Hobby Hubby Forum - Build Blueprint & Implementation Prompts

Overview
This document contains a step-by-step blueprint for building the Hobby Hubby forum platform, broken down into small, testable chunks that build incrementally. Each prompt is designed for a code-generation LLM to implement in a test-driven manner.

Architecture Overview
- **Backend**: Django with PostgreSQL
- **Frontend**: Django templates with responsive design
- **Authentication**: Django auth + Google OAuth
- **File Storage**: Local storage initially, scalable to S3
- **Deployment**: Heroku-ready configuration

Build Phases

Phase 1: Foundation Setup

Prompt 1.1 - Django Project Initialization
```text
Create a new Django project called 'hobby_hubby' with proper structure and configuration. Set up:
1. Django project with settings split into base/development/production
2. PostgreSQL database configuration using environment variables
3. Basic .gitignore file for Python/Django projects
4. Requirements files (base.txt, development.txt, production.txt)
5. README.md with initial setup instructions
6. pytest configuration for testing
7. Create a simple test to verify Django is properly configured

Test requirements:
- Test that Django settings load correctly
- Test that database connection works
- Test that static files are configured properly

Use Django 4.2 LTS and include essential packages like python-decouple for environment variables.
```

Prompt 1.2 - Core Django Apps Structure
```text
Create the core Django apps structure for hobby_hubby:
1. Create 'accounts' app for user management
2. Create 'forums' app for forum functionality
3. Create 'core' app for shared utilities and base models
4. Set up proper app configuration in settings
5. Create base model mixins in core app (TimestampedModel with created_at, updated_at)
6. Configure Django admin basic setup

Test requirements:
- Test that all apps are properly registered
- Test TimestampedModel mixin functionality
- Test Django admin is accessible
```

Include proper __init__.py files and app configs.
```

### Phase 2: User System Foundation

#### Prompt 2.1 - Custom User Model
```text
Implement a custom User model extending Django's AbstractUser:
1. Create CustomUser model in accounts app with fields:
   - display_name (required, max 50 chars)
   - location (optional, max 100 chars)
   - bio (optional, text field)
   - is_email_verified (boolean, default False)
   - profile_picture (ImageField, optional)
2. Create and run migrations
3. Update settings to use custom user model
4. Create basic UserAdmin for Django admin

Test requirements:
- Test user creation with required fields
- Test user creation with all fields
- Test email uniqueness constraint
- Test user string representation
- Test admin can create/edit users

Include proper validation and help texts.
```

Prompt 2.2 - User Registration System
```text
Implement user registration with email verification:
1. Create registration form with fields: email, password, password_confirm, display_name
2. Create registration view that creates inactive user
3. Implement email verification token generation (using Django's built-in tokens)
4. Create email verification view that activates user
5. Create basic templates for registration and verification email
6. Add URL patterns for registration flow

Test requirements:
- Test registration form validation (matching passwords, unique email)
- Test user creation creates inactive user
- Test verification token generation and validation
- Test email verification activates user
- Test invalid token handling
- Test already verified user handling

Use Django's built-in password validators.
```

Prompt 2.3 - Login and Authentication
```text
Implement login system with session management:
1. Create custom login form with email instead of username
2. Create login view with "remember me" functionality

3. Create logout view
4. Create password reset flow using Django's built-in views
5. Create login_required middleware/decorators
6. Create basic templates for all auth pages

Test requirements:
- Test login with valid credentials
- Test login with invalid credentials
- Test login with unverified email
- Test "remember me" extends session
- Test logout clears session
- Test password reset email generation
- Test login_required redirects properly

Style templates with basic Bootstrap 5 for responsive design.
```

Phase 3: Forum Structure

Prompt 3.1 - Category and Subcategory Models
```text
Create forum category structure:
1. Create Category model with fields:
   - name (unique, max 100 chars)
   - slug (unique, auto-generated)
   - description (text)
   - color_theme (CharField with choices)
   - icon (optional, for future use)
   - order (IntegerField for sorting)
2. Create Subcategory model with fields:
   - category (ForeignKey)
   - name (max 100 chars)
   - slug (unique with category)
   - description (text)
   - member_count (IntegerField, default 0)
3. Create migrations and admin interface

