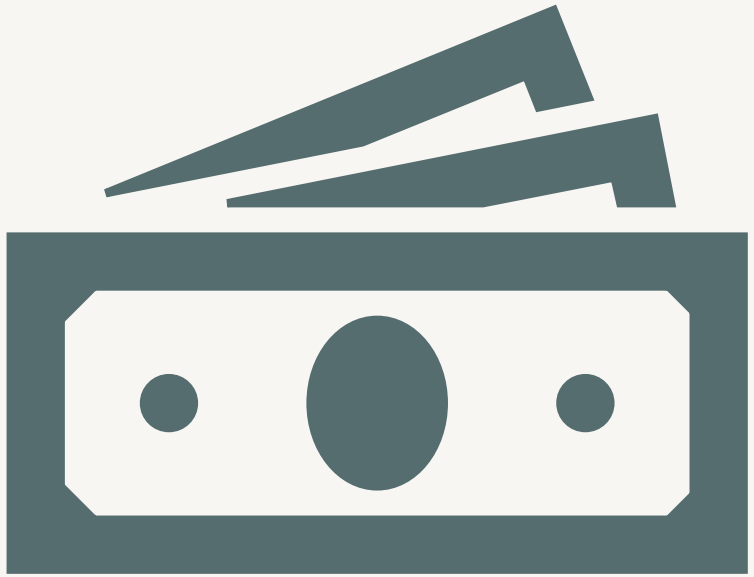
Group members

Eunseong Han

Samuel Alemu

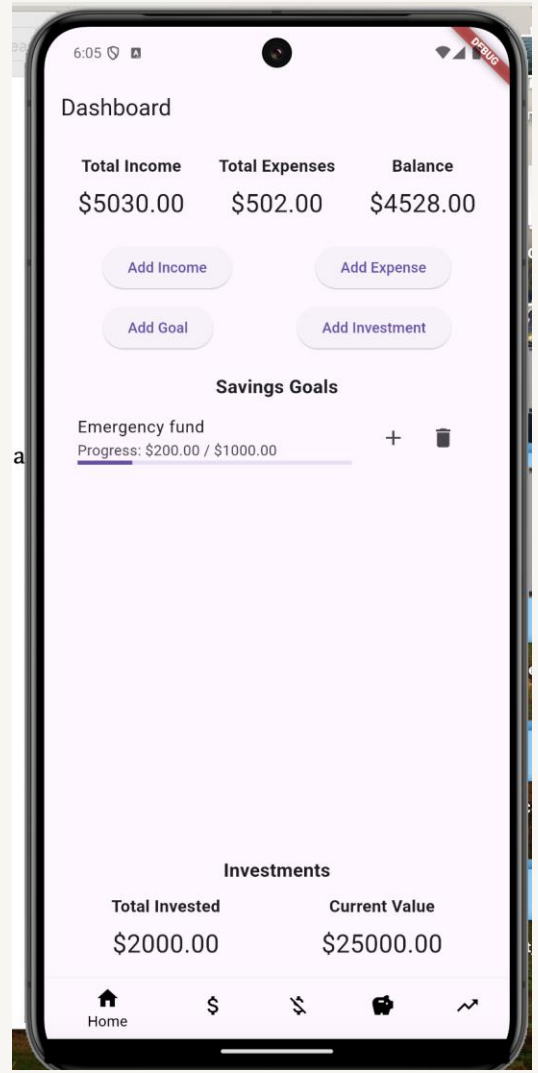
Personal finance manager

- *"I got paid recently, but I don't know where my money goes."* If this sounds familiar, then the Finance Manager app is designed just for you. In today's world of high consumption, keeping track of expenses and income can be a challenge. The Finance Manager app empowers users to effectively track and manage their hard-earned money. By monitoring spending, setting saving goals, and keeping tabs on investments, users can gain a clear picture of their financial health. We chose to work on this project to address a common problem faced by many: understanding where their money is going and managing it wisely.



Dashboard

The dashboard presents a clear and organized summary of the user's financial status. It includes key financial metrics like net income, expenses, balance, saving goals, and investments. Integrated buttons allow users to quickly add income, expenses, goals, and investment amounts. A convenient Navigation Bar at the bottom guides users to dedicated pages for specific financial sections.



