*A project report on*

**VOYAGER - SAAS PLATFORM FOR ONLINE TRAVEL AGENCIES AND AIRLINES**

*Submitted in partial fulfillment for the award of the degree of*

Bachelor of Technology in

Computer Science and Engineering

*by*

**ANCHURI HARISH (18BCE0342)**



# SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

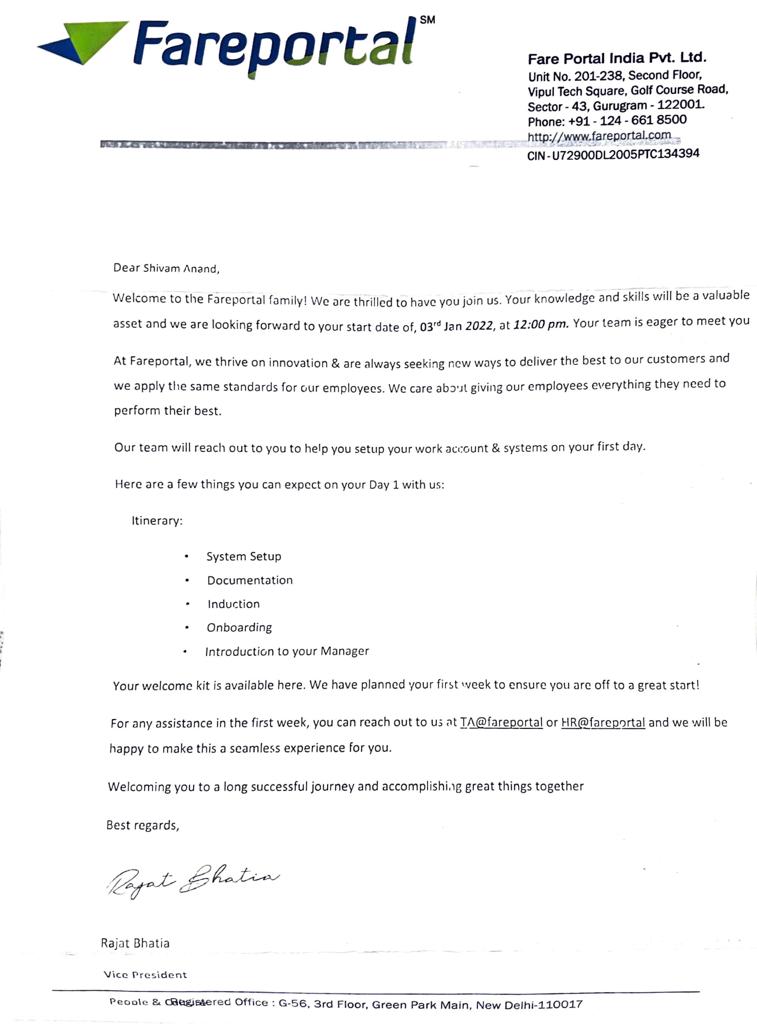
June 2022

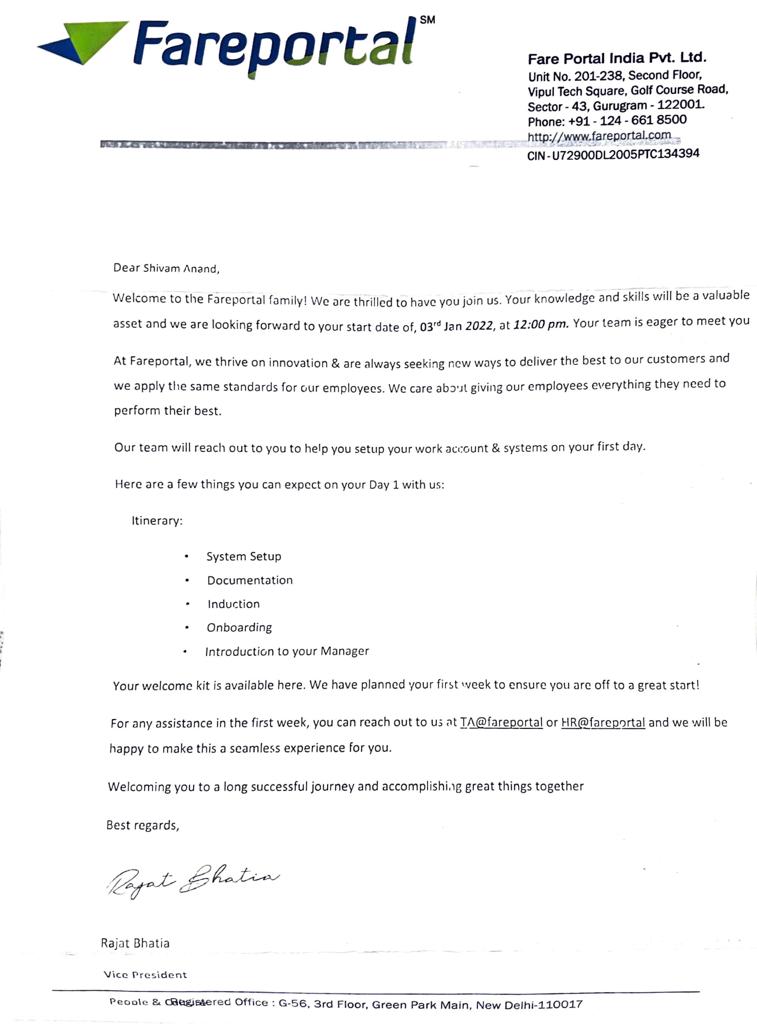
## DECLARATION

I hereby declare that the thesis entitled **“Voyager - SaaS Platform for OTAs and Airlines”** submitted by me to Vellore Institute of Technology, Vellore, for the award of the degree of **Bachelors in Technology** in **Computer Science and Engineering** is a record of bonafide work carried out by me under the supervision of **Mr. Shashikant Singh**. I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Gurugram A. HARISH

Date:10/06/2022 Signature of the Candidate





## ABSTRACT

Fareportal Pvt. Ltd. has many in-house solutions that we use on a regular basis which caters to our business needs. These solutions/ services are developed, maintained, updated and handled by our in-house teams. This is possible because of the scale and size at which our company operates and we can afford to have such solutions which are custom made and used by thousands of people worldwide through our products.

There are many OTAs, Airlines, Agents, etc., who are not large enough in scale or capacity that they can have an independent solution to their business needs. Here we have an opportunity to provide these solutions to others on a SaaS platform through which they can just Signup and choose the service that they require and make use of it.

They will be charged for the services that they make use of on a subscription-based model. It is a win-win situation for both. Because these OTAs, Airlines, etc., get the benefits of our services without the need to worry about the overhead costs that are associated with creating or maintaining these services. They just need to subscribe and get going. Fareportal Pvt. Ltd. Also, benefits from such an association because this adds on as a source of revenue other than the normal streams that are generated through our products.

Voyager is a proprietary tool/ product which is a SaaS platform that is being developed by Fareportal Pvt. Ltd. to provide the services available with us for addressing their needs to other businesses.

## ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to Dr G. Viswanathan, Chancellor, VIT University for creating an environment conducive for learning and growth academically as well as in extracurricular activities that enable every student to reach greater heights.

I would like to thank Dr. S Vairamuthu, HOD School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, for always encouragement and support, without which this project and the last four years would not have been possible.

I would also like to thank all the faculty and laboratory staff of the Department of Computer Science and Engineering.

I would like to express my gratitude to Rajat Bhatia SVP, People & Culture Fareportal India Pvt. Ltd for giving me an internship opportunity at Fareportal India Pvt. Ltd. Additionally, I would like to thank my managers and mentors Nikunj Dhawan, Ashutosh Shandilya, Nitin Mehrotra, Anuj Abrol, Gaurav Aggarwal, Shashikant Singh, Lalit Saini and team Shantanu M G for guiding and being a helping hand throughout the internship.

Lastly, I would like to thank the entire Operations team at Fareportal India Pvt. Ltd. for welcoming me into the organization and helping me tirelessly during every point in the internship.