Test requirements:
- Test category creation and slug generation
- Test subcategory unique constraint within category
- Test cascade deletion
- Test ordering works correctly
- Test member count default value

Include Meta classes for proper ordering and unique constraints.
```

Prompt 3.2 - Thread and Post Models
```text
Implement forum thread and post structure:
1. Create Thread model:
   - subcategory (ForeignKey)
   - author (ForeignKey to User)
   - title (max 200 chars)

   - slug (auto-generated)
   - is_pinned (boolean)
   - is_locked (boolean)
   - view_count (IntegerField, default 0)
   - last_post_at (DateTimeField)
2. Create Post model:
   - thread (ForeignKey)
   - author (ForeignKey to User)
   - content (TextField)
   - is_edited (boolean)
   - edited_at (DateTimeField, nullable)
3. Add post_count to Thread model (denormalized for performance)

Test requirements:
- Test thread creation and slug generation
- Test post creation updates thread's last_post_at
- Test cascade deletion
- Test thread locking prevents new posts
- Test edit tracking

Use Django signals to update denormalized fields.
```

Prompt 3.3 - Forum Views and URLs
```text
Create basic forum browsing views:
1. Create CategoryListView showing all categories with subcategories
2. Create SubcategoryDetailView showing threads (paginated)
3. Create ThreadDetailView showing posts (paginated)
4. Create URL patterns with slug-based routing
5. Create basic templates with Bootstrap styling
6. Implement view counting for threads

Test requirements:
- Test category list displays all categories
- Test subcategory view paginates threads
- Test thread view paginates posts
- Test view count increments on thread view
- Test 404 for invalid slugs
- Test pagination edge cases

Use Django's generic views where appropriate.
```

Phase 4: User Interactions

Prompt 4.1 - Thread and Post Creation
```text
Implement content creation functionality:
1. Create ThreadCreateView with form validation
2. Create PostCreateView for replies
3. Implement @login_required decorators
4. Create forms with proper validation
5. Add CSRF protection

6. Create success messages using Django messages framework
7. Add "Preview" functionality using AJAX

Test requirements:
- Test authenticated users can create threads
- Test unauthenticated users are redirected
- Test form validation (empty fields, length limits)
- Test successful creation redirects properly
- Test post creation updates thread last_post_at
- Test preview doesn't save to database

Include rate limiting preparation (structure, not implementation).
```

Prompt 4.2 - Upvoting System
```text
Implement upvoting for posts:
1. Create Vote model:
   - user (ForeignKey)
   - post (ForeignKey)
   - created_at (DateTimeField)
   - UniqueConstraint on user+post
2. Add vote_count to Post model (denormalized)
3. Create VoteView for AJAX voting
4. Add vote status to post display
5. Create JavaScript for async voting

Test requirements:
- Test user can vote on posts
- Test user cannot vote twice
- Test vote count updates correctly
- Test user can remove vote
- Test unauthenticated users cannot vote
- Test AJAX endpoint returns proper JSON

Use Django signals to update vote counts.
```

Prompt 4.3 - Bookmarking System
```text
Implement thread bookmarking:
1. Create Bookmark model:
   - user (ForeignKey)
   - thread (ForeignKey)
   - created_at (DateTimeField)
   - UniqueConstraint on user+thread
2. Create BookmarkView for AJAX bookmarking
3. Create UserBookmarksView to list bookmarks
4. Add bookmark status to thread display
5. Update user profile to show bookmarks

Test requirements:
- Test user can bookmark threads
- Test user cannot bookmark twice

- Test bookmark list shows user's bookmarks
- Test removing bookmarks works
- Test bookmark list is paginated
- Test privacy (users can't see others' bookmarks)

Include bookmark count on user profile.
```

Phase 5: User Profiles and Social Features

Prompt 5.1 - Enhanced User Profiles
```text
Enhance user profiles with hobbies and galleries:
1. Create UserHobby model (many-to-many through table):
   - user (ForeignKey)
   - subcategory (ForeignKey)
   - joined_at (DateTimeField)
2. Create UserProfileView with tabs
3. Implement hobby selection/management
4. Add post count and join date display
5. Create profile edit view
6. Style with responsive tabs

