# Implementation Plan

- [ ] 1. Set up project structure and core data models
  - Create modular directory structure for components (data, ui, business_logic, models)
  - Implement data classes for ClientBrief, MediaPlan, FormatAllocation, RateCard, and SiteData
  - Create base configuration management for API keys and settings
  - *Requirements: 1.1, 5.1*
- [ ] 2. Implement data parsing and management system
- [ ] 2.1 Create RateCardParser for Excel file processing
  - Write parser to extract APX Impact and Reach pricing from Excel sheets
  - Implement data validation and error handling for malformed files
  - Create unit tests for various Excel file formats and edge cases
  - *Requirements: 5.1, 5.3*
- [ ] 2.2 Create SiteListParser for site categorization data
  - Write parser to extract market-specific site data from Excel files
  - Implement site grouping by format and category functionality
  - Create validation for site data completeness and accuracy
  - *Requirements: 5.2, 1.3*
- [ ] 2.3 Implement DataManager for centralized data access
  - Create caching mechanism for rate cards and site lists
  - Implement data freshness validation and update notifications
  - Write methods for market-specific data retrieval
  - Create unit tests for data loading and caching functionality
  - *Requirements: 5.1, 5.2, 5.4*
- [ ] 3. Build core Streamlit user interface
- [ ] 3.1 Create MediaPlannerForm component
  - Implement form fields for brand name, budget, country, campaign period, and objective
  - Add real-time budget validation with positive number constraints
  - Create dynamic country selection with available markets
  - Implement planning mode toggle (AI vs Manual selection)
  - *Requirements: 1.1, 1.2, 1.4, 2.1*
- [ ] 3.2 Add manual format selection functionality
  - Display available ad products with rate card information when manual mode selected
  - Implement multi-select interface for format selection
  - Create real-time budget allocation preview for selected formats
  - Add validation to ensure selections fit within budget constraints

  - *Requirements: 2.3, 2.4*

- [ ] 3.3 Implement form validation and error handling
  - Create comprehensive input validation with specific error messages
  - Add required field validation with clear user feedback
  - Implement budget range suggestions for insufficient budget scenarios
  - Create user-friendly error displays for data loading issues
  - *Requirements: 1.4, 3.4*

- [ ] 4. Develop AI plan generation system

- [ ] 4.1 Create AIPlanGenerator class
  - Implement OpenAI API integration with proper error handling
  - Create dynamic system prompt generation based on market data and constraints
  - Write method to generate exactly 3 distinct media plan options
  - Implement plan diversity algorithms to ensure strategic differences
  - *Requirements: 3.1, 3.2, 6.1, 6.4*

- [ ] 4.2 Implement budget optimization and allocation logic
  - Create algorithms for budget distribution across selected formats
  - Implement reach optimization and frequency capping considerations
  - Add logic for diverse media mix when budget allows
  - Create high-impact placement prioritization for limited budgets
  - *Requirements: 6.1, 6.2, 6.3*

- [ ] 4.3 Add plan parsing and validation
  - Implement parser to extract structured data from AI-generated plans
  - Create validation to ensure plans stay within specified budget
  - Add calculation verification for impressions and reach estimates
  - Implement error handling for malformed AI responses
  - *Requirements: 3.2, 4.1*

- [ ] 5. Build plan display and comparison system

- [ ] 5.1 Create PlanDisplayComponent for plan visualization
  - Implement side-by-side plan comparison interface
  - Create detailed breakdown display showing sites, products, costs, and allocations
  - Add estimated impressions, reach, and frequency display where available
  - Implement expandable plan details with comprehensive information
  - *Requirements: 4.1, 4.2, 4.3*

- [ ] 5.2 Add interactive budget breakdown and charts
  - Create visual budget allocation charts for each plan
  - Implement interactive elements to explore plan details
  - Add comparison highlighting to show key differences between options

    - Create summary statistics and performance indicators
    - *Requirements: 4.1, 4.3*

- [ ] 5.3 Implement plan export functionality

    - Create CSV export functionality for selected plans
    - Add PDF report generation with formatted plan details
    - Implement save functionality for plan persistence
    - Create export validation and error handling
    - *Requirements: 4.4*

- [ ] 6. Integrate and test complete workflow

- [ ] 6.1 Create MediaPlanController orchestration

    - Implement main workflow coordination between components
    - Create input validation and sanitization methods
    - Add plan comparison and ranking logic
    - Write integration tests for complete user journeys
    - *Requirements: 1.1, 2.1, 3.1*

- [ ] 6.2 Add comprehensive error handling and user feedback

    - Implement graceful handling of API failures with retry mechanisms
    - Create user notifications for rate limiting and network issues
    - Add clear messaging for unsupported markets and missing data
    - Create fallback mechanisms for offline scenarios
    - *Requirements: 5.3, 5.4*

- [ ] 6.3 Implement end-to-end testing and validation

    - Create automated tests for complete media planning workflows
    - Test with real rate card and site list files from Adzymic
    - Validate plan quality and accuracy with sample campaigns
    - Perform cross-browser compatibility testing for Streamlit interface
    - *Requirements: All requirements validation*

- [ ] 7. Implement model training and fine-tuning capabilities

- [ ] 7.1 Create ModelTrainingManager for data collection

    - Implement system to collect and store historical campaign briefs and successful plans
    - Create data quality validation and formatting for OpenAI fine-tuning format
    - Add functionality to export training data in required JSON format
    - Write validation to ensure training data meets OpenAI requirements
    - *Requirements: 6.1, 6.4*

- [ ] 7.2 Add fine-tuning job management

    - Implement OpenAI fine-tuning job initiation and monitoring
    - Create progress tracking and status reporting for training jobs

  - Add functionality to deploy and switch to fine-tuned models
  - Implement A/B testing framework to compare base vs fine-tuned model performance
  - *Requirements: 6.1, 6.4*

- [ ] 7.3 Integrate fine-tuned model usage
  - Update AIPlanGenerator to support both base and fine-tuned models
  - Implement model selection logic based on availability and performance
  - Add cost tracking and optimization for fine-tuned model usage
  - Create performance monitoring and quality assessment tools
  - *Requirements: 6.1, 6.4*