BABY SITTING APP

A report submitted in partial fulfilment of the requirement for the award of degree of

BACHELORS OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted By:

Jatin Garg (190280421)

Rimzim (190280450**)**

Pardeep Singh (190280442)

Under the supervision of

Er. Manpreet Kaur

Assistant Professor, Computer Science & Engineering

Department of Computer Science & Engineering

Baba Farid College of Engineering and Technology

Muktsar Road, Bathinda, 151001

May, 2023

CONTENTS

ACKNOWLEDGEMENT

Word often fails to express one’s feelings of indebtedness to one's benefactors. In fact, words are not the proper media for this purpose. It is my proud privilege to express my profound debt of gratitude and sense of deep indebtedness from the core of my heart to **Er. Manpreet Kaur** under whose guidance I did my Major Project and other staff members who has been kind enough to allow me the benefit of their inspiring advice, rich experience.

The help rendered by **Er. Ashish Kumar**, Mentor of Our Batch, Computer Science Engineering Department, **Er. Sunil Nagpal, Head of Department (HoD)**, Computer Science Engineering, Baba Farid College of Engineering and Technology is greatly acknowledged. I would like to express gratitude to other faculty members of Computer Science Engineering Department, **Baba Farid College of Engineering & Technology**, and Bathinda for their intellectual support throughout the course of this work.

JATIN GARG (190280421)

RIMZIM (190280450)

PARDEEP SINGH (190280442)

ABSTRACT

The increasing demands of modern life have led to a growing need for reliable and efficient babysitting services. To address this need, we have developed Baby Sitting App, a cutting-edge mobile application designed to connect parents with qualified and trustworthy babysitters. This project report provides a comprehensive overview of the development process and features of Baby Sitting App.

The Baby Sitting App app leverages the power of technology to streamline the babysitting process, ensuring a secure and convenient experience for both parents and babysitters. The app offers a user-friendly interface with intuitive navigation, making it accessible to parents and babysitters of all backgrounds. Through Baby Sitting App, parents can effortlessly search for available babysitters based on their preferences, such as location, availability, and experience.

Key features of Baby Sitting App include an advanced search algorithm, real-time availability updates, secure messaging, and a comprehensive review and rating system. These features facilitate seamless communication and trust-building between parents and babysitters, enabling parents to make informed decisions about their childcare choices.

Furthermore, Baby Sitting App incorporates robust safety measures to ensure the well-being of children. Babysitters undergo a rigorous verification process, including background checks, reference checks, and identity verification, prior to being listed on the platform. Additionally, parents have the option to view detailed profiles, including qualifications, certifications, and previous experience, to make informed decisions about the babysitter that best suits their needs.

The development of Baby Sitting App involved the utilization of modern technologies such as mobile application development frameworks, cloud computing, and secure authentication protocols. The app is compatible with iOS platforms, ensuring widespread accessibility.

In conclusion, Baby Sitting App revolutionizes the traditional babysitting model by providing a reliable and efficient platform for parents to find trusted babysitters. The app offers a seamless user experience, robust safety measures, and advanced features that simplify the process of hiring a babysitter. Baby Sitting App aims to alleviate the stress of finding reliable childcare services, providing peace of mind for parents while ensuring the well-being of their children.

List of Figures

List of Tables

Chapter - 1

Introduction

1.1 Introduction

In today's fast-paced society, the trend of nuclear families residing in metropolitan areas with working mothers have increased. These families require dependable and convenient babysitting services. With their busy schedules and demanding lifestyles, finding a suitable babysitter at short notice can be difficult for parents. Our project, the BabySitting App, aims to simplify and alleviate this process, providing a stress-free experience for parents.

Our baby-sitting app project aims to provide a convenient and reliable platform for parents to find and book trusted babysitters for their children. The app allows parents to search for babysitters based on their location, availability, and credentials. They can view the profiles of potential babysitters, including their experience, hourly rates, and reviews from other parents. Parents can also message or call the babysitter directly to discuss details and arrange bookings.

Babysitters can create a profile on the app and showcase their experience, qualifications, and availability. They can manage their schedules and receive notifications when a parent requests their services. Babysitters can also accept or decline booking requests, communicate with parents, and receive payments through the app.

The app includes a rating and review system to ensure transparency and accountability. Parents can leave feedback on their experience with a babysitter, and babysitters can improve their profile and reputation based on their ratings.

Our goal is to provide a safe, easy-to-use, and efficient platform that connects parents and babysitters and makes the process of finding a trusted babysitter stress-free. We believe that our app will help parents and babysitters build lasting relationships and provide peace of mind for parents knowing their children are in good hands.

1.2 Problem Statement

The challenge and stress parents experience when trying to find a dependable and trustworthy babysitter for their children is the issue we are attempting to solve with our baby-sitting software project. Finding a reliable babysitter on short notice can be difficult and time-consuming for parents who have hectic schedules and need someone to watch their kids.

Asking friends or relatives for referrals is one of the more conventional ways to find a babysitter, but it can be constrained and untrustworthy. To protect children's safety and wellness, online job boards and classifieds can be daunting and frequently need to undergo thorough vetting. Parents may not know who they can trust with their kids because many babysitters do not have the required training or expertise.

By giving parents a platform to interact with pre-screened, certified babysitters who satisfy certain requirements and have a history of providing high-quality care, our app strives to alleviate these difficulties.

By providing a reliable and easy-to-use platform, we hope to eliminate the stress and uncertainty that parents face when searching for a babysitter and ensure that their children are in safe and capable hands.

Furthermore, our app will provide a seamless experience for both parents and babysitters, with features such as in-app messaging, scheduling, and payment processing. Parents can easily find and book a babysitter, and communicate directly with them through the app. They can also view the babysitter's profile, which includes their qualifications, experience, and ratings from previous clients. The app also provides a secure payment system, eliminating the need for cash transactions and ensuring a transparent and hassle-free process for both parties.

Methods of finding a babysitter, such as asking friends or family members for recommendations, can be limiting and unreliable. Online classifieds or job boards can be overwhelming and often require extensive vetting to ensure the safety and wellbeing of children. Additionally, many babysitters may not have the necessary qualifications or experience, leaving parents unsure about who they can trust with their children.

Our app aims to address these challenges by providing a platform that connects parents with pre-screened and certified babysitters who meet specific qualifications and have a proven track record of providing quality care. By providing a reliable and easy-to-use platform, we hope to eliminate the stress and uncertainty that parents face when searching for a babysitter and ensure that their children are in safe and capable hands.

1.3 Objectives

Our application's goal is to offer a user-friendly interface. The programme should be easy to use, with straightforward directions and an intuitive user interface that guides users through the full delivery process.

The objective of this application are as follows:

- Connect parents with reliable and trustworthy babysitters who meet specific qualifications and have been pre-screened for safety and background checks.
- Provide a platform that makes it easy for parents to search for and book babysitters on short notice.
- Enable babysitters to create profiles with their qualifications, experience, and availability, and give them tools to manage their schedules and bookings.
- Ensure that parents can communicate with babysitters before and during the booking to establish trust and ensure that their children's needs are met.
- Provide a rating and review system for parents to give feedback on their experience with babysitters, which can help improve the quality of care over time.
- Allow parents to easily pay for babysitting services through the app, and provide a secure and convenient payment system for babysitters to receive payment.
- Ensure that the app complies with all relevant safety regulations and legal requirements for babysitting services.

1.4 Methodology

In this section, we will discuss the methodology used in the project step by step. For this application, we have utilised Xcode and Swift programming languages for developing the front-end mobile application.