It is indeed a pleasure to thank my family and friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Gurugram, India

# Date:10/06/2022 A.HARISH

# CONTENTS

ABSTRACT....................................................................................................i

ACKNOWLEDGEMENTS..........................................................................ii

CONTENTS................................................................................................. iii

LIST OF FIGURES........................................................................................vi

LIST OF ACRONYMS...............................................................................vii

CHAPTER 1 INTRODUCTION.................................................................1

* 1. [ABOUT THE COMPANY.......................................................................1](#_TOC_250009)
  2. [PRODUCTS..............................................................................................2](#_TOC_250008)

**CHAPTER 2 BACKGROUND.....................................................................5**

2.1 WHAT IS AN OTA?.................................................................................5

2.2 HOW OTAs WORK?................................................................................5

2.3 WHAT ARE THE ADVANTAGES OF BOOKING THROUGH AN OTA?..............................................................................................................6

2.4 WHAT ARE THE VARIOUS BUSINESS MODELS? ..........................7

2.5 WHAT IS SAAS?.....................................................................................9

2.6 WHAT ARE THE CHARACTERISTICS OF SAAS? ..........................9

2.7 BUSINESS PROBLEM..........................................................................11

CHAPTER 3 PROJECT OVERVIEW.....................................................12

* 1. [SYSTEM](#_TOC_250007) ANALYSIS..........................................................................13
  2. FEATURES OF OUR PLATFORM.....................................................13
  3. BRIEF OVERVIEW OF TECHNOLOGY...........................................14
  4. TOOLS USED.......................................................................................15

[CHAPTER 4 FEASIBILITY](#_TOC_250002) STUDY ....................................................31

* 1. ECONOMIC FEASIBILITY .................................................................31

4.2 TECHNICAL FEASIBILITY ...............................................................32

4.3 OPERATIONAL FEASIBILITY...........................................................32

CHAPTER 5 DESIGN STRATEGY ........................................................33

* 1. REQUIREMENT GATHERING...........................................................33
  2. WORK SCHEDULE .............................................................................36

CHAPTER 6 ARCHITECTURE .............................................................37

* 1. MONOLITHIC ARCHITECTURE......................................................37

6.2 MICROSERVICES ARCHITECTURE ..............................................40

6.3 OUR APPROACH ..............................................................................51

CHAPTER 7 MODULE ANALYSIS......................................................52

7.1 MODULES..........................................................................................52

CHAPTER 8 CONCLUSIONS AND FUTURE WORK ......................54

8.1 CONCLUSION ...................................................................................54

8.2 FUTURE WORK ................................................................................54

8.3 REFERENCES ...................................................................................55

**List of Figures**

FIGURE 1: LOGO OF COMPANY

FIGURE 2: LOGO OF CHEAPOAIR

FIGURE 3: LOGO OF ONETRAVEL

FIGURE 4: LOGO OF TRAVEL LONG

FIGURE 5: LOGO OF ROYAL SCENIC

FIGURE 6: LOGO OF DUKE’S COURT TRAVEL

FIGURE 7: SAAS DESCRIPTION

FIGURE 8 LOGO OF HTML

FIGURE 9: LOGO OF CSS

FIGURE 10: LOGO OF JAVASCRIPT

FIGURE 11: LOGO OF REACT

FIGURE 12: LOGO OF TYPESCRIPT

FIGURE 13: LOGO OF C#

FIGURE 14: LOGO OF DOT NET

FIGURE 15: LOGO OF MYSQL

FIGURE 16: LOGO OF MONGO DB

FIGURE 17: LOGO OF REDIS

FIGURE 18: LOGO OF VS CODE

FIGURE 19: LOGO OF VISUAL STUDIO

FIGURE 20: LOGO OF MICROSOFT AZURE

FIGURE 21: LOGO OF POSTMAN

FIGURE 22: LOGO OF MONOLITHIC ARCHITECTURE

FIGURE 23: LOGO OF MICROSERVICES ARCHITECTURE

FIGURE 24: LOGO OF SYNCHRONOUS COMMUNICATION

FIGURE 25: LOGO OF ASYNCHRONOUS COMMUNICATION

FIGURE 26: LOGO OF RESTFUL COMMUNICATION

FIGURE 27: LOGO OF RPC COMMUNICATION

FIGURE 28: LOGO OF MESSAGE INTEGRATION

FIGURE 29: LOGO OF STREAMING INTEGRATION

**LIST OF ACRONYMS**

1. SaaS: Software as a Service

2. SQL: Structured Query Language

3. RDBMS: Relational Database Management System

4. CI/CD: Continuous Integration/ Continuous Deployment

5. API: Application Programming Interface

6. NPM: Node Packet Manager

7. VM: Virtual Machine

8. JS: Java Script

9. OTA: Online Travel Agency

10. B2B: Business to Business

11. B2C: Business to Consumer

12. ATOL: Air Travel Organizers' Licensing

13.DDL: Data Definition Language

14. SSPL: Server Side Public License

15. JSON: Javascript Object Notation

16. FTP: File Transfer Protocol

**Chapter 1**

**Introduction**

* 1. **ABOUT THE COMPANY**



Figure 1. Company Logo

Fareportal is a technology company that powers leading hybrid travel agencies like CheapOair, OneTravel, and Travelong, a veteran corporate travel agency founded in Morristown, New Jersey in 1976. our strong knowledge and experience in the air travel booking vertical, including corporate travel management and automation of wholesale distribution of complex international airfares, today we operate in North America, Europe, and Asia. We’re also proud to partner with over 500 airlines, over 1 million hotels, and hundreds of car agencies worldwide.

Fareportal’s unique hybrid business model bridges the gap between an online travel agency and a traditional travel agency by providing a convenient online booking capability as well as a 24/7 personalized trip booking experience arranged by hundreds of trained and certified travel agents in multiple countries and in multiple languages. By leveraging - 5 -company owned and operated contact centers located worldwide, we are able to deliver one of the highest levels of customer service and support in the air travel business.

Travelers around the world can find and book their perfect trip on our websites, mobile and tablet apps, and by calling one of our hundreds of trained and certified travel agents.

Fareportal’s partner airlines benefit from the broad customer reach and strong customer value proposition as compared to the pure OTA model. Focused more on selling international and higher yield complex airfares, partner airlines obtain higher revenue on a per seat basis.

Fareportal’s constant innovation allows partner airlines to benefit from selling ancillaries on the booking path thereby further optimizing their revenues and making them more predictable. This also helps the Airlines in streamlining airport operations, having less people wait in line to pay for preferred seats and bags. Read more about how Fareportal has been first to sell ancillaries for American Airlines, Air Canada and Spirit Airlines.

* 1. **PRODUCTS**

**CheapOair**

CheapOair is the smartest way for savvy travelers to book their next trip. CheapOair allows customers to compare and book flights on over 450 airlines online, on award-winning mobile apps, or by phone with live travel agents.



*Figure 2. CheapOair Logo*

**OneTravel**

OneTravel simplifies the travel shopping experience by combining value-driven deals on flights, hotels, cars and vacation packages with 24/7 customer care only one mouse click or phone call away.



*Figure 3. OneTravel Logo*

**Fareportal Media Group**

Fareportal Media Group is the independent media division that manages the advertising operations, sales and business development for Fareportal’s travel websites. Through custom, integrated solutions, Fareportal ensures that its advertising partners get the best return on their investment.

**Travelong**

Travelong is a full-service travel agency serving more than 100 corporate clients and 30 home-based agents across the United States. Established in 1933, Travelong’s innovative and highly experienced travel consultants have provided client satisfaction for over eight decades.



*Figure 4. Travelong Logo*

**Royal Scenic**

Royal Scenic is a national wholesale supplier, air consolidator and preferred Fareportal partner with offices in Vancouver, Toronto and Montreal. Royal Scenic offers quality products and customer service exclusively to the retail travel community for both corporate and leisure markets.



*Figure 5. Royal Scenic Logo*

**Dukes Court Travel Ltd.**

Duke’s Court Travel is based in London and offers a wide range of travel services. Established in 1989, Duke’s Court Travel is an appointed agency for over 50 major international airlines. Licensed by the UK Civil Aviation Authority under ATOL, it offers full financial protection to any who book directly through the agency.



*Figure 6. Duke’s Court Travel Logo*

**Chapter-2**

**Background**

This chapter comprises the overall background required for the understanding of the project. The existing system in our organization, and the requirements and need as well as the gains that can be achieved through our product, Voyager. It also talks in detail about the Cause, Motive, Problem Statement, Objectives and end goals of the project.

#### 2. 1 WHAT IS AN ONLINE TRAVEL AGENCY?

An online travel agency, or OTA, is a website or online service, which sells travel related products to customers. These products may include hotels, flights, travel packages, activities and car rentals. Crucially, OTAs are third parties, reselling these services on behalf of other companies.

Typically, an OTA will offer many of the benefits of using an offline travel agency, with added convenience and more of a self-service approach. They will also include a built-in booking system, allowing for instant bookings.

Examples of online travel agencies include Fareportal, Expedia, Booking.com, Agoda, etc.

2.2HOW ONLINE TRAVEL AGENCIES WORK?

Online travel agencies can be used by the general public for leisure travel, and by organizations for business travel. The platforms differ in what they offer, but the premise remains the same: to facilitate travel and its associated admin and costs.

Hotels, airlines, and car hire companies pay online travel agencies to appear on their website. On the other hand, the advantage for the service provider is the added visibility that OTAs can provide.

