# **AI-Powered User Journey Analytics Tool**

## **📌 Overview**

This project is a modern web application that enables teams to **visually design, capture, and analyze user journeys** for mobile and web applications. By leveraging **AI**, it automatically generates **analytics event tracking specifications** from user-drawn flows, streamlining the process from **UX design to analytics implementation**.

## **🚀 Key Features**

### **1. Visual Journey Design**

* **Drawing Canvas**: Create multi-screen flows using rectangles, arrows, and annotations.
* **Screen Linking**: Connect screens to represent navigation and data flow.
* **UI Element Markup**: Highlight interactive elements (buttons, banners, tabs) for analysis.

### **2. Screenshot Capture & Preview**

* **Journey Screenshot**: Capture the entire user journey diagram as an image.
* **Preview Modal**: Review screenshots, add custom analysis instructions, and trigger AI analysis.

### **3. AI-Powered Analysis**

* **Event Detection**: AI identifies all interactive elements and possible user actions on each screen.
* **Property Classification**: Classifies event properties as:  
  + *On-screen*
  + *Carried-forward*
  + *Global*
* **Custom Prompts**: Add specific instructions to tailor the analysis (e.g., focus on a particular screen or flow).
* **Confidence Scoring**: Each event and property is assigned a confidence score.

### **4. Analytics Specification Generation**

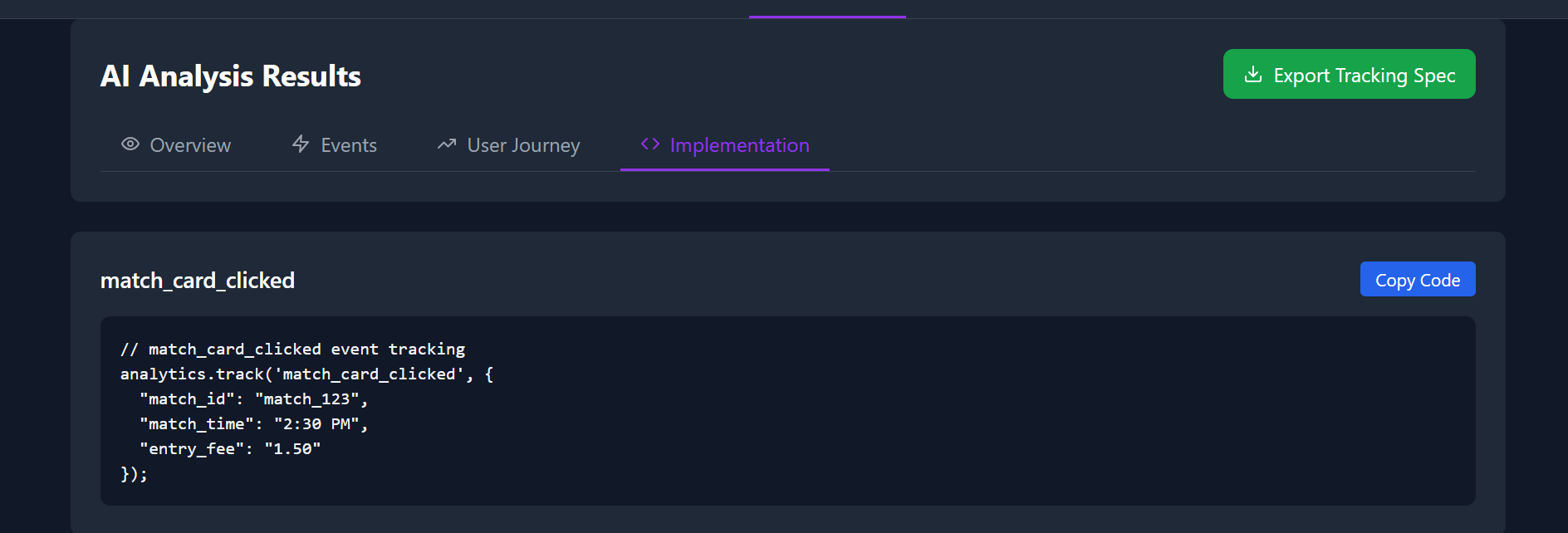