The development and deployment of iOS applications can be done in various ways. However, for this project, we have chosen to use the native iOS development approach. This allows us to have greater control over the development process and take advantage of the features and capabilities provided by Apple's iOS platform.

Our project's unique objectives and requirements can be met by a personalised, scalable mobile application created with iOS development. Additionally, we can make use of the extensive libraries and frameworks offered by the iOS ecosystem, which enables us to quickly and effectively create a high-quality mobile application.

We want to create a reliable and intuitive babysitting application that offers a seamless and pleasurable experience for both parents and babysitters by employing the native iOS development technique.

After selecting the platform and framework, the first step in our project is to gather all necessary information and requirements. This phase involves identifying user demands, determining necessary features, and establishing the project's scope. Then, we move onto the design and wireframing stage where we focus on creating UI/UX designs for the application. This involves conceptualising and prototyping the application, determining user flows, developing interface functionality and user experience, and creating an aesthetic that aligns with the application and intended audience's needs. To achieve this, we follow certain rules, such as identifying target users by performing user studies and collecting feedback. Additionally, we keep the layout simple, clear, and straightforward, avoiding complicated navigation and cluttered interfaces. Consistency is also crucial, ensuring the use of the same layout, fonts, colours, and icons throughout the entire application for a uniform and consistent user experience. Finally, we choose appropriate colours that are suitable for the brand and intended use of the product and refrain from using excessive colours while ensuring adequate contrast for easy comprehension and navigation.

In order to achieve this we have to follow certain rules:

1. **Identify target consumers:** Conducting user research and gathering feedback can help you gain insight into your users' needs, goals, and pain points. This information can be used to develop an application that meets the needs and expectations of your target users.

2. **Simple Design:** it is crucial to ensure that the layout is simple, intuitive and easy to use. A complicated and cluttered interface can lead to confusion and frustration for users, making it less likely that they will continue to use the app.
3. **Consistency in design :** Consistency is a vital aspect of designing a successful application. To ensure a seamless and consistent user experience, it is essential to use the same layout, fonts, colours, and icons throughout the entire application. This consistency will help users easily identify and navigate through the application, creating a sense of familiarity and comfort with the interface.
4. **Using the right colours:** Choosing the right colours is an important aspect of creating an effective user interface. The colours and palettes we select should not only be consistent with the brand image, but also be appropriate for the intended use of our product. It is important to strike a balance between using enough colours to make your design interesting, but not so many that it becomes overwhelming or confusing to users.

Swift, Xcode	Front-end framework
Node.js	Back-end
Express.js	API
Stripe	Payment gateway
Firebase	Back-end authentication

Table 1.1 Technologies used for application

Once the UI of the application is created and the user needs are understood, the next step is to determine the technology requirements for the project. This includes selecting the technological stack. Swift is the main technology used for developing the mobile application for iOS users. Additionally, Node.js is used for API connectivity and Firebase is used for backend services.

Google's Firebase platform provides a wide range of backend capabilities that enable developers to create apps with greater efficiency and speed. Programmers can leverage real-time cloud communication to distribute warnings to consumers across numerous platforms, including iOS and Android, the web, and Firebase features like authentication and cloud messaging. The most important phase of writing the code begins after the technologies are chosen. This entails developing the front-end and back-end of the programme, integrating APIs, and testing it for bugs and other difficulties. To make sure the application meets its functionality and operational requirements, testing is done throughout the testing and quality management phase. Documenting all work done throughout this phase and making sure the source code is flexible and managed to handle upcoming updates and improvements are essential.

Chapter-2

Literature Survey

A literature review on babysitting apps for iOS users reveals that there is a growing demand for such apps, as parents look for ways to manage their busy schedules while ensuring their children's safety. A study by InMobi found that over 80% of parents with children aged 5 and under used mobile devices to assist with parenting tasks, including finding babysitters. Another survey by UrbanSitter showed that parents increasingly prefer to find and book babysitters through mobile apps.

The literature has explored the impact of technology on parenting, with a particular focus on the benefits and drawbacks of using babysitting apps. For instance, one study found that parents who used babysitting apps reported greater flexibility, convenience, and ease of use, but also expressed concerns around trust and security [1].

Another study examined the features and functionalities that are most valued by users, finding that parents prioritise safety, reliability, and ease of communication. The survey also suggests that trust and reputation are critical factors in the babysitting app ecosystem, with many users relying on user ratings and reviews to evaluate potential sitters [2].

Privacy and security are also important considerations, with several studies highlighting the need for robust safeguards to protect children and their families. For instance, researchers have called for enhanced identity verification measures, background checks, and secure messaging systems to ensure safe and secure communication between parents and sitters.

Overall, the literature survey suggests that babysitting apps have significant potential to improve the childcare experience by providing greater flexibility and convenience for parents while also offering job opportunities and flexibility for sitters. However, it is essential to carefully consider safety, privacy, and trustworthiness to ensure that these apps are a safe and reliable option for families.

Chapter - 3

System Development

3.1 Requirement Analysis

Prior to developing a babysitting app, a technical analysis should be conducted to ensure the feasibility of the project. This analysis should include evaluating the app's requirements, such as the features and functionalities that the app will provide. The analysis should also consider the technical constraints, such as the mobile platforms on which the app will run, and the compatibility of the app with various devices. Other factors to consider in the technical analysis include the cost of development, including the resources and time required to build the app, as well as any potential security or data privacy concerns that need to be addressed. By conducting a thorough technical analysis, developers can identify potential issues or challenges that may arise during development and take steps to address them before beginning the project. This can help ensure that the app is developed efficiently and effectively, and meets the needs of its users.

In the context of a babysitting app, Firebase could be a great option for handling the backend. The real-time database feature could be particularly useful for tracking babysitting sessions, allowing parents and sitters to see updates in real-time. The authentication feature could also help ensure that only authorised users have access to the app, providing an additional layer of security.

Moreover, Firebase's hosting and cloud functions can help improve the app's performance and scalability, allowing it to handle a large volume of requests and users. This is especially important for a babysitting app that needs to be available and responsive at all times.

On the other hand, Node.js is a server-side runtime environment that allows developers to build fast and scalable web applications using JavaScript. It offers a vast library of modules and packages, making it easy to develop and customise a backend system that fits the specific needs of a babysitting app.

3.2 Design

This step involves gathering, analysing, identifying, compiling, and evaluating the requirements and functionality for the application. It is necessary to locate the application's essential elements, such as new user registration, logging in as a registered user, identifying the pick-up and delivery points for current users, figuring out the delivery cost, processing payments, and delivery tracking.

Based on these requirements we made a system design that covers the structure, interface for users, and database structure based on the specifications. Make a design paper that specifies the system's technical specifications. We need to create the system with the aid of essential platforms and technologies, including Xcode and Firebase (for backend services). Design the user interfaces, combine the back-end services, and put the system logic into practice. After implementation we have to run tests on our application. The platform's construction utilising Swift, Firebase, and other pertinent resources and structures is the main emphasis of the implementation phase.Firebase is used to create the backend services and offers features like cloud computing, database storage, and authentication.

Firebase is a popular backend-as-a-service (BaaS) platform Firebase offers numerous services for mobile and web applications, such as cloud storage, real-time databases, verification, and more. There are a number of benefits of utilising Firebase in Ios projects:

Numerous advantages of using Firebase in OS projects include simple integration, scalability, real-time updates, strong authentication

This phase involves designing a technological strategy for the item's delivery application as part of the design process. This comprises the database schema, user interface, and architecture. The system's backend services, APIs (Application Programming Interface), and user interface components are all described in the structure of the system plan along with how they interact. To assure accessibility and ease of use, the user interface layout should adhere to best practices and guidelines for design. The data model, relationships between entities, and data storage techniques should all be specified in the database architecture.

