**Mingus Implementation Guide for Cursor Platform**

*Step-by-Step Instructions for Non-Technical Founders*

**Overview**

This guide breaks down each step into simple, actionable tasks you can complete using Cursor AI. Each step includes what to do, why you're doing it, and how to ask Cursor for help.

**Phase 1: Setup & Environment (Week 1)**

**Step 1: Organize Your Project Structure**

**What you're doing**: Creating a clean folder structure for your code **Why**: Keeps everything organized and makes development easier

**Actions:**

1. **Create a new folder** on your computer called mingus-app
2. **Open Cursor** and click "Open Folder"
3. **Select your mingus-app folder**
4. **Ask Cursor**: "Please create a proper folder structure for a personal finance web application with Python backend and vanilla JavaScript frontend"

**Expected Result**: Cursor will create folders like:

mingus-app/

├── backend/

├── frontend/

├── database/

├── scripts/

└── docs/

**Step 2: Set Up Your Development Environment**

**What you're doing**: Installing the tools you need to code **Why**: Your computer needs these tools to run Python and connect to databases

**Actions:**

1. **Ask Cursor**: "I need to set up a Python development environment for a web application that connects to Supabase. What do I need to install and how?"
2. **Follow Cursor's instructions** to install:
   * Python (if not already installed)
   * pip (Python package manager)
   * Required Python libraries
3. **Ask Cursor**: "Please create a requirements.txt file with all the Python packages I'll need for a personal finance app using Supabase"

**Expected Result**: You'll have a file listing all the Python tools you need

**Step 3: Connect to Your Existing Supabase Database**

**What you're doing**: Creating a connection between your Python code and your database **Why**: Your application ann needs to read and write data to your existing financial data

**Actions:**

1. **Go to your Supabase dashboard** and find your project URL and API keys
2. **Ask Cursor**: "I have a Supabase database with URL [your-url] and I need to create a Python connection file. Help me create a secure connection that uses environment variables"
3. **Create a .env file** (Cursor will help) to store your secret database keys
4. **Test the connection** by asking Cursor: "Create a simple Python script to test if I can connect to my Supabase database"

**Expected Result**: A working connection to your existing database

**Phase 2: Fix Your Cash Flow System (Week 2)**

**Step 4: Update Your Database Table Structure**

**What you're doing**: Making small changes to your daily\_cashflow table to prevent problems **Why**: Some column names can cause database errors

**Actions:**

1. **Ask Cursor**: "I have a Supabase table called daily\_cashflow with these columns: [list your 9 columns]. I need to rename 'date' to 'forecast\_date' and 'status\_color' to 'balance\_status'. Show me the SQL commands to do this safely."
2. **Go to your Supabase dashboard** → SQL Editor
3. **Copy and paste** the SQL commands Cursor gives you
4. **Run the commands** one at a time

**Expected Result**: Your table has better column names that won't cause problems

**Step 5: Improve Your Date Generation Code**

**What you're doing**: Taking your existing Python code and making it work for a full year **Why**: 30 days isn't enough for proper financial forecasting

**Actions:**

1. **Show Cursor your existing code**: Copy and paste your generate\_schedule.py file
2. **Ask Cursor**: "This code only generates 30 days of dates. I need it to generate 12 months of dates and handle business days for paychecks. Please improve this code."
3. **Save the improved code** as enhanced\_schedule.py
4. **Test it** by asking Cursor: "Create a simple test script to make sure this enhanced code works correctly"

**Expected Result**: Code that generates a full year of payment dates

**Step 6: Connect Your Date Generation to Supabase**

**What you're doing**: Making your Python code work with your existing database tables **Why**: You need to populate your user\_income\_due\_dates and user\_expense\_due\_dates tables

**Actions:**

1. **Ask Cursor**: "I have tables called 'user\_income\_due\_dates' and 'user\_expense\_due\_dates' in Supabase. I need Python functions that use my date generation code to populate these tables with a user's income and expense schedules. Here's my current database schema: [paste your schema]"
2. **Save the code** Cursor creates as populate\_schedules.py
3. **Test with one user** by asking Cursor: "Create a test script that adds sample data for one fake user to test if the schedule population works"

**Expected Result**: Functions that can fill your database tables with payment dates

**Phase 3: Fix Daily Cash Flow Calculations (Week 3)**

**Step 7: Create the Daily Balance Calculator**

**What you're doing**: Building the system that calculates your daily cash balance **Why**: This is the core of your cash flow forecasting feature

