**Mingus Splash Page Implementation Steps**

**Phase 1: Database Setup & Meme Management**

**Step 1: Create Meme Database Schema**

CREATE TABLE memes (

id INTEGER PRIMARY KEY,

image\_url TEXT NOT NULL,

category TEXT NOT NULL,

caption TEXT,

alt\_text TEXT,

is\_active BOOLEAN DEFAULT TRUE,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE user\_meme\_history (

id INTEGER PRIMARY KEY,

user\_id INTEGER,

meme\_id INTEGER,

viewed\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (user\_id) REFERENCES users(id),

FOREIGN KEY (meme\_id) REFERENCES memes(id)

);

**Step 2: Populate Meme Categories**

Create initial meme collections for each theme:

* **Faith**: Motivational quotes, Sunday vibes, prayer reminders
* **Work Life**: Monday motivation, career growth, workplace humor
* **Friendships**: Squad goals, friend check-ins, social connections
* **Children**: Parenting wins, kid activities, family time
* **Relationships**: Love reminders, date night ideas, relationship goals
* **Going Out**: Weekend plans, social events, self-care outings

**Step 3: Build Meme Management System**

* Admin interface to upload/manage memes
* Category assignment functionality
* Ability to activate/deactivate memes
* Preview functionality for testing

**Phase 2: Splash Page Logic Development**

**Step 4: Create Meme Selection Algorithm**

def get\_user\_splash\_meme(user\_id):

# Get user's recent meme history (last 30 days)

recent\_memes = get\_recent\_user\_memes(user\_id, days=30)

# Get user's wellness data to determine relevant category

user\_context = analyze\_user\_context(user\_id)

# Select category based on:

# - Day of week (Sunday = faith, Monday = work life)

# - Recent check-in data (stress levels, relationship status)

# - Upcoming events in their calendar

category = determine\_optimal\_category(user\_context)

# Get memes from category, excluding recently shown ones

available\_memes = get\_category\_memes(category, exclude=recent\_memes)

# Return random meme from available options

return random.choice(available\_memes)

**Step 5: Implement User Flow Logic**

* Check if user is returning (not first-time login)
* Determine splash page frequency (daily, every login, etc.)
* Create skip/continue mechanisms
* Track user engagement with splash content

**Phase 3: Frontend Development**

**Step 6: Design Splash Page UI**

**Mobile-First Design Elements:**

* Full-screen meme display with optimal sizing
* Clean, minimal interface matching Mingus branding
* "Continue to Dashboard" button
* Optional "Share" functionality
* Subtle category indicator

**Step 7: Build Responsive Components**

// React component structure

const SplashPage = ({ user, onContinue }) => {

const [meme, setMeme] = useState(null);

const [loading, setLoading] = useState(true);

useEffect(() => {

fetchUserMeme(user.id).then(setMeme);

}, [user.id]);

return (

<div className="splash-container">

<MemeDisplay meme={meme} />

<ActionButtons onContinue={onContinue} meme={meme} />

</div>

);

};

**Step 8: Implement Loading States**

* Skeleton loading for meme content
* Fallback content if meme fails to load
* Smooth transitions between splash and dashboard

**Phase 4: Integration & User Experience**

**Step 9: Integrate with Authentication Flow**

# Update login/session logic

def user\_login\_flow(user\_credentials):

user = authenticate\_user(user\_credentials)

if user.is\_returning\_user():

if should\_show\_splash(user):

return redirect\_to\_splash(user)

return redirect\_to\_dashboard(user)

**Step 10: Add Analytics & Tracking**

* Track meme engagement rates by category
* Monitor splash page skip rates
* A/B test different meme types and frequencies
* Measure impact on user retention and app usage

**Step 11: Create Content Moderation System**

* Review process for new memes
* Community reporting functionality
* Content guidelines aligned with Mingus values
* Regular auditing of meme relevance and appropriateness

**Phase 5: Advanced Features**

**Step 12: Personalization Engine**

* Learn from user interactions (likes, shares, skips)
* Adjust category preferences based on behavior
* Seasonal/holiday-themed content rotation
* Integration with user's financial milestones

**Step 13: Social Features (Optional)**

* Allow users to suggest memes
* Community voting on favorite content
* Share memes within Mingus community
* Create user-generated content campaigns

**Step 14: Performance Optimization**

* Image compression and CDN integration
* Preload next potential memes
* Cache frequently accessed content
* Optimize for various device speeds