The flow and architecture of the entire project included dividing the model into three sections. The parents/ Customer panel, The NannY panel and the Admin Panel.

It was important to analyse the needs and requirements of each section and create front end along with functionalities of each module accordingly.

Parents Panel

The customer panel is designed for parents who are looking for a babysitter. It allows them to search and filter through available babysitters, view their profiles, and book a sitter for a specific date and time. The customer panel also enables parents to view their past bookings, rate and review babysitters, and make payments for their bookings.

There are several modules and pages in this project starting from the Launch Screen. We have many, numerous screens or "views" that collectively make up the application's user interface. Each screen serves as an individual component of the mobile application and have various features and functionalities.The parents panel included the following screens.

S. No.	Screen Title	Functionality
1.	launch Screen	This is the launch screen of the app i.e. the initial screen that is displayed when the app is opened by the user.
2.	Welcome Screen	This screen welcomes the user to the app and gives a brief introduction to the product.
2.	Login Screen	In this screen the user needs to enter the registered mobile number and press submit and he will move to the next screen.
3.	Register Screen	If the user is not registered (new user), he has to enter a mobile number to get himself registered.

4.	Verification Screen	In this screen there is a user input of 6 digits in which the user needs to enter the OTP which he got via SMS on the entered mobile number.
5.	Home Screen	This screen lets the user select whether he/she wants to look for a nanny, or babysit. This screen navigates which panel the user will be navigated to.
6.	Location Screen	In this screen the user needs to enter the location where they need babysitting services at.
7.	User Profile Screen	This screen gives a personalised information of the account holder and helps user find their account information, notifications, reviews etc
8.	Job Card Screen	This screen allows the user to select their requirements and specifications like timings, price range etc
9.	Select Nanny Screen	This displays all the nannies near the user that have been filtered out by the previous requirements filled by the customer.
10.	Hiring Nanny Screen	Gives insight of the nanny selected, and lets the user hire or invite accordingly.
11.	Payment Screen	After the nanny is hired, the user is redirected to this screen to complete payment.
12.	Ratings and Reviews Screen	This is a feature screen of this application where the user can rate and review the nanny for his/her service.

13.	Notification Screen	This screen shows the notifications for the user.
14.	Edit Profile Screen	The user can edit his/ her personal profile and update his/her details using this screen.

Table 3.1 List of screens in Parent panel

Nanny Panel

The nanny panel is designed for babysitters who are looking for work. It allows them to create a profile, list their availability, and accept or reject bookings. The nanny panel also enables babysitters to view their past bookings, receive payments for their services, and communicate with parents. After getting redirected to the nanny panel from the home screen, The nanny panel follows the following flow of screens.

S. No.	Screen Title	Functionality
1.	Location Screen	In this screen the user needs to enter the location where they need babysitting services at.
2.	User Profile Screen	This screen gives a personalised information of the account holder and helps user find their account information, notifications, reviews etc
2.	Job Card Screen	This screen allows the nanny to fill in their details like hourly rate, availability etc.
3.	Search Family	Allows the nanny to search families which have

	Screen	registered accounts to provide long term services.
4.	Application Screen	Allows the nanny to apply for a family
5.	Requests Home Screen	Allows the nanny to get a list of the customers along with their details that have invited him/her.
6.	Notification Screen	In this screen the nanny can see all the notifications/ communications
7.	Ratings and Reviews Screen	This screen is for keeping track of the ratings and reviews the nanny has been awarded by previous clients.
8.	User Profile Screen	This screen is the personalised account holder screen
9.	Add account Screen	Lets the nanny create an account and add her bank details to receive payments from clients.
10.	Edit Profile Screen	This screen enables the nanny to edit her public profile.
11.	Set Price Screen	Allows the nanny to fix a price which will be publicly displayed for his/her services.

Table 3.2 List of screens in Nany panel

Admin Panel

The admin panel is designed for the app's administrators who manage the app's operations. It allows them to view and manage all the bookings made through the app, track payments, manage the database of babysitters and customers, and resolve any disputes or issues that arise. The admin panel also enables administrators to view analytics and generate reports on app usage, user behaviour, and revenue.

The Admin Panel included the following screens:

S. No.	Screen Title	Functionality
1.	Manage Users	Lets the admin manage both customers and nanny profiles.
2.	Manage Complaints	Keeps a track of the complaints issued by the customers.
2.	Payments	Keeps a record of all the payments made by the customers.
3.	Customer Support	Keeps track of customer support.
4.	Invoices	Keep a track of invoices.

Table 3.3 List of screens in Admin panel

Source Code of Admin Panel

```
const jwt = require("jsonwebtoken");
const models = require("../models");
const User = models.User;
const Age = models.Age;
const Booking = models.Booking;
const Parent = models.Parent;
const Nany = models.Nany;

exports.register = async (req, res) => {
 try {
  const { phone_number, account_type } = req.body;

  const checkUser = await User.findOne({
   where: { phone_number: phone_number },
  });

  if (checkUser) {
   res.status(500).json({ message: "User already exist" });
   return;
  }

  const user = await User.create({
   phone_number: phone_number,
```

```
      account_type: account_type,
    });

    const payload = {
      id: user.id,
      account_type: user.account_type,
    };

    const secret = process.env.SECRET_KEY;
    const token = jwt.sign(payload, secret, { expiresIn: "120m" });
    res.status(200).json({
      success: "ok",
      msg: "Register Successful",
      data: user,
      token: token,
    });
  } catch (error) {
    console.log(error);
    res.status(500).send(error);
  }
};

exports.login = async (req, res) => {
  try {
```

```
    const { phone_number } = req.body;

    const user = await User.findOne({ where: { phone_number: phone_number } });
    if (!user) {
      res.status(500).json({ message: "Please Sign up" });
      return;
    }

    const payload = {
      id: user.id,
      account_type: user.account_type,
    };

    const secret = process.env.SECRET_KEY;
    const token = jwt.sign(payload, secret, { expiresIn: "120m" });
    res.status(200).json({
      success: "ok",
      msg: "Register Successful",
      data: user,
      token: token,
    });
  } catch (error) {
    console.log(error);
    res.status(500).send(error);
```

```
  }
};

exports.updateProfile = async (req, res) => {
  try {
    const userid = req.userid;
    const user = await User.findOne({ where: { id: userid } });

    const {
      name,
      email,
      address,
      city,
      postal_code,
      latitude,
      longitude,
      child_category,
      gender,
      price,
    } = req.body;

    const updatedata = await user.update(
      {
        name: name,
```

```
      email: email,
      address: address,
      city: city,
      postal_code: postal_code,
      latitude: latitude,
      longitude: longitude,
      gender: gender,
      avatar: req.file.filename,
    },
    {
      where: { id: userid },
    }
  );

  if (user.account_type === 1) {
    const parent = await Parent.findOne({ where: { user_id: user.id } });
    if (parent) {
      const updateparent = await Parent.update(
        {
          child_category: child_category,
        },
        {
          where: {
            user_id: user.id,
```

```
        },
      }
    );
  } else {
    const parentdata = await Parent.create({
      user_id: user.id,
      child_category: child_category,
    });
  }
 }

 if (user.account_type === 2) {
  const nany = await Nany.findOne({ where: { user_id: user.id } });

  if (nany) {
    const updatennay = await Nany.update(
      {
        price: price,
      },
      {
        where: {
          user_id: user.id,
        },
      }
```

```
      );
    } else {
      const nanydata = await Nany.create({
        user_id: user.id,
        price: price,
      });
    }
  }

  res.status(200).json({
    success: true,
    msg: "Profileupdated sucessfully",
    data: updatedata,
  });
 } catch (e) {
  console.log(e);
  res.status(500).send(e);
 }
};

exports.Age = async (req, res) => {
 try {
  const age = await Age.bulkCreate(req.body);
```

```
    res.status(200).json({
      success: true,
      msg: "child category created sucessfully",
      data: age,
    });
  } catch (error) {
    console.log(error);
    res.status(500).send(error);
  }
};

exports.getAge = async (req, res) => {
  try {
    const age = await Age.findAll();

    res.status(200).json({
      success: true,
      msg: "child category fetch successfully",
      data: age,
    });
  } catch (error) {
    console.log(error);
    res.status(500).send(error);
  }
```

```
};


exports.getNany = async (req, res) => {
  try {
    const id = req.query.id;
    const data = await User.findOne({
      where: { id: id },
      include: [
        {
          model: Nany,
        },
      ],
    });
    res.json({ success: true, msg: "nany data", data: data });
  } catch (e) {
    res.status(500).send(e);
  }
};
```

