SplitPay:
A Payments Ledger Application

J COMPONENT PROJECT REPORT

Special Summer Semester 2022-2023

Submitted by

Rishabh Nagar (19BCE0722)
Shrivats Poddar (19BCE0750)
Sarthak Bajaj (19BCE0710)

in partial fulfilment for the award of the degree of

B. Tech

In

Computer Science and Engineering

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Vellore-632014, Tamil Nadu, India
School of Computer Science and Engineering
April 2022

TABLE OF CONTENT

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to Professor Akila Victor for her expert advice and encouragement throughout this project and for guiding and clarifying all my doubts as well as for his support in the lab. I would even like to thank my institute, Vellore Institute of Technology, Vellore, for giving me this golden opportunity in the course of Information Security Management to create this project and explore different tools. Most of all, my deepest thanks go to my parents and my family. I cannot thank them enough for their love, support, sacrifice, and their belief in me.

ABSTRACT

With the advent of technology and personal smartphones, many real-world avenues have turned to technical solutions like apps and websites. One such field is that of Banking and Payments. In this day in age a lot of people are using smartphones and nowadays every person wants their smartphone to have all the applications that can help them in everyday life, one of those applications that we have seen has risen in use are the Payments app or E-wallets. We have seen payments apps like PayTM, Google Pay and Phone Pay come to light in recent years. As students of a university, we don’t like to use cash as the primary way of paying either shopkeepers or our friends so we are usually using these E-wallets and Payments App as the primary source of spending our money. However, we face difficulties when it comes to splitting bills, which is the case many times.So, we applied a user-centered design approach to develop an e-ledger application which not only supports maintaining payment records but also lets a group of users split a payment amongst themselves, and settle them at their own ease, within the app.

CHAPTER I : INTRODUCTION

We have developed an interface to allow users, mainly university students, to record and maintain payments ledgers to avoid and resolve any conflicts regarding the settlement of payments. With this application we aim to provide an easy and comprehensive solution to any payment settlement problems. Users can choose to record a single payment to a person or choose to split it, a unique feature of the app.

One unique feature that we have added to this application which is not present in any of the applications present in the current day is A SplitPay – functionality. In this SplitPay functionality users can split their bills among their friends or colleagues, i.e. if more than one person has purchased some amount of goods, for e.g., food, then they have the ability to split their bill according to the amount of food that each person eats.

CHAPTER II(A)
REQUIREMENT ANALYSIS

The first step of the project was to employ two kinds of data collection methods to understand the needs of our primary stakeholders better. We chose to conduct User Interviews and an Online Survey.

2.1. USER INTERVIEWS

The interviews helped us understand the unique behaviours of a small number of people which let us find specific details about the way they perceive the current online payment platforms. We conducted user interviews with 6 people. We reached out to our friends/classmates to gain insights about the processes and mechanisms they follow while paying bills and splitting expenses.

The following is the list of questions we asked during these interviews:

1. What are the online payments apps you have experience of using? Tell me about the last time you used one.
2. What was the kind of setting? Were you with friends or alone?
3. Do you face any issues when using an payment app? What are they?
4. Do you prefer using a payments app over paying Cash or by Card?
5. What is the scale of amounts that you regularly pay? Would you use a payment app to transfer larger amounts of money?
6. Do you trust these apps to keep your money safe?
7. Would you recommend someone else to use a payments app in day to day life? Why?
8. When was the last time that you can remember splitting a bill? Tell me about that experience.

9. How many of you were there when you split the bill?
10. Do you frequently split bills with these same people?
11. What is the way in which you split? Do you use any tools? Tell me the process that goes into it.
12. Do you split equally amongst all, by what each person has consumed, or does it vary from time to time?
13. Are you aware about any platforms or mechanisms which can help you split bills with people?

These were semi-structured interviews, so we often cross-questioned the interviewees on the basis of their answers.

In summary, the following are the insights we gained from these interviews:

• People generally found e-wallets and online payment as a more convenient means of payment as compared to more conventional ones. All of them would recommend the use of such platforms for day-to-day expenses. This means there is a good scope for the usage of such a platform.
• Their major issues with such apps are concerned with server errors. While most are satisfied with the usability of the big scale apps, various apps come with various difficulties in setting up these apps and linking them with other services. Some smaller scale apps have some usability issues as well. So, it is very important to have a highly usable design to ensure user satisfaction. People
• People generally find themselves using Cash or Card for larger amounts due to more security during the payment process. One of the interviewees also voiced concern as he may not be sure if the large amount has been transferred. Still many are satisfied with the security measures in a payment app when it comes to smaller amounts.
• One of the six interviewees very rarely uses such apps. He voiced his reasons as cash being reliable enough for him. He usually uses such apps in social situations where expenses need to be split.
• All the interviewees told us that they find themselves splitting bills in social situations, like when they meet for meals, when they travel, etc. So, there is always a need to have an efficient way to split payments on-the-fly.

• They usually find themselves splitting bills in the same groups (i.e. same people) multiple times. So, this product should have a functionality to create groups among which payments can be split for longer durations.
• Most interviewees spend long durations of time splitting expenses manually using penpaper or calculators. So, a convenient digital way would help them do this task faster and keep track of the same for later, as many find that in such situations one or two people pay the entire bill and the rest pay back the dues later.
• They don't always split the bill equally; They may split according to each person's share. This means the platform requires a mechanism to enable uneven splits. Moreover, one or two people may pay upfront which is returned to them later.

2.3.CONCLUSIONS DRAWN FROM REQUIREMENT ANALYSIS

We have a lot of insight into how our target users behave while making payments, on their own, or in social situations. This can help us in clearly defining our project’s goals, scope and requirement:
Goal: The goal of this project is to develop a mobile payment platform (an application) which lets its users easily pay for expenses and split those expenses amongst groups of two or more. These splits should occur in whichever way the user demands as there is no fixed way of splitting bills.
Problems in Existing Systems: Existing systems include big scale payment apps like PayTM and Google Pay. Their users are largely satisfied with their services. However, all such apps lack the ability to split expenses, which would be unique in our product. Moreover, by learning from the mistakes of smaller scale payment apps we are implementing a user centred approach to understand what the users want.
Context of use of the Design: This design will look to aim young adults as its primary users. It is an app which would be used in a social context, when people go out and spend money as a group. Moreover, due to the Coronavirus situation in 2020, this app also promotes Contactless Payments and thus can be used as a safe means of payment

2.4 FUNCTIONAL REQUIREMENTS

- Create an account/register into the system.
- Manage their account.
- Log in to the system.
- Make a new payment
- Split payment
- Add user into split billing system
- View account balance/history
- Request payment
- Send payment reminder

2.5 NON FUNCTIONAL REQUIREMENTS

- We plan to use HTML CSS for frontend
- JS, nodejs, expressJS for the backend and
- Mongodb for database management.
- This product is comprised of several modules, which include modules like viewing account balance and making payment which are similar to other payment portals and apps available in the market,
- Whereas it also has some unique modules like splitting bills, and adding users/contacts as partial bill payers in a single transaction.

CHAPTER II (B)
LITERATURE SURVEY

The Inefficiency of Splitting the Bill
Uri Gneezy, Ernan Haruvy and Hadas Yafe

In this paper unscrupulous diner’s dilemma is discussed which is a problem faced frequently in social settings. When a group of diners jointly enjoys a meal at a restaurant, often an unspoken agreement exists to divide the check equally. A selfish diner could thereby enjoy exceptional dinners at bargain prices. Whereas a naive approach would appear to suggest that this problem is not likely to be severe, it appears that even the best of friends can sometimes find it rather severe. This paper observes and manipulate conditions for several groups of six diners at a popular dining establishment. In one treatment the diners pay individually; in a second treatment they split the bill evenly between the six group members. In yet a third treatment, the meal is paid for entirely by the experimenter. Economic theory prescribes that consumption will be smallest when the payment is individually made, and largest when the meal is free, with the even split treatment in-between the other two. The restaurant findings are consistent with these predictions. A fourth treatment, in which each participant pays only 1/6 of her own consumption costs and the experimenter pays the remainder, is introduced to control for possible unselfish and social considerations. The marginal cost imposed on the participants in this treatment is the same as in the even split treatment. However, the externalities are removed: in the even split case, increasing an individual’s consumption by $1 increases the individual’s cost, as well as the cost of each of the other participants, by $1/6. In the fourth treatment, this will increase only the individual’s cost by $1/6 but will have no effect on the payment of the other participants. In other words, the negative externality present in the even-split treatment is completely eliminated. If participants are completely selfish, the fourth treatment should not affect their consumption relative to the second treatment of the even split. On the other hand, if they care also for the well being of the other participants (or for social efficiency), they can be expected to consume more in the last treatment than in the even split treatment.

Accounting system on cloud: a case study
Mozhdeh Sadighi

Facilitating user’s access (organizations or individuals) to the IT (Information Technology) resources was the main idea leading to cloud computing. By this concept users can gain access to all their applied hardware and software via internet or intranet and without need to own them. Accounting software is application software which permits accountants to do financial activities like recording and processing transactions electronically. Accounting software are programs used to maintain books of account on computers. The software can be used to record transactions, maintain account balances, and prepare financial statements and reports. In the past, all these activities should have been performed by hand. So, the development of accounting software (also called financial software or budgeting software) was a great progress in accounting. Businesses started applying the software to increase speed and precision and decrease costs of works on paper. Generating immediate financial reports is another benefit of applying this software.

Cloud-based accounting software: choice options in the light of modern international tendencies

Yuliia Popivniak

The purpose of the article is to study the international experience of cloud accounting software use, its strengths and weaknesses, threats and opportunities and to develop approaches to the choice of cloud services at the enterprises. The methodology which was used for analysing the advantages and disadvantages of implementing accounting software based on cloud technologies is SWOT analysis.

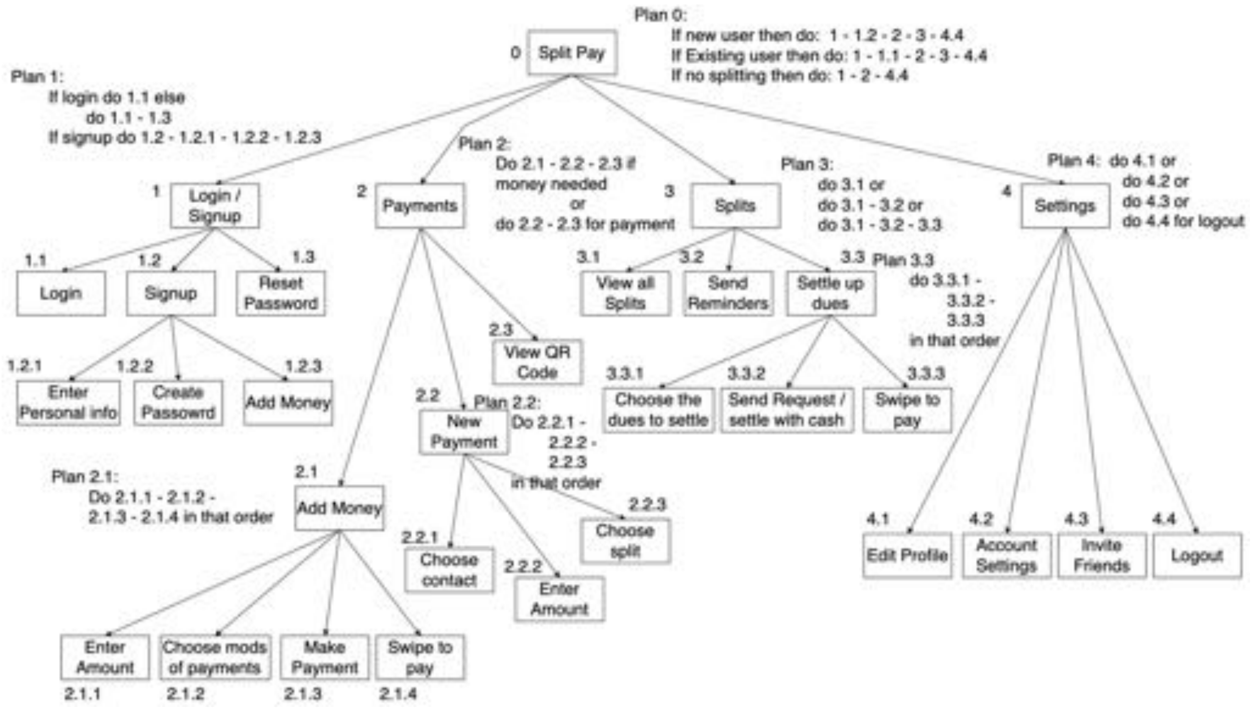
A study on cloud accounting software applications: awareness & acceptance among accounting professionals w.r.t Mumbai region