**Actions:**

1. **Ask Cursor**: "I need a Python function that reads from user\_income\_due\_dates and user\_expense\_due\_dates tables, then calculates daily cash balances for the next 12 months. The results should go into my daily\_cashflow table with these columns: [list your 9 columns]"
2. **Save this code** as calculate\_cashflow.py
3. **Ask Cursor**: "Create a simple test to make sure this calculation works correctly with sample data"

**Expected Result**: A function that creates accurate daily cash flow forecasts

**Step 8: Put It All Together**

**What you're doing**: Creating one master function that does everything **Why**: You want one button that updates a user's entire financial forecast

**Actions:**

1. **Ask Cursor**: "I want to combine all my previous functions into one master function called 'refresh\_user\_forecast' that takes a user\_id and updates their entire financial forecast. Please create this using my existing functions."
2. **Save this** as master\_forecast.py
3. **Test the complete system** by asking Cursor: "Create a test script that runs the complete forecast refresh for a fake user and shows me the results"

**Expected Result**: One function that completely updates a user's financial forecast

**Phase 4: Build the Web Interface (Week 4)**

**Step 9: Create a Simple Web Backend**

**What you're doing**: Building a web server that your frontend can talk to **Why**: Your website needs a way to trigger the forecast calculations

**Actions:**

1. **Ask Cursor**: "I need a simple Flask web server that can trigger my master\_forecast function when someone visits a webpage. Please create this with basic error handling."
2. **Save this** as app.py
3. **Test it** by asking Cursor: "Show me how to run this Flask app and test it in my browser"

**Expected Result**: A simple web server that can run your forecast calculations

**Step 9a: Create Key Dates Capability**

**Use the Key Date Doc in folder Mingus 2025 - Cursor**

I'll create comprehensive prompts for a frictionless account creation and onboarding process that balances ease of use with the data collection needed for accurate cash flow forecasting.I've created a comprehensive set of 5 prompts that will guide you through building a complete account creation and onboarding system. Here's the strategic breakdown:

**Recommended Information to Collect**

**Required (Minimal Friction)**

* **Name & Contact:** First/last name, email, phone
* **Basic Demographics:** Date of birth, ZIP code
* **Primary Income:** Main job amount and frequency
* **Essential Expenses:** Housing, transportation, utilities
* **Starting Balance:** Current checking account balance

**Progressive Collection (Higher Value)**

* **Additional Income Sources:** Side hustles, child support, investments
* **Detailed Expenses:** Cultural categories, discretionary spending
* **Financial Goals:** Emergency fund, savings targets

**Social Sign-On Recommendations**

**Priority Order:**

1. **Google** (highest trust, professional association)
2. **Instagram** (high engagement with your demographic)
3. **LinkedIn** (professional credibility, potential income verification)
4. **Facebook** (broad reach, family connections)

**Why This Order:**

* **Google:** Most secure, widely adopted, professional feel
* **Instagram:** Your target demographic actively uses it
* **LinkedIn:** Adds credibility for financial services
* **Facebook:** Backup option for broader accessibility

**Key Strategic Benefits**

**Frictionless Experience:**

* Social login reduces barriers by 60-80%
* Progressive data collection prevents form abandonment
* Immediate value (cash flow forecast) after minimal input
* Skip options prevent user frustration

**Cultural Sensitivity:**

* Expense categories include fraternity/sorority, family support, cultural celebrations
* Non-judgmental language around income sources
* Trust-building messaging throughout process
* Community-focused rather than individual-only approach

**Technical Advantages:**

* OAuth integration with Supabase Auth
* Progressive data enhancement
* Secure token management
* GDPR/CCPA compliance built-in

**User Flow Summary**

**New Users:** Splash → Welcome → Social Login → Income Setup → Expense Setup → Starting Balance → Dashboard Preview

**Returning Users:** Quick Login → Dashboard (immediate cash flow status, upcoming dates, insights)

**Incomplete Onboarding:** Resume exactly where they left off with preserved data

**Implementation Priority**

**Week 1:** Core authentication (Prompt 1) **Week 2:** Income collection (Prompt 2)  
**Week 3:** Expense setup (Prompt 3) **Week 4:** Balance integration & dashboard (Prompts 4-5)

Each prompt includes complete technical specifications, database schemas, cultural considerations, and user experience guidelines. The system balances ease of use with the comprehensive data collection needed for accurate cash flow forecasting.

