BidNexus

(Platform for digitizing auction)

Project report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology

in

Computer Science and Engineering

By

KUNWAR PRATAP SINGH(191272)

Under the supervision of

Mr. ARVIND KUMAR

to

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology Waknaghat, Solan-173234,

Himachal Pradesh

CERTIFICATE

I hereby declare that the work presented in this report entitled **“BidNexus** (Platform for digitizing auction)**”** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from Feb 2023 to May 2023 under the supervision of **Mr. Arvind Kumar**, Assistant Professor (Grade-II), Department of CSE Jaypee University of Information Technology, Wakhnaghat.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

KUNWAR PRATAP SINGH 191272

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Mr. Arvind Kumar

Assistant Professor (Grade-II)

Department of Computer Science and Engineering

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ....

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | [illegible]

Name:______________________________**Department:**____________________**Enrolment No** ___________

Contact No._______________________________**E-mail.**____

Name of the Supervisor: ____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): ______________________________

____

____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at (%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	**Excluded**	**Similarity Index (%)**	**Generated Plagiarism Report Details** (Title, Abstract & Chapters)	
	All Preliminary Pages Bibliography/Images/Quotes 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature **Librarian**

....
....

ACKNOWLEDGEMENT

First, I express my heartiest thanks and gratefulness to Almighty God for His divine blessing to make it possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to Supervisor **Mr. Arvind Kumar, Assistant Professor (Grade-II)**, Department of CSE Jaypee University of Information Technology, Wakhnaghat. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, and reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to **Mr. Arvind Kumar,** Department of CSE, for his kind help in finishing my project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

Kunwar Pratap Singh 191272

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The shift of the market from offline to online has been a great boon for Information Technology and business. There are different websites that are based on business related to shipping, buying and selling of products, buying and selling of used products. Some of the sites are Flipkart, Amazon and Olx. This has covered a lot of business processes that were only taking place in offline mode. But still there is a gap that lies in the auction process which is still done offline, there are ways to do it online but still no such site is that popular among vendors and clients. This project lies in creating a platform where people can request items that they want and their specifications and the bidders can bid on the items requested by the requester.

The project explores the idea of a demand-driven online marketplace that fuses conventional bidding and auction events with online shopping. Users of this market are able to take part in online auctions for a variety of things, from commonplace items like watches and cellphones to ancient assets. This marketplace offers consumers a demanding and safe environment in which to put bids and make purchases, in contrast to other classified ad websites like Olx. A user-friendly online application, secure real-time payment processing, scalable and effective system architecture, deployment of Google Cloud containers, and handling of race situations are all necessary for the system to function. In order to create a safe and secure backend, it also emphasizes the significance of adhering to the DRY and SOLID principles, appropriate coding techniques, and creating unit test cases.

Chapter 01: INTRODUCTION

1.1 Introduction

In recent years, online auctions have grown in popularity as a means of buying and selling products. Online auctions have become a simple tool for buyers to locate one-of-a-kind things and for sellers to reach a larger audience as online shopping grows in popularity. We will examine the world of online auctions in this introduction and discover why consumers and sellers find them to be so alluring. Online auctions also have the advantage of frequently being more effective than conventional auctions. There is no requirement for bidders and sellers to physically attend an auction because everything is conducted online. This may help customers save time and money, and it also makes it simpler for them to take part in auctions from any location.

Although they may be a terrific method to purchase and sell items, internet auctions can have certain hazards. Fraud is one of the most serious hazards. It's possible for buyers to not receive the item they ordered or for the item to not match the description. Sellers may also be at risk of fraud, as buyers may try to scam them out of their items or refuse to pay for items they have won. To reduce the risk of fraud, it is important to only buy and sell items on reputable auction websites. Buyers should also read the item description thoroughly and ask the seller any questions they may have before placing a bid.
Sellers should also be sure to adequately describe their things and give clear images.

E-commerce and online classifieds are two potent instruments that have completely changed how we purchase and sell products and services. They each have distinct functions, but they both aim to link consumers and sellers in an online marketplace.

Users can submit ads for products and services on websites called online classifieds. Depending on the platform, these adverts may be put for free or at a cost. Because users may choose a specific geographic region to target, online classifieds can be a wonderful method to sell products locally. Craigslist, Facebook Marketplace, and Gumtree are a few well-known online classifieds websites.

1.2 Problem Statement

With the growing digitalisation and demand and supply chain flow there needs to be a solution that can digitally solve the problem of selling and buying online as well as auction along with all the safety that the user wants in their online experience and with cutting edge technology.

I. To develop a full stack web application deployed on google cloud using **Google Cloud Platform** as the backend technology along with **Flask and React with TypeScript** as frontend technology.

II. **Demand driven marketplace** for household items(pre-owned/new/refurbished everything). Buyer will post their requirement with a budget, and sellers can bid for fulfilling that requirement.

III. **User**: Any user can sign up on the platform as a buyer or a seller. A user can invite more users on the platform(invite via csv file also supported) but max. 25 users a day only. Each user has to verify their email address during the signup process, without which they can't access any features.

IV. **Item**: A user can post their requirement as a form of Item.
   1. Name(Required)
   2. Description
   3. Requester
   4. Date & Time (in increments of 1 hour) for closure cannot be earlier than 48 hours (Live bidding will be done in the last 1 hour)
   5. Item status(pre-owned/new/refurbished)
   6. Not old than x months/years
   7. No. of items
   8. Max. price

V. **Item Bid**: Any user can bid for that item requirement by providing product details(max. 6 photos and other details). Item requesters can not view a bid price until a bid is disclosed. The bidders can add all the information with their bid amount.

VI. **Eligible bidder**: Before a bid is disclosed, the item requester can reject a seller bid after viewing product specification/details with a reason for rejection.

VII. **Bid disclosure**: An eligible bidder/seller with the minimum sell price will be selected as the bid winner.

VIII. **Transactions**:

A. Each item requester has to pay 1% or 1 cent(which one is max.) of the item price to place an item request.

B. Each seller has to pay 1% or 1 cent(which one is max.) of their bid amount to place a bid.

C. Use Stripe for all payments.

IX. **Live bidding**:

A. The buyer can see how many bids are being placed in real time when the bidding session starts, and can see the details of those bids.

B. Bidders get the option to place a bid or modify their existing bids, and also see the lowest bid amount currently present.

C. Once the bidding session closes, the winner will be decided.

1.3 Objectives

The online demand driven marketplace is the concept about bringing the traditional bidding and auction events and the e-commerce like digital idea together at a place where the auction can take place online in which several users can participate but this time not only for antique or costly assets but also for simple things like smartphones or watches. Like the website Olx where users can sell their items and buyer can buy but Olx does not charge anything or participate within the selling process, this product brings the concept of taking these classified ads further more than Olx like website to give a more challenging and safe environment where users can place bid on the item and will be charged respectively.

I. Users can place item requests and bid using a web application that has to be user friendly and fast.
II. Make payment transactions safe and consistent in real time.
III. Make scalable and efficient system design to handle several requests in parallel.
IV. Deploying Google Cloud containers in an efficient way and running backend applications in them.
V. Handling race conditions and making a safe full proof backend.
VI. Write unit test cases of the application.
VII. Following DRY and SOLID principles and good coding practices all over the codebase.

1.4 Methodology

Planning:
In planning all the user requirements were laid down first. After that the technical requirements as well. Both were matched to see if they can fulfil each requirement. After this a **work breakdown structure** has been made in an excel sheet that estimated around a month in which all the modules required to complete the project were broken down into smaller components with their respective estimated time. Each module included the detailed components that have to be made, testing that has to be done and review fixes that need to be fixed after the code review.

Analysis:
Features were implemented serially in the order they were in the WBS sheet. For each feature a complete analysis is done on the tech stack required with the requirements. For backend features the analysis included finding and exploring what cloud tools to use, i.e. whether to use Cloud Run or Cloud Function, similarly for each feature various approaches were analysed varying from asynchronous to synchronous behaviour.

Design:
Firstly the architecture is decided in a make as you explore modules manner, in which for the current feature the best services get laid down and used which is then added to the architecture. After that the design for database is laid down since we are using NoSQL based database i.e. Google Firestore, all the docs and their subcollections were laid down and discussed with the mentor. Further design included architecture level design, feature breakdown, UI design and code level design along with the algorithm/procedure to be used to

Implementation
Once the implementation started, the project was set up with the GitHub repository for each feature. For the frontend code base, the file structure was laid in a React practice manner where each functionality is broken down into components, containers and services. For the backend code base, the file structure was laid in modules and class services, helper methods and the requirements for them were installed. After the project

setup, firstly low level design of any functionality was laid down on the paper then only code was done.

Maintenance and Support:

Since this was a project not a product so the maintenance and support only includes getting the code reviewed by the mentors. Writing unit test cases and integration test cases for each module in the project using various testing libraries. To do end to end testing using Postman for APIs.

Chapter 02: LITERATURE REVIEW

The paper[1] focuses on a quick introduction to cloud computing and the three primary cloud platform designs follow in the paper. Additionally, it has compared these platforms based on a variety of factors, including cost, specifications, support and administration, database support, machine learning and artificial intelligence support, storage, deployment tools, networking, and security. The results of this study will assist cloud users in choosing the appropriate cloud platform for their unique needs.

A. Private Cloud
   Private clouds are primarily used by independent businesses and are ideal for needs involving high security.
B. Public Cloud
   Public clouds are ideal for expanding businesses and are frequently used in networks. It can holster the needs for both small scale & large-scale businesses.
C. Community Cloud
   Community cloud is utilised by collaborative organisations to exchange data effectively. These are very helpful for businesses like banks.
D. Hybrid Cloud
   Hybrid clouds are systems that combine both private and public clouds. It is extremely beneficial for organisations whose requirements include both. It helps them to privately interact within the organisation & publicly interact with customers using public cloud

General:

- Of the three cloud platforms, Microsoft Azure is the most widely used.
- Google Cloud Platform's Cloud9 offers native IDE support.
- Of the three, Amazon EC2 is the oldest and has a firm grasp of the IaaS service model.

Database and virtualization:

- Google cloud platform offers the most database possibilities, whereas Azure offers the fewest.

- The widest range of virtualization solutions is offered by Amazon EC2.

Pricing:

- Each of the three has a unique pricing structure based on the consumption of the service.

Specifications:

- Amazon EC2 has the most pre-configured operating systems, whereas Microsoft Azure has the most support for machine learning frameworks.

The three cloud platforms that were previously evaluated each offer advantages that make them useful in their own right. Even while Amazon EC2 is the most established and supports the most number of pre-configured operating systems, it falls behind in terms of accessibility and support.
Similar to the Google Cloud Platform, which supports the most databases and has a large collection of built-in libraries but lacks SDK support and uses a pay-to-help approach where the length of time it takes to get assistance varies depending on the level of service used. Although the Microsoft Azure platform has the broadest reach of the three, its database support is significantly lacking.
Thus, we draw the conclusion that the choice of cloud platform is strongly influenced by the user's requirements, which vary from user to user.

From website[2] following conclusions are made:

The following are some of the main distinctions between AWS and GCP:

Services: Compute, storage, networking, databases, and other cloud services are all available through both AWS and GCP. However, AWS offers a wider range of services, whereas GCP focuses more on a smaller number of services.