OTAs work by providing a streamlined approach to all things travel. It involves the customer accessing a website or platform on which they’re offered a wide variety of travel options from a number of different travel and accommodation providers.

If the airlines and hotels are the raw data points, OTAs are those that gather and present all the necessary details for consumers to make a travel decision.

2.3 WHAT ARE THE ADVANTAGES OF BOOKING THROUGH AN ONLINE TRAVEL AGENCY?

Some of the benefits of booking travel services through an OTA are:

1. **Access to comparison tools**

The main advantage of using an OTA is that we can quickly and easily compare the prices of accommodation, flights, and more. Most have search filters and comparison tools so we can target our search and find exactly what we’re looking for.

1. **Peer reviews to help us with our decisions**

Another benefit to booking through an online travel agency is the review system, whereby users leave ratings and written reviews for future customers. The option to leave a review is usually only available after a user has booked a service, so they’re a reliable source of feedback.

1. **Flexible cancellation policies**

To be featured on an online travel agency, hotels, airlines, and other service providers often must adhere to the OTA’s cancellation policy. All of the information is clearly laid out for the customer in an email or on the website upon booking, which makes it very user-friendly and customer-centric.

Note: Not all OTA’s have their own cancellation and changes policy, some take us to the vendors website to confirm our ticket. This then means we are subject to the vendor’s cancellation and changes policy, which may not be all that flexible.

1. **All our travel in one place**

We can also use one account for all our different travel plans, saving the hassle of entering details over and over again. A good OTA is an intuitive OTA—we want to be able to easily navigate our options without the need for constant support.

Again, some platforms—mainly aggregator sites—will send us to a third-party website for booking. This means we’ll need to liaise directly with a third-party for any changes.

1. **Local flights and deals**

OTAs personalize things like measurements and currencies to improve the individual user’s experience. Consequently, independently run establishments that may not have the resources to adapt their website internationally are unable to compete.

We’re offered the best deals for our area in a currency you know—making sure everything’s crystal clear when it comes to booking.

1. **Rewards programs**

Lots of online travel agencies offer reward systems, where there are points and levels based on the frequency of tickets booked. These offer frequent users access to special deals and discounts—much like the rewards programs we get with an airline or hotel.

2.4 WHAT ARE VARIOUS BUSINESS MODELS?

There are two basic kinds of business models when a product/ service is offered to the consumers of the Online Travel Agency Industry.

1. **B2C (Business-to-Consumer) model**

It is a retail model where products move directly from our business to the end user who has purchased the product or service for personal use. Typically, our existing products cater to the needs of individual customers directly and hence are involved in personal dealings with people.

B2C business model has been associated with a number of potential benefits, including:

1. **Globalization**. Larger-scale B2C businesses frequently have very large target audiences. Therefore, advertisements and marketing campaigns can reach millions of potential customers, via the internet and social media.
2. **Relatively low cost**. For the sale of our services, B2C models can cut down on operational and physical infrastructure costs.
3. **Customer personalization**. A B2C business can directly market to consumer segments and niche target audiences, allowing our company to personalize their marketing for individual consumers.
4. **Direct customer experience** Both online and brick-and-mortar businesses have complete control over user experiences. This control can lead to better customer service, increased cross-selling and customer loyalty.
5. **Customer data.** Valuable customer data -- such as sales conversion stats, email addresses for marketing automation, analytics, customer behavior patterns, geographic regions and psychographics -- can strengthen a business's marketing strategy and provide insights about users.

1. **B2B (Business-to-Business) model**

Fareportal has already had many brands under the parent company which caters to customers directly and they have decided to venture into B2B business space by making a provision of providing the services that are proprietary and are being used internally by their brands on a subscription basis where other airline, OTAs, etc., can avail themselves of the services used by our brands. This is planned to be achieved by the product, Voyager, in the form of a Saas offering.

2.5 WHAT IS SOFTWARE AS A SERVICE (SAAS)?

Software as a service (or SaaS) is a way of delivering applications over the Internet—as a service. Instead of installing and maintaining software, you simply access it via the Internet, freeing yourself from complex software and hardware management.



Figure 7. SaaS depiction

SaaS applications are sometimes called Web-based software, on-demand software, or hosted software. Whatever the name, SaaS applications run on a SaaS provider’s servers. The provider manages access to the application, including security, availability, and performance.

SaaS customers have no hardware or software to buy, install, maintain, or update. Access to applications is easy: You just need an Internet connection.

2.6 WHAT ARE CHARACTERISTICS OF SAAS?

Software as a service (or SaaS) has a way of protecting the privacy of each customer while providing service that is reliable and secure—on a massive scale. Some of the prevalent characteristics are:

1. **Multitenant Architecture**

A multitenant architecture, in which all users and applications share a single, common infrastructure and code base that is centrally maintained. Because SaaS vendor clients are all on the same infrastructure and code base, vendors can innovate more quickly and save the valuable development time previously spent on maintaining numerous versions of outdated code.

1. **Easy Customization**

The ability for each user to easily customize applications to fit their business processes without affecting the common infrastructure. Because of the way SaaS is architected, these customizations are unique to each company or user and are always preserved through upgrades. That means SaaS providers can make upgrades more often, with less customer risk and much lower adoption cost.

1. **Better Access**

Improved access to data from any networked device while making it easier to manage privileges, monitor data use, and ensure everyone sees the same information at the same time.

1. **SaaS Harnesses the Consumer Web**

Anyone familiar with Amazon.com or My Yahoo! will be familiar with the Web interface of typical SaaS applications. With the SaaS model, we can customize with point-and-click ease, making the weeks or months it takes to update traditional business software seem hopelessly old fashioned.

1. **SaaS Trends**

Organizations are now developing SaaS integration platforms (or SIPs) for building additional SaaS applications. This the “third wave” in software adoption: when SaaS moves beyond standalone software functionality to become a platform for mission-critical applications.

2.7 BUSINESS PROBLEM/ BUSINESS OPPORTUNITY/ PAIN POINT IDENTIFIED

Fareportal Pvt. Ltd. has many in-house solutions that we use on a regular basis which caters to our business needs. These solutions/ services are developed, maintained, updated and handled by our in-house teams. This is possible because of the scale and size at which our company operates and we can afford to have such solutions which are custom made and used by thousands of people worldwide through our products.

There are many OTAs, Airlines, Agents, etc., who are not large enough in scale or capacity that they can have an independent solution to their business needs.

Here we have an opportunity to provide these solutions to others on a SaaS platform through which they can just Signup and choose the service that they require and make use of it.

They will be charged for the services that they make use of on a subscription-based model. It is a win-win situation for both. Because these OTAs, Airlines, etc., get the benefits of our services without the need to worry about the overhead costs that are associated with creating or maintaining these services. They just need to subscribe and get going.

Fareportal Pvt. Ltd. Also, benefits from such an association because this adds on as a source of revenue other than the normal streams that are generated through our products.

**Chapter- 3**

**Project Overview**

Today, the Internet have created a new economic scenario that not only stresses on the classical concept of the **“Customer”** but also on the modern concept of **“Service”.** It is this level of service that dictates whether a customer get an Ease or not.

To provide a high accessibility of service we develop a SaaS platform.

The following are the objectives of the project:

1. Easy User Onboarding & Registration where the user should be able to quickly enter basic required details, choose the services and start using our services.
2. Account Activation & Setup of the account for a particular tenant under whom many other users belonging to the organization may be further registered.
3. User Management at Service level where access of a user to a particular service can be controlled and availed.
4. Roles & Permissions Management allows the admin user of a particular account to create roles like manager, executive, etc., with different authorization permission over availing the services.
5. Service Configuration Modeling allows the users to make changes to configurations of the services that they have chosen and also view them.
6. Service Integrations & Usage Management is the component that is responsible for tracking the usage of the services made by the tenants so that they can be billed accordingly and also collecting data for analytics.
7. Support Management is responsible for handling the management of the services and providing support to our tenants regarding usage of the
8. Dashboard & Reports is a place where all the analytic data regarding the tenant account as well as the usage of services by them is provided.

3.1 SYSTEM ANALYSIS

System Analysis is a problem-solving activity that requires intensive communication between the system users and system developers.

It is basically a process of gathering information about the customer requirements for continuous improvement in the Application.

It involves the process of observing systems for troubleshooting or development purposes. The project is totally built at administrative end and thus only the administrator is guaranteed the access.

The purpose of the project is to build a SaaS platform that allows other users to avail the services that are available to Fareportal to cater to their business needs and also provide the ease to the customer.

Our project tries to enhance access to care and improve the continuity and efficiency of service to the customer. It is easy to maintain, ready to run, scalable, affordable and reliable cost saving tool.

