CONTENTS

List Of Tables

List Of Figures

CHAPTER - 1
COMPANY PROFILE

1.1 Introduction About Kakcho

At **Kakcho**, people are working on solving the challenges that take us a step closer to our mission every day to ensure everyone looks good.

Whether it's building beautiful products to help lakhs of people discover what to wear every morning, or helping them shop, there's plenty to sink your teeth into.

Kakcho was started with a very simple aim and that is to help people decide what to wear and buy every time they get confused by connecting them to Fashion stylists who have complete knowledge about various parameters such as body type, complexion, existing clothes etc. which are taken into consideration before wearing or buying clothes.

Kakcho only live by one motto i.e. Hustle your way through everything. Be it acquisition of users, building technology, sales etc. Our idea to approach every problem is dealing with it head on and doing wonders while learning our way through different skills.

Kakcho is a fashion styling application. We connect users with a fashion stylist and provide them with Instant and Personalised Fashion Advice. Every Morning when we wake up we stand in front of our wardrobes and say “What should I wear”. We ask our partners/friends “How am I looking” and we keep on looking for a confirmation which can make us believe that yes, we are good to go. If we are not comfortable in what we are wearing and if we have doubts about how we are looking, it has a direct impact on our personal and professional growth. When a person is not feeling comfortable then the possibility of not being happy also increases. According to a study done by Stanford 81% of people feel that clothing has a huge impact on their personal and professional growth.

The whole application has been designed in a manner to give it a look and feel of a styling store. The manager attends a user every time he/she comes to the application and the movement within the application is also designed to give a feeling of a store.

1.2 Industry At Glance

India is one of the fastest growing economies in the world. In the past decade India has seen a huge shift in the manner in which people use internet, smartphones etc. India currently has more than 200 million smartphone users and with 350 Million + internet users these numbers are only going to increase with time. India has a become huge E-commerce market with people now being comfortable in shopping online and taking services online. The indian E-commerce industry is set to reach $20 Billion by 2020. Also, at an average out of the 3 Billion queries on google everyday 1 Billion are related to fashion and accessories. People search for different things like 'What to wear to a wedding', 'How to wear a gown', 'When to wear a LBD' etc. These results just add to the existing validation that the problem exists in the market and people are continuously searching for solutions to gain utility. The fashion styling market in India is a growing industry. Colleges like NIft, Pearl etc have started specific courses to teach students about fashion styling. In india by 2020 it is expected that the fashion styling market(Excluding E-commerce) would be around $2 Billion. We did a primary research with 1000 people asking them questions related to the usability and the price they would be willing to pay if a service like this is offered to them.

The results of the research were as follows:

1. 64.5% i.e 645 people out of 1000 get confused in deciding what to wear
2. 86.5% i.e. 865 people out of 1000 have searched for solutions on different platforms to solve their fashion dilemmas
3. 86.7% i.e. 867 people want to take advice from a fashion stylist while deciding what to wear
4. 91% i.e. 910 people out of 1000 think that a fashion stylist can dress them better and can make them look better.

Moreover, there are more than 100 million online shoppers in India who continuously look for a value addition when it comes to buying clothes. Kakcho provides them with an unmatched styling advice which helps them look amazing every time they plan on wearing clothes.

1.3 Marketing Plan

In the First Phase i.e the users we target 2017-18 are as follows:

1. Age Group: 18-27
2. Profession: Students and Young working professionals
3. Places they Visit: Cafes, Clubs, Restaurants, Conferences in Hotels
4. Things they spend on: Grooming, Eating out, Branded Clothes
5. Cities they are present in: Tier 1 cities like Delhi, Bengaluru, Hyderabad, Mumbai, Chennai, Kolkata, Ahmedabad

This age bracket is more adopting towards new things and have high acceptance rate when it comes to new products and because our team is part of this age bracket it becomes easy to access and reach these people.

We have divided our Marketing Plans into 3 sections i.e. :

1. **Social Media**: We have a very active blog with more than 25000 page views and 2500 subscribers. We believe content is the king and we are focussed on building a strong community of readers via our blog. Moreover, we are very active on Instagram where we find most of our target group. On a regular basis we do photo shoots which help us define our problems and also at the same time this brings a huge followership and leads to a cult formation. We have a huge emphasis on our facebook group as well, we try to keep our community updated with what is going on in the company and on a regular basis hold social media campaigns on facebook.
2. **BTL activities and Community Building**: Regular offline meetups are organised in order to reach the target group directly and to get a chance to interact with

them. Moreover, sponsoring campus fests this will help us get a lot of users and also will make our community strong

3. **Growth Hacks**: To gain a sudden traction and to get bulk downloads instantly we are going to work on various growth hacks like Influencer marketing, community building etc. This would help us get a great boost in our target group and would lead to new users on the platform.

1.4 Revenue Model

Kakcho has 3 different revenue streams which is expected to bring in the money they are as following:

1. **Affiliate Marketing**: We have clothes from different ecommerce platforms like jabong, myntra, koovs etc and whenever a user buys a product from our platform, we get a commission ranging between 7-20% (When a user clicks on the buy now button in the app he/she is transferred to the product link via in-app browser)
2. **Chat Packs**: Once we reach a considerable number of users, we plan to make the styling with some stylists paid on the platform and offline as well. We have different bollywood stylists on the platform as well and sessions with these stylists would be paid
3. **Brand Sharing**: We give options to a user to share products from the wardrobe, general store, chat. Every time a user shares a product on any platform like Facebook, Instagram, Twitter the message goes like this " My stylist @kakcho suggested me these cool trousers form #allensolly". In this message the brand allen solly is getting clear promotion and these brands usually spend more than Rs 600 in getting one user on there platform from social media. Our way of promotion will get these companies users at comparatively less price.

It is estimated that these 3 revenue streams would make us reach a gross revenue of 12 crores.

1.5 Competitive Landscape

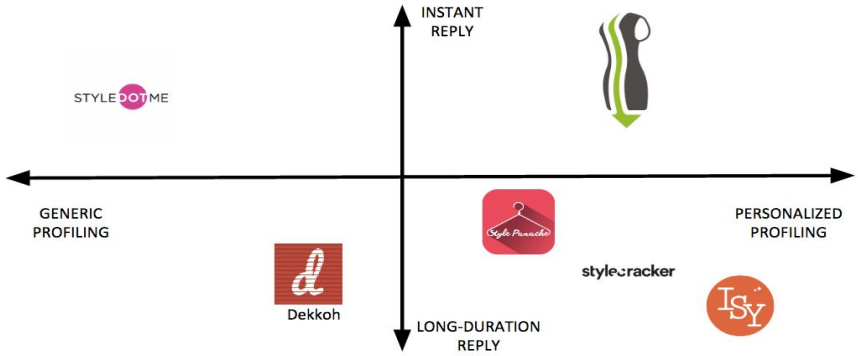


Fig 1.1 Landscape Division Of Kakcho

CHAPTER - 2
INTRODUCTION

2.1 OVERVIEW OF THE PROJECT

This report explains all the details of the development process for the **Kakcho Fashion Android Application** software system. The main feature of this android application is to provide users instant and personalized fashion content. The application connects the user to a fashion stylist and helps them to personalize their attire and get some recommendations on their dressing.

The android application connect users with fashion stylists and provide them with instant and personalized fashion advice, be it a confusion in deciding what to wear from the clothes that they already own or confusion in deciding what to buy while shopping online/offline.

2.2 MOTIVATION

Every time we stand in front of our wardrobe we are confused what should we wear. What will look good on us. What will help us make an impact. Something similar happens when we shop for clothes online/offline we get confused what to buy. What will match with the clothes that we already have. In short we end up wearing clothes that do not suit us or we end up buying clothes that don't look good on us. In this world where first impression is made by your looks or the way you present yourself, this comes out as a very big problem. People don't feel comfortable in what they are wearing and also end up making bad fashion choices which in return affects their performance and also their confidence. This problem directly and indirectly affect their lives as their personal growth is directly related to them making a mark and a lot of times they miss that one chance because they don't feel confident enough in their attire.

2.3 PROBLEM STATEMENT

People need some personalized advice on their day to day clothing. The advice must be personalized according to the personality of the user and should also depend on the availability of clothes to them. The recommendation should be precise i.e. it should depend on the occasion, weather, gender and several other factors as all of these defines what one should wear on a day.

2.4 PROJECT DESCRIPTION

2.4.1 Objective:

The objective of this project is to provide a recommendation system in form of an android application which recommends user on what they should wear, what they should buy, daily style tips, daily style content. Also application should provide user a personal stylist that can help him to know how is he/she looking on a particular look. Also they can help them to make their recommendation system more personalize and give user a better experience. The user should be able to chat with stylist and ask them about their style content.

2.4.2 Purpose:

If an user wants to know what they should wear on a particular occasion then they should get instant suggestion on it. Also if a user wants to shop new clothes then they should get recommendation on what should they buy as it can help them to avoid purchase of similar clothes. Also personal stylist chat system helps them to get more personalized content.

2.4.3 Alternatives Available:

- **Wardrobe dilemma:** People usually use whatsapp and different facebook groups as one medium to try and solve these problems. Clicking photos of their clothes and asking friends and families is the most widespread solution to this problem. Also there exists different facebook groups like "stylestamp" wherein a user uploads a pic of clothes that they are wearing and then people comment on their outfit.

- **Shopping:** Whenever a user is shopping offline the best and the only form of assistance is provided by the sales executive present. Also while shopping online and at times offline, people try and take inspiration from various fashion blogs and fashion discovery apps wherein they try and visualize the clothes worn by someone else on their own self.

2.4.4 What Solution Kakcho Application Provides

- **Provide personalized fashion advice:** Our stylists are well trained professionals. They have a very deep understanding of body types, colors, mixing and matching, new trends, in this way they become very efficient in letting a user know what will look good on them according to the occasion, time, and event type. also
- **Instant:** A user does not have to wait for long times and spend their mornings and precious time in deciding what to buy or wear. Our stylists are going to do it for them in a blink of an eye. This actually saves up a lot of time and energy of a user which they can direct on some more productive work.
- **Getting online clothes reviewed:** Everybody is moving towards online shopping and with this increasing rate and lack of time that people have, our stylists are also going to suggest them clothes that will suit them. Eventually making their lives a lot easier.
- **Availability:** A service which was reserved to celebrities and HNI's of the society is now going to be available to every person and that is the aim that we have. Making every person feel confident about their looks and making them grow professionally and personally is what we want to do.

CHAPTER - 3
WEEKLY JOB SUMMARY

Task	Start Date	End Date	Timeline	Status
Phase 1 And Phase 2	08-Jun	8-Aug		
Admin Panel	08-Jun	20-Jul		Active
Basic Chat Application	08-Jun	10-Jun		Complete
Login With Google And Facebook	10-Jun	12-Jun		Complete
Looks Databse	11-Jun	19-Jun		Complete
Host The Server	13-Jun	14-Jun		Complete
Chat Integration With Android	16-Jun	17-Jun		Complete
Otp System	16-Jun	20-Jun		Complete
Wear Type And Product System	16-Jun	21-Jun		Complete
Wardrobe	20-Jun	27-Jun		Complete
Automation In Chat	20-Jun	24-Jun		Complete
What To Wear	22-Jun	3-Jul		Complete
How Am I Looking	24-Jun	26-Jun		Complete
Notifications	30-Jun	2-Jul		Complete
UI For Admin Chat Panel	2-Jul	22-Jul		Active
Daily Look	6-Jul	18-Jul		Complete
Daily Content	6-Jul	17-Jul		Complete
Daily Feedback	9-Jul	24-Jul		Complete
Feedback And Rating	13-Jul	21-Jul		Complete
Referral	13-Jul	24-Jul		Complete
Payment Integration	14-Jul	29-Jul		Complete
User Profile	15-Jul	29-Jul		Complete
What To Buy	18-Jul	8-Aug		Active

Fig 3.1 GANTT Chart For Kakcho Fashion App

WEEK	Module COVERED	STATUS
Week 1	**Project Setup** The basic project was setup on ruby on rails , node.js and android. ● Admin Panel Integration ● Login Functionality ● Basic Chat Application on Socket.io ● Database Design ● Host server	Completed
Week 2	**Chat Integration In Android** The basic chat module was meant to be integrated with android ● Setup socket on android ● Login integration in android ● Otp verification system	Completed
Week 3	**Automation and wardrobe** Recommendation automation and virtual wardrobe setup ● What to wear ● How am i looking ● Virtual Wardrobe ● Automation	Completed

	● Server Scaling	
Week 4	**Notifications And Chron Jobs** Daily look and content notification chron jobs were meant to be setup ● Daily look ● Daily content ● FCM notification ● Daily Feedback for suggested looks	Completed
Week 5	**Feedback, Referral and chat module ui** ● User feedback and rating system ● Referral system ● Semantic ui and spa on stylist chat panel ● Looks suggestion and sorting on the panel ● Calendar	Completed
Week 6	**Payment Integration** ● Payment Integration with Razorpay ● User profile ● What to buy ● Bug fixing	Completed

CHAPTER - 4
TECHNOLOGIES USED

4.1 ANDROID

Android is a mobile operating system developed by Google, based on a modified version of the Linux kernel and other open source software and designed primarily for touchscreen mobile devices such as smartphones and tablets. In addition, Google has further developed Android TV for televisions, Android Auto for cars, and Wear OS for wrist watches, each with a specialized user interface. Initially developed by Android Inc., which Google bought in 2005, Android was unveiled in 2007, with the first commercial Android device launched in September 2008. The operating system has since gone through multiple major releases, with the current version being 9.0 "Pie", released in August 2018. The core Android source code is known as Android Open Source Project (AOSP), and is primarily licensed under the Apache License.

Android is also associated with a suite of proprietary software developed by Google, including core apps for services such as Gmail and Google Search, as well as the application store and digital distribution platform Google Play, and associated development platform. These apps are licensed by manufacturers of Android devices certified under standards imposed by Google, but AOSP has been used as the basis of competing Android ecosystems, such as Amazon.com's Fire OS, which utilize their own equivalents to the Google Mobile Services.

4.2 ANDROID STUDIO

Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ IDEA . On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps.

4.3 RUBY

Ruby is a dynamic, interpreted, reflective, object-oriented, general-purpose programming language. It was designed and developed in the mid-1990s by Yukihiro "Matz" Matsumoto in Japan.

According to the creator, Ruby was influenced by Perl, Smalltalk, Eiffel, Ada, and Lisp. It supports multiple programming paradigms, including functional, object-oriented, and imperative. It also has a dynamic type system and automatic memory management.

4.3.1 Features

Thoroughly object-oriented with inheritance, mixins and metaclasses

- Dynamic typing and duck typing
- Everything is an expression (even statements) and everything is executed imperatively (even declarations)
- Succinct and flexible syntax that minimizes syntactic noise and serves as a foundation for domain-specific languages
- Dynamic reflection and alteration of objects to facilitate metaprogramming
- Lexical closures, iterators and generators, with a block syntax
- Literal notation for arrays, hashes, regular expressions and symbols
- Embedding code in strings (interpolation)
- Default arguments
- Four levels of variable scope (global, class, instance, and local) denoted by sigils or the lack thereof
- Garbage collection
- First-class continuations
- Strict boolean coercion rules (everything is true except false and nil)
- Exception handling
- Operator overloading
- Built-in support for rational numbers, complex numbers and arbitrary-precision arithmetic
- Custom dispatch behavior (through method_missing and const_missing)
- Native threads and cooperative fibers (fibers are a 1.9/YARV feature)

- Initial support for Unicode and multiple character encodings (no ICU support
- Native plug-in API in C
- Interactive Ruby Shell (a REPL)
- Centralized package management through RubyGems
- Implemented on all major platforms
- Large standard library, including modules for YAML, JSON, XML, CGI, OpenSSL, HTTP, FTP, RSS and curses .

4.4 RUBY ON RAILS

Ruby on Rails, or **Rails**, is a server-side web application framework written in Ruby under the MIT License. Rails is a model–view–controller (MVC) framework, providing default structures for a database, a web service, and web pages. It encourages and facilitates the use of web standards such as JSON or XML for data transfer, and HTML, CSS and JavaScript for display and user interfacing. In addition to MVC, Rails emphasizes the use of other well-known software engineering patterns and paradigms, including **convention over configuration (CoC)**, **don't repeat yourself(DRY)**, and the **active record pattern**.[4]

Ruby on Rails' emergence in the 2000s greatly influenced web app development, through innovative features such as seamless database table creations, migrations, and scaffolding of views to enable rapid application development. Ruby on Rails' influence on other web frameworks remains apparent today, with many frameworks in other languages borrowing its ideas, including Django in Python, Laravel in PHP, Phoenix in Elixir, and Sails.js in Node.js.Like other web frameworks, Ruby on Rails uses the **model–view–controller** (**MVC**) pattern to organize application programming.

4.4.1 Why Ruby On Rails?

- The process of programming is much faster than with other frameworks and languages, partly because of the object-oriented nature of Ruby and the vast collection of open source code available within the Rails community.

- The Rails conventions also make it easy for developers to move between different Rails projects, as each project will tend to follow the same structure and coding practices.
- Rails is good for **Rapid Application Development** (**RAD**), as the framework makes it easy to accommodate changes.
- Ruby code is very readable and mostly self-documenting. This increases productivity, as there is less need to write out separate documentation, making it easier for other developers to pick up existing projects.
- Rails has developed a strong focus on testing, and has good testing frameworks.
- Rails and most of its libraries are open source, so unlike other commercial development frameworks there are no licensing costs involved.
- Rails has an army of committers that make sure it stays in tip-top condition. Many projects simmer down with age, yet with Rails, sparks still fly when decisions need to be made. It feels like the maintainers (still) truly care and want people to use Ruby on Rails and understand its benefits.
- Inbuilt support for chat application development.
- It's scalable. If you expect to get a lot of users for your application you should make sure that it can cope with all the visitors you're hoping to attract.
- Some security features are built into the framework and enabled by default. Using Ruby on Rails also means following the Secure Development Lifecycle, which is a complex security assurance process.
- Ruby on Rails contains many ready-made plugins and modules, which allow developers not to waste time on writing boilerplate code. It's proven that RoR teams build applications 30–40% faster than teams using other programming languages and frameworks.

4.4.2 Trends

The following picture shows the worldwide comparison of four major web frameworks used worldwide i.e **Ruby on Rails**, **Laravel**, **Django**, **Express.JS.** The trends clearly shows how Ruby on Rails popularity overshoots every other major web framework used. The RAD and COC paradigms makes rails one of the most popular, famous, efficient, scalable, maintainable and easy to use web framework.

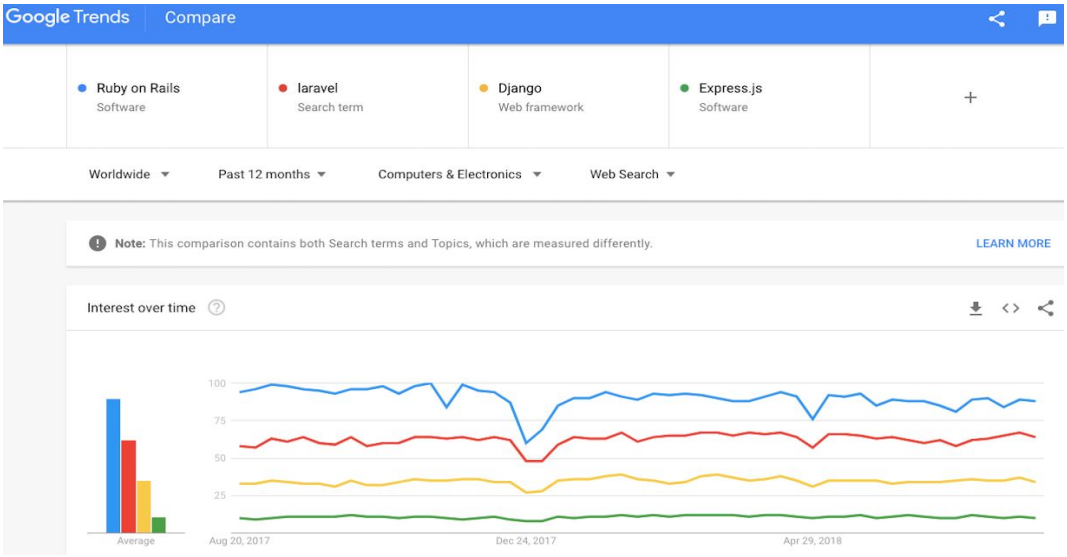


Fig 4.1 Ruby On Rails Google Trends Comparison With Other Frameworks

4.4.2 Who all use Ruby On Rails?

There is an endless list of companies who has ruby on rails as a part of their tech stack. Some of them are:

- Github
- AirBnB
- Code Academy
- Twitter
- Coding Ninjas
- Intuit

- Groupon
- Heroku
- Digital Ocean
- Shopify
- Tree House
- Stripe
- Hackerrank

These were some of the most famous companies that majorly use ruby on rails.

4.4.3 Components of Ruby On Rails

Rails components are modules which are included by default in `application.rb` with `require rails/all`:

- **Action Mailer** is a module for designing email service layers
- **Action Pack** is a module for handling and responding to web requests (*includes action_controller, action_dispatch*)
- **Action View** is a module for handling view template lookup and rendering
- **Active Job** is a module for declaring jobs and making them run on a variety of queueing backends
- **Active Model** is non-database functionality extracted from Rails 2 Active Record (`validates :name, presence: true`)
- **Active Record** connects classes to relational database tables (*migrations, associations*)
- **Active Support** contains all Ruby extensions
- **Action Cable** seamlessly integrates WebSockets with the rest of your Rails application. It allows for real-time features to be written in Ruby in the same style and form as the rest of your Rails application, while still being performant and scalable.

4.4.4 Class Hierarchy in Ruby On Rails

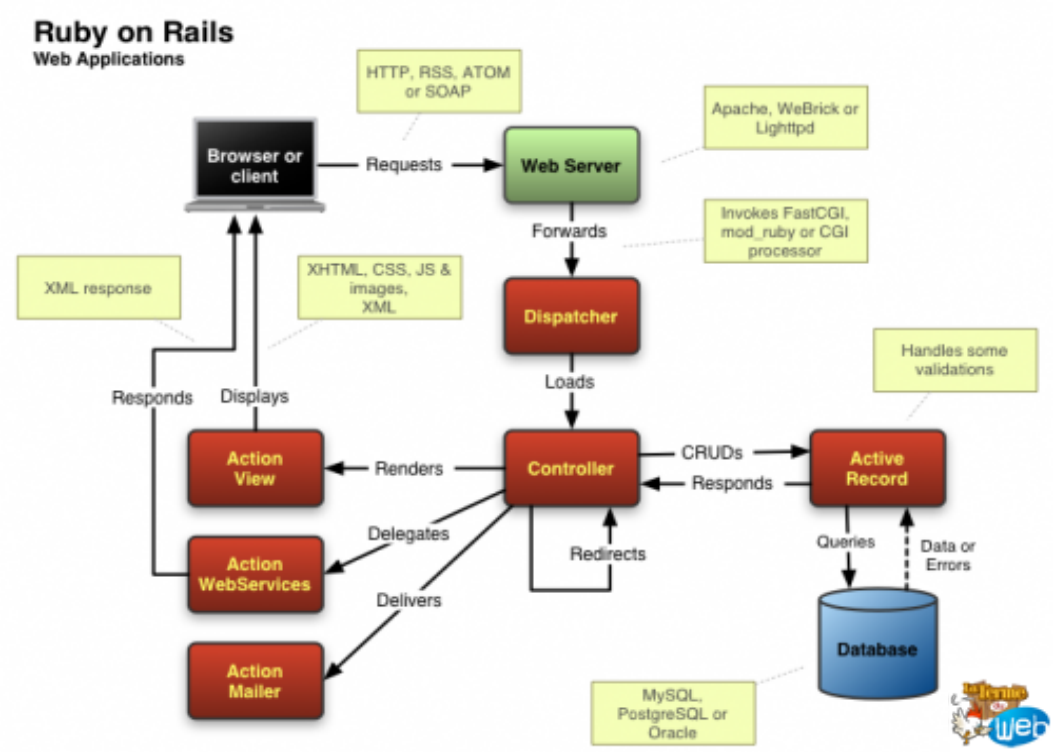


Fig 4.2 Class Hierarchy For Ruby On Rails

4.4.4 MVC in Ruby On Rails

In a default configuration, a model in the Ruby on Rails framework maps to a table in a database and to a Ruby file. For example, a model class User will usually be defined in the file 'user.rb' in the app/models directory, and linked to the table 'users' in the database. While developers are free to ignore this convention and choose different names for their models, files, and database table, this is not common practice and is usually discouraged in accordance with the "**convention-over-configuration**" philosophy.

