USSD Flow Editor - Developer Guide

Table of Contents

Getting Started

Prerequisites

- **Node.js**: Version 18.0 or higher
- **npm**: Version 8.0 or higher
- **Git**: For version control and maker-checker workflow
- **VS Code**: Recommended IDE with React extensions

Quick Start

```
# Clone the repository
git clone <repository-url>
cd ussd-editor

# Install dependencies
npm install

# Start development server
npm run dev

# Open browser to http://localhost:5173
```

Project Structure Overview

```
ussd-editor/
├── public/                  # Static assets
├── src/                    # Source code
│   ├── components/         # React components
│   ├── utils/             # Utility functions
│   ├── styles/            # CSS files
│   └── assets/            # Images, icons
├── docs/                  # Documentation
```

```
├── load-testing/         # K6 test files
├── workflow-data/        # Git workflow storage
├── package.json          # Dependencies and scripts
├── vite.config.js        # Build configuration
└── docker-compose.yml    # Container deployment
```

Development Setup

Environment Configuration

VS Code Extensions (Recommended)

```
// .vscode/extensions.json
{
  "recommendations": [
    "bradlc.vscode-tailwindcss",
    "esbenp.prettier-vscode",
    "dbaeumer.vscode-eslint",
    "ms-vscode.vscode-react",
    "ms-vscode.vscode-json"
  ]
}
```

Development Environment

```
# Install development dependencies
npm install --save-dev

# Set up git hooks (optional)
npx husky install

# Create local environment file
cp .env.example .env.local
```

Environment Variables

```
# .env.local
VITE_APP_TITLE="USSD Flow Editor"
VITE_GIT_WORKFLOW_URL="http://localhost:3001"
VITE_API_BASE_URL="http://localhost:8080"
VITE_ENABLE_DEBUG=true
```

Development Scripts

```
# Development server with hot reload
npm run dev

# Build for production
npm run build

# Preview production build
npm run preview

# Lint code
npm run lint

# Fix linting issues
npm run lint:fix

# Start git workflow server
npm run start:workflow

# Run integration tests
npm run test:integration
```

Code Organization

Component Structure

Custom Node Development

Create new node types following the established pattern:

```
// src/components/NodeTypes/CustomNode.jsx
import { memo } from 'react';
import { Handle, Position } from 'reactflow';

const CustomNode = ({ data, selected }) => {
  const { config, label } = data;

  return (
    <div className={`custom-node ${selected ? 'selected' : ''}`}>
      {/* Input Handle */}
      <Handle
        type="target"
        position={Position.Left}
        id="input"
        className="node-handle"
      />

      {/* Node Content */}
      <div className="node-header">
        <span className="node-icon">🔧</span>
        <span className="node-title">{label}</span>
```

```
        </div>

        <div className="node-content">
          {/* Custom node content */}
          <div className="custom-content">
            {config.customProperty && (
              <div className="property-display">
                {config.customProperty}
              </div>
            )}
          </div>
        </div>

        {/* Output Handles */}
        <Handle
          type="source"
          position={Position.Right}
          id="output"
          className="node-handle"
        />
      </div>
    );
};

export default memo(CustomNode);
```

Configuration Panel Development

Add configuration forms for new node types:

```
// src/components/NodeConfigPanel.jsx - Add to switch statement
case 'CUSTOM':
  return <CustomConfig config={config} onChange={setConfig} />;

// src/components/TemplateForms/CustomConfig.jsx
const CustomConfig = ({ config, onChange }) => {
  const [localConfig, setLocalConfig] = useState(config || {});

  const updateConfig = (field, value) => {
    const updated = { ...localConfig, [field]: value };
    setLocalConfig(updated);
    onChange(updated);
  };

  return (
    <div className="config-form">
      <h3>Custom Node Configuration</h3>

      <div className="form-group">
        <label>Custom Property:</label>
        <input
```

```
            type="text"
            value={localConfig.customProperty || ''}
            onChange={(e) => updateConfig('customProperty', e.target.value)}
            placeholder="Enter custom value"
          />
        </div>

        {/* Multi-language support */}
        <div className="form-group">
          <label>Prompts:</label>
          {['en', 'es', 'fr', 'ar'].map(lang => (
            <div key={lang} className="language-input">
              <label>{lang.toUpperCase()}:</label>
              <textarea
                value={localConfig.prompts?.[lang] || ''}
                onChange={(e) => updateConfig('prompts', {
                  ...localConfig.prompts,
                  [lang]: e.target.value
                })}
                placeholder={`Prompt in ${lang}`}
              />
            </div>
          ))}
        </div>
      </div>
    );
};
```