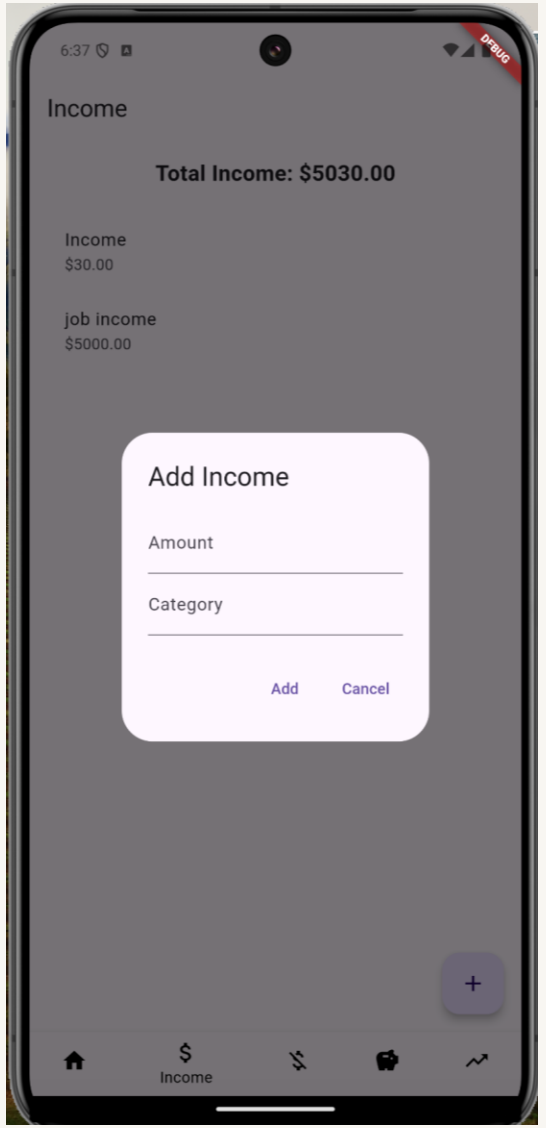
DashBoard code snippet

```
class DashboardPage extends StatefulWidget {
  @override
  _DashboardPageState createState() => _DashboardPageState();
}

class _DashboardPageState extends State<DashboardPage> {
  double totalIncome = 0.0;
  double totalExpenses = 0.0;
  double totalInvested = 0.0;
  double currentInvestmentValue = 0.0;
  List<Map<String, dynamic>> savingsGoals = [];
  List<Map<String, dynamic>> investmentList = [];

  @override
  void initState() {
    super.initState();
    _loadDashboardData();
  }

  Future<void> _loadDashboardData() async {
    final income = await DatabaseHelper.instance.getTotalIncome();
    final expenses = await DatabaseHelper.instance.getTotalExpenses();
    final goals = await DatabaseHelper.instance.getAllSavingsGoals();
    await _loadInvestmentData();

    setState(() {
      totalIncome = income ?? 0.0;
      totalExpenses = expenses ?? 0.0;
      savingsGoals = goals;
    });
  }
```

```
  Future<void> _loadInvestmentData() async {
    final investments = await DatabaseHelper.instance.getAllInvestments();

    double invested = 0.0;
    double currentValue = 0.0;
    for (var investment in investments) {
      invested += investment['amount'];
      currentValue += investment['current_value'];
    }

    setState(() {
      investmentList = investments;
      totalInvested = invested;
      currentInvestmentValue = currentValue;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Dashboard')),
      body: // Add your UI here,
      bottomNavigationBar: BottomNavBar(currentIndex: 0),
    );
  }
}
```

Adding to the list

- The Floating Action Button provides a convenient way to add items to the Income and Expense pages. It enhances the user experience by offering a quick and accessible way to input financial details. When pressed, the Floating Action Button triggers an **Adding List** that allows users to input the amount and select relevant categories for income or expense entries. This streamlined approach helps organize financial information efficiently.



Add income code snippet

```
void _showAddIncomeDialog() {
  final _amountController = TextEditingController();
  final _categoryController = TextEditingController();
  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: Text("Add Income"),
        content: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            TextField(
              controller: _amountController,
              keyboardType: TextInputType.number,
              decoration: InputDecoration(labelText: "Amount"),
            ),
            TextField(
              controller: _categoryController,
              decoration: InputDecoration(labelText: "Category"),
            ),
          ],
        ),
        actions: [
          TextButton(
            onPressed: () {
              final amount = double.tryParse(_amountController.text);
              final category = _categoryController.text;
              if (amount != null && category.isNotEmpty) {
                _addIncome(category, amount);
                Navigator.of(context).pop(true);
              } else {
                Navigator.of(context).pop(false);
              }
            },
            child: Text("Add"),
          ),
          TextButton(
            onPressed: () => Navigator.of(context).pop(false),
            child: Text("Cancel"),
          ),
        ],
      );
    },
  );
}

Future<void> _addIncome(String category, double amount) async {
  await DatabaseHelper.instance.insertIncome({'category': category, 'a
  _loadIncomeData();
}
```

Income page

- The Income Page allows users to efficiently record their generated income. It offers a streamlined way to categorize and input earnings. A **Floating Action Button** is conveniently placed to let users add specific income categories and their corresponding amounts. This helps maintain a clear record of various income streams.