Asst. Prof. Dr. Neelam Shaikh

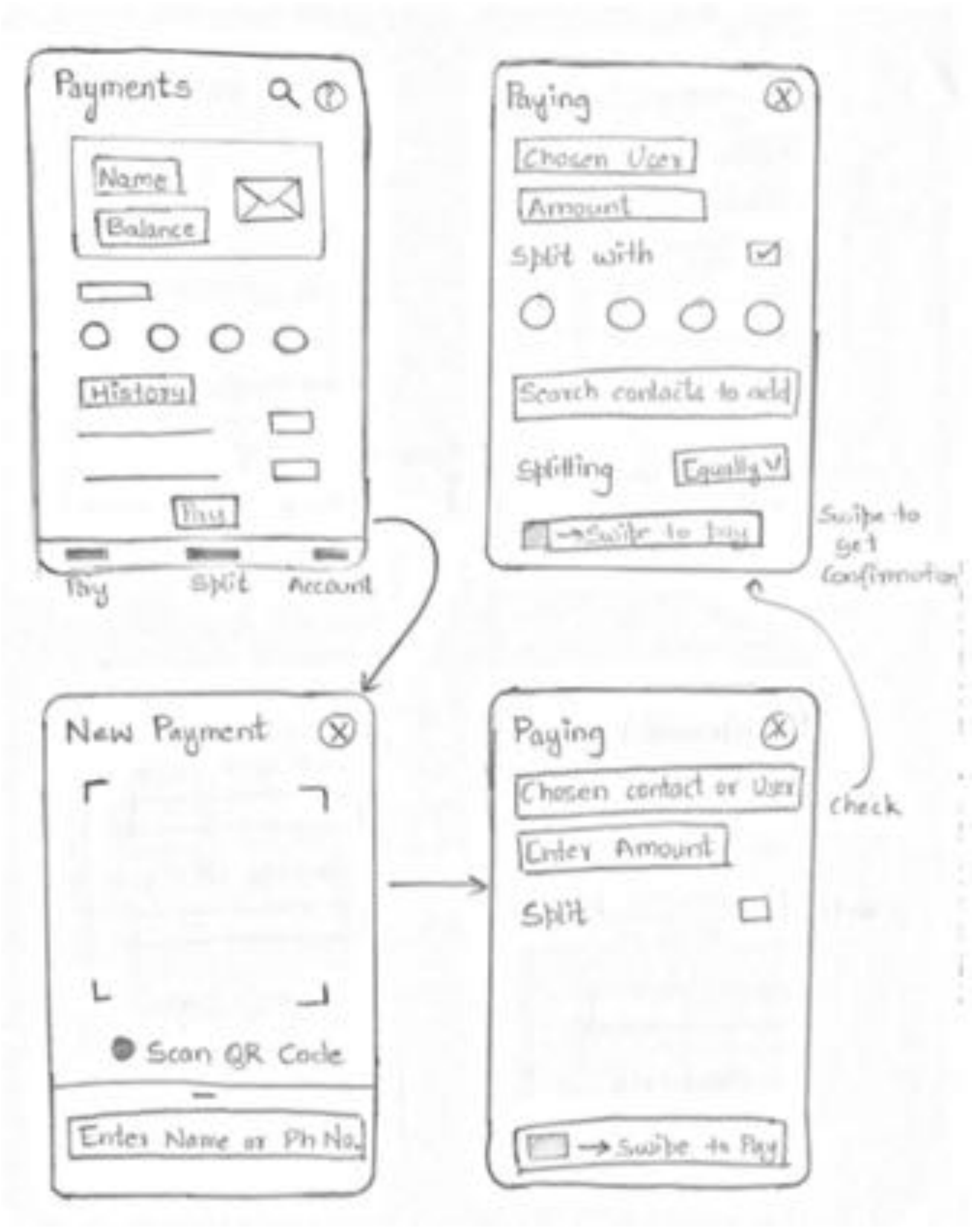
The scope of the research is to understand the cloud accounting software. And its adoption among the accounting firms in Mumbai. 1. The research is based on primary as well as secondary data. 2. This research is studied broadly about the various cloud accounting software and its features. This Explanatory Study Research was constructed with Descriptive (Quantitative) methods. Conducting quantitative research involved survey questionnaires. Cloud accounting software utilizes the Cloud to store accounting data. It makes financial information accessible to owners and employees anywhere with an Internet connection. Every day, more and more businesses are turning to cloud computing. From connected appliances to Internet-based education programs, people all over the world are utilizing the cloud as a way of connecting with customers and making their own business practices more effective. The rapid development in technology creating a lot of opportunities to accounting professionals, but there is gap between the rapidly growing technologies and usage of those technologies by accounting professionals.

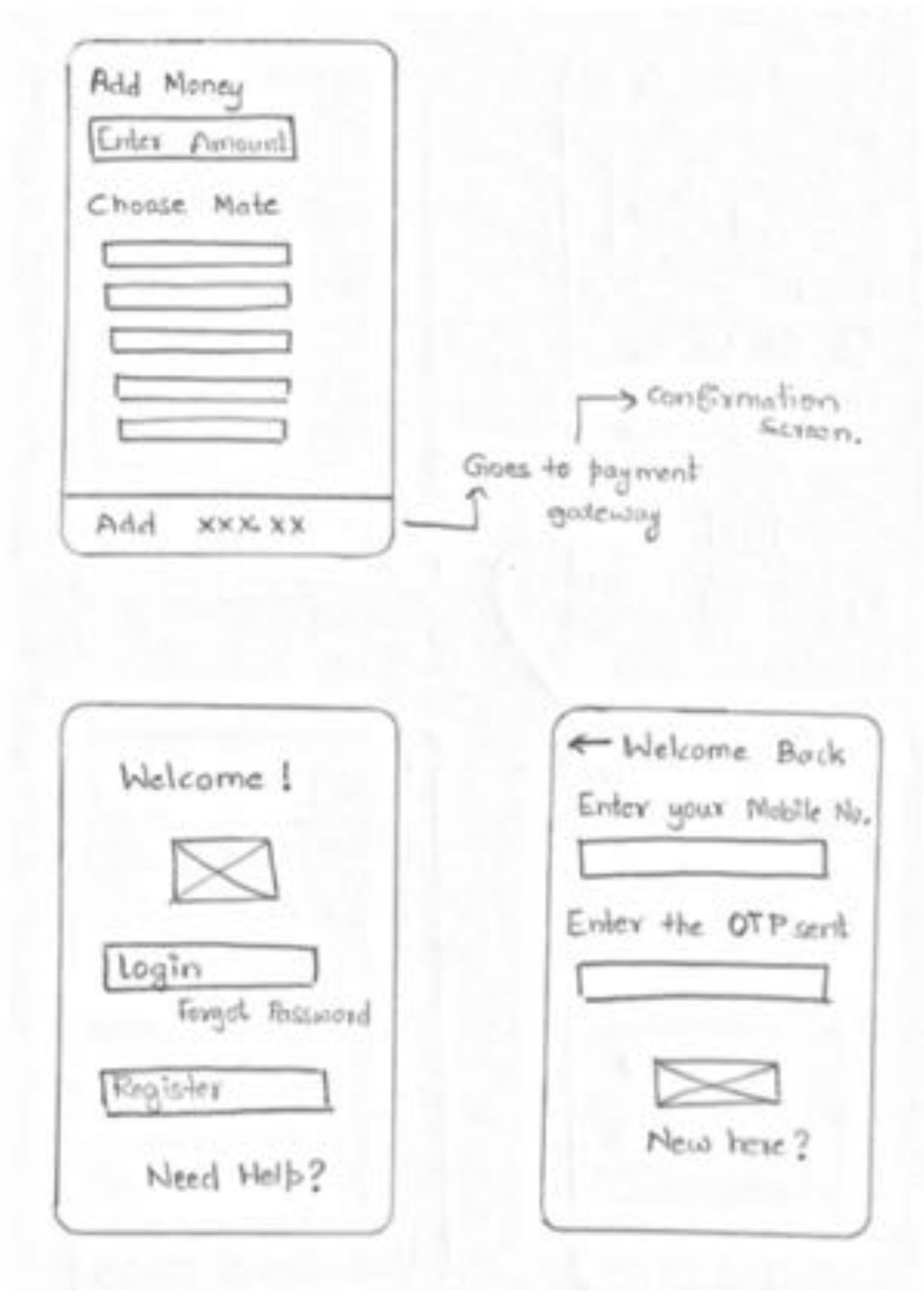
CHAPTER III:
DATAFLOW

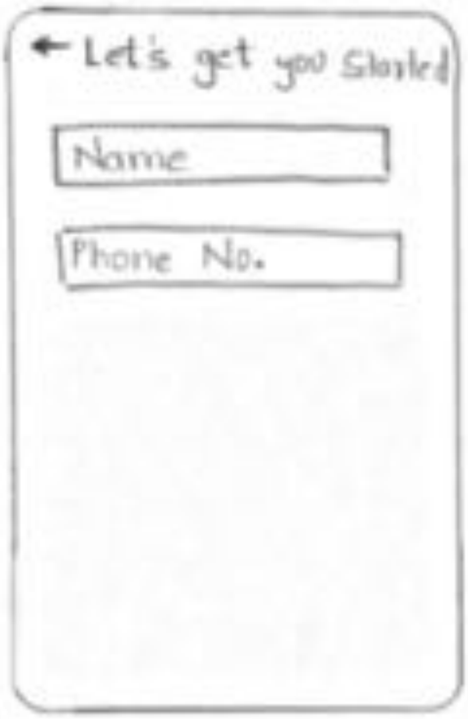
3.1 HTA



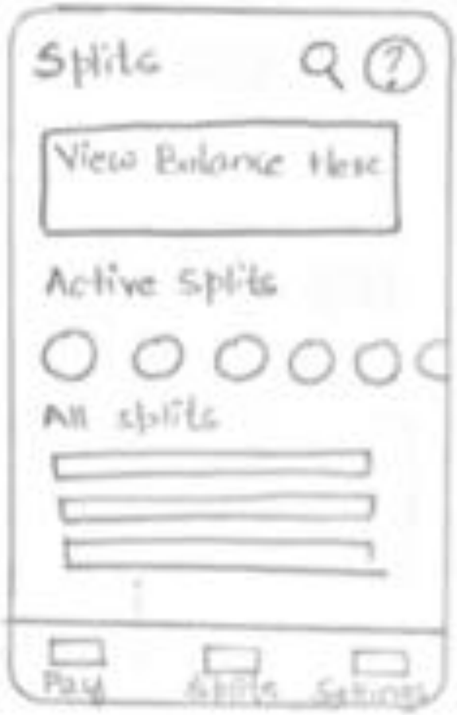
3.2 STORYBOARD



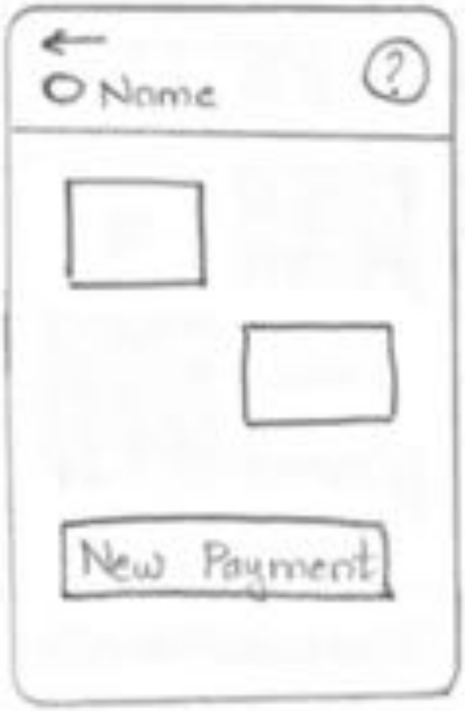

Add Money
Enter Amount
Choose Mode
Add xxx.xx
Goes to payment gateway
confirmation screen.
Welcome !
Login
Forgot Password
Register
Need Help?
← Welcome Back
Enter your Mobile No.
Enter the OTP sent
New here?



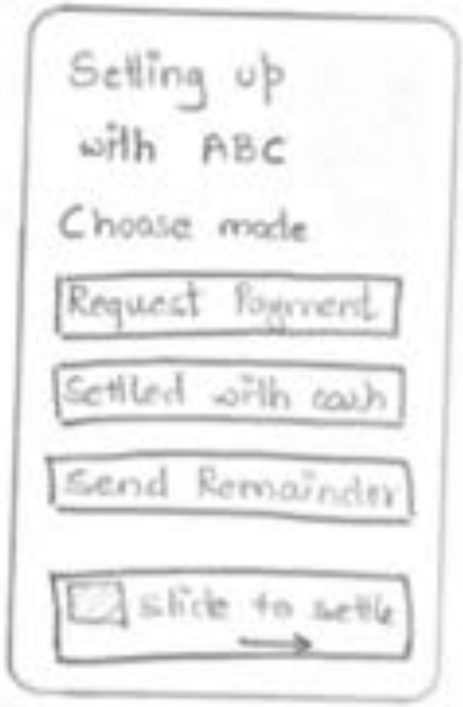
← Let's get you Started
Name
Phone No.



Splits
View Balance Here
Active Splits
All splits
Pay
Splits
Settings



←
Name
?
New Payment



Settling up
with ABC
Choose mode
Request Payment
Settled with cash
Send Remainder
slide to settle

We have a lot of insight into how our target users behave while making payments, on their own, or in social situations. This can help us in clearly defining our project’s goals, scope and requirement.

GOAL: The goal of this project is to develop a mobile payment platform (an application) which lets its users easily pay for expenses and split those expenses amongst groups of two or more. These splits should occur in whichever way the user demands as there is no fixed way of splitting bills.

PROBLEMS IN THE EXISTING SYSTEM: Existing systems include big scale payment apps like PayTM and Google Pay. Their users are largely satisfied with their services. However, all such apps lack the ability to split expenses, which would be unique in our product. Moreover, by learning from the mistakes of smaller scale payment apps we are implementing a user centred approach to understand what the users want.

CONTEXT OF THE USE OF DESIGN: This design will look to aim young adults as its primary users. It is an app which would be used in a social context, when people go out and spend money as a group. Moreover, due to the Coronavirus situation in 2020, this app also promotes Contactless Payments and thus can be used as a safe means of payment.

CHAPTER IV:
DESCRIPTION OF MODULES

4.1 USER PROFILES

	Bio	Goals	Frustration
USER 1 STUDENT OF VIT UNIVERSITY 	Rohit is a 20 year old student from VIT Vellore and has planned to go out on a chennai trip with his 5 other friends.	He wants to make sure that he gets an ease method to pay his expenses instead of cash everywhere and also he gets to keep the track of who paid what money and for whom, so that they can collect the total sum at the end from that person.	He tries to keep the track of money in his mobile notes as well as in the rough notes, yet sometimes he misses out small amount and gets confused who owes him some money and he owes someone how much.

