDetail = tagDetailFactory({});
  // const removeTag = removeTagFactory({});
  // const updateTag = updateTagFactory({})

  const tagOperations = new TagOperations()
```

```
return {  
  
  /**  
   * POST /  
   * @param {object} req  
   */  
  create: async function (req) {  
    const { fields } = formParser(req);  
    const writeResult = await tagOperations.create(fields);  
    return writeResult  
  },  
  
  /**  
   * GET /  
   * @param {object} req  
   */  
  list: async function (req) {  
    const query = queryParser(req)  
    const filter = {  
      search: query.search && query.search.trim()  
    };  
    const resolve = {  
      cover: query.resolveCover == 1, //default false  
      image: query.resolveImage == 1, //default false,  
      productCount: query.productCount == 1  
    }  
  
    const listResult = await tagOperations.list(filter, resolve, query)  
    return listResult;  
  }  
}
```

```

},

/**
 * GET /:id
 * @param {object} req
 */
detail: async function (req) {
  let { id } = parameterParser(req)
  id = id.indexOf(",") > -1 ? id.split(",").map(i => i.trim()) : id;

  const query = queryParser(req); const resolve = {
    cover: query.resolveCover == 1, //default false
    image: query.resolveImage == 1, //default false,
    productCount: query.productCount == 1
  }

  const detail = await tagOperations.detail(id, resolve, '/tag/');
  return detail;
},

/**
 * DELETE /:id
 * @param {object} req
 */
remove: async function (req) {
  let { id } = parameterParser(req)
  id = id.split(",").map(i => i.trim()).filter(i => i);
  const deleteResult = await tagOperations.remove(id);
  return deleteResult;
},

```



```

/**
 * PUT /
 * @param {object} req
 *
 */
update: async function (req) {
  const { fields } = formParser(req)
  const results = await tagOperations.update(fields);
  return results;
}
}
}

module.exports = tagApiFactory;

```

3.5 Template API

Note: A Template API, also known as a Templates Management API, is a set of programming interfaces that allow developers to access and control certain functionalities within a web application or system related to templates. Templates are pre-designed layouts, formats or structured documents that can be reused multiple times.

A Template API allows developers to create, read, update, and delete templates, as well as to associate templates with specific data. It can provide a set of endpoints that can be consumed by a web or mobile application, allowing users to:

- Create new templates with specific format, layout, or structure
- Retrieve information about existing templates
- Update or delete existing templates
- Associate templates with specific data
- Search for templates based on specific criteria

The API can be protected by authentication and authorization mechanisms to ensure that only authorized users have access to it. The API typically returns data in various formats

such as JSON or XML, and can be accessed via HTTP requests. Additionally, the API can be designed to be modular and extensible, allowing developers to add new functionalities over time.

In summary, a Template API is a useful tool for managing templates within a web application or system, it allows users to create, use, and manage reusable templates, making the process of creating similar documents or layouts more efficient and consistent.

```
const SectionOperations = require("../operation/section");
const TemplateOperations = require("../operation/template");

function templateApiFactory(options = {}) {

  const { formParser, queryParser, parameterParser } = options;

  const templateOperations = new TemplateOperations();
  return {

    /**
     * POST /
     * @param {object} req
     */
    create: async function (req) {
      const { fields } = formParser(req);
      const writeResult = await templateOperations.create(fields);
      return writeResult
    },

    /**
     * GET /
     * @param {object} req
     */
  }
}
```

```

list: async function (req) {
  const query = queryParser(req)
  const filter = {
    search: query.search && query.search.trim(),
    id: query.id && query.id.split(",").map(i => i.trim()).filter(i => i)
  };
  const resolve = {
    cover: query.resolveCover == 1, //default false
  }
  const listResult = await templateOperations.list(filter, resolve, query)
  return listResult;
},

/**
 * DELETE /:id
 * @param {object} req
 */
remove: async function (req) {
  let { id } = parameterParser(req)
  id = id.split(",").map(i => i.trim()).filter(i => i);
  const deleteResult = await templateOperations.remove(id);
  return deleteResult;
},

/**
 * PUT /
 * @param {object} req
 *
 */
update: async function (req) {

```

```

    const { fields } = formParser(req)

    const results = await templateOperations.update(fields);

    return results;
  }
}

}

module.exports = templateApiFactory;

```

3.6 Website API

Note: A Website API, also known as a Web API, is a set of programming interfaces that allow developers to access and control certain functionalities within a website. A Website API allows developers to create, read, update, and delete data, as well as perform other actions, on a website using a programming language.

A Website API typically provides a set of endpoints that can be accessed via HTTP requests, such as GET, POST, PUT and DELETE, allowing developers to:

- Retrieve data from a website
- Update data on a website
- Delete data from a website
- Perform other actions on a website

The API can be protected by authentication and authorization mechanisms to ensure that only authorized users have access to it. The API typically returns data in various formats such as JSON or XML. Additionally, the API can be designed to be modular and extensible, allowing developers to add new functionalities over time.

Website APIs are becoming more common as websites are increasingly being used as platforms for delivering services, such as e-commerce, social networking, and more. They enable developers to easily access and manipulate website data, as well as to build new applications and services on top of a website.

In summary, A Website API is a useful tool for developers to access and control the functionalities of a website, it allows them to interact with the website's data and functionality by using a programming language rather than a web browser, which enables them to automate, integrate and extend the functionality of the website.

```

const SectionOperations = require("../operation/section");

const WebsiteOperations = require("../operation/website");

```

```

function websiteApiFactory(options = {}) {

  const { formParser, queryParser, parameterParser } = options;

  const websiteOperations = new WebsiteOperations();
  return {

    /**
     * POST /
     * @param {object} req
     */
    create: async function (req) {
      const { fields } = formParser(req);
      const writeResult = await websiteOperations.create(fields);
      return writeResult
    },

    /**
     * GET /
     * @param {object} req
     */
    list: async function (req) {
      const query = queryParser(req)
      const filter = {
        search: query.search && query.search.trim(),
        id: query.id && query.id.split(",").map(i => i.trim()).filter(i => i)
      };
      const resolve = {
        cover: query.resolveCover == 1, //default false

```

```
    }  
    const listResult = await websiteOperations.list(filter, resolve, query)  
    return listResult;  
  },  
  
  /**  
   * DELETE /:id  
   * @param {object} req  
   */  
  remove: async function (req) {  
    let { id } = parameterParser(req)  
    id = id.split(",").map(i => i.trim()).filter(i => i);  
    const deleteResult = await websiteOperations.remove(id);  
    return deleteResult;  
  },  
  
  /**  
   * PUT /  
   * @param {object} req  
   *  
   */  
  update: async function (req) {  
    const { fields } = formParser(req)  
    const results = await websiteOperations.update(fields);  
    return results;  
  }  
}  
  
}
```

```
module.exports = websiteApiFactory;
```

CHAPTER 4

WRITING SECTION

4.1 Create AI Section

Note: Start by defining the topic of the article and determining the focus or angle of the piece. This can be done by researching the topic and identifying key areas of interest or by providing a specific prompt or question to the OpenAI model.

Gather data: Use OpenAI's natural language processing and machine learning techniques to gather data on the topic from various sources such as news articles, scientific papers, and social media. This data can be used to generate ideas, facts, and statistics for the article.

Generate a thesis statement: Use OpenAI's natural language generation to generate a thesis statement for the article. This statement should clearly outline the main point or argument of the article.

Create an outline: Use OpenAI's natural language generation to create an outline for the article, including the main sections and subtopics that will be covered.

Write the article: Use OpenAI's natural language generation to write the article based on the data gathered, the thesis statement, and the outline. This can include things like text, images, and other media.