Pricing: When selecting a cloud platform, price is an important factor. Although both AWS and GCP have numerous pricing options, their systems can be convoluted and challenging to comprehend. In general, GCP is more affordable for large-scale projects whereas AWS is more expensive for modest workloads.

User Interface: Each platform has a unique user interface. While GCP's UI is simpler and more streamlined than AWS', the latter features a console that is more feature-rich and complicated.

Security is built into both AWS and GCP, however AWS has a longer history and more certifications in this area. More security capabilities are available through AWS, including Virtual Private Cloud (VPC) and Identity and Access Management (IAM).

Help: The help and support for the two systems are comparable and they provide varying degrees of support, which varies from free to paid support. The only difference is that Google Cloud Platform provides a more focused and individualistic kind of help but AWS does it on a large team basis and with greater resources.

CHAPTER 03: SYSTEM DEVELOPMENT

3.1 Frameworks and Libraries Used

3.1.1 Python

Python is a simple typed yet powerful language which lies in high-level programming and it has experienced a large amount of growth in many fields not only AI. The popular feature covers the range of web development, machine learning, big data and it is highly adaptable, has a large number of libraries and support.

Python is popular because of its readability and the way it simplifies coding. It is also a great choice for beginners as it has simple syntax. Python developers and programmers all over the world provide a great community support with a variety of libraries that simplify complicated processes and shorten the time for development focusing more on core logic.

Python is highly adaptable as it can be used in different projects like machine learning, automation, web development and scientific research computing. It is also great for cross-platform programming as it is supported by various OS, including Windows, Linux and MaxOS

Python is not type strict programming but still provides support for typings using its library. Python supports both procedural and object oriented programming paradigms.
This is because it has a large number of libraries and tools—like NumPy, Pandas, and Matplotlib—that offer strong capabilities for handling data and displaying the findings.

With a sizable ecosystem of third-party libraries and tools, Python is also very expandable. Web frameworks like Flask and Django, database libraries like SQLAlchemy, and machine learning libraries like TensorFlow and PyTorch are some examples of these libraries.

3.1.2 Flask

A well-liked and slim web framework for Python called Flask offers a versatile and user-friendly platform for creating online applications. As a result of Flask's well-known simplicity, small- to medium-sized projects or the creation of microservices are appropriate applications for it.

Flask gives developers the resources they need to easily create and deploy web applications by building on top of the Werkzeug WSGI toolkit and the Jinja2 templating engine. One of Flask's primary characteristics is its modular architecture, which frees developers from being forced to use a full-stack framework by allowing them to select only the components they require.

With support for routing, views, and templates, Flask offers a straightforward and understandable API for processing HTTP requests and answers. It also makes it simple to develop secure online apps by supporting cookies, sessions, and authentication.

Flask offers a large variety of plugins and extensions that may be used to enhance the framework's capabilities. From email support to OAuth authentication to database connectivity, these extensions cover it all.

Flask offers an object that represents the HTTP request (Request) and an object that represents the HTTP response (Response). These objects offer a straightforward and standardised approach to handle HTTP requests and replies.

The application context in Flask is the environment in which the Flask application is running. It includes crucial application-specific data, including configuration options, database connections, and other resources that the programme requires.

The application context and the request context are the two different categories of context in Flask. When a Flask application first launches, the application context is generated, and it remains active for the duration of the programme. However, the request context is only formed once the request has been completed; it is not established for every request that the application gets.

3.1.3 Marshmallow

In order to serialise and deserialize complicated data types between Python objects and JSON or other serialisation formats, developers can use the Python library marshmallow. Data conversion between a web API and a client application is frequently used in online applications to make it simpler to send and handle data across various platforms.

The capability of Marshmallow to design data schemas in Python using classes is one of its primary advantages. These schemas may be used to serialise and verify data, ensuring that it is in the right format and is simple to transfer to other systems and utilise.

Additionally, Marshmallow offers sophisticated features like nested schemas, enabling programmers to manage intricate data structures with several levels of layering with ease. This may be especially helpful when data has to be sent across several systems or APIs.

The integration of Marshmallow with well-known web frameworks like Flask and Django is another benefit. It offers a straightforward and understandable method for defining data schemas for these frameworks, which makes it simple to include Marshmallow into already-existing online applications.

Marshmallow is a strong and adaptable module that may make data serialisation and deserialization in Python applications simpler overall. Marshmallow enables developers to speed up data handling and transfer, resulting in time and labour savings during the development process.

3.1.4 React

A well-known open-source JavaScript library for creating user interfaces is called React. It was created by Facebook and is extensively utilised by organisations and programmers to create sophisticated online apps.

Because React is a component-based framework, the user interface is divided into manageable, reusable components. It is simpler to manage and maintain a big codebase since these components may be integrated and reused across the programme.

React's virtual DOM, which enables fast and speedy user interface updates, is one of its core characteristics. React updates only the necessary portions of the DOM when a component's state changes rather than re-rendering the entire page. This increases the responsiveness of the user interface and enhances the application's overall performance.

Additionally, React has a sizable and vibrant community that has created several outside libraries and tools to expand its capability. These consist of debugging and profiling tools for React apps, as well as libraries for testing, state management, and routing.

React also has the benefit of being compatible with other frameworks and technologies, like Next.js for server-side rendering and Redux for state management. React is hence a very flexible and adaptable library that can be tailored to fit a variety of use cases and project needs.

React is an all-around strong and adaptable toolkit that has completely changed how online applications are created and maintained. It is a well-liked option for software professionals and organisations all around the world thanks to its component-based design, virtual DOM, and active community.

3.1.5 TypeScript

JavaScript's rigorous syntactical superset, TypeScript, is a programming language. It enhances JavaScript by adding optional static typing, which can assist programmers in identifying flaws early in the development cycle and enhance the overall dependability and maintainability of their code.

Support for type annotations is one of TypeScript's core features. Developers can declare the data types of variables, function arguments, and return types in their code by using type annotations. The TypeScript compiler can check the code for type mismatches and other issues, making it simpler to find errors and bugs early in the development process.

This example code shows how to give types to variables in TypeScript

```
class UserAccount {
  name: string;
  age: number;
  email: string;

  constructor(name: string, age: number, email: string) {
    this.name = name;
    this.age = age;
    this.email = email;
  }

  describe(): string {
      return `Name: ${this.name}, Age: ${this.age}, Email: 
${this.email}`;
  }
}
```

3.1.6 Client Server architecture

Today, a traditional Web application comprises both client-side and server-side components. A client-side is essentially a document displayed to a user that is an HTML page filled with some data. Thanks to JavaScript scripts, a user may interact with the content in the browser. These scripts are included in the HTML code of the document. Scripts enable applications to respond interactively to a variety of events, including keyboard inputs, mouse movements, and user clicks. Additionally, the document may be changed directly in the browser.

Each document node, such as a heading, paragraph, or text included within an HTML tag, is processed into a Document Object Model (DOM), an object representation of an HTML page that an embedded script may alter. A script that handles a request is on the server-side. The server side may carry out authentication and authorization, store and retrieve data from a database, and carry out any operations that standard software can carry out, depending on the characteristics of an application.

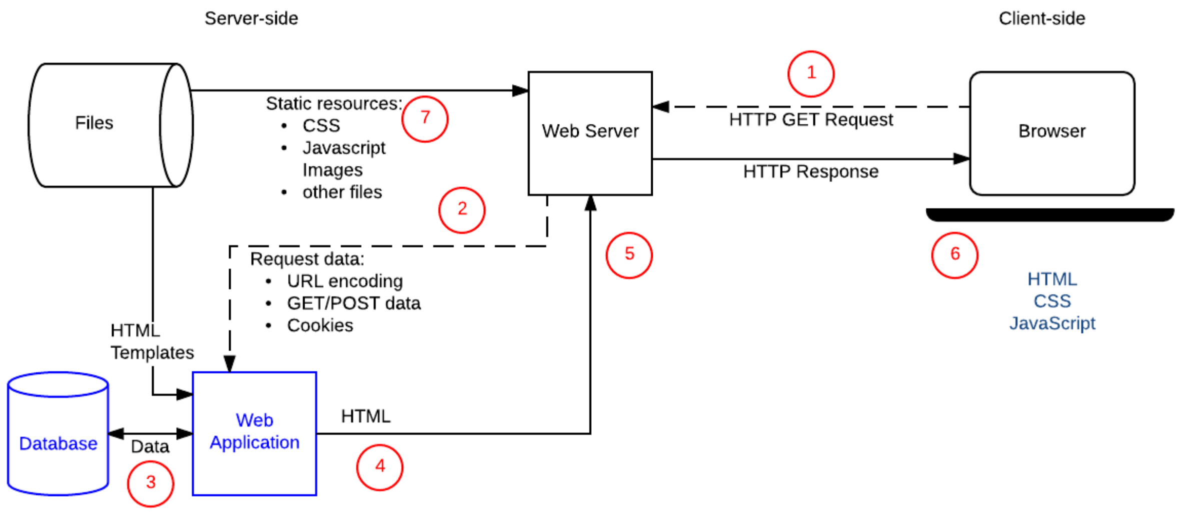


Figure 1. Showing basic client server architecture

3.1.7 Serverless architecture

A cloud-native development methodology called "serverless" enables developers to create and execute apps without having to worry about managing servers. It is an additional abstraction layer on top of the PaaS architecture, where the CSP is responsible for scaling the server infrastructure in addition to resource supply and software maintenance. Unlike the IaaS cloud computing architecture, serverless infrastructure only runs when the application is in use, such as when an HTTP request is received. Scaling the infrastructure when the application is under a heavier demand is the CSP's duty.

Cloud computing:

(IaaS) Infrastructure as a Service:

This concept makes use of the cloud to supply virtualized computer resources including CPU, RAM, OS, and application software. It may at any time be dynamically released, scaled, and provisioned for the customer's needs. Access to enterprise-grade IT infrastructure and resources is one of the primary advantages. Amazon Web Services (AWS), Google Compute Engine (GCE), and Microsoft Azure are a few examples of IaaS.

(PaaS) Platform as a Service:

A more sophisticated cloud computing solution is PaaS. The OS and other software required to execute clients' applications under this model are supplied and maintained by CSP. For access to the platforms where they may host their apps, customers must pay. The responsibility for maintaining and updating the programme falls to CSP. Customers are freed from having to own hardware or software in this way. Examples include Heroku, Google App Engine, and AWS Elastic Beanstalk.

(SaaS) Software as a service:

According to this approach, CSP creates, manages, and operates application software. Customers may utilise its services using a browser on any device since it offers access to the application through a web-based interface. The benefit of the concept is that the consumer does not need to purchase a licence, upgrade the programme, or maintain it. The SaaS services are scalable, effective, and simple to configure. Gmail, Dropbox, and Zoom are a few examples of SaaS products.

BaaS (Backend as a Service):

Access to third-party services including hosting, push alerts, cloud-based databases, and authentication services is made possible via BaaS. Developers just utilise the public API to create applications; they are not actually aware of how the service operates within.