Users	Bio	Goals	Frustration
USER 2 Co-Owner 	Jatin is the co-owner of a apartment at the Periyar flats in Vellore. He wants that he can easily pay his part of the property tax without having to send or take the other part of the tax from his other Co-owner.	He want to have a flexible system that enables him to split his property tax with his other Co-owners. Moreover, he want this process to be happening through the app and not being concerned about cheques of other co-owners being bounced due to various reasons.	His main concern is the payment of tax on time in order to avoid getting fined. This includes delays due to non-payment by other co-owners and also when other medium of transfer like cheque is used, the happening to check bouncing.

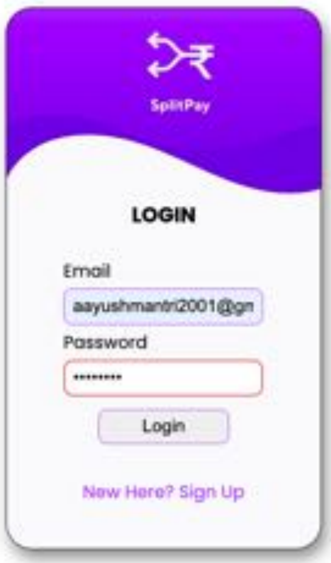
Users	Bio	Goals	Frustration
 Shopkeeper Outside VIT	aayush is a 33 year old shopkeeper working for a store(ALLMART) outside VIT and usually his task is to handle transaction domain and billing counter.	He wants to take safety precautions after pandemic and make sures that the transaction should be done in an indirect mode without any contact.	Since, he is more likely prone to virus because of getting in physical contact with the customer and moreover at the billing counter splitting bills into people from same group makes it a tedious procedure for him.

Main Features of our model includes:

- Register / Login / Logout
- Edit profile details
- Add Money
- Initialise P2P Payment / Initialise Split-Payment
- View Pending Payments
- View Transaction History

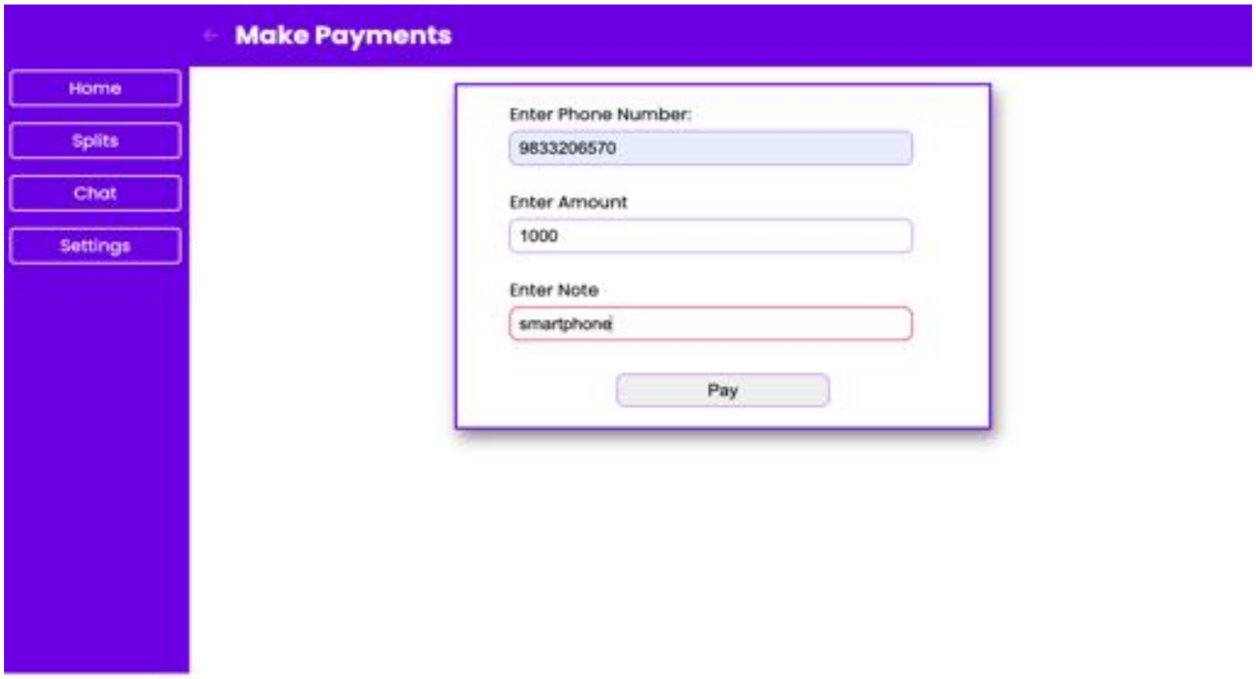
CHAPTER V:
IMPLEMENTATION

5.1 SCREENSHOTS



Email: aayushmantri2001@gm
Password: •••••••••

LOGIN PAGE



Enter Phone Number:

9833206570

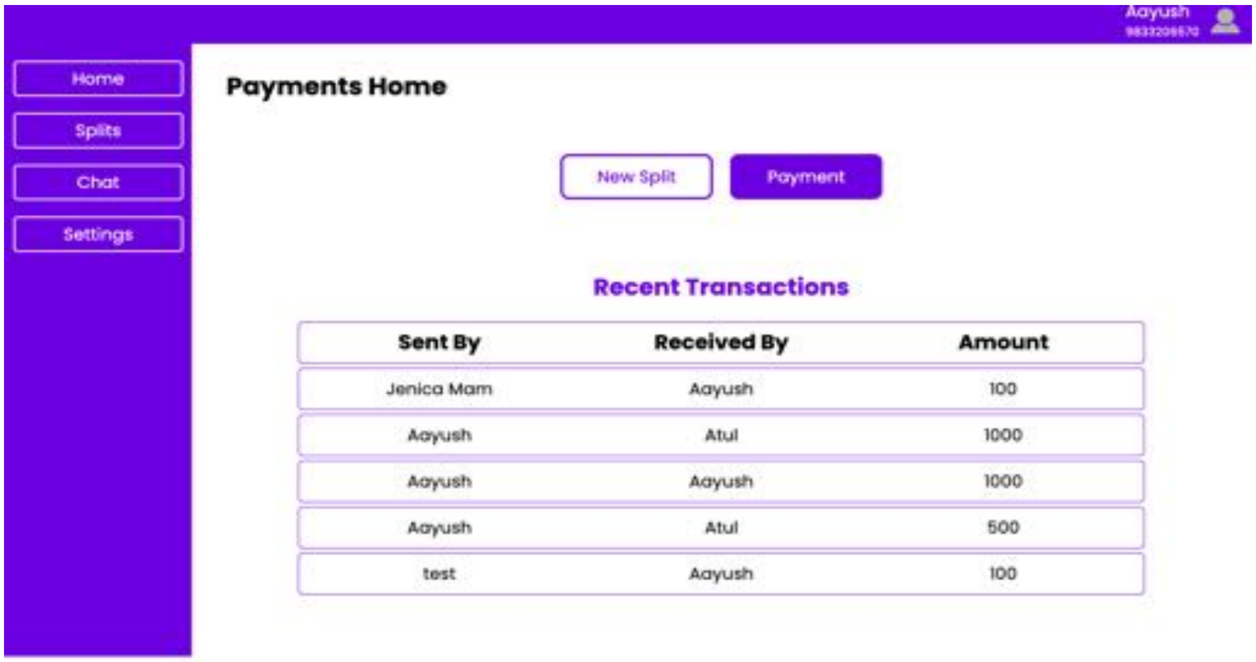
Enter Amount

1000

Enter Note

smartphone

PAYMENT PORTAL(FOR 1 to 1 TRANSACTION)

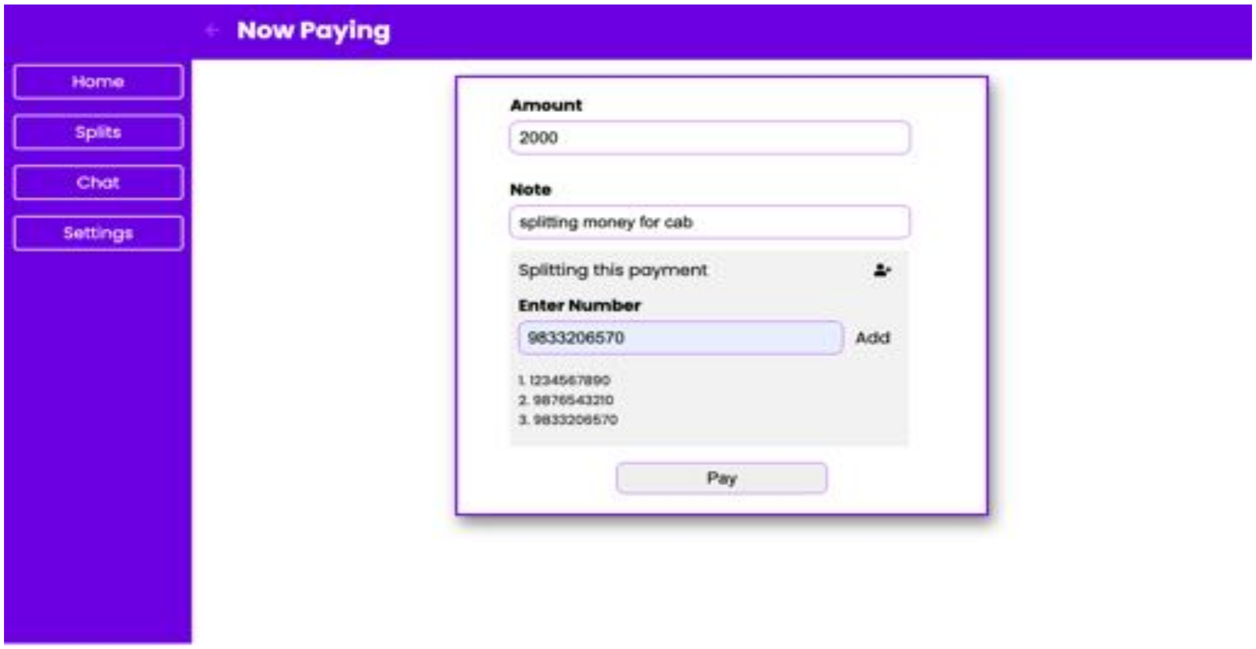


Payments Home

Recent Transactions

Sent By	Received By	Amount
Jenica Mam	Aayush	100
Aayush	Atul	1000
Aayush	Aayush	1000
Aayush	Atul	500
test	Aayush	100