Income code snippet

```
class IncomePage extends StatefulWidget {
  @override
  _IncomePageState createState() => _IncomePageState();
}

class _IncomePageState extends State<IncomePage> {
  double totalIncome = 0.0;
  List<Map<String, dynamic>> incomeHistory = [];

  @override
  void initState() {
    super.initState();
    _loadIncomeData();
  }

  Future<void> _loadIncomeData() async {
    final data = await DatabaseHelper.instance.getAllIncome();
    final sum = await DatabaseHelper.instance.getTotalIncome();
    setState(() {
      incomeHistory = data;
      totalIncome = sum ?? 0.0;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Income')),
      body: // Add your UI here,
      bottomNavigationBar: BottomNavBar(currentIndex: 1),
    );
  }
}
```

Expense page

- The Expense Page is designed for users to accurately record and categorize their expenses. This ensures a clear understanding of spending patterns. Similar to the Income Page, a **Floating Action Button** is provided to add specific expense categories and their respective amounts. This feature helps users keep track of various types of expenditures effectively.

Expense code snippet

```
class ExpensePage extends StatefulWidget {
  @override
  _ExpensePageState createState() => _ExpensePageState();
}

class _ExpensePageState extends State<ExpensePage> {
  double totalExpenses = 0.0;
  List<Map<String, dynamic>> expenseHistory = [];

  @override
  void initState() {
    super.initState();
    _loadExpenseData();
  }

  Future<void> _loadExpenseData() async {
    final data = await DatabaseHelper.instance.getAllExpenses();
    final sum = await DatabaseHelper.instance.getTotalExpenses();
    setState(() {
      expenseHistory = data;
      totalExpenses = sum ?? 0.0;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Expense')),
      body: // Add your UI here,
      bottomNavigationBar: BottomNavBar(currentIndex: 2),
    );
  }
}
```

Saving page

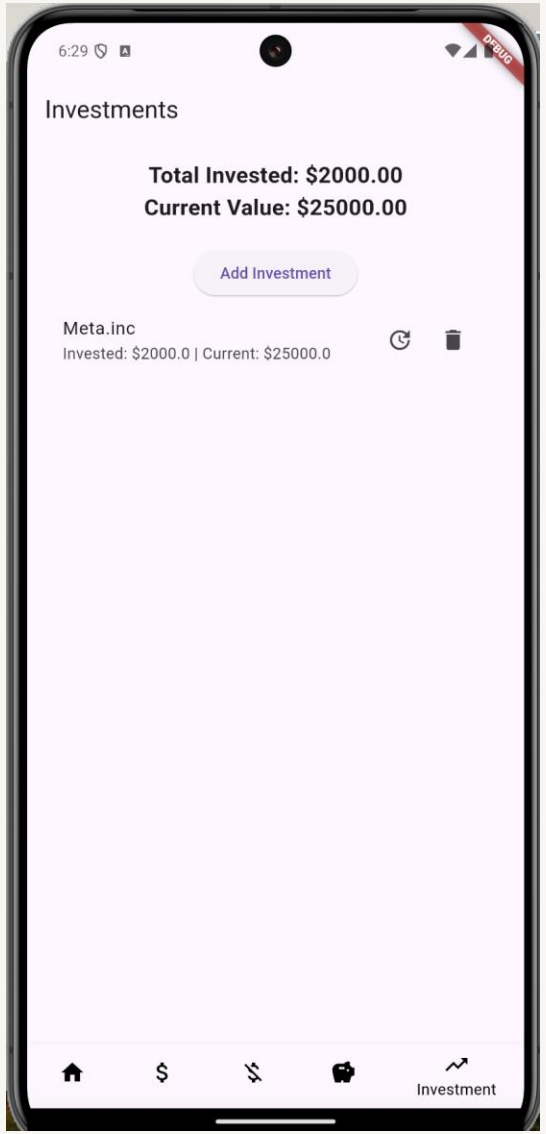
- The Saving Page enables users to set specific saving goals and monitor their progress on a single, dedicated page. It helps users stay focused on achieving their financial targets. Users can easily define their saving goals, set target amounts, and track the progress visually. The page provides a clear overview of each goal's status to motivate consistent saving habits.