Utility Development

Adding New Flow Utilities

```
// src/utils/customUtils.js
import { generateUniqueId } from './flowUtils.js';

export const customFlowOperation = (nodes, edges, options = {}) => {
  // Custom flow manipulation logic
  const processedNodes = nodes.map(node => {
    if (node.data.type === 'CUSTOM') {
      return {
        ...node,
        data: {
          ...node.data,
          processed: true,
          timestamp: new Date().toISOString()
        }
      };
    }
    return node;
  });
```

```
  return { nodes: processedNodes, edges };
};

export const validateCustomFlow = (flowData) => {
  const errors = [];
  const warnings = [];

  // Custom validation logic
  flowData.nodes.forEach(node => {
    if (node.data.type === 'CUSTOM') {
      if (!node.data.config?.customProperty) {
        errors.push({
          type: 'missing_custom_property',
          message: `Custom node ${node.id} missing required property`,
          nodeId: node.id
        });
      }
    }
  });

  return { errors, warnings };
};
```

Template System Extensions

```
// src/utils/CustomTemplateProcessor.js
export class CustomTemplateProcessor {
  static process(template, context) {
    // Custom template processing logic
    const processed = {
      ...template,
      processedAt: new Date().toISOString(),
      context: context
    };

    // Apply custom transformations
    if (template.customTransformations) {
      processed.transformedData = this.applyTransformations(
        template.customTransformations,
        context
      );
    }

    return processed;
  }

  static applyTransformations(transformations, context) {
    return transformations.reduce((data, transform) => {
      switch (transform.type) {
        case 'uppercase':
          return data.toUpperCase();
```

```
          case 'variable_substitution':
            return this.substituteVariables(data, context.variables);
          default:
            return data;
        }
      }, context.input);
    }

    static substituteVariables(text, variables) {
      return text.replace(/\{\{(\w+)\}\}/g, (match, varName) => {
        return variables[varName] || match;
      });
    }
  }
```

Development Workflows

Feature Development Process

1. Planning Phase

```
# Create feature branch
git checkout -b feature/new-node-type

# Document feature requirements
echo "# New Node Type Feature" > docs/feature-new-node-type.md
```

2. Implementation Phase

```
// Follow test-driven development
// 1. Write tests first
// 2. Implement functionality
// 3. Refactor and optimize

// Example test structure
describe('CustomNode', () => {
  it('should render with correct props', () => {
    // Test implementation
  });

  it('should handle configuration updates', () => {
    // Test implementation
  });
});
```

3. Integration Phase

```
# Test integration with existing system
npm run dev
# Manual testing on canvas
# Validate export/import functionality
# Check performance impact
```

4. Review Phase

```
# Submit for maker-checker review
curl -X POST http://localhost:3001/api/submit-flow \
  -H "Content-Type: application/json" \
  -d '{"flowData": {...}, "submitter": "developer"}'
```

Git Workflow Integration

Maker-Checker Development

```
// git-workflow-server.js - Add custom endpoints
app.post('/api/custom-review', (req, res) => {
  const { flowData, reviewType } = req.body;

  try {
    // Custom review logic
    const reviewResult = performCustomReview(flowData, reviewType);

    if (reviewResult.approved) {
      // Merge to main branch
      execSync('git checkout main');
      execSync(`git merge ${reviewResult.branchName}`);
      execSync('git push origin main');
    }

    res.json(reviewResult);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

const performCustomReview = (flowData, reviewType) => {
  // Implement custom review logic
  const checks = [];

  // Validate flow structure
  const structureCheck = validateFlowStructure(flowData);
  checks.push(structureCheck);

  // Validate business rules
```

```
  const businessCheck = validateBusinessRules(flowData);
  checks.push(businessCheck);

  const approved = checks.every(check => check.passed);

  return {
    approved,
    checks,
    branchName: flowData.metadata.branchName,
    reviewedAt: new Date().toISOString()
  };
};
```

Adding New Features

New Node Type Development

Step 1: Define Node Schema

```
// src/schemas/nodeSchemas.js
export const CUSTOM_NODE_SCHEMA = {
  type: 'CUSTOM',
  defaultConfig: {
    customProperty: '',
    prompts: {
      en: '',
      es: '',
      fr: '',
      ar: ''
    },
    transitions: {},
    validation: {
      required: ['customProperty']
    }
  },
  handles: {
    input: { type: 'target', position: 'left' },
    output: { type: 'source', position: 'right' }
  }
};
```

Step 2: Create Node Component

```
// src/components/NodeTypes/CustomNode.jsx
// (Implementation shown in Code Organization section)
```

Step 3: Register Node Type

```
// src/App.jsx - Add to nodeTypes object
const nodeTypes = {
  start: StartNode,
  menu: MenuNode,
  'dynamic-menu': DynamicMenuNode,
  input: InputNode,
  action: ActionNode,
  end: EndNode,
  custom: CustomNode, // Add new node type
};
```

