NOVASYNC
-Expense Splitting Application

INTRODUCTION

Novasync is a group-based expense management web application
It helps users create groups, add members, add expense, view ledger and obtain a minimal-set settlement plan that tells who should pay whom.

Tech Stack

Technologies Used- Frontend: React
Backend: Node.js (Express)
Database: MongoDB (Mongoose)
Tools: VS code, MongoDB Atlas, Thunder Client

SYSTEM ARCHITECTURE

Architecture Diagram:

User → React UI → Express API → MongoDB

- User interacts with UI
- React sends API requests
- Express handles logic
- MongoDB stores data

PROJECT FOLDER STRUCTURE

frontend/

pages/

- Login.js
- Register.js
- Dashboard.js
- Group.js
- AddGroup.js
- AddMember.js
- AddExpense.js
- Ledger.js
- Settlement.js

components/

- Navbar.js

App.js

style.css

backend/

models/

- user.js
- group.js
- expense.js

controllers/

- userController.js
- groupController.js
- expenseController.js
- settlementController.js

routes/

- userRoutes.js
- groupRoutes.js
- expenseRoutes.js
- settlementRoutes.js

db.js

server.js

DATABASE MODELS

Database Schema Design:

Separate schemas are used to maintain clear relationships between users, groups, and expenses

```
  name:String,
  email:String,
password:String
```

User Model

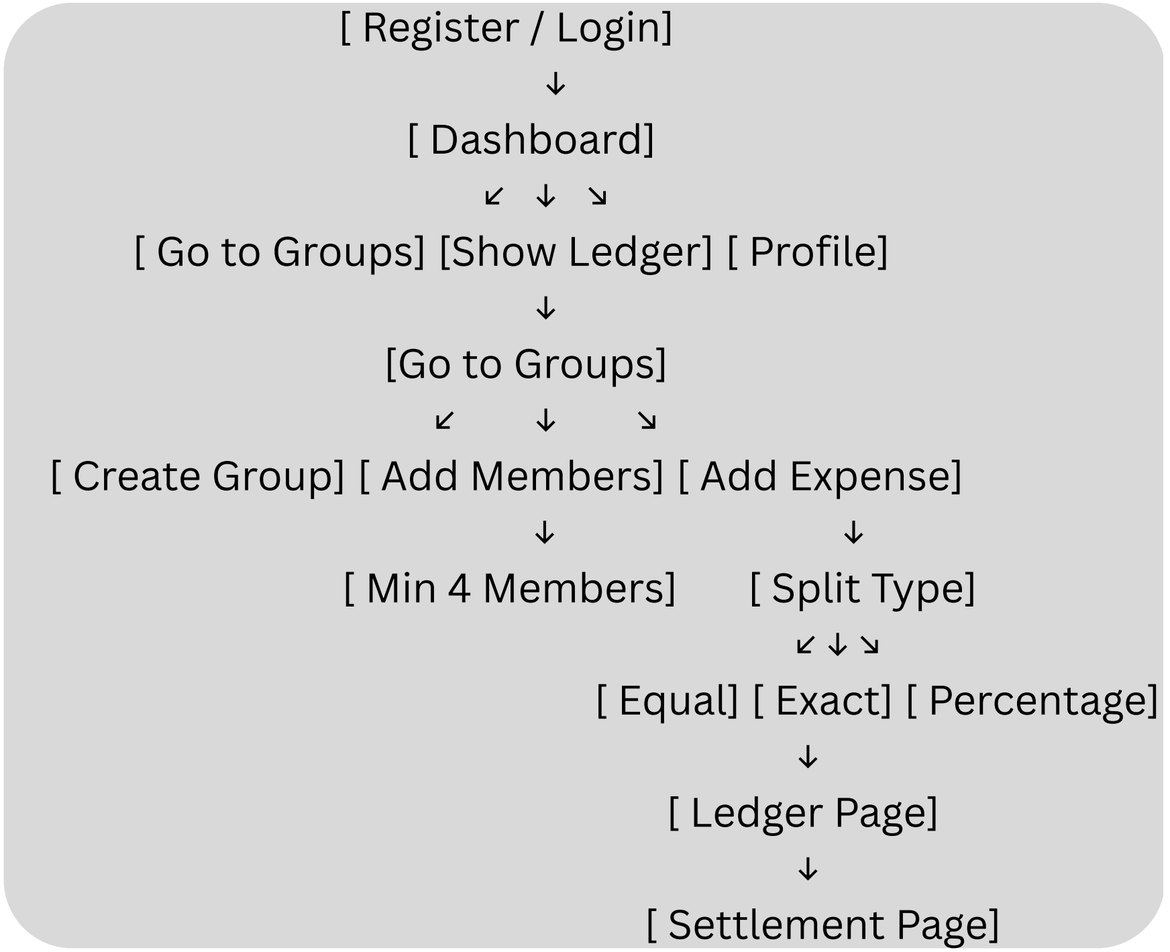
```
  groupName:String,
  members:{
  type:[
    {
      name: String,
  email: String,
}
],
  default:[]
},
  createdBy: {
  type: mongoose.Schema.Types.ObjectId,
  ref : "User",
},
```