3.2 FEATURES OF OUR PLATFORM “VOYAGER”

1. Save responding time
2. Available anytime and anywhere
3. Improve customer engagement
4. Personalize availing of services according to needs
5. Quicker resolutions and comfortable communication
6. It is also affordable

3.3 BRIEF OVERVIEW OF THE TECHNOLOGY  
**Front end: HTML, CSS, JavaScript, React.**

1. HTML: HTML is used to create and save web document. E.g. Notepad/Notepad++
2. CSS: (Cascading Style Sheets) Create attractive Layout
3. JavaScript: it is a programming language, commonly use with web browsers.
4. React is a declarative, efficient, and flexible javascript  library for building user interfaces. It lets you compose complex UIs from small and isolated pieces

**Back end: C#, TypeScript , MySQL, mongoDB**

1. MySQL: MySql is a database, widely used for accessing querying, updating, and managing data in databases.
2. TypeScript: TypeScript is a programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and adds optional static typing to the language. It is designed for the development of large applications and transpiles to JavaScript.
3. C#: **it** a simple, modern, general-purpose, object-oriented programming language developed by Microsoft within its .NET initiative led by Anders Hejlsberg.**C#** is one of the most popular programming languages and can be used for a variety of things

.

3.4 TOOLS USED

**3.4.1 HTML**



Figure 82: Logo of HTML

The HyperText Markup Language or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. Tags such as <img /> and <input /> directly introduce content into the page. Other tags such as <p> surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags but use them to interpret the content of the page.

HTML can embed programs written in a scripting language such as JavaScript, which affects the behaviour and content of web pages. Inclusion of CSS defines the look and layout of content. The World Wide Web Consortium (W3C), former maintainer of the HTML and current maintainer of the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997.

**3.4.2 CSS**



Figure 93: Logo of CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTMLCSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts.[3] This separation can improve content accessibility; provide more flexibility and control in the specification of presentation characteristics; enable multiple web pages to share formatting by specifying the relevant CSS in a separate.css file, which reduces complexity and repetition in the structural content; and enable the.css file to be cached to improve the page load speed between the pages that share the file and its formatting.

Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device.[4]

The name cascading comes from the specified priority scheme to determine which style rule applies if more than one rule matches a particular element. This cascading priority scheme is predictable.

**3.4.3 JavaScript**



Figure 104: Logo of JavaScript

JavaScript often abbreviated JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. Over 97% of websites use JavaScript on the client side for web page behaviour, often incorporating third-party libraries. All major web browsers have a dedicated JavaScript engine to execute the code on users' devices.

JavaScript is a high-level, often just-in-time compiled language that conforms to the ECMAScript standard. It has dynamic typing, prototype-based object-orientation, and first-class functions. It is multi-paradigm, supporting event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

The ECMAScript standard does not include any input/output (I/O), such as networking, storage, or graphics facilities. In practice, the web browser or other runtime system provides JavaScript APIs for I/O.

JavaScript engines were originally used only in web browsers, but are now core components of some servers and a variety of applications. The most popular runtime system for this usage is Node.js.

During the period of Internet Explorer dominance in the early 2000s, client-side scripting was stagnant. This started to change in 2004, when the successor of Netscape, Mozilla, released the Firefox browser. Firefox was well received by many, taking significant market share from Internet Explorer. Meanwhile, very important developments were occurring in open-source communities not affiliated with ECMA work. In 2005, Jesse James Garrett released a white paper in which he coined the term Ajax and described a set of technologies, of which JavaScript was the backbone, to create web applications where data can be loaded in the background, avoiding the need for full page reloads. This sparked a renaissance period of JavaScript, spearheaded by open-source libraries and the communities that formed around them. Many new libraries were created, including jQuery, Prototype, Dojo Toolkit, and MooTools.

**3.4.4 React**

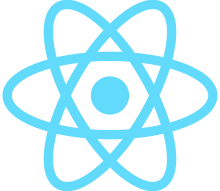


Figure 115: Logo of React

React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library for building user interfaces based on UI components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies. React can be used as a base in the development of single-page, mobile, or server-rendered applications with frameworks like Next.js. However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

React code is made of entities called components. These components are reusable and must be formed in the SRC folder following the Pascal Case as its naming convention (capitalize camelCase). Components can be rendered to a particular element in the DOM using the React DOM library. When rendering a component, one can pass the values between components through "props":

Lifecycle methods for class-based components use a form of hooking that allows the execution of code at set points during a component's lifetime.

* shouldComponentUpdate allows the developer to prevent unnecessary re-rendering of a component by returning false if a render is not required.
* componentDidMount is called once the component has "mounted" (the component has been created in the user interface, often by associating it with a DOM node). This is commonly used to trigger data loading from a remote source via an API.
* componentWillUnmount is called immediately before the component is torn down or "unmounted". This is commonly used to clear resource-demanding dependencies to the component that will not simply be removed with the unmounting of the component (e.g., removing any setInterval () instances that are related to the component, or an "eventListener" set on the "document" because of the presence of the component)
* Render is the most important lifecycle method and the only required one in any component. It is usually called every time the component's state is updated, which should be reflected in the user interface.

JSX, or JavaScript Syntax Extension, is an extension to the JavaScript language syntax. Similar in appearance to HTML, JSX provides a way to structure component rendering using syntax familiar to many developers. React components are typically written using JSX, although they do not have to be (components may also be written in pure JavaScript). JSX is similar to another extension syntax created by Facebook for PHP called XHP.

Hooks are functions that let developers "hook into" React state and lifecycle features from function components. Hooks do not work inside classes — they let you use React without classes.

React provides a few built-in hooks like useState, useContext, useReducer , useMemo and useEffect. Others are documented in the Hooks API Reference. useState and useEffect, which are the most commonly used, are for controlling state and side effects respectively.

**3.4.5 TypeScript**



Figure 128: Logo of TypeScript

TypeScript is a programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and adds optional static typing to the language. It is designed for the development of large applications and trans piles to JavaScript. As it is a superset of JavaScript, existing JavaScript programs are also valid TypeScript programs.

TypeScript may be used to develop JavaScript applications for both client-side and server-side execution (as with Node.js or Deno). Multiple options are available for transpiration. The default TypeScript Compiler can be used, or the Babel compiler can be invoked to convert TypeScript to JavaScript.

TypeScript supports definition files that can contain type information of existing JavaScript libraries, much like C++ header files can describe the structure of existing object files. This enables other programs to use the values defined in the files as if they were statically typed TypeScript entities. There are third-party header files for popular libraries such as jQuery, MongoDB, and D3.js. TypeScript headers for the Node.js basic modules are also available, allowing development of Node.js programs within TypeScript.

TypeScript originated from the shortcomings of JavaScript for the development of large-scale applications both at Microsoft and among their external customers. Challenges with dealing with complex JavaScript code led to demand for custom tooling to ease developing of components in the language.

TypeScript is a strict superset of ECMAScript 2015, which is itself a superset of ECMAScript 5, commonly referred to as JavaScript. As such, a JavaScript program is also a valid TypeScript program, and a TypeScript program can seamlessly consume JavaScript. By default the compiler targets ECMAScript 5, the current prevailing standard, but is also able to generate constructs used in ECMAScript

**3.4.6 C#**

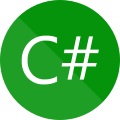


Figure 139: Logo of C#

C# is a general-purpose, multi-paradigm programming language. C# encompasses static typing, strong typing, lexically scoped, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. C# is a simple, modern, general-purpose, object-oriented programming language developed by Microsoft within its .NET initiative led by Anders Hejlsberg. C# is one of the most popular programming languages and can be used for a variety of things.it is a general-purpose, multi-paradigm programming language. C# encompasses static typing, strong typing, lexically scoped, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines.

By design, C# is the programming language that most directly reflects the underlying Common Language Infrastructure (CLI). Most of its intrinsic types correspond to value-types implemented by the CLI framework. However, the language specification does not state the code generation requirements of the compiler: that is, it does not state that a C# compiler must target a Common Language Runtime, or generate Common Intermediate Language (CIL), or generate any other specific format.