FaaS (Function as a Service):

FaaS allows the developer a lot more control over the application's functionality than BaaS does. Functions that will execute in containers completely controlled by a cloud provider are created by developers. Because FaaS is an event-driven cloud computing platform, it runs as needed.

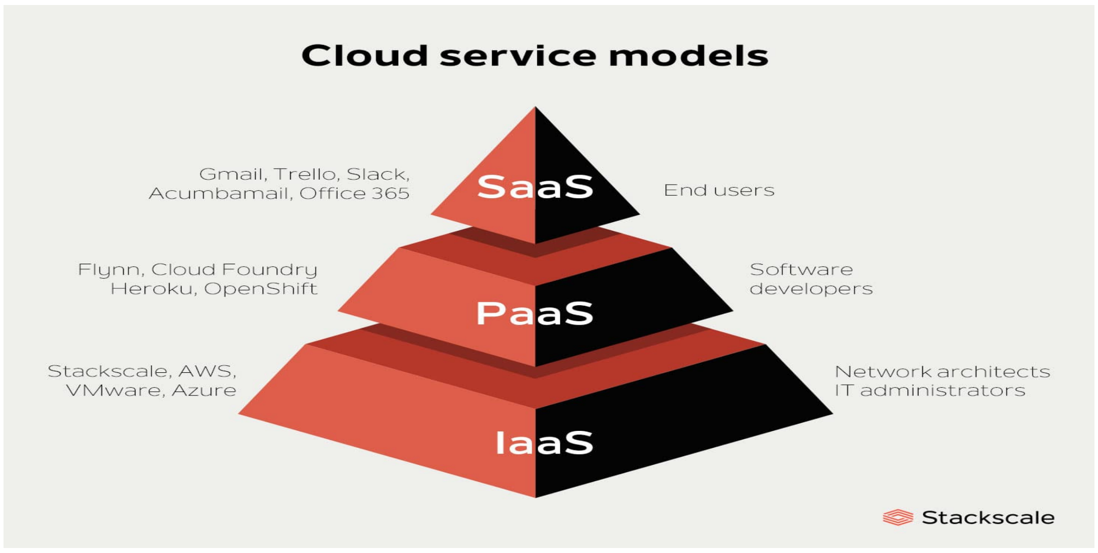

Cloud service models
Gmail, Trello, Slack, Acumbamail, Office 365
SaaS
End users
Flynn, Cloud Foundry Heroku, OpenShift
PaaS
Software developers
Stackscale, AWS, VMware, Azure
IaaS
Network architects IT administrators
Stackscale

Figure 2. Cloud service models

3.1.8 Google Cloud Platform

A group of cloud computing services are offered by Google under the name Google Cloud Platform (GCP). It is constructed on the same platform that Google utilises to power its own SaaS 2.4 products, including Google Mail, Google Docs, Google Drive, and others. The IaaS, PaaS, and serverless computing options offered by Google Cloud Platform.

Resources are allocated according to each project. Every account on GCP has the ability to manage several projects. Users may add editors and provide them precise, pertinent rights, such as project resource control permissions or access to the VM's billing. Additionally, it enables the establishment of service accounts, a unique account type utilised by programmes as opposed to humans. Its permissions can be set up similarly as well.

Nearly every Google Cloud product is restricted to a certain area and zone within that region. The area is known as the "regional failure domain." Zonal outages may affect some or all of the resources in the zone since it represents an underlying structure of physical resources. Where to place services can be decided by the developer. The location affects the cost, latency, and availability of the service. The Europe area was selected in the effort to have a low latency there.

Google Cloud Run:

A GCP service for executing containerized apps is Google Cloud Run. It offers a practical Web user interface for quickly setting up and deploying the container images.

Google Artifact Registry:

An image of a container must be uploaded to a registry before it can be used for deployment. Google provides the Artifact Registry service, which allows the creation of repositories for different artifacts and using them inside GCP infrastructure. Among available Artifact repository formats are Docker containers, Maven artifacts, npm, and Python packages.

3.1.9 Firebase

Firebase is an app development platform built by Google. It offers tools like serverless cloud functions, machine learning, non-relational databases, file storage, hosting, and authentication.

CLI for Firebase

Tools are available for managing, setting up, and deploying Firebase projects using the command-line interface. Additionally, it adds the capability of downloading and using emulators for virtually the whole Firebase architecture, which includes services like Firestore, Storage, Realtime Database, Authentication, Functions, and more.

Authentication

Back-end services for user authentication are offered by Firebase Authentication. The developer side of authentication requires little work. It is sufficient to utilise the handy SDK offered by Firebase. A password, phone number, or several well-known identity providers like Google, Facebook, and Twitter can all be used to identify users. Additionally, Firebase Authentication interacts without difficulty with other Firebase services.

Datastore Firestore

A real-time, NoSQL 2.2 cloud database is called Firestore Database. In contrast to traditional relational databases, the data in this database is stored as documents rather than columns and rows. Documents have a structure akin to the JavaScript Object Notation (JSON) format with a few modifications, such as the map, timestamp, geopoint, and reference data types in the case of Firestore Database. Collections are used to organise the Firestore Database's top-level data. The collection itself consists of documents rather than actual data. Fields that correspond to values are present in documents.

Values may be nested collections or common data types. The adaptable querying technology offered by SDK is used to get data from the database.

Storage

Based on Google Cloud Storage, Firebase Storage is a cloud storage service. It is an object storage solution that allows for safe cloud file storage. Storage organises and restricts access to data using buckets. Files and folders can be stored in a bucket, but not other buckets. Since Firebase Storage files are immutable, uploaded files cannot be changed afterwards but can be changed instead. It leverages Google Cloud Storage underneath, making the buckets available from both services.
Data objects are distinct units of information that come with user-specified object metadata.
A collection of key-value pairs makes up metadata.

3.1.10 unittest

A testing framework that offers a selection of tools for creating and executing tests is the unittest library for Python. It is a popular option for Python developers since it is integrated into the Python standard library and offers a variety of tools for designing and executing unit tests.

Developers may define test cases as classes with unittest, and each test can be thought of as a method on the class. The predicted results and actual results can be compared using a variety of assertion methods using these approaches. This makes it simple to test specific methods, classes, and modules as well as to confirm the functionality of each part of a Python programme.

In addition, unittest offers a number of other functionalities, such as:

Test discovery: Unittest makes it simple to run tests throughout a whole project by automatically finding and executing tests based on naming conventions.

Test fixtures provide context for test cases to run and unittest also support their use. This helps in creating shared testing environments along with test data.

Command line tools: Unittest supports command-line tool that helps in running tests directly from the command line which helps a lot in continuous integration systems to be specific.

The major functions delivered by unittest library are as follows:

- unittest.TestCase: It is a base class for testing. We can subclass another test class using unittest.TestCase and add test methods to it, we can make our own test case classes for testing modules.
- TestCase.setUp(): This is a function that unittest library provides and is invoked before each test method. This is basically a set up for running tests and provides any resources that are needed before each test case. This helps in making code less redundant.
- TestCase.tearDown(): This is a method that runs after each test case has finished executing. This removes the resources and cleans up them for any other test case that is to be run in future.
- TestCase.assertEqual(first, second): A method that asserts that first and second are equal. If they are not equal, a warning message will appear.

3.1.11 jest

Popular testing software for JavaScript apps is called Jest. A software developer may find the following important points useful:

Jest is made to be simple to set up and operate. It can be used to test various JavaScript application types and comes with built-in support for testing React apps.

Snapshot testing, one of the capabilities of Jest, enables you to record the output of a component or function and compare it to a previously recorded snapshot. This can assist you in identifying regressions and confirming that your code is operating as intended.

You may mimic the modules and functions in your application using Jest's strong mocking features. For the purpose of testing isolated components or functions, this might be helpful.

Jest contains capabilities like async/await and the capacity to wait for promises to resolve that facilitate asynchronous testing.

3.1.12 Bootstrap

Bootstrap provides a set of pre-built UI components that you can use to quickly create common UI elements like forms, buttons, and navigation menus.

Due to the adaptable architecture of Bootstrap, your application will automatically adjust to various screen sizes and device kinds. This can help you create mobile-friendly applications more quickly and easily.

There's a grid system too in Bootstrap that helps to design responsive layouts for web applications. Grid system is built on a 12-column layout. The size and placement of the content is flexible on various screen sizes.

With a large selection of configuration options and themes available, Bootstrap is extremely configurable. Using variables and mixins, you may alter the way your application looks and feels, or you can utilize a premade theme to get going right away.

Bootstrap has a sizable user and development community and is well-documented. This implies that you may seek assistance with any problems you run across and discover answers to your concerns with ease.

3.2 Google Cloud Platform

3.2.1 Firebase

Firebase is a Backend-as-a-Service (Baas). It provides developers with a variety of tools and services to help them develop quality apps, grow their user base, and earn profit. It is built on Google's infrastructure.

Following are the products that firebase provides to us:

Realtime Database: The Firebase Realtime Database is a NoSQL database that runs in the cloud and maintains data at a blistering millisecond pace. It may be compared to a large JSON file in the simplest terms.

Cloud Firestore: A NoSQL document database, the cloud Firestore offers capabilities including worldwide syncing, querying, and storage through the application. In the form of objects, sometimes referred to as Documents, it holds data. It can store any type of data, including texts, binary data, and even JSON trees, and has a key-value pair.

Authentication: To allow users to access your app, the Firebase Authentication service offers simple-to-use UI frameworks and SDKs. The amount of labour and time needed to create and maintain the user authentication service is decreased. Even complicated manual chores like merging accounts are handled by it.

Remote Configuration: The service for remote configuration aids in instantly publishing updates to the user. The modifications might involve anything from modifying UI elements to altering how the programmes behave. These are frequently utilised for posting seasonal offers and contents to applications with short lifespans.

Firebase Cloud Messaging (FCM): The FCM service establishes a link between the application's end users and the server so that messages and alerts may be received and sent between them. These connectors work well and use little power.

3.2.2 Firestore

A document-oriented, NoSQL database is Cloud Firestore. There aren't any tables or rows like in a SQL database. Data is instead kept in documents that are arranged into collections.

A collection of key-value pairs are present in every document. It is best to save huge quantities of tiny documents in the Cloud Firestore.

Collections must be used to hold all papers. Subcollections and nested objects, both of which may include simple objects like lists or sophisticated ones like lists, may both be found in documents.

In Cloud Firestore, collections and documents are produced automatically. Simply assign information to a set of documents. Cloud Firestore produces the collection or document if one or both are missing.

Documents

The document serves as the storage unit in Cloud Firestore. A document is a compact record with fields that correspond to values. A name is used to identify each document.