Group Model

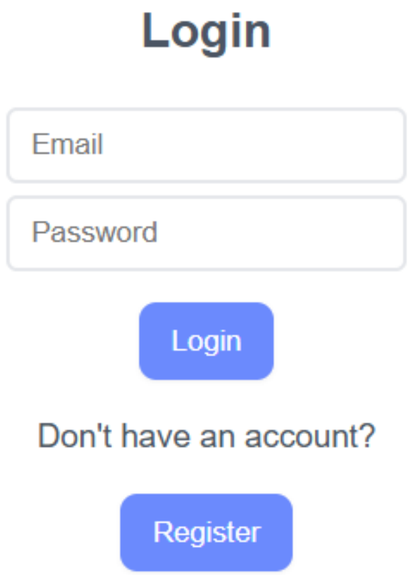
```
groupId:{
  type: mongoose.Schema.Types.ObjectId,
  ref:"Group",
  required: true,
},
title:{
  type: String,
equired:true,
},

amount:{
  type: Number,
  required: true,
},

splitType:{
  type: String,
  enum: ["equal", "exact", "percentage"],
  required: true,
},

splits:[
  {
  name: String,
  share: Number,
},
,
```

Expense Model

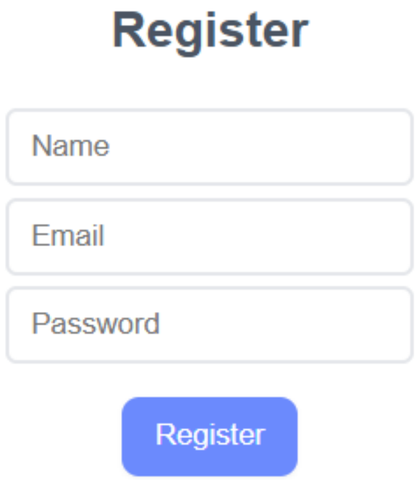
APPLICATION FLOW



USER AUTHENTICATION



Login Page



Registration Page

Backend Logic

```
const hashedPassword=await
bcrypt.hash(password, 10)
const isMatch= await
bcrypt.compare(password,
user.password)
```