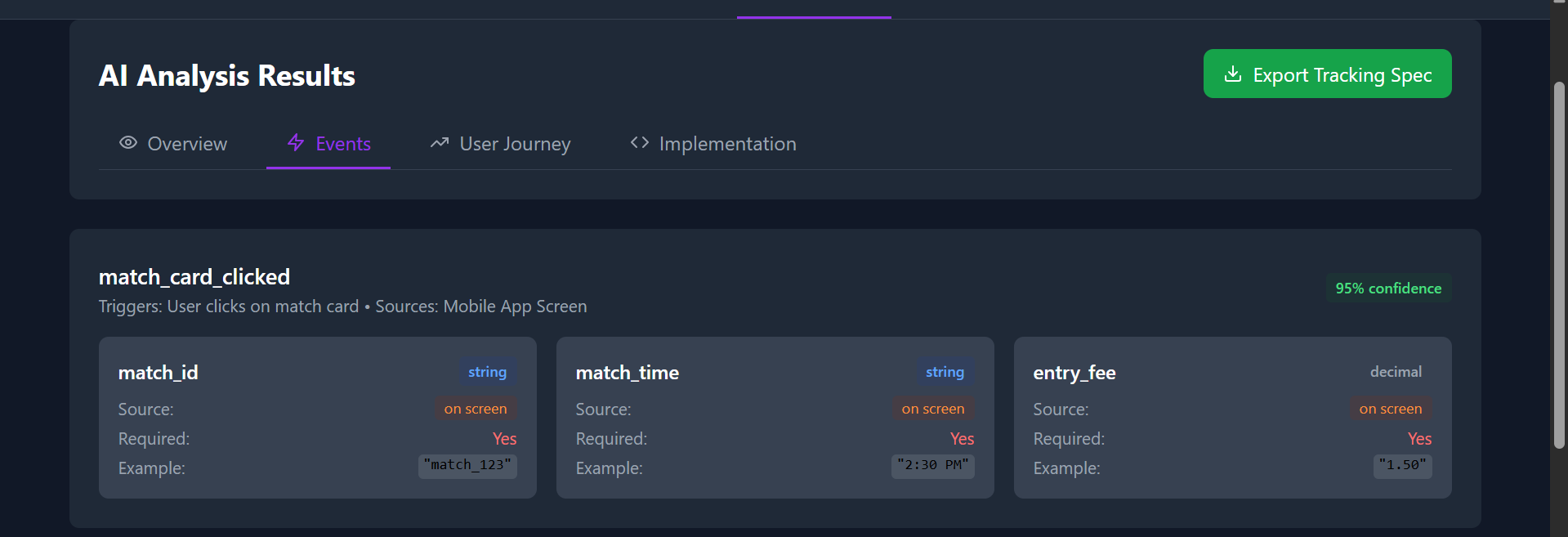
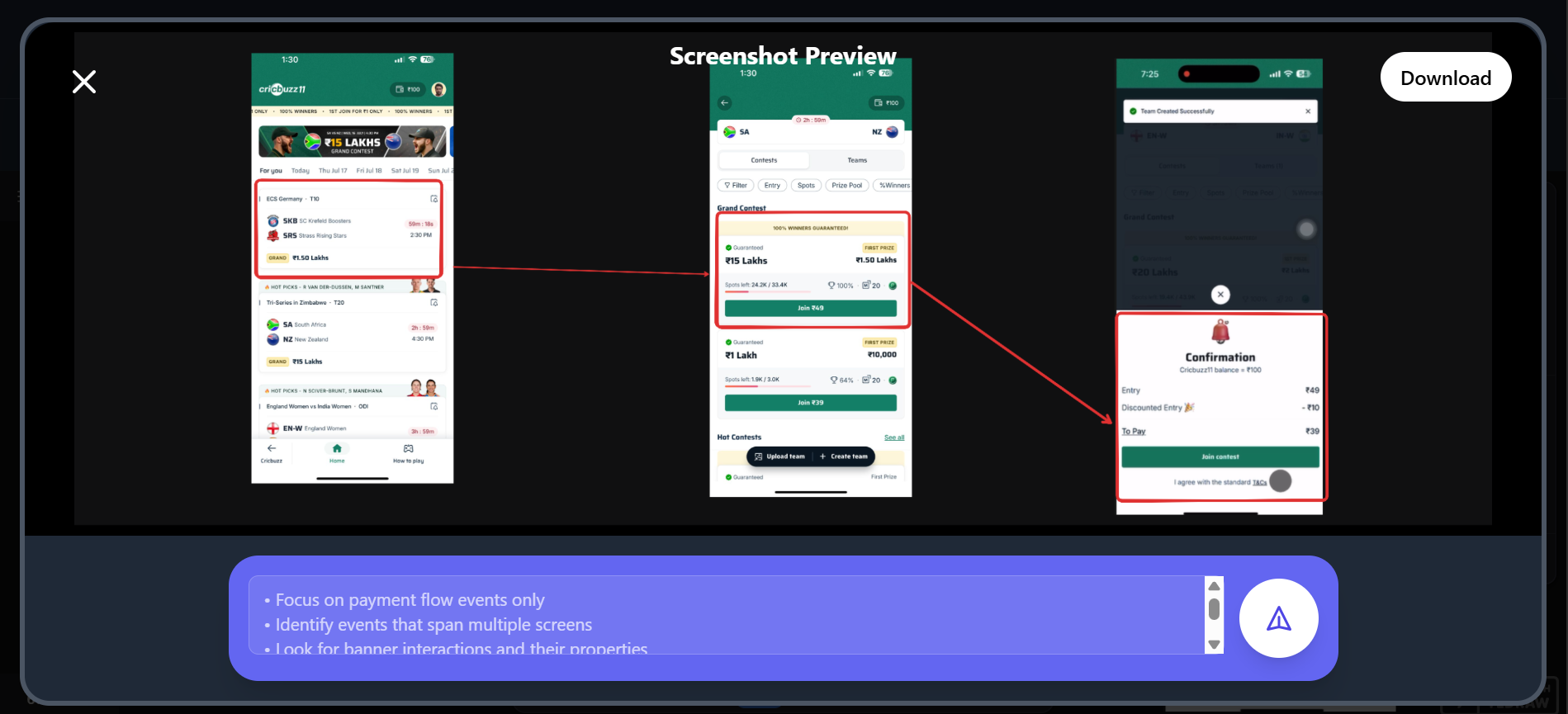
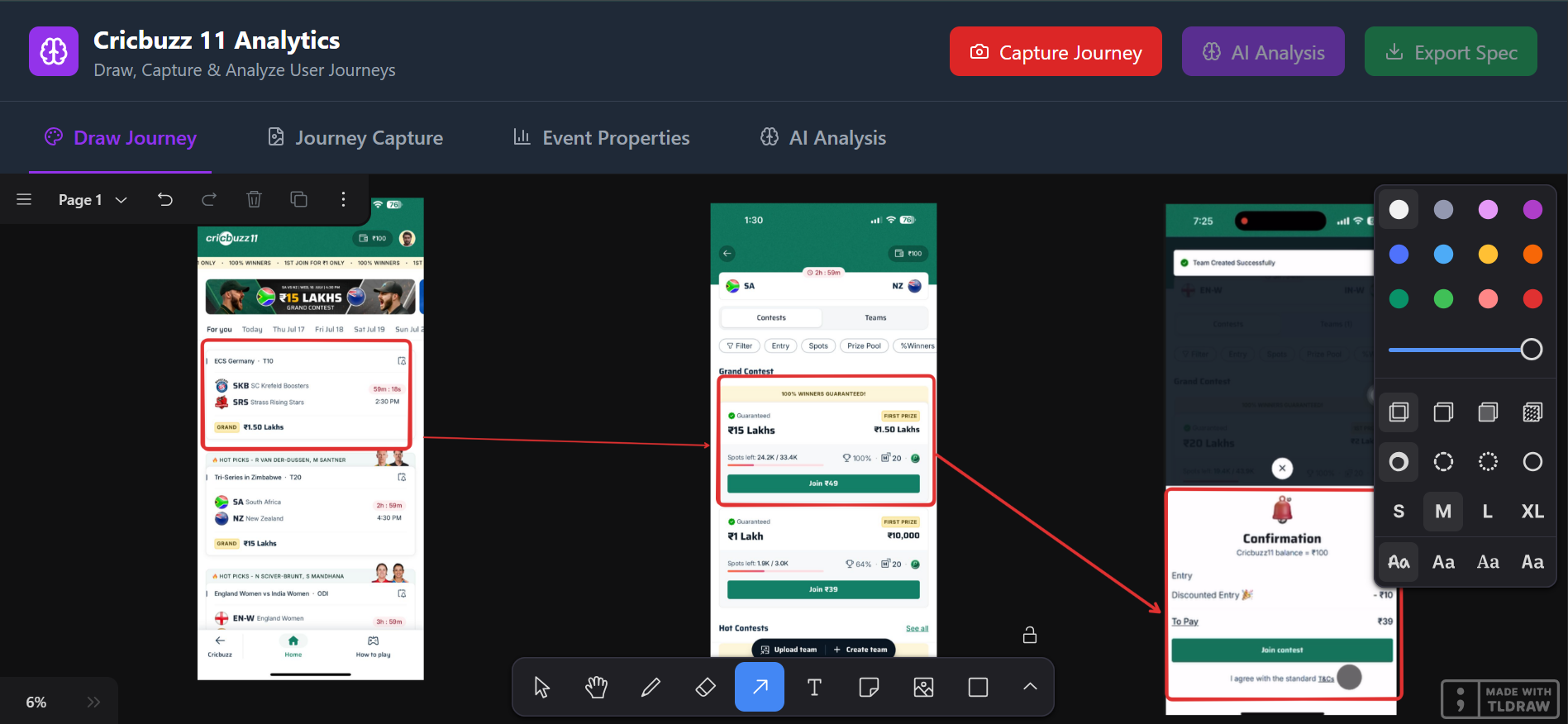
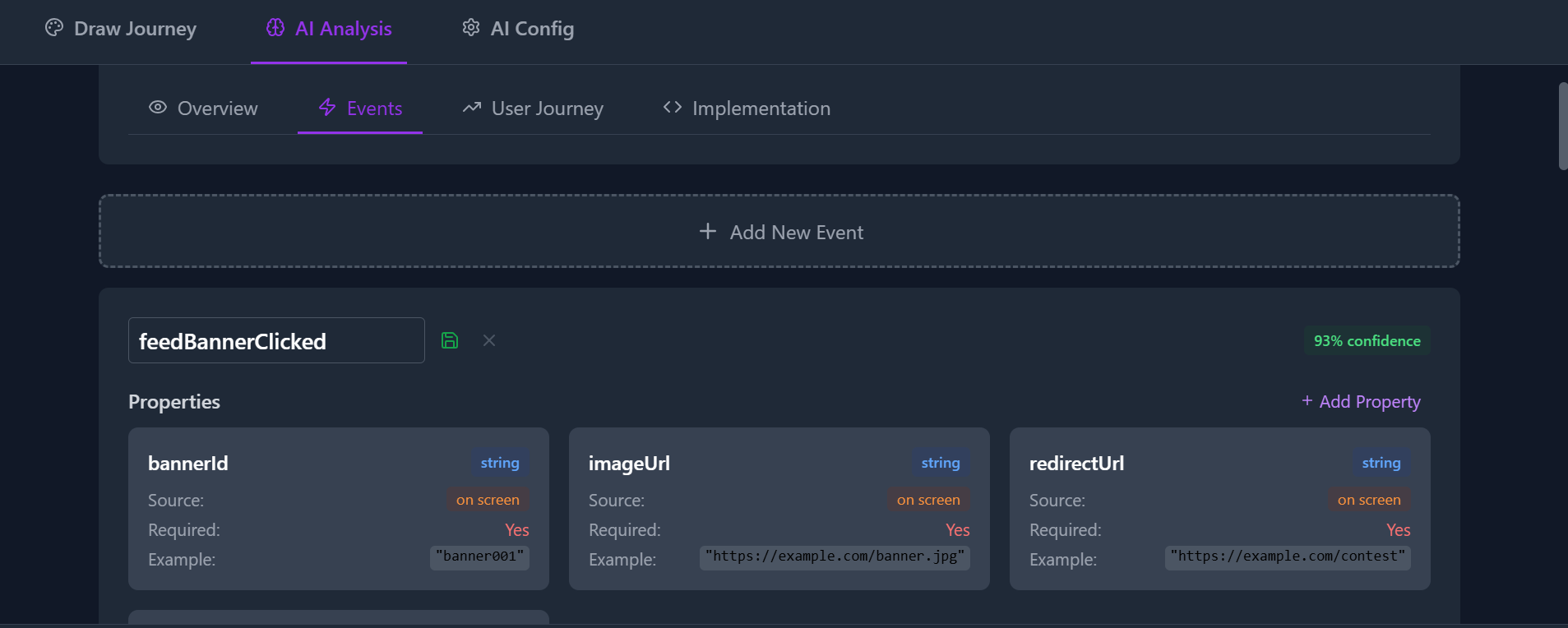
* **Event List**: Automatically generates a list of analytics events with names, triggers, and property details.
* **Property Types**: Specifies data types (string, number, boolean, decimal) for each property.
* **Recommendations**: AI provides suggestions for improving analytics coverage and event design.