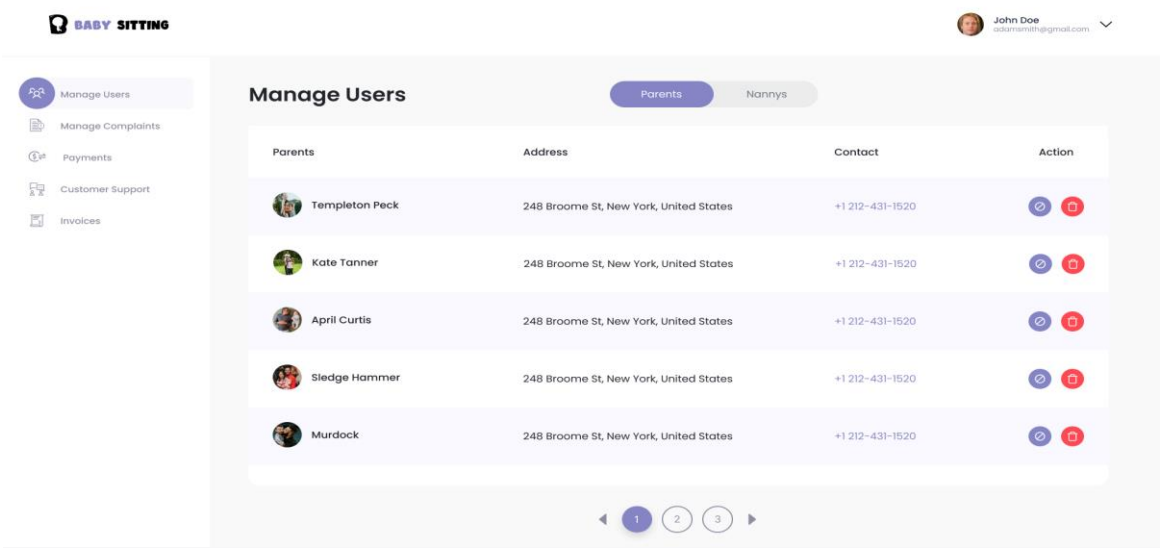

Fig. 3.1 User manage in Admin Panel

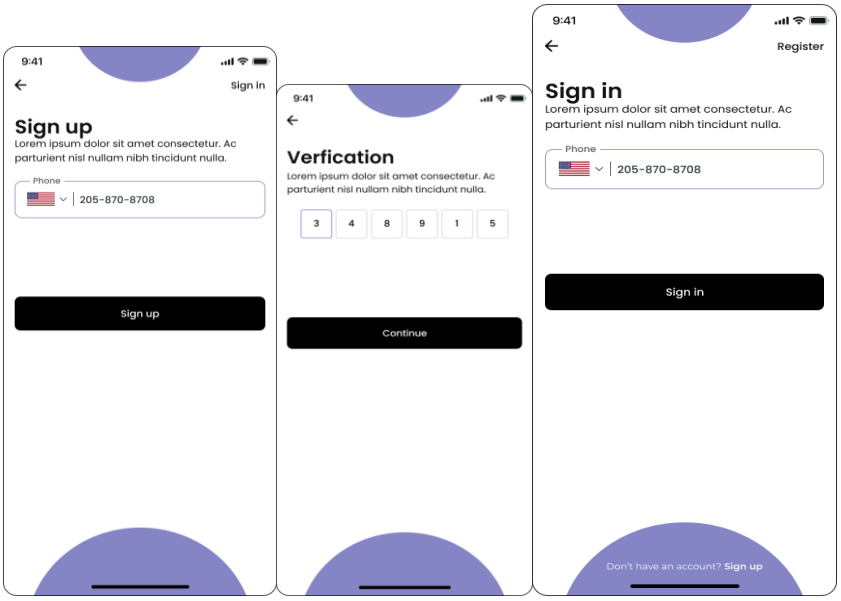

Fig. 3.2 Sign Up and Sign In Screens

User authentication : It is a common functionality in all three panels. If the user is new to this application he needs to sign up and if an individual is already registered then he has to simply sign in. In both the cases the user can only validate himself with the mobile number. The user gets an OTP on their registered mobile number.

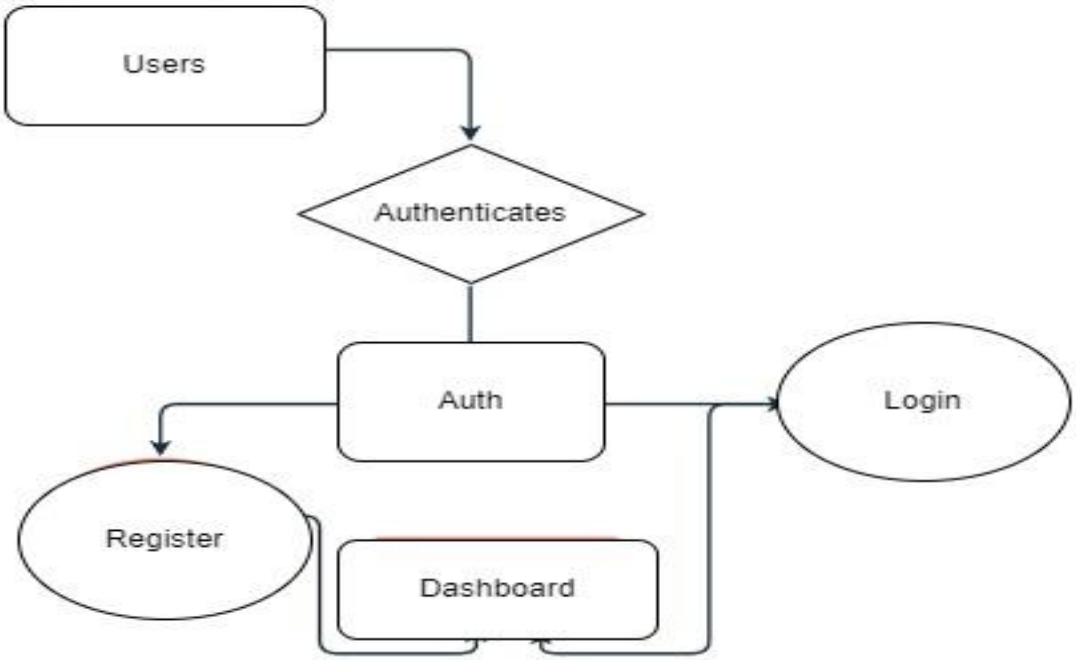

Fig. 3.3 ER diagram of user authentication

Location, Pricing Selection

Moving further in this application after signing-in the user will get an option of In this module of our application the user have to fill following details :