PAYMENT PORTAL STATUS



Amount

2000

Note

splitting money for cab

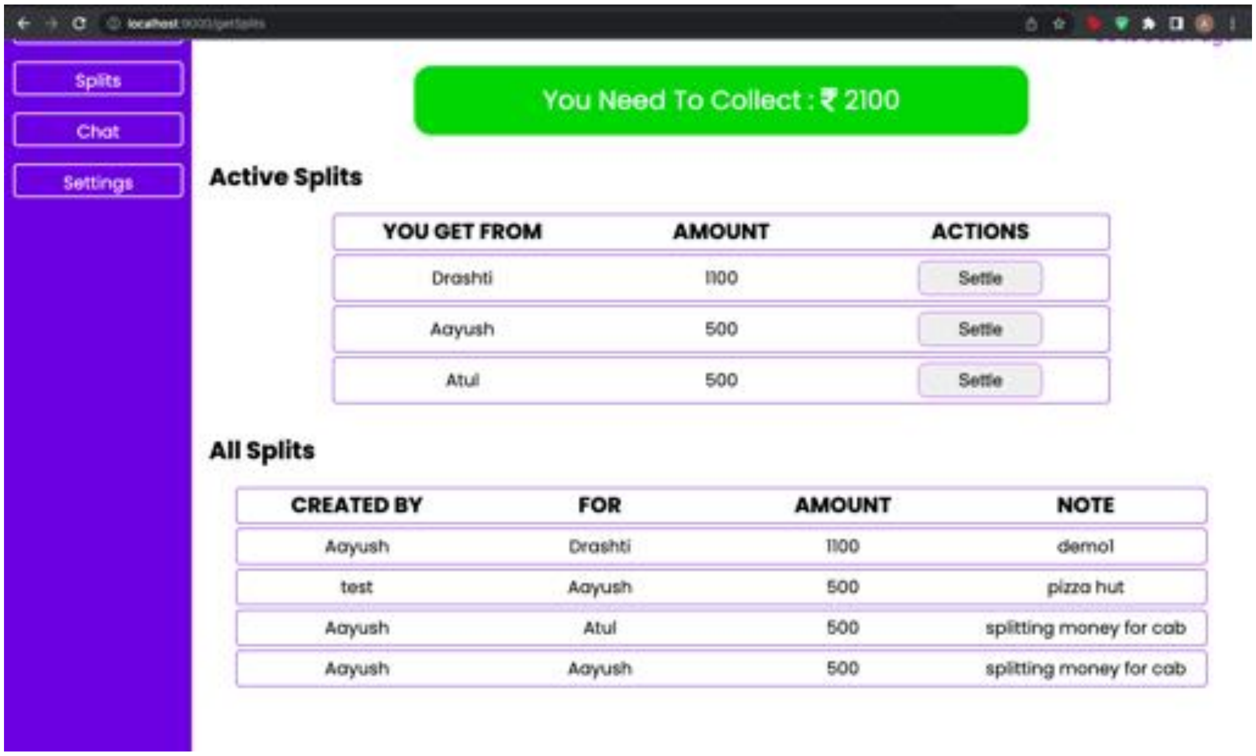
Splitting this payment

Enter Number

9833206570

1. 1234567890
2. 9876543210
3. 9833206570

SPLITS PORTAL



You Need To Collect : ₹ 2100

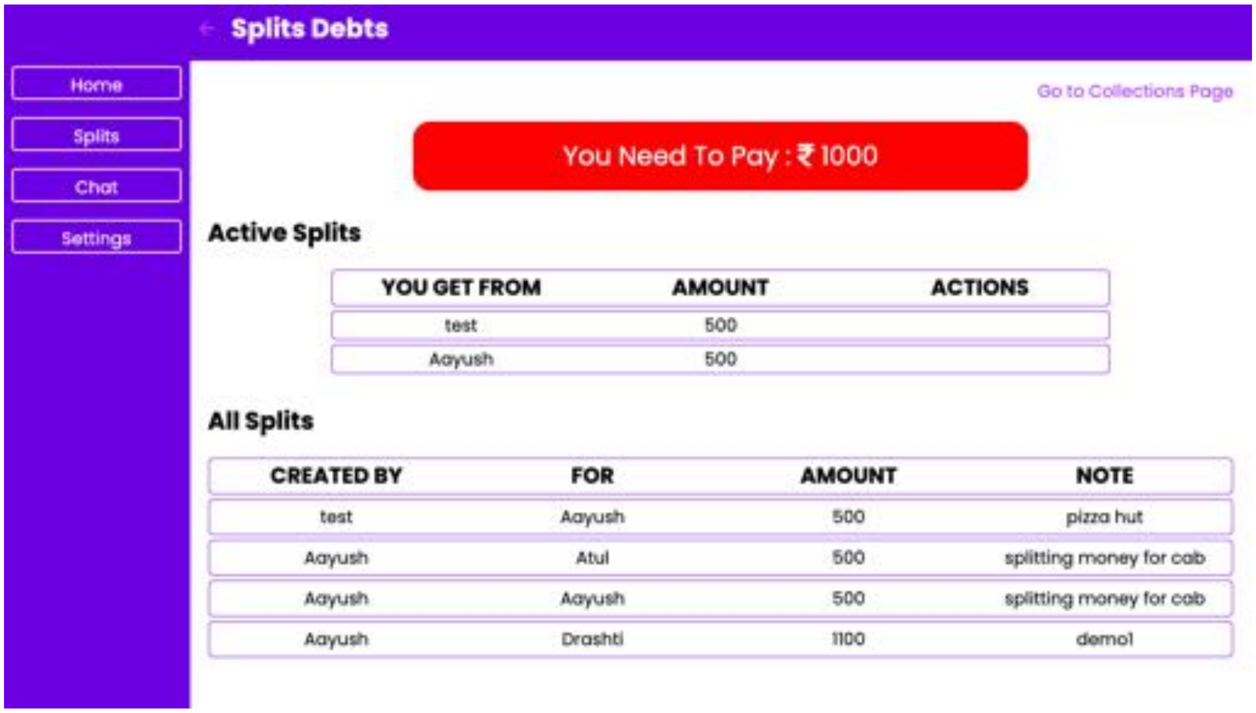
Active Splits

YOU GET FROM	AMOUNT	ACTIONS
Drashti	1100	Settle
Aayush	500	Settle
Atul	500	Settle

All Splits

CREATED BY	FOR	AMOUNT	NOTE
Aayush	Drashti	1100	demo1
test	Aayush	500	pizza hut
Aayush	Atul	500	splitting money for cab
Aayush	Aayush	500	splitting money for cab

VIEW SPLITS PORTAL



You Need To Pay : ₹ 1000

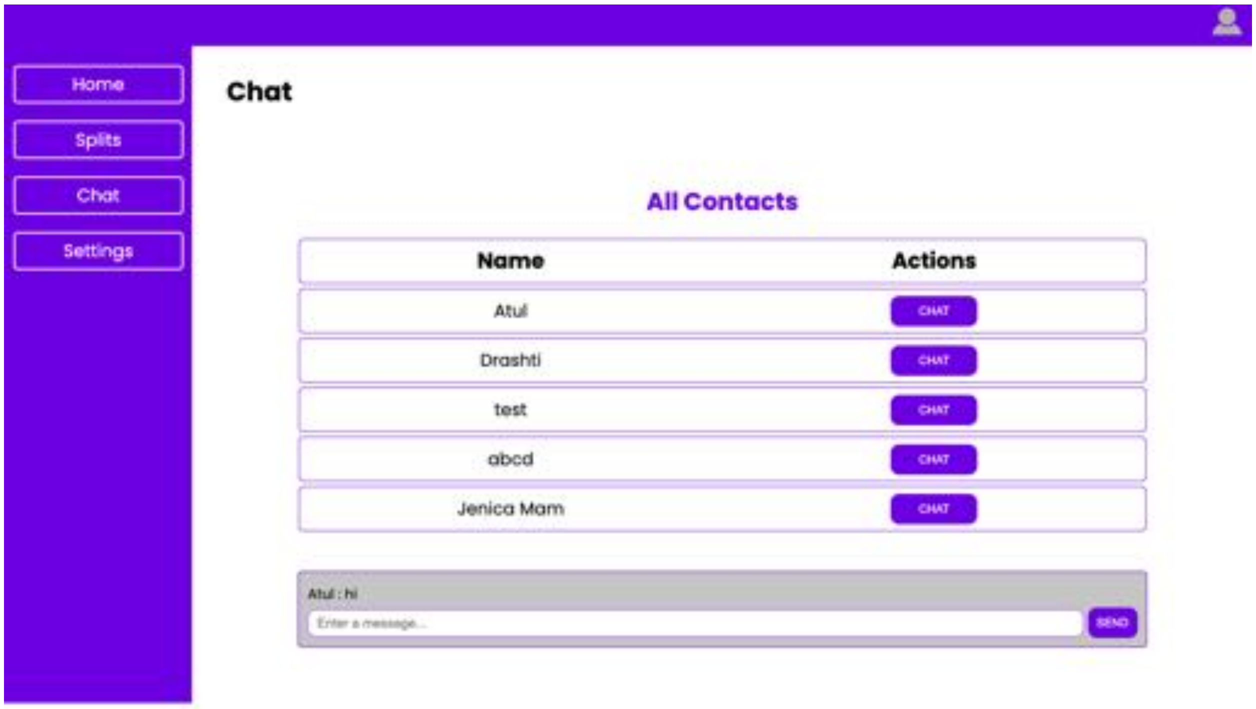
Active Splits

YOU GET FROM	AMOUNT	ACTIONS
test	500	
Aayush	500	

All Splits

CREATED BY	FOR	AMOUNT	NOTE
test	Aayush	500	pizza hut
Aayush	Atul	500	splitting money for cab
Aayush	Aayush	500	splitting money for cab
Aayush	Drashti	1100	demo1

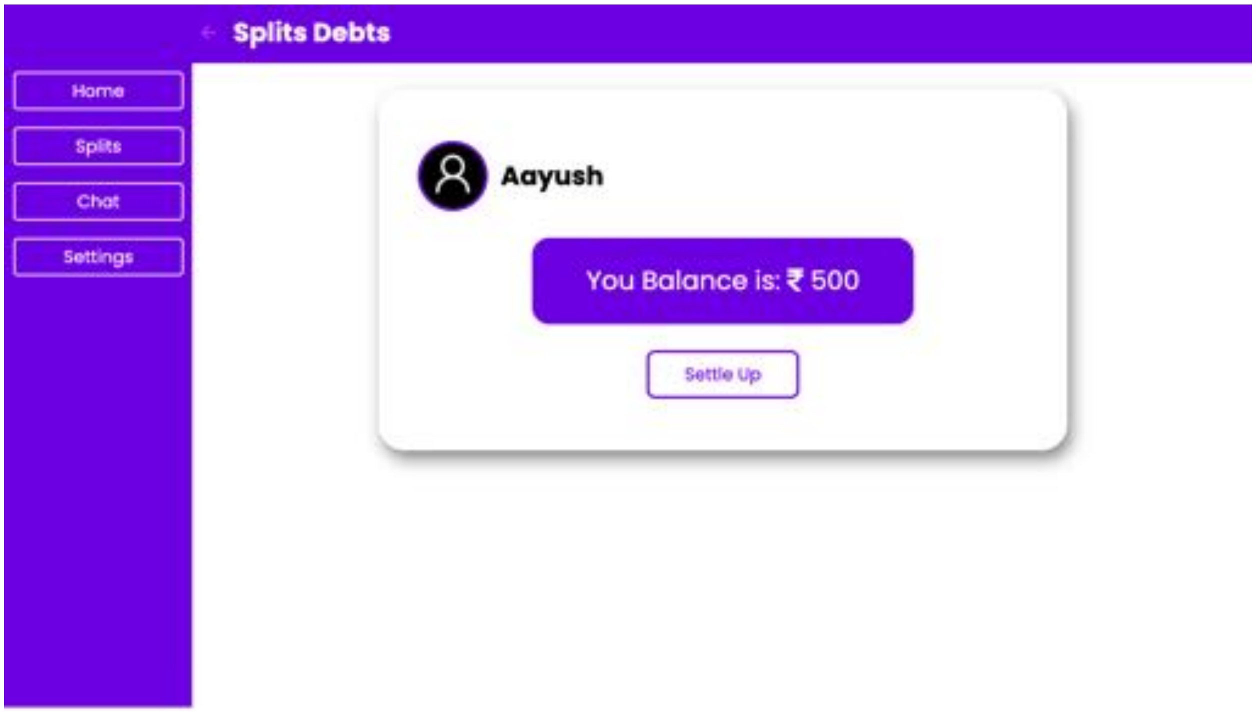
VIEW DEBT PORTAL



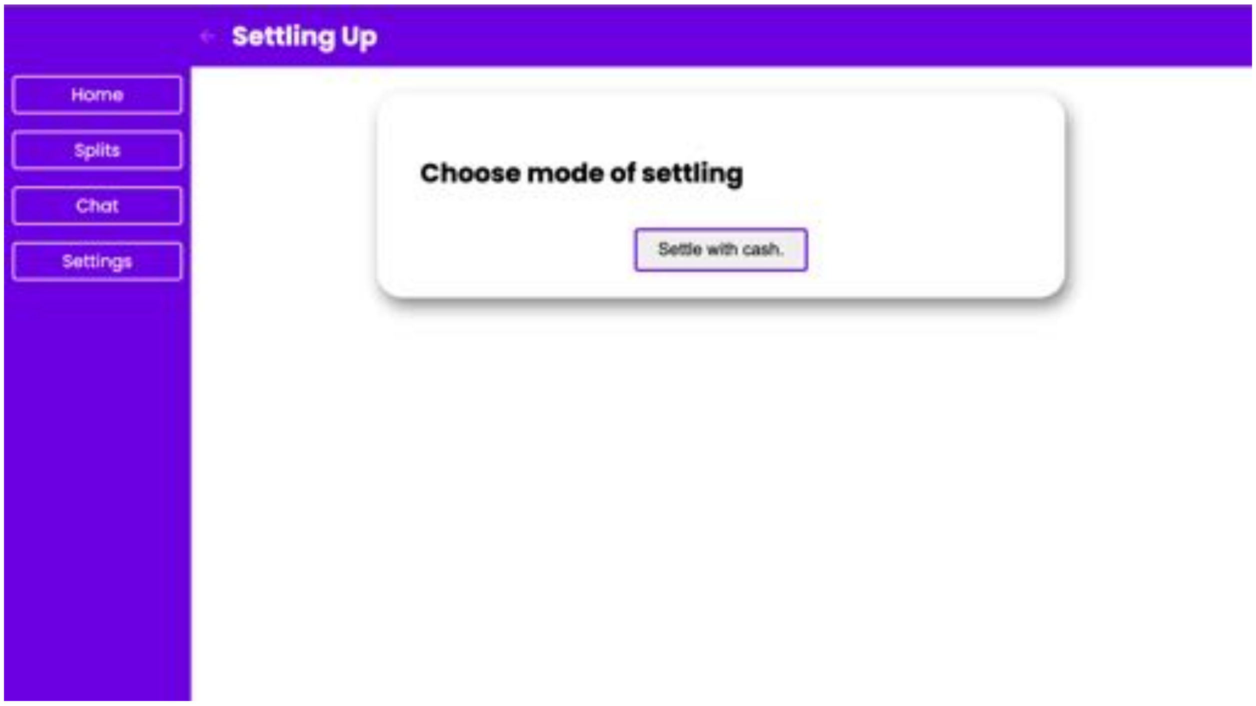
Name	Actions
Atul	CHAT
Drashti	CHAT
test	CHAT
abcd	CHAT
Jenica Mam	CHAT