```
const { Configuration, OpenAIApi } = require("openai");  
const config = require("../config/config.json");  
const TokenUsageDB = require("../database/mongodb/tokenUsage");  
const { getOpenAIKey } = require("./keyManager");
```

```
async function createAISection(article, section) {
```

```
const configuration = new Configuration({
  apiKey: await getOpenAIKey(),
});

const openAI = new OpenAIApi(configuration);

const tokenUsageDB = new TokenUsageDB()
const options = {
  model: section.model || "text-davinci-002",
  prompt: section.promptText,
  temperature: section.temperature || 0.5,
  max_tokens: section.maxToken || 2000
}
const completion = await openAI.createCompletion(options);

if (!completion) throw new Error("failed to get AI completion")

const usedTokens = completion.data.usage.total_tokens;
tokenUsageDB.writeOne({ article: article._id, section: section.title, site: article.site,
token: usedTokens, type: "aiToken", createdAt: new Date() });

return completion.data.choices[0].text;

}

module.exports = createAISection;
```

4.1 Create Loop Section

Note: The Loop Section is a feature that allows for the creation of multiple articles at a time using OpenAI. It works by defining a topic and gathering data on that topic, then generating a thesis statement and outline for the article.

Next, OpenAI's natural language generation is used to write the article based on the data, thesis statement, and outline. The process is then repeated for additional articles on the same topic, allowing for the efficient creation of multiple pieces of content. This feature can save time and resources by automating the process of article writing.

The Loop Section also includes a proofreading and editing step, where OpenAI's natural language processing and machine learning techniques are used to ensure that the articles are free of errors and are easy to read. Additionally, the Loop Section can also include a formatting step, where OpenAI's natural language generation is used to format the articles according to the publication's guidelines.

And also it includes step for citing the sources used in the article, OpenAI's natural language generation is used to include any necessary citation for any source used in the article. After all the steps, the articles can be submitted to the appropriate publication or platform and can be promoted using OpenAI's natural language generation to write a promotional text and share it on social media or other relevant platforms to reach a wider audience.

```
const { Configuration, OpenAIApi } = require("openai");
const config = require("../config/config.json")
const ejs = require("ejs");
const TokenUsageDB = require("../database/mongodb/tokenUsage");
const { getOpenAIKey } = require("./keyManager");

const tokenUsageDB = new TokenUsageDB()
async function createLoopSection(article, section) {

  const options = {
    model: section.model || "text-davinci-002",
    prompt: section.subjectPrompt && ejs.render(section.subjectPrompt, article, {
      delimiter: "?" }
    ),
    temperature: section.temperature || 0.5,
    max_tokens: section.maxToken || 2000
  }
}
```

```

const configuration = new Configuration({
  apiKey: await getOpenAIKey(),
});
const openAI = new OpenAIApi(configuration);
const completion = await openAI.createCompletion(options);

if (!completion) throw new Error("failed to get Subject AI completion")

const subjectList = completion?.data?.choices?.length &&
completion.data.choices[0]?.text?.split("\n").filter(i => i);

const usedTokens = completion.data.usage.total_tokens;

tokenUsageDB.writeOne({ article: article._id, section: section.title, site: article.site,
token: usedTokens, type: "aiToken", createdAt: new Date() });

const resultList = {};

// const promises = [];
for (let subject of subjectList) {
  resultList[subject] = "";
  // promises.push(async () => {
  const options = {
    model: section.model || "text-davinci-002",
    prompt: section.resultPrompt && ejs.render(section.resultPrompt, { ...article,
subject }, { delimiter: "?" }),
    temperature: section.temperature || 0.5,
    max_tokens: section.maxToken || 2000
  }

  const configuration = new Configuration({

```

```

    apiKey: await getOpenAIKey(),
  });
  const openAI = new OpenAIApi(configuration);
  const completion = await openAI.createCompletion(options);

  if (!completion) throw new Error("failed to get Subject AI completion")

  resultList[subject] = completion?.data?.choices[0].text;

  const usedTokens = completion.data.usage.total_tokens;
  tokenUsageDB.writeOne({ article: article._id, section: section.title, site: article.site,
  token: usedTokens, type: "aiToken", createdAt: new Date() });

  // })()
}

// await Promise.all(promises)
return resultList;

}

module.exports = createLoopSection;

```

4.3 Format Content

Note: The format of the content is important for readability and engagement. In the Loop Section, OpenAI's natural language generation can be used to format the articles in a way that is easy to read and visually appealing. This can include things like headings, subheadings, bullet points, and images.

Additionally, OpenAI can be used to ensure that the articles are formatted according to the guidelines of the publication or platform they will be submitted to. This can include things

like word count, font and font size, and margins. By formatting the content in an attractive and easy-to-read way, it can increase engagement and readability, which can lead to more readers and higher engagement.

Additionally, OpenAI can also be used to optimize the content for SEO (Search Engine Optimization). This can include things like incorporating relevant keywords throughout the article, optimizing the meta-description, and adding alt tags to images, this can help the content to rank higher in search engine results, which can drive more traffic to the article.

Another way to format the content is to make it more interactive, OpenAI can generate interactive elements such as polls, quizzes and infographics to make the content more engaging, interactive and shareable. And also OpenAI can be used to generate audio and video content for the same topic, that can also be used to reach a wider audience.

The format of the content is also important for accessibility, OpenAI can be used to ensure that the content is accessible to people with disabilities by generating alternative text for images and videos, and making sure that the text is in a high-contrast font and large enough to be read easily.

Overall, using OpenAI to format the content can help to increase engagement, readability, accessibility, and discoverability, which can lead to more readers, higher engagement and better results.

```
const { Configuration, OpenAIApi } = require("openai");
const config = require("../config/config.json")
const ejs = require("ejs");
const TokenUsageDB = require("../database/mongodb/tokenUsage");
const { getOpenAIKey } = require("../keyManager");

const tokenUsageDB = new TokenUsageDB()
async function createLoopSection(article, section) {

  const options = {
    model: section.model || "text-davinci-002",
    prompt: section.subjectPrompt && ejs.render(section.subjectPrompt, article, {
      delimiter: "?" } ),
    temperature: section.temperature || 0.5,
    max_tokens: section.maxToken || 2000
```

```

}

const configuration = new Configuration({
  apiKey: await getOpenAIKey(),
});

const openAI = new OpenAIApi(configuration);
const completion = await openAI.createCompletion(options);

if (!completion) throw new Error("failed to get Subject AI completion")

const subjectList = completion?.data?.choices?.length &&
completion.data.choices[0]?.text?.split("\n").filter(i => i);

const usedTokens = completion.data.usage.total_tokens;

tokenUsageDB.writeOne({ article: article._id, section: section.title, site: article.site,
token: usedTokens, type: "aiToken", createdAt: new Date() });

const resultList = {};

// const promises = [];
for (let subject of subjectList) {
  resultList[subject] = "";
  // promises.push(async () => {
  const options = {
    model: section.model || "text-davinci-002",
    prompt: section.resultPrompt && ejs.render(section.resultPrompt, { ...article,
subject }, { delimiter: "?" }),
    temperature: section.temperature || 0.5,
    max_tokens: section.maxToken || 2000
  }
}

```

```

const configuration = new Configuration({
  apiKey: await getOpenAIKey(),
});
const openAI = new OpenAIApi(configuration);
const completion = await openAI.createCompletion(options);

if (!completion) throw new Error("failed to get Subject AI completion")

resultList[subject] = completion?.data?.choices[0].text;

const usedTokens = completion.data.usage.total_tokens;
tokenUsageDB.writeOne({ article: article._id, section: section.title, site: article.site,
token: usedTokens, type: "aiToken", createdAt: new Date() });