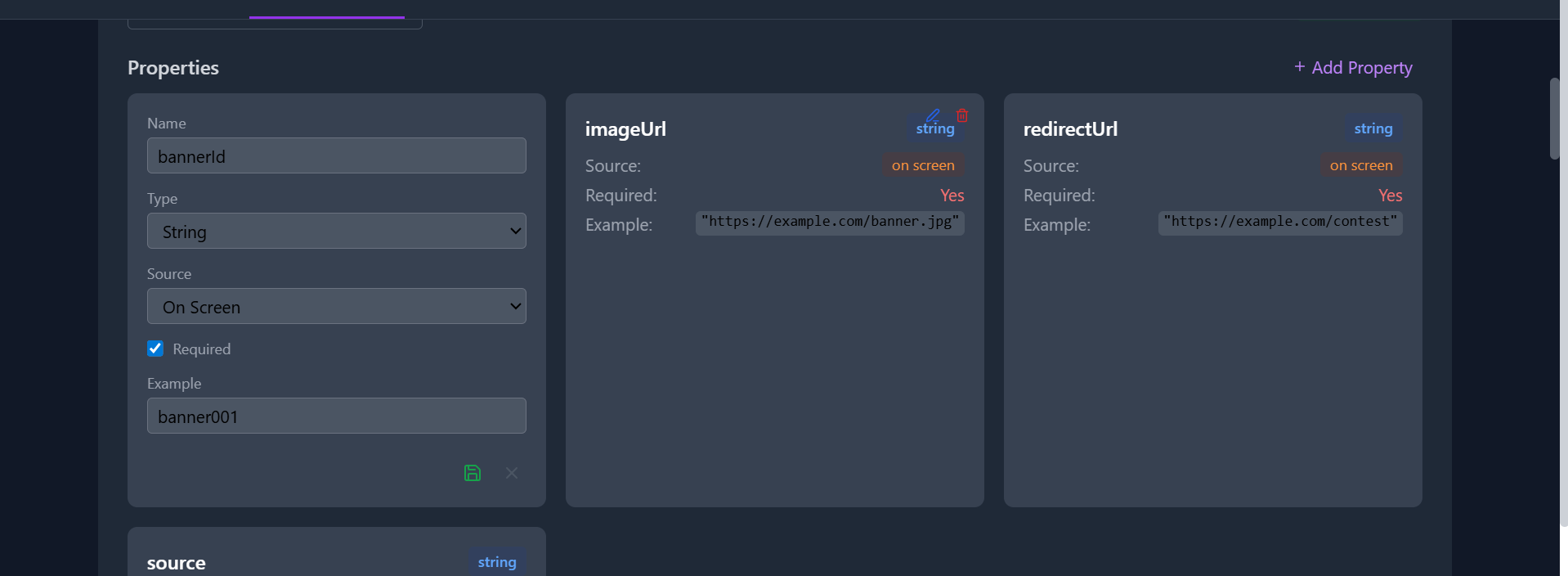
### **4. Editable events and properties**