A controller is a server-side component of Rails that responds to external requests from the web server to the application, by determining which view file to render. The controller may also have to query one or more models for information and pass these on to the view. For example, in an airline reservation system, a controller implementing a

flight-search function would need to query a model representing individual flights to find flights matching the search, and might also need to query models representing airports and airlines to find related secondary data. The controller might then pass some subset of the flight data to the corresponding view, which would contain a mixture of static HTML and logic that use the flight data to create an HTML document containing a table with one row per flight. A controller may provide one or more actions. In Ruby on Rails, an action is typically a basic unit that describes how to respond to a specific external web-browser request. Also, note that the controller/action will be accessible for external web requests only if a corresponding route is mapped to it. Rails encourages developers to use RESTful routes, which include actions such as create, new, edit, update, destroy, show, and index. These mappings of incoming requests/routes to controller actions can be easily set up in the routes.rb configuration file.

A view in the default configuration of Rails is an erb file, which is evaluated and converted to HTML at run-time. Alternatively, many other templating systems can be used for views.

Ruby on Rails includes tools that make common development tasks easier "out-of-the-box", such as scaffolding that can automatically construct some of the models and views needed for a basic website. Also included are PUMA, a simple Ruby web server that is distributed with Ruby, and Rake, a build system, distributed as a gem. Together with Ruby on Rails, these tools provide a basic development environment.

4.5 JAVASCRIPT

JavaScript often abbreviated as **JS**, is a high-level, interpreted programming language. It is a language which is also characterized as dynamic, weakly typed, prototype-basedand multi-paradigm.

Alongside HTML and CSS, JavaScript is one of the three core technologies of the World Wide Web. JavaScript enables interactive web pages and thus is an essential part of web applications. The vast majority of websites use it, and all major web browsers have a dedicated JavaScript engine to execute it.

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative (including object-oriented and prototype-based) programming styles. It has an API for working with text, arrays, dates, regular expressions, and basic manipulation of the DOM, but the language itself does not include any I/O, such as networking, storage, or graphics facilities, relying for these upon the host environment in which it is embedded.

Initially only implemented client-side in web browsers, JavaScript engines are now embedded in many other types of host software, including server-side in web servers and databases, and in non-web programs such as word processors and PDF software, and in runtime environments that make JavaScript available for writing mobile and desktop applications, including desktop widgets.

Although there are strong outward similarities between JavaScript and Java, including language name, syntax, and respective standard libraries, the two languages are distinct and differ greatly in design; JavaScript was influenced by programming languages such as Self and Scheme.

4.6 NODE.JS

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside the browser. Historically, JavaScript was used primarily for client-side scripting, in which scripts written in JavaScript are embedded in a webpage's HTML and run client-side by a JavaScript engine in the user's web browser. Node.js lets developers use JavaScript to write Command Line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm,unifying web application development around a single programming language, rather than different languages for server side and client side scripts.

Though .js is the conventional filename extension for JavaScript code, the name "Node.js" does not refer to a particular file in this context and is merely the name of the

product. Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications (e.g., real-time communication programs and browser games).

The Node.js distributed development project, governed by the Node.js Foundation, is facilitated by the Linux Foundation Collaborative Projects program.Node.js allows the creation of Web servers and networking tools using JavaScript and a collection of "modules" that handle various core functionality. Modules are provided for file system I/O, networking (DNS, HTTP, TCP, TLS/SSL, or UDP), binary data (buffers), cryptography functions, data streams, and other core functions.Node.js's modules use an API designed to reduce the complexity of writing server applications. Though initially the module system was based on commonjs module pattern, the recent introduction of modules in the ECMAScript specification has shifted the direction of using ECMAScript Modules in Node.js by default instead.

Node.js was built on the Google V8 JavaScript engine since it was open-sourced under the BSD license. It is extremely fast and proficient with internet fundamentals such as HTTP, DNS, TCP. Also, JavaScript was a well-known language, making Node.js immediately accessible to the entire web development community.

Microsoft has also come up with ChakraCode(Same is backend of MS Edge browser) as the javaScript engine in Node.

Features :- Following are some of the important features that make Node.js the first choice of software architects.

- **Asynchronous and Event Driven** − All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
- **Very Fast** − Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.

- **Single Threaded but Highly Scalable** − Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
- **No Buffering** − Node.js applications never buffer any data. These applications simply output the data in chunks.

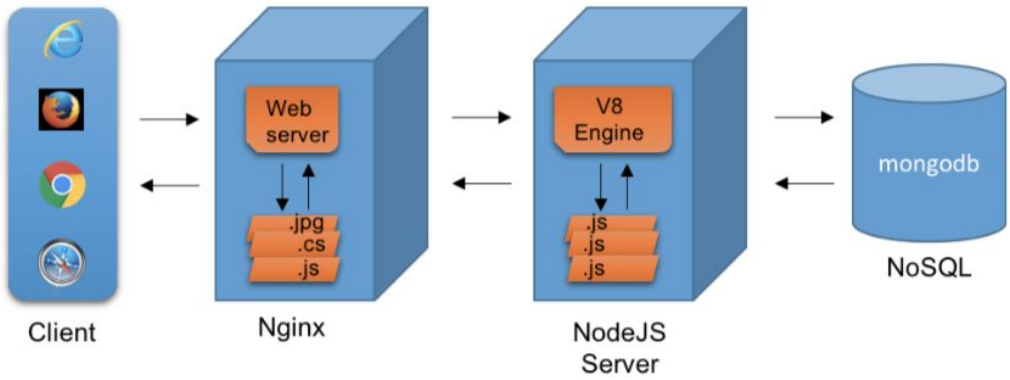


Fig 4.3 Node.JS General Server Architecture

Disadvantages of Node.js:

- Single Threaded: here one need not care about the synchronization between threads which means that in cases of any problems, the programmers themselves has to solve those problems.
- Lack of Maturity: Most of the core libraries have reached the status of stable, and you can trust them to usually do the right thing. But the ecosystem itself is still fairly immature. It's also not that easy to assess the trait of a particular module (core or otherwise) because of the lack of features for ensuring the quality of code from JavaScript itself.

- Hard to make things fault oriented: In JAVA script there are no mechanisms to resolve the errors. For any software particular mechanisms are not found to solve all the problems to make no errors in the coming future.

4.7 SOCKET.IO

Socket.IO is a JavaScript Library for realtime web applications. It enables real time, bi-directional communication between web clients and servers. It has two parts: a client-side library that runs in the browser, and a server-side library for Node.js. Both components have a nearly identical API. Like Node.js, it is event-driven.

Socket.IO primarily uses the WebSocket protocol with polling as a fallback option,[3] while providing the same interface. Although it can be used as simply a wrapper for WebSocket, it provides many more features, including broadcasting to multiple sockets, storing data associated with each client, and asynchronous I/O. It can be installed with the npm tool.

Socket.IO provides the ability to implement real-time analytics, binary streaming, instant messaging, and document collaboration. Notable users include Microsoft Office, Yammer, and Zendesk.

Socket.IO handles the connection transparently. It will automatically upgrade to WebSocket if possible. This requires the programmer to only have Socket.IO knowledge.

Socket.IO is not a WebSocket library with fallback options to other real time protocols. It is a custom real time transport protocol implementation on top of other real time protocols. Its protocol negotiation parts cause a client supporting standard WebSocket to not be able to contact a Socket.IO server. And a Socket.IO implementing client cannot talk to a non-Socket.IO based WebSocket or Long Polling Comet server. Therefore, Socket.IO requires using the Socket.IO libraries on both client and server side.

As of version 2.0, Socket.IO makes use of µWebSockets as the underlying WebSocket library.

4.6.1 Web Sockets Architecture

The following figure represent the architecture of web socket and a web socket connection.

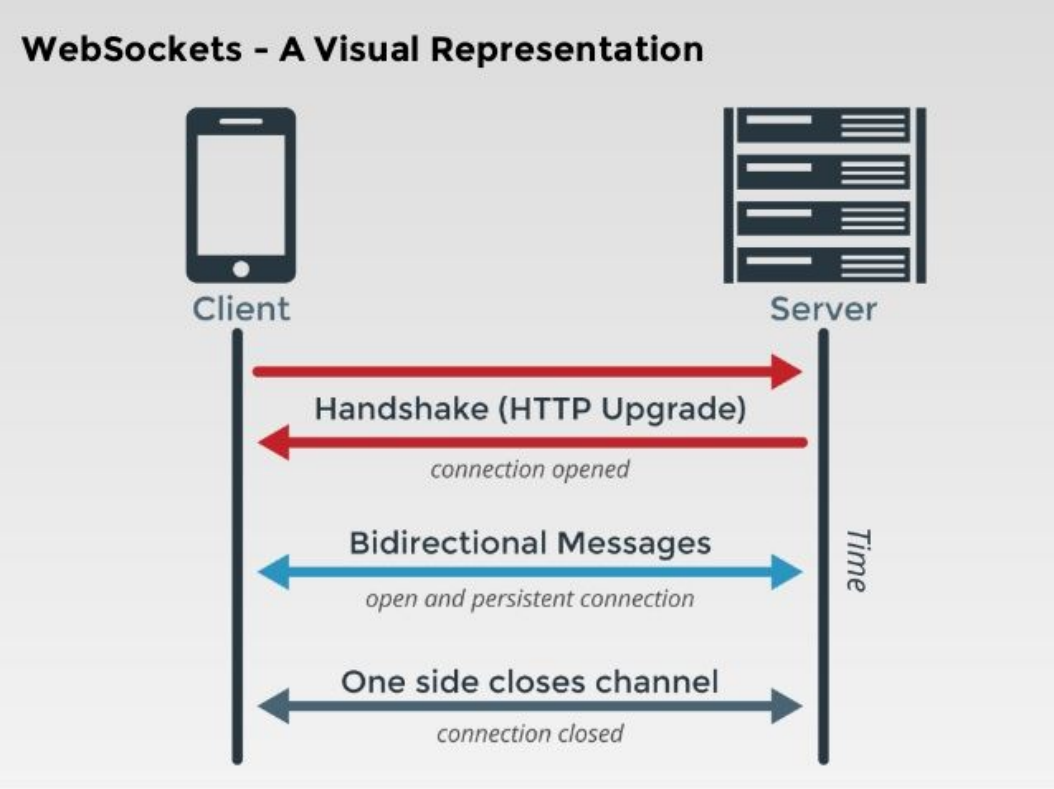


Fig 4.4 Websockets Architecture

4.7 AMAZON WEB SERVICES

Amazon Web Services (AWS) is a comprehensive, evolving cloud computing platform provided by Amazon. It provides a mix of infrastructure as a service (IaaS), platform as a service (PaaS) and packaged software as a service (SaaS) offerings.AWS launched in 2006 from the internal infrastructure that Amazon.com built to handle its online retail operations. AWS was one of the first companies to introduce a pay-as-you-go cloud

computing model that scales to provide users with compute, storage or throughput as needed.

Amazon Web Services provides services from dozens of data centers spread across availability zones (AZs) in regions across the world. An AZ represents a location that typically contains multiple physical data centers, while a region is a collection of AZs in geographic proximity connected by low-latency network links. An AWS customer can spin up virtual machines (VMs) and replicate data in different AZs to achieve a highly reliable infrastructure that is resistant to failures of individual servers or an entire data center.

More than 100 services comprise the Amazon Web Services portfolio, including those for compute, databases, infrastructure management, application development and security. These services, by category, include:

- Amazon Elastic Compute Cloud (EC2)
- Amazon Simple Storage Service (S3)
- Amazon DynamoDB
- Amazon Relational Database Service
- Amazon Virtual Private Cloud
- Amazon Route 53
- Amazon AI
- AWS Mobile Hub
- Amazon Simple Notification Service

4.7 POSTGRESQL

PostgreSQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. The origins of PostgreSQL date back to 1986 as part of the POSTGRES project at the University of California at Berkeley and has more than 30 years of active development on the core platform.

PostgreSQL comes with many features aimed to help developers build applications, administrators to protect data integrity and build fault-tolerant environments, and help you manage your data no matter how big or small the dataset. In addition to being free and open source, PostgreSQL is highly extensible. For example, you can define your own data types, build out custom functions, even write code from different programming languages without recompiling your database!

PostgreSQL tries to conform with the SQL standard where such conformance does not contradict traditional features or could lead to poor architectural decisions. Many of the features required by the SQL standard are supported, though sometimes with slightly differing syntax or function. Further moves towards conformance can be expected over time. As of the version 10 release in October 2017, PostgreSQL conforms to at least 160 of the 179 mandatory features for SQL:2011 Core conformance, where as of this writing, no relational database meets full conformance with this standard.

Below is an inexhaustive of various features found in PostgreSQL, with more being added in every major release:

- **Data Types**
    - Primitives: Integer, Numeric, String, Boolean
    - Structured: Date/Time, Array, Range, UUID
    - Document: JSON/JSONB, XML, Key-value (Hstore)
    - Geometry: Point, Line, Circle, Polygon
    - Customizations: Composite, Custom Types
- **Data Integrity**
    - UNIQUE, NOT NULL
    - Primary Keys
    - Foreign Keys
    - Exclusion Constraints
    - Explicit Locks, Advisory Locks
- **Concurrency, Performance**
    - Indexing: B-tree, Multicolumn, Expressions, Partial

    - Advanced Indexing: GiST, SP-Gist, KNN Gist, GIN, BRIN, Bloom filters
    - Sophisticated query planner / optimizer, index-only scans, multicolumn statistics
    - Transactions, Nested Transactions (via savepoints)
    - Multi-Version concurrency Control (MVCC)
    - Parallelization of read queries
    - Table partitioning
    - All transaction isolation levels defined in the SQL standard, including Serializable
- **Reliability, Disaster Recovery**
    - Write-ahead Logging (WAL)
    - Replication: Asynchronous, Synchronous, Logical
    - Point-in-time-recovery (PITR), active standbys
    - Tablespaces
- **Security**
    - Authentication: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, Certificate, and more
    - Robust access-control system
    - Column and row-level security
- **Extensibility**
    - Stored procedures
    - Procedural Languages: PL/PGSQL, Perl, Python (and many more)
    - Foreign data wrappers: connect to other databases or streams with a standard SQL interface
    - Many extensions that provide additional functionality, including PostGIS
- **Internationalisation, Text Search**
    - Support for international character sets, e.g. through ICU collations
    - Full-text search

There are many more features that you can discover in the PostgreSQL documentation. Additionally, PostgreSQL is highly extensible: many features, such as indexes, have defined APIs so that you can build out with PostgreSQL to solve your challenges. PostgreSQL has been proven to be highly scalable both in the sheer quantity of data it can manage and in the number of concurrent users it can accommodate. There are active PostgreSQL clusters in production environments that manage many terabytes of data, and specialized systems that manage petabytes.

4.7 REDIS

Redis is one of most known and used NoSQL databases among developers. Being open sourced it is equipped with over 20000 stars in GitHub and it's found here. Redis is written in C and is mainly supported for Linux and related operating systems, but there are few ways to run Redis on Windows. According to Redis homepage, Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker.

It supports various data structures such as Strings, Hashes, Lists, Sets etc.

- It is blazingly fast! After all, it has been written in C.
- Currently, it is being used by tech-giants like GitHub,Weibo, Pinterest, Snapchat, Craigslist, Digg, StackOverflow, Flickr.
- In order to save your cloud database calls and eventually saving some dollars out there, you can of course opt for caching so the **Redis**.
- It is Developer friendly and by that I mean to say that **Redis** is being supported in most of the languages (*Perks of using an Open Source Technology*). Languages like JavaScript, Java, Go, C, C++, C#, Python, Objective-C, PHP and almost every famous language out there has support for this.

4.7 UNICORN SERVER

Unicorn is an HTTP server for Rack applications designed to only serve fast clients on low-latency, high-bandwidth connections and take advantage of features in Unix/Unix-like kernels. Slow clients should only be served by placing a reverse proxy capable of fully buffering both the the request and response in between unicorn and slow clients.

Features

- Designed for Rack, Unix, fast clients, and ease-of-debugging. We cut out everything that is better supported by the operating system, nginx or Rack.
- Compatible with Ruby 1.9.3 and later. unicorn 4.x remains supported for Ruby 1.8 users.
- Process management: unicorn will reap and restart workers that die from broken apps. There is no need to manage multiple processes or ports yourself. unicorn can spawn and manage any number of worker processes you choose to scale to your backend.
- Load balancing is done entirely by the operating system kernel. Requests never pile up behind a busy worker process.
- Does not care if your application is thread-safe or not, workers all run within their own isolated address space and only serve one client at a time for maximum robustness.
- Builtin reopening of all log files in your application via USR1 signal. This allows logrotate to rotate files atomically and quickly via rename instead of the racy and slow copytruncate method. unicorn also takes steps to ensure multi-line log entries from one request all stay within the same file.
- nginx-style binary upgrades without losing connections. You can upgrade unicorn, your entire application, libraries and even your Ruby interpreter without dropping clients.

- before_fork and after_fork hooks in case your application has special needs when dealing with forked processes. These should not be needed when the "preload_app" directive is false (the default).
- Can be used with copy-on-write-friendly memory management to save memory (by setting "preload_app" to true).
- Able to listen on multiple interfaces including UNIX sockets, each worker process can also bind to a private port via the after_fork hook for easy debugging.
- Simple and easy Ruby DSL for configuration.
- Decodes chunked requests on-the-fly.

CHAPTER - 5
FEATURES AND REQUIREMENTS

This chapter discusses about features and requirements of the system that can solve the problem. This section considers the functional and nonfunctional requirements for the project. As the choice of development, the **Agile** method was chosen since the project's plan or project's requirements were likely to change and the testing results or customer's feedback were supposed to effect the project and also this helped to achieve **Rapid Application Development**.

5.1 USE CASES AND SYSTEM REQUIREMENT

The selected frameworks have been analyzed to discover the required system features. The following cases are necessary for the application:

- Login in to the application.
- Phone Number verification using otp.
- Age and Gender selection of user.
- Selection of what user wears on weekdays & weekends.
- Selection of body type and body complexion of user.
- Selection of what users have in their wardrobe as topwear, bottom wear and footwear.
- Allocating users corresponding stylists for personalised styling.
- Asking users at what time they want daily content and daily look for them.
- User can ask what to wear depending on a particular occasion and sub occasion.
- User can ask how they are looking on a particular outfit.
- Sending users daily look and daily content on their selected time slot.
- User and stylist can do real time chatting.
- Notifications on new messages.
- Calendar to store the timeline of suggestions for a user.
- 7 Day trial for newly signed up user giving them all the features for free .

- Payment for full version subscription of the app.
- Referral system for users.
- Personal virtual wardrobe for user where they can add or remove clothes.
- Admin panel for backend data entry.
- Users can ask what to buy.
- Users can browse some fashion content on the daily content feed.
- User feedback and rating system.

5.1.1 Login to the application

Use Case Name	Login to the application
Brief Description	The login process includes signing up users using google and facebook accounts.
Flow Of Events	1. User should click the sign in with google or sign in with facebook button. 2. The account selection dialog box appears where user can choose the respective account. 3. Authorization of the selected account. 4. Storing the important details for the user in the database and generating an API key for them.
Actors	User and Google And Facebook Auth API
Pre Conditions	A user must have a google or facebook account.
Post Conditions	User will be able to use rest of the features of the app. If the user is already logged in somewhere then he/she will not be able to use the features on the earlier login and will be logged out.
Exceptional Conditions	If the user enters wrong login details then the login process is cancelled and the user is notified.

Table 5.1

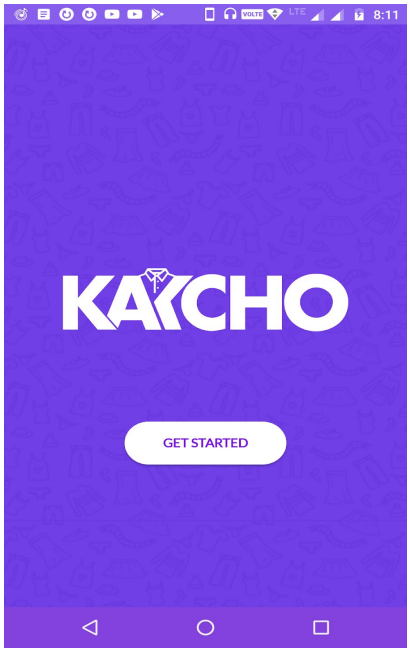


Fig 5.1 Splash Screen

Fig 5.2 Login Screen

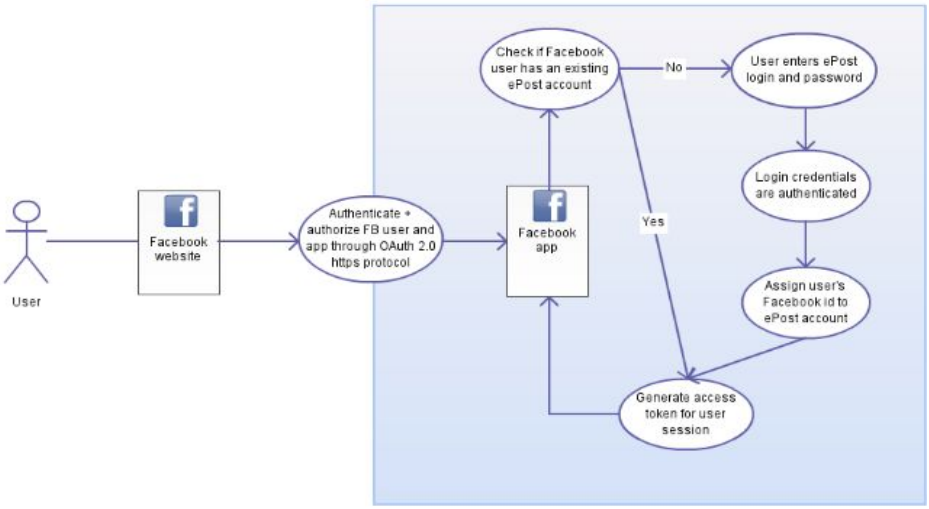


Fig 5.3 Use Case Diagram For Google/FB Login

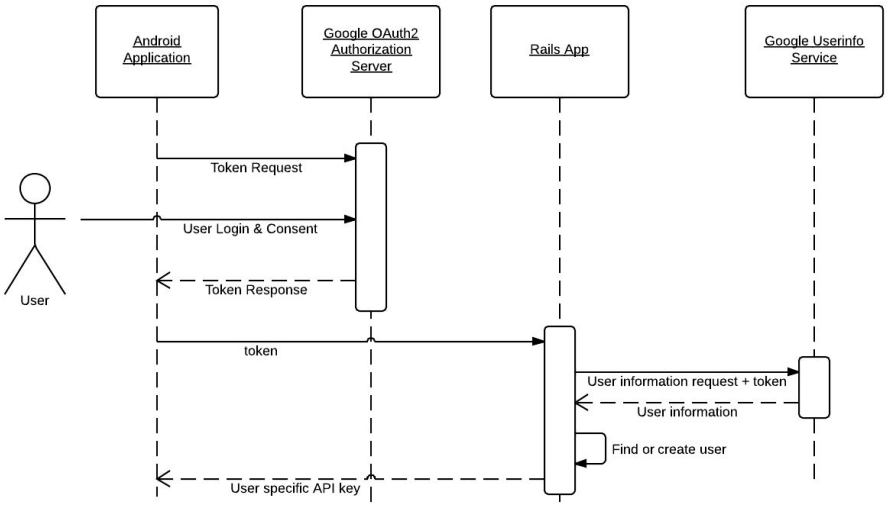


Fig 5.4 Sequence Diagram For Google/FB Login

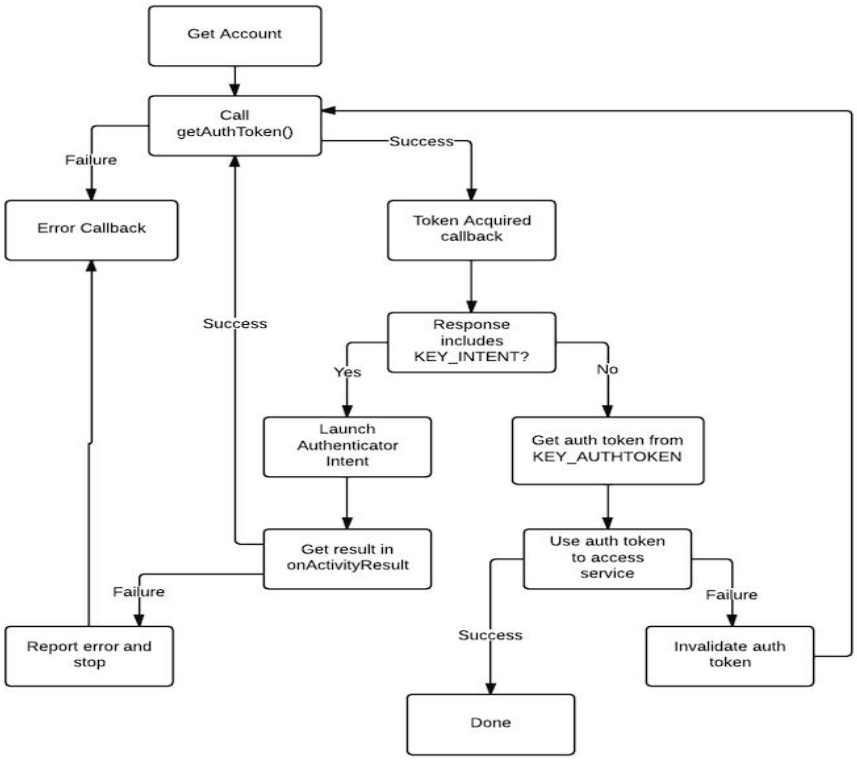


Fig 5.5 Data Flow Diagram For Google/FB Login

5.1.2 Phone number verification using OTP

Use Case Name	Phone number verification using OTP.
Brief Description	The verification process includes a otp verification of user's phone number.
Flow Of Events	1 After signing up successfully the user is redirected to the phone number verification screen. 2 The user then enters a 10 digit authorized and valid mobile number and press the verify button. 3 A request is generated to the backend where a secret 6 digit passcode is generated & sent to the users mobile number via SMS. 4 The user is then redirected to the enter otp screen where the otp is supposed to be written and the user presses the submit button. 5 After verification from backend that correct otp entered is correct the user can enter the next screen in the app. 6 If the phone number is already registered with some other account then the process should restart. 7 If the user doesn't receive an OTP then he/she can request to resend the otp.
Actors	User and SMSIndiaHub API
Pre Conditions	The users phone number must not be already available in the system. The user must be successfully signed in to the application.
Post Conditions	The user details in the backend gets the phone number along with it.
Exceptional Conditions	If a user enters an invalid number then the system notifies it. If the phone number is already registered with any other account then the user must be notified.