// })()
}

// await Promise.all(promises)
return resultList;

}

module.exports = createLoopSection;

```

4.4 Generate Article

Note: Generating an article using OpenAI's API involves making a request to the API and providing it with the necessary inputs. The inputs typically include the topic or keywords for the article, the desired length, and any other parameters such as the tone or style of the article.

To make a request to the API, a developer would need to use their API key and send a POST request to the appropriate endpoint, along with the inputs in the request body. The API will then use its natural language generation and machine learning algorithms to generate the article.

The API can generate articles in various formats such as HTML, plain text, and JSON. The response from the API includes the generated article and other information such as the title, summary, and keywords.

The generated article can be used as-is or can be further edited and proofread by a human before publishing. The quality of the generated article will depend on the quality of the input and the parameters provided to the API.

It is worth noting that OpenAI's API is a paid service with different plans and pricing options. Each plan provides different features, usage limits, and access to different models. It's also worth noting that the OpenAI's API is constantly evolving and gaining new capabilities, so it's a good idea to stay updated with the latest developments.

Overall, OpenAI's API can be a great tool for automating the process of article writing and to generate large amounts of content efficiently, it can be integrated into a wide range of applications and services, such as content generation for websites, news articles, and more.

```
const TemplateDB = require("../database/mongodb/template");
const createAISection = require("./createAISection");
const createLoopSection = require("./createLoopSection");
const ejs = require("ejs");
const getYoutubeVideo = require("./getYoutubeVideo");

const fs = require("fs");
const formatContent = require("./formatContent");
const publishPostWP = require("./publishPostWP");

const utilityFunctions = require("./utilityFunctions");
const getGoogleImage = require("./getGoogleImage");

const templateDB = new TemplateDB()
```

```
async function generateArticle(article, siteConfig) {
  const content = [];

  if (!article.template) throw new Error("Template id absent");

  //bind helper functions to article, so that they are available to ejs parser
  for (let funcName in utilityFunctions) {
    const func = utilityFunctions[funcName]
    utilityFunctions[funcName] = (...args) => func(...args, article)
  }
  article = {
    ...article,
    ...utilityFunctions
  }

  const template = await templateDB.readOne(article.template);
  if (!template) throw new Error("Template not found " + article.template.toHexString());

  const promises = [];

  for (let index in template.section) {

    const section = template.section[index];

    // promises.push((async () => {
    try {
      if (section.type === 'autoGenerate') {
```



```
    if (section.heading) section.heading = ejs.render(section.heading, article, {
delimiter: "?" })
```

```
    section.promptText = ejs.render(section.promptText, article, { delimiter: "?" })
```

```
    const generatedSection = await createAISection(article, section);
```

```
    content.push({
      paragraph: generatedSection,
      config: section,
      index
    })
```

```
  }
```

```
  if (section.type === 'autoLoop') {
```

```
    const generatedSection = await createLoopSection(article, section);
```

```
    content.push({
      loopList: generatedSection,
      config: section,
      index
    })
```

```
  }
```

```
  } catch (err) {
```

```
    console.log(err)
```

```
    throw err
```

```
  }
```

```
  try {
```

```
if (section.type === 'image') {
  section.keyword = ejs.render(section.keyword, article, { delimiter: "?" })
  if (section.captionTag && section.caption) {
    section.caption = ejs.render(section.caption, article, { delimiter: "?" })
  }
  section.keyword = ejs.render(section.keyword, article, { delimiter: "?" })
  const image = await getImage(article, section);
  content.push({
    image,
    config: section,
    index
  })
}

if (section.type === 'video') {
  section.keyword = ejs.render(section.keyword, article, { delimiter: "?" })
  const video = await getYoutubeVideo(article, section);
  content.push({
    video,
    config: section,
    index
  })
}
} catch (err) {
  console.log(err)
}

try {

  if (section.type === "text") {
```

```
    if (section.text && section.textTag) {
      section.text = ejs.render(section.text, article, { delimiter: "?" })
      section.textTag = ejs.render(section.textTag, article, { delimiter: "?" })
    }

    if (section.heading && section.headingTag) {
      section.heading = ejs.render(section.heading, article, { delimiter: "?" })
      section.headingTag = ejs.render(section.headingTag, article, { delimiter: "?" })
    }
  }
  content.push({
    config: section,
    index
  })
}
} catch (err) {
  console.log(err)
  throw err;
}

// })()

}

// await Promise.all(promises);
```

```

//sort sections according to index
const sortedContent = content.sort((a, b) => a.index - b.index)
// console.log(sortedContent.map(c => c.index + "" + c.config.title))

const formattedContent = await formatContent(sortedContent);

const postTitle = ejs.render(template.postTitle || "", article, { delimiter: "?" });

const uploadResponse = await publishPostWP(siteConfig, { title: postTitle, content:
formattedContent })

}

module.exports = generateArticle

```

4.5 Get Google Image

Note: Google Images API allows developers to access and search for image content on the internet. The API can be used to search for images based on keywords, image size, and other parameters.

To use the API, developers must first create a project on the Google Cloud Platform and enable the Google Images API. Next, they will need to generate an API key, which is used to authenticate requests to the API.

Once the API key is generated, developers can use it to make requests to the API. The API supports several endpoints, including search, which allows developers to search for images based on keywords, and details, which allows developers to retrieve detailed information about a specific image.

To search for images, developers can make a GET request to the search endpoint and include their API key, the search keywords, and any other parameters they want to specify. For example, the following request searches for images of "kittens" and limits the results to images with a width of 800 pixels or less:

https://www.googleapis.com/customsearch/v1?q=kittens&imgSize=small&key=API_KEY

The API will then return a JSON object containing the search results, including the image URLs, titles, and other relevant information.

Additionally, the API also supports image recognition, which allows developers to perform image-based searches using image data. For example, developers can use the API to identify the subject of an image or to find similar images.

It's worth noting that Google has a usage limit on the number of requests that can be made to the API per day, and it also requires to credit the source of the images.

Overall, the Google Images API is a powerful tool for developers looking to access and search for image content on the internet, it allows them to integrate image search capabilities into their own applications and services and enhance the user experience.

```
const config = require("../config/config.json")
const fetch = require("node-fetch");
const { getImageKey } = require("./keyManager");
const engineConfig = require("../config/keyToSearchEngineMap.json");
const TokenUsageDB = require("../database/mongodb/tokenUsage");

const tokenUsageDB = new TokenUsageDB()

async function getGoogleImage(article, section) {

  let url = "https://www.googleapis.com/customsearch/v1";
  const apiKey = await getImageKey();
  const engineKey = engineConfig[apiKey]
  if (!apiKey) return;

  const options = {
    q: section.keyword,
    key: apiKey,
    cx: engineKey,
  }
}
```

```

url += "?";
for (let key in options) {
  url += `${key}=${options[key]}&`
}

try {
  const response = await fetch(url);
  const result = await response.json();
  if (!result || !result.items) throw new Error("Google Image: no result");

  tokenUsageDB.writeOne({ article: article._id, section: section.title, site: article.site,
token: 1, type: "imageToken", createdAt: new Date() });

  return {
    url:          !!result.items[0]?.pagemap?.cse_image?.length          &&
result.items[0]?.pagemap?.cse_image[0]?.src,
    credit: result.items[0]?.displayLink
  }
} catch (err) {
  console.log(err);
  throw new Error("Google Image: unknown error")
}
}

module.exports = getGoogleImage;