Atul : hi

CHAT PORTAL

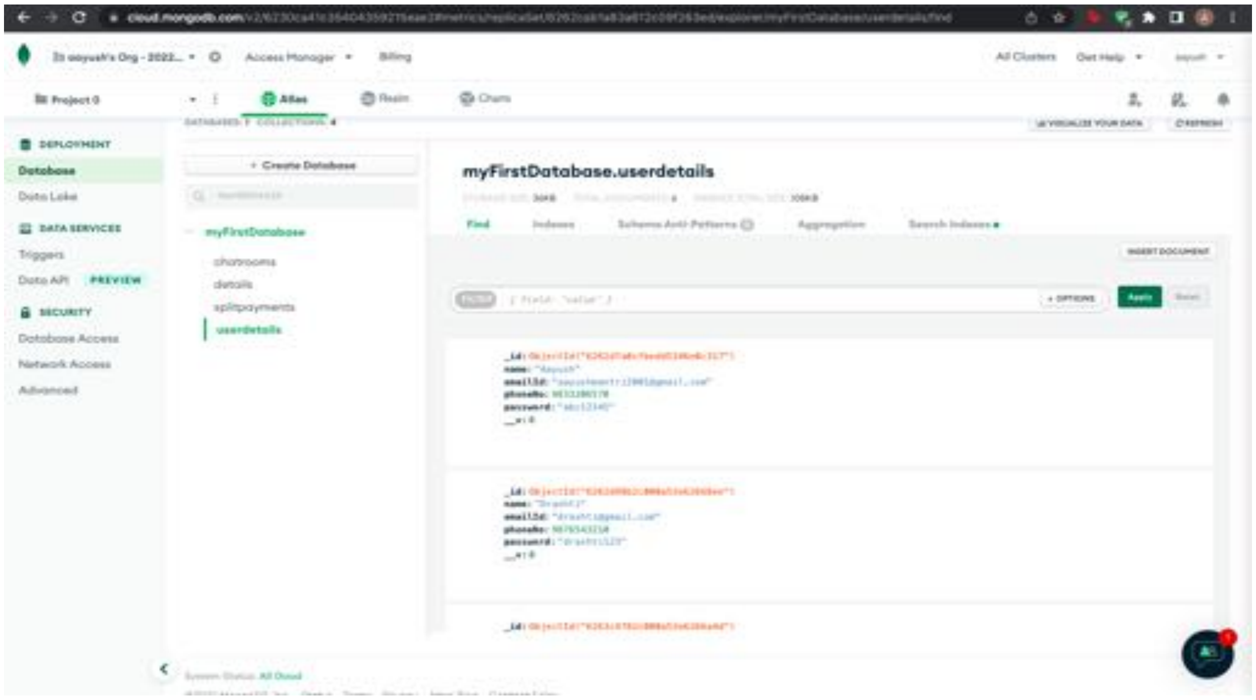


SETTLE UP PAGE



CHOOSING THE MODE TO SETTLE UP

5.2 DATABASE



myFirstDatabase.userdetails

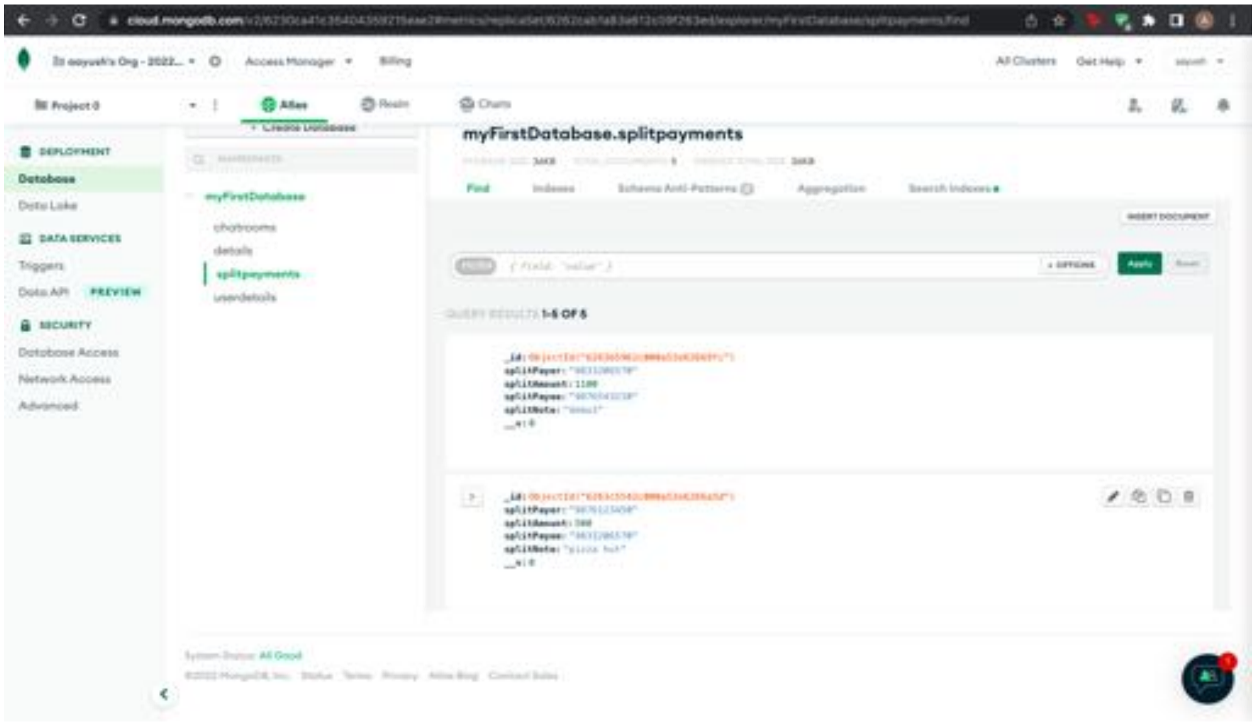
myFirstDatabase: chatrooms, details, splitpayments, userdetails

```
_id: ObjectId("[illegible]")
name: "[illegible]"
emailId: "[illegible]@gmail.com"
phoneNo: [illegible]
password: "[illegible]"
__v: 0

_id: ObjectId("[illegible]")
name: "[illegible]"
emailId: "[illegible]@gmail.com"
phoneNo: [illegible]
password: "[illegible]"
__v: 0

_id: ObjectId("[illegible]")
```

USER DETAILS



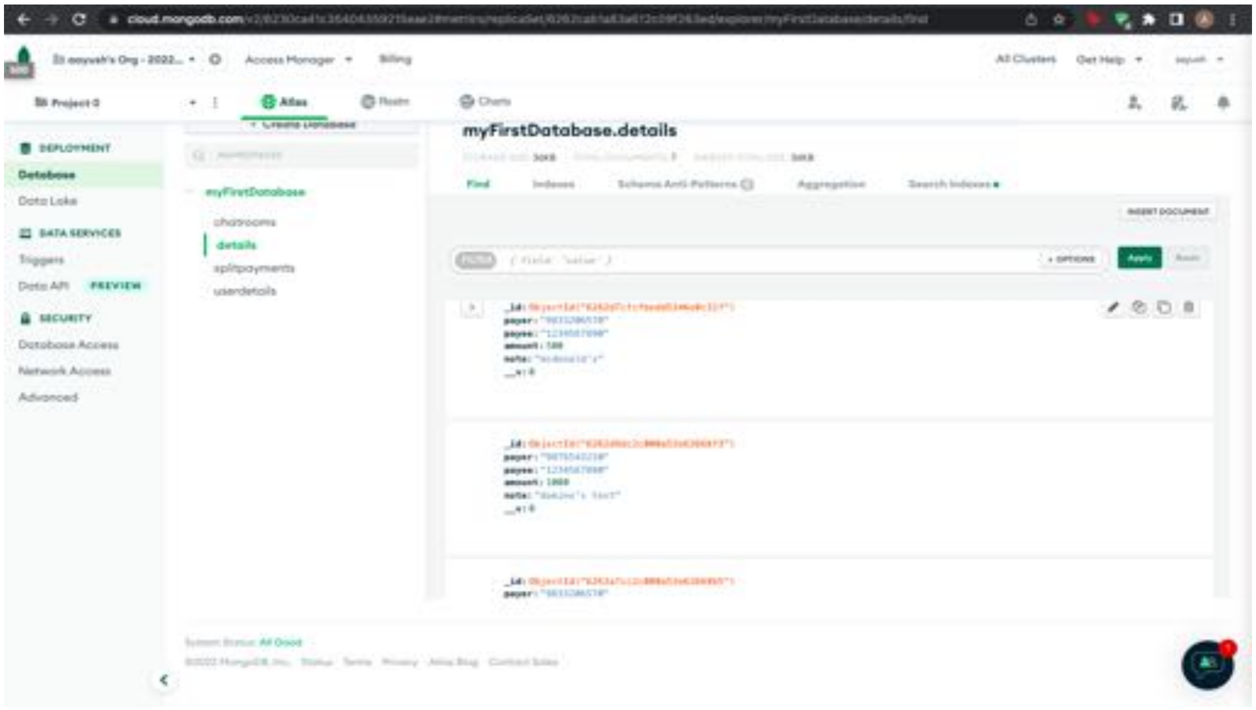
myFirstDatabase.splitpayments

QUERY RESULTS 1-5 OF 5

```
_id: ObjectId("[illegible]")
splitPayer: "[illegible]"
splitAmount: 1100
splitPayee: "[illegible]"
splitNote: "[illegible]"
__v: 0

_id: ObjectId("[illegible]")
splitPayer: "[illegible]"
splitAmount: 300
splitPayee: "[illegible]"
splitNote: "pizza hut"
__v: 0
```

SPLITPAYMENTS



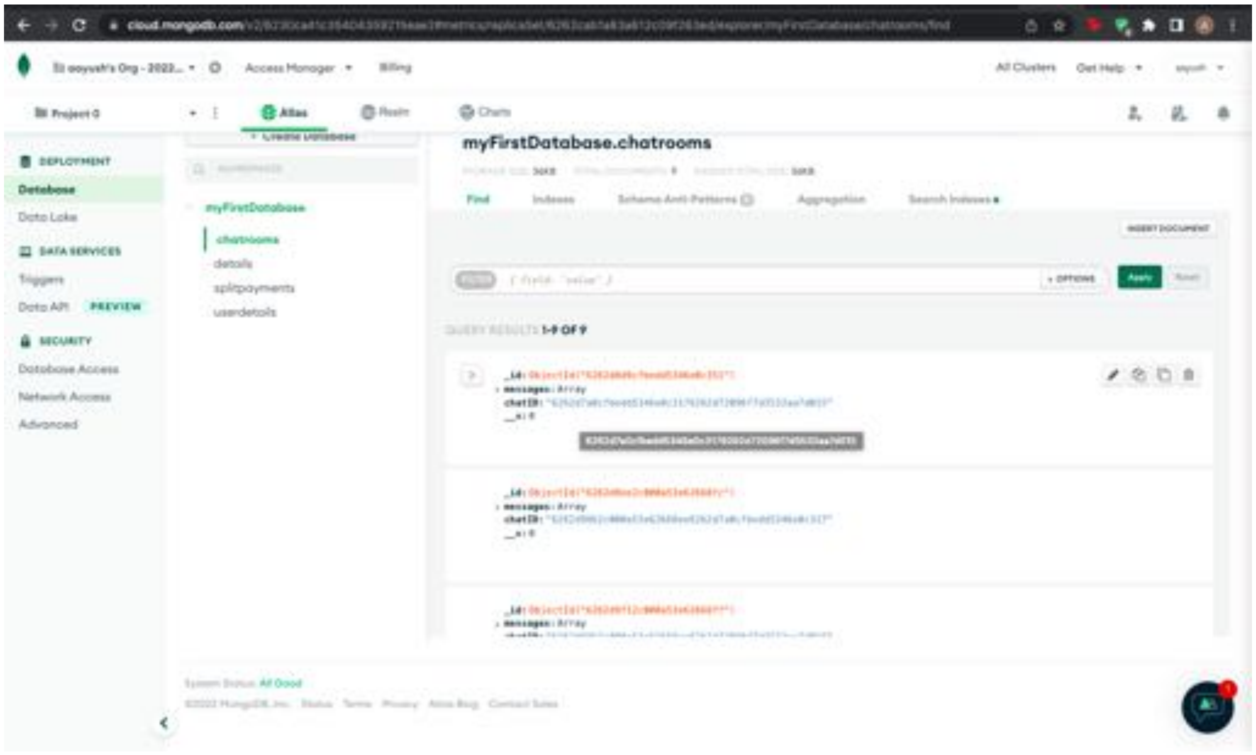
myFirstDatabase.details

```
_id: ObjectId("[illegible]")
payer: "[illegible]"
payee: "[illegible]"
amount: 500
note: "[illegible]"
__v: 0

_id: ObjectId("[illegible]")
payer: "[illegible]"
payee: "[illegible]"
amount: 1000
note: "[illegible]"
__v: 0

_id: ObjectId("[illegible]")
payer: "[illegible]"
```

DETAILS



myFirstDatabase.chatrooms

QUERY RESULTS 1-9 OF 9

```
_id: ObjectId("[illegible]")
> messages: Array
chatID: "[illegible]"
__v: 0

_id: ObjectId("[illegible]")
> messages: Array
chatID: "[illegible]"
__v: 0

_id: ObjectId("[illegible]")
> messages: Array
```