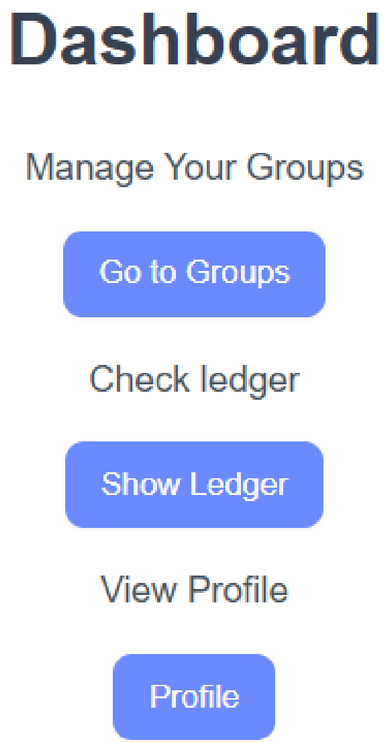
User Route

POST /api/users/signup→ Register user

POST /api/users/login → Login user

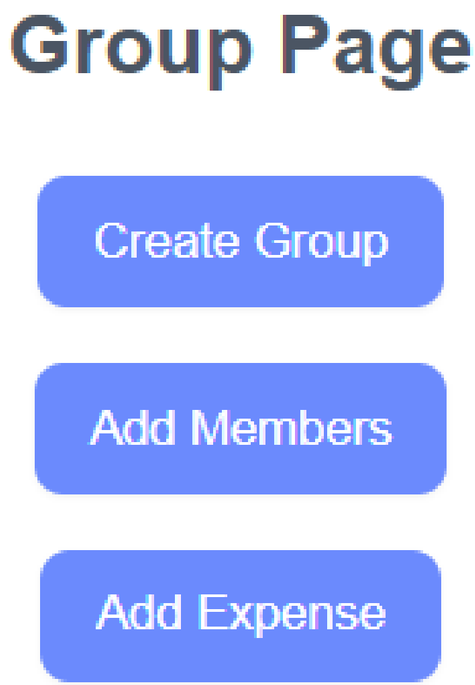
- User Authentication is implemented using bcrypt for password security.

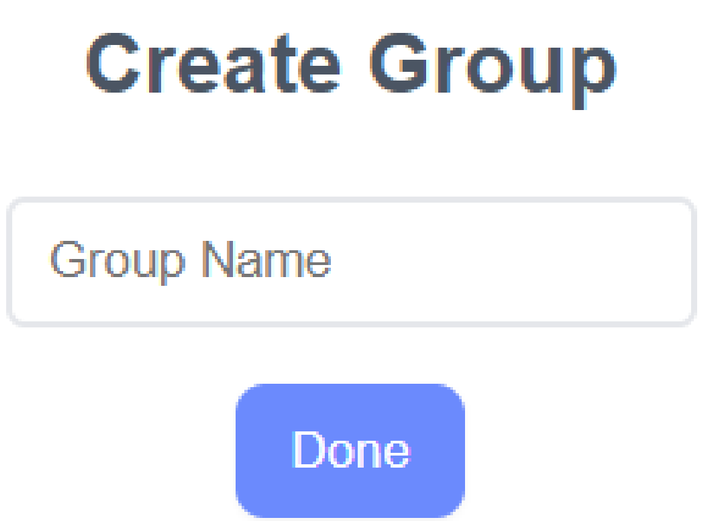
DASHBOARD PAGE



- Acts as navigation hub
- User can go to Groups, Ledger, Profile
- Page navigation handled using React state

GROUP CREATION





Backend Logic

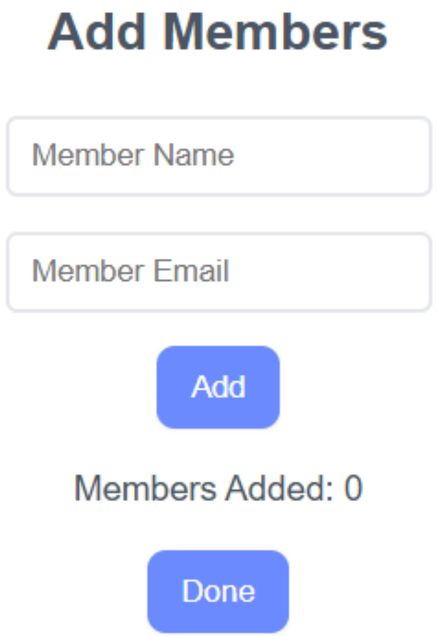
```
const group=new Group({
  groupName,createdBy:userId,members:[]
});
await group.save();
```

Group Route

POST /api/groups/create

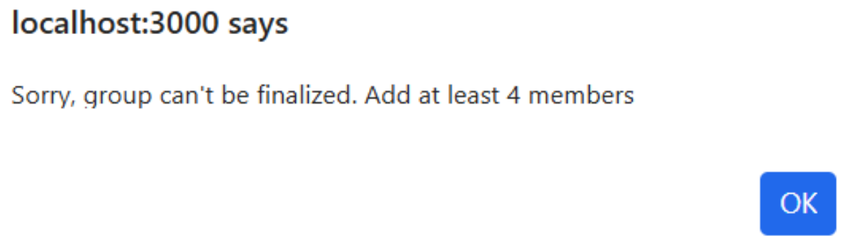
- Creates a new group in MongoDB
- Stores selected group ID
- Controls flow between add members & add expense