- Address
- City
- Timings
- Price Range

Source Code of Authentication

```
const { validationResult } = require("express-validator");
const db = require("../models");
const sequelize = require("sequelize");
const Feedback = db.Feedback;
const Booking = db.Booking;
const User = db.User;
const Nany = db.Nany;

exports.getNany = async (req, res, next) => {
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  const s = `(
    6371 * acos(
          cos(radians(${req.body.latitude}))
          * cos(radians(latitude))
          * cos(radians(longitude) - radians(${req.body.longitude}))
          + sin(radians(${req.body.latitude})) * sin(radians(latitude))
      )
   )`;
```

```
  const nany = await User.findAll({
    where: {
      account_type: 2,
    },
    include: [
      {
        model: Nany,
        where: {
          price: {
            [sequelize.Op.lte]: req.body.price,
          },
        },
      },
    ],
    attributes: ['id','name','email','address',[sequelize.literal(s), "distance"]],
    order: sequelize.col("distance"),
    having: sequelize.literal(`distance <= 1000`),
  });

  return res.status(200).json({
    data: nany
  });
};
```

```
exports.bookNany = async (req, res) => {
  try {
    const userid = req.userid;
    const { nany_id, start_date, end_date } = req.body;

    const booking = await Booking.create({
      parent_id: userid,
      nany_id: nany_id,
      start_date: start_date,
      end_date: end_date,
    });

    res
      .status(201)
      .json({
        success: true,
        msg: "Nany hired successfully",
        booking: booking,
      });
  } catch (error) {
    res.status(500).send(error);
  }
};
```

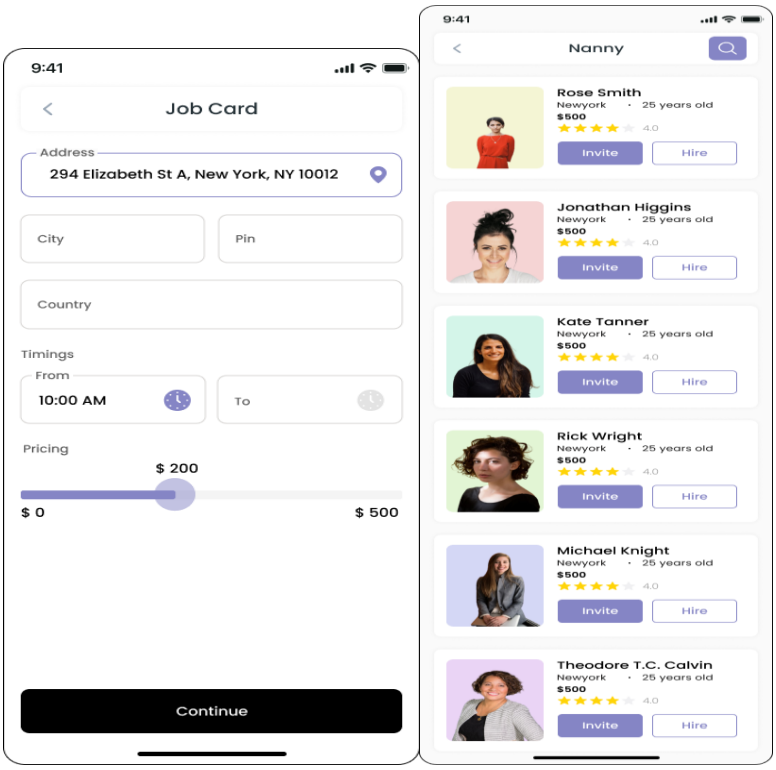

Fig 3.4 Shows the flow of this process, first the user will set the details, and then will be directed to a screen that contains a list of nannies that match their requirements.

After selecting the desired nanny, the user is directed to the make payment page. As the user confirms the source and destination points he will be getting a detailed bill for the delivery and he will get a make payment page.The user has the functionality to add a new card to make payment.

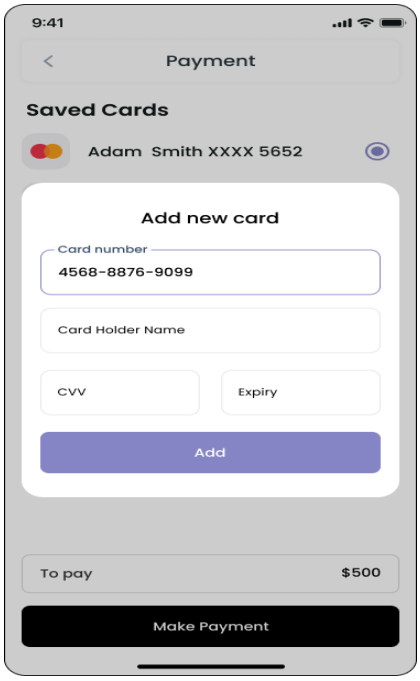

Fig. 3.5 Payment using card

The user will be choosing each time from which card he wants to make a payment, if he wants to add a new card, a dialogue box will appear in the middle of the screen, where he needs to fill in the card details. There is a proper validation done in the front-end, if the card details are correct or not. The card details whether the card exists or not, the card details match the existing card or not is done on the backend using Stripe, we will be discussing this technology in the further section of this chapter.

The user of this application will also get a number of features like, to see and edit profile, give rating and reviews to the Nanny, order history and notification.

Source Code of Booking Nany

```
const { validationResult } = require("express-validator");

const db = require("../models");

const sequelize = require('sequelize');

const Feedback = db.Feedback;
```

```
const Booking = db.Booking;
const Parent = db.Parent;
const Nany = db.Nany;
const User = db.User;

exports.feedBack = async (req, res, next) => {
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array() });
  }

  const booking = await Booking.findOne({ where: { id: req.body.booking_id } });

  const parent = await Parent.findOne({ where: {user_id: req.userid}});
  const nany = await Nany.findOne({ where: {id: booking.nany_id}});

  const obj = {
    parent_id: parent.id,
    nany_id: booking.nany_id,
    booking_id: booking.id,
    stars: req.body.stars,
    comment: req.body.comment,
  }
```

```
  const feedback = await Feedback.create(obj);

  if(feedback) {
    const average_rating = await Feedback.findAll({
      where: {nany_id: booking.nany_id},
      attributes: [[sequelize.fn('AVG', sequelize.col('stars')), 'avgRating']]
    });

    nany.average_rating = average_rating[0].dataValues.avgRating;
    nany.save();
  }

  return res.status(200).json({
    feedback: feedback,
  });
};

exports.listFeedbacks = async (req, res, next) => {
  const parent = await Parent.findOne({ where: {user_id: req.userid}});

  const feedbacks = await Feedback.findAll({
    where: { parent_id: parent.id },
    include: [{
      model: Parent,
```

```
      include: {
        model: User,
      }
    },{
      model: Nany,
      include: {
        model: User,
      }
    }],
    order: [["createdAt","DESC"]]
  });

  return res.status(200).json({
    feedbacks: feedbacks,
  });
};

exports.getNanyInfo = async (req, res, next) => {
  const booking = await Booking.findOne({
    where: {
      id: req.params.booking_id
    },
    include: [
      {
```

```
        model: Nany,
        include: {
          model: User,
        }
      }
    ]
  });

  return res.json({
    booking: booking
  });
}
```