A user's profile document may seem something like this:

```
wiejewewo02i2              → document id
{
        first : "Ada"      → fields
        last : "Lovelace"
        born : 1815
}
```

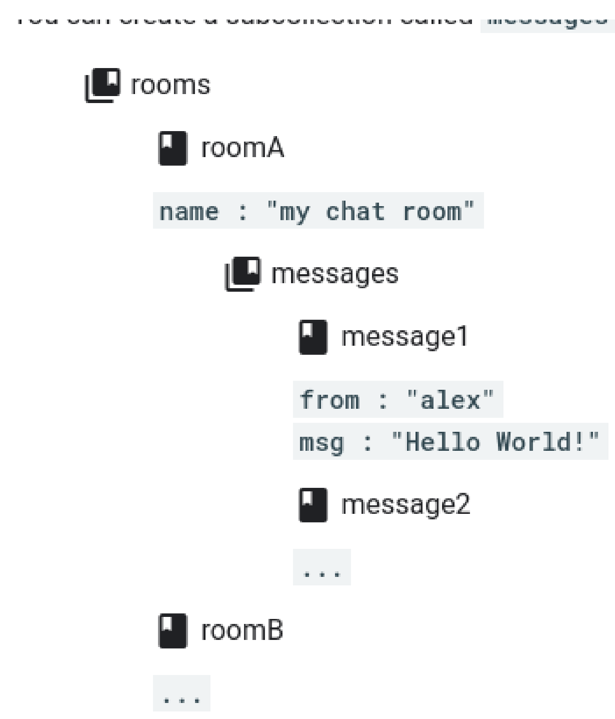
Collections

Documents live in collections, which are simply containers for documents. For example, you could have a users collection to contain your various users, each represented by a document

```
Users                                   → Collection
        mnwniqaq922jne92                → document id of doc 1
        {
                first : "Ada"
                last : "Lovelace"
                born : 1815
        }

        m2nwn223iqaq922jne92            → document id of doc 2
        {
                first : "Robert"
                last : "William"
                born : 1812
        }
```

Subcollections

The best way to store messages in this scenario is by using subcollections. A subcollection is a collection associated with a specific document.



```
rooms
    roomA
    name : "my chat room"
        messages
            message1
            from : "alex"
            msg : "Hello World!"
            message2
            ...
    roomB
    ...
```

3.2.3 Cloud Run

Developers may execute their code in a containerized environment using Cloud execute and Cloud Run Service, two cloud computing services provided by Google Cloud Platform, without having to worry about infrastructure management.

Developers may install and operate stateless containers in a controlled environment using the fully managed serverless computing platform known as Cloud Run. Developers can quickly and simply create and deploy containerized apps using Cloud Run, and they only have to pay for the resources they really use. Developers are free to concentrate on creating and operating their apps since Cloud Run abstracts away the underlying infrastructure.

On the other side, Cloud Run Service is a more developed version of Cloud Run that offers further features and capabilities. In order to construct scalable and highly available applications, Cloud Run Service enables developers to design fully managed microservices that can be automatically deployed and scaled. With Cloud Run Service, developers may benefit from sophisticated capabilities like traffic splitting and canary deployments, as well as automated scaling, monitoring, and logging.

The well-known open-source container orchestration platform Kubernetes serves as the foundation for both Cloud Run and Cloud Run Service. This indicates that developers don't require specialized training or expertise to benefit from Kubernetes' strength and adaptability.

The flexibility and mobility of Cloud Run and Cloud Run Service are two of their main advantages. Developers may simply migrate their programmes between multiple cloud platforms and even operate them on-premises if they so desire by deploying them to Cloud operate or Cloud Run Service using Docker containers.

All things considered, Cloud Run and Cloud Run Service are strong and adaptable cloud computing services that can assist developers in creating and running containerized applications fast and efficiently. They are the perfect option for developers who want to concentrate on creating excellent apps rather than worrying about infrastructure maintenance because of their inherent scalability, dependability, and ease of use.

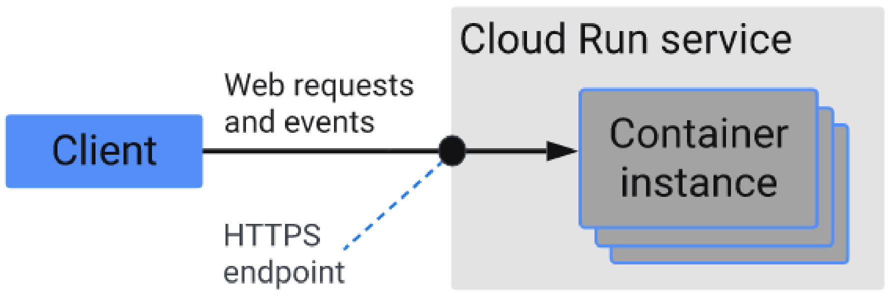


Figure 3. Cloud run architecture

3.2.4 Cloud Function vs Cloud Run

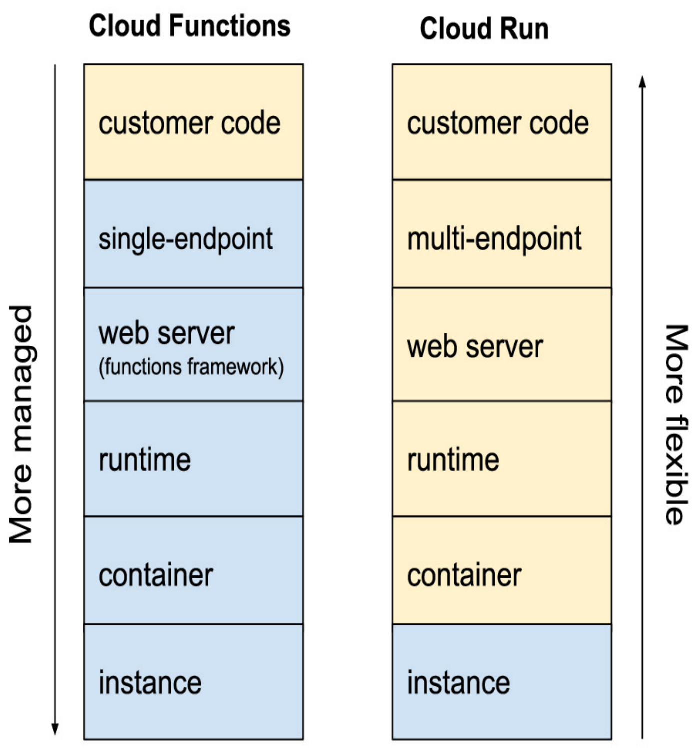


Figure 4. Cloud functions vs Cloud run

You are only in charge of the code thanks to Cloud Functions, which enables integrating your platform simple to construct and straightforward to manage. Anyone on your team who knows how to code can come up with a solution without packaging the code. There are additionally seven widely used languages available. Data scientists, for instance, don't need to be experts in infrastructure to execute a Python script on the cloud.

By making each function its own distinct component and insulating it from having a direct influence on other workloads, Cloud Functions maintains productivity high and operations low. It's unlikely that updates and changes to one function will have an effect on another.

3.2.5 Cloud Task

The Google Cloud Platform's Cloud Tasks service enables developers to build and carry out tasks that do background processing in a dependable and adaptable way. It is fully managed, scalable, and asynchronous. Developers can construct and manage distributed applications that are built to manage heavy workloads and scale up and down as necessary with Cloud Tasks. Developers may construct tasks with Cloud Tasks and describe when and how they should be carried out using a straightforward API. Tasks can be set to be completed immediately or at a future date and time. They can also be completed only once or again at regular intervals. In order to provide developers choice in how they construct their applications, tasks may be set to employ a number of execution modalities, including HTTP requests, App Engine tasks, and Cloud Functions.

Scalability is one of Cloud Tasks' main advantages. Because Cloud Tasks is built on top of the architecture of Google Cloud, it can easily manage heavy workloads. Jobs get split up into several instances so that the throughput remains as low as possible, the progress of the tasks can be checked and any logs can be checked and fixed in the codebase, Cloud task provides a monitoring system for that.

Overall, Cloud Tasks is a powerful and flexible service that can help developers create distributed applications that are designed to handle large workloads and scale up and down as needed. With its simple API, flexible execution modes, and robust monitoring and logging capabilities, Cloud Tasks is an ideal solution for developers who need to perform background processing in a reliable and scalable manner.

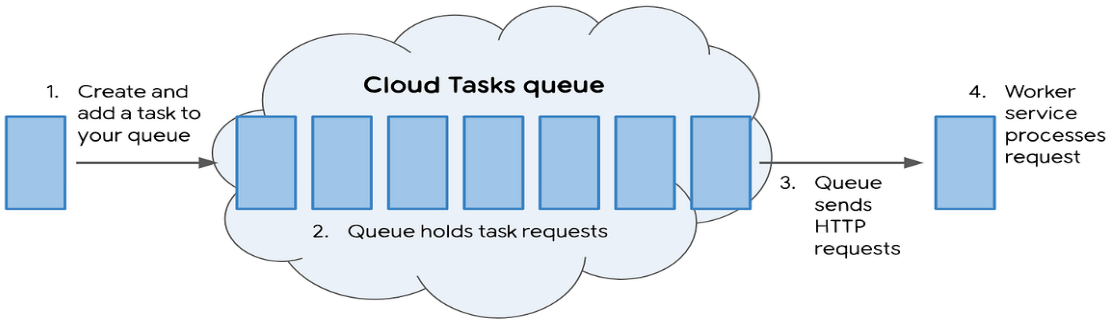

1. Create and add a task to your queue
Cloud Tasks queue
2. Queue holds task requests
3. Queue sends HTTP requests
4. Worker service processes request

FIgure 5. Cloud task queue

3.2.6 Cloud Pub/Sub

Developers may send and receive messages across separate apps using Google Cloud Pub/Sub, a fully-managed, real-time messaging service provided by Google Cloud Platform. Using Pub/Sub, programmers may establish scalable event-driven architectures and build asynchronous, decoupled systems.

A quick and dependable method of message transmission across unrelated applications and services is provided by pub/sub. It is perfect for high-throughput and real-time messaging settings since it can process millions of messages per second and handle messages of any size.

Both scalability and high availability are built into Pub/Sub. Without any human involvement, it can handle increases in message traffic and provides automatic scalability. Strong durability guarantees are also provided, and communications are repeated over many zones to assure that they are never lost.

REST, gRPC, and client libraries for Java, Python, Go, Node.js, and other programming languages are just a few of the protocols and languages that developers may use to connect with Pub/Sub. In addition to third-party services like Apache Kafka, Pub/Sub works smoothly with other Google Cloud services like Cloud Functions, Dataflow, and Kubernetes Engine.

Additionally, Pub/Sub provides sophisticated capabilities like message ordering, message expiry, and message filtering that enable programmers to build sophisticated messaging systems that can manage a variety of use cases.

All things considered, Google Cloud Pub/Sub is a strong and trustworthy messaging service that can assist programmers in creating highly scalable and decoupled applications. Pub/Sub is the perfect option for developers who need to send messages between different apps because of its adaptable protocols, support for a variety of programming languages, cutting-edge functionality, and close connection with other Google Cloud services.

Pub/Sub Triggers

Google Cloud Pub/Sub provides several types of triggers that can be used to execute code or perform other actions in response to incoming messages. These are the triggers:

1. Cloud Functions Trigger: By allowing developers to design serverless functions that react to incoming messages in real-time, cloud functions may be triggered by Pub/Sub messages. Node.js, Python, Java, and Go are just a few of the languages in which Cloud Functions may be created. They can also be dynamically scaled based on the number of incoming messages.
2. Cloud Run Trigger: A serverless computing platform called Cloud Run lets programmers launch containerized programmes in response to incoming signals. Cloud Run services may be activated by Pub/Sub messages, making it simple for developers to create and set up scalable microservices that process messages in real time.
3. Cloud Dataflow Trigger: A fully-managed data processing service called Cloud Dataflow may be utilised to instantly handle huge datasets. Developers may respond to incoming messages by performing complicated data processing activities using Cloud Dataflow pipelines that can be triggered by Pub/Sub messages.
4. Cloud Storage Trigger: Events in Cloud Storage buckets may also be triggered using Pub/Sub. As a result, programmers may construct apps that react to alterations in data saved in cloud storage, such as the upload or deletion of new files.