Table 5.2

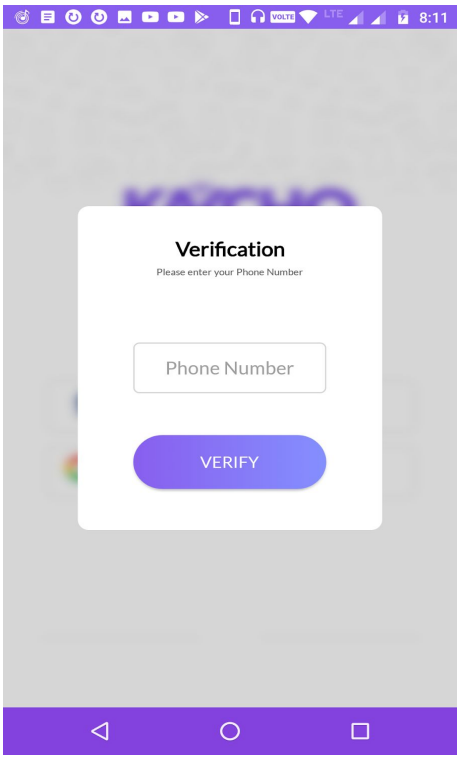


Fig 5.6 Mobile Verification Screen

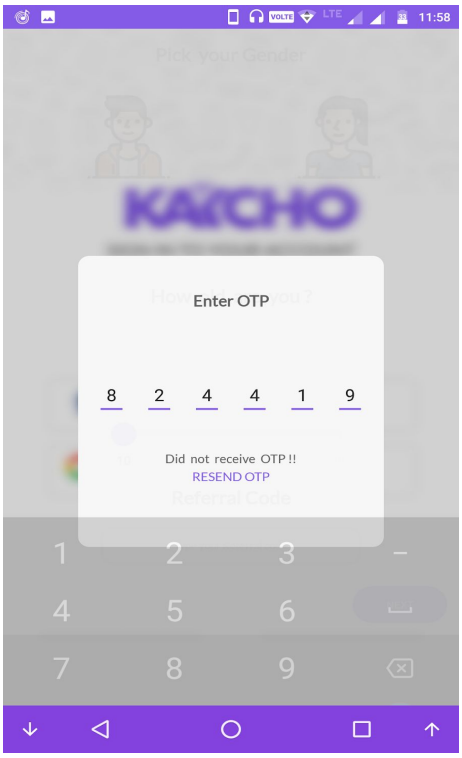


Fig 5.7 Otp Input Screen

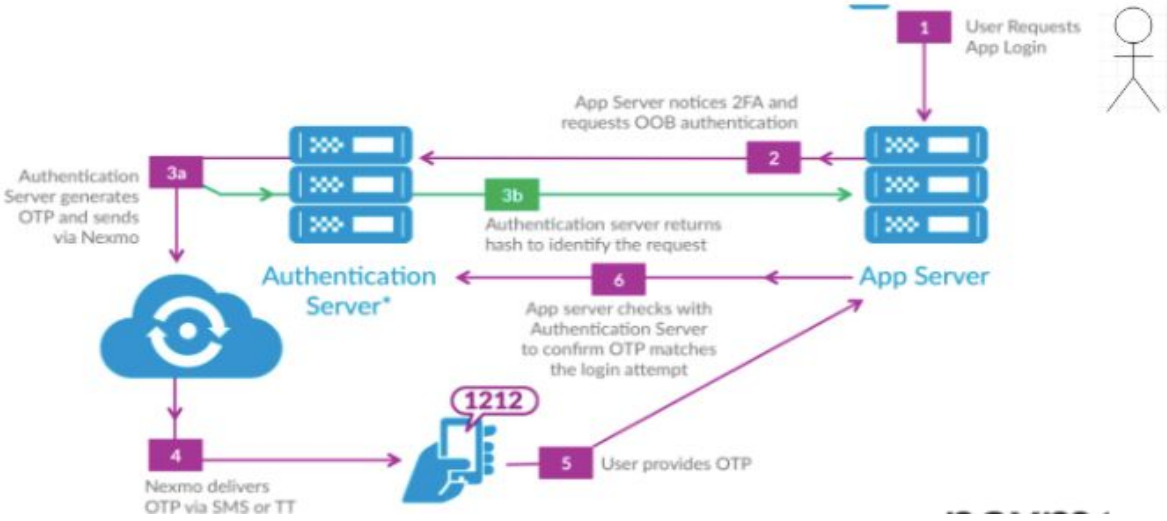


Fig 5.8 Use Case Diagram For 2 Factor OTP Verification

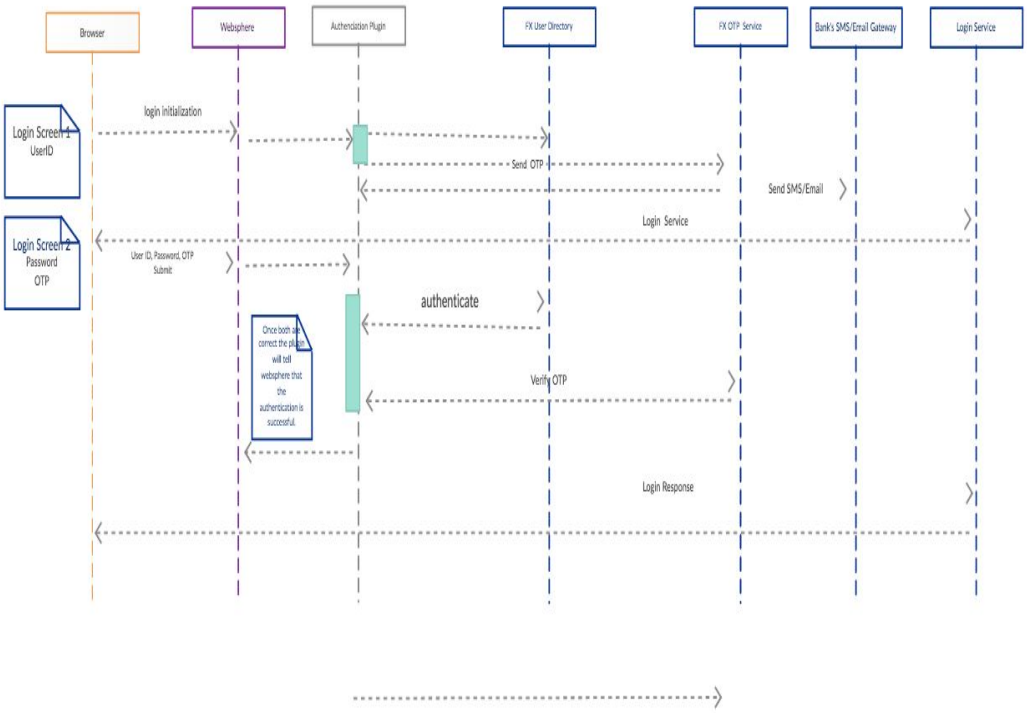


Fig 5.9 Sequence Diagram For 2 Factor OTP Verification

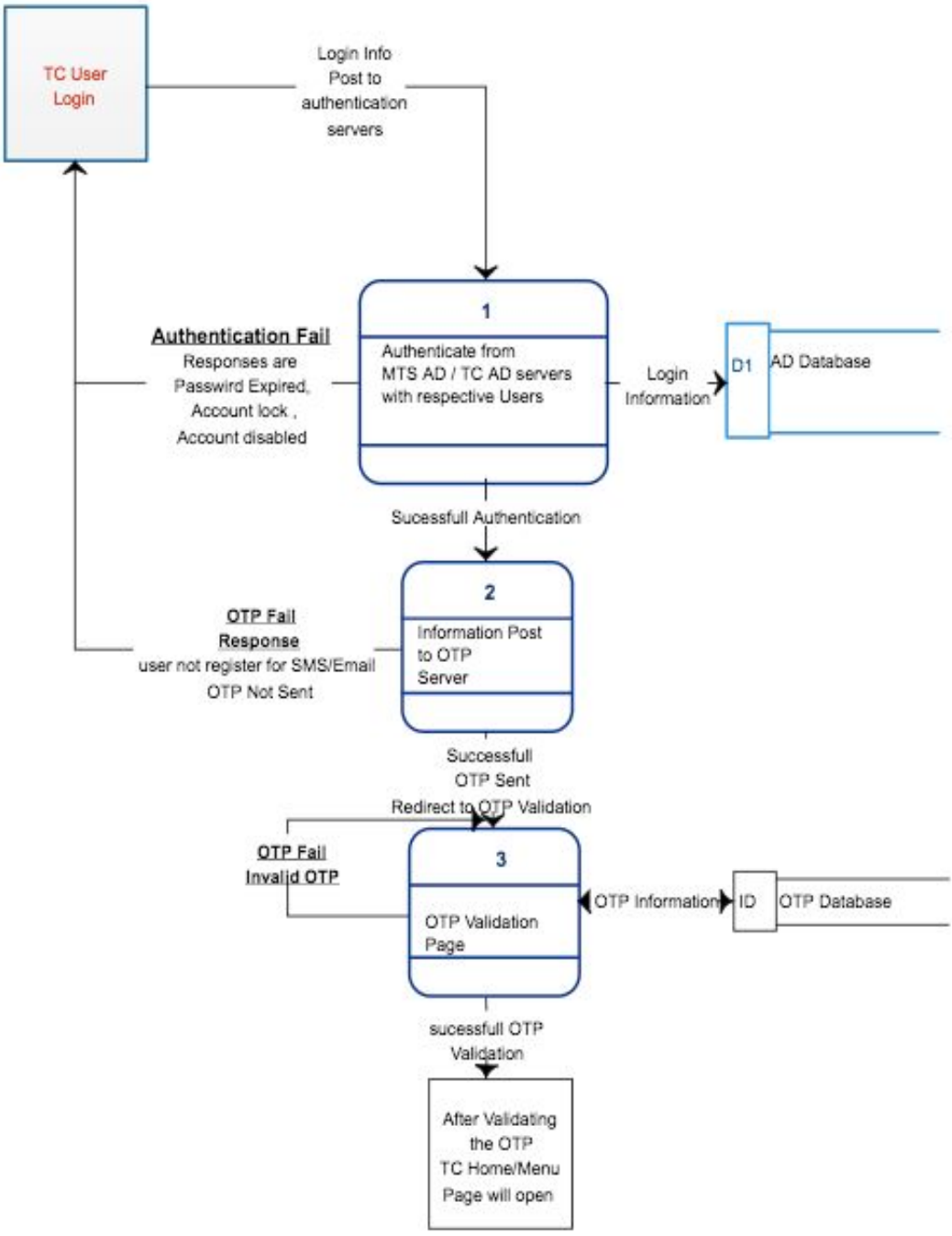


Fig 5.10 Data Flow Diagram For 2 Factor OTP Verification

5.1.3 Age and Gender selection of user

Use Case Name	Age and gender selection of user.
Brief Description	The user must select one gender and the current age.
Flow Of Events	1 After the otp verification user is redirected to the gender and age selection screen 2 User can choose either Male or Female as the gender. 3 User can select his/her age by sliding a slider on the appropriate age value.
Actors	User
Pre Conditions	The user details in the backend doesn't have a gender and an age. The user must be successfully signed in to the application.
Post Conditions	The user details in the backend gets the gender and age value with it.
Exceptional Conditions	If user doesn't selects a gender or doesn't enters an age value the he/she will be notified and will not be able to proceed.

Table 5.3

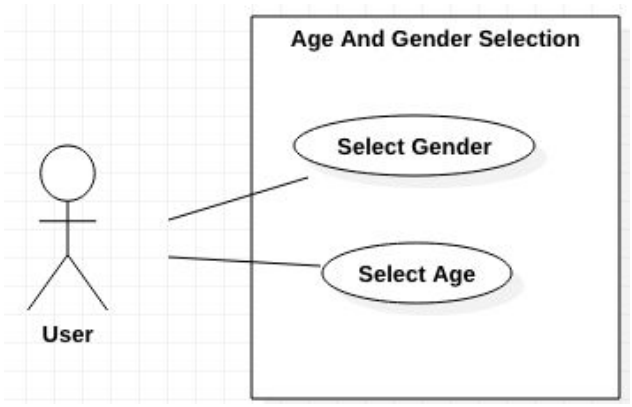


Fig 5.11 Use Case Diagram For Gender And Age Selection

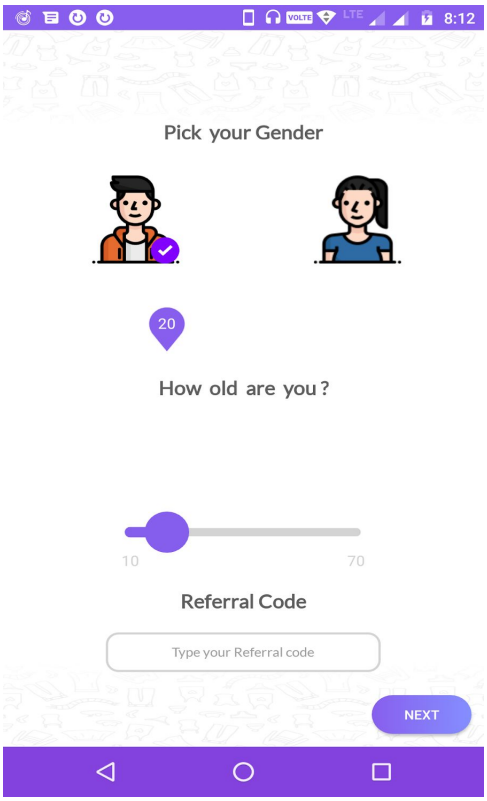


Fig 5.12 Gender And Age Selection Screen

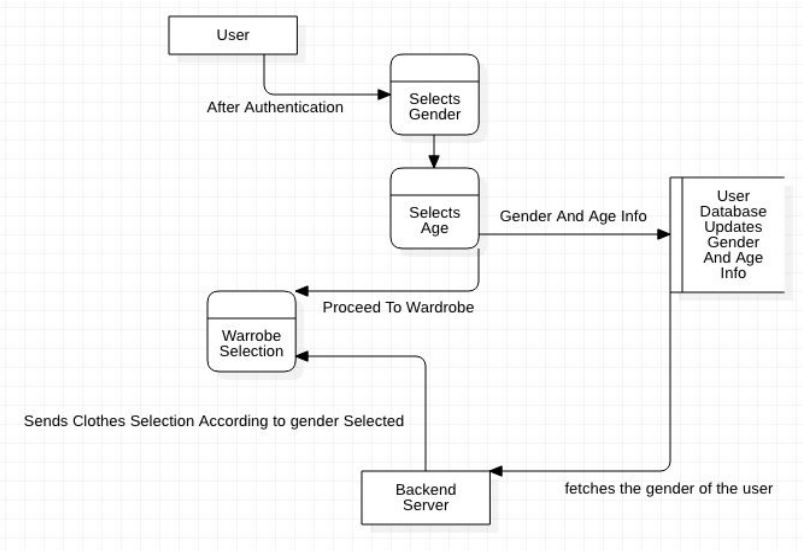


Fig 5.13 Gender And Age Selection Data Flow Diagram

5.1.4 Selection of body type and body complexion of user

Use Case Name	Selection of body type and body complexion of user
Brief Description	A user can select his/her body type from Skinny, Athletic, Average and bulky as the four available options. Also he/she can select body complexion from Fair, Medium, Tan and Dark as the four available options.
Flow Of Events	1 After the gender and age selection screen the user is redirected to the complexion, body type screen. 2 On this screen the user is asked about their body type and complexion. 3 The user must select one of the body type and one body complexion.
Actors	User
Pre Conditions	The user must be successfully signed in to the application.
Post Conditions	The user's choice on body type and body complexion is updated.
Exceptional Conditions	If the user has logged in first time and tries to escape this question then system notifies the user that he/she can't proceed without answering this.

Table 5.4

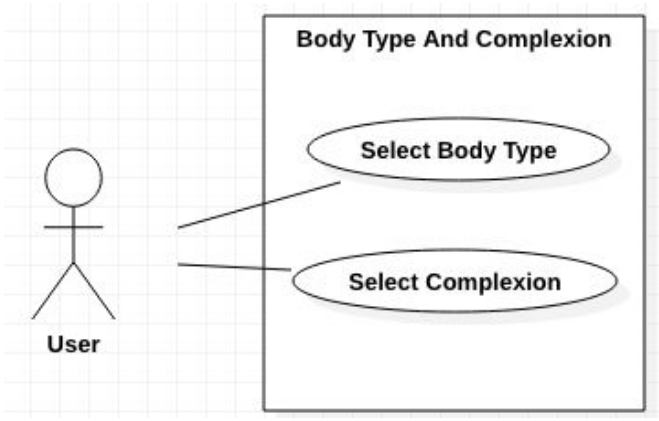


Fig 5.14 Use Case Diagram For Body Type And Complexion Selection

5.1.5 Selection of what user wears on weekdays & weekends

Use Case Name	Selection of what user wears on weekdays and weekends.
Brief Description	A user can choose either formal or semi formal or casual as a wear type for what he/she wears on weekdays and weekends.
Flow Of Events	1 After the gender and age selection screen the user is redirected to the complexion, body type screen. 2 On this screen the user is also asked what they wear on weekday and on weekend and three options are shown to them i.e. Formal, Semi Formal And Casual. 3 The user can select one or more than one wear type for both the questions.
Actors	User
Pre Conditions	The user must be successfully signed in to the application.
Post Conditions	The user's choice on what they wear is updated.
Exceptional Conditions	If the user has logged in first time and tries to escape this question then system notifies the user that he/she can't proceed without answering this.

Table 5.5

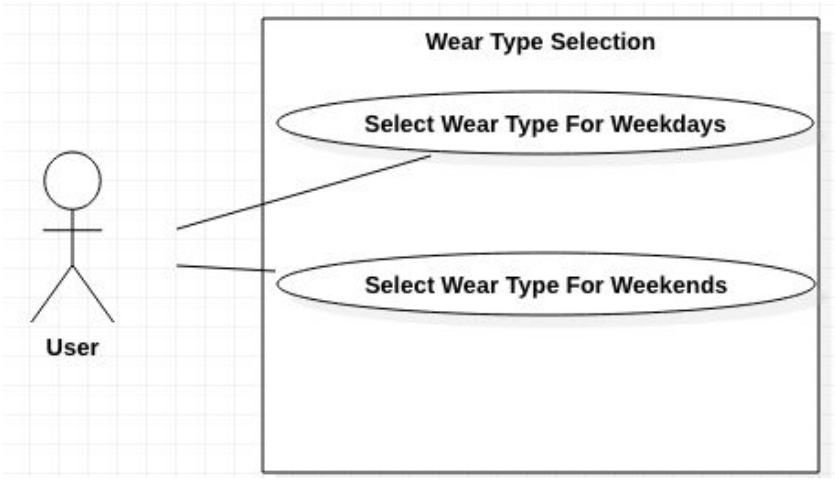


Fig 5.15 Use Case Diagram For Wear Type Selection

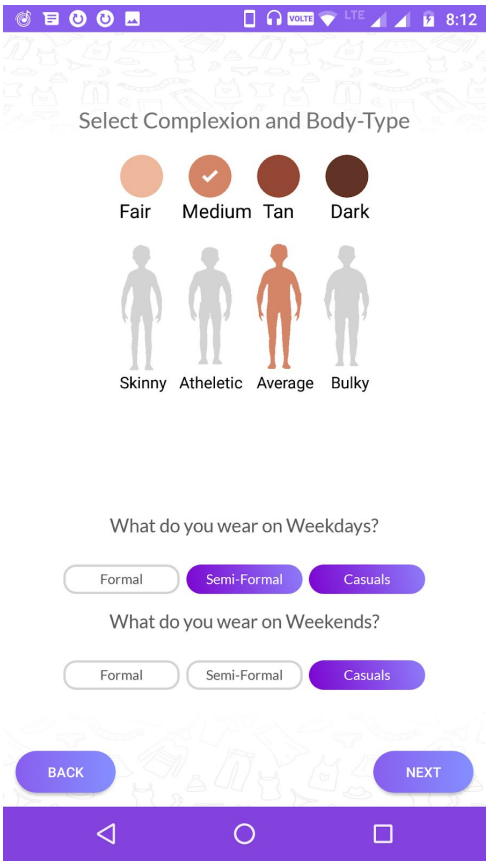


Fig 5.16 Body Type, Complexion And Wear Type Selection Screen

5.1.6 Selection of what users have in their wardrobe as top wear, bottom wear and footwear

Use Case Name	Selection of what users have in their wardrobe as top wear, bottom wear and footwear
Brief Description	A user can choose what top wear, bottom wear and footwear he/she has in their wardrobe from a pool of essential and common wears that have a high probability of availability in the wardrobe of any person.
Flow Of Events	1 After the complexion, body type screen user is redirected one by one to 3 different screens. 2 On each screen the user is shown a pool of essential and common wears depending on the screen (i.e. screen for top wear, bottom wear and footwear). 3 The user can select one or more than one wear for each wear type by tapping on the wear or by drag select on multiple wears. 4 User must select at least 3 wears for each category.
Actors	User
Pre Conditions	The user must be successfully signed in to the application.
Post Conditions	The user's choice on their wardrobe is updated.
Exceptional Conditions	If the user has logged in first time and tries to escape this question then system notifies the user that he/she can't proceed without answering this. If user select less than three wears for any category then system notifies the user that they can't proceed without selecting at least 3 wears.