CHATROOMS

5.3 CODE

```
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");
const mongoose = require("mongoose");
const jwt = require("jsonwebtoken");
const bcrypt = require("bcryptjs");
const nodemailer = require("nodemailer");

const details = require("./server/models/paymentSchema");
const splitpayment = require("./server/models/splitSchema");
const chatMessages = require("./server/models/messageSchema");
const userCreate = require("./server/routes/user");
require("dotenv").config();

const uri =
`mongodb+srv://admin:aayush@cluster0.osaye.mongodb.net/myFirstDatabase?retryWrites=tru
e&w=majority`;

const app = express();
app.use(bodyParser.urlencoded({ extended: true, limit: "30mb" }));
app.use(bodyParser.json({ extended: true }));
app.use(cors());

app.use("/createUser", userCreate);

let accessToken;
let user;
const PORT = process.env.PORT || 9000;

mongoose
 .connect(uri, {
   useCreateIndex: true,
   useFindAndModify: false,
   useNewUrlParser: true,
   useUnifiedTopology: true,
 })
 .then(() => {
   console.log("Connection Completed");
   // res.json("Connection Completed");
 })
 .catch((err) => {
   console.log(err);
   // res.json(err);
 });

app.listen(PORT, () => {
 console.log("Server Started");
});

const conn = mongoose.connection;

conn.on("open", () => {
 console.log("Success");
});

app.use(express.json());
app.set("view engine", "ejs");
app.use(express.static("public"));
app.use("/css", express.static(__dirname + "public/css"));
app.use("/css", express.static(__dirname + "public/images"));

// //////////////////
// ROUTES
// //////////////////

app.get("/", (req, res) => {
 res.redirect("/login");
});

app.get("/chat", (req, res) => {
 res.sendFile(__dirname + "/views/chat.html");
});

app.get("/updatePay", (req, res) => {
 res.sendFile(__dirname + "/views/updatePay.html");
});

app.get("/login", (req, res) => {
 res.sendFile(__dirname + "/views/login.html");
```

```
});

app.get("/signup", (req, res) => {
 res.sendFile(__dirname + "/views/signup.html");
});

app.get("/makeUserPayment", (req, res) => {
 res.sendFile(__dirname + "/views/pay.html");
});

app.get("/deletePay", (req, res) => {
 res.sendFile(__dirname + "/views/deletePay.html");
});

app.get("/splits", async (req, res) => {
 res.sendFile(__dirname + "/views/newSplit.html");
});

app.get("/getSplits", async (req, res) => {
 res.sendFile(__dirname + "/views/splitCollect.html");
});

app.get("/splitDebt", async (req, res) => {
 res.sendFile(__dirname + "/views/splitDebt.html");
});

app.get("/settleDebt", async (req, res) => {
 res.sendFile(__dirname + "/views/settleUp.html");
});

app.get("/userSettings", async (req, res) => {
 res.sendFile(__dirname + "/views/userSettings.html");
});

// //////////////////
// AUTHENTICATION
// //////////////////

function authenticateToken(req, res, next) {
 const authHeader = req.headers["authorization"];
 const token = authHeader.split(" ")[1];
 jwt.verify(token, process.env.ACCESS_TOKEN_SECRET, (err, user) => {
   if (err) return res.json(err);
   req.user = user;
   next();
 });
}

// //////////////////
// CONTROLLERS
// //////////////////

app.post("/createToken", async (req, res) => {
 try {
   const checkAll = await conn
     .collection("userdetails")
     .findOne({ emailId: req.body.uEmail });
   if (bcrypt.compare(checkAll?.password, req.body.uPass)) {
     user = checkAll;
     accessToken = jwt.sign(user, process.env.ACCESS_TOKEN_SECRET);
     console.log(accessToken);
     console.log(user);
     res.redirect("/home");
     // console.log(accessToken)
     console.log("Login Success");
   } else {
     res.redirect("/signup");
   }
 } catch (err) {
   res.redirect("/signup");
   console.log(err);
 }
});

app.post("/updateDet", (req, res) => {
 // console.log(req.body)
 const makeUpdate = conn
   .collection("details")
   .updateOne(
     { payer: req.body.payerNum },
     {
       $set: { payee: req.body.updateVal },
     }
   )
   .then((results) => {
     res.redirect("/home");
   })
   .catch((error) => {
     console.log(error);
   });
 // console.log(makeUpdate)
});

app.post("/deleteDet", (req, res) => {
 // console.log(req.body)
 const makeUpdate = conn
   .collection("details")
   .deleteOne({ payer: req.body.payerDel })
   .then((result) => {
     // console.log(result)
     res.redirect("/home");
   })
   .catch((error) => {
     console.log(error);
   });
});

app.get("/home", (req, res) => {
 const cursor1 = conn
   .collection("details")
   .find()
   .toArray()
   .then((results) => {
     // console.log(results)
     res.sendFile(__dirname + "/views/home.html");
   })
   .catch((error) => console.log(error));
 // console.log(cursor1);
});

app.get("/getcontacts", authenticateToken, async (req, res) => {
 let userTrans = await conn.collection("userdetails").find().toArray();
 userTrans = userTrans.filter((u) => String(u._id) !== String(user._id));
 let updatedList = [];
 userTrans = userTrans.map((u) =>
   updatedList.push({
     name: u.name,
     phoneNo: u.phoneNo,
     chatID: u._id > user._id ? `${u._id}${user._id}` : `${user._id}${u._id}`,
   })
 );
 res.json(updatedList);
});

app.get("/getmessages/:id", authenticateToken, async (req, res) => {
 const { id } = req.params;
 let userTrans = await conn
   .collection("chatrooms")
   .find({ chatID: String(id) })
   .toArray();
 if (userTrans.length === 0) {
   let newRoom = new chatMessages({
     chatID: String(id),
     messages: [],
   });
   await newRoom.save();
 }
 res.json(userTrans);
});
app.get("/addmessage/:id/:message", authenticateToken, async (req, res) => {
 const { id, message } = req.params;
 let userTrans = await conn
   .collection("chatrooms")
   .find({ chatID: String(id) })
   .toArray();
 userTrans[0].messages.push({ senderName: user.name, message: message });
 const updated = await conn
   .collection("chatrooms")
   .updateOne({ _id: userTrans[0]._id }, { $set: userTrans[0] });

 res.json(updated);
});

// app.post("/pays", (req, res) => {
// console.log(req.body);
// });
```

```javascript
app.get("/fetchUserDetail", authenticateToken, async (req, res) => {
 res.json(user);
});

app.get("/fetchTransactions", authenticateToken, async (req, res) => {
 const userTrans = await conn
   .collection("details")
   .find({
     $or: [
       { payer: user.phoneNo.toString() },
       { payee: user.phoneNo.toString() },
     ],
   })
   .toArray();
 let paymentArray = [];
 userTrans.forEach(async (element) => {
   let payerName = await conn
     .collection("userdetails")
     .find({ phoneNo: Number(element.payer) })
     .toArray();
   let payeeName = await conn
     .collection("userdetails")
     .find({ phoneNo: Number(element.payee) })
     .toArray();
   let transObject = {
     giverName: payerName[0].name,
     getterName: payeeName[0].name,
     giveReason: element.note,
     giveAmt: element.amount,
     giveImg: payerName[0].userImage,
     getImg: payeeName[0].userImage,
   };
   paymentArray.push(transObject);
   if (paymentArray.length == userTrans.length) {
     res.json(paymentArray);
   }
 });
});

app.get("/fetchToken", async (req, res) => {
 res.json(accessToken);
});
```

```
app.post("/makeSplit", authenticateToken, async (req, res) => {
 const userLength = req.body.length;
 const amtDiv = req.body.amount / userLength;
 let splitUsers = req.body.payee;
 const splitReason = req.body.reason;
 let i = 0;
 for (i = 0; i < userLength - 1; i++) {
   const userFromDB = await conn
     .collection("userdetails")
     .find({ phoneNo: Number(splitUsers[i]) })
     .toArray();
   const userSplit = await conn
     .collection("splitpayments")
     .find({
       $and: [
         { splitPayer: user.phoneNo.toString() },
         { splitPayee: splitUsers[i].toString() },
       ],
     })
     .toArray();
   if (userSplit.length == 0) {
     let newSplit = new splitpayment({
       splitPayer: user.phoneNo,
       splitAmount: amtDiv,
       splitPayee: splitUsers[i],
       splitNote: splitReason,
     });
     await newSplit.save();
     try {
       (transporter = nodemailer.createTransport({
         service: "gmail",
         auth: {
           user: "splitpayiwp@gmail.com",
           pass: process.env.GMAIL_PASSWORD,
         },
       })),
         (mailOption = {
           from: "splitpayiwp@gmail.com",
           to: userFromDB[0].emailId,
           subject: "New Split Created",
           html: `A New Split Payment has been created!!<br /><br />You need to pay
${user.name}(${user.phoneNo}) a total of ${amtDiv}<br /><br
/><b>Note:</b>&nbsp;${splitReason}`,
         }),
         transporter.sendMail(mailOption, (err, data) => {
           console.log("Email Sent!");
         });
     } catch (error) {
       console.log(error);
     }
   } else {
     let paymentBalance = Number(userSplit[0].splitAmount) + Number(amtDiv);
     const splitUpdate = await conn
       .collection("splitpayments")
       .findOneAndUpdate(
         {
           $and: [
             { splitPayer: user.phoneNo.toString() },
             { splitPayee: splitUsers[i].toString() },
           ],
         },
         { $set: { splitAmount: Number(paymentBalance) } }
       );
   }
 }
 res.json("User Added Successfully");
 // console.log(splitUsers)
});

app.get("/showCollect", authenticateToken, async (req, res) => {
 const splitCollect = await conn
   .collection("splitpayments")
   .find({ splitPayer: user.phoneNo.toString() })
   .toArray();
 let splitArray = [];
 splitCollect.forEach(async (element) => {
   let splitCollectName = await conn
     .collection("userdetails")
     .find({ phoneNo: Number(element.splitPayee) })
     .toArray();
   let splitObject = {
     name1: splitCollectName[0].name,
     img1: splitCollectName[0].userImage,
     amount1: element.splitAmount,
     reason1: element.splitNote,
     number1: splitCollectName[0].phoneNo,
   };
   splitArray.push(splitObject);
   if (splitArray.length == splitCollect.length) {
     res.json(splitArray);
   }
 });
 // res.json(splitCollect)
});

app.get("/showDebt", authenticateToken, async (req, res) => {
 const splitDebt = await conn
   .collection("splitpayments")
   .find({ splitPayee: user.phoneNo.toString() })
   .toArray();
 let debtArray = [];
 splitDebt.forEach(async (element) => {
   let splitDebtName = await conn
     .collection("userdetails")
     .find({ phoneNo: Number(element.splitPayer) })
     .toArray();
   let DebtObject = {
     name2: splitDebtName[0].name,
     amount2: element.splitAmount,
     reason2: element.splitNote,
     number2: splitDebtName[0].phoneNo,
     img2: splitDebtName[0].userImage,
   };
   debtArray.push(DebtObject);
   if (debtArray.length == splitDebt.length) {
     res.json(debtArray);
   }
 });
});

app.get("/allSplits", authenticateToken, async (req, res) => {
 const splitAll = await conn
   .collection("splitpayments")
   .find({
```

```
    $or: [
      { splitPayee: user.phoneNo.toString() },
      { splitPayer: user.phoneNo.toString() },
    ],
  })
  .toArray();
 let allArray = [];
 splitAll.forEach(async (element) => {
  let splitName = await conn
    .collection("userdetails")
    .find({ phoneNo: Number(element.splitPayer) })
    .toArray();
  let debtName = await conn
    .collection("userdetails")
    .find({ phoneNo: Number(element.splitPayee) })
    .toArray();
  let allObject = {
    giverName: splitName[0].name,
    getterName: debtName[0].name,
    giveReason: element.splitNote,
    giveAmt: element.splitAmount,
    giverImg: splitName[0].userImage,
    getterImg: debtName[0].userImage,
  };
  allArray.push(allObject);
  if (allArray.length == splitAll.length) {
    res.json(allArray);
  }
 });
});

app.get("/settleUser/:id", authenticateToken, async (req, res) => {
 // console.log(req.params['id'])
 const splitBetween = await conn
  .collection("splitpayments")
  .find({
    $and: [
      { splitPayer: user.phoneNo.toString() },
      { splitPayee: req.params["id"].toString() },
    ],
  })
  .toArray();
 const userSplit = await conn
   .collection("userdetails")
   .find({ phoneNo: Number(req.params["id"]) })
   .toArray();

 const userArray = [];
 userArray.push(splitBetween);
 userArray.push(userSplit);
 res.json(userArray);
});

app.get("/settleOption", async (req, res) => {
 res.sendFile(__dirname + "/views/settleOptions.html");
});

app.get("/settlePayment/:id", authenticateToken, async (req, res) => {
 const settleCollect = await conn
   .collection("splitpayments")
   .deleteOne({
     $and: [
       { splitPayer: user.phoneNo.toString() },
       { splitPayee: req.params["id"].toString() },
     ],
   })
   .then((ans) => {
     res.sendFile(__dirname + "/views/paymentComplete.html");
   })
   .catch((err) => {
     console.log(err);
   });
});

app.post("/payment", authenticateToken, async (req, res) => {
 const makePayment = new details({
   payer: user.phoneNo,
   payee: req.body.payee,
   amount: req.body.amount,
   note: req.body.note,
 });
 const userFromDB = await conn
   .collection("userdetails")
   .find({ phoneNo: Number(req.body.payee) })
   .toArray();
 try {
   const makeNew = await makePayment.save();
   try {
     (transporter = nodemailer.createTransport({
       service: "gmail",
       auth: {
         user: "splitpayiwp@gmail.com",
         pass: process.env.PASS,
       },
     })),
       (mailOption = {
         from: "splitpayiwp@gmail.com",
         to: userFromDB[0].emailId,
         subject: "New Split Created",
         html: `${user.name} just payed you <b>Rs.${req.body.amount}<b/><br /><br
/><b>Note:</b>&nbsp;${req.body.note}`,
       }),
       transporter.sendMail(mailOption, (err, data) => {
         console.log("Email Sent!");
       });
   } catch (error) {
     console.log(error);
   }
   res.redirect("/");
   // res.json(makeNew)
 } catch (err) {
   res.send(err);
 }
});
```

CHAPTER VI:
TEN HEURISTIC EVALUATION

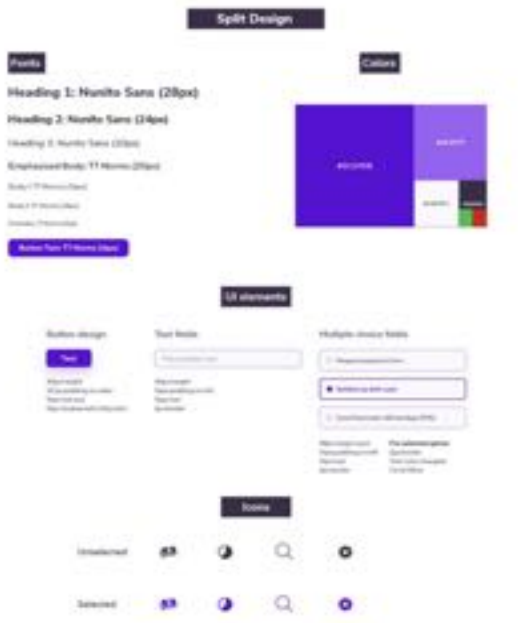
10 Heuristic evaluation & Shneiderman’s 8 Golden Rules matching with UI

Evaluation of SplitPay using Schneiderman’s 8 Golden Rules

All examples have been marked with a blue box

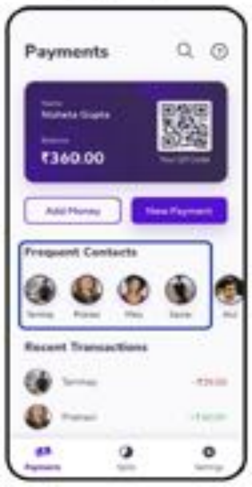
Rule 1: Strive for consistency

To maintain consistency in our project, we came up with a design system called Split Design. Split Design is a collection of fonts, colors, icons and interface elements that help us ensure that our app design is consistent throughout. You can view the various elements of the system as shown:
Furthermore, you can view in the screens that very important buttons are always placed at the bottom, to go with Hick’s Law. Also, a 8pt grid maintains visual consistency.