Saving code snippet

```
class SavingsPage extends StatefulWidget {
  @override
  _SavingsPageState createState() => _SavingsPageState();
}

class _SavingsPageState extends State<SavingsPage> {
  List<Map<String, dynamic>> savingsGoals = [];

  @override
  void initState() {
    super.initState();
    _loadSavingsGoals();
  }

  Future<void> _loadSavingsGoals() async {
    final goals = await DatabaseHelper.instance.getAllSavingsGoals();
    setState(() {
      savingsGoals = goals;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Savings')),
      body: // Add your UI here,
      bottomNavigationBar: BottomNavBar(currentIndex: 3),
    );
  }
}
```

Investment page

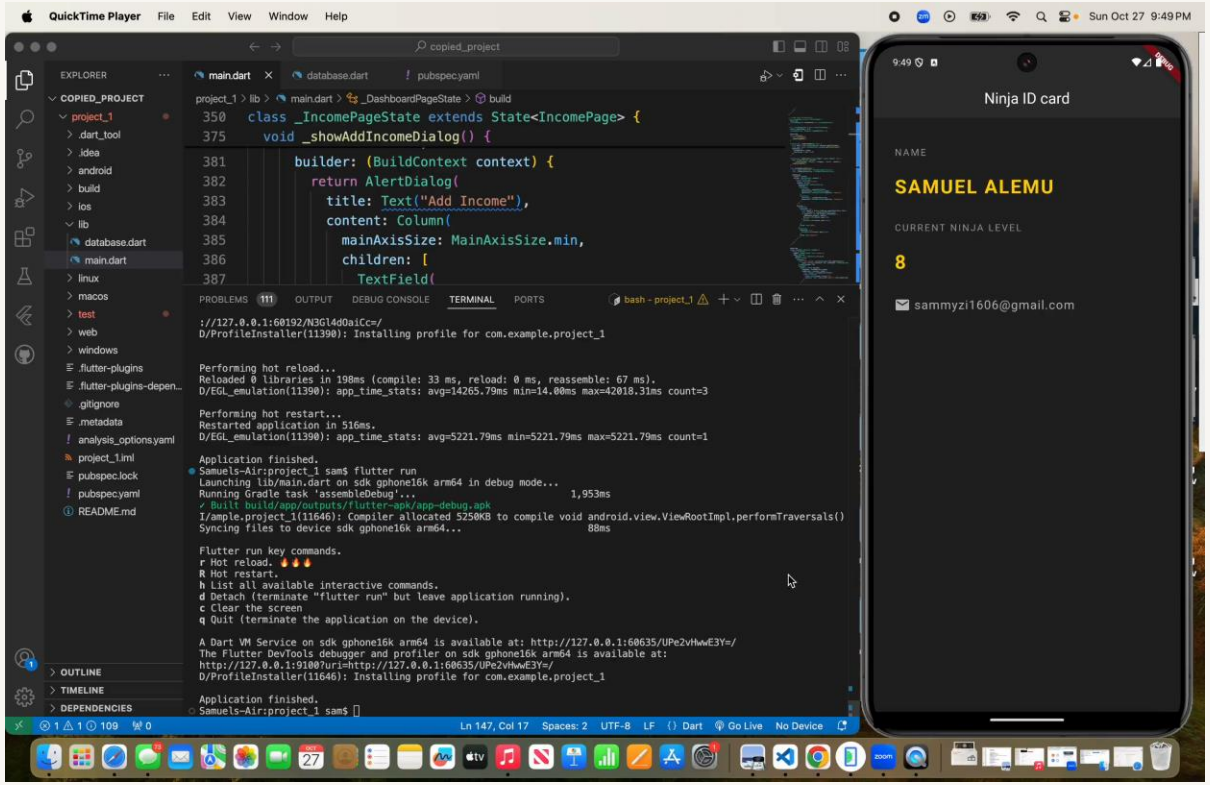
The Investment Page is dedicated to providing users with a comprehensive view of their investments. It allows users to track their investments and assess their financial growth. This page displays the current value of investments alongside the originally invested amount, enabling users to easily compare and evaluate their profits or losses. This clear comparison helps in making informed financial decisions.



Investment code snippet

```
class InvestmentPage extends StatefulWidget {
  @override
  _InvestmentPageState createState() => _InvestmentPageState();
}

class _InvestmentPageState extends State<InvestmentPage> {
  List<Map<String, dynamic>> investments = [];
  double totalInvested = 0.0;
  double currentTotalValue = 0.0;

  @override
  void initState() {
    super.initState();
    _loadInvestmentData();
  }

  Future<void> _loadInvestmentData() async {
    final investmentList = await DatabaseHelper.instance.getAllInvestme

    double invested = 0.0;
    double currentValue = 0.0;
    for (var investment in investmentList) {
      invested += investment['amount'];
      currentValue += investment['current_value'];
    }

    setState(() {
      investments = investmentList;
      totalInvested = invested;
      currentTotalValue = currentValue;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Investments")),
      body: // Add your UI here,
      bottomNavigationBar: BottomNavBar(currentIndex: 4),
    );
  }
}
```

Demo video


QuickTime Player File Edit View Window Help
Sun Oct 27 9:49 PM
Ninja ID card
NAME
SAMUEL ALEMU
CURRENT NINJA LEVEL
8
sammyzi1606@gmail.com