/** sample response item
// {

```

```

// "kind": "customsearch#result",
// "title": "Apple",
// "htmlTitle": "<b>Apple</b>",
// "link": "https://www.apple.com/",
// "displayLink": "www.apple.com",
// "snippet": "Discover the innovative world of Apple and shop everything iPhone, iPad,
Apple Watch, Mac, and Apple TV, plus explore accessories, entertainment, ...",
// "htmlSnippet": "Discover the innovative world of <b>Apple</b> and shop everything
iPhone, iPad, <b>Apple</b> Watch, Mac, and <b>Apple</b> TV, plus explore
accessories, entertainment,&nbsp;...",
// "cacheId": "xEELJvdODswJ",
// "formattedUrl": "https://www.apple.com/",
// "htmlFormattedUrl": "https://www.<b>apple</b>.com/",
// "pagemap": {
//   "cse_thumbnail": [
//     {
//       "src": "https://encrypted-
tbn0.gstatic.com/images?q=tbn:ANd9GcRU17BvguBEtrzqLvgQkLdknkk8vZsRlqab1b8
SC5XiOzqGTdFtArLLMw",
//       "width": "150",
//       "height": "79"
//     }
//   ],
//   "metatags": [
//     {
//       "analytics-s-bucket-1": "applestoreww",
//       "analytics-s-bucket-0": "applestoreww",
//       "og:image": "https://www.apple.com/ac/structured-
data/images/open_graph_logo.png?202110180743",
//       "og:type": "website",
//       "og:site_name": "Apple",
//       "og:title": "Apple",

```

```

//      "ac-gn-store-key": "SFX9YPYY9PPXCU9KH",
//      "og:description": "Discover the innovative world of Apple and shop everything
//      iPhone, iPad, Apple Watch, Mac, and Apple TV, plus explore accessories, entertainment,
//      and expert device support.",
//      "analytics-s-channel": "homepage",
//      "viewport": "width=device-width, initial-scale=1, viewport-fit=cover",
//      "og:locale": "en_US",
//      "og:url": "https://www.apple.com/",
//      "analytics-track": "Apple - Index/Tab",
//      "analytics-s-bucket-2": "applestoreww"
//    }
//  ],
//  "cse_image": [
//    {
//      "src": "https://www.apple.com/ac/structured-
//      data/images/open_graph_logo.png?202110180743"
//    }
//  ]
// }
// },

```

4.6 Get Youtube Videos

Note: Getting YouTube videos using an API involves making a request to the YouTube Data API and providing it with the necessary parameters. The YouTube Data API allows developers to retrieve information about YouTube videos, channels, playlists, and more.

To use the YouTube Data API, developers must first create a project on the Google Cloud Platform and enable the YouTube Data API. Next, they will need to generate an API key, which is used to authenticate requests to the API.

Once the API key is generated, developers can use it to make requests to the API. The API supports several endpoints, including search, which allows developers to search for videos based on keywords, and videos, which allows developers to retrieve information about a specific video.

To search for videos, developers can make a GET request to the search endpoint and include their API key, the search keywords, and any other parameters they want to specify. For example, the following request searches for videos related to "coding tutorials" and limits the results to videos in the English language:

https://www.googleapis.com/youtube/v3/search?part=snippet&q=coding+tutorials&type=video&key=API_KEY&hl=en

To retrieve information about a specific video, developers can make a GET request to the videos endpoint and include the video's ID and their API key. For example, the following request retrieves information about the video with the ID "abcdefghijkl":

https://www.googleapis.com/youtube/v3/videos?part=snippet&id=abcdefghijkl&key=API_KEY

The API will then return a JSON object containing the requested information, such as the video's title, description, thumbnail, and more.

It's worth noting that Google has a usage limit on the number of requests that can be made to the API per day, and it also requires to credit the source of the videos.

Overall, the YouTube Data API is a powerful tool for developers looking to access and retrieve information about YouTube videos, it allows them to integrate video search and retrieval capabilities into their own applications and services and enhance the user experience.

```
const config = require("../config/config.json")
const fetch = require("node-fetch");
const { getVideoKey } = require("./keyManager");
const TokenUsageDB = require("../database/mongodb/tokenUsage");

const tokenUsageDB = new TokenUsageDB()
async function getYoutubeVideo(article, section) {

  let url = "https://www.googleapis.com/youtube/v3/search";
  const apiKey = await getVideoKey();
  if (!apiKey) return;

  const options = {
    type: "video",
```

```

    q: section.keyword,
    part: "snippet",
    key: apiKey
  }

  url += "?";
  for (let key in options) {
    url += `${key}=${options[key]}&`
  }

  try {
    const response = await fetch(url);
    const result = await response.json();
    if (!result || !result.items) throw new Error("Youtube Video: no result");

    tokenUsageDB.writeOne({ article: article._id, section: section.title, site: article.site,
    token: 1, type: "youtubeToken", createdAt: new Date() });

    //not all youtube videos allows embedding; checking for valid embed enabled video
    here
    let indexWithValidVideo;
    for (let index in result.items) {
      try {
        const embedResponse = await
        fetch(`https://www.youtube.com/embed/${result.items[index]?.id.videoId}`);
        const textResponse = await embedResponse.text();
        if (textResponse.indexOf("Video unavailable") === -1) indexWithValidVideo =
        index;
      } catch (err) {
        console.log(err);
        continue;
      }
    }
  }

```

```

    }
}

if (!indexWithValidVideo) throw new Error("Youtube Video: no embed")

return result.items[indexWithValidVideo];
} catch (err) {
    throw new Error("Youtube Video: unknown error")
}
}

module.exports = getYoutubeVideo;

/** sample response
// {
//   "kind": "youtube#searchResult",
//   "etag": "TFLvN15vUpvN9KIwXitPd8AqCa8",
//   "id": {
//     "kind": "youtube#video",
//     "videoId": "dpGxK8nRWKQ"
//   },
//   "snippet": {
//     "publishedAt": "2021-10-09T17:42:12Z",
//     "channelId": "UCNngxIZ6g3gO5W1GmFTXsbZw",
//     "title": "How to install windows 11 step by step in Bangla | Setup Windows 11 |
Install Windows 11 Any Version",
//     "description": "https://realmati.com Windows 10 Pro OEM Key Buying link
https://fb.com/RealMati Facebook Page Link: ...",
//     "thumbnails": {
//       "default": {
//         "url": "https://i.ytimg.com/vi/dpGxK8nRWKQ/default.jpg",

```

```

//      "width": 120,
//      "height": 90
//    },
//    "medium": {
//      "url": "https://i.ytimg.com/vi/dpGxK8nRWKQ/mqdefault.jpg",
//      "width": 320,
//      "height": 180
//    },
//    "high": {
//      "url": "https://i.ytimg.com/vi/dpGxK8nRWKQ/hqdefault.jpg",
//      "width": 480,
//      "height": 360
//    }
//  },
//  "channelTitle": "RealTech Master",
//  "liveBroadcastContent": "none",
//  "publishTime": "2021-10-09T17:42:12Z"
// }
// },

```

4.7 Key Manager

Note: A key manager is a component that is responsible for managing and handling the cryptographic keys used in a system. It is typically used to secure sensitive information such as passwords, credit card numbers, and other personal information.

One way to implement a key manager in code is to create a class that handles the generation, storage, and retrieval of keys. The class could have methods for generating a new key, storing a key, and retrieving a key.

Here is an example of what the class for a key manager might look like in JavaScript:

```

class KeyManager {
  constructor() {
    this.keys = {};
  }
}

```

```
}  
  
generateKey(keyId) {  
  const key = crypto.randomBytes(32).toString('hex');  
  this.keys[keyId] = key;  
  return key;  
}  
  
getKey(keyId) {  
  return this.keys[keyId];  
}  
  
storeKey(keyId, key) {  
  this.keys[keyId] = key;  
}  
}
```

The class uses the `crypto` module that is built-in `node.js` to generate random bytes for a new key. The `generateKey` method takes a `keyId` as an argument and generates a new key using the `crypto.randomBytes()` method. The new key is then stored in the `keys` object using the `keyId` as the key.

The `getKey` method takes a `keyId` as an argument and returns the key associated with that `keyId` from the `keys` object.

The `storeKey` method allows to store a key with a specific `keyId`.

This is just one example of how a key manager could be implemented in code. In practice, it is important to use a secure method for storing keys and to use a secure method for encrypting and decrypting sensitive data. It is also important to use a secure method for generating random keys, such as using a cryptographically secure random number generator. Additionally, a Key Manager should handle the expiration of keys, and the rotation of the keys.