**3.4.7 .NET**



Figure 1410: Logo of .NET

The .NET Framework (pronounced as "dot net") is a proprietary software framework developed by Microsoft that runs primarily on Microsoft Windows. It was the predominant implementation of the Common Language Infrastructure (CLI) until being superseded by the cross-platform .NET project. It includes a large class library called Framework Class Library (FCL) and provides language interoperability (each language can use code written in other languages) across several programming languages. Programs written for .NET Framework execute in a software environment (in contrast to a hardware environment) named the Common Language Runtime (CLR). The CLR is an application virtual machine that provides services such as security, memory management, and exception handling. As such, computer code written using .NET Framework is called "managed code". FCL and CLR together constitute the .NET Framework. .NET Framework began as proprietary software, although the firm worked to standardize the software stack almost immediately, even before its first release. Despite the standardization efforts, developers, mainly those in the free and open-source software communities, expressed their unease with the selected terms and the prospects of any free and open-source implementation, especially regarding software patents. Since then, Microsoft has changed .NET development to more closely follow a contemporary model of a community-developed software project, including issuing an update to its patent promising to address the concerns. .NET Framework includes an implementation of the CLI foundational Standard Libraries. The .NET Framework Class Library (FCL) is organized in a hierarchy of namespaces.

NET platforms are encouraged to implement a version of the standard library allowing them to re-use extant third-party libraries to run without new versions of them. The .NET Standard Library allows an independent evolution of the library and app model layers within the .NET

**3.4.8 MySQL**



Figure 1511: Logo of MYSQL

MySQL is an open-source relational database management system (RDBMS). Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language. A relational database organizes data into one or more data tables in which data may be related to each other; these relations help structure the data. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

MySQL is written in C and C++. Its SQL parser is written in yacc, but it uses a home-brewed lexical analyser. MySQL is offered under two different editions: the open-source MySQL Community Server and the proprietary Enterprise Server. MySQL Enterprise Server is differentiated by a series of proprietary extensions which install as server plugins, but otherwise shares the version numbering system and is built from the same code base.

Major features as available in MySQL: -

* A broad subset of ANSI SQL 99, as well as extensions
* Cross-platform support
* Stored procedures, using a procedural language that closely adheres to SQL/PSM
* Triggers
* Cursors
* Updatable views
* Online Data Definition Language (DDL) when using the InnoDB Storage Engine.
* Information schema
* Performance Schema that collects and aggregates statistics about server execution and query performance for monitoring purposes.
* A set of SQL Mode options to control runtime behavior, including a strict mode to better adhere to SQL standards.

**3.4.9 Mongo DB**



Figure 16. Logo on mongoDB

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License (SSPL) which is deemed non-free by several distributions.MongoDB supports field, range query, and regular-expression searches. Queries can return specific fields of documents and also include user-defined JavaScript functions. Queries can also be configured to return a random sample of results of a given size.

MongoDB provides high availability with replica sets. A replica set consists of two or more copies of the data. Each replica-set member may act in the role of primary or secondary replica at any time. All writes and reads are done on the primary replica by default. Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically conducts an election process to determine which secondary should become the primary. Secondaries can optionally serve read operations, but that data is only eventually consistent by default.

If the replicated MongoDB deployment only has a single secondary member, a separate daemon called an arbiter must be added to the set. It has a single responsibility, which is to resolve the election of the new primary. As a consequence, an idealized distributed MongoDB deployment requires at least three separate servers, even in the case of just one primary and one secondary.

Major features available in MongoDB :-

* High availability through built-in replication and failover
* Horizontal scalability with native sharding.
* End-to-end security
* Native document validation and schema exploration with Compass
* Management tooling for automation, monitoring, and backup
* Fully elastic database as a service with built-in best practices

**3.4.10 Redis server**



Figure 17. Logo of redis

Redis (Remote Dictionary Server) is an in-memory data structure store, used as a distributed, in-memory key–value database, cache and message broker, with optional durability. Redis supports different kinds of abstract data structures, such as strings, lists, maps, sets, sorted sets, HyperLogLogs, bitmaps, streams, and spatial indices. The project was developed and maintained by Salvatore Sanfilippo.From 2015 until 2020, he led a project core team sponsored by Redis Labs. Salvatore Sanfilippo left Redis as the maintainer in 2020. It is open-source software released under a BSD 3-clause license. In 2021, not long after the original author and main maintainer left, Redis Labs dropped the Labs from its name and now redis, the open source DB as well as Redis Labs, the commercial company, are referred to as "redis".

Redis popularized the idea of a system that can be considered at the same time a store and a cache, using a design where data is always modified and read from the main computer memory, but also stored on disk in a format that is unsuitable for random access of data, but only to reconstruct the data back in memory once the system restarts. At the same time, Redis provides a data model that is very unusual compared to a relational database management system (RDBMS). User commands do not describe a query to be executed by the database engine but rather specific operations that are performed on given abstract data types. Hence, data must be stored in a way which is suitable later for fast retrieval, without help from the database system in form of secondary indexes, aggregations or other common features of traditional RDBMS. The Redis implementation makes heavy use of the fork system call, to duplicate the process holding the data, so that the parent process continues to serve clients, while the child process creates a copy of the data on disk.

Many programming languages have Redis language bindings on the client side, including: ActionScript, C, C++, C#, Chicken, Clojure, Common Lisp ,Crystal ,D ,Dart, Elixir, Erlang, Go, Haskell, Haxe, Io, Java, Nim, JavaScript (Node.js), Julia, Lua, Objective-C, OCaml, Perl, PHP, Pure Data ,Python ,R ,Racket ,Ruby ,Rust ,Scala, Smalltalk, Swift, and Tcl. Several client software programs exist in these languages.

**3.4.11 Visual Studio Code**



Figure 1815: Logo of Visual Studio Code

Visual Studio Code is a source code editor developed by Microsoft for Windows, Linux and MacOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is free and open-source, although the official download is under a proprietary license. Visual Studio Code can be extended via [extensions](https://en.wikipedia.org/wiki/Plug-in_(computing)), available through a central repository. This includes additions to the editor and language support. A notable feature is the ability to create extensions that add support for new [languages](https://en.wikipedia.org/wiki/Programming_language), [themes](https://en.wikipedia.org/wiki/Theme_(computing)), and [debuggers](https://en.wikipedia.org/wiki/Debugger), perform [static code analysis](https://en.wikipedia.org/wiki/Static_code_analysis), and add [code linters](https://en.wikipedia.org/wiki/Lint_(software)) using the [Language Server Protocol](https://en.wikipedia.org/wiki/Language_Server_Protocol).

Visual Studio Code includes multiple extensions for [FTP](https://en.wikipedia.org/wiki/FTP), allowing the software to be used as a free alternative for web development. Code can be synced between the editor and the server, without downloading any extra software. Visual Studio Code allows users to set the [code page](https://en.wikipedia.org/wiki/Code_page) in which the active document is saved, the [newline](https://en.wikipedia.org/wiki/Newline) character, and the programming language of the active document. This allows it to be used on any platform, in any locale, and for any given programming language.

**3.4.12 Visual studio 2019**

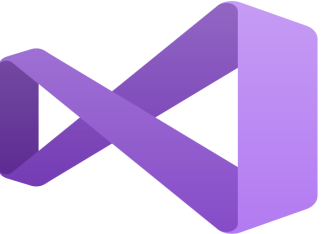


Figure 19. Logo of Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.

Visual Studio includes a code editor supporting IntelliSense (the code completion component) as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a code profiler, designer for building GUI applications, web designer, class designer, and database schema designer. It accepts plug-ins that expand the functionality at almost every level—including adding support for source control systems (like Subversion and Git) and adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle (like the Azure DevOps client: Team Explorer).

Visual Studio supports 36 different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C, C++, C++/ CLI, Visual Basic .NET, C#, F# JavaScript, TypeScript , XML , XSLT, HTML, and CSS. Support for other languages such as Python, Ruby, Node.js, and M among others is available via plug-ins. Java (and J#) were supported in the past.

**3.4.13 Microsoft Azure**



Figure 2016: Logo of Microsoft Azure

Microsoft Azure, often referred to as Azure is a cloud computing service operated by Microsoft for application management via Microsoft-managed data centres. It provides software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS) and supports many different programming languages, tools, and frameworks, including both Microsoft-specific and third-party software and systems. Azure Web Sites allows developers to build sites using ASP.NET, PHP, Node.js, Java, or Python and can be deployed using FTP, Git, Mercurial, Team Foundation Server or uploaded through the user portal. This feature was announced in preview form in June 2012 at the Meet Microsoft Azure event. Customers can create websites in PHP, ASP.NET, Node.js, or Python, or select from several open source applications from a gallery to deploy. This comprises one aspect of the platform as a service (PaaS) offering for the Microsoft Azure Platform.

* Azure Active Directory is used to synchronize on-premises directories and enable SSO
* Azure Active Directory B2C allows the use of consumer identity and access management in the cloud.
* Azure Active Directory Domain Services is used to join Azure virtual machines to a domain without domain controllers.
* Azure information protection can be used to protect sensitive information.
* Storage Services provides REST and SDK APIs for storing and accessing data on the cloud.
* Table Service lets programs store structured text in partitioned collections of entities that are accessed by partition key and primary key. Azure Table Service is a NoSQL non-relational database.