* **Add / Edit / Delete**: Easily add missing events or make changes in events genrated or delete unnecessary events using the GUI.

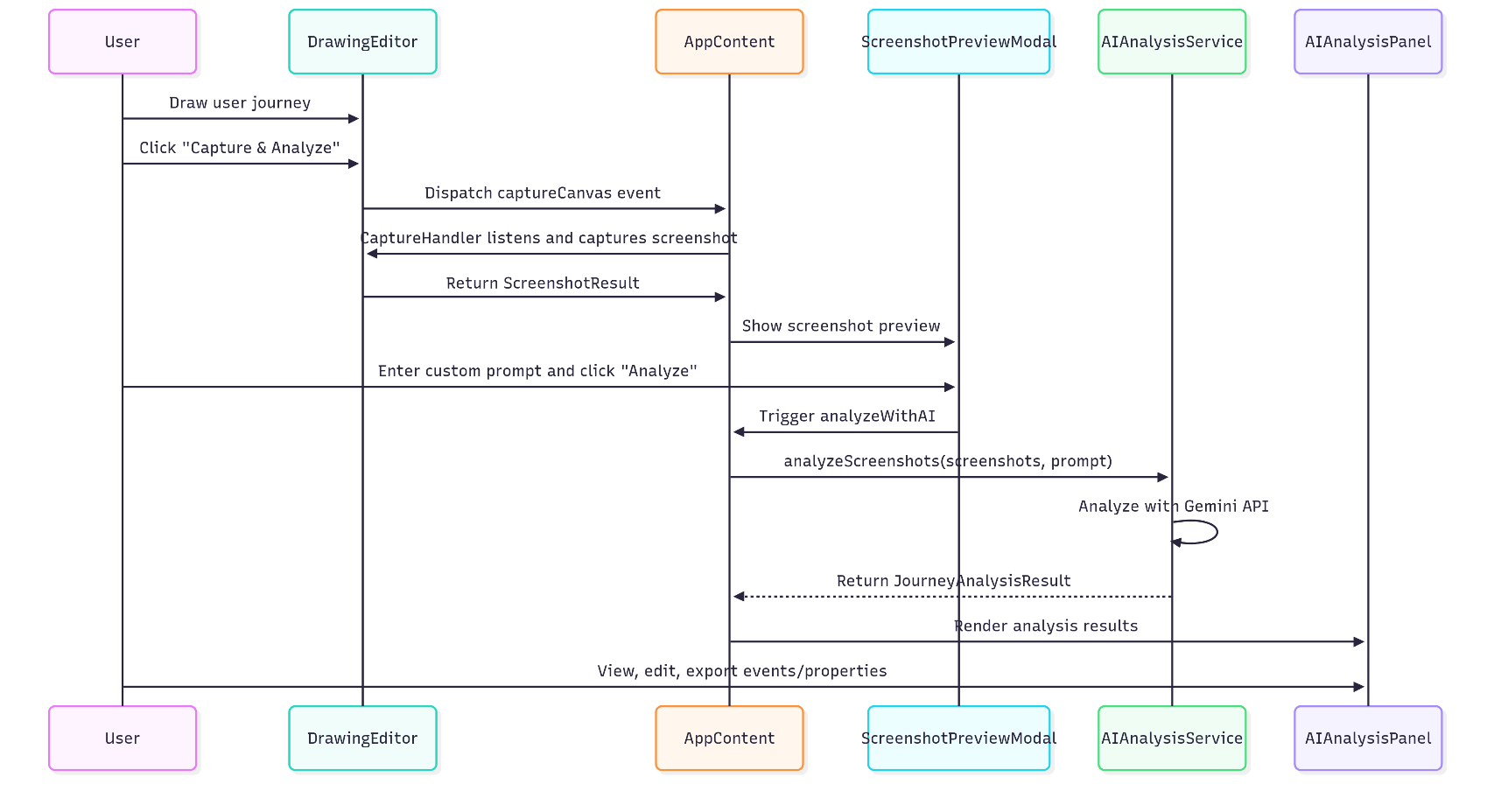
### **5. Export & Integration**

* **Export Formats**: Download event specifications as **CSV**.
* **Implementation Code**: Generate code snippets for seamless analytics SDK integration