```
const fs = require("fs");  
const config = require("../config/config.json")
```

```

const path = require("path");
const keyMapPath = path.join(process.cwd(), "config", "keyMap.json");

async function getVideoKey() {
  const keyMapFile = await fs.promises.readFile(keyMapPath);
  const keyMap = JSON.parse(keyMapFile);
  for (let key of config.GOOGLE_API_KEY) {
    if (!keyMap[key] || !keyMap[key].youtube || keyMap[key].youtube <
keyMap[key].limit) {
      if (!keyMap[key]) keyMap[key] = {};
      if (!keyMap[key].youtube) keyMap[key].youtube = 1;
      else keyMap[key].youtube = keyMap[key].youtube + 1;

      await fs.promises.writeFile(keyMapPath, JSON.stringify(keyMap, null, 4));

      return key;
    }
  }
}
module.exports.getVideoKey = getVideoKey;

```

```

async function getImageKey() {
  const keyMapFile = await fs.promises.readFile(keyMapPath);
  const keyMap = JSON.parse(keyMapFile);
  for (let key of config.GOOGLE_API_KEY) {
    if (!keyMap[key] || !keyMap[key].image || keyMap[key].image < keyMap[key].limit)
    {
      if (!keyMap[key]) keyMap[key] = {};

```

```

    if (!keyMap[key].image) keyMap[key].image = 1;
    else keyMap[key].image = keyMap[key].image + 1;

    await fs.promises.writeFile(keyMapPath, JSON.stringify(keyMap, null, 4));

    return key;
  }
}
}
module.exports.getImageKey = getImageKey;

async function getOpenAIKey() {
  if (process.env.openAiLocked === "1") {
    await sleep(100 * Math.random());
    return await getOpenAIKey()
  }

  process.env.openAiLocked = "1";

  const keyMapFile = await fs.promises.readFile(keyMapPath);
  const keyMap = JSON.parse(keyMapFile);
  for (let key of config.OPENAI_API_KEY) {
    if (!keyMap[key] || !keyMap[key].openAi || keyMap[key].openAi <
keyMap[key].limit) {
      if (!keyMap[key]) keyMap[key] = {};
      if (!keyMap[key].openAi) keyMap[key].openAi = 1;
      else keyMap[key].openAi = keyMap[key].openAi + 1;

      await fs.promises.writeFile(keyMapPath, JSON.stringify(keyMap, null, 4));

```

```
    process.env.openAiLocked = "0";

    return key;
  } else {
    process.env.openAiLocked = "0";
    await sleep();
    return await getOpenAIKey()
  }
}
}
}
module.exports.getOpenAIKey = getOpenAIKey;

async function resetOpenAiUsage() {
  const keyMapFile = await fs.promises.readFile(keyMapPath);
  const keyMap = JSON.parse(keyMapFile);

  for (let key in keyMap) {
    if ("openAi" in keyMap[key]) {
      keyMap[key].openAi = 0;
    }
  }
}

await fs.promises.writeFile(keyMapPath, JSON.stringify(keyMap, null, 4));
}

setInterval(resetOpenAiUsage, 60000);

async function resetGoogleUsage() {
```



```

const keyMapFile = await fs.promises.readFile(keyMapPath);
const keyMap = JSON.parse(keyMapFile);

for (let key in keyMap) {
  if ("image" in keyMap[key]) {
    keyMap[key].image = 0;
  }
  if ("youtube" in keyMap[key]) {
    keyMap[key].youtube = 0;
  }
}

await fs.promises.writeFile(keyMapPath, JSON.stringify(keyMap, null, 4));
}

setInterval(resetGoogleUsage, 86400000);

function sleep(ms = 60000) {
  return new Promise(resolve => {
    setTimeout(resolve, ms)
  })
}

```

4.8 Process Queue

Note: In Node.js, a queue can be implemented using an array or a linked list data structure. The process queue refers to the way in which items are added and removed from the queue.

One way to implement a process queue in Node.js is to use the "async" and "await" keywords to handle asynchronous operations. The "async" keyword is used to define a function that returns a promise, and the "await" keyword is used to pause the execution of the function until the promise is resolved.

Here is an example of a process queue function in Node.js:

```
const queue = []

async function processQueue() {
  while (queue.length > 0) {
    const item = queue.shift()
    await item()
  }
}

queue.push(() => {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Processing item 1")
      resolve()
    }, 1000)
  })
})

queue.push(() => {
  return new Promise((resolve) => {
    setTimeout(() => {
      console.log("Processing item 2")
      resolve()
    }, 2000)
  })
})

processQueue()
```

In this example, the `processQueue` function is an asynchronous function that continuously checks the queue array for items. If there are items in the queue, it takes the first item and awaits for it to resolve, then moves on to the next item in the queue.

You can add items to the queue by pushing a function that returns a promise to the queue array. In this example, two items are added to the queue that each log a message to the console and resolve after a certain amount of time.

Keep in mind that this is just an example and you can customize the `processQueue` function according to your needs.

Additionally, you can also use existing queue libraries like "bull" or "kue" in node js to handle the queue process.

```
const ArticleDB = require("../database/mongodb/article")
const config = require("../config/config.json");
const con = require("@imtiazhchowdhury/mongopool");
const generateArticle = require("./generateArticle");
const SiteDB = require("../database/mongodb/site");

con.dbName = config.db;
con.url = "mongodb://127.0.0.1:27017";

const articleDB = new ArticleDB();
const siteDb = new SiteDB();

async function processQueue() {
  console.log("Searching ...")

  const nextItem = await articleDB.updateOneByStatus("pending", { status: "processing",
  "message": "Generating article" });

  if (nextItem.value) {
    try {
      const siteConfig = await siteDb.readOne(nextItem.value.site);
      if (!siteConfig) throw new Error("Site not found");
    }
  }
}
```

```

    const article = await generateArticle(nextItem.value, siteConfig);
    await articleDB.updateOne(nextItem.value._id, { status: "Completed", message:
"Article posted" });
  } catch (err) {
    console.log(err)
    await articleDB.updateOne(nextItem.value._id, { status: "error", message:
err.message });
  }

  processQueue()
} else {
  setTimeout(processQueue, 1000);
}
}

async function rescueStuckProcesses() {
  const stuckProcesses = await articleDB.readManyByStatus("processing");
  for (let process of stuckProcesses) {
    await articleDB.updateOne(process._id, { status: "pending" })
  }
}

rescueStuckProcesses().then(() => {
  processQueue()
})

module.exports = processQueue

```

4.9 Publish Post on WP

Note: To publish a post on WordPress, you will first need to have a WordPress website set up and have access to the administration panel (**wp-admin**). Once you have access, you can follow these steps to publish a new post:

- Log in to your WordPress website's administration panel (**wp-admin**)
- Click on the "Posts" option in the left-hand menu and then select "Add New"

In the post editor, add a title and content for your post. You can also add images, videos, and other media to your post by clicking on the "Add Media" button.

In the right sidebar, you can set the post's categories, tags, and other options such as visibility and featured image.

Once you have finished editing your post, you can publish it by clicking on the "Publish" button in the top right corner of the post editor.

Alternatively, you can also schedule a post to be published at a later date or time by clicking on the "Edit" link next to the "Publish" button and selecting a date and time.

You can also use the WordPress API which allows you to interact with WordPress content and functionality. It can be used to create, read, update, and delete posts, pages, and other content. The API can be accessed via RESTful endpoints, which can be accessed using the URL of your website with the `/wp-json/wp/v2/` added to it.

You will need to have the **WP-REST-API** plugin enabled on your WordPress website and also have an API key to access the API. You can then use a programming language like JavaScript or PHP to make requests to the API to create, read, update, and delete posts.

It is important to note that the API does not include all features of the WordPress platform, and it may require additional plugins for full functionality.