Table 5.6

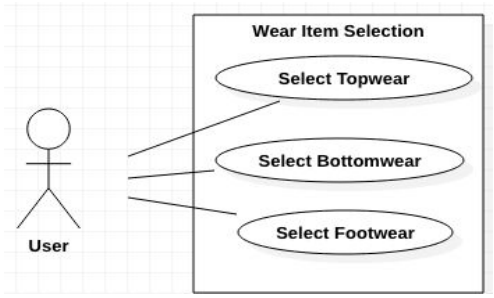


Fig 5.17 Use Case Diagram For Wear Item Selection

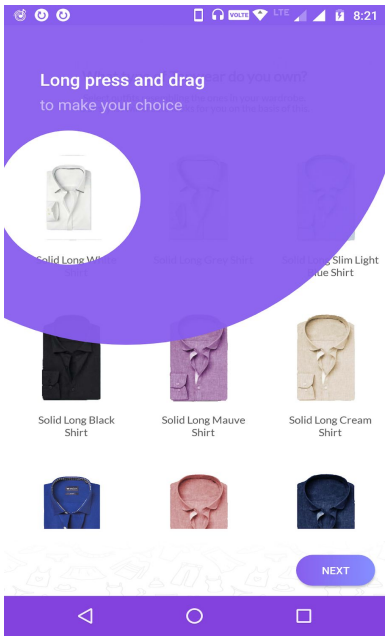


Fig 5.18 Top Wear Selection Screen

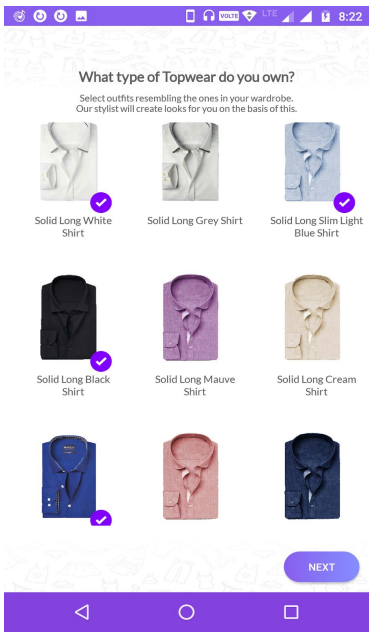


Fig 5.19 Top Wear Selection Screen

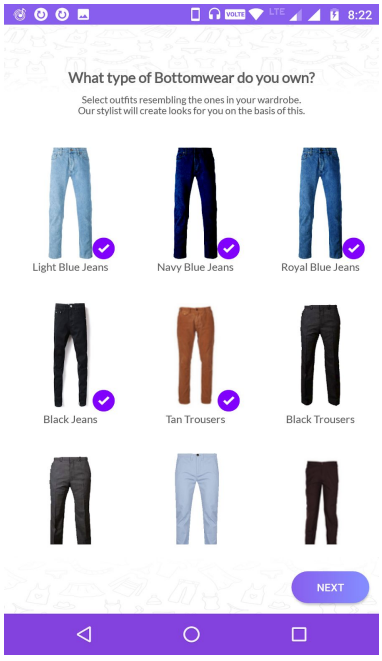


Fig 5.20 Bottom Wear Selection Screen

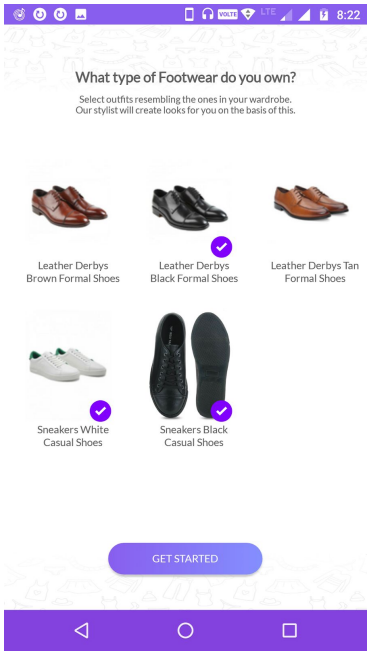


Fig 5.21 Footwear Selection Screen

5.1.7 Allocating users corresponding stylists for personalised styling

Use Case Name	Allocating users one stylist for personalised styling.
Brief Description	Every user who signed up in the app will be allocated a personal stylist for personalised advice.
Flow Of Events	1. After the user successfully login to the app for the first time then he/she is allocated a personal stylist. 2. If an already existing user logs in to the app then the already assigned stylist is again allocated to him/her. 3. If a stylist discontinues his/her service at kakcho then all the users associated with that stylist gets updated with a new stylist.
Actors	None
Pre Conditions	The user must be successfully signed in to the application.
Post Conditions	The user is allocated a stylist
Exceptionas	None

Table 5.7

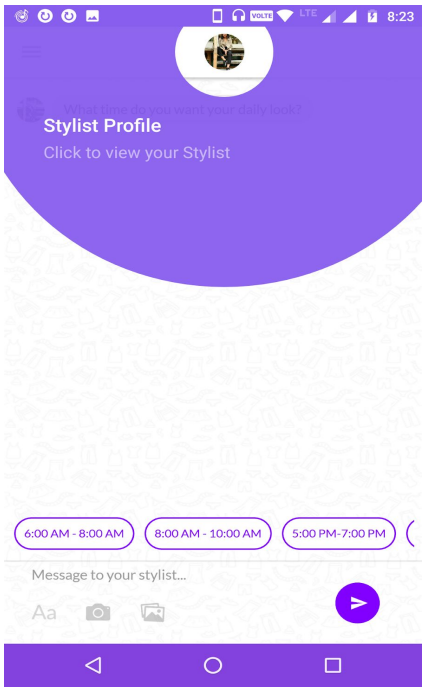


Fig 5.22 Chat Landing Screen

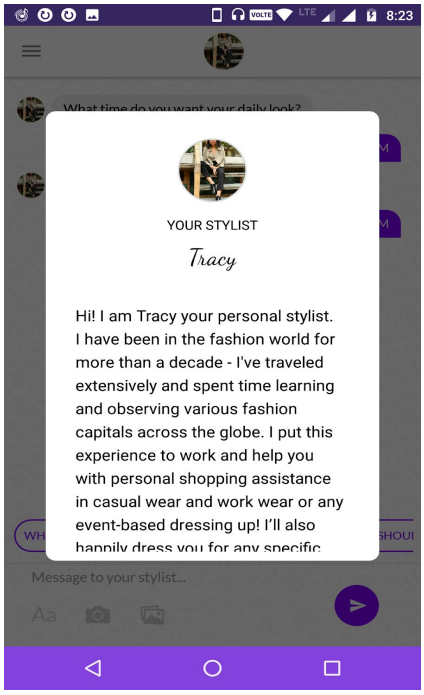


Fig 5.23 Stylist Profile Screen

5.1.8 Asking users at what time they want daily content and daily look for them

Use Case Name	Asking users at what time they want daily content and daily look
Brief Description	Every user who signed up in the app will be able to get daily look suggestion and daily fashion content on the time of their preference.
Flow Of Events	1 After the user successfully login to the app for the first time then he/she will be asked about their time preferences in the chat interface only with time bubbles shown above the input text bar. 2 If an already existing user logs in to the app then no such question will be prompted. 3 User will be sent the content and look between the time range they selected. 4 User can update the time from their profile section.
Actors	User
Pre Conditions	The user must be successfully signed in to the application.
Post Conditions	The user data is stored at the backend.
Exceptional Conditions	User has to select one time slot for each question otherwise he/she will be prompted the same question every time he/she restarts the app.

Table 5.8

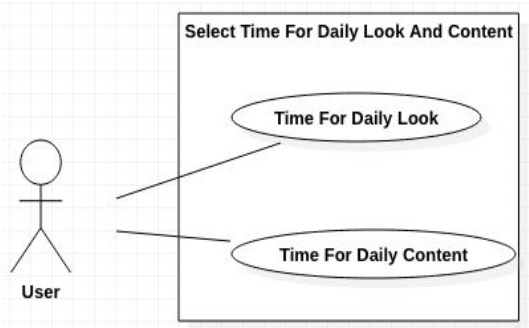


Fig 5.24 Case Diagram For Time Selection

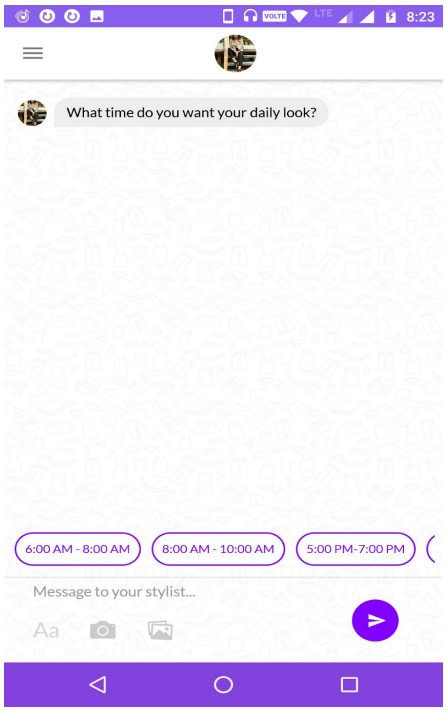


Fig 5.25 Chat With Dailylook Question

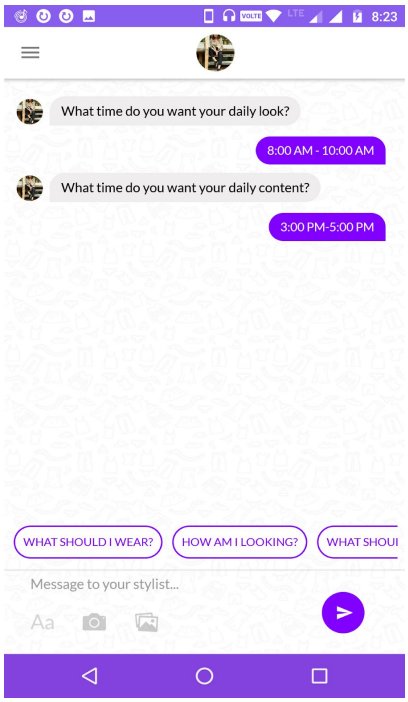


Fig 5.26 Questions Answered In The Chat

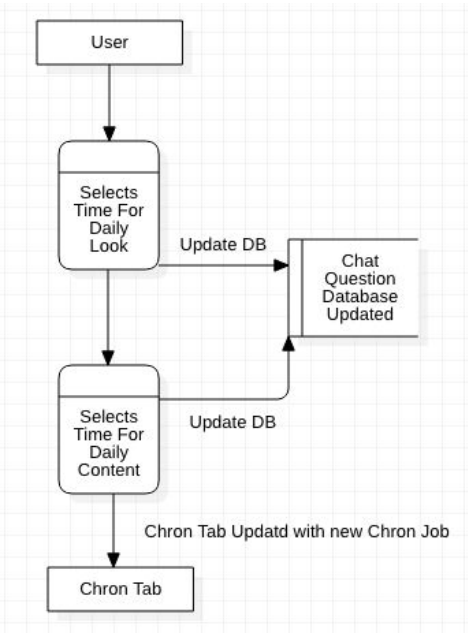


Fig 5.27 Data Flow Diagram For Chat Questions

5.1.9 User can ask what to wear depending on a particular occasion and sub occasion

Use Case Name	User can ask for what to wear on a particular occasion and sub occasion
Brief Description	Every user who signed up in the app will be able to ask for what to wear on a particular occasion and sub occasion
Flow Of Events	1 After the user successfully login to the app he/she can click on the what to wear button to ask a suggestion on what to wear. 2 Then a list of occasions will be prompted to the user and the user has to choose one occasion among the group. 3 Then a list of sub occasions will be prompted to the user and the user has to choose one sub occasion among the group. 4 Then the user will be suggested one look keeping in mind about his/her wardrobe, body type, age, complexion etc. 5 After sending a particular look the system asks whether user likes the look or not , if users selects “yes”, then the flow stops with a thank you message but if a user selects “no” then the system sends one more look and the same goes on for four times whenever user selects “no”.
Actors	User
Pre Conditions	The user must be successfully signed in to the application and must be a paid user.
Post Conditions	The user data is stored at the backend.
Exceptional Conditions	User has to select one time slot for each question otherwise he/she will be prompted the same question every time he/she restarts the app.

Table 5.9

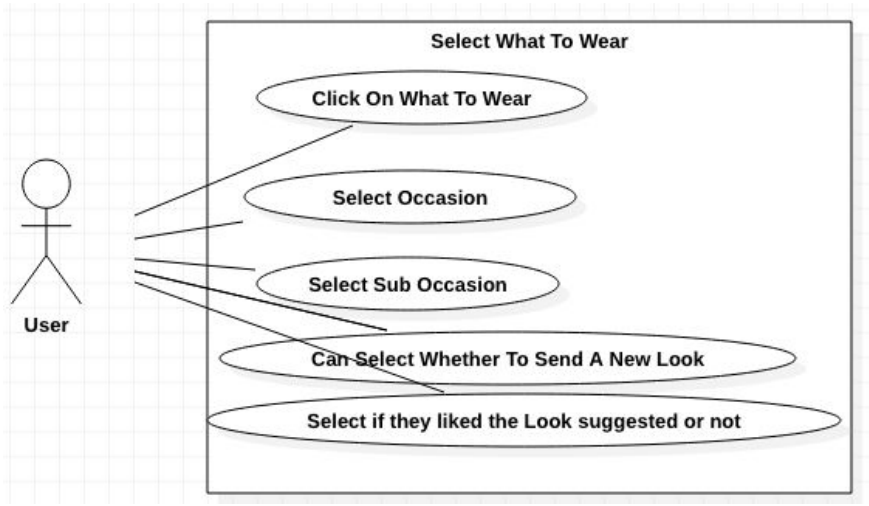


Fig 5.28 Use Case Diagram For What To Wear

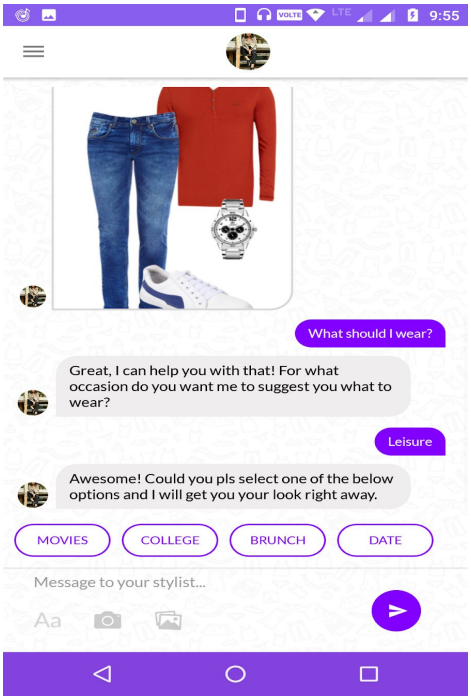


Fig 5.29 What Should I Wear Flow

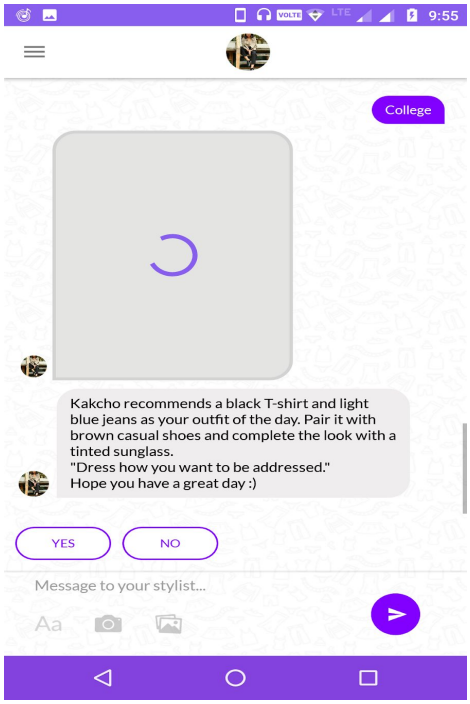


Fig 5.30 First Suggestion Sent

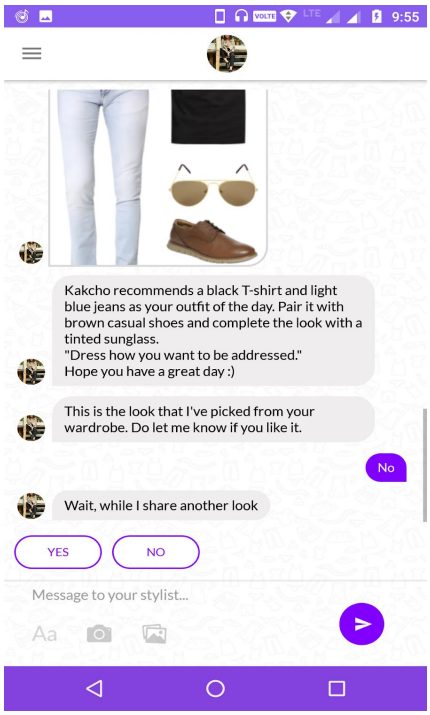


Fig 5.31 Another Look Sent

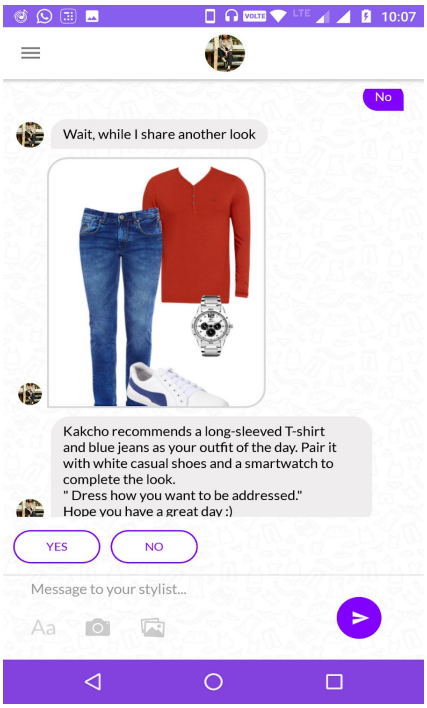


Fig 5.32 Another Look Sent

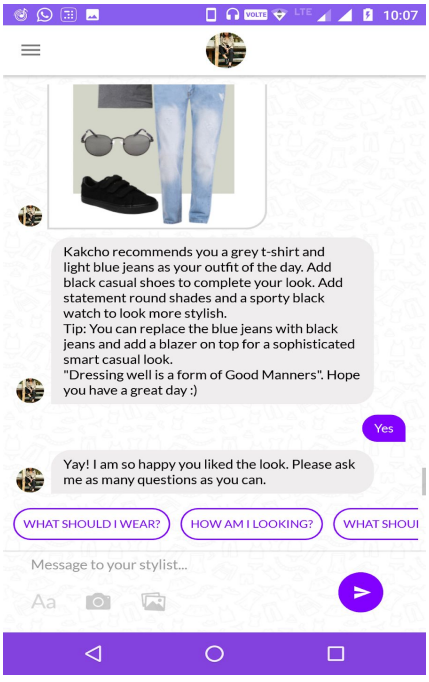


Fig 5.33 What To Wear Flow Completed

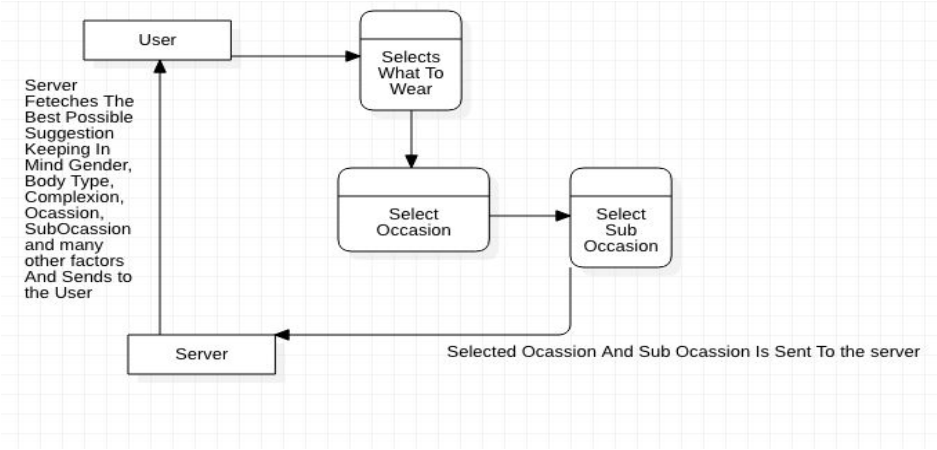


Fig 5.34 Data Flow Diagram What To Wear

5.1.10 User can ask how they are looking on a particular outfit

Use Case Name	User can ask how they are looking on a particular outfit
Brief Description	Every user who signed up in the app will be able to ask for how they are looking on a particular outfit from the allocated stylist
Flow Of Events	1 After the user successfully login to the app he/she can click on the “how am i looking” button to ask a how good or bad they are looking on a particular outfit. 2 Then the user will be asked to upload a picture of his/her wearing the outfit they want a suggestion on. 3 Then the user can either click on the camera button to upload a new picture or can click the gallery button to select a picture from the gallery. 4 Then the user will send the desired picture to the stylist in the chat. 5 Then stylist will review the user’s picture and give a suggestion.
Actors	User, Stylist
Pre Conditions	The user must be successfully signed in to the application and must be a paid user.The user must give camera and gallery access to the android application
Post Conditions	None
Exceptions	None

Table 5.10

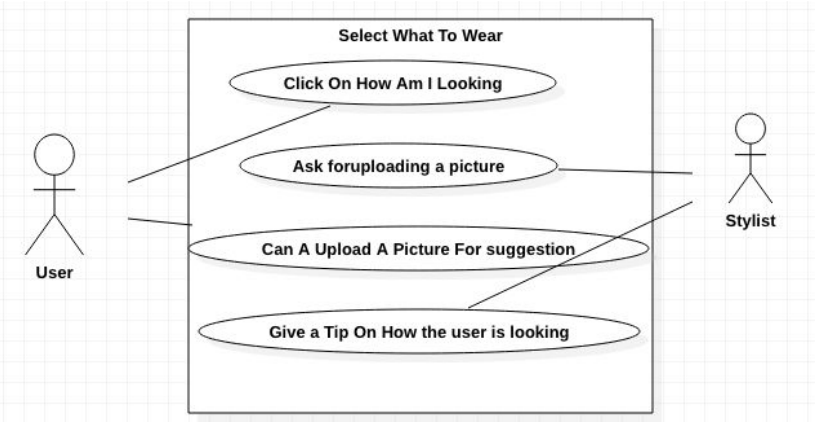


Fig 5.35 Use Case Diagram For How Am I Looking

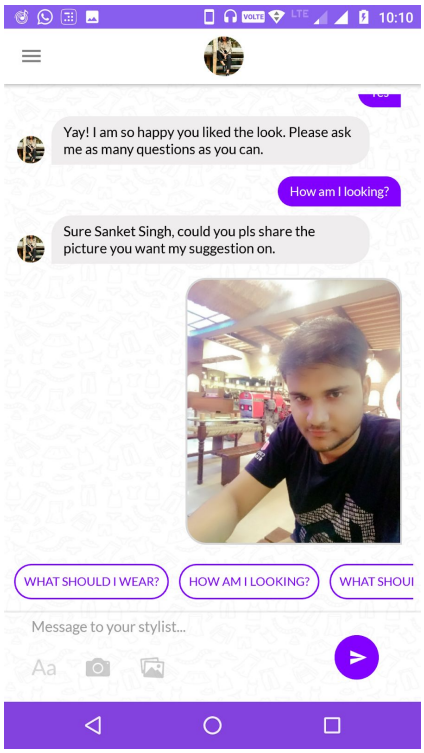


Fig 5.36 How Am I Looking Screen

5.1.11 Sending users daily look and daily content on their selected time slot

Use Case Name	Sending users daily look and daily content on their selected time slot
Brief Description	Every user who signed up in the app and answered the question for time to deliver the daily content and daily look will be sent the corresponding daily look and daily content on the time slot selected.
Flow Of Events	1. The user will get the daily content and daily look on the selected time slots keeping in mind about what they wear on weekdays and weekend. 2. If the user is not active on the app then notifications will also be sent to him/her.
Actors	None
Pre Conditions	The user must be successfully signed in to the application and must be a paid user.The user must answered the questions on daily look and daily content.
Post Conditions	None
Exceptional Conditions	If a user's free trial is over or his/her paid subscription has ended then the system will notify the user to repay for the subscription to use the service

Table 5.11

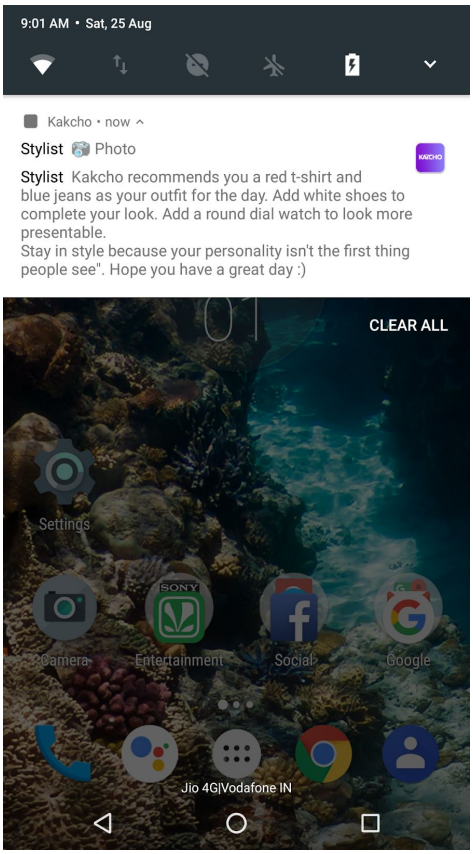


Fig 5.37 Notification For Daily Look

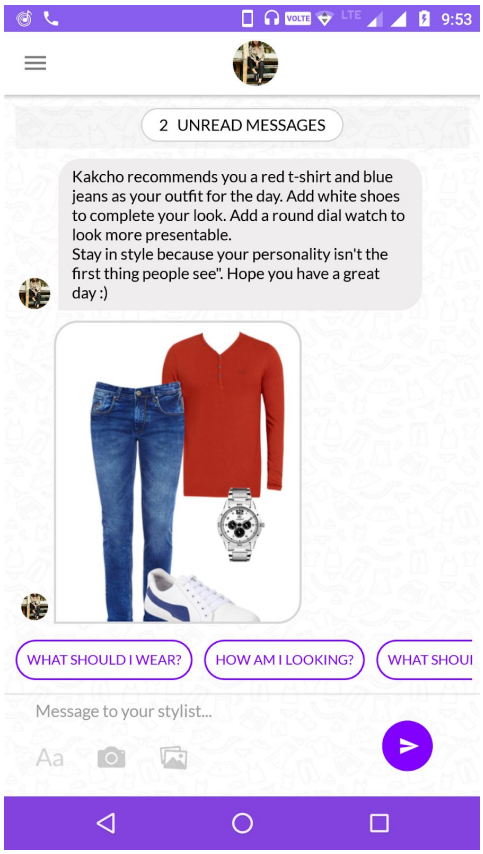


Fig 5.38 Daily Look On Chat Screen

5.1.12 User and stylist can do real time chatting

Use Case Name	User and stylist can do real time chatting
Brief Description	Every user who signed up in the app can chat with the allocated stylist in real time.
Flow Of Events	1 The user and stylist can exchange text messages between each other. 2 The user and stylist can also exchange images as attachments between them 3 Both can do chatting in real time. 4 If a user or stylist is not available at the other end and new messages comes then whenever they open the chat they can get notified about new messages. 5 If a user is not active in the app then he/she will get a notification of the new message.
Actors	User, Stylist
Pre Conditions	The user must be successfully signed in to the application.
Post Conditions	None
Exceptional Conditions	User will be unable to send a text message beyond a particular limit.