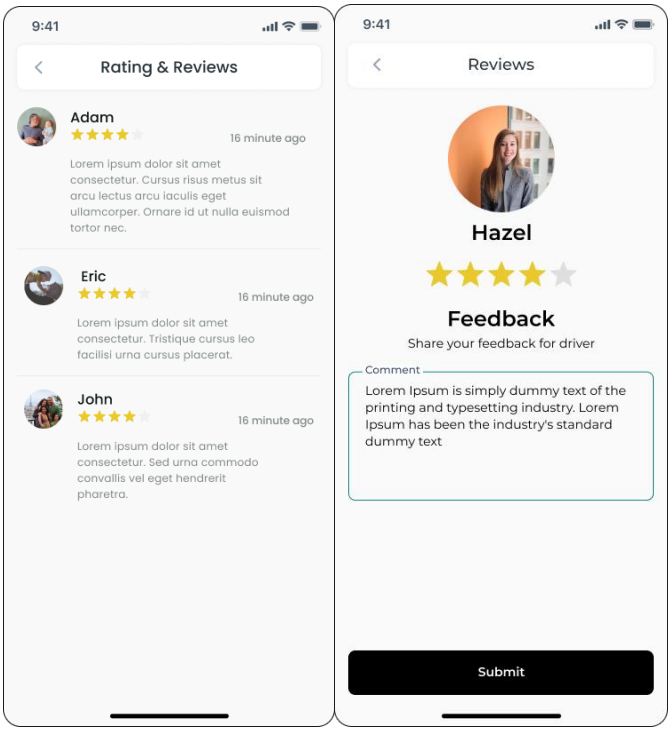

Fig. 3.6 Rating and Review to Nanny

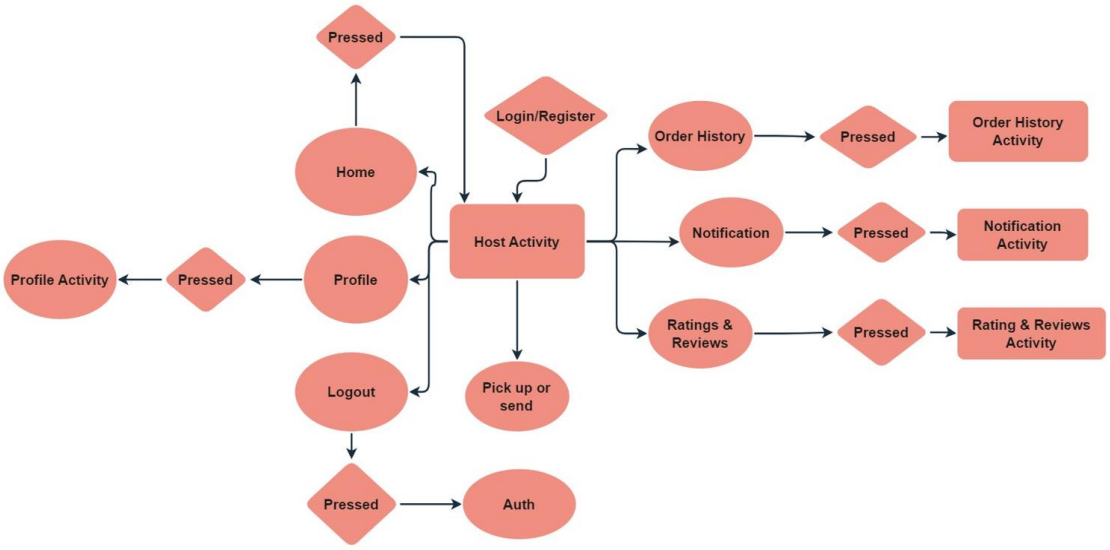

Fig 3.7 ER diagram of application

3.3 Development

This section focuses on thorough discussion of the building process of our application. It also contains the tools and technologies used, as well as the challenges faced during the creation portion of the project on Xcode application. This contains the detailed information of the technologies used in this project.

- Swift
- Xcode
- Cocoapods
- Node.js
- Express.js
- Firebase

Now, we will discuss each and every technology in detail and how we have used it in our project from initialising our application to adding features to the application and use of these technologies in our application at various stages.

Xcode

Xcode is an integrated development environment (IDE) created by Apple Inc. for developing software applications for macOS, iOS, iPadOS, watchOS, and tvOS platforms. It includes a suite of software development tools that allow developers to create and test applications, as well as publish them on the App Store. One of Xcode's key features is its user-friendly interface that makes it easy for developers to navigate and customise their workspace. Xcode provides a range of tools for creating graphical user interfaces, including interface builder, which enables developers to create application layouts visually, as well as storyboard, which allows developers to design the flow of an application and its individual screens.

Xcode also features a source code editor that provides a range of powerful editing features, including syntax highlighting, autocomplete, and refactoring tools. The editor supports multiple programming languages, including Swift, Objective-C, C++, and Python. The debugger in Xcode is another key feature that helps developers identify and fix errors in their code. The debugger provides real-time feedback on application behaviour, allowing developers to track variables and memory usage, set breakpoints, and step through code.

Xcode also includes a suite of performance analysis tools that help developers optimise their application's performance. These tools allow developers to identify performance bottlenecks, such as CPU usage and memory usage, and provide detailed analysis reports to help developers fine-tune their code. In addition, Xcode offers testing tools that enable developers to test their applications in a variety of scenarios, including user interaction, network connectivity, and performance under stress.

Overall, Xcode is a powerful and versatile tool for developers working on Apple's various platforms. Its comprehensive suite of tools for designing, coding, debugging, and testing applications make it an essential component of the software development process for iOS, iPadOS, watchOS, tvOS, and macOS applications.

Swift

Swift is a general-purpose, open-source programming language developed by Apple Inc. It is designed to be safe, fast, and modern. Swift is used to build apps for iOS, macOS, watchOS, and tvOS. Swift is built with a syntax that is easy to read and write, which makes it approachable for developers new to programming. It also supports a range of programming paradigms, including object-oriented, functional, and imperative styles. With its safety features like optionals, type safety, and automatic memory management, Swift helps prevent common programming errors and improves app performance. Overall, Swift is a powerful and versatile language that is ideal for building a wide range of applications.

CocoaPods

CocoaPods is a dependency manager for Swift and Objective-C Cocoa projects. It simplifies the process of adding external libraries and frameworks to iOS projects, as developers can easily search for and add third-party libraries to their projects through a simple command-line interface. CocoaPods automatically handles versioning and updates for these libraries, making it easier for developers to maintain and update their code. Additionally, CocoaPods allows for better collaboration between developers by making it easy to share and reuse code. With its extensive library of community-contributed pods, CocoaPods has become a popular tool for iOS developers seeking to streamline their development process and improve the quality of their code.

StoryBoard [1]

A storyboard is a visual representation of the user interface of an iOS application, showing screens of content and the connections between those screens. A storyboard is composed of a sequence of scenes, each of which represents a view controller and its views; scenes are connected by segue objects, which represent a transition between two view controllers.

Xcode provides a visual editor for storyboards, where you can lay out and design the user interface of your application by adding views such as buttons, table views, and text views onto scenes. In

addition, a storyboard enables you to connect a view to its controller object, and to manage the transfer of data between view controllers. Using storyboards is the recommended way to design the user interface of your application because they enable you to visualise the appearance and flow of your user interface on one canvas.