**Implementation Timeline**

**Week 1-2**: Database setup and basic meme management **Week 3-4**: Core algorithm development and testing **Week 5-6**: Frontend UI/UX development **Week 7-8**: Integration testing and refinement **Week 9-10**: Analytics implementation and launch preparation

**Success Metrics**

* **Engagement**: 70%+ users interact with splash page
* **Retention**: Increased daily active user rate
* **Sentiment**: Positive user feedback on meme relevance
* **Performance**: <2 second load times for splash content

**Technical Considerations**

* **Content Storage**: Use cloud storage (AWS S3, Cloudinary) for meme images
* **API Design**: RESTful endpoints for meme retrieval and tracking
* **Caching**: Implement Redis for frequently accessed memes
* **Mobile Optimization**: Ensure smooth performance on various devices
* **Accessibility**: Alt text for all images, screen reader compatibility

# **Cursor AI Prompts for Mingus Meme Splash Page Feature**

**Prompt 1: Database Schema Setup**

Create a SQLite database schema for a meme splash page feature in a personal finance app called Mingus. I need:

1. A "memes" table that stores:

- Unique ID

- Image URL or file path

- Category (faith, work\_life, friendships, children, relationships, going\_out)

- Caption text

- Alt text for accessibility

- Active/inactive status

- Created and updated timestamps

2. A "user\_meme\_history" table that tracks:

- Which user saw which meme

- When they viewed it

- Foreign key relationships to users table

3. The SQL CREATE statements

4. Sample INSERT statements for 3 memes in each category

Please include proper indexing for performance and follow best practices for a production app.

**Prompt 2: Backend Meme Selection Logic**

Create a Python function for a personal finance app that selects the best meme to show a user. Context: Mingus is a finance app for African Americans age 25-35, connecting wellness to money decisions.

Requirements:

- Function should take user\_id as input

- Return a meme object with image, caption, category

- Logic should:

\* Avoid showing memes the user saw in last 30 days

\* Consider day of week (Sunday=faith, Monday=work\_life, Friday=going\_out)

\* Have fallback if no memes available in preferred category

\* Include error handling for database issues

\* Log analytics data for tracking

- Use SQLite database queries

- Include type hints and docstrings

- Make it beginner-friendly with clear comments

- Include a simple caching mechanism for performance

Database table structure: memes(id, image\_url, category, caption, alt\_text, is\_active, created\_at) and user\_meme\_history(user\_id, meme\_id, viewed\_at)

**Prompt 3: Meme Management Admin Interface**

Create a simple Flask admin interface for managing memes in a personal finance app. I need:

1. A web form to upload new memes with:

- File upload for image

- Dropdown for category selection (faith, work\_life, friendships, children, relationships, going\_out)

- Text field for caption

- Text field for alt text

- Checkbox for active/inactive status

2. A page that lists all memes in a table showing:

- Thumbnail image

- Category

- Caption (truncated)

- Status (active/inactive)

- Edit and Delete buttons

3. Basic CRUD operations (Create, Read, Update, Delete)

4. File handling that:

- Validates image file types

- Saves images to a secure uploads folder

- Generates web-friendly filenames

Please include error handling, basic styling with Bootstrap, and security best practices. Keep the code simple since I'm a beginner programmer.

**Prompt 4: React Frontend Splash Component**

Create a React component for a meme splash page in a mobile finance app.

Requirements:

- Full-screen overlay that appears before dashboard

- Display meme image with caption

- "Continue to Dashboard" button

- Optional "Skip this feature" link

- Loading state with skeleton

- Error fallback if meme fails to load

- Smooth animations/transitions

- Mobile-first responsive design

- Accessibility features (alt text, keyboard navigation)

The component should:

- Fetch meme data from API endpoint /api/user-meme

- Track user interaction (view, skip, continue)

- Send analytics to /api/meme-analytics endpoint

- Auto-advance after 10 seconds if no interaction

- Work with existing app authentication

Use modern React hooks (useState, useEffect) and include TypeScript types. Style with Tailwind CSS classes. Include error boundaries and loading states. Make it production-ready but easy to understand for a beginner.