Table 5.12

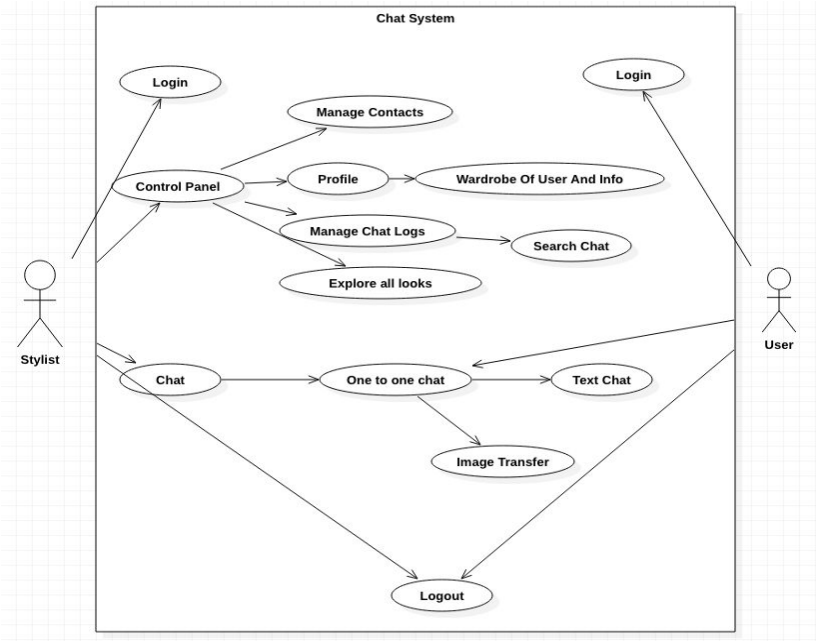


Fig 5.39 Use Case Diagram For Chatting

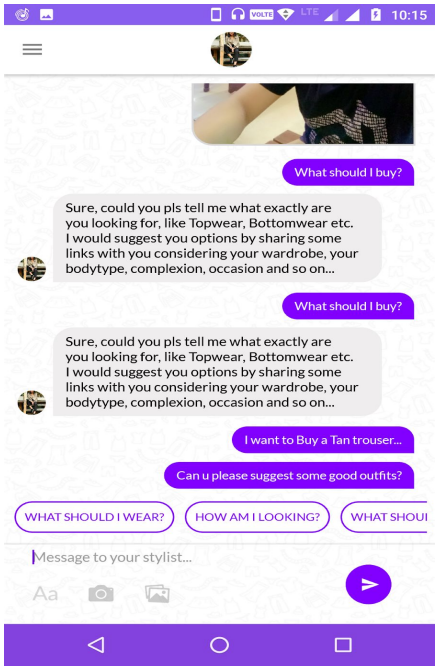


Fig 5.40 Chatting Screen

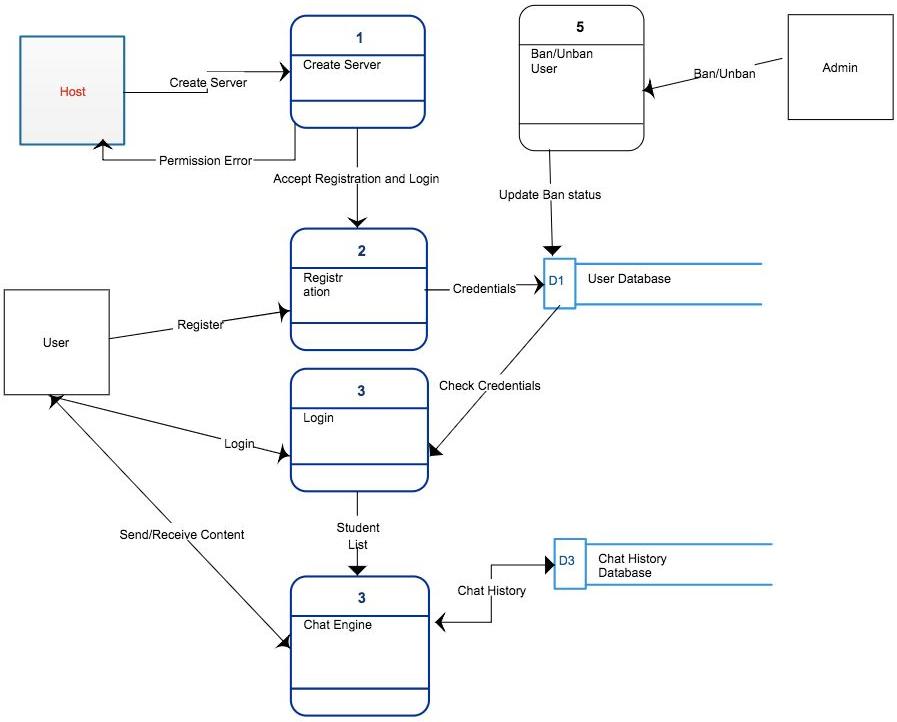


Fig 5.41 Data Flow Diagram For Chat System

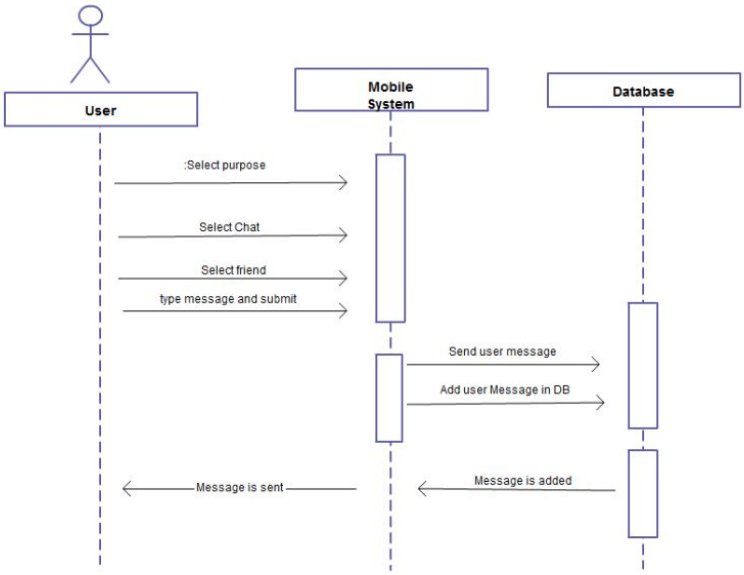


Fig 5.42 Sequence Diagram For Chat System

5.1.13 Notifications on new messages

Use Case Name	Notifications on new messages
Brief Description	Every user will get notification on new messages when they have app in the background.
Flow Of Events	1 The user will get notifications on new messages in the chat
Actors	User
Pre Conditions	The user must be successfully signed in to the application.
Post Conditions	None
Exceptional Conditions	None

Table 5.13

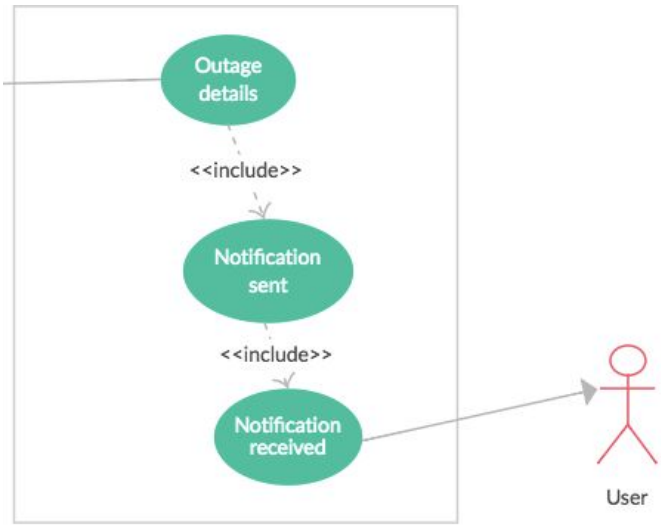


Fig 5.43 Use Case Diagram For Notifications

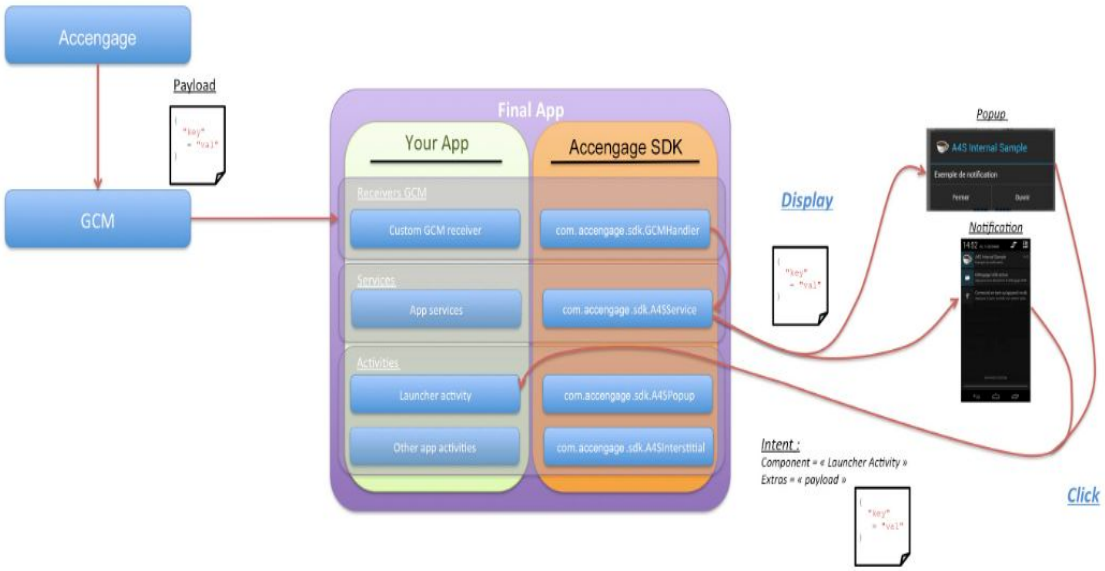


Fig 5.44 Data Flow Diagram For Android Notifications

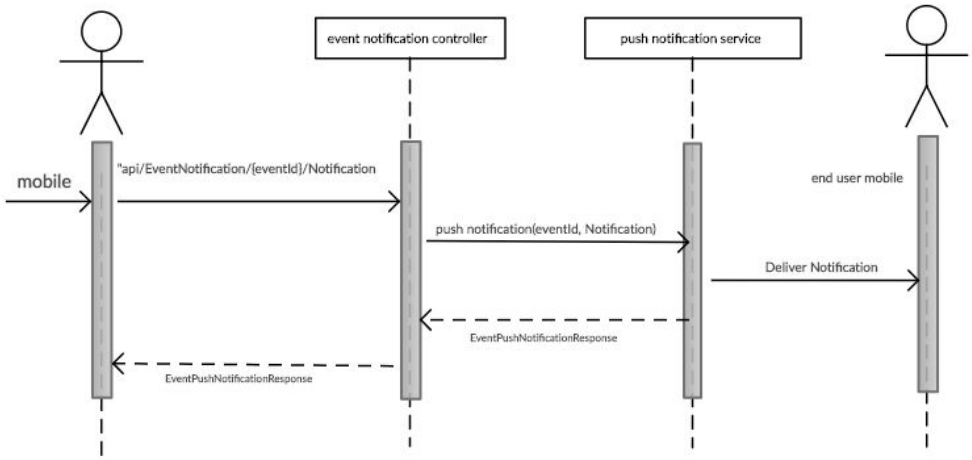


Fig 5.45 Sequence Diagram For Android Notifications

5.1.14 Seven Day trial for newly signed up user giving them all the features for free for 7 days

Use Case Name	Seven day trial period for newly signed up users
Brief Description	Every newly signed up used will get a free 7 day trial period where he/she can access all the features for free.
Flow Of Events	1 The user can initially access all the features for free for 7 days.
Actors	None
Pre Conditions	The user must be successfully signed in to the application and must be newly signed up.
Post Conditions	None
Exceptional Conditions	If the user's email is already registered then no such trial period will be provided.

Table 5.14

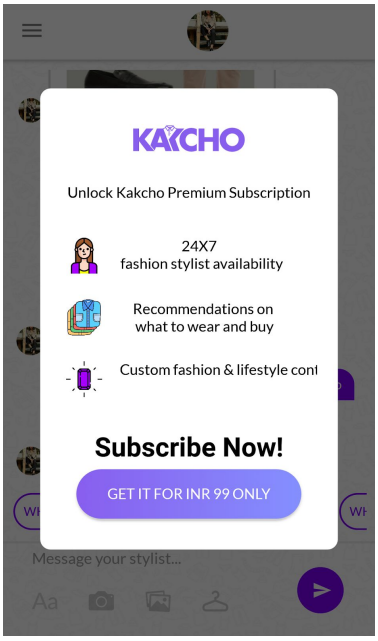


Fig 5.46 Seven Day Trial Expired Popup Screen

5.1.15 Payment for full version subscription of the app

Use Case Name	Payment for full version subscription for the application.
Brief Description	A secure payment method must be integrated for user's subscription payment.
Flow Of Events	1 The user can select a subscription plan according to his/her need and click proceed. 2 Then a secure payment gateway of razorpay will be opened and the user can enter the details for payment and complete the subscription. 3 After a successful subscription the user will get an invoice for the payment. 4 The user status will be updated to paid in the backend. 5 The payment system must be recurring in nature i.e. the user should not have to pay again and again every month when the subscription ends. The payment should be automatically deducted from the account. 6 If a user wants to discontinue the subscription then he/she must be able to disable the recurring deduction of the payment.
Actors	User, Razorpay
Pre Conditions	The user must be successfully signed in to the application. User must have a valid payment method available i.ee net banking, credit card, debit card.
Post Conditions	User's subscription will be updated to paid according to the plan.
Exceptional Conditions	If the user's payment method is invalid due to any reason then the system notifies user about the error. If a user has successfully subscribed a plan then he cannot get refund of the current cycle of subscription but can disable renewal of subscription.