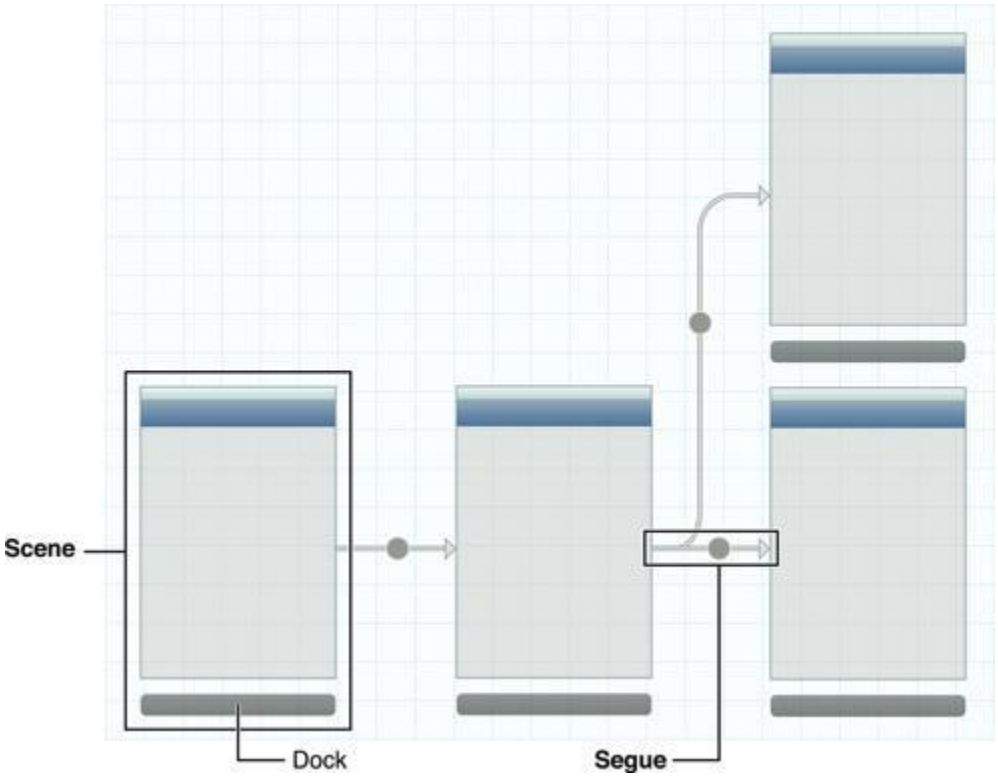

Fig 3.8 Scene Corresponds to a Single View Controller and Its Views

On iPhone, each scene corresponds to a full screen's worth of content; on iPad, multiple scenes can appear on screen at once—for example, using popover view controllers. Each scene has a dock, which displays icons representing the top-level objects of the scene. The dock is used primarily to make action and outlet connections between the view controller and its views.

As with all objects loaded from a storyboard, to finish initialising a view controller loaded from a storyboard you override awakeFromNib.

Node.js [2]

Node.js is a cross-platform, open-source server environment that can run on Windows, Linux, Unix, macOS, and more. Node.js is a back-end JavaScript runtime environment, runs on the V8 JavaScript Engine, and executes JavaScript code outside a web browser.

Node.js lets developers use JavaScript to write command line tools and for server-side scripting. The ability to run JavaScript code on the server is often used to generate dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm,[6] unifying web-application development around a single programming language, as opposed to using different languages for the server- versus client-side programming.

Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimise 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).[7]

The Node.js distributed development project was previously governed by the Node.js Foundation,[8] and has now merged with the JS Foundation to form the OpenJS Foundation. OpenJS Foundation is facilitated by the Linux Foundation's Collaborative Projects program.

In this project Node.js is used for building the complete back-end of this project. The features of Node.js for choosing it over other technologies are as follows:

- **JavaScript Runtime:** Using the framework of Node programmers are able to execute the code that uses JavaScript independently of a web browser. As a result, programmers can apply JavaScript to create the server-side portion apps, command-line tools, and other kinds of software.
- **Event-Driven Architecture:** Node.js's event-driven, autonomous I/O approach enables it to manage many concurrent connections without delaying other requests. Because of this, Node.js is very extensible and ideally suited for creating applications that operate in real time, such as chat, gaming, and other kinds of apps that need data to be processed immediately.
- **Support for Multiple Databases:** Both relational and non-relational databases are supported by Node.js. This makes it simple to select the ideal database for your application in accordance with your unique demands and specifications.
- **Simple to Learn:** For programmers who are already comfortable with JavaScript, for them Node.js is rather simple to learn and implement. This implies that developers won't need to

learn an entirely novel programming language or environment in order to get started immediately developing server-side apps leveraging Node.js.

- Node.js was developed on top of the V8 JavaScript engine, that has been substantially optimised for speed and is compact. Node.js is hence able to operate quickly and consume less of the system's resources compared to other server-side platforms. Asynchronous programming, or asynchronous programming, is another feature of Node.js that enables programmers to create quick-running code.

- **Wide-ranging Modules:** Wide range of modules and packages are created and maintained by a huge and active community: Node.js has a tremendous and engaged community of contributors. To offer additional capabilities and features, these extensions may be readily incorporated into applications written in Node.js.

Express.js

Express.js serves as a quick, lean web framework based on Node.js that offers a number of advantageous features for creating APIs and online apps. It is based on Node.js and offers a simple API that makes the procedure of creating web and mobile apps easier. The server-side functions offered by Express.js may be utilised to carry out a variety of activities, including processing requests that come in, authorising users, and managing exceptions. Complex request-response flows may be made by connecting these services provided by middleware in a pipeline.

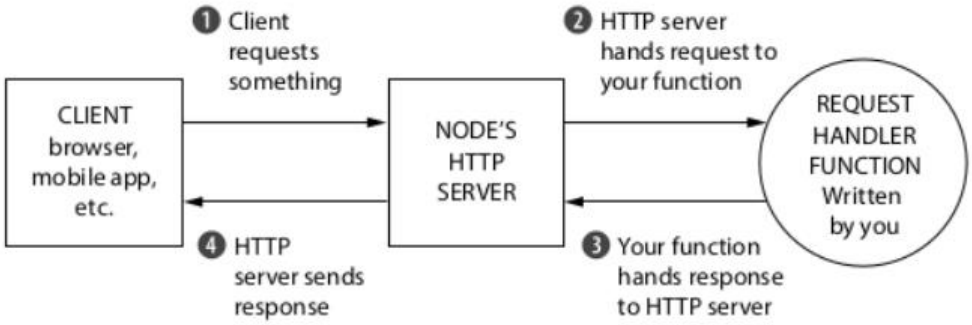

Fig. 3.9 Flow of requests and response from client to server

Express.js is frequently used to create APIs, or interfaces for application programming, which enable interaction between different parts of a programme. A common method for software to communicate data and services is provided by an API, which facilitates the development of sophisticated software platforms. Express.js offers a straightforward and understandable API that makes creating APIs simple. Developers may create functions that handle GET, POST, PUT, and DELETE requests by defining routes that correlate to the various HTTP methods. A variety of intermediate operations, including processing receiving JSON data, authorising users, and managing errors, are available in Express.js. A RESTful API, which is an architectural framework for developing APIs that adheres to a set of restrictions, may also be built using Express.js. RESTful APIs modify resources (like data objects) while offering replies in an accepted format (like JSON) via HTTP methods.