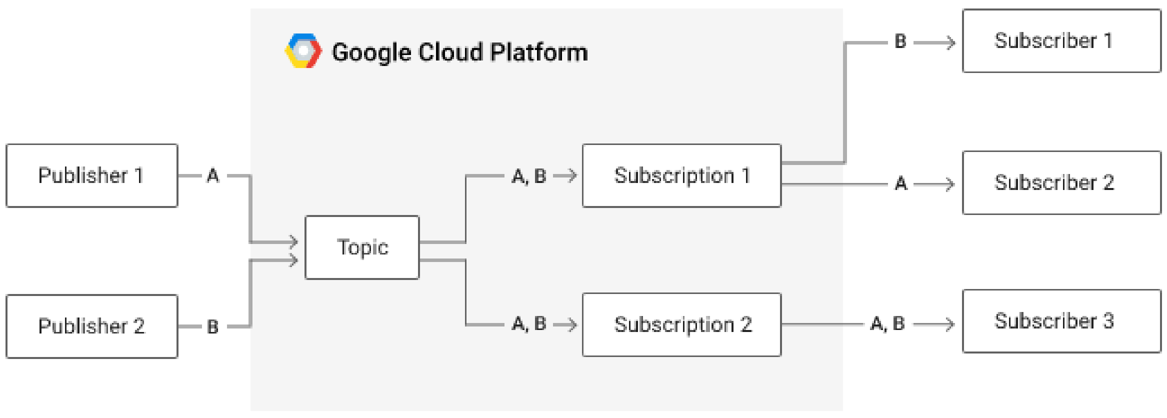


Figure 6. Pub sub architecture

3.3 Database design

3.3.1 DB Design in JSON format

users (Level 0 collection)

```
userId: {
   createdAt: datetime,
   age: number,
   country: string,
   name: string,
   isAdmin: boolean,
   profilePic: string,
   phoneNumber: string,
   currentInviteStreak: ref,
   userId: string,
   updatedAt: datetime
```

invites (Level 1 collection)

```
   inviteStreakId: {
               count: number,
               invitedBy: {
                        id: string,
                        ref: ref
               }
               streakStartDatetime: datetime,
               invites: [
                           {
                                    email: string,
                                    inviteId: string,
                                    status: "sent" | "initiated"
                           }
                        ]
               }
```

```
}
```

items (Level 0 collection)

```
itemId: {
   itemPaymentRef: string | null,
   name: string,
   description: string,
   requester: {
                  id: string,
                  ref: reference
         }
   createdAt: datetime,
   itemStatus: string,
   closureDateTime: datetime,
   notOlderThan: {
          months: integer,
          years: integer
          },
   numberOfItems: integer,
   maxPrice: float,
   status: string (“live”, “stale”, “ended”, “active”),
   currencyCode: string,
   winner: {                    [Optional]
                  price: float,
                  ref: string,
                  name: string
         }
```

bids (Level 1 collection)

```
   BidId: {
          bidder: {
                       id: string,
                       ref: reference
          },

 item: {
                    id: string,
                    ref: reference
          },
 images: array of length 5 {
                    url: null | string
},
 details: string,
 createdAt: datetime,
 rejectionReason: string         [Optional]
 status: (“stale”, “active”, “won”, “rejected”, “lost”),
 isDeleted: Boolean,
 olderThan: {
          months: integer,
          years: integer
          },
```

bidPayment (Level 2 collection)

```
bidTransactionId: {
          paymentIntentId: string | null,
          bid: {
                    id: string,
                    ref: reference
                    },
          bidder: {
                    id: string,
                    ref: reference
                    }
          lastPaymentAttemptAt: datetime | null, (to remove)
          bidPrice: float,
          priceToBePaid: float,
         currencyCode: string
          }
```

```
        },

    itemPayment (Level 1 collection)
    itemId: {
        paymentIntentId: string,
         item: {
                id: string,
                ref: reference
                },
         requester: {
                id: string,
                ref: reference
                },
         priceToBePaid: float
    },
    minBidPrice (Level 1 collection)
    itemId: {
        minPrice: number | null,
        visibility: boolean
     }
},
```

3.3.2 DB design explanation

users collection :

Contains the use related data of the user. Doc id is the user id of the user created in the firebase authentication. User doc is only created when the user authenticates (signs up) successfully with the firebase authentication system and creates its user profile.

profilePic field contains the url of the user profile picture.

currentInviteStreakRef field contains the ref path of the current invite streak that the user holds.

isAdmin field is a check for future usage that can give user admin level access.

invites sub-collection:

A document in this sub collection tells the invite streak of the user. It has the following main properties:

counts: representing the number of users invited in this current streak.
invites: contains an array of users that are invited in this particular streak.
The array contains email, inviteId (unique uuid) and status (sent | initiated).
streakStartDatetime: datetime of the point when this streak was started.

This is to maintain the constraint that the number of invites sent by the user in a day i.e. in a streak of 24 hours should be under a particular count i.e. 20 or 25 stated in the requirement.

items collection:

itemPaymentRef: contains the path of the doc under itemPayment sub collection which is related to this item.
requester: contains the ref id of the user requested this item.
status: contains the db status of the item i.e. whether it is live, stale, ended or active
maxPrice represents the maximum price bid that can be accepted.
winner: is an optional field that will be added to item doc only when the live bidding will end and the winner will be declared which will be kept here.

Bids sub collection under items collection:

bidder: field contains the id, ref of the bidder.
Item: field contains the id, ref of the item on which bid is placed.
Images: is an array of dict { url: string } type which contains the images of the bids.
RejectionReason: denotes the reason for rejection for the bid.
Status: denotes the current status of the bid which can be stale, active, won, rejected or lost.
OlderThan: field tells about the description of the item.

bidPayment sub collection under bids sub collection:

contains the payment related information related with the bid.

Bid: field denotes the bid id, ref

bidder: field denotes the id, ref for the bidder that placed the bid and paid for it.

BidPrice: field contains the bid price.

PriceToBePaid: field contains the price that actually bidder has to pay for placing the bid.

CurrencyCode: is the currency code of the amount that has to be paid by the bidder.

PaymentIntentId: id of the stripe payment intent

ItemPayment sub collection under items collection:

paymentIntentId: id of the stripe payment intent

item: contains id, ref of the item

requester: contains id, ref of the requester

priceToBePaid: contains the price that item requester has to pay

minBidPrice sub collection under items collection:

visibility: field contains whether this doc will be visible, defined in security rules.

MinPrice: tells the current minimum bid price placed on this item.

3.4 Authentication

This part contains the authentication system used in this app and the flow from scratch to user sign up completely. We have enabled Firebase with Identity Platform for extra security in our app as Firebase has given a huge flexibility to Web SDKs that clients directly use. We can create users, send email verification and what not using the functions provided by the library therefore it becomes a point of concern if we should give users to create themselves and delete too. Since backend is being used in the app hence its role in creating users should also come into play therefore the flow is kept such that backend intervention is important.

User cannot create or delete himself currently using Web SDK.

Steps in creating a user:

1. User sends data to sign up to our cloud run service.
2. Service performs all the validation checks and then creates the user using firebase admin SDK.
3. User then Signs In with the email and password and since it's not verified, it's redirected to the Verify Email page.
4. From frontend send email verification method is used to send email to the user to verify it's email.
5. After the email gets verified the user is in emailVerified = True state in its access token.
6. Now the user is redirected to create a user page.
7. The user is still not verified by the backend.
8. Backend now puts an extra level of security, user hits the create user API the backend checks if its email is verified it then creates the user doc after validations and adds a special custom claim as verified = True on its access token.
9. Now on frontend after user creation responds in 201 success the frontend revokes new user token containing custom claim added by the backend.
10. Now users can use all the functionalities of the APIs and app.

3.4.1 Access token

The service provides an access token to the client application when a user logs in to a Firebase app using Firebase Authentication. The access token includes details about the user's identification, including their distinct user ID and any other data kept in their user profile. The access token also has a signature that is used to confirm the token's legitimacy. After that, access to Firebase resources is authorized using the access token. Firebase validates the access token when a user asks for access to a resource, such as reading or writing data in the Firebase Realtime Database, to ensure that the user has permission to do so. Firebase permits access to the resource if the access token is legitimate and the user is authorized.

Access tokens are temporary and are automatically updated by Firebase Authentication as necessary. They normally expire after one hour. By lowering the possibility of unauthorized access via stolen or compromised access tokens, this contributes to the security of Firebase resources.

```
firebase.auth().signInWithEmailAndPassword(email, password)
    .then((userCredential) => {
        // User signed in successfully
        const user = userCredential.user;
        // Get the ID token
        return user.getIdToken();
    })
    .then((idToken) => {
        // Use the ID token
        console.log(idToken);
    })
    .catch((error) => {
        console.error(error);
    });
```

3.4.2 Custom Claims

To manage access to particular resources in your app, Firebase lets you apply custom claims on a user's ID token. When a user logs into your app, custom claims, which are key-value pairs, are added to their authentication profile using the Firebase Admin SDK.

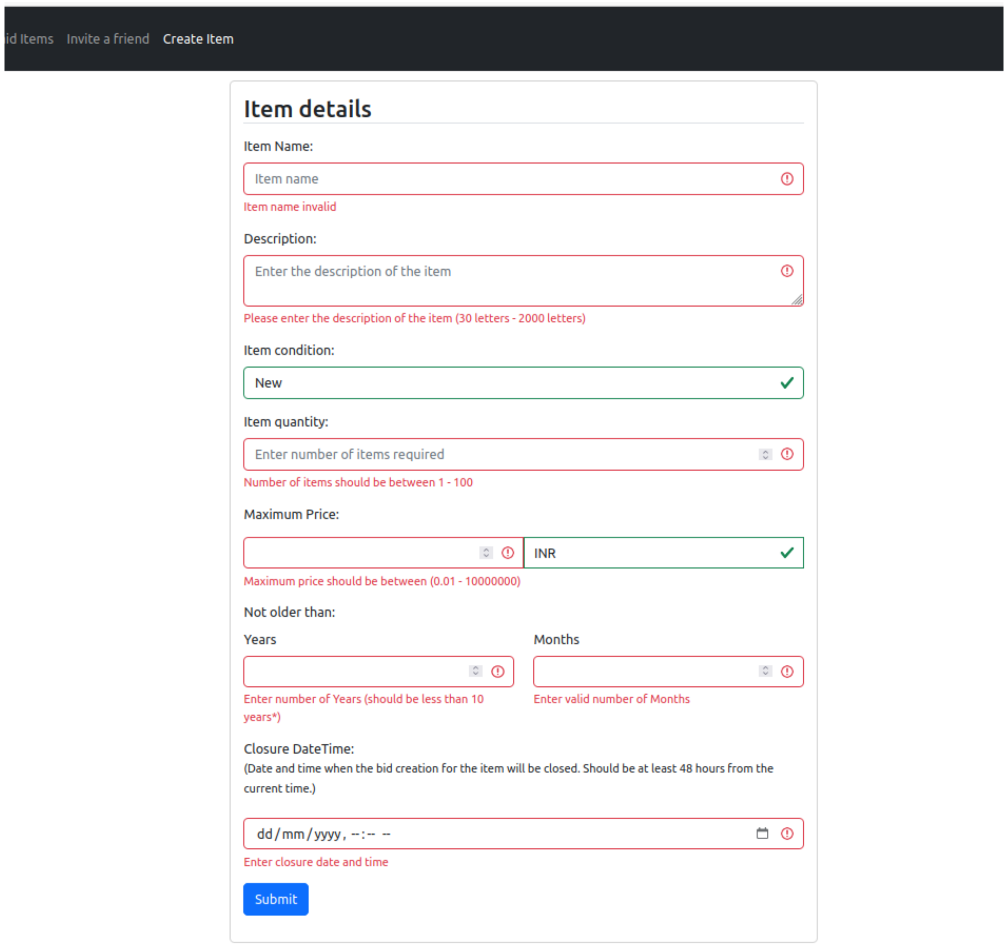
```
import firebase_admin
from firebase_admin import auth

# Initialize Firebase Admin SDK
firebase_admin.initialize_app()

# Set custom user claims on user with uid
uid = "user-uid"
claims = {'admin': True}

auth.set_custom_user_claims(uid, claims)
```