**Website Preview  
  
**

****

***Sequence Diagram***



## ***Code Snippets -***

## ***1.*** *AI model selection and configuration*

| **const handleProviderChange = (provider: 'openai' | 'gemini') => {  updateConfig({  selectedProvider: provider  }); };  const testConnectionForProvider = async (provider: 'openai' | 'gemini') => {  if (!config.apiKeys[provider]) {  alert(`Please enter a valid ${provider} API key first`);  return;  }   setIsTestingConnection(prev => ({ ...prev, [provider]: true }));    try {  const result = await testConnection(provider);  if (result) {  alert(`✅ ${provider} connection test successful!`);  } else {  alert(`❌ ${provider} connection test failed. Please check your API key.`);  }  } catch (error) {  console.error(`${provider} test failed:`, error);  } finally {  setIsTestingConnection(prev => ({ ...prev, [provider]: false }));  } };** |
| --- |

## 

*2. Analyze a screenshot with a custom prompt*

| const aiService = new AIAnalysisService();  const result = await aiService.analyzeScreenshots(  [screenshot], // Array of ScreenshotResult  "Only analyze the payment confirmation screen." ); console.log(result.events); |
| --- |

*3. Event Structure*

| interface AIEventResponse {  name: string;  description: string;  screen: string;  category: string;  business\_value: string;  properties: AIEventProperty[];  mandatory: string[];  optional: string[];  triggers: string[];  similar\_events\_in\_other\_domains?: string[];  // Legacy support  eventName?: string;  eventType?: string;  element?: string;  selector?: string;  additionalProperties?: Record<string, string | number | boolean>; } |
| --- |

*4. Analyze with AI*

| private async analyzeWithGemini(screenshots: ScreenshotData[], customPrompt?: string, model?: string, apiKey?: string): Promise<JourneyAnalysisResult> {  // Using basic prompt  const basePrompt = this.buildAnalysisPrompt();    const selectedModel = model || 'gemini-1.5-flash';  const geminiKey = apiKey || '';    console.log('🧠 Using standard Gemini analysis');  console.log('Using Gemini model:', selectedModel);  console.log('API Key (first 10 chars):', geminiKey.substring(0, 10) + '...');   if (!geminiKey) {  throw new Error('Gemini API key not configured');  }   try {  const genAI = new GoogleGenerativeAI(geminiKey);  const generativeModel = genAI.getGenerativeModel({ model: selectedModel });   // Convert screenshots to inline data format for Gemini  const imageParts = screenshots.map(screenshot => ({  inlineData: {  data: screenshot.dataUrl.split(',')[1], // Remove data:image/png;base64, prefix  mimeType: "image/png"  }  }));   const textPrompt = `${basePrompt}    Analyze these ${screenshots.length} mobile app screenshots for comprehensive event tracking specification.  ANALYSIS REQUIREMENTS:  1. SCREEN-BY-SCREEN EVENT DETECTION:  - Identify ALL interactive elements on each screen  - Detect events like: bannerClicked, tabSwitched, seeAllClicked, dateScrolled  - Look for navigation elements, form inputs, buttons, cards, lists  2. PROPERTY SOURCE CLASSIFICATION:  - ON-SCREEN: Values visible to user (entryFee, contestType, bannerId)  - CARRIED-FORWARD: Data from previous screens (roundId, matchId from earlier selection)  - GLOBAL: Session data (userId, balances, platform)  3. CONNECTED SCREEN ANALYSIS:  - If screens show connected flow (arrows/lines), identify carried properties  - Example: Round selected on Screen 1 → roundId carried to Screen 2 contests  4. DATA TYPE PRECISION:  - Use 'number' for monetary values, counts, IDs that need math operations  - Use 'string' for text, status values, non-numeric IDs  - Use 'boolean' for true/false flags  - Use 'decimal' for precise monetary calculations  5. EVENT NAMING:  - Use camelCase format  - Be specific but reusable (feedBannerClicked vs generic bannerClicked)  - Include context when needed (homeWalletClicked vs contestWalletClicked)  Focus on fantasy sports app context: contests, matches, teams, tournaments, wallet, etc. Ensure each event has complete property set for meaningful analytics. ${customPrompt ? `\n\*\*SPECIFIC REQUIREMENTS:\*\*\n${customPrompt}\n` : ''} Follow user requirements and provide detailed event tracking specifications. `;      const result = await generativeModel.generateContent([textPrompt, ...imageParts]);  const response = await result.response;  const analysisText = response.text();    console.log('Gemini API Success Response');  console.log('Gemini Analysis Text:', analysisText);  console.log('Analysis text length:', analysisText.length);    return this.parseAIResponse(analysisText, screenshots);  } catch (error) {  console.error('Gemini API request failed:', error);  throw error;  }  } |
| --- |