Rule 2: Cater to Universal Usability (Enable frequent users to usse shortcuts)

Our system aims to makes the lives of its frequent users by giving them the option to save information like credit card information, and by giving them a list of their frequently interacted contacts.

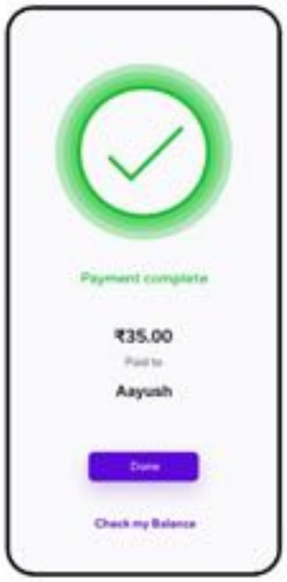


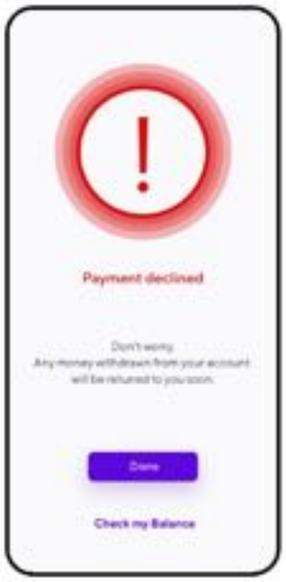
As you can see here, we have provided a list of frequent contacts, with whom the user frequently does transactions/bill splitting with. This helps the user quickly select these contacts and start payments in one click.

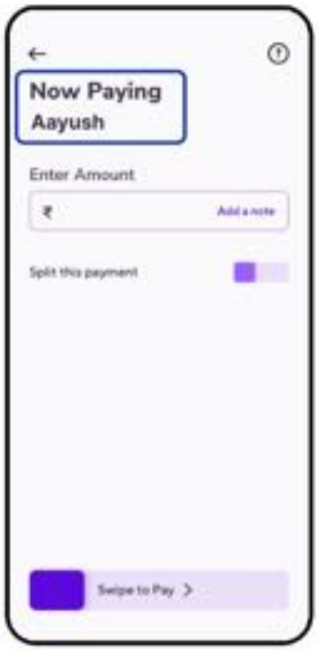
Rule 3: Offer informative feedback

Feedback is extremely important to inform the user the state of the system and in this case, the state of a transaction. That's why ever screen has a big heading telling the current state.

Informative confirmation and alerts on completion or decline of a payment respectively



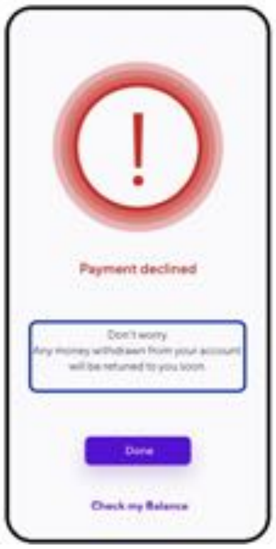

Payment declined
Don't worry.
Any money withdrawn from your account will be returned to you soon.
Done
Check my Balance



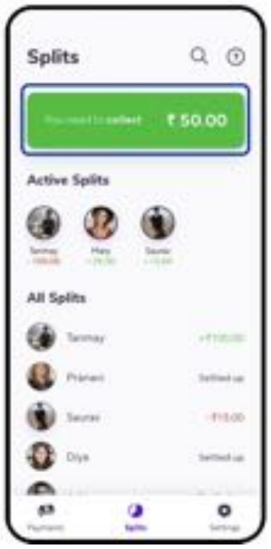
By giving the name of the contact below "Now Paying" (in this case Tanmay), we alert the user of the contact they have selected for payment.

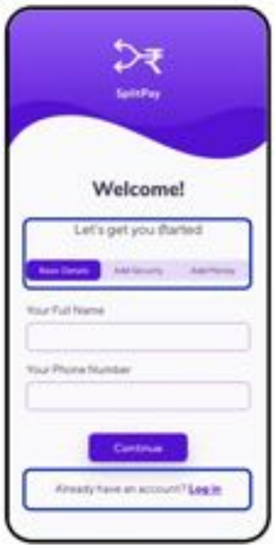
Rule 4: Design dialogs to yield closure

The language used in the copy text is very simple to understand. It was noted during data collection that our target user would know conversational English and thus the dialogs in the system will be very easy for them to understand. Some examples are:


Payment declined

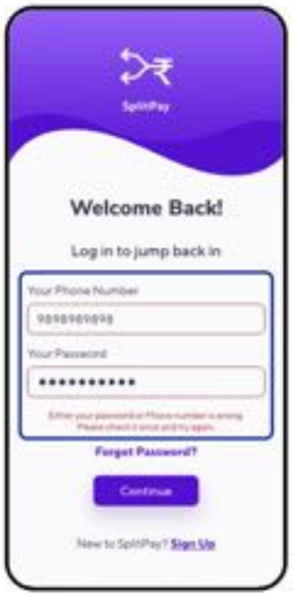
Don't worry
Any money withdrawn from your account will be returned to you soon.





Rule 5: Offer error prevention and simple error handling

In the log-in form, while taking password, the user will be prompted if any details are invalid as shown below:


Welcome Back!

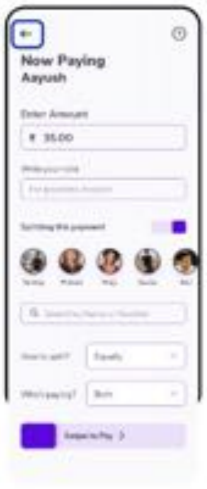
Log in to jump back in

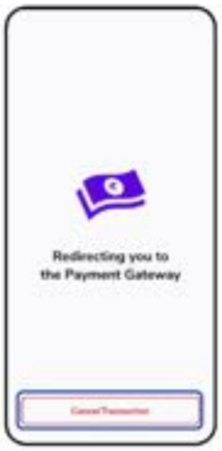
Your Phone Number: 9898989898

Your Password: ●●●●●●●●●●●

Either your password or Phone number is wrong. Please check it once and try again.

Rule 6: Permit easy reservsal of actions



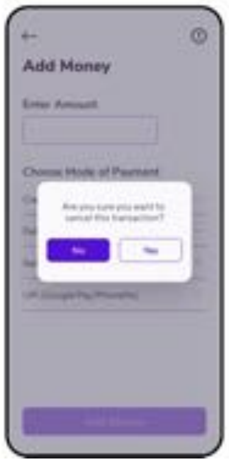


As can be seen on all screens of the app, there is always a "Go back" arrow button on the top left corner, to allow the user to go back to the previous page or module.

The only place where we haven't put a "Swipe to..." functionality is in the Add money page. This is because, after initiating the transaction, the app will give the user a brief time period to cancel the transaction as shown below

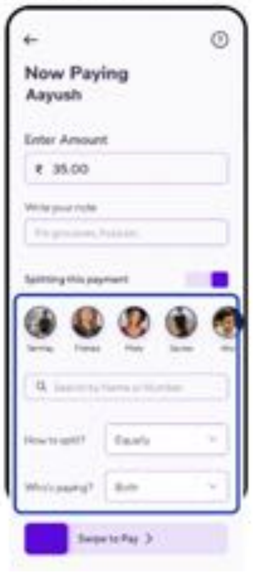
Rule 7: Support internal locus of control

Letting the user feel in-charge of the entire flow is very important. Therefore, there are certain places and hints where the user is asked for permission to do tasks.
One such example is the Swipe to Pay mechanism. It reassures the user that the payment is only initiated on swiping the bar. The mechanism is shown above.
Another example is when you try to cancel a transaction, you are asked for confirmation. This lets the user feel like they are in control of whether the transaction will go through or not. An example is shown below:

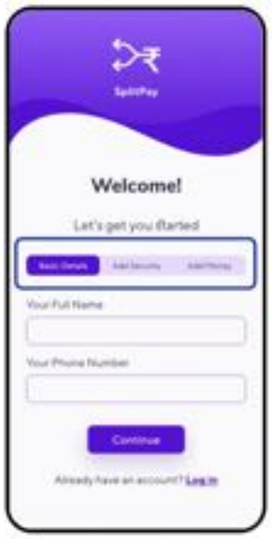


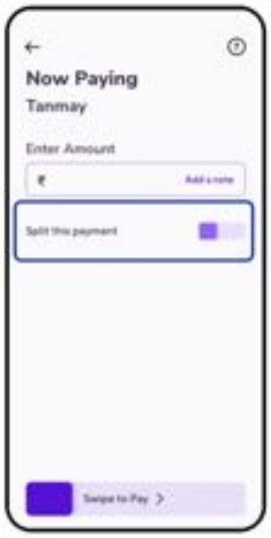
Rule 7: Support internal locus of control

Lastly, a lot of customisability is provided to the user when it comes to splitting expense. For example, you can select who is paying the bill currently and how it is split. This lets the users decide the way the bill is split.



Rule 8: Reduce short-term memory load





This has been done on the Sign up screens. The entire process is divided into 3 segments so that the user can focus on completing just one task at a time.

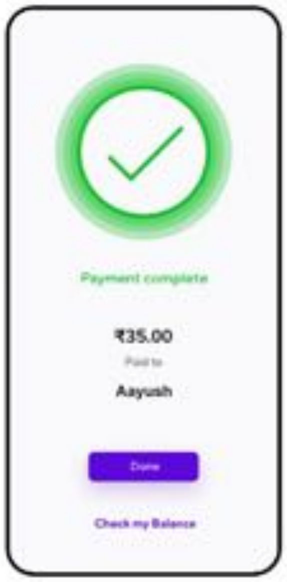
Moreover, the expense splitting options are hidden at first, so as to not confuse the user with too many choices. These can be toggled on by clicking on the split this payment button.

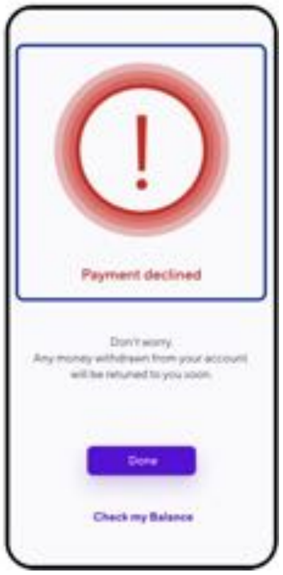
Evaluation of SplitPay using 10 Usability Heuristics

All examples have been marked with a blue box

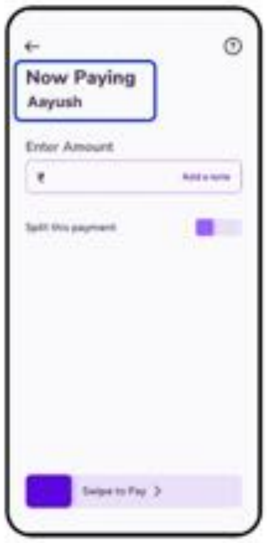
#1: Visibility of system status

The design should always keep users informed about what is going on, through appropriate feedback within a reasonable amount of time. When users know the current system status, they learn the outcome of their prior interactions and determine next steps. Predictable interactions create trust in the product as well as the brand.

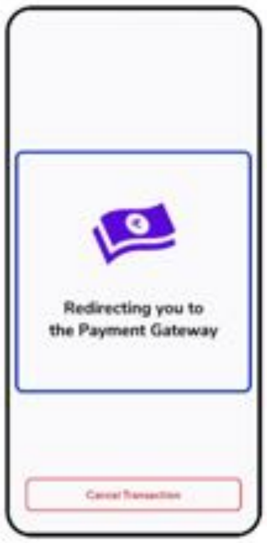



Payment declined
Don't worry.
Any money withdrawn from your account will be returned to you soon.
Done
Check my Balance

Informative confirmation and alerts on completion or decline of a payment respectively



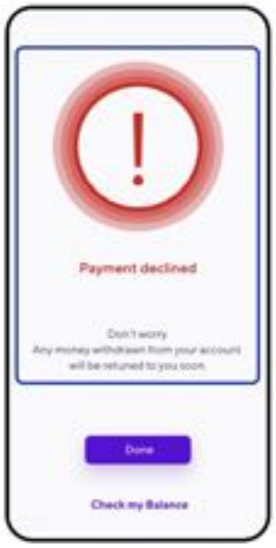
By giving the name of the contact below "Now Paying" (in this case Tanmay), we alert the user of the contact they have selected for payment.