In the aforementioned example, claims is a dictionary object that contains the custom claims you wish to set, and uid is the user ID of the user to whom you wish to add custom claims. The user's custom claims are set using the set_custom_user_claims() function.

The user's ID token will contain the custom claims after the custom claims have been set. The custom claims may then be checked in your server-side code to control access to particular resources in your app.

3.5 Create item

3.5.1 Frontend


id Items Invite a friend Create Item

Item details

Item Name:

Item name

Item name invalid

Description:

Enter the description of the item

Please enter the description of the item (30 letters - 2000 letters)

Item condition:

New

Item quantity:

Enter number of items required

Number of items should be between 1 - 100

Maximum Price:

INR

Maximum price should be between (0.01 - 10000000)

Not older than:

Years

Enter number of Years (should be less than 10 years*)

Months

Enter valid number of Months

Closure DateTime:
(Date and time when the bid creation for the item will be closed. Should be at least 48 hours from the current time.)

dd/mm/yyyy, --:-- --

Enter closure date and time

Submit

Figure 7. Create item form

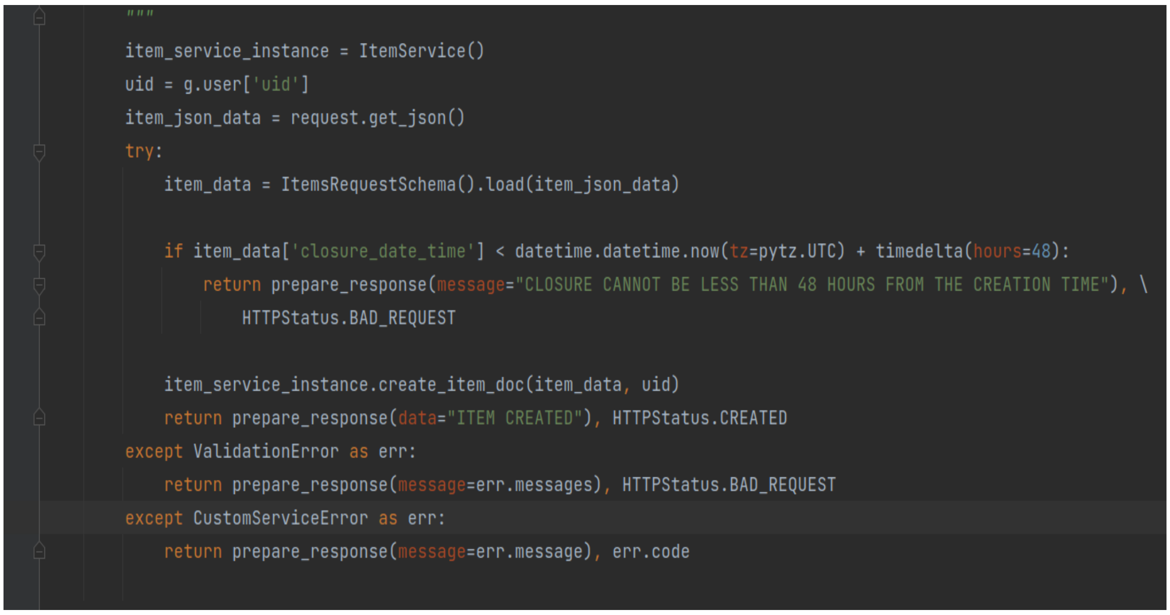
The item details form is responsible for creating item requests. All the validations are added on the input fields including regex, min max length and min max range in number input. For closure date time the entered value should be at least 48 hours away from the date time item request is made.

After the user submits the form a popup comes that contains a message from the backend. Bootstrap form is used along with default html validations. Any error that is encountered by default form validation is automatically shown by using ‘invalid-feedback’ on divs that are responsible for showing error messages. After that another logical validation is performed on the frontend side after which the request then goes to the backend via API.

3.5.1 Backend

After the request comes for the create item the following flow takes place in backend:

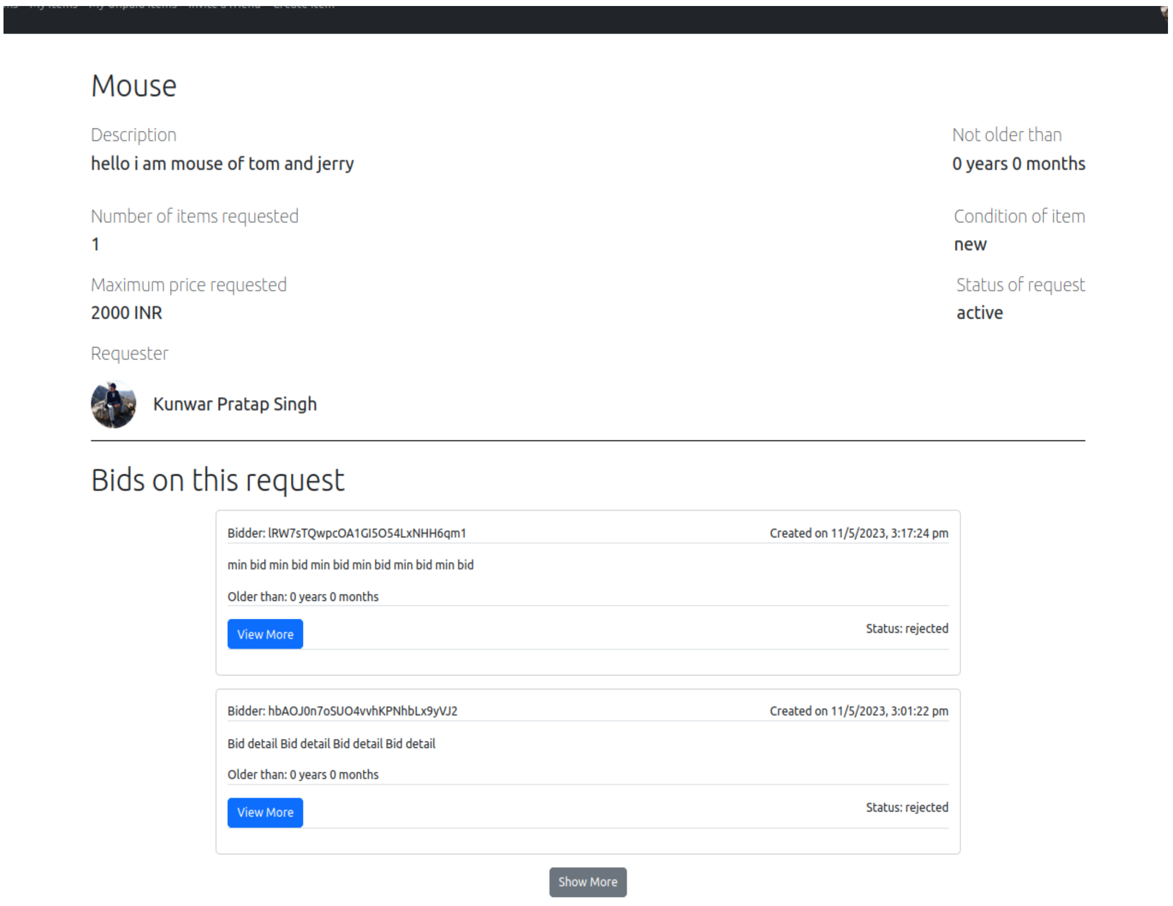
1. Check if the user is a claimed user using the decorator method.
2. A claimed user contains verified=True key, value in its access token under custom claims.
3. Validate the incoming json data using marshmallow using the schema present.
4. If validation passes, then create the data for item doc and validate it using DB schema to dump the item data into db.
5. Dump the data in db.
6. Create the minBidPrice subcollection and add a doc with the same id as item id in it.
7. Initialize the minBidPrice subcollection doc with the initial values.
8. Return the success response with some message to the user with status code 201.



```
"""
item_service_instance = ItemService()
uid = g.user['uid']
item_json_data = request.get_json()
try:
    item_data = ItemsRequestSchema().load(item_json_data)

    if item_data['closure_date_time'] < datetime.datetime.now(tz=pytz.UTC) + timedelta(hours=48):
        return prepare_response(message="CLOSURE CANNOT BE LESS THAN 48 HOURS FROM THE CREATION TIME"), \
            HTTPStatus.BAD_REQUEST

    item_service_instance.create_item_doc(item_data, uid)
    return prepare_response(data="ITEM CREATED"), HTTPStatus.CREATED
except ValidationError as err:
    return prepare_response(message=err.messages), HTTPStatus.BAD_REQUEST
except CustomServiceError as err:
    return prepare_response(message=err.message), err.code
```

Figure 8. Create item backend code

3.6 View Item


Mouse

Description
hello i am mouse of tom and jerry

Not older than
0 years 0 months

Number of items requested
1

Condition of item
new

Maximum price requested
2000 INR

Status of request
active

Requester
Kunwar Pratap Singh

Bids on this request

Bidder: lRW7sTQwpcOA1GI5O54LxNHH6qm1
Created on 11/5/2023, 3:17:24 pm
min bid min bid min bid min bid min bid min bid
Older than: 0 years 0 months
Status: rejected

Bidder: hbAOJ0n7oSUO4vvhKPNhbLx9yVJ2
Created on 11/5/2023, 3:01:22 pm
Bid detail Bid detail Bid detail Bid detail
Older than: 0 years 0 months
Status: rejected