Test requirements:
- Test profile displays user information
- Test hobby list links to subcategories
- Test profile edit updates data
- Test profile picture upload
- Test other users can view profiles
- Test private information is hidden

Include proper image handling for profile pictures.
```

Prompt 5.2 - Photo Gallery
```text
Implement user photo galleries:
1. Create Photo model:
   - user (ForeignKey)
   - image (ImageField)
   - caption (CharField, optional)
   - uploaded_at (DateTimeField)
2. Create gallery view with grid layout
3. Implement photo upload with validation
4. Add image resizing using Pillow
5. Create lightbox for full-size viewing

Test requirements:
- Test photo upload with size limits
- Test image format validation
- Test automatic resizing works
- Test caption is optional
- Test gallery pagination
- Test delete photo functionality

Max 10MB upload, resize to 800x600 for gallery display.
```

Prompt 5.3 - Friend System
```text
Implement friend request system:
1. Create Friendship model:
   - from_user (ForeignKey)
   - to_user (ForeignKey)
   - status (choices: pending, accepted, rejected)
   - created_at (DateTimeField)
   - responded_at (DateTimeField, nullable)
2. Create friend request views
3. Create friend list view
4. Add friend status to profiles
5. Create notifications for friend requests

Test requirements:
- Test sending friend requests
- Test accepting/rejecting requests
- Test cannot send duplicate requests
- Test friend list shows accepted only
- Test friendship is bidirectional
- Test proper notifications are created

Include unique constraint to prevent duplicate requests.
```

Phase 6: Messaging System

Prompt 6.1 - Private Messaging Models
```text
Create private messaging system:
1. Create Conversation model:
   - participants (ManyToMany to User)
   - created_at (DateTimeField)
   - last_message_at (DateTimeField)
2. Create Message model:
   - conversation (ForeignKey)
   - sender (ForeignKey to User)
   - content (TextField)
   - sent_at (DateTimeField)
   - is_read (boolean)
3. Create ConversationParticipant through model with read tracking

Test requirements:
- Test conversation creation between users
- Test message sending
- Test read status tracking
- Test conversation ordering by last message
- Test participant limit (2 for private messages)

Plan for future group messaging support.

```

Prompt 6.2 - Messaging Interface
```text
Create messaging UI:
1. Create inbox view with conversation list
2. Create conversation detail view
3. Implement real-time updates using Django Channels (structure only)
4. Create message compose form
5. Add unread message counter
6. Create message notification system

Test requirements:
- Test inbox shows user's conversations
- Test sending messages updates conversation
- Test unread count is accurate
- Test marking messages as read
- Test friends-only messaging setting
- Test blocked users cannot message

Include pagination for long conversations.
```

Phase 7: Search and Discovery

Prompt 7.1 - Basic Search Implementation
```text
Implement search functionality:
1. Create SearchView with query processing
2. Implement full-text search using PostgreSQL
3. Create search indexes on key fields
4. Add search form to navigation
5. Create search results template with tabs
6. Implement search history tracking

Test requirements:
- Test searching posts by content
- Test searching threads by title
- Test searching users by display name
- Test searching categories/subcategories
- Test empty query handling
- Test SQL injection prevention

Use Django's PostgreSQL search features.
```

Prompt 7.2 - Advanced Search Filters
```text
Add search filtering and sorting:
1. Create SearchForm with filter fields:
   - Date range
   - Specific categories
   - Content type (posts, threads, users)
   - Sort options

2. Implement filter logic in SearchView
3. Create filter UI with collapsible sections
4. Add search result highlighting
5. Implement saved searches

Test requirements:
- Test date range filtering
- Test category filtering
- Test combining multiple filters
- Test sort options work correctly
- Test filter persistence in UI
- Test saved search functionality