Table 5.15

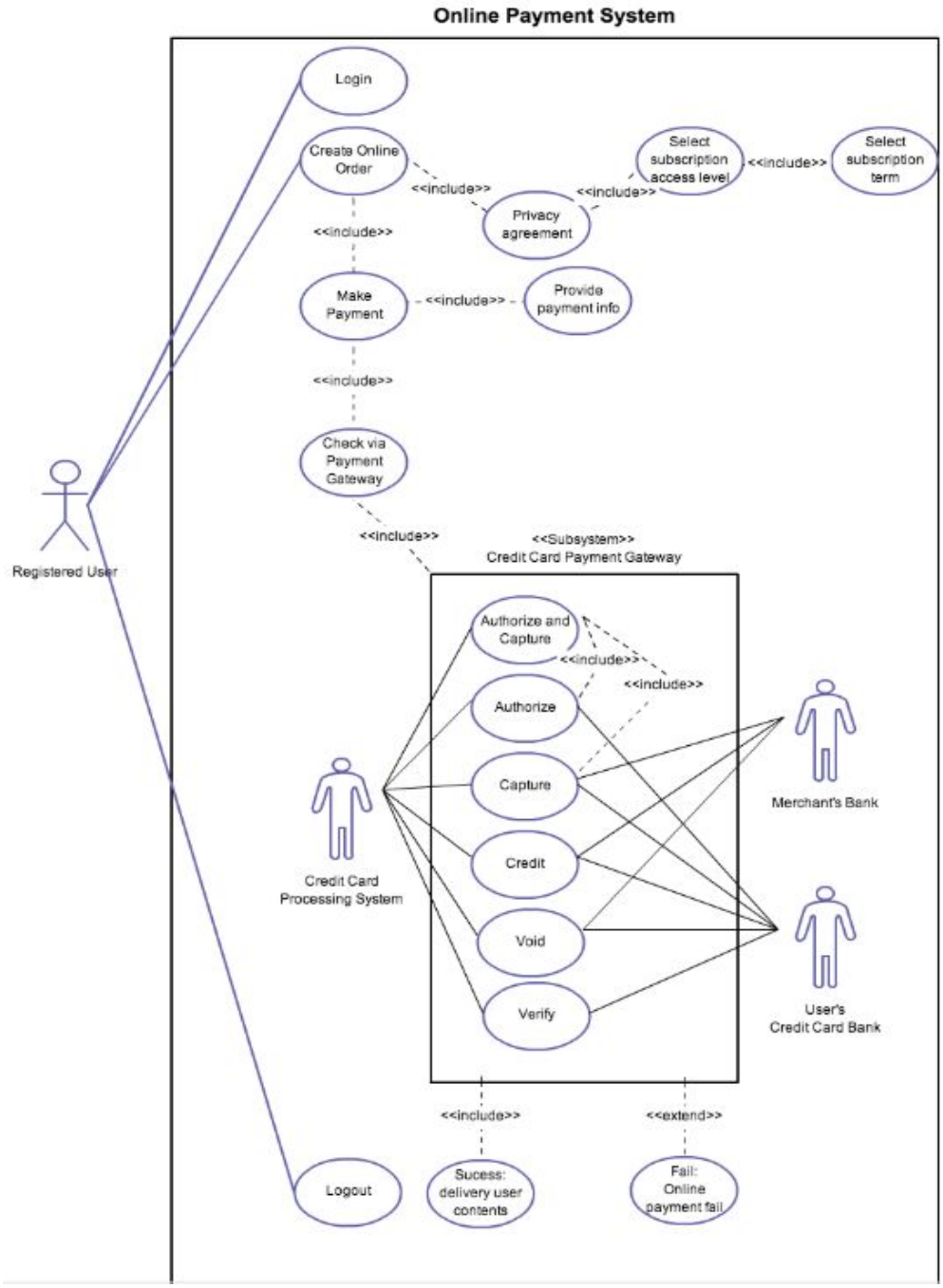


Fig 5.47 Use Case Diagram for payment

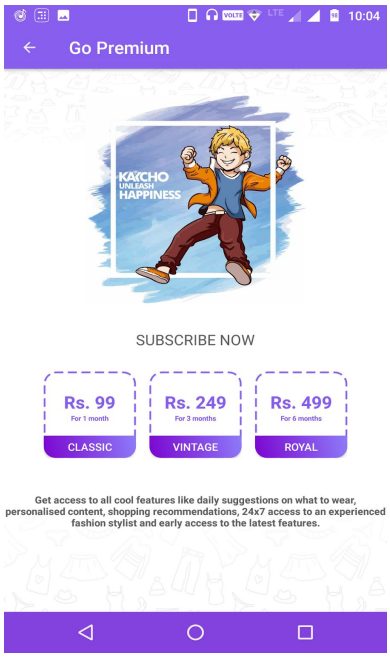


Fig 5.48 Go Premium Screen

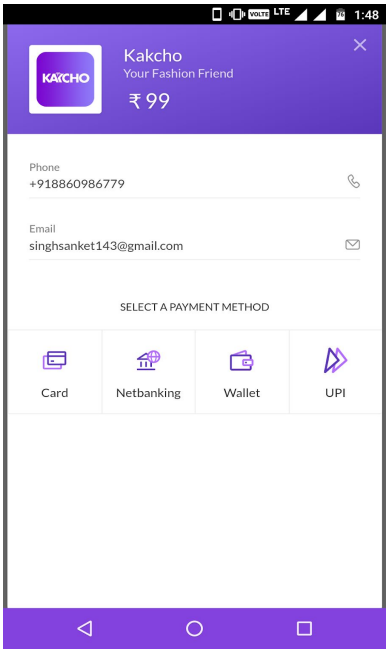


Fig 5.49 Razorpay Screen Step 1

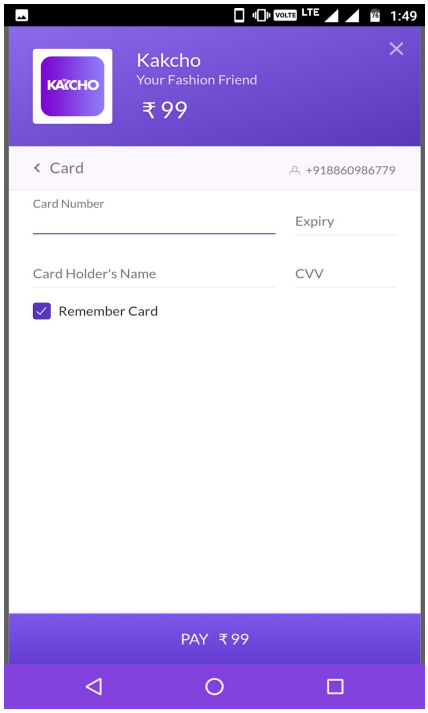


Fig 5.50 Razorpay Screen Step 2

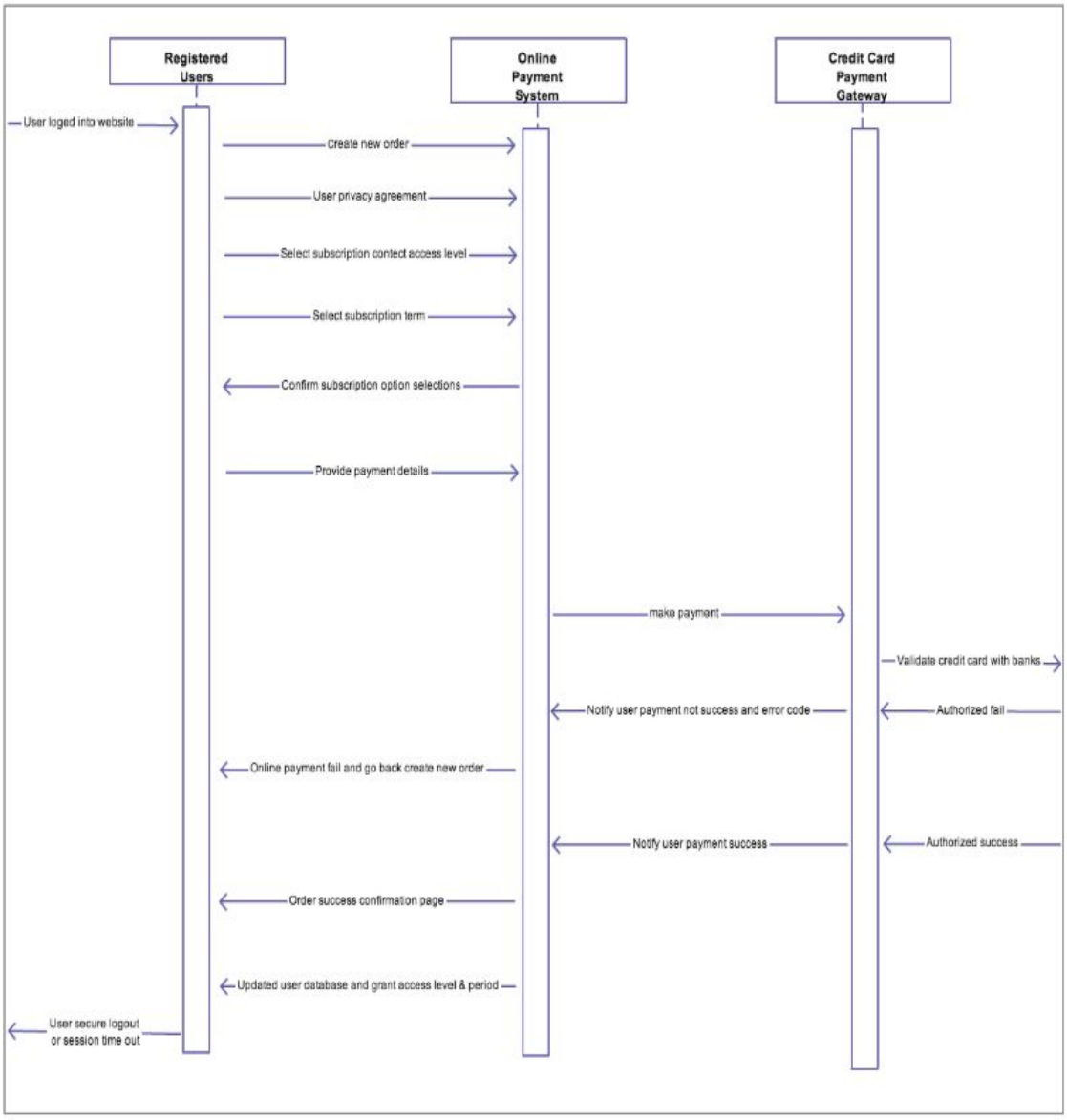


Fig 5.51 Sequence Diagram For Online Payment

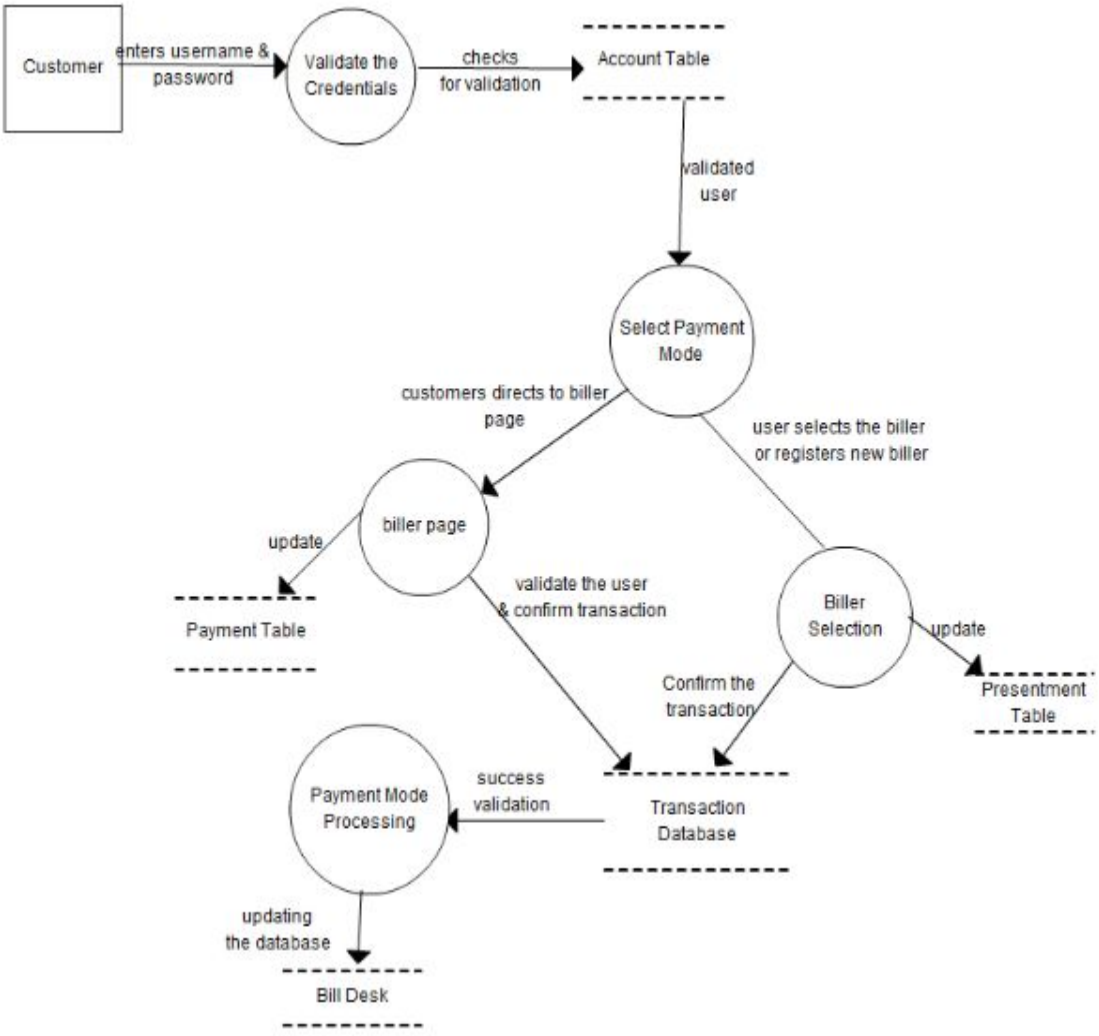


Fig 5.52 Data Flow Diagram For Online Payment

5.1.16 Referral system for users

Use Case Name	Referral system
Brief Description	A referral system must be added in the application to give some sers special free subscription depending on how many referrals they got.
Flow Of Events	1 Every user who successfully signed up in the app is provided a special and unique referral code. 2 Every new user is asked to enter any referral code on the gender and age selection screen. 3 If 5 users adds a referral code of same user, the the user will get free one month subscription to all the features of the app. 4 Adding a referral code is not mandatory. 5 Users can also share their referral codes along with a custom message to other social platforms like messenger, whatsapp wrc.
Actors	User1, User 2
Pre Conditions	The user must be successfully signed in to the application.
Post Conditions	If a user gets 5 successful referrals then he/she will be rewarded free one month additional subscription to all the paid features of the app.
Exceptional Conditions	If the user add any wrong referral code then the system notifies the user to enter a valid referral code.

Table 5.16

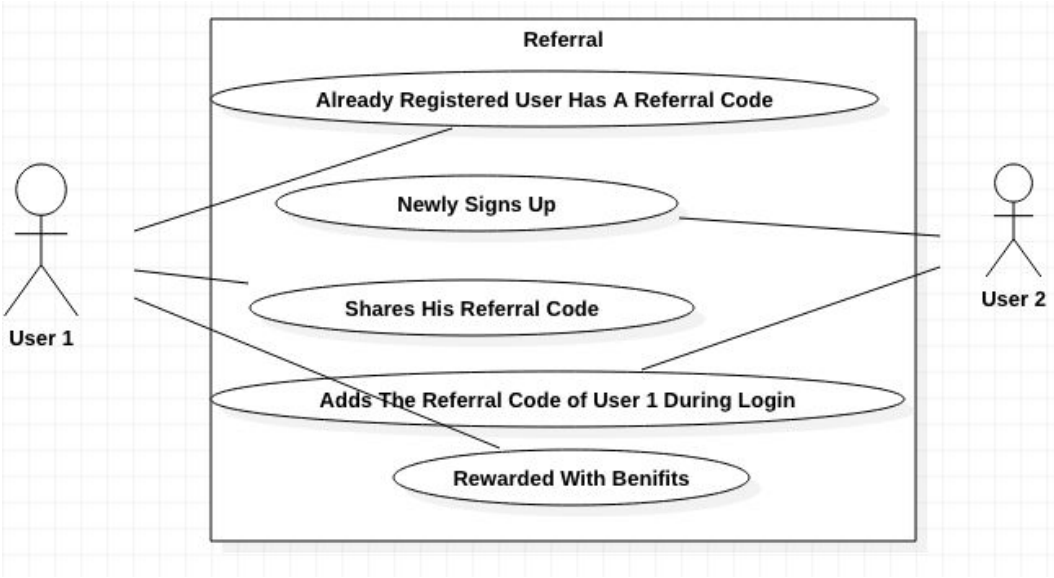


Fig 5.53 Use Case DIagram For Referral System

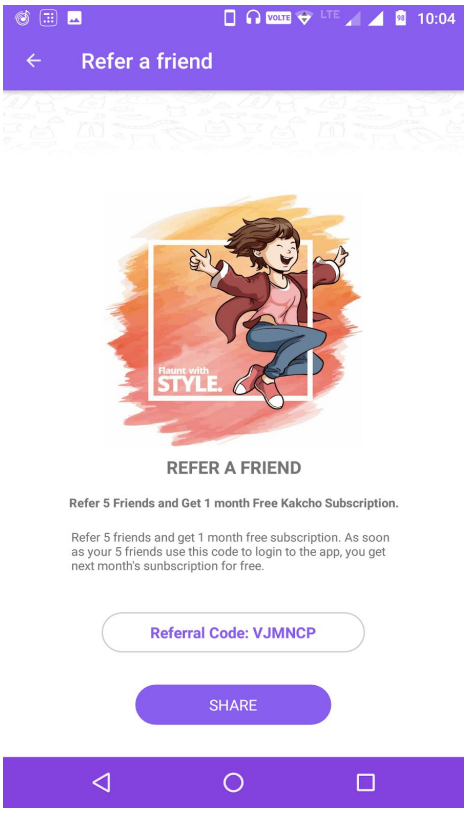


Fig 5.54 Refer A Friend Screen

5.1.17 Personal virtual wardrobe for user where they can add or remove clothes

Use Case Name	Personal virtual wardrobe
Brief Description	Every user who successfully signed in to the app will be given a virtual wardrobe where he/she can add or remove clothes according to their own original wardrobe.
Flow Of Events	1 Every user who successfully signed up in the app can navigate to the wardrobe section. 2 There he/she can see their already added clothes. 3 They can add or remove some clothes depending on their own wardrobe. 4 A better wardrobe information will lead to better suggestions and recommendations. 5 User can choose clothes from different categories of different wear types and the virtual wardrobe gets updated in real time.
Actors	User
Pre Conditions	The user must be successfully signed in to the application.
Post Conditions	None
Exceptional Conditions	None

Table 5.17

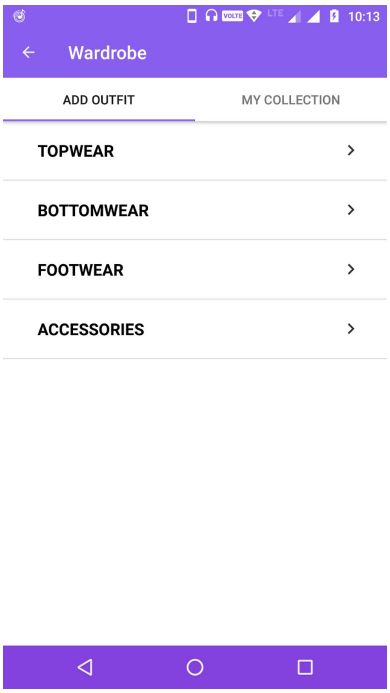


Fig 5.55 Wardrobe Screen

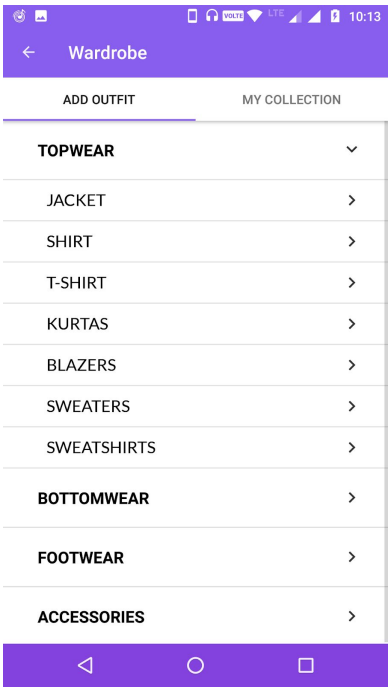


Fig 5.56 Wardrobe Screen With Weartypes

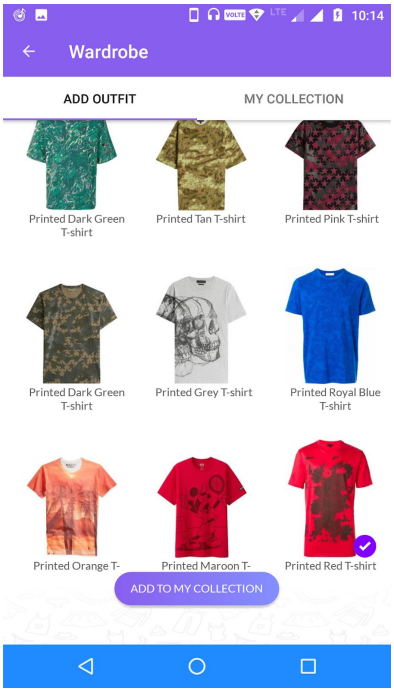


Fig 5.57 Selection On Wardrobe Screen

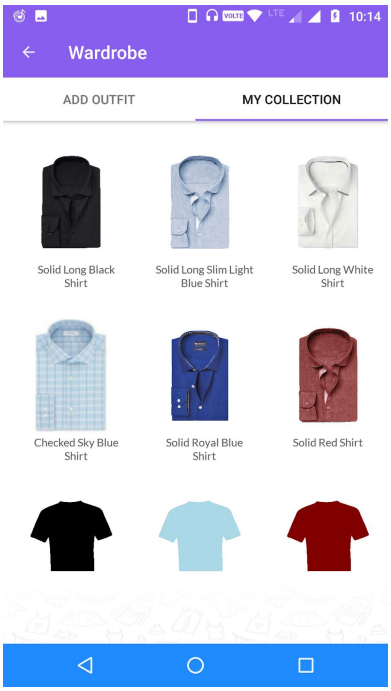


Fig 5.58 My Collection Screen

5.1.18 Users can ask what to buy

Use Case Name	Users can ask for what to buy
Brief Description	Every user who successfully signed in to the app and is a paid user can ask for shopping recommendations.
Flow Of Events	1 Every user who successfully signed up in the app and is a paid user can click on the what to buy button. 2 Then the system asks what budget the users has and what clothes he/she wants to purchase and of what color it should be. 3 Additional filters can also be integrated 4 Then user will get a recommendation on what clothes he/she can buy
Actors	User, Stylist
Pre Conditions	The user must be successfully signed in to the application and must be a paid user.
Post Conditions	None
Exceptional Conditions	None

Table 5.18

5.1.19 Users can browse some fashion content on the daily content feed

Use Case Name	Users can browse some fashion content and daily content feed
Brief Description	Every user who successfully signed in to the app can access the content feed
Flow Of Events	1 Every user who successfully signed in to the app can access the content feed. 2 The user can navigate to the content feed screen and can view the content and posts. 3 User can like or dislike a particular post. 4 User can also share a particular post on other social media platforms.
Actors	User
Pre Conditions	The user must be successfully signed in to the application
Post Conditions	None
Exceptional Conditions	None

Table 5.19

5.1.20 User feedback and rating system

Use Case Name	Users rating and review system.
Brief Description	Every user who successfully signed in to the app review and rate the app.
Flow Of Events	1 Every user who successfully signed in to the app can access the review screen of the app. 2 There they can select a particular feature and can then rate and review on a scale of 1-5.
Actors	User, Admin
Pre Conditions	The user must be successfully signed in to the application
Post Conditions	None
Exceptional Conditions	None

Table 5.20

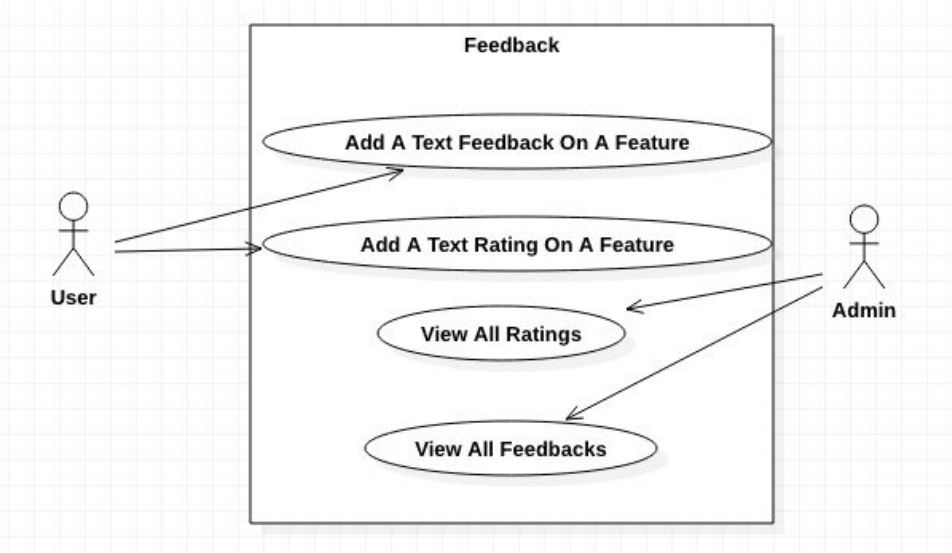


Fig 5.58 Use Case Diagram For Rating And Feedback

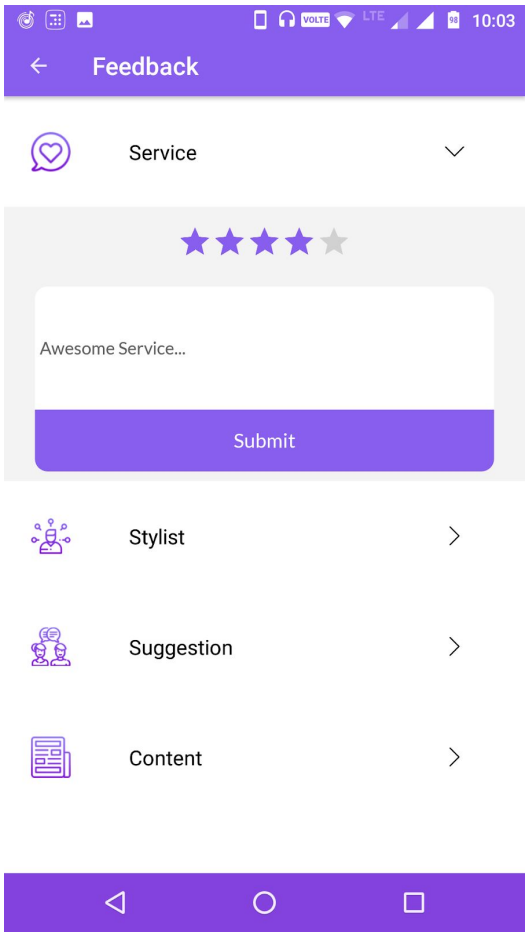


Fig 5.59 Rating And Feedback Screen

5.1.21 Calendar to store the timeline of suggestions for a user

Use Case Name	Calendar to store the timeline of suggestions for a user
Brief Description	Every user will get a calendar timeline containing all the suggested looks, looks they chose for specific occasions, and pictures they shared for getting suggestion on how they are looking.
Flow Of Events	1. The user can view the calendar section corresponding to their profile. 2. The calendar contains suggested looks, looks they chose for specific occasions, and pictures they shared for getting suggestion on how they are looking. 3. User can click any picture and get an expanded look of the picture. 4. The pictures are sorted according to the date when they are sent.
Actors	None
Pre Conditions	The user must be successfully signed in to the application and must be a paid user.
Post Conditions	None
Exceptional Conditions	None

Table 5.21

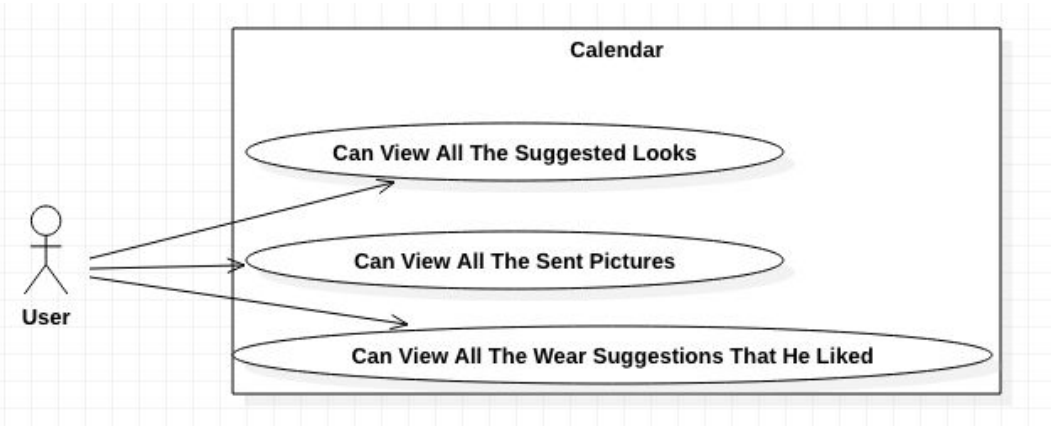


Fig 5.60 Use Case Diagram For Calendar

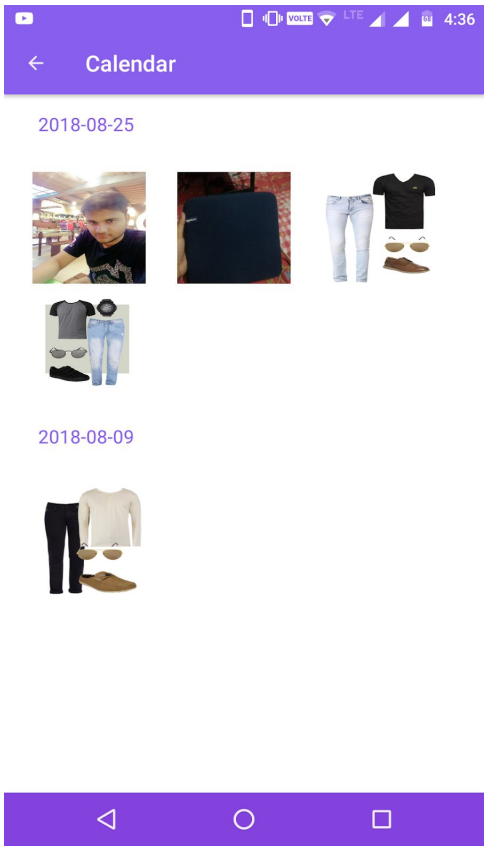


Fig 5.61 Calendar Screen

5.1.22 Admin Panel

Use Case Name	Admin Panel For Admins And Stylist
Brief Description	Every stylist and admin can access the admin panel for chat application and data entry services.
Flow Of Events	1. The stylist or admin can view the admin panel and and the chat panel. 2. They can manipulate and update the data for various services
Actors	Stylist, Admin
Pre Conditions	The stylist or admin must be successfully signed in to the application .
Post Conditions	None
Exceptional Conditions	None