```
const WPAPI = require("wpapi");
```

```
const fs = require("fs")
```

```
async function publishPostWP(siteConfig, postContent) {
```

```
  // return fs.promises.writeFile(process.cwd() + "/out/formattedContent.html",
  postContent.content)
```

```
  let wp;
```

```
  try {
```

```
    wp = await WPAPI.discover(siteConfig.url);
```

```
  } catch (err) {
```

```
    throw new Error("wpApi: Error Discovery")
```

```
    }

    let authWp;

    try {
        authWp = await wp.auth({ username: siteConfig.username, password:
siteConfig.password });
    } catch (err) {
        throw new Error("wpApi: Error Auth")
    }

    const payload = {
        title: postContent.title,
        content: postContent.content,
        status: siteConfig.postStatus?.toLowerCase(),
        author: siteConfig.postAuthor,
        categories: siteConfig.postCategory && siteConfig.postCategory.split(","),
        tags: siteConfig.postTag && siteConfig.postTag.split(",")
    }

    let response;

    try {
        response = await authWp.posts().create(payload);
    } catch (err) {
        console.log(err)
        throw new Error("wpApi: " + err.message)
    }

    return response;

}
```

```
module.exports = publishPostWP;
```

4.10 Utility Function

Note: A utility function is a small, standalone piece of code that performs a specific, reusable task. Utility functions are often used to encapsulate complex logic or to perform commonly used operations in a program. They are typically designed to be self-contained and independent of the rest of the codebase, making them easy to test and reuse.

Utility functions can be used for a variety of purposes, such as:

- Performing mathematical or string operations
- Formatting data
- Validating input
- Making API calls
- Logging

Here is an example of a utility function in JavaScript that takes a string as input and returns it in uppercase:

```
function toUpperCase(str) {
  return str.toUpperCase();
}

console.log(toUpperCase("hello world")); // "HELLO WORLD"
```

Another example of a utility function that validates an email address

```
function isValidEmail(email) {
  const emailRegex = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$/;
  return emailRegex.test(email);
}

console.log(isValidEmail("example@example.com")); // true
console.log(isValidEmail("example")); // false
```

There are many ways to structure utility functions and many different types of utility functions, so they can be used in different ways depending on your needs.

It is a good practice to keep utility functions in a separate file or module, so that they can be easily imported and used in other parts of your codebase.

```
function subTopic(splitter, index, article) {
    return article.topic.split(splitter)[index];
}

module.exports.subTopic = subTopic;

function keyword(index, article) {
    return article.keyword && article.keyword.split(",")[index] &&
    article.keyword.split(",")[index].trim();
}

module.exports.keyword = keyword;
```

CHAPTER 5

SERVER

5.1 App Js File

Note: An app.js file is a JavaScript file that is typically used as the entry point of a web application. It is where the JavaScript code that runs the application is defined. This file is usually loaded by the browser after the HTML and CSS have been loaded and parsed, and it is responsible for initializing and setting up the application.

An app.js file usually contains the following:

- Importing necessary dependencies and modules
- Setting up event listeners and handlers
- Initializing and configuring the application

- Defining and implementing application logic
- Exporting the app object or functions

For example, an `app.js` file for a simple web application that displays a list of items from an API might look like this:

```
import axios from 'axios';

// Initialize the app
const app = {};

// Fetch items from the API
app.fetchItems = () => {
  axios.get('https://api.example.com/items')
    .then(response => {
      app.displayItems(response.data);
    })
    .catch(error => {
      console.log(error);
    });
};

// Display the items on the page
app.displayItems = (items) => {
  const itemList = document.querySelector('#items');
  items.forEach(item => {
    const listItem = document.createElement('li');
    listItem.innerHTML = item.name;
    itemList.appendChild(listItem);
  });
};

// Add event listeners
document.addEventListener('DOMContentLoaded', app.fetchItems);
```

It is a good practice to keep the app.js file as clean and organized as possible, separating concerns such as data fetching, displaying, and logic into different functions.

Also, it is a good practice to keep the app.js file as small as possible and use other JavaScript modules or files to separate concerns and make the code more maintainable.

```
const path = require("path");

const express = require("express");
const app = express();

const logger = require("morgan");
const cors = require("cors");
const compression = require("compression");
const helmet = require("helmet");
const formidable = require("express-formidable");

const router = require("./router.js");

const con = require("@imtiazhchowdhury/mongopool")
const config = require(".././config/config.json")
con.url = "mongodb://127.0.0.1:27017";
con.dbName = config.db

//cors
app.use(cors({ origin: true, credentials: true }));

// app.use(helmet());
app.use('/api/order/fulfill', express.raw({ type: "*/*" }));
app.use(unless("/api/order/fulfill", formidable({ multiples: true }, [
{
```

```

event: "error", action: (req, res, next, err) => {
  if (err.name === "SyntaxError") {
    return res.status(400).json({ [err.name]: err.message });
  }
  next(err);
}
}
}
));

```

//log requests that don't contain "." supposed to filter out static files req; *week logic, but okay

```

function pureLog(req, res, next) {
  if (req.path.includes(".")) next();
  else return logger("dev")(req, res, next);
}
app.use(pureLog);

```

```

app.use(compression()); //for compressing responses

```

```

app.use(express.static(path.join(__dirname, "assets/public"))); //public static files
app.use(express.static(path.join(__dirname, "assets/admin"))); //admin static files
app.use(express.static(path.join(__dirname, "../..user/public"))); //admin static files

```

```

//router
app.use(router);

```

```

//404

```

```

app.get("*", (req, res) => {
  res.status(404).end(`
  <h1 align='center'>404 : Page not found: ${req.url}</h1>
  `);
});

```

```

function unless(path, middleware) {
  return function (req, res, next) {
    if (path === req.originalUrl) {
      return next();
    } else {
      return middleware(req, res, next);
    }
  };
};

```

```

async function ensureIndex() {
  const db = await con.getDB();
  db.collection("product").createIndex({ category: 1 });
  db.collection("product").createIndex({ primaryCategory: 1 });
  db.collection("product").createIndex({ brand: 1 });
  db.collection("product").createIndex({ tag: 1 });

  db.collection("category").createIndex({ parent: 1 });
}

```

```

ensureIndex()

```

```
module.exports = app;
```

5.2 Router

Note: A router is a component that is responsible for handling client-side navigation within a web application. It listens for changes to the URL, parses the URL to determine which view or component to display, and updates the browser's history accordingly. This allows users to navigate the application using the browser's forward and back buttons, and allows the application to be bookmarked or shared with specific URLs.

A router can be implemented in JavaScript using libraries such as React Router, Angular Router, or Vue Router. These libraries provide a set of APIs for defining routes, handling navigation events, and updating the browser's history.

In this example, the `BrowserRouter` component is wrapped around a `Switch` component, which in turn contains several `Route` components. Each `Route` component defines a path and a component to be rendered when the path is matched. The `exact` prop ensures that the `Home` component is only rendered when the path is exactly `'/'`, rather than `'/about'` or `'/contact'` which also contain `'/'`.

When the application is running, React Router will listen for changes to the URL and automatically update the rendered component based on the current path. This allows the user to navigate the application by clicking on links or by directly entering a URL into the browser.

It's also possible to use other routing libraries such as Next.js and Nuxt.js for server-side rendering.