**3.4.14 Postman**



Figure 2117: Logo of Postman

Postman helps in a type of software testing that involves testing application programming interfaces (APIs) directly and as part of integration testing to determine if they meet expectations for functionality, reliability, performance, and security. Since APIs lack a GUI, API testing is performed at the message layer. API testing is now considered critical for automating testing because APIs now serve as the primary interface to application logic and because GUI tests are difficult to maintain with the short release cycles and frequent changes commonly used with Agile software development and DevOps.

**Chapter- 4**

**Feasibility Study**

The feasibility study investigates the problem and the information needs of the stakeholders. It seeks to determine the resources required to provide an information systems solution, the cost and benefits of such a solution, and the feasibility of such a solution. The analyst conducting the study gathers information using a variety of methods, the most popular of which are:

* Interviewing users, employees, managers, and customers.
* Developing and administering questionnaires to interested stakeholders, such as potential users of the information system.
* Observing or monitoring users of the current system to determine their needs as well as their satisfaction and dissatisfaction with the current system.
* Collecting, examining, and analyzing documents, reports, layouts, procedures, manuals, and any other documentation relating to the operations of the current system.
* Modelling, observing, and simulating the work activities of the current system.

The goal of the feasibility study is to consider alternative information systems solutions, evaluate their feasibility, and propose the alternative most suitable to the organization. The feasibility of a proposed solution is evaluated in terms of its components. These components are:

4.1 ECONOMIC FEASIBILITY

The economic viability of the proposed system. The proposed project's cost and benefits are evaluated. Tangible costs include fixed and variable costs, while tangible benefits include cost savings, increased revenue, and increased profit. A project is approved only if it covers its cost in a given period of time. However, a project may be approved only on its intangible benefits such as those relating to government regulations, the image of the organization, or similar considerations.

4.2 TECHNICAL FEASIBILITY

The possibility that the organization has or can procure the necessary resources. This is demonstrated if the needed hardware and software are available in the marketplace or can be developed by the time of implementation.

4.3 OPERATIONAL FEASIBILITY

The ability, desire, and willingness of the stakeholders to use, support, and operate the proposed computer information system. The stakeholders include management, employees, customers, and suppliers. The stakeholders are interested in systems that are easy to operate, make few, if any, errors, produce the desired information, and fall within the objectives of the organization.

**Chapter- 5**

**Design Strategy**

5.1 REQUIREMENT GATHERING

**5.1.1 HARDWARE REQUIREMENT**

* Processor: Pentium Processor ISA 32 Family
* Secondary Storage: 80 GB HDD
* RAM: 52X CD ROM Drive
* Memory: 4 GB RAM
* Network Adapter: Ethernet Adapter
* Modem: 128kbps Voice Fax Data
* Others: 17" Colour Monitor, Printer, Scanner, Keyboard, Mouse.
* Server to Deploy the Application

**5.1.2 SOFTWARE REQUIREMENT**

* Platform: Windows 10
* Operating System: Windows XP Professional (64 GB)
* Platform: Visual Studio code, visual studio 2019
* Front-End Tool: HTML, CSS, JavaScript, React.
* Back- End Tool: Node.js, Typescript, C#.
* Scripting Tool: Java script, XML (style Sheet (.xsl))

**5.1.3 IMPLEMENTATION REQUIREMENTS**

The execution stage is noteworthy stages in the improvement of the task, as it gives the last answer for the issues. During this procedure, low-level plans are changed over into language- explicit projects to meet the details set out in the determination of programming particulars. This stage incorporates actual usage of thoughts which have been characterised in the period of examination and plan. The approach and strategies utilised for programming usage must advance reusability, simplicity of support and ought to be notable.

**5.1.4 PROGRAMMING LANGUAGE SELECTION**

The programming language picked to actualise the undertaking is C#. C# is a general-purpose programming language with a multiparadigm approach that encompasses several programming disciplines, such as static typing, imperative, declarative, functional, object-oriented, and component-oriented programming. That approach is what allows C# to be so versatile to the point where you can use it for a lot of different projects.. A portion of the advantages that C# gives, which were critical to picking the equivalent is:

* Fast development time, high scalability and is object-oriented.
* Extensive support for Web Applications
* Platform independent and portability
* Automatic Garbage Collection
* Multi-threaded approach and annotation processing.
* Better Integrity and Interoperability

**5.1.5 CODE CONVENTIONS**

This segment tends to the coding rules that were followed all through the whole venture. It contains the product applications required to finish the venture. Appropriate coding norms ought to be followed since coding of massive undertakings in a predictable style is needed. It makes each part of the code more obvious absent a lot of difficulties. Code shows are significant in light of the fact that they improve coherence in programming, empowering the software engineers to obviously understand code.

**5.1.6 NAMING CONVENTIONS**

Naming conventions help programs in a comprehensible way which makes reading easier. The names given to packages, texts, graphs, and classes must be concise and straightforward in order to be readily understood as their content. The project uses both C# and React JS, and the naming convention followed in the two are slightly divergent from each other.

The standards followed throughout this project are as follows:

* Classes: Names of classes are typically nouns. The first letter of every word is capital according to upper camel casing method.

Example: CaseAssignment.

* Methods: Methods musty be doing words or verbs for better quality. Here too, the method name starts with a non- capital letter and the rest of the name adheres to upper camel casing.

Example: getValue( ).

**5.1.7 COMMENTS**

Comments are a vital part of any programming conventions as it enhances the comprehensibility of the established code. In the project files, owing to the IDE and GUI based environments, grey is the default colour for showing commented texts, so

they are easy to identify. This project contains a variety of comments throughout the entire code to enable easy understanding and handover of the project.

Comments help in better understanding of code. In both c# and React JS, if the comment spans for only one line then its starts with ‘//’ and if the comment spans more than one line then its starts and ends with ‘/\*’ and ‘\*/’ respectively. These comments increase the readability of the code and make it more lucid.

5.2 WORK SCHEDULE

1. January

* Company orientation and onboarding activities.
* Brief about the Travel industry.
* Training on C sharp.

1. February
   * Training on dotnet Core including MVC.
   * Training on React JS and SQL.
2. March
   * Assigned to the project Voyager.
   * Understanding of codebase of the project.
3. April
   * Understanding of all admin functions and integration frameworks.
   * Debugging and working on the errors.
4. May
   * Learn agile and different parts of product development
   * worked on the user stories and added modules to the project.
5. June
   * Fixing issues and other maintenance activities.
   * Started Exploring micro services.
   * Converting our project from monolithic to microservices.
   * Submission of final project report for six months project.

**Chapter - 6**

**Architecture**

6.1 MONOLITHIC ARCHITECTURE

A monolithic architecture is the traditional unified model for the design of a software program. Monolithic, in this context, means "composed all in one piece." According to the Cambridge dictionary, the adjective monolithic also means both too large and unable to be changed.

Monolithic applications are designed to handle multiple related tasks. They’re typically complex applications that encompass several tightly coupled functions.

Monolithic software is designed to be self-contained, wherein the program's components or functions are tightly coupled rather than loosely coupled, like in modular software programs. In a monolithic architecture, each component and its associated components must all be present for code to be executed or compiled and for the software to run.

Monolithic applications are single-tiered, which means multiple components are combined into one large application. Consequently, they tend to have large codebases, which can be cumbersome to manage over time.

Furthermore, if one program component must be updated, other elements may also require rewriting, and the whole application has to be recompiled and tested. The process can be time-consuming and may limit the agility and speed of software development teams. Despite these issues, the approach is still in use because it does offer some advantages. Also, many early applications were developed as monolithic software, so the approach cannot be completely disregarded when those applications are still in use and require updates.

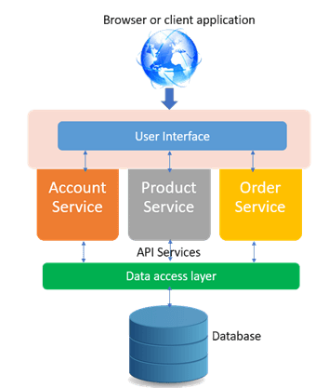


Figure 22. Monolithic Architecture

**6.1.1 Key components of monolithic applications**

Monolithic applications typically consist of multiple components that are interconnected to form one large application. These components may include these features:

* **Authorization**. To authorize a user and allow them to use the application.
* **Presentation.** To handle Hypertext Transfer Protocol requests and respond with Hypertext Markup Language, Extensible Markup Language or JavaScript Object Notation.
* **Business logic**. The underlying business logic that drives the application's functionality and features.
* **Database layer**. Includes the data access objects that access the application’s database.
* **Application integration**. Controls and manages the application’s integration with other services or data sources.

Some applications may also include a notification module to control and send automated email communications to users.