Include faceted search counts.
```

Phase 8: Image Handling

Prompt 8.1 - Image Upload System
```text
Implement image handling for posts:
1. Create PostImage model:
   - post (ForeignKey)
   - image (ImageField)
   - uploaded_at (DateTimeField)
2. Update PostForm to handle multiple images
3. Implement image validation and resizing
4. Create image upload progress indicator
5. Add image display in posts

Test requirements:
- Test image upload with posts
- Test file size validation (10MB limit)
- Test format validation (JPEG, PNG, WebP, GIF)
- Test automatic resizing
- Test multiple image handling
- Test orphaned image cleanup

Resize to max 1920x1080, 85% quality.
```

Prompt 8.2 - Image Optimization
```text
Optimize image delivery:
1. Implement lazy loading for images
2. Create thumbnail generation
3. Add WebP conversion for supported browsers
4. Implement image CDN preparation
5. Create image deletion safeguards
6. Add EXIF data stripping for privacy

Test requirements:
- Test thumbnail generation
- Test WebP conversion

- Test EXIF removal
- Test lazy loading functionality
- Test CDN URL generation
- Test cascading deletion protection

Include fallbacks for unsupported formats.
```

Phase 9: Moderation System

Prompt 9.1 - Content Reporting
```text
Implement user reporting system:
1. Create Report model:
   - reporter (ForeignKey to User)
   - content_type (Generic relation)
   - object_id (Generic relation)
   - reason (choices)
   - description (TextField)
   - status (pending, reviewed, resolved)
   - created_at (DateTimeField)
2. Create reporting forms and views
3. Add report buttons to posts/profiles
4. Create moderation queue view

Test requirements:
- Test reporting posts
- Test reporting users
- Test report reason validation
- Test duplicate report handling
- Test report queue ordering
- Test anonymous reporting option

Include rate limiting for report spam.
```

Prompt 9.2 - Moderation Tools
```text
Create admin moderation interface:
1. Create ModerationAction model:
   - moderator (ForeignKey to User)
   - action_type (warning, ban, delete)
   - target_user (ForeignKey, nullable)
   - target_content (Generic relation)
   - reason (TextField)
   - duration (for temporary bans)
2. Create moderation dashboard
3. Implement content editing/deletion
4. Create user warning system
5. Implement ban functionality

Test requirements:
- Test content deletion
- Test user warnings

- Test temporary bans
- Test permanent bans
- Test ban expiration
- Test moderation logging

Include audit trail for all actions.
```

Phase 10: Notifications

Prompt 10.1 - Notification System
```text
Implement in-app notifications:
1. Create Notification model:
   - recipient (ForeignKey to User)
   - type (choices: reply, mention, friend_request, etc.)
   - related_object (Generic relation)
   - message (TextField)
   - is_read (boolean)
   - created_at (DateTimeField)
2. Create notification creation signals
3. Create notification center view
4. Add notification badge to navigation
5. Implement mark as read functionality

Test requirements:
- Test notification creation on replies
- Test notification creation on friend requests
- Test notification list view
- Test mark as read
- Test mark all as read
- Test notification deletion

Include notification preferences per user.
```

Phase 11: Final Integration

Prompt 11.1 - Feature Integration Testing
```text
Perform comprehensive integration testing:
1. Create end-to-end test scenarios
2. Test complete user journeys
3. Verify all features work together
4. Test edge cases and error handling
5. Security testing

Test requirements:
- Test complete registration to posting flow
- Test friend system with messaging
- Test moderation workflow
- Test search with filters
- Test mobile responsiveness
- Test concurrent user scenarios

Document any discovered issues.
```

Prompt 11.2 - Documentation
```text
Finalize documentation:
1. Create user documentation
2. Write admin guide
3. Create API documentation

Include troubleshooting guides.
```

Implementation Notes

Each prompt should:
1. Start with understanding existing code
2. Write tests first (TDD approach)
3. Implement minimal code to pass tests
4. Refactor for clarity and performance
5. Update documentation
6. Ensure integration with previous steps

Key Principles
- No orphaned code - everything connects
- Incremental complexity - start simple
- Write tests around core functionality only - comprehensiveness can be a later priority.
- Security first - validate all inputs
- Performance aware - optimize queries
- Mobile first - responsive by default
```