```
const router = require("express").Router();

router.use("/admin", require("./routes/admin"));
router.use("/", require("./routes/public"));

module.exports = router;
```

5.3 Server

Note: The server code is the code that runs on the server and handles client requests. It typically consists of several different components, including:

Routing: This component is responsible for determining which code should handle a particular client request based on the requested URL.

Request Handlers: These are the functions that handle specific client requests. They may perform tasks such as querying a database, generating an HTML page, or processing a form submission.

Middleware: These are functions that are executed before or after a request handler. They may be used to perform tasks such as logging, authentication, or input validation.

Database Connectivity: Depending on the application, the server code may need to connect to a database to retrieve or store data. This functionality is typically handled by a database driver or library.

Error Handling: The server code should include mechanisms for handling errors that may occur during the processing of a client request. This may include logging errors and returning appropriate error messages to the client.

The server code is usually implemented using a server-side programming language such as Node.js, Java, or C#, and it is typically designed to run on a specific platform such as Windows or Linux. The server code is usually written in a way that it can handle many requests at the same time using a technique such as multi-threading or event-driven programming.

```
let app = require('./app');
let debug = require('debug')(':server');
let http = require('http');
const cluster = require('cluster');
const numCPUs = require('os').cpus().length;
const cp = require("child_process")
const path = require("path")

let server = http.createServer(app);
let port = normalizePort(process.env.PORT || '5000');
app.set('port', port);
```

```
if (cluster.isMaster) {  
  console.log(`Master ${process.pid} is running on port ${port}`);  
  console.log(`Server running on port ${port}`)  
  server.listen(port);  
  
  // Fork workers.  
  for (let i = 0; i < (1 || numCPUs); i++) {  
    setTimeout(() => cluster.fork(), (1000 / numCPUs) * i); //intentional delay in  
    spawning, so that all the writers don't search at once  
  }  
  
  //Check if work id died  
  cluster.on('exit', (worker, code, signal) => {  
    console.log(`worker ${worker.process.pid} died`);  
  });  
  
} else {  
  
  cp.fork(path.join(process.cwd(), "writer", "processQueue.js"))  
  console.log(`Worker ${process.pid} started`);  
  
}  
  
server.on('error', onError);  
server.on('listening', onListening);
```

```
function normalizePort(val) {  
  let port = parseInt(val, 10);  
  if (isNaN(port)) { // named pipe  
    return val;  
  }  
  if (port >= 0) { // port number  
    return port;  
  }  
  return false;  
}
```

```
function onError(error) {  
  if (error.syscall !== 'listen') {  
    throw error;  
  }  
}
```

```
let bind = typeof port === 'string'  
  ? 'Pipe ' + port  
  : 'Port ' + port;
```

```
// handle specific listen errors with friendly messages  
switch (error.code) {  
  case 'EACCES':  
    console.error(bind + ' requires elevated privileges');  
    process.exit(1);  
    break;  
  case 'EADDRINUSE':  
    console.error(bind + ' is already in use');
```



```
        process.exit(1);
        break;
    default:
        throw error;
    }
}