**6.1.2 Benefits of Monolithic Architecture**

* Simple to develop.
* Simple to test. For example you can implement end-to-end testing by simply launching the application and testing the UI with Selenium.
* Simple to deploy. You just have to copy the packaged application to a server.
* Simple to scale horizontally by running multiple copies behind a load balancer.

**6.1.3 Drawbacks of monolithic architecture**

Generally, monolithic architectures suffer from drawbacks that can delay application development and deployment. These drawbacks become especially significant when the product's complexity increases or when the development team grows in size.

The codebase of monolithic applications can be difficult to understand because they may be extensive, which can make it difficult for new developers to modify the code to meet changing business or technical requirements. As requirements evolve or become more complex, it becomes difficult to correctly implement changes without hampering the quality of the code and affecting the overall operation of the application.

Following each update to a monolithic application, developers must compile the entire codebase and redeploy the full application rather than just the part that was updated. This makes continuous or regular deployments difficult, which then affects the application's and team's agility.

The application's size can also increase startup time and add to delays. In some cases, different parts of the application may have conflicting resource requirements. This makes it harder to find the resources required to scale the application.

In addition to limited scalability, reliability is another concern with monolithic software. A bug in any one component can potentially bring down the entire application. Considering the banking application example, suppose there's a memory leak in the user authorization module. This bug can bring the entire application down and make it unavailable to all users.

Finally, by virtue of their size and complexity, monolithic applications are not particularly adaptable to new technologies. A new development framework or language can affect the application as a whole, so adopting it can be both time-consuming and costly. Small organizations or companies on tight budgets may not have the funds or staff available to update the application, so they may end up maintaining the status quo, potentially leaving them unable to take advantage of a new language or framework.

6.2 MICROSERVICES ARCHITECTURE

The idea is to split your application into a set of smaller, interconnected services instead of building a single monolithic application. Each microservice is a small application that has its own hexagonal architecture consisting of business logic along with various adapters. Some microservices would expose a REST, RPC or message-based API and most services consume APIs provided by other services. Other microservices might implement a web UI.

The Microservice architecture pattern significantly impacts the relationship between the application and the database. Instead of sharing a single database schema with other services, each service has its own database schema. On the one hand, this approach is at odds with the idea of an enterprise-wide data model. Also, it often results in duplication of some data. However, having a database schema per service is essential if you want to benefit from microservices, because it ensures loose coupling. Each of the services has its own database. Moreover, a service can use a type of database that is best suited to its needs, the so-called polyglot persistence architecture.

Some APIs are also exposed to the mobile, desktop, web apps. The apps don’t, however, have direct access to the back-end services. Instead, communication is mediated by an intermediary known as an API Gateway. The API Gateway is responsible for tasks such as load balancing, caching, access control, API metering, and monitoring.

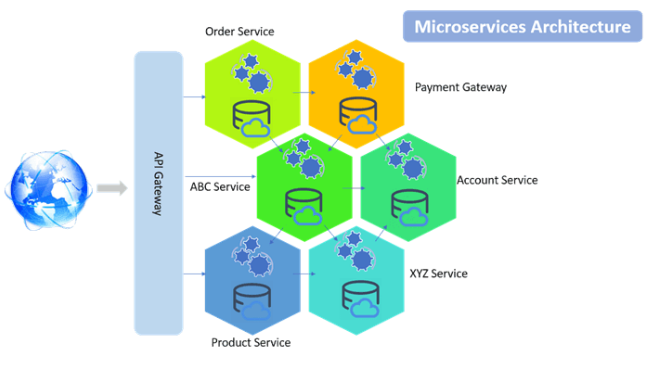


Figure 23. Microservice Architecture

**6.2.1 Key components of microservices architecture:**

* **Clients:**The client apps usually need to consume functionality from more than one microservice. The client needs to handle multiple calls to microservice endpoints if that consumption is performed directly. Client apps need to be updated frequently, making the solution harder to evolve.
* **Identity Providers:** The services are fine-grained and lightweight. The Identity Microservice must allow both user-driven and server-to-server access to identity data. Microservices allow applications to be created using a collection of loosely coupled services.
* **API Gateway:** The API Gateway is responsible for request routing, composition, and protocol translation. It provides each of the application’s clients with a custom API. For most microservices‑based applications, implementation of an API Gateway is very important for a single-entry point into a system.
* **Messaging Formats**: Synchronize and Asynchronized are the 2 types of messages through which they communicate. Every microservice in order to communicate either synchronously or asynchronously with other microservices. “Synchronous – HTTP is a synchronous protocol. The client sends a request and waits for a response from the service. The client code or message sender usually does not wait for a response.
* **Databases**: Microservice owns a private database to capture their data and implement the respective business functionality. Microservices databases are updated through their service API. The services provided by Microservices are carried forward to any remote service which supports inter-process communication for different technology stacks.
* **Static Content**: After the microservices communicate within themselves, they deploy the static content to a cloud-based storage service that can deliver them directly to the clients via Content Delivery Networks (CDNs).
* **Management:** Management feature is a capability that allows operations and business users to configure services in run-time. For load balancers, the approaches to feature flags management in microservices architecture are a complex topic, especially when one business feature spans multiple microservices.
* **Service Discovery:** In a microservices application, the set of running service instances changes dynamically. Instances have dynamically assigned network locations. Consequently, for a client to make a request to service it must use a service‑discovery mechanism. A key part of service discovery is the service registry.

**6.2.2 Benefits of Microservices:**

* **Agility:** Microservices foster an organization of small, independent teams that take ownership of their services. Teams act within a small and well understood context, and are empowered to work more independently and more quickly. This shortens development cycle times. You benefit significantly from the aggregate throughput of the organization.
* **Flexible Scaling:** Microservices allow each service to be independently scaled to meet demand for the application feature it supports. This enables teams to right-size infrastructure needs, accurately measure the cost of a feature, and maintain availability if a service experiences a spike in demand
* **Easy Deployment:** Microservices enable continuous integration and continuous delivery, making it easy to try out new ideas and to roll back if something doesn’t work. The low cost of failure enables experimentation, makes it easier to update code, and accelerates time-to-market for new features.  
  **Technological Freedom:** Microservices architectures don’t follow a “one size fits all” approach. Teams have the freedom to choose the best tool to solve their specific problems. As a consequence, teams building microservices can choose the best tool for each job.
* **Reusable Code:** Dividing software into small, well-defined modules enables teams to use functions for multiple purposes. A service written for a certain function can be used as a building block for another feature. This allows an application to bootstrap off itself, as developers can create new capabilities without writing code from scratch.
* **Resilience:** Service independence increases an application’s resistance to failure. In a monolithic architecture, if a single component fails, it can cause the entire application to fail. With microservices, applications handle total service failure by degrading functionality and not crashing the entire application.

**6.2.3 Communication between Microservices**

**6.2.3.1 Synchronous Communication**

Figure 24. Synchronous Communication

A synchronous microservice is one in which data moves to and from an endpoint of a service in a blocking interaction. A typical example of a synchronous data exchange is an HTTP request/response interaction. Others are gRPC, REST or GraphQL etc.

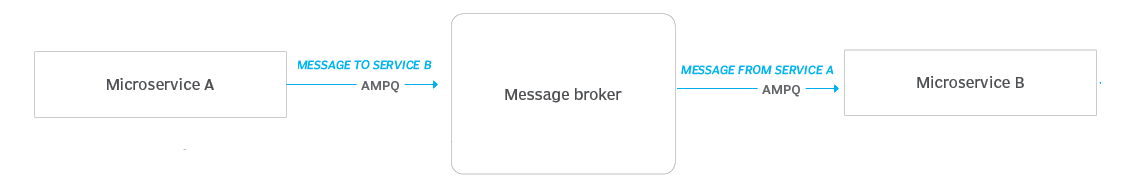
The caller will be locked in interaction and cannot move forward to the next task until the response is received.

The caller might receive the response in a mere millisecond or in a few seconds. (Time gap)

Use cases

1. The operation is a simple query which does not change any state
2. The operation result is needed to move forward in the current process
3. The operation can fail and does not require a complex retry mechanism
4. The operation needs to be synchronous

**6.2.3.1 Asynchronous Communication**

*Figure 25. Asynchronous Communication*

An asynchronous microservice is one in which a request to a service and the subsequent response occurs independently from each other. The intended service receives the message in its own time. The sending service is not locked to the broker. It simply fires and forgets.

The general practice for implementing an asynchronous microservice is to use message broker technology. (Eg. Kafka, RabbitMQ).

One service will publish a message to another service using the message broker. Asynchronous communication can use a protocol such as AMQP to exchange messages via a message broker.