Figure 9. View item frontend page

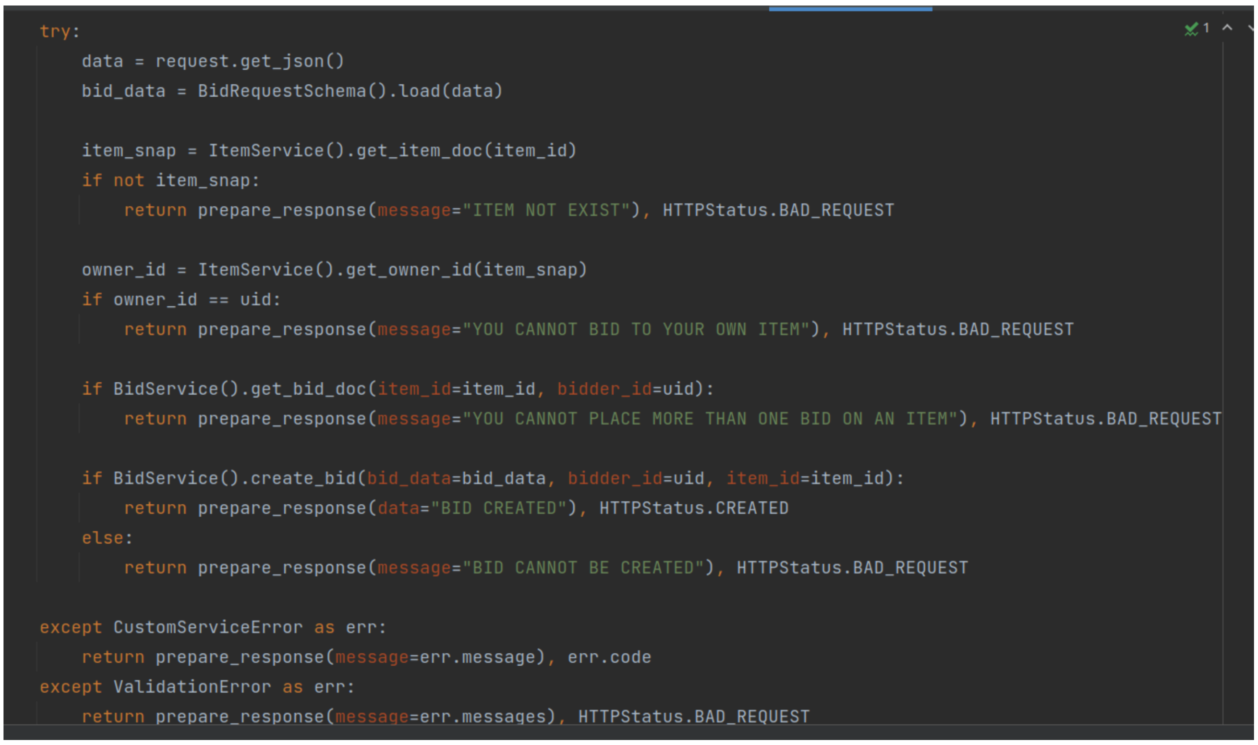
The item details container takes the item details from the API. As the item detail pages is rendered the call to db is made for fetching the required information about the item detail page. Along with the details fetched for item the details are also fetched by the child component containing list of the bids made on this item from the service layer which then calls the db.

Bids list are paginated here and sorted by createdAt date time in descending order.
On pressing show more only the data after the last item is fetched.

3.7 Place Bid

3.7.1 Backend

As the request comes for creating a bid on an item the following flow takes place in backend:

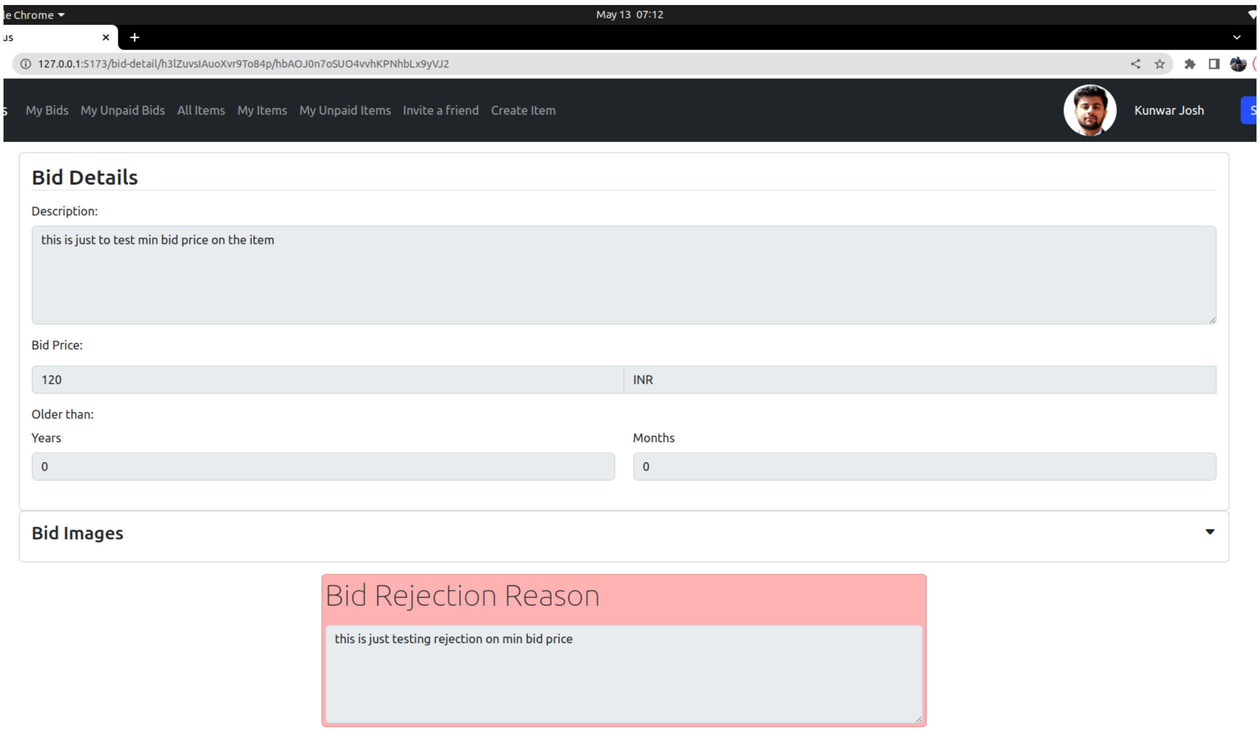


```python
try:
    data = request.get_json()
    bid_data = BidRequestSchema().load(data)

    item_snap = ItemService().get_item_doc(item_id)
    if not item_snap:
        return prepare_response(message="ITEM NOT EXIST"), HTTPStatus.BAD_REQUEST

    owner_id = ItemService().get_owner_id(item_snap)
    if owner_id == uid:
        return prepare_response(message="YOU CANNOT BID TO YOUR OWN ITEM"), HTTPStatus.BAD_REQUEST

    if BidService().get_bid_doc(item_id=item_id, bidder_id=uid):
        return prepare_response(message="YOU CANNOT PLACE MORE THAN ONE BID ON AN ITEM"), HTTPStatus.BAD_REQUEST

    if BidService().create_bid(bid_data=bid_data, bidder_id=uid, item_id=item_id):
        return prepare_response(data="BID CREATED"), HTTPStatus.CREATED
    else:
        return prepare_response(message="BID CANNOT BE CREATED"), HTTPStatus.BAD_REQUEST

except CustomServiceError as err:
    return prepare_response(message=err.message), err.code
except ValidationError as err:
    return prepare_response(message=err.messages), HTTPStatus.BAD_REQUEST
```

Figure 10. Create bid backend code

1. Verify the custom claim using the access token of the user.
2. Take the request data in json format and validate it using a marshmallow schema made for it.
3. Check if the item exists or not.
4. Get the owner info and compare if the bidder is not the owner of the item.
5. Check if the bid is already made or not.
6. Create the bid doc in a transaction.
7. Send required responses to the client.

3.7.2 Frontend



Bid Details

Description:

this is just to test min bid price on the item

Bid Price:

120 INR

Older than:

Years: 0

Months: 0

Bid Images

Bid Rejection Reason

this is just testing rejection on min bid price

Figure 11. Bid details frontend page

This is a multi role and state based multi use container made in react. For a bidder where no bid exists this will show an empty form where users can fill bid details and press on create button after which the form will become a form to update the details of the bid placed by the bidder. Till the bid amount is not paid by the bidder this form will only be displayed and opened by the bidder not the requester or public.

After the bid amount has been paid the bid becomes active and is visible to all the public and the requester as well. Requester will get an option and form to even reject the bid whereas no one else can see this form and button to reject the bid.

After the bid has been rejected or lost/won, the bidder cannot edit the bid details.

Before the bid has been rejected or lost/won, the bidder can edit the form and others can only see the values of the form.

Chapter 04: Experiment and Result Analysis

4.1 Testing of frontend (React app):

Test cases were written to test the part of the application till auth and user details feature including sign in and sign up using React Testing Library and Jest.

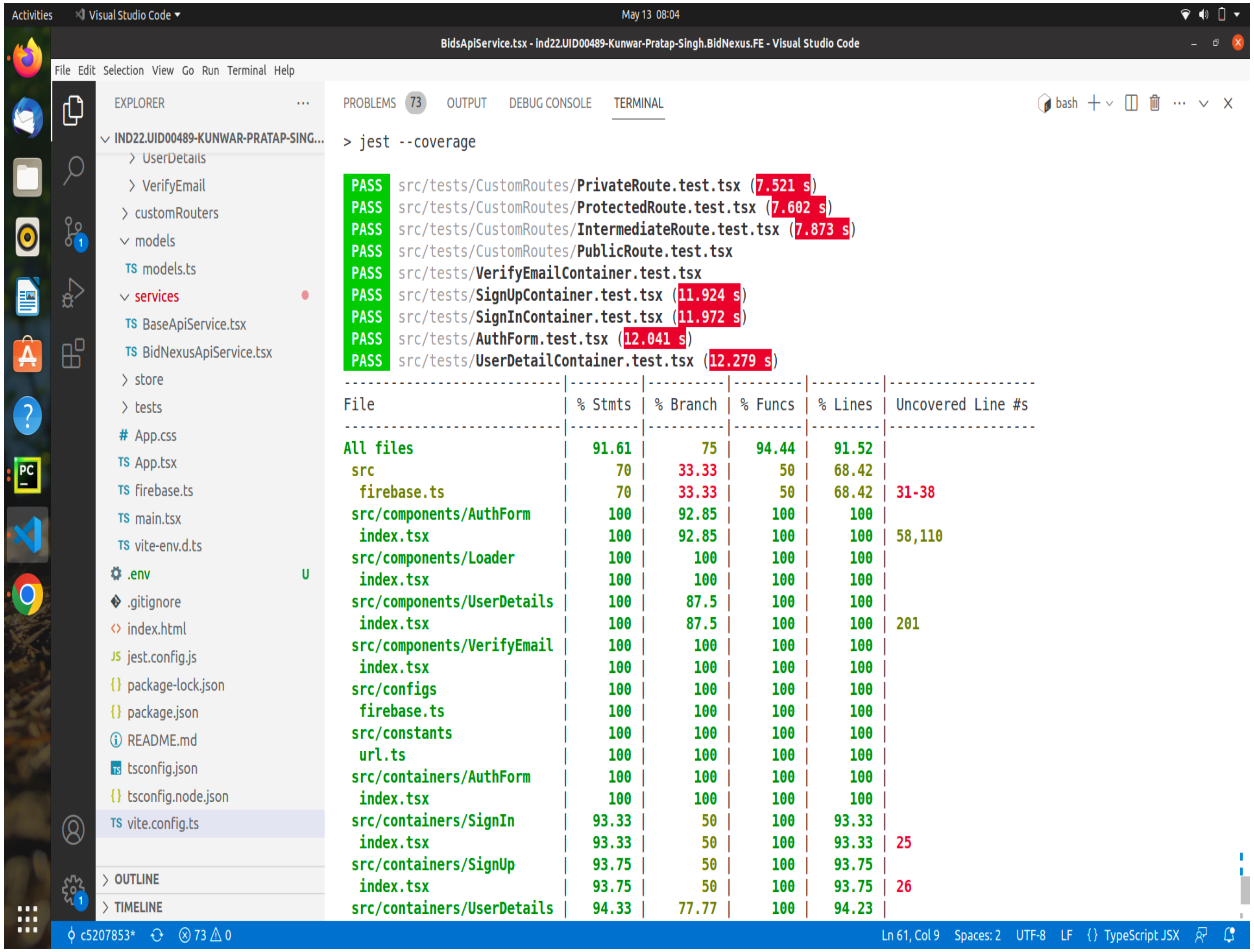
The following methods were used for writing unit tests of a dummy component:

1. Figure out the unit level component you want to test.
2. Test it using the following methods:
    - Test if the component has been rendered on screen or not. This is the most basic test case.
    - Pass the fake prop variables and see how the component behaves.
    - For input fields try to add fake data that can break the validation applied on the fields.
    - Assert the outcome from the component with the expected outcome.

The following methods were used for writing unit test cases of a container that is made up of other components:

1. Check if the container is rendered correctly.
2. Check the buttons, input fields and text all rendered.
3. Mock the service call using the fake function with the help of jest.
4. Now return the fake promise in response to the containers calling functions while test cases.
5. Test for each possible response from service methods that it can give to the container and assert its reaction with the expected value.

The following results were obtained from tests:



```
> jest --coverage

PASS  src/tests/CustomRoutes/PrivateRoute.test.tsx (7.521 s)
PASS  src/tests/CustomRoutes/ProtectedRoute.test.tsx (7.602 s)
PASS  src/tests/CustomRoutes/IntermediateRoute.test.tsx (7.873 s)
PASS  src/tests/CustomRoutes/PublicRoute.test.tsx
PASS  src/tests/VerifyEmailContainer.test.tsx
PASS  src/tests/SignUpContainer.test.tsx (11.924 s)
PASS  src/tests/SignInContainer.test.tsx (11.972 s)
PASS  src/tests/AuthForm.test.tsx (12.041 s)
PASS  src/tests/UserDetailContainer.test.tsx (12.279 s)
----------------------------|---------|----------|---------|---------|-------------------
File                        | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
----------------------------|---------|----------|---------|---------|-------------------
All files                   |   91.61 |       75 |   94.44 |   91.52 |
 src                        |      70 |    33.33 |      50 |   68.42 |
  firebase.ts               |      70 |    33.33 |      50 |   68.42 | 31-38
 src/components/AuthForm    |     100 |    92.85 |     100 |     100 |
  index.tsx                 |     100 |    92.85 |     100 |     100 | 58,110
 src/components/Loader      |     100 |      100 |     100 |     100 |
  index.tsx                 |     100 |      100 |     100 |     100 |
 src/components/UserDetails |     100 |     87.5 |     100 |     100 |
  index.tsx                 |     100 |     87.5 |     100 |     100 | 201
 src/components/VerifyEmail |     100 |      100 |     100 |     100 |
  index.tsx                 |     100 |      100 |     100 |     100 |
 src/configs                |     100 |      100 |     100 |     100 |
  firebase.ts               |     100 |      100 |     100 |     100 |
 src/constants              |     100 |      100 |     100 |     100 |
  url.ts                    |     100 |      100 |     100 |     100 |
 src/containers/AuthForm    |     100 |      100 |     100 |     100 |
  index.tsx                 |     100 |      100 |     100 |     100 |
 src/containers/SignIn      |   93.33 |       50 |     100 |   93.33 |
  index.tsx                 |   93.33 |       50 |     100 |   93.33 | 25
 src/containers/SignUp      |   93.75 |       50 |     100 |   93.75 |
  index.tsx                 |   93.75 |       50 |     100 |   93.75 | 26
 src/containers/UserDetails |   94.33 |    77.77 |     100 |   94.23 |
```

Figure 12. Testing report

4.2 Results

In total 5 Cloud runs, and 1 Pub / Sub topic and one Cloud task queue has been deployed for the proper functioning of the application. The roles and responsibilities of each service are discussed below:

User and Invite feature cloud run:

- This service is responsible for the user authentication and invites sending along with the create, update of the user data.

- When the user first signs up the request is handled by this CR service which then validates the data and makes the appropriate call.

- After this the creation of the user document and the updation of the user detail is also handled by this service.

- When the user uploads a CSV this service validates the CSV file and removes any anomaly after which this is responsible for sending messages to a pub sub topic after properly formatting them to send invites.

Items and Bids feature cloud run:

- This service is responsible for creating the documents of the item requests and handles the payment related work for the item requests.

- This is also responsible for creating bid requests on an item and handles the payment related work of the bid and also the updation of the bid request made by the bidder.

- This contains two heavy services in items and bids modules that perform these tasks independently.

- The image upload functionality of the bid where a user can upload images of products is also handled by the image upload service class in this cloud run. Where a user can request a signed url which it signs by the google cloud and returns to the client on which the client then uploads the image.

Invite cloud run:

- This service is the PUSH subscription service that is a subscriber of the topic in pub/sub where the message related to invites are being published.
- This service takes the request and reads the message and decodes it.
- It then sends the email to the recipient using a third party service API like mailjet.
- After the mail has been successfully sent this cloud run also updates the status of the invites in the invite streak doc.

Payment Webhook cloud run:

- This service is an endpoint to the stripe.
- When a successful payment has been made with the stripe, the stripe sends a request to this endpoint to handle the post payment process.
- It has two endpoints, one for handling the items payment and another for handling the bids payment.
- It updates the status of items and bids to active after it gets the payment succeeded request event from the stripe.

Live Bidding cloud run:

- This service is responsible for handling the cloud task events through HTTP calls made to this endpoint.
- It has two types of endpoints, one for handling the event to start the live bidding process. Another for handling the event to end the live bidding process.
- The service runs asynchronously and is invoked by the cloud task.
- On live bidding starts, it simply changes the status of the item doc.
- On live bidding ends event, it changes the status of the item doc as well as bulk updates the status of all the bids.

Chapter 05: Conclusions

5.1 Conclusions

Cloud technology is emerging and Google Cloud Platform gives an enormous amount of services that we can use and deliver the solutions to real world problems. In this project we took the problem of online auction and we successfully built it using various GCP tools and services. The full stack technology is itself a magic but it’s traditional and running from decades, but combining full stack along with language independent cloud tools gives the developer a lot of flexibility because the cloud tools do the operational work like deployment (most of the part) on their own which saves a lot of time for the developer. Deploying a server and maintaining it and then scaling it requires an extra amount of effort that GCP solves.

Serverless architecture scales rapidly as the requests grow and does not take any effort from the developer side. Google cloud firestore DB is the best example of serverless. It reduces the effort of having a backend to control the application, backend is not a necessity but an option. In our application we have used the backend for all the write operations to make our app extra safe from hackers. The firestore provides the database access directly to the web clients that are using SDK. Firestore has a feature called security rules that it uses to prevent the client from accessing. In general Client Server architecture the client makes a request which goes through the backend, here the backend as a security layer prevents the client from accessing the DB without control. In firestore this layer is provided by the firestore itself in the security rules feature.

We deployed 5 cloud runs for our app and 1 pub / sub topic with deadlettering and 1 cloud task. We tested our application using the deployed services from the frontend and these were write operations, we then fetched the data and displayed them in the frontend. The cloud task was working perfectly in creating tasks at the scheduled time. The cloud pub/sub retires 5 times after which it sends the message to the dead letter topic.

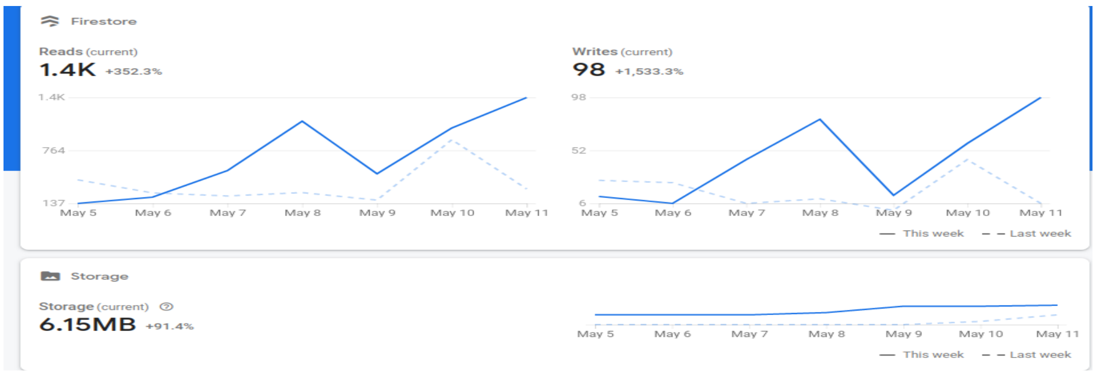


Figure 13. Firestore stats

5.2 Future Improvements

Refund Policy

In the following project the requester can reject the bid of the bidder and bidder won't be able to do anything. This is a loss-loss situation for the bidder therefore, adding a refund feature will do the best. Refund feature will work in the following way:

- If the requester rejects the bid then the bidder will get refunded the amount he paid.

- If the live bidding ends and the bidder gets lost, all the bidders will get their refunds back except the bidder who won the bid.

Admin account

There was no admin account in the project, an admin account became necessary to view all the statistics of the project and manage everything.

Counts

- Number of buyers on the platform
- Number of sellers on the platform
- Number of active item requests
- Number of bidding sessions live at the moment
- Total amount on hold. (i.e payments which may be refunded after bid closure)
- Total amount captured in the system.

Graphs (for last 30 days)

- Number of new item requests Per day
- Number of bidding sessions held Per day
- Net amount captured in the system per day

References

[1]https://www.researchgate.net/publication/342157054_COMPARATIVE_STUDY_OF_CLOUD_PLATFORMS_-MICROSOFT_AZURE_GOOGLE_CLOUD_PLATFORM_AND_AMAZON_EC2/fulltext/5ee5792d92851ce9e7e380b6/COMPARATIVE-STUDY-OF-CLOUD-PLATFORMS-MICROSOFT-AZURE-GOOGLE-CLOUD-PLATFORM-AND-AMAZON-EC2.pdf

[2] https://intellipaat.com/blog/aws-vs-google-cloud/

[3] https://firebase.google.com/docs/firestore/query-data/get-data

[4] https://cloud.google.com/pubsub/docs

[5] https://firebase.google.com/docs/firestore