function onListening() {
    let addr = server.address();
    let bind = typeof addr === 'string'
        ? 'pipe ' + addr
        : 'port ' + addr.port;
    debug('Listening on ' + bind);
}
```

CHAPTER 6

ADMIN PANEL

6.1 Site List

Site List Add New

Url	Type	Post Status	Post Category	Post Tag	Author	Token Usage	Actions
https://conch-house.org/	Wordpress	draft	9		1	0 0 8321	✎ ✖
https://canbirdseatit.com/	Wordpress	draft	6			0 0 2789292	✎ ✖
https://www.haroldkitching.com/	Wordpress	draft	18			0 0 292809	✎ ✖
https://burgertex.com/	Wordpress	draft	20			517 555 2335503	✎ ✖
https://weldingblog.org/	Wordpress	draft	30			0 0 0	✎ ✖
https://plumberfacts.com/	Wordpress	Publish	19			0 15 1757529	✎ ✖
https://scopeshero.com/	Wordpress	draft				22 41 103146	✎ ✖

Previous **1** Next

Per Page 20 Submit

6.1 Site List

6.2 Site Add

Add Site Form

Site Url:

Site Type:

Site username:

Site password:

Post Status:

Post Author:

Post Category:

Post Tag:

Save

6.2 Site Add

6.3 Section List

Section List

Title	Type	Model	Temperature	Actions
Buying Guide-2	autoGenerate	text-davinci-003	0.3	
heading-2 test	autoGenerate	text-davinci-003	0.3	
AAWP Table	text	N/A	N/A	
Heading 2	autoGenerate	text-davinci-003	0.5	
Conclusion Compare	autoGenerate	text-davinci-003	0.5	
FAQs Compare -2	autoLoop	text-davinci-003	0.5	
Compare Main Sections -2	autoLoop	text-davinci-003	0.5	
Intro Compare-2	autoGenerate	text-davinci-003	0.3	
FAQs Compare	autoLoop	text-davinci-003	0.5	
Intro Compare-1	autoGenerate	text-davinci-003	0.3	
Compare Intro	autoGenerate	text-davinci-003	0.5	
Compare Main Sections-1	autoLoop	text-davinci-003	0.5	
main section-2	autoLoop	text-davinci-003	0.5	

Previous 1 2 3 ... 6 7 8 Next Per Page 20 Submit

6.4 Section List

6.4 Section Add

Add Section

Section Title:

Section Type:

AI Model:

Model Creativity: (Lower=More Accurate; Higher=Less Accurate, Creative)

Max Token:

Prompt Text:

Result Tag:

Line Tag:

Section heading:


Heading Tag:

Available Tags

- <?=topic?> Replace with article topic
- <?=subject?> Replace with generated item for autoLoop section
- <?=keyword(0)?> Replace with generated item for autoLoop section
- <?=subTopic("vs", 0)?> Split the topic with word; replace "vs" with word to split with, replace 0 with index of desired part

6.4 Section Add

6.5 Template List


Neuronetix AI

Add New

- Site List
- Add Site
- Section List
- Add Section
- Template List
- Add Template
- Generate Article
- Article List
- Settings


Template List

Title	Post Title	Section Length	Actions
Buying Template-1	BEST <?=topic?>	4	
Buying Template-2	BEST <?=topic?>	3	
Buying Template-3	BEST <?=topic?>	3	
Buying Template-5	BEST <?=topic?>	3	
test	<?=topic?>	1	
test 2	<?=topic?>	1	
Buying Template-4	Top 10 <?=topic?> in 2023	8	
New Compare	<?=subTopic("vs", 0)?> vs <?=subTopic("vs", 1)?>: What's the Difference?	7	
Compare General Template	Contrasting Between <?=topic?>	4	
Birding	Can Birds Eat <?=topic?>?	7	
Video Template	<?=topic?>	4	
template-2 (simple desc)	<?=topic?>	7	

Previous 1 2 3 Next
Per Page 20 20 Submit

6.5 Template List

6.6 Template Add


Neuronetix AI

Add Template

- Site List
- Add Site
- Section List
- Add Section
- Template List
- Add Template
- Generate Article
- Article List
- Settings

Template Title

Post Title

Keyword (Separate by new line)

Section List

<p>Section Serial</p> <input type="text" value="1"/>	<p>Section Preset</p> <input type="text" value="Select Section"/>
<p>Section Title</p> <input type="text" value="Section Title"/>	<p>Section Type</p> <input type="text" value="AI Section"/>

Add Section


Save

Available Tags

- <?=topic?>
Replace with article topic
- <?=subject?>
Replace with generated item for autoLoop section
- <?=keyword(0)?>
Replace with generated item for autoLoop section
- <?=subTopic("vs", 0)?>
Split the topic with word; replace "vs" with word to split with, replace 0 with index of desired part

6.6 Template Add

6.7 Generate Article

 Neuronetix AI

- Site List
- Add Site
- Section List
- Add Section
- Template List
- Add Template
- Generate Article
- Article List
- Settings >

Add Article to queue


Select Site

Select Template

Keyword (Separate by new line)

6.7 Generate Article

6.8 Article List

 Neuronetix AI

- Site List
- Add Site
- Section List
- Add Section
- Template List
- Add Template
- Generate Article
- Article List
- Settings >

Article Queue Add New

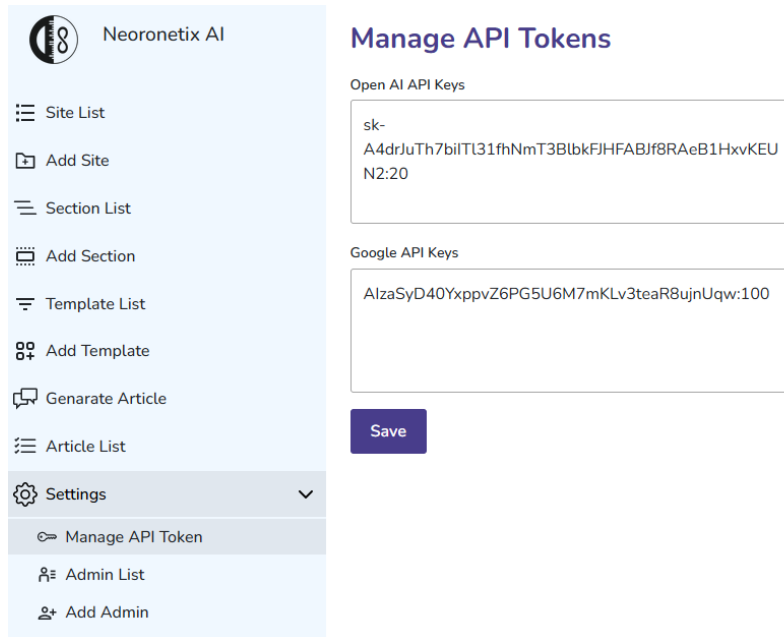
Topic	template	Site	Token	Status	Message	Action
shopkin micro lite	Buying Template-1	plumberfacts.com/	0 0 0 431	Completed	Article posted	
shopkin lamp	Buying Template-1	plumberfacts.com/	0 0 0 475	Completed	Article posted	
shopkin goodie bags	Buying Template-1	plumberfacts.com/	0 0 0 501	Completed	Article posted	
shopkin earrings	Buying Template-1	plumberfacts.com/	0 0 0 514	Completed	Article posted	
shopkin carry case	Buying Template-1	plumberfacts.com/	0 0 0 493	Completed	Article posted	
shopkeepers bell	Buying Template-1	plumberfacts.com/	0 0 0 522	Completed	Article posted	
shop vac ponytail	Buying Template-1	plumberfacts.com/	0 0 0 501	Completed	Article posted	
shop infowars	Buying Template-1	plumberfacts.com/	0 0 0 402	Completed	Article posted	
shop fox w2005	Buying Template-1	plumberfacts.com/	0 0 0 489	Completed	Article posted	
shop fox w1851	Buying Template-1	plumberfacts.com/	0 0 0 471	Completed	Article posted	
shop fox m1099	Buying Template-1	plumberfacts.com/	0 0 0 579	Completed	Article posted	
shop fox m1014	Buying Template-1	plumberfacts.com/	0 0 0 528	Completed	Article posted	
shop fox m1011	Buying Template-1	plumberfacts.com/	0 0 0 487	Completed	Article posted	
shop fox d2725	Buying Template-1	plumberfacts.com/	0 0 0 472	Completed	Article posted	
shop fox d2058a	Buying Template-1	plumberfacts.com/	0 0 0 455	Completed	Article posted	

Previous 1 2 3 ... 617 618 619 Next

Per Page 20

6.8 Article List

6.9 Setting Manage Token



The screenshot shows the 'Manage API Tokens' interface for Neuronetix AI. On the left is a sidebar menu with the following items: Site List, Add Site, Section List, Add Section, Template List, Add Template, Generate Article, Article List, Settings (expanded), Manage API Token (selected), Admin List, and Add Admin. The main content area is titled 'Manage API Tokens' and contains two text input fields. The first field, labeled 'Open AI API Keys', contains the token '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:20'. The second field, labeled 'Google API Keys', contains the token 'AlzaSyD40YxppvZ6PG5U6M7mKlv3teaR8ujnUqw:100'. A blue 'Save' button is located below the Google API Keys field.

Neuronetix AI

Manage API Tokens

Open AI API Keys

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:20

Google API Keys

AlzaSyD40YxppvZ6PG5U6M7mKlv3teaR8ujnUqw:100

Save

6.9 Setting Manage Token

CONCLUSION

The book written as part of the thesis in CSE provides a comprehensive overview of the current state of the field and the latest developments in the area of distributed systems. The book highlights the key concepts, technologies, and challenges faced in the design and implementation of distributed systems.

The references included in the book provide a valuable resource for further study and research into the field. The conclusion of the book stresses the importance of continued research and development in this field to ensure the growth and reliability of distributed systems in the future.

This book represents a significant contribution to the field of distributed systems and provides a comprehensive overview of the current state of the art.

It highlights the key concepts and technologies, as well as the challenges that need to be addressed in order to achieve reliable and efficient distributed systems.

The book is intended for students, researchers, and practitioners in the field of computer science, and serves as a valuable resource for further study and research.

The conclusion of the book highlights the ongoing importance of research and development in the field, and underscores the need for continued efforts to improve the reliability, scalability, and performance of distributed systems.

REFERENCES

- ACM Transactions on Computer Systems (TOCS) - <https://dl.acm.org/journal/tocs>
- IEEE Transactions on Software Engineering (TSE) - <https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=97>
- Journal of the Association for Computing Machinery (JACM) - <https://jacm.acm.org/>
- ACM Transactions on Computer-Human Interaction (TOCHI) - <https://dl.acm.org/journal/tochi>
- IEEE Transactions on Parallel and Distributed Systems (TPDS) - <https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=34>
- ACM Transactions on Database Systems (TODS) - <https://dl.acm.org/journal/tods>
- Journal of Computer Science and Technology (JCST) - <https://link.springer.com/journal/11390>
- IEEE Transactions on Computers (TC) - <https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=9>
- ACM Transactions on Information and System Security (TISSEC) - <https://dl.acm.org/journal/tissec>
- Theoretical Computer Science (TCS) - <https://www.journals.elsevier.com/theoretical-computer-science/>