Would you like me to elaborate on any specific aspect or help you prioritize which prompt to implement first?

**Step 10: Create a Basic Dashboard**

**What you're doing**: Building a webpage that shows your cash flow data **Why**: Users need to see their financial forecast in a user-friendly way

**Actions:**

1. **Ask Cursor**: "I need a simple HTML page with vanilla JavaScript that connects to my Supabase database and shows a user's daily cash flow data in a table and simple chart. Please create this."
2. **Save the HTML** as frontend/dashboard.html
3. **Save the JavaScript** as frontend/dashboard.js
4. **Save the CSS** as frontend/styles.css

**Expected Result**: A basic webpage that displays financial data

**Step 11: Add a Forecast Refresh Button**

**What you're doing**: Adding a button that updates the forecast **Why**: Users need a way to recalculate their forecast when they change their income or expenses

**Actions:**

1. **Ask Cursor**: "Add a 'Refresh Forecast' button to my dashboard that calls my Flask backend to update the user's forecast, then refreshes the displayed data"
2. **Test it** by clicking the button and seeing if the data updates

**Expected Result**: A working button that updates forecasts

**Phase 5: Basic Testing (Week 5)**

**Step 12: Create Test Data**

**What you're doing**: Making fake user data to test your system **Why**: You need to make sure everything works before real users try it

**Actions:**

1. **Ask Cursor**: "Create a script that adds realistic test data to my Supabase database - including sample income, expenses, and user information for testing"
2. **Run the script** to populate your database
3. **Test your dashboard** with this fake data

**Expected Result**: Fake data that lets you test your entire system

**Step 13: Test Everything End-to-End**

**What you're doing**: Making sure the complete system works from start to finish **Why**: You need to find and fix problems before launch

**Actions:**

1. **Ask Cursor**: "Create a testing checklist for my personal finance app that covers all the key features"
2. **Go through each item** on the checklist
3. **Fix any problems** by asking Cursor for help with specific errors

**Expected Result**: A fully working cash flow forecasting system

**Phase 6: Prepare for Next Steps (Week 6)**

**Step 14: Set Up Basic Security**

**What you're doing**: Making sure your app is secure for real users **Why**: Financial apps need strong security

**Actions:**

1. **Ask Cursor**: "What basic security measures do I need for a personal finance web app? Please help me implement user authentication and data protection"
2. **Follow Cursor's recommendations** step by step

**Expected Result**: Basic security measures in place

**Step 15: Plan Your Health Integration**

**What you're doing**: Getting ready to add the health features that make Mingus unique **Why**: This is what will differentiate your app from competitors

**Actions:**

1. **Ask Cursor**: "Based on my existing database schema, help me plan how to add health check-in features that track physical activity, relationships, and mindfulness, then correlate them with spending patterns"
2. **Create a plan** for implementing these features next

**Expected Result**: A clear roadmap for adding health integration

**How to Use This Guide**

**Daily Work Pattern:**

1. **Start each work session** by opening Cursor and your mingus-app folder
2. **Focus on one step at a time** - don't try to do everything at once
3. **Ask Cursor specific questions** about each step
4. **Test everything** as you go - don't wait until the end

**When You Get Stuck:**

1. **Copy the exact error message** and ask Cursor: "I'm getting this error: [paste error]. How do I fix it?"
2. **Be specific** about what you're trying to do
3. **Ask for simpler explanations** if Cursor's answer is too technical

**Key Phrases to Use with Cursor:**

* "Explain this like I'm a beginner"
* "Show me the exact steps to do this"
* "Create a simple test to make sure this works"
* "What could go wrong with this approach?"
* "How do I debug this problem?"

**Time Expectations:**

* **Each step**: 2-4 hours of focused work
* **Each week**: 15-20 hours total
* **Complete system**: 6 weeks to working prototype

**Success Indicators:**

* **Week 1**: You can run Python code and connect to your database
* **Week 2**: You can generate payment schedules automatically
* **Week 3**: You can calculate daily cash flow balances
* **Week 4**: You have a working web interface
* **Week 5**: Everything works with test data
* **Week 6**: You're ready to add health features and real users

Remember: **Cursor is your coding partner.** Ask it questions, have it explain things, and don't be afraid to ask for help when you're stuck!