**Prompt 5: API Endpoints Integration**

Create Flask API endpoints for the meme splash page feature. I need:

1. GET /api/user-meme/<user\_id>

- Returns personalized meme for user

- Includes image URL, caption, category

- Handles user not found, no available memes

- Updates user\_meme\_history table

2. POST /api/meme-analytics

- Tracks user interactions (viewed, skipped, continued)

- Stores timestamp and user\_id

- Returns success/error response

3. GET /api/user-meme-preferences/<user\_id>

- Returns user's meme category preferences

- Shows skip rate and engagement stats

4. PUT /api/user-meme-preferences/<user\_id>

- Allows user to disable/enable meme feature

- Update category preferences

Include:

- Proper HTTP status codes

- JSON response format

- Error handling with descriptive messages

- Request validation

- Basic rate limiting

- CORS setup for frontend

- Authentication middleware (assume JWT token in headers)

Use Flask-RESTful or plain Flask. Include logging for debugging. Make it beginner-friendly with clear comments.

**Prompt 6: User Settings Integration**

Create a user settings page component that allows Mingus users to control their meme splash page experience.

Features needed:

1. Toggle to enable/disable daily memes completely

2. Checkboxes for each category (faith, work\_life, friendships, children, relationships, going\_out)

3. Frequency setting (every login, once per day, weekly)

4. "Preview memes" button to see sample content

5. Reset preferences button

The component should:

- Fetch current preferences from API

- Save changes with optimistic updates

- Show loading/saving states

- Include form validation

- Provide helpful descriptions for each setting

- Match existing Mingus app styling

Build this as a React component that integrates with the main settings page. Include proper error handling and user feedback. Use modern React patterns but keep code readable for a beginner. Include accessibility features like proper labels and keyboard navigation.

**Prompt 7: Testing Suite Setup**

Create a comprehensive testing suite for the meme splash page feature. I need:

1. Unit tests for the Python backend:

- Test meme selection algorithm

- Test database operations

- Test API endpoints

- Mock external dependencies

2. Frontend component tests:

- Test React splash page component

- Test user interactions (skip, continue)

- Test error states and loading

- Test accessibility features

3. Integration tests:

- Test full user flow from login to dashboard

- Test meme display and tracking

- Test user preferences saving

4. Performance tests:

- Test meme loading speed

- Test database query performance

- Test image optimization

Use pytest for Python tests and Jest/React Testing Library for frontend. Include test fixtures and mock data. Set up CI/CD automation with GitHub Actions. Make tests easy to run and understand for a beginner developer. Include examples of good test data and edge cases.

**Prompt 8: Analytics and Monitoring Setup**

Create an analytics system to track the success of the meme splash page feature. I need:

1. Event tracking for:

- Meme views by category

- Skip rates by user demographics

- Time spent viewing memes

- Conversion from meme to wellness check-in completion

- User preference changes over time

2. Dashboard queries to measure:

- Daily/weekly meme engagement rates

- Most/least popular meme categories

- User retention correlation with meme usage

- Performance metrics (load times, errors)

3. Automated alerts for:

- High skip rates (>70%)

- Technical errors

- Unusual usage patterns

Build this using Python with simple database queries and basic data visualization. Include CSV export functionality for deeper analysis. Create a simple admin dashboard with charts. Use libraries like matplotlib or plotly for visualizations. Keep it simple but production-ready. Include sample queries and reports that a non-technical person could understand.

**Prompt 9: Production Deployment Configuration**

Create deployment configuration for the meme splash page feature in a production environment. I need:

1. Docker configuration:

- Dockerfile for the Flask app

- Docker-compose for local development

- Environment variable management

2. Database migration scripts:

- Safe migration to add new tables

- Rollback procedures

- Data seeding for initial memes

3. Image storage setup:

- Configuration for AWS S3 or similar

- Image optimization pipeline

- CDN integration for fast loading

4. Monitoring and logging:

- Application logging setup

- Error tracking integration

- Performance monitoring

5. Security considerations:

- File upload validation

- SQL injection prevention

- Rate limiting configuration

Include step-by-step deployment instructions for a cloud platform like Heroku or AWS. Make it beginner-friendly with clear explanations of each step. Include troubleshooting guides for common issues. Focus on security and performance best practices while keeping complexity manageable.

**Usage Instructions:**

1. **Start with Prompt 1** to set up your database structure
2. **Use Prompts 2-3** to build the backend functionality
3. **Apply Prompt 4** for the frontend component
4. **Implement Prompt 5** for API integration
5. **Add Prompt 6** for user control features
6. **Use Prompts 7-9** for testing, analytics, and deployment

Each prompt is designed to be self-contained but builds on previous work. Paste them into Cursor one at a time and iterate on the results before moving to the next prompt.