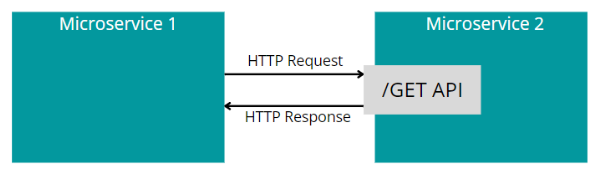
Use Cases

1. The operation involves multiple services reacting to it
2. The operation must be performed while allowing failures & retries
3. The operation takes a lot of time

**6.2.3.1 Different ways of Communication between Microservices**

# RESTful Web API Endpoint Integration

Using REST, services expose resources which are available on dedicated endpoints so that one microservice can call other microservices. We can use different HTTP verbs depending on the action we want to perform. Information is transported using JSON which leads to serialization & deserialization of each request's body.



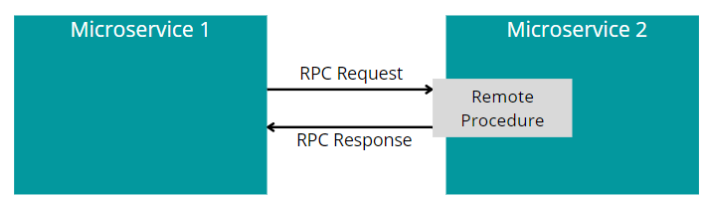
*Figure 26. RESTful Communication*

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * Easy to understand (text protocol) | * Tight coupling between microservices. |
| * Web Infrastructure already built on top of HTTP | * Changing the interface of an endpoint used by other microservices can result in breaking changes. |
| * low latency between calls that are routed from one microservice to another | * Client-side error handling issue. Must store information about requests to some storage that could not be processed due to failure. |
| * No additional components such as message brokers required. | * Operations are difficult to model and streaming is difficult. |
| * High-quality HTTP implementations in every language and great tooling for testing, inspection, and modification. | * No single standard for API contract, thereby requiring the developers to write client libraries |

# RPC / gRPC Integration

gRPC is a framework that allows developers to implement a remote procedure call (RPC) integration pattern for communication between microservices.

Using gRPC, services are defined using Protocol buffers. Clients can be generated from other service's protobuf definitions in many languages to send & receive Protobuf messages which are strongly typed.



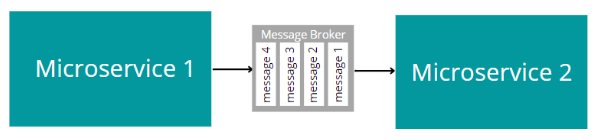
*Figure 27. RPC Communication*

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * better latency and high throughput | * Lack of support for additional content types. Since gRPC depends upon Protobuf, other content types are not supported out-of-the-box as with standard HTTP + REST-based APIs. |
| * High performance. gRPC is much faster than REST+JSON communication. | * Limited Browser Support, since gRPC heavily relies on HTTP/2, you can’t call a gRPC service from a web browser directly, because no modern browsers can access HTTP/2 frames. |
| * Built-in code generation. gRPC has automated code generation in different programming languages. | * Lack of consistent error handling, while gRPC describes the concept of a status code and message, there is no clear and consistent way to properly catch the errors across programming languages |
| * Reduced network usage with Protobuf binary serialization. | * Lack of infrastructure and monitoring support outside of GKE. |
| * Supports client, server, and bi-directional streaming calls. | * Lack of edge caching, While HTTP supports intermediaries for edge caching, gRPC calls use the POST method, which is neither safe nor idempotent |

# Messaging Integration

Message queues are a way for services to produce messages that need to be consumed by other services by acting as a transient buffer. Microservice 1 will only wait for a response from the service bus that the message was successfully submitted to the queue, but will not wait for microservice 2 to finish processing the message.

Once the message has been acknowledged by a consumer, it is removed from the queue, pretty much like a to-do list.



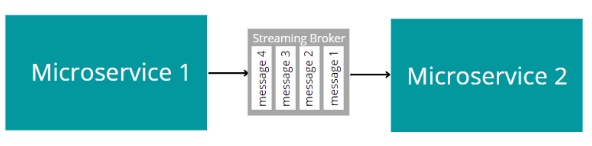
*Figure 28. Message Integration*

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * Messaging pattern allows microservices to communicate *indirectly.* Communication goes through the message bus. | * If the service bus component fails, the application will stop working. |
| * High performance. gRPC is much faster than REST+JSON communication Messaging integration pattern is well suited for asynchronous (fire-and-forget) communication. | * Service bus must be highly available to not be a single point of failure. |
| * Microservices are decoupled from each other. Temporal coupling is avoided (if service A depends on service B, service B being unavailable at the exact moment service A needs it is not an issue) | * Newly created microservices cannot receive old messages already been processed by another microservice. |
| * There is no need to change and recompile the client microservice if it needs to deliver to more consumers later. |  |
| * Service bus provides a better separation of concerns with functionalities like redelivery, dead-letter queue storage, monitoring features, sessions, etc |  |

# Streaming Integration

Streaming broker is a way for multiple services to record that something happened in time & let other services read & process these events. One microservice sends messages to the streaming broker, and one or more other services can subscribe to consume and process them.

Services can write to the same stream of events (topic) & others can read from it, from the beginning or from the time they start listening to new events.



*Figure 29. Streaming Integration*

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * Implemented through a streaming broker (e.g. Kafka, Redis). | * Streaming broker must be highly available. |
| * Architecturally similar to messaging integration. Streaming broker acts the same as a messaging broker. | * If the streaming broker component fails, the application will stop working. |
| * Can also be done through gRPC but services will be tightly coupled. |  |
| * Messages are not removed from the streaming broker after they have been processed. |  |
| * Can get messages from any point in time in the past to be processed again by any other microservice. |  |
| * Stream brokers additionally act as a data store along with message transport. |  |

6.3 OUR APPROACH

At present we are using monolithic architecture as there are many advantages to microservices architecture than monolithic we migrating our project from monolithic to microservices.

**Due to company policy, we are not allowed to share architecture diagrams.**

**Chapter – 7**

**Modules Analysis**

**Due to company policy, and the project being under development and not yet being released to the public, we are not allowed to share specific details (or) screenshots (or) data pertaining to our project.**

7.1 MODULES

Some of the important modules we are working in our project are

1. Account module
2. User module
3. Role module

**Account:** A Voyager account creation is a very specific action and it registers a user as well. The user who creates the Voyager account becomes the admin user of that account.

**User:** Client who is signing up under Voyager (they can be customer or internal users). We will have 2 placements for user management; one is from left navigation which will show users of the service selected and in header from which all the users will be accessible. Header placement will be visible to global admin. User will be govern through email ids.

**Role:** A group defined by level of permissions a user will have. What is the level of access that user will have will be defined under the role. Every role name will have prefix/suffix of service name, ex: RR-Admin, ABC-Partner. Here Admin & Partner are roles and RR & ABC are services.

**Left navigation items for Admin Service**

Coming to the navigation items we have several features we have added to our project as per the requirement which includes some of the basic features like the dashboard, user management, role management...

1. **Services**: It will show all the services which are subscribed under that account. Users can manage the services from there. Subscribe, unsubscribe, increase limit or usage. They can also add new services under that account.
2. **User Management**: This will give access to users who are given access to this service only
3. **Account Management**: This will be for our internal client management
4. **Roles & Permission**: Role management access will be given here
5. **Global Settings**: Global account level settings will be visible here
6. **Usage**: Usage summary of all the subscribed services with the link to manage individual service
7. **Reports & Dashboard**: List of service wise reports and dashboards
8. **Billings**: We will show credit card details which can be filled/edited by admin users. We will also show billing details of each service on a separate card with a feature to download individual invoices and combined invoices.
9. **Support**: Support related details will be given here. Details of relationship manager and FAQs can be provided.

**Chapter – 8**

**Conclusion and Future Works**

8.1 CONCLUSION

“Voyager” is a product which will be helpful for many small Online Travel Agencies and Airlines to manage their users as well as to track their company sales and get a brief idea about their average sales and profits. We even provide different services to the clients based on their requirement they can choose which service best suits their company and use them accordingly. It will be very beneficial to these airlines and online travel agencies as they can't afford and manage to make an enterprise solution and maintain the infrastructure on their own. Here we are solving their problem by providing a product for these agencies so that they can focus on their business and also their overhead expenses will be reduced when a pay as you use model/ subscription model is provided.

8.2 FUTURE WORK

We are migrating from Monolithic Architecture to Microservices Architecture due to the benefits that we can drive out of building our product on it which allows for easy scalability, availability and also helps us in following the necessary guidelines and policy with respect to the individual tenants which are applicable according to the laws of their countries.

We will be even working on many other services in near future and will try to acquire as many clients as possible

8.3 REFERENCES