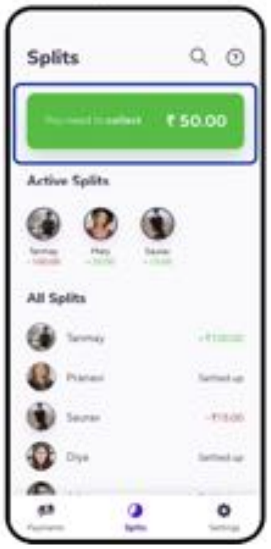


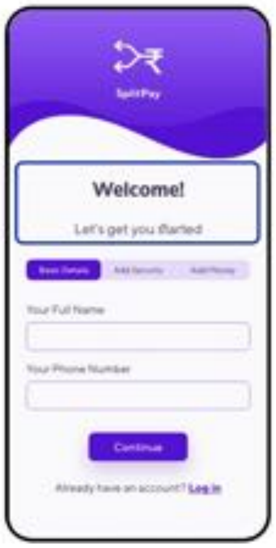
Giving the user proper feedback and showing them what is going on in, and why the app is taking time to "load" the confirmation page

#2: Match between system and the real world

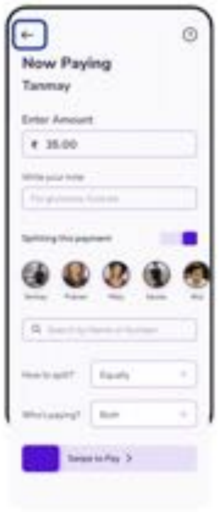
The language used in the copy text is very simple to understand. It was noted during data collection that our target user would know conversational English and thus the dialogs in the system will be very easy for them to understand. Some examples are:

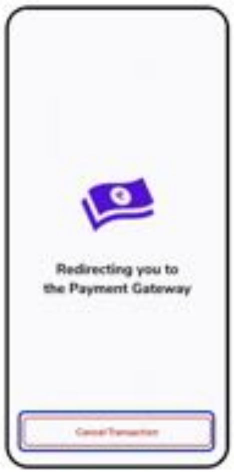

Payment declined
Don't worry
Any money withdrawn from your account will be returned to you soon.
Done
Check my Balance





#3: User control and freedom

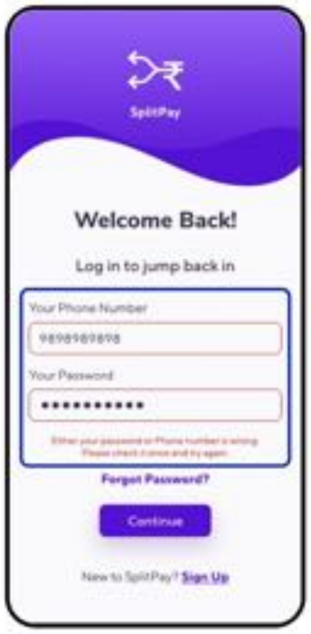


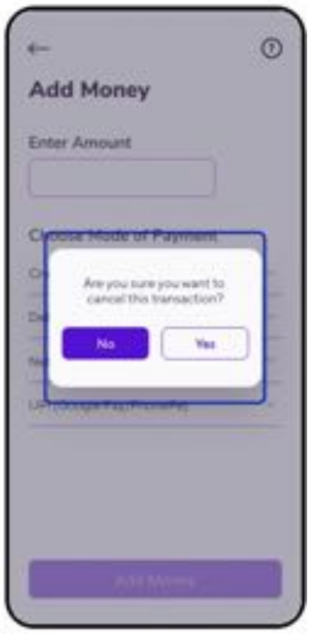


As can be seen on all screens of the app, there is always a "Go back" arrow button on the top left corner, to allow the user to go back to the previous page or module.

The only place where we haven't put a "Swipe to..." functionality is in the Add money page. This is because, after initiating the transaction, the app will give the user a brief time period to cancel the transaction as shown below

#5: Error prevention

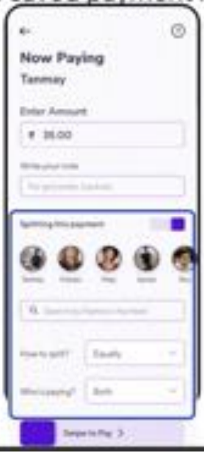


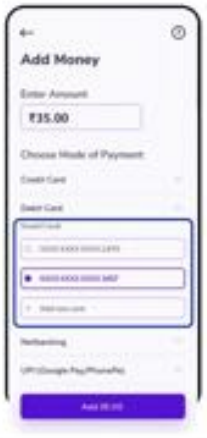


In the log-in form, while taking password, the user will be prompted if any details are invalid as shown

#6: Recognition rather than recall

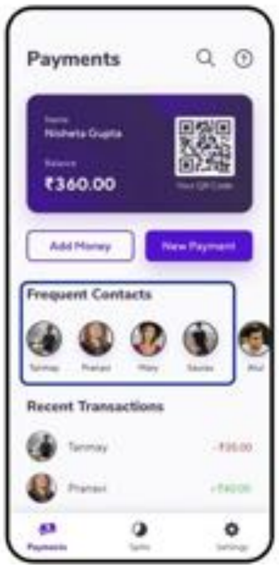
The objective here will be to reduce the memory load the user by making all their options for a task visible. In the Payment Module, while determining the splitting parties, the user can choose to type each name (RECALL) or select from a scrollable list of all available users within the group (RECOGNIZE).
Also, within the Payment Module, the user can choose to add the details for their payment method or choose from a list of saved payment methods.



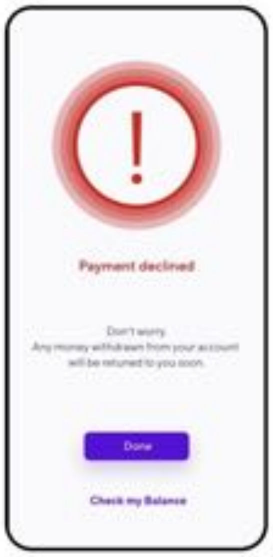


#7: Flexibility and efficiency of use

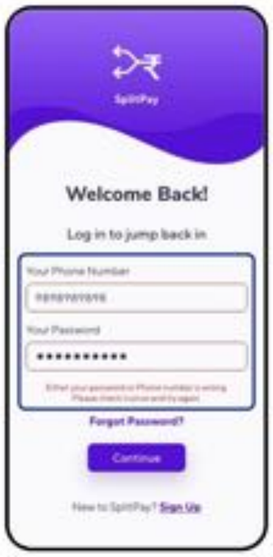
An active user can view their frequent contacts. This will make the app more efficient for them and also be easier and faster to use.



#8: Help users recognize, diagnose, and recover from errors


Payment declined
Don't worry
Any money withdrawn from your account will be returned to you soon.
Done
Check my Balance

Informative alerts on decline of a payment



In the log-in form, while taking password, the user will be prompted if any details are invalid as shown

#10: Aesthetic And Minimalististic Design

A minimalism is not only a fashion of last few years, but it certainly is a lasting trend with the aim to reduce the description of a subject just to its necessary elements. It has many applications in art, music, and literature. Minimalism helps users to quickly access important information and come to the result quickly.

CHAPTER VII:
TESTING

7.1 Unit Testing

Registration/Sign Up Page:

1.Testing Name

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Entering name with a digit	aayush123	Proceed to successfully register on the site.	Registration failed, as no digits allowed in name	FAIL
b	Entering valid name	aayush	Proceed to successfully register on the site.	Proceed to successfully register on the site.	PASS

2.Testing Phone Number

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Entering more/less than 10 digits	88797285989 887972859	Proceed to successfully register on the site.	Registration failed, only 10 digit number allowed	FAIL
b	Entering valid name	8879728598	Proceed to successfully register on the site.	Proceed to successfully register on the site.	PASS

3.Testing Email:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Entering invalid email format	aayush.com	Proceed to successfully register on the site.	Registration failed, only valid email formats allowed	FAIL
b	Entering valid name	aayush@gmail.com	Proceed to successfully register on the site.	Proceed to successfully register on the site.	PASS

4. Entering Password

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Entering password with no digits	aayushagarwal	Proceed to successfully register on the site.	Registration failed, password needs digits	FAIL
b	Entering password of length less than 8	aayush123	Proceed to successfully register on the site.	Registration failed, password needs at least 8 characters	FAIL
c	Entering Valid Password	aayushagarwal123	Proceed to successfully register on the site.	Proceed to successfully register on the site.	PASS

5.Confirm Password Test:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Entering a password that does not match	Password: aayushagarwal123 Confirm Password: aayushagarwal	Proceed to successfully register on the site.	Registration failed, fields must match	FAIL
b	Entering matching Passwords	Password: aayushagarwal123 Confirm Password: aayushagarwal123	Proceed to successfully register on the site.	Proceed to successfully register on the site.	PASS

6.Adding an Image

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Uploading a pdf/ ppt/ any other file format	Da.doc/da.pdf	Proceed to successfully register on the site.	Registration failed, only image type files allowed	FAIL

7. Testing Repetitive Entry:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Entering an email/phone number that exists already	aayush@gmail.com or8879728598	Proceed to successfully register on the site.	Registration failed, user already exists	FAIL
b	Entering completely new values	agarwal@gmail.com	Proceed to successfully register on the site.	Proceed to successfully register on the site.	PASS

Testing Login Page:

8.Entering Credentials:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Entering an invalid/ unrecognized phone number	88797285989/ 1234657890	Proceed to successfully log on the site.	Login failed, invalid format or not recognized error shown	FAIL
b	Entering valid and identified phone numbers	8879728598	Proceed to successfully log on the site.	Proceed to successfully log on the site.	PASS

9.Testing Password Credential:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Entering an invalid/ unrecognized password	Password: aayushagarwal123 Confirm Password: aayushagarwal12	Proceed to successfully Login the site.	Login failed, password incorrect must match	FAIL
b	Entering correct Passwords	Password: aayushagarwal123 Confirm Password: aayushagarwal123	Proceed to successfully Login on the site.	Proceed to successfully Login on the site.	PASS

10. Add new Split:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Entering a negative number as Amount	-200	Proceed to add split to the ledger	Failed, only positive amounts can be splitted	FAIL
b	Entering 0 as split amount	0	Proceed to add split to the ledger	Failed, as amount must be greated than 0	FAIL
c	Entering a positive non zero amount	200	Proceed to add split to the ledger	Proceed to add split to the ledger	PASS

11.Adding Note:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Leaving the note empty	“”	Proceed to add split to the ledger	Failed, note must have a value.	FAIL
b	Entering a note	“Pizza”	Proceed to add split to the ledger	Proceed to add split to the ledger	PASS

12. Adding Numbers to Split Payment with:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Adding invalid number format	88797285989	Proceed to add split to the ledger	Failed, phone number must be 10 digit long only.	FAIL
b	Adding a number that does not exist on platform	1234568790	Proceed to add split to the ledger	Failed, number not recognized	FAIL
c	Adding own number	8879728598	Proceed to add split to the ledger	Failed, number cannot be your own	FAIL
d	Adding valid numbers	8879728599	Proceed to add split to the ledger	Proceed to add split to the ledger	PASS

13. Trying to Pay:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Trying to add payment without any number	“”	Proceed to add split to the ledger	Failed, need to click on add number button first.	FAIL
b	Adding payment after clicking add number button	8879728598	Proceed to add split to the ledger	Proceed to add split to the ledger	PASS

14. View Recent/ All transactions on the main page:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Failure to fetch from the API	Error on client machine/ server status	Proceed to view transactions	Failed nothing fetched.	FAIL
b	Successful fetch	Correct fetch link with key	Proceed to view transactions	Proceed to view transactions	PASS

15. View Settle Page:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Make Transition to Home/Setting Page	Href access to user respective page	Proceed to respective Page	Page Loads	PASS
b	Choose to settle active transaction	Choose correct settle contact page	Proceed to view history	Specific transaction page loads	PASS

16. Settle Payments:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	View history with specific user	Select correct user detail.	Proceed to View correct history	All user history in table format displayed	PASS
b	Make a payment to the user, to settle payment.	Choose settle option	Proceed to payment page	Page correctly identifies both parties	PASS

17.Settle Options:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Settle With Cash	User balance both users and ID.	Book settled	Balance between both users settled.	PASS
b	Settle with UPI/Online Payment	User online card details.	Book Settled	Works only in testing phase, cannot make transaction (key not acquired)	FAIL

18.QR Code Development:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Make user specific QR code.	User phone number on weak network.	Generate, user QR code.	QR Code generated after long delay.	PASS
b	Make efficient QR code.	User phone number on weak network.	Generate, user QR code.	QR code generated in time.	PASS

19. QR Code scan:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Scan QR Code using UPI app.	Generated QR Code.	User payment details.	Key required, cannot generate at this point.	FAIL
b	Scan QR Code using QR code scanner.	Generated QR Code.	User payment details.	User payment details.	PASS

20. User Help:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Request Help Document	Request HREF access.	User help documentation page.	User help documentation page.	PASS

21.User Navigation

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Click on Navigate Link	Request HREF access.	Specific page loaded	Respective page access given	PASS

22.User Switch Between Sign Up and Login:

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Click on link for page	Request HREF access.	Specific page loaded	Respective page access given	PASS

23.Successful Fetch from API

Test No	Function	Input	Expected Result	Actual Result	Pass/Fail
a	Request Data using token and Display	Fetch on page load	Correct data display	Data fetched and displayed	PASS

CHAPTER 8

CONCLUSION

Through this project we have concluded that a payments maintenance app including the SplitPay functionality is a great feature which has a lot of good responses from the survey conducted by us.This can be easily used by group of members who usually pay bills for several things and then they could easily split the money among themselves using this app which will serves as a reminder for them in long run. we face difficulties when it comes to splitting bills, which is the case many times.So, this will be a user-centered design approach to develop an e-ledger application which not only supports maintaining payment records but also lets a group of users split a payment amongst themselves, and settle them at their own ease, within the app. Also we concluded through testing that this project is ready for deployment once API access code from Google Developers portal is received.