MEMBER ADITION



- Members added one by one
- Duplicate members prevented

- Minimum 4 members required to finalize group



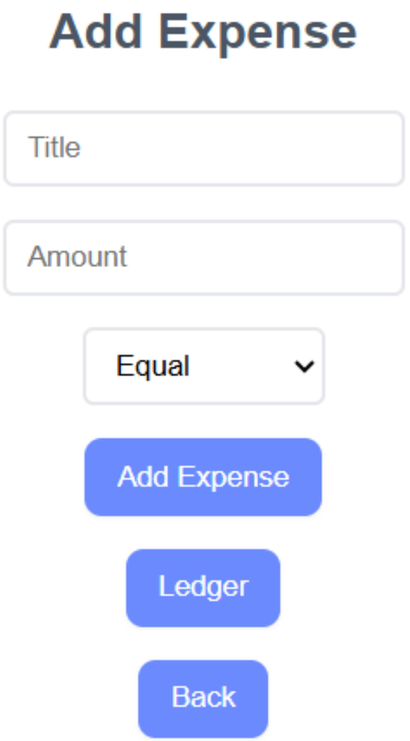
Backend logic

```
  if (!group.members || group.members.length < 4) {
    return res.json({
      success: false,
      message: "Sorry, group can't be finalized. Add at least 4 members",
    });
  }
  return res.json({ success: true, message: "Group finalized successfully" });
}

// Regular add member flow
if (!name || !email) {
  return res.json({ success: false, message: "Missing name or email" });
}

// Avoid duplicates
if (group.members.find((m) => m.email === email)) {
  return res.json({ success: false, message: "Member already added" });
}

group.members.push({ name, email });
await group.save();
```

Members Route

POST /api/groups/add-members

EXPENSE MANAGEMENT



Split Types

- Percentage
- Equal
- Exact

Expense Route

POST /api/expenses/add

Backend Logic

```
if(splitType==="percentage"){
  participants = splits.map((s)=>({
    name: s.name,
    share: (amount*s.percent)/100,
  }));
}
```

- Each participant's share is calculated as: share= (total amount*participant percent)/100
- Paid:0 no one has paid yet
- Useful when participants contribute different proportions

```
if(splitType==="exact"){
  participants = splits.map((s)=>({
    name: s.name,
    share: s.amount,
    paid: 0
  }));
}
```

- Each Participant's share is predefined by the expense creator.
- Paid:0 no payment has occurred yet, ledger will track actual payments
- Useful when participants pay for different items or unequal contributions

```
if(splitType==="equal"){
  const share = amount/group.members.length;
  participants = group.members.map((m)=>({
    name: m.name,
    share,
    paid: 0
  }));
}
```

- Divide the total amount equally among all participants
- Each participant owes the same amount, regardless of what actually paid
- Paid:0 initial state before any payment

- currently does not store who actually paid

LEDGER FEATURE

- Chronological record of all expenses shows participants, their share, and paid amount
- Calculate Net Balance = Paid-Share
- A positive balance means they are owed money, a negative balance means they owe money

Backend Logic

```
const expenses= await Expense.find({groupId:groupId})
```

Ledger Route

```
GET /api/expenses/ledger/:groupId
```

SETTLEMENT PLAN

- The settlement algorithm reduces the number of transactions by matching debtors to creditors until all balances are zero

Pseudo Code

```
Input: net_balances = {name: balance}  // +ve = owed, -ve = owes

creditors = list of participants with balance > 0
debtors = list of participants with balance < 0

for each debtor in debtors:
for each creditor in creditors:
if debtor.balance == 0:
break
pay_amount = min(abs(debtor.balance), creditor.balance)
print(debtor.name, "pays", pay_amount, "to", creditor.name)
debtor.balance += pay_amount
creditor.balance -= pay_amount
```

Settlement Route

GET /api/settlement/:groupId

CONCLUSION

- **Implemented Features**

User authentication (register & login), Expense group creation, Add members to a group,
Add expense request handling: Accepts expense details (amount, participants, split type)
Split calculation logic implemented: Equal split, Exact amount split, Percentage-based split

- **Current Limitations**

Expense data is not yet persisting in the database, Ledger computation depends on stored expenses and is not functional, Settlement execution is not implemented, Paid amount per participant is not recorded

- **Overall Status**

Core user and group management features are functional, Expense splitting logic is implemented at API level, Ledger and settlement are designed conceptually but not executed

Thank You !