## **⚙️ How It Works**

1. **Draw**: Use the canvas to design the user journey, marking screens and interactions.
2. **Capture**: Take a screenshot of the journey flow.
3. **Customize** *(Optional)*: Add custom instructions in the prompt box to guide the AI analysis.
4. **Analyze**: Trigger AI analysis. The tool combines static and custom prompts for precise results.
5. **Review**: View detected events, properties, and recommendations in the analytics dashboard and make changes if necessary.
6. **Export**: Download the tracking specification or implementation code for integration.

🔄 **System Architecture Overview**

### **Core Application Flow**

#### **Entry Point & Setup -**

* **main.tsx** serves as the entry point of the application. It bootstraps the React app and renders the root component.
* **App.tsx** is the main orchestrator. It manages tab navigation and the overall application state.  
   It wraps the app with multiple context providers to handle global state management.

### **Drawing & Input Capture**

* **DrawingEditor.tsx** provides a canvas using TLDraw for users to sketch wireframes.  
   - Users can draw UI mockups, user flows, or upload existing images.
* When the user clicks the "Analyze" button, the current canvas or uploaded image is captured as a screenshot.
* If no drawing exists, it falls back to a demo screenshot for processing.

### **Screenshot Processing**

* **visionService.ts** handles image data processing.  
   It converts the captured canvas or image into a base64 format suitable for API use.
* It prepares the screenshot data and sends it to the AI engine for further analysis.

### **AI Analysis Engine**

* **aiService.ts** is the core logic module that communicates with AI APIs.  
   It supports both **OpenAI GPT-4 Vision** and **Google Gemini** APIs.
* It formulates detailed prompts asking the AI to:  
  + Identify UI elements in the screenshot
  + Generate event tracking specifications
* It parses the AI’s response and converts it into structured, usable event data.

### **Configuration Management**

* **AIConfigContext.tsx** manages AI settings, including:  
  + API keys
  + Selected AI model/provider (OpenAI or Gemini)
* **AIConfigPanel.tsx** provides a user interface to:  
  + Enter and test API keys
  + Switch between different AI providers

### **Results Display & Interaction**

* **AIAnalysisPanel.tsx** is an interactive dashboard where users can:  
  + View AI-generated events
  + Edit event names, add or remove properties
  + Modify or enhance event specifications
* Multiple views are supported:  
  + Overview
  + Events
  + Implementation Code

### **Export & Output**

* The application allows users to:  
  + Export tracking specifications as **CSV** files
  + Generate **code snippets** for implementation

## 

## **🛠 Technology Stack**

| **Layer** | **Tools Used** |
| --- | --- |
| **Frontend** | React 18, TypeScript, Tailwind CSS |
| **Canvas** | tldraw |
| **AI** | OpenAI, Gemini, Claude *(configurable)* |
| **State** | React Context API |
| **Bundler** | Vite |

## **🧩 Customization & Extensibility**

* **AI Provider Selection**: Easily configure and switch between supported AI providers.
* **Prompt Customization**: Provide detailed instructions per analysis session.
* **Component-Based Architecture**: Modular design makes it easy to extend/replace UI components and backend services.

## **🎯 Typical Use Cases**

* **Product Analytics Teams**: Rapidly generate event tracking plans from UX diagrams.
* **QA & Testing**: Validate that all user actions are traceable via analytics.
* **A/B Testing**: Document and compare user journeys and event coverage.
* **Cross-Platform Consistency**: Ensure analytics events are consistent across web, mobile, and tablet flows.

## **🏁 Getting Started**

### **1. Install dependencies:**

npm i

### **2. Start development server:**

npm run dev

### **3. Configure AI API keys:**

Use the **in-app configuration panel** to input your AI provider keys.

### 

### **4. Draw and analyze:**

Use the drawing tools to design your user flow, capture it, and analyze it using AI.