Table 5.22

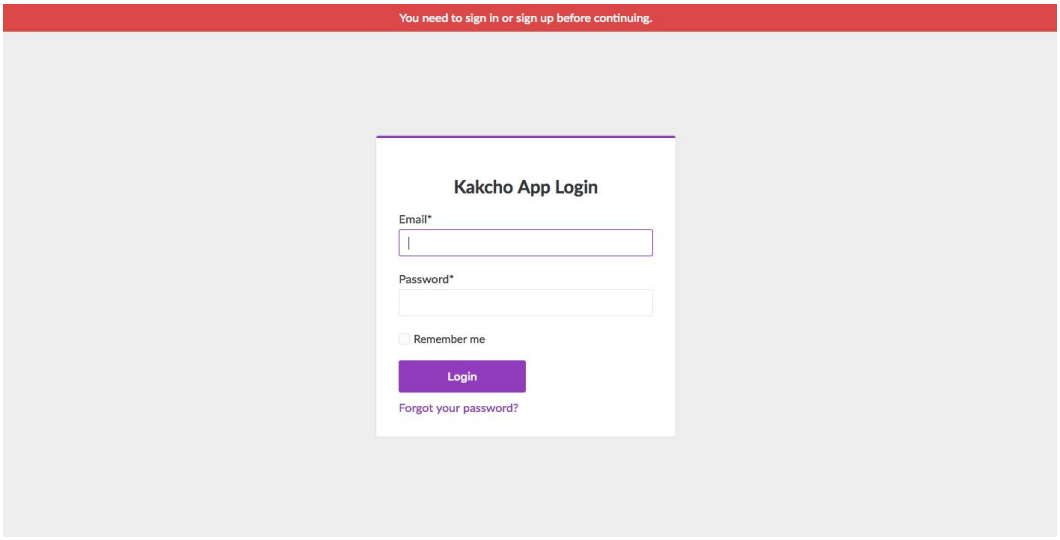


Fig 5.62 Admin Panel Login

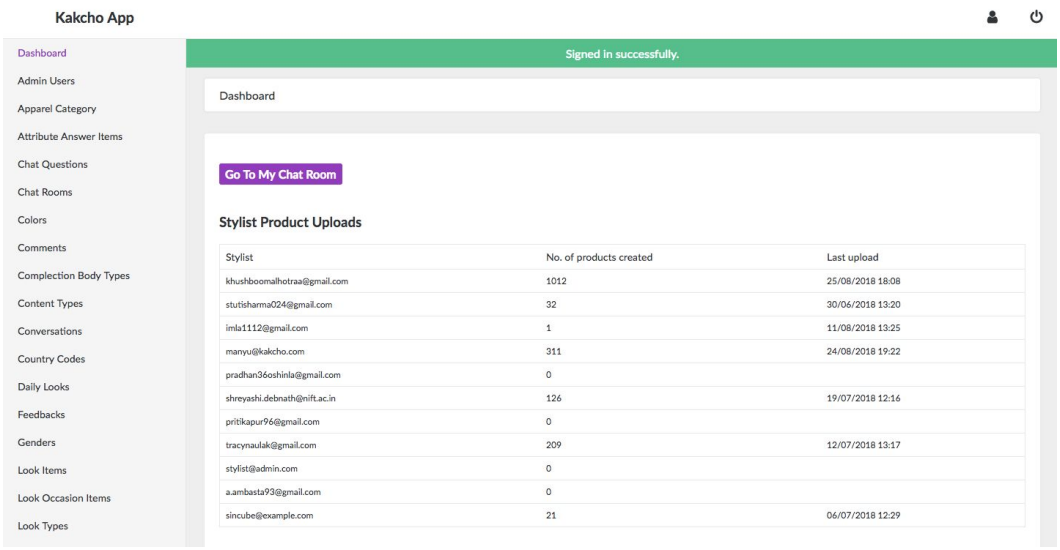


Fig 5.62 Admin Panel Dashboard

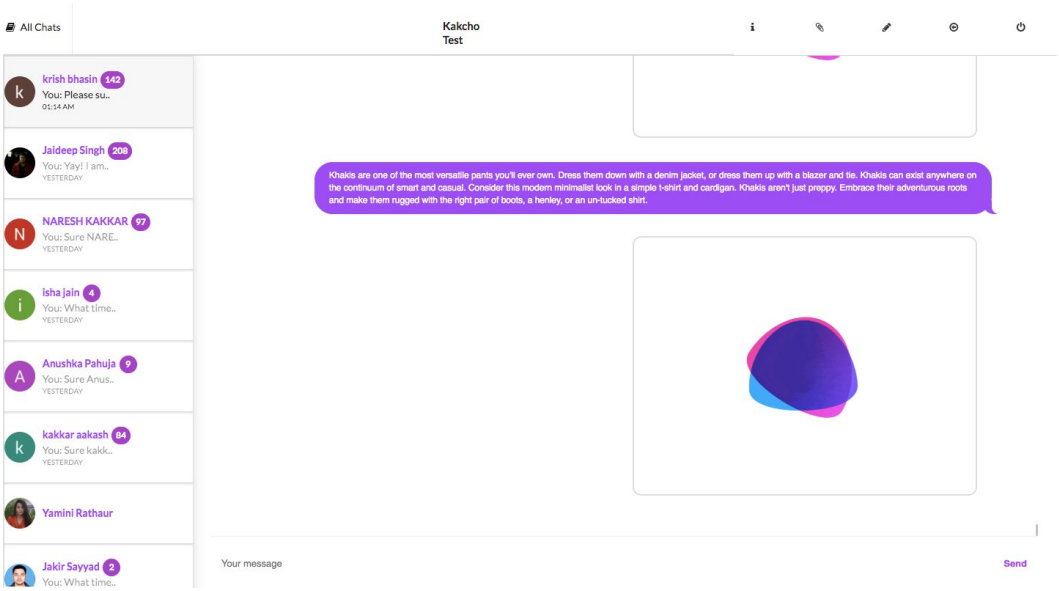


Fig 5.63 Admin Chat Panel

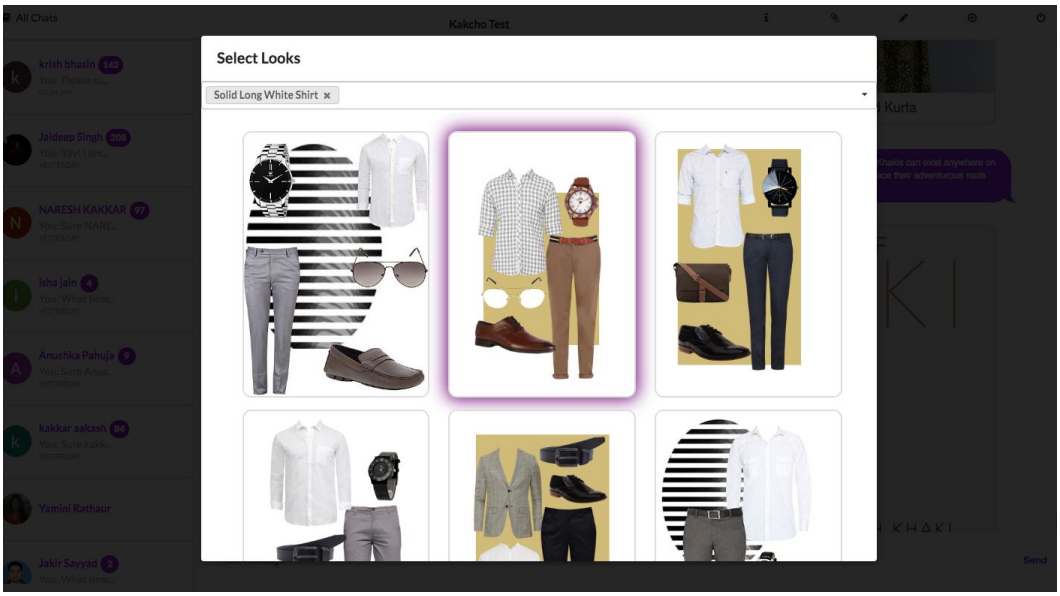


Fig 5.64 Admin Chat Panel Look Suggesting Dialog

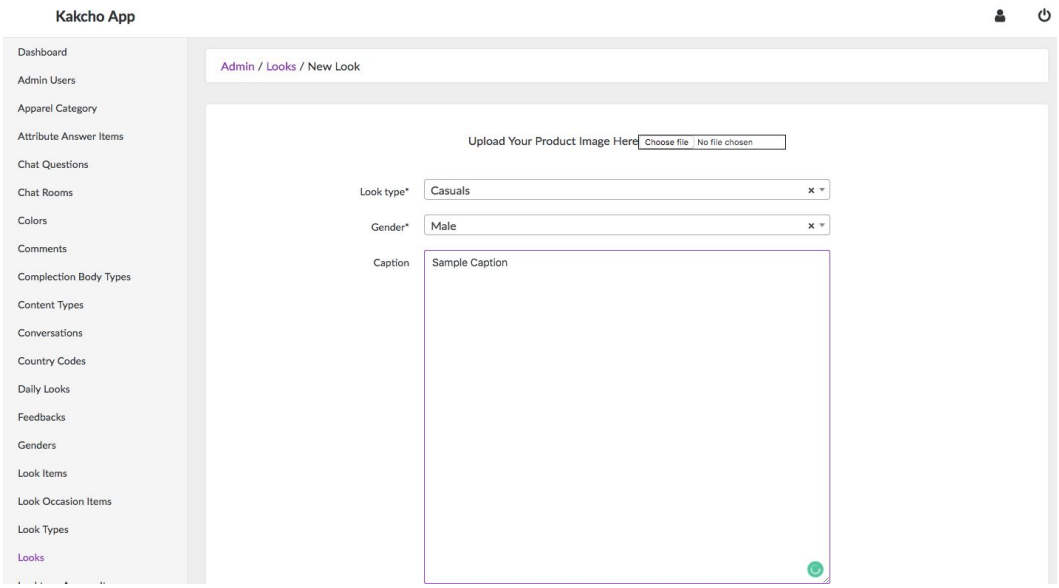


Fig 5.65 Admin Panel New Look Form 1

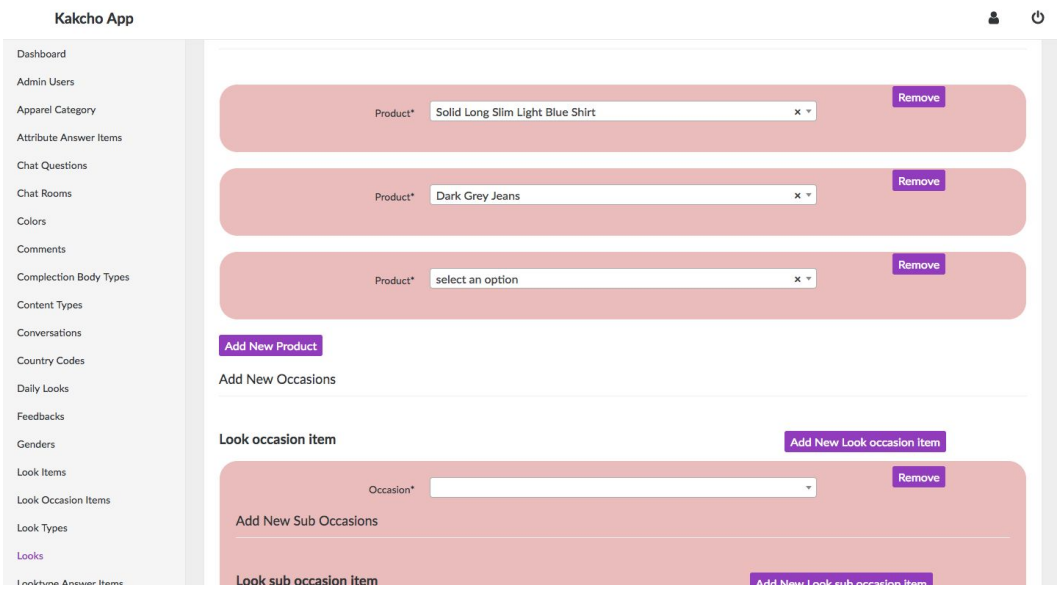


Fig 5.66 Admin Panel New Look Form 2

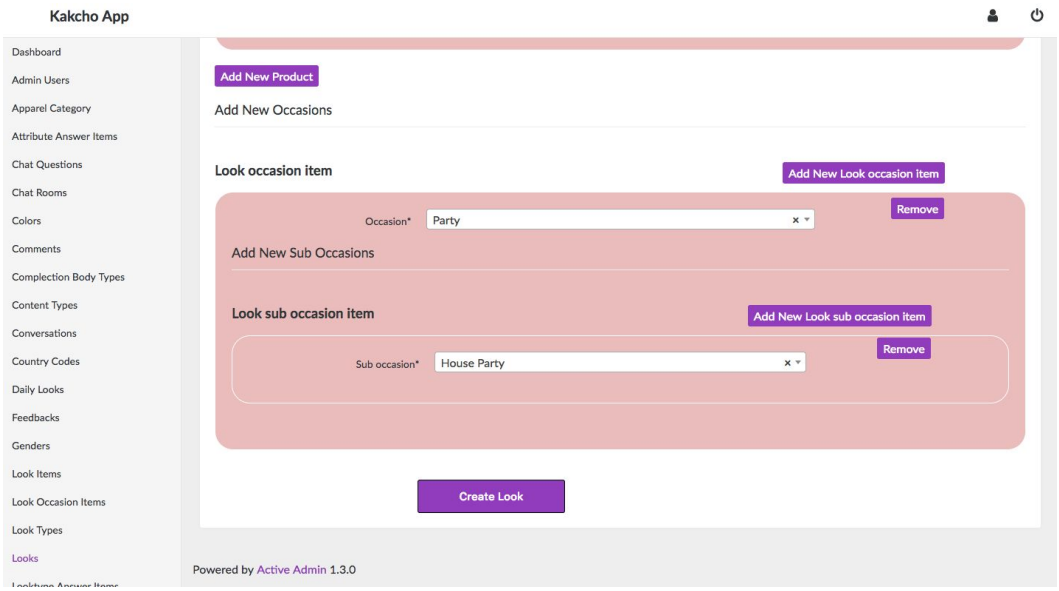


Fig 5.67 Admin Panel New Look Form 3

5.2 SYSTEM FEATURES & SPECIFICATIONS & ESTIMATION

- The system must be capable of handling 10 million users.
- The system just supports 1:1 chatting and no group chats.
- The chat must support pictures as attachments along with text.
- A text message can only be 128 Kb in size otherwise it will be distributed as more than one.
- Notifications should be integrated for messages with custom notification layout.
- Assuming 25 million messages everyday and 160 characters per message on an average, then the system will generate approximately 4 GB messaging data everyday assuming no message metadata.
- If we have to provision the message data size then there will be approx 7300 GB data in 5 yrs.
- Latency will be a very important metric in the system so chat is supposed to be realtime, and hence the end to end time actually matters.
- Consistency is an important factor because if someone sends a sequence of message and the other user don't see some of them. That could lead to huge confusion.
- The most important features for the chat system are:
    - Sending a message to another person.
    - For a user, fetching the most recent conversations.
    - For every conversation, fetching the most recent messages.

5.3 Class UML Diagrams

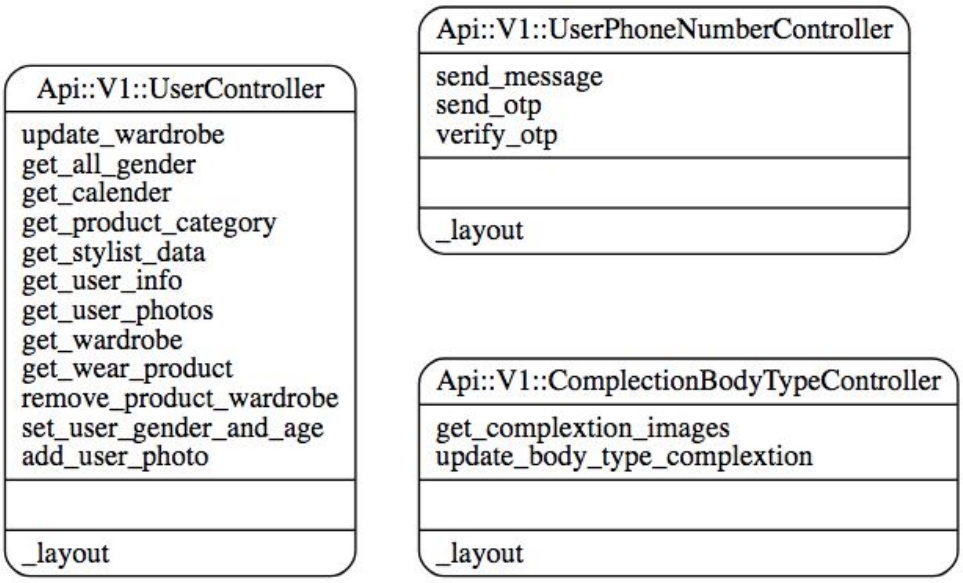


Fig 5.68 Class Diagrams For User, ComplexionBodyType And UserPhoneNumber Controller

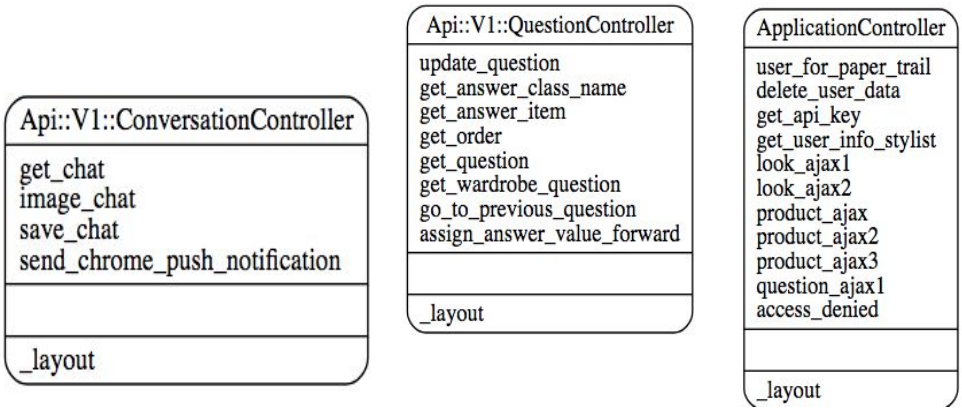


Fig 5.69 Class Diagrams For Conversation, Question And Application Controller

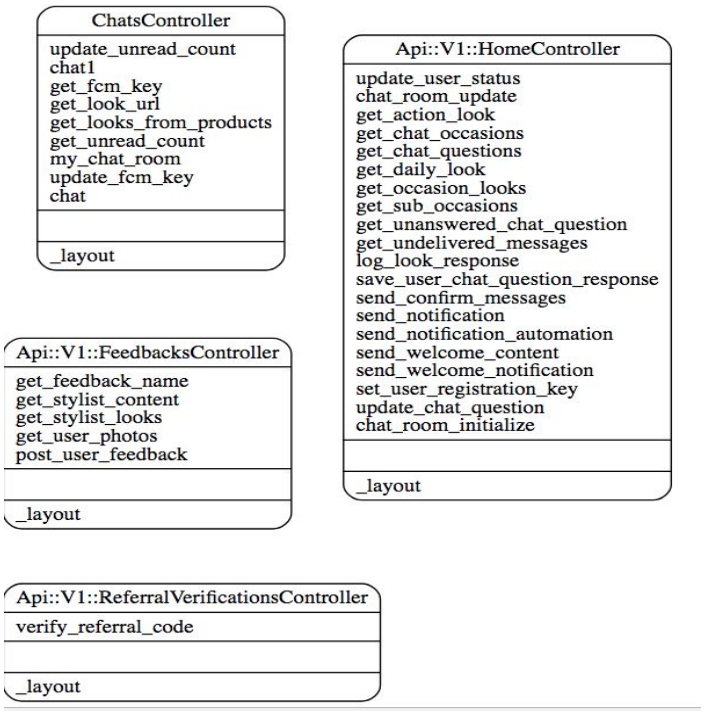


Fig 5.70 Class Diagrams For Chats, Feedbacks, Home And ReferralVerification

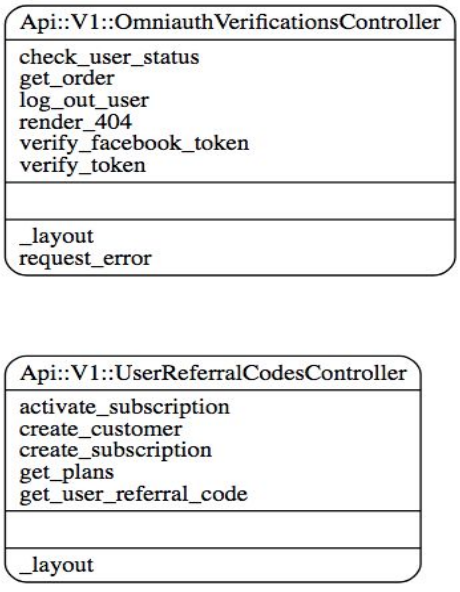


Fig 5.71 Class Diagrams For UserReferralCodes And OmniauthVerification Controller

5.3 DataBase Design

The kakcho Fashion Application comprises of some complex database models associated with each other. The design has been done keeping in mind about the flexibility, durability of the application. The basic design has been explained below:

5.3.1 Ability

5.3.2 AdminUser

- Belongs to UserRole
- Is Recoverable
- Is Trackable
- Is Rememberable
- Is Validatable

5.3.2 AnswerItem -

- Belongs to Question

5.3.3 ApiKey

5.3.4 ApplicationRecord -

5.3.5 AttributeAnswerItem -

- Belongs to Question
- Belongs to UserBodyAttributeValue

5.3.6 BooleanAnswer -

- Belongs to User

5.3.7 ChatAnswerOption -

- Belongs to ChatQuestion
- Has many and Belongs To User

5.3.8 ChatQuestion -

- Has many ChatAnswerOption

- Has many and Belongs To User
- Accepts Nested Attributes For ChatAnswerOption

5.3.9 ChatRoom -

- Belongs To User
- Belongs To AdminUser

5.3.10 Color

5.3.11 ComplexionBodyType

5.3.12 ContentType

5.3.13 Conversation -

- Belongs To ChatRoom

5.3.14 JoinTableUserBodyAttributeUserBodyAttributeValue

5.3.15 JoinTableUserBodyAttributeValueUser

5.3.16 CountryCode -

- Has one UserPhoneNumber

5.3.17 DailyLook -

- Belongs to User
- Belongs To Look

5.3.18 Feedback

5.3.19 Gender

5.3.20 InputAnswer -

- Belongs To User

5.3.21 Look -

- Has PaperTrail
- Has many LookItems which is destroy dependent
- Accepts Nested Attributes For LookItems

- Has many LookOccasionItems which is destroy dependent
- Accepts Nested Attributes For LookOccasionItems
- Has many Occasion Through LookOccasionItems
- Belongs to LookType
- Has many Products Through LookItems
- Belongs to Gender

5.3.22 LookItem -

- Belongs to Look
- Belongs to Product

5.3.23 LookOccasionItem -

- Belongs to Look
- Belongs to Occasion
- Has many LookSubOcassionItems which is destroy dependent
- Has many SubOcassions Through LookSubOccasionItems

5.3.24 LookSubOccasionItem -

- Belongs to SubOccasion
- Belongs to LookOccasionItem

5.3.25 LookType

5.3.26 LookTypeAnswerItem -

- Belongs to LookType
- Belongs to Question

5.3.27 OauthUser

5.3.28 Occasion -

- Has many SubOccasions
- Accepts nested attributes for SubOccasions

5.3.29 Product -

- Has PaperTrail
- Belongs To Gender
- Belongs To WearItem
- Has many and belongs to WearItemStyleNames

5.3.30 ProductAnswerItem -

- Belongs to Product
- Belongs to Question

5.3.31 ProductStyleItem -

- Belongs to Product
- Belongs to WearItemStyleName
- Belongs to WearItemStyleType

5.3.32 ProductWearItemStyleName -

- Belongs to Product
- Belongs to WearItemStyleName

5.3.33 Question -

- Has many and belongs to User
- Has many ProductAnswerItems
- Has many QuestionAnswerItems
- Accepts nested attributes for QuestionAnswerItems
- Accepts nested attributes for AttributeAnswerItems
- Has many AttributeAnswerItems
- Belongs to Gender
- Has many LookTypeAnswerItems
- Accepts nested attributes for ProductAnswerItems

- Accepts nested attributes for LookTypeAnswerItems

5.3.34 QuestionAnswerItem -

- Belongs to Question

5.3.35 RazorpayCustomer -

- Belongs To User

5.3.36 RazorpaySubscription -

- Belongs To User

5.3.37 ReferralVerification -

- Belongs to User
- Belongs to UserReferralCode

5.3.38 ReferredCode -

- Belongs to User
- Belongs to UserReferralCode

5.3.39 SentUserContent -

- Belongs to User
- Belongs to UserContent

5.3.40 SubOccasion -

- Belongs to Ocassion

5.3.41 SuggestedLook -

- Belongs to Look
- Belongs to User

5.3.42 User -

- Has and belongs to many Questions
- Has and belongs to many UserBodyAttributeValue
- Has and belongs to many LookTypeAnswerItem