Security and scalability are crucial factors to take into account while developing an API with Express.js. Data in transit may be made more secure with the support for SSL/TLS encryption that Express.js offers. Additionally, it encourages rate limiting along with other attack-prevention strategies. Benefits of using Express.js for building APIs in iOS apps:

- **Scalability:** Express.js is very scalable and can handle many concurrent connections without experiencing any lag. It can be set up on a group of servers or a platform that uses the cloud, like AWS or Google Cloud.
- **Security:** Express.js supports SSL/TLS encryption, which helps to protect data while it is being transmitted. Additionally, it encourages limitation of rate and other attack-prevention strategies.
- **Large Community:** Substantial and prominent developer community: Express.js has a sizable and active developer ecosystem that builds and maintains a variety of modules and libraries that are readily incorporated into Express.js programmes to bring additional features and possibilities. Additionally, the platform has excellent documentation that makes it simple for developers to comprehend and use.

Google Firebase

Google has created a platform called Firebase that offers an array of tools for developing and expanding mobile and web applications. Due to this it provides a variety of services and tools that

can enable development and quicken time-to-market, it is a well-liked option for iOS developers. Fig 3.9 shows the list of features provided by the firebase in the built section.

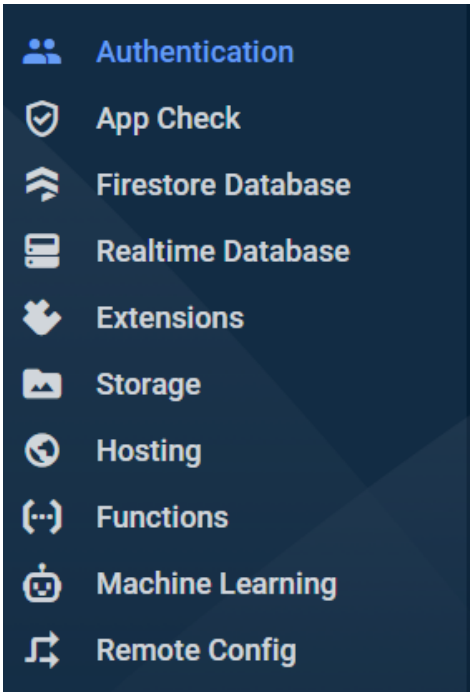

Fig 3.10 List of Firebase features

Key details regarding Firebase for iOS applications is provided below:

1. To save and sync data instantly, one option is to use Firebase Realtime Database, a cloud-hosted database. It provides a NoSQL database, making it easy to manage and maintain data in a flexible, scalable way. The Firebase Live Database can be used to construct applications that respond instantly to data changes, enhancing their interactivity and user attractiveness.

2. A quick and easy approach to add authentication to your iOS application is with Google Authentication. A variety of authentication techniques are supported, including social media, phone, email and password, and more. You may create safe apps that need authentication from users and permission with Firebase Identification. In this project we are working only on one kind

of authentication and that is phone number authentication. It generates OTP and sends it on the entered mobile number. This OTP has a variable life span for more security.

3. User-generated material with a large size, such photographs and videos, may be easily stored and served with Firebase cloud-based storage. It allows for both local and remote storage, allowing access to material from any location in the globe. For a full storage solution for your iOS application, Firebase Cloud Storage interfaces with other Firebase services like Authorization and Dynamic Database.

4. Google Firebase also provides a feature called FCM (Firebase cloud messaging). Delivering messages to clients on Android, iOS, and the web is made possible by the dependable and scalable Firebase cloud-based messaging service. It offers a variety of capabilities, including targeting, planning, and statistical analysis, making it simple to give people timely and relevant communications. It is a popular option for iOS due to its simplicity of use, scalability, and interaction with other Google services.

Stripe for Payment

A service for handling payments called Stripe makes it possible for companies of any kind to receive and handle payments online. It delivers a variety of services and applications that enable businesses to make payments easily and effectively. Many companies, which include entrepreneurs, small enterprises, and large corporations, use Stripe on a global scale. Stripe works for all kinds of payments starting from card payment to UPI payments, card payments also includes all kinds of cards, i.e. master-card, visa-card, in debit card and credit card options. Some key features of Stripe are given below:

- **Payment methods :** Online payment processing is made simple and safe by Stripe. Numerous payment options, such as debit and credit card transactions, and payments via mobile devices are supported. All aspects of payment processing, such as identifying fraudulent transactions, reimbursements, and currency conversions, are handled by Stripe. Additionally, it supports memberships and periodic payments, making it simple for businesses to handle customers' payments over time.

- **Security :** Stripe prioritises safety and delivers a variety of tools to support organisations in keeping their transactions safe. It holds a PCI Level 1 Service Provider authorization, which is the highest kind of validation available to payment processors. Additionally, Stripe's platform offers fraud identification and avoidance tools like multiple-factor authentication, real-time risk assessment, and machine learning algorithms.

- **Global Reach :** With around 135 recognised denominations and over 40 operating nations, Stripe makes it simple for business establishments to take payments through clients all around the globe. International payments are handled entirely by Stripe, involving conversions of currencies and observance of regional laws.

- **Billing :** A set of software programmes called invoicing makes it simple for companies to handle membership and payments that recur. It offers capabilities for creating price structures, maintaining users and payments, and responding to unsuccessful payments. To offer a complete billing solution, Stripe Billing connects with other Stripe services like Payment Processing and Hawkeye.

Organisations can handle and accept payments via the internet with ease thanks to a variety of products and services offered by Stripe. Stripe offers a complete platform for managing payments in your iOS application with payment processing, developer tools, security, global reach, billing purposes, and detection of fraudulent transactions solutions. Businesses across all kinds choose it because of its versatility, protection, and simplicity of use.

Chapter - 4

Conclusion

4.1 Conclusion

In conclusion, a babysitting app can be a great tool for parents and babysitters alike. With the increasing demand for trustworthy and reliable childcare, such an app can help connect parents with qualified and experienced babysitters. The app can include features such as a secure payment system, background checks for sitters, and real-time GPS tracking for parents to ensure their child's safety. Additionally, the app can offer features such as scheduling, messaging, and ratings and reviews to facilitate communication and build trust between parents and sitters. Overall, a well-designed and user-friendly babysitting app can provide peace of mind for parents and create a convenient and efficient platform for babysitters to find work.

4.2 Future Scope

As societies become increasingly advanced, the market for at-home services continues to expand. There are numerous ways to upgrade and improve this app.

Here are some potential future scope points the app:

1. In-app video conferencing: As remote work and online schooling become more prevalent, parents may want to be able to check in on their children and communicate with babysitters through the app.
2. AI-powered matching: Artificial intelligence algorithms could help match parents with babysitters based on their preferences, availability, and location. This could make it easier for parents to find the right babysitter quickly and efficiently.
3. Integration with smart home devices: Babysitting apps could potentially integrate with smart home devices like cameras, thermostats, and door locks, allowing parents to monitor their children remotely and control their home environment.

4. On-demand services: Similar to other on-demand services like food delivery or transportation, a babysitting app could offer on-demand babysitting services for parents who need immediate or last-minute childcare.
5. Educational resources: Babysitting apps could offer resources for parents and sitters to learn about child development, safety, and other relevant topics.
6. Virtual activities: As more activities move online, babysitting apps could offer virtual activities for children to do with their sitters, such as interactive games, educational activities, or storytime.
7. Parenting support: Babysitting apps could offer parenting support resources for parents, such as access to parenting coaches or online support groups. This could help parents navigate challenges and get advice from experienced professionals.

References

[1]https://developer.apple.com/library/archive/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html

[2] https://en.wikipedia.org/wiki/Node.js25