- Belongs to Gender
- Has and belongs to many ChatQuestions
- Has and belongs to many ChatAnswerOptions
- Has many UserPhotos
- Has one UserPhoneNumber
- Has one RazorpayCustomer
- Has many RazorpaySubcriptions

5.3.43 UserAnswer -

- Belongs to User
- Belongs to QuestionAnswerItem

5.3.44 UserBodyAttribute -

- Has many UserBodyAttributeValues

5.3.45 UserBodyAttributeValue -

- Belongs to UserBodyAttribute
- Has and belongs to many UserContents

5.3.46 UserContent -

- Belongs to ContentType
- Has and belongs to many UserBodyAttrbuteValue

5.3.47 UserFeedback -

- Belongs to User
- Belongs to Feedback

5.3.48 UserPhoneNumber -

- Belongs To User
- Belongs to CountryCode

5.3.49 UserPhoto

- Belongs to User

5.3.50 UserReferralCode -

- Belongs to User

5.3.51 UserRole

5.3.52 Wardrobe -

- Belongs to User
- Has many WardrobeItems

5.3.53 WardrobeItem

- Belongs to User
- Belongs to Product

5.3.54 Wear

- Has many WearItems which is destroy dependent
- Accepts nested attributes for WearItems

5.3.55 WearItem

- Belongs to Wear
- Has many WearItemStyleTypes which is destroy dependent
- Accepts nested attributes for WearItemStyleTypes
- Belongs to Gender

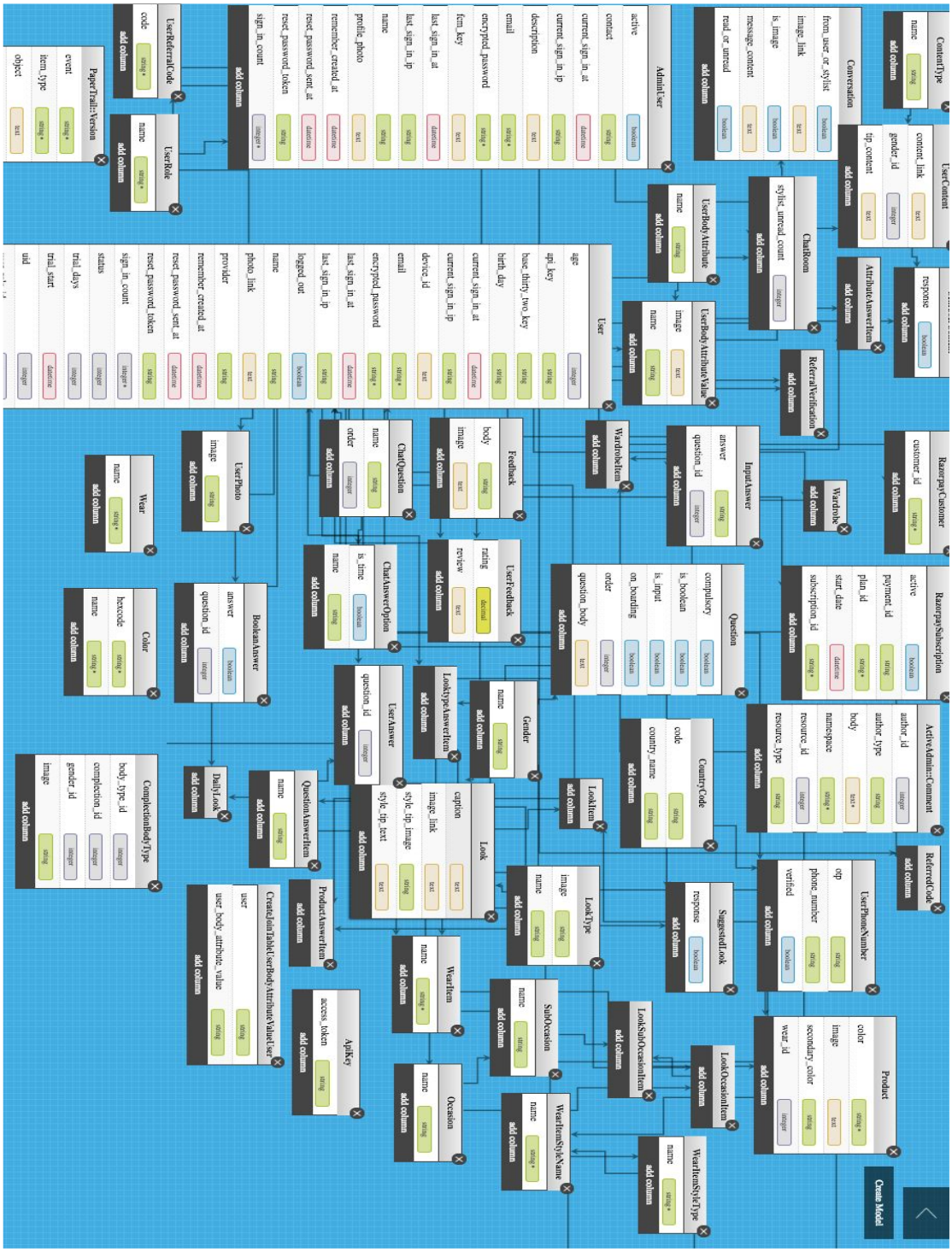
5.3.55 WearItemStyleName -

- Belongs to WearItemStyleType
- Has and belongs to many Products

5.3.56 WearItemStyleType -

- Belongs to WearItem
- Has many WearItemStyleNames which is destroy dependent
- Accepts nested attributes for WearItemStyleNames

5.4 DataBase ER Diagram



CHAPTER - 6
SYSTEM ARCHITECTURE AND MODEL

In this chapter, we will discuss what architectures the application follows for backend services, mobile application and web app frontend. Also this section has a brief explanation of how the application is scaled and how it can handle failure cases on the web server end.

6.1 MODEL VIEW CONTROLLER PATTERN

Model view controller (MVC) is a very useful and popular design pattern. If you're writing software, you should know it. Unfortunately, it's also one of the hardest to truly understand. The application uses MVC design for the backend development part of the application.

6.1.1 How MVC pattern works?

MVC patterns separate input, processing, and output of an application. This model divided into three interconnected parts called the model, the view, and the controller. All of the three above given components are built to handle some specific development aspects of any web or software application. In the MVC development, controller receives all requests for the application and then instruct the model to prepare any information required by the view. The view uses that data prepared by the controller to bring the final output.

6.1.2 What is Model View Controller (MVC)?

In a typical application you will find these three fundamental parts:

- Data (Model)
- An interface to view and modify the data (View)
- Operations that can be performed on the data (Controller)

6.1.3 Three levels of MVC Model:

Model

This level is very important as it represents the data to the user. This level defines where the application's data objects are stored. The model doesn't know anything about views and controllers. So, whenever there are changes done in the model it will automatically notify observers that the changes are made. The model may be a single object or a structure of objects.

Views

A view is a visual representation of the MVC model. This level creates an interface to show the actual output to the user. However, a view will not display anything itself. It is the controller or model that tells view what to display to the user. It also handles requests from the user and informs controller. A view is connected to its model and gets the data necessary for the presentation by asking certain questions.

Controller

Controller is a level which acts like a brain of the entire MVC system. A controller also acts as a link between a user and the system. It provides the user with input by providing appropriate views to present it appropriately on the screen. The controller understands user output, converts it into the appropriate messages and passes the same to views.

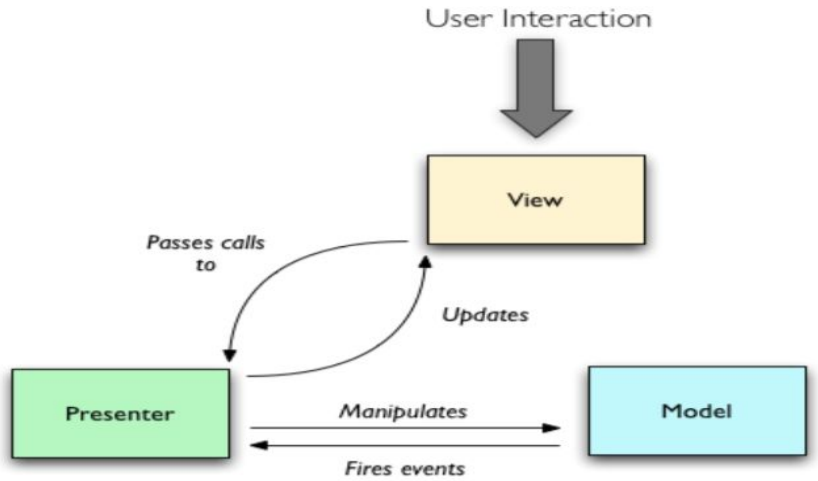


Fig 6.1 MVC Architecture

6.1.4 Important advantages of MVC Model:

Faster development process:

MVC supports rapid and parallel development. If an MVC model is used to develop any particular web application then it is possible that one programmer can work on the view while the other can work on the controller to create business logic of the web application. Hence this way, the application developed using MVC model can be completed three times faster than applications that are developed using other development pattern.

Ability to provide multiple views:

In the MVC Model, you can create multiple views for a model. Today, there is an increasing demand for new ways to access your application and for that MVC development is certainly a great solution. Moreover, in this method, Code duplication is very limited because it separates data and business logic from the display.

Support for asynchronous technique:

The MVC architecture can also integrate with the JavaScript Framework. This means that MVC applications can be made to work even with PDF files, site-specific browsers, and also with desktop widgets. MVC also supports asynchronous technique, which helps developers to develop an application that loads very fast.

Modification does not affect the entire model:

For any web application, user interface tends to change more frequently than even the business rules of the company. It is obvious that you make frequent changes in your web application like changing colors, fonts, screen layouts, and adding new device support for mobile phones or tablets. Moreover, Adding new type of view sare very easy in MVC pattern because the Model part does not depend on the views part. Therefore, any changes in the Model will not affect the entire architecture.

MVC model returns the data without formatting:

MVC pattern returns data without applying any formatting. Hence, the same components can be used and called for use with any interface. For example, any kind of data can be formatted with HTML, but it could also be formatted with Macromedia Flash or Dreamweaver.

6.2 Load Balancer, Cache And Master-Slave Database Replication

It is possible to load balance the caching servers, in addition to the application servers, and use database replication in a single environment. The purpose of combining these techniques is to reap the benefits of each without introducing too many issues of complexity. Here is an example diagram of what a server environment could look like:

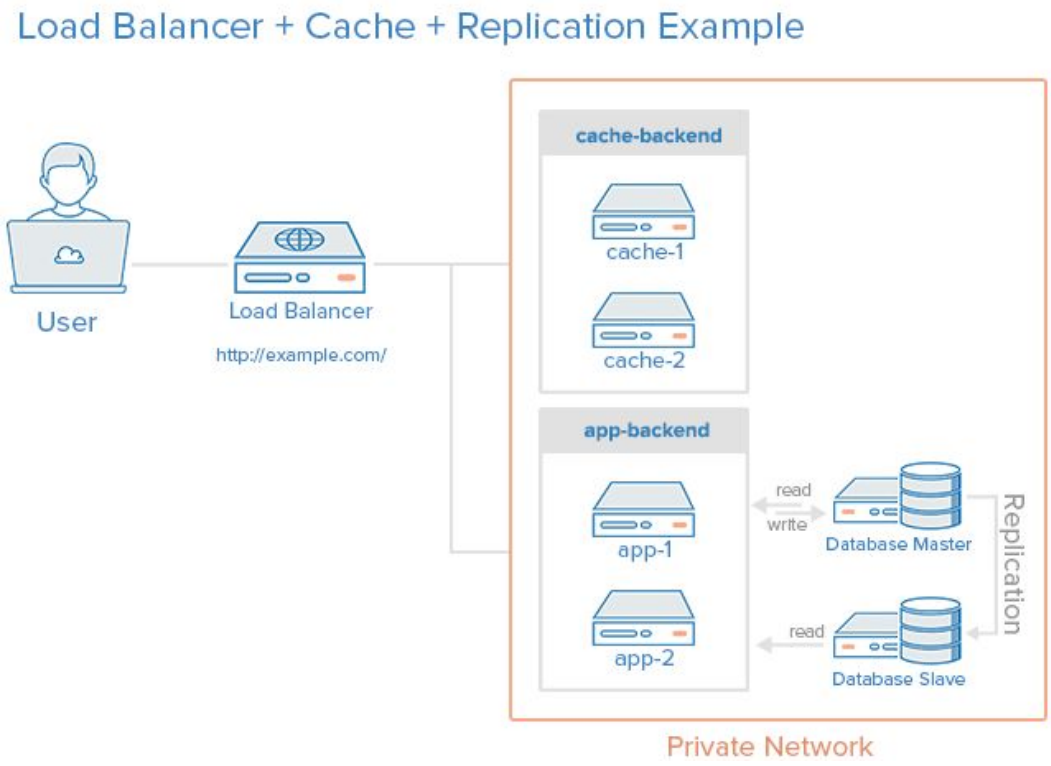


Fig 6.2 Load Balancer, Cache And Replication

Let's assume that the load balancer is configured to recognize static requests (like images, css, javascript, etc.) and send those requests directly to the caching servers, and send other requests to the application servers.

Here is a description of what would happen when a user sends a requests dynamic content:

1. The user requests dynamic content from *http://example.com/* (load balancer)
2. The load balancer sends request to app-backend
3. app-backend reads from the database and returns requested content to load balancer
4. The load balancer returns requested data to the user

If the user requests static content:

1. The load balancer checks cache-backend to see if the requested content is cached (cache-hit) or not (cache-miss)
2. *If cache-hit*: return the requested content to the load balancer and jump to Step 7. *If cache-miss*: the cache server forwards the request to app-backend, through the load balancer
3. The load balancer forwards the request through to app-backend
4. app-backend reads from the database then returns requested content to the load balancer
5. The load balancer forwards the response to cache-backend
6. cache-backend *caches the content* then returns it to the load balancer
7. The load balancer returns requested data to the user

This environment still has two single points of failure (load balancer and master database server), but it provides the all of the other reliability and performance benefits that were described in each section above.

6.3 Two+ web servers, Two+ databases

With this model, you have two options: databases store identical data or have the data evenly distributed among them. In the first case, no more than 2 databases is usually

needed; when one is down, the other can replace it, loss-free. Since data aren't replicated in the second case, some data may become temporarily unavailable if one of the many databases crashes.

Still, this model is considered the most fail-proof: neither web servers, nor databases have single points of failure. If the scale is large, with more than 5 web servers or databases, it's also wise to have load balancers installed. They will analyze all incoming requests and shrewdly allocate them to keep the workload under control.

6.4 Horizontal Scaling

An application's scalability is the potential for an application built to be able to grow and manage more user requests per minute (RPM) in the future. We should acknowledge that, to a certain extent, application architecture does have an impact on scalability.

Let's take a look at the example below. This is what the entire server architecture looks like at the very beginning of a Rails project.

Fig 6.3 Single Server Architecture

What we usually have is a single server, on which we install the following software:

- Nginx server;
- Rack application server – Puma, Passenger, or Unicorn;
- Single instance of your Ruby on Rails application;
- Single instance of your database; usually, a relational database like PostgreSQL.

This server computer will be able to cope with, say, 1,000 or even up to 10,000 requests per hour easily. But let's assume that if the marketing is very successful, and application becomes much more popular, server starts getting 10 or 100 times more requests. When the load increases to a high enough level, a single server architecture cracks under pressure. That is, the application becomes unresponsive to users.

That's why we applied Horizontal Scaling to the application.

Horizontal scaling means converting the single server architecture of the app to a three-tier architecture, where the server and load balancer (Nginx), app instances, and database instances are located on different servers. In such a way, we allocate equal and smaller loads among machines.

Fig 6.4 Horizontal Scaling Architecture

6.4.1 Nginx

Nginx, a commonly used server software for Ruby On Rails applications, is deployed on a single machine to serve as a load balancer and reverse-proxy. We need a medium-powered server for Nginx, because this server requires little computing power to function normally under high loads. Nginx's sole aim is to filter and distribute the load among multiple servers.

We set up this server to receive the initial request and forward it to the first machine. The second request will be sent from Nginx to the second machine, and so on. When you have only three machines with your Rails application instances, then the fourth request from the client (browser) will be sent, naturally, to the first machine again.

6.4.2 App Instances

As mentioned previously, we need additional servers to run app instances separately from the Nginx server. From the user's perspective, the app stays the same, they simply access different app instances thanks to Nginx.

For communication between the application and Nginx, we use a special interface called Unicorn, which is an application server. There are many application servers for Rails-based apps, the best known being Unicorn, Phusion Passenger, and Puma(Used In Development Environment). Application servers are responsible for input-output, letting the application deal with user requests.

Although Rails applications are mostly monolithic, we can extract a functional block and use it as a standalone service. This will also lessen the load on servers.

6.4.3 Database Instances

Scaling a database is another important aspect of scaling an application. We can deploy a database on the same server as the application to economize, but this economization has several drawbacks. When every application instance saves and retrieves data from its

own database instance, then data for the entire application is spread across many machines. This is very inconvenient for website users. If Nginx redirects a user to a different app instance than where their data is stored, they can't even sign in because their data is located on a different machine!

Another reason to separate a database from other servers is to create a fault-tolerant architecture. If our database receives too many requests and can't manage them, it collapses. But other database instances in the data tier can accommodate this load, and the Rails app will continue to work.

That's why, when we are ready to scale your Rails application, our database should be transferred to its own server right away. The database tier can be a single server with a database that's used by all app instances. Or, this tier can consist of several machines each running databases. In order to update data across all databases, we use database replication. Once a database has new data to share, it informs all others about changes (which is standard procedure for multi-master relationships among databases).

In a master-slave replication variant, databases communicate with the master (or main) database, which stores important data about other databases. All other database instances receive updates only when the master database updates data. Multi-master database replication and master-slave replication are the two common approaches to making our data layer consistent across multiple databases.

When the database receives a large quantity of data, it must save it quickly, update its status quickly, and send new data back to the user. An ordinary relational MySQL database is usually replaced by PostgreSQL or MongoDB databases to accommodate large quantities of data when you scale an app.

In order to further reduce the load on the database, several other techniques can be used. Caching is one important solution, although it's indirectly related to database scalability. Caching provides cached data from a storehouse to a client more quickly. Imagine that our Rails app compiles statistics about user activity, such as user sessions. This produces

a high load on the CPU. We can calculate this information once per day, and then show the cached data to the user at any time.

The other two solutions for reducing the load on a database are Redis and Memcached, which are special databases that perform read/write operations extremely fast.

6.4.4 Code Optimisations

Let's assume your service which is demanding on the the CPU. Given that the startup has evolved very quickly, it’s almost a given fact that our algorithm should be optimized in its initial form, and the code must be properly refactored. Code optimization helps to get more out of existing server resources.

6.4.5 Service Oriented Architecture

It's also possible to stick with Ruby and Ruby on Rails in the project, but deploy a service-oriented architecture in order to scale our application. As pur project uses chat functionality, the chat can be loaded with many requests from users, while the other parts of this application are used infrequently. It's unnecessary to break the server structure into three tiers. But it's possible to break the application into two parts.

Fig 6.5 Service Oriented Division

We deployed the application part responsible for chat functionality on a separate server(on Node.js) as a stand-alone application. Chat could then communicate with the

original application and database, but the load on the main server was decreased. Further, it will be easier, faster, and less expensive to scale just this small part of your application – either vertically or horizontally – should the load increase once more. The service-oriented architecture is a great solution for scaling Rails application and is used by many companies, including Github and Shopify.

6.4.6 Caching

To increase the efficiency of the system we used caching mechanism. Most distributed caching system are good with availability and they become eventually consistent. But they are not tightly consistent. For example, let's say I have a distributed cache. If a user's messages or conversations are spread across machines, then it starts causing trouble for us because, The changes are no more atomic. Consider the case when messages for a user are one machine and conversations on another. When a message is added, the update request is sent to the server with messages and server with conversations for this user. There could potentially be a period when one server has processed the update and the other has not. If changes are not atomic, the system is not tightly consistent anymore. One way to resolve this is to make sure that caching for a user completely resides on one server. The same server can also have other users as well, but users are assigned to exactly one server for caching. To further ensure consistency, all the writes for that user should be directed through this server and this server updates its cache when the write is successful on DB before confirming success.

There are some issues with this system :

- A single point of failure for the user : My reads and writes are routed through this caching server. If this caching server suddenly dies, then my reads and writes suddenly start failing.
- To resolve this, we should be able to quickly detect the failed machine, mark it as dead and start from scratch (as cache, reading from DB) on a separate machine. If we have servers for backup, we can just have a heartbeat mechanism to detect the server going down and we can activate the backup server.

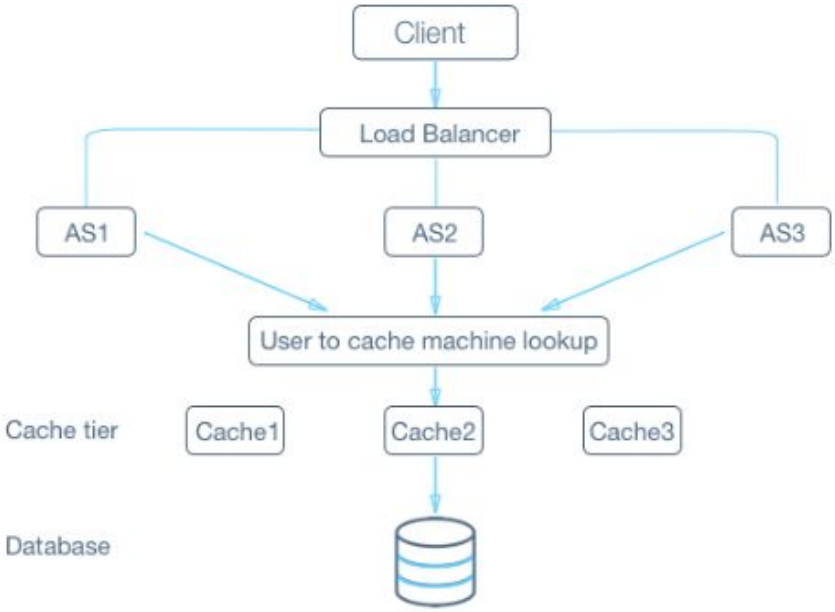


Fig 6.6 Cache Server Architecture

CHAPTER - 7
FINDING AND SUGGESTIONS

To increase the features, feasibility and interactivity in the application the following suggestions are proposed :

1. Integration of deep learning for automatic detection of clothes a user is wearing. This will help in automatic wardrobe updation in the application.
2. Integration of augmented reality and DITTO like algorithms for virtual trial of clothes on user body.
3. Integration of machine learning and deep learning algorithms for better recommendations to the user depending on the current mood and other factors related to the user.
4. End to end encryption in the chat module.
5. Integration with other social apps for better in trend fashion recommendation.
6. Integration of google assistant in the application.

CHAPTER - 8
CONCLUSION

As discussed earlier, getting a personalized feedback on how someone looks in a particular outfit, and what outfit they should prefer on which day is a problem of concern. An outfit defines your confidence for the day and also describes your personality.

The goal was to make a android application with a solid scalable backend that can provide personalized feedback to a person's fashion sense and suggest them an appropriate look based on their own wardrobe. A flexible interface was setup for the user which indeed made the work much easier and convenient.

We met the criteria and implemented all the requirements mentioned in the report. The application is highly scalable and has a flexible architecture that can easily update future aspects of the idea.

To access the mobile application, the user must have an android application with internet privileges. The application is available at https://goo.gl/UaOAdw .

Now that all the requirements have been fulfilled, we claim that we solved the problem and reached our goal.

CHAPTER - 9
BIBLIOGRAPHY

1. Ruby On Rails Documentation :- https://rubyonrails.org/
2. Socket.IO Documentation :- https://socket.io/
3. Github :- https://github.com/
4. Node.JS Documentation :- https://nodejs.org/en/docs/
5. Ruby Gems :- https://rubygems.org/
6. Active Admin Documentation :- https://activeadmin.info/
7. Medium :- https://medium.com/
8. Code Pen :- https://codepen.io/
9. Mozilla Developer Network :- https://developer.mozilla.org/en-US/
10. Stack Over Flow :- https://stackoverflow.com/
11. FCM Docs :- https://firebase.google.com/docs/cloud-messaging/
12. Android Documentation :- https://developer.android.com/docs/
13. Rails Cast :- http://railscasts.com/
14. You Tube :- https://www.youtube.com/
15. Semantic Ui Docs :- https://semantic-ui.com/