Step 4: Add to Palette

```
// src/components/NodePalette.jsx
const nodeTypes = [
  { type: 'start', label: 'Start', icon: '🚀' },
  { type: 'menu', label: 'Menu', icon: '📋' },
  { type: 'dynamic-menu', label: 'Dynamic Menu', icon: '🔢' },
  { type: 'input', label: 'Input', icon: '⌨️' },
  { type: 'action', label: 'Action', icon: '⚡' },
  { type: 'end', label: 'End', icon: '🏁' },
  { type: 'custom', label: 'Custom', icon: '🔧' }, // Add to palette
];
```

Step 5: Export Integration

```
// src/utils/flowUtils.js - Update export logic
export const exportToFlowFormat = (nodes, edges) => {
  return nodes.map(node => {
    const baseExport = {
      id: node.id,
      type: node.data.type,
      transitions: getNodeTransitions(node, edges)
    };

    // Handle custom node export
    if (node.data.type === 'CUSTOM') {
      return {
        ...baseExport,
        customProperty: node.data.config?.customProperty,
        customData: extractCustomData(node.data.config)
      };
    }

    return baseExport;
  });
};
```

New Utility Development

Creating Reusable Utilities

```
// src/utils/customValidation.js
export const createValidator = (rules) => {
  return (data) => {
    const errors = [];
    const warnings = [];

    rules.forEach(rule => {
      const result = rule.validate(data);
      if (result.error) {
        errors.push({
          rule: rule.name,
          message: result.message,
          severity: rule.severity || 'error'
        });
      }
      if (result.warning) {
        warnings.push({
          rule: rule.name,
          message: result.warning,
          severity: 'warning'
        });
      }
    });

    return { errors, warnings, valid: errors.length === 0 };
  };
};

// Usage example
const flowValidator = createValidator([
  {
    name: 'start_node_check',
    validate: (flow) => {
      const startNodes = flow.nodes.filter(n => n.type === 'START');
      if (startNodes.length === 0) {
        return { error: 'Flow must have a START node' };
      }
      if (startNodes.length > 1) {
        return { warning: 'Multiple START nodes found' };
      }
      return { valid: true };
    }
  }
]);
```

Testing Guidelines

Unit Testing

```
// src/components/__tests__/CustomNode.test.jsx
import { render, screen } from '@testing-library/react';
import { ReactFlowProvider } from 'reactflow';
import CustomNode from '../NodeTypes/CustomNode';

const renderWithReactFlow = (component) => {
  return render(
    <ReactFlowProvider>
      {component}
    </ReactFlowProvider>
  );
};

describe('CustomNode', () => {
  const mockData = {
    label: 'Test Custom Node',
    type: 'CUSTOM',
    config: {
      customProperty: 'test value',
      prompts: {
        en: 'Test prompt'
      }
    }
  };

  it('renders node with correct content', () => {
    renderWithReactFlow(
      <CustomNode data={mockData} selected={false} />
    );

    expect(screen.getByText('Test Custom Node')).toBeInTheDocument();
    expect(screen.getByText('test value')).toBeInTheDocument();
  });

  it('applies selected styling when selected', () => {
    renderWithReactFlow(
      <CustomNode data={mockData} selected={true} />
    );

    const nodeElement = screen.getByRole('generic');
    expect(nodeElement).toHaveClass('selected');
  });
});
```

Integration Testing

```
// src/__tests__/integration/flowOperations.test.js
import { exportToFlowFormat } from '../../utils/flowUtils';
import { validateFlow } from '../../utils/validation';

describe('Flow Operations Integration', () => {
  const sampleFlow = {
    nodes: [
      {
        id: 'start_1',
        data: { type: 'START', config: { ussdCode: '*123#' } }
      },
      {
        id: 'custom_1',
        data: { type: 'CUSTOM', config: { customProperty: 'value' } }
      }
    ],
    edges: [
      {
        source: 'start_1',
        target: 'custom_1',
        sourceHandle: '*123#'
      }
    ]
  };

  it('exports custom nodes correctly', () => {
    const exported = exportToFlowFormat(sampleFlow.nodes, sampleFlow.edges);

    const customNode = exported.find(n => n.type === 'CUSTOM');
    expect(customNode).toBeDefined();
    expect(customNode.customProperty).toBe('value');
    expect(customNode.transitions['*123#']).toBe('custom_1');
  });

  it('validates flows with custom nodes', () => {
    const validation = validateFlow(sampleFlow.nodes, sampleFlow.edges);
    expect(validation.errors).toHaveLength(0);
  });
});
```

Load Testing

```
// load-testing/custom-node-test.js
import http from 'k6/http';
import { check } from 'k6';

export const options = {
  scenarios: {
    custom_node_load: {
      executor: 'constant-vus',
```

```
      vus: 50,
      duration: '5m',
    },
  },
};

export default function () {
  // Test custom node flow processing
  const flowData = {
    nodes: [
      { type: 'START', config: { ussdCode: '*123#' } },
      { type: 'CUSTOM', config: { customProperty: 'load_test' } }
    ]
  };

  const response = http.post(
    'http://localhost:8080/api/process-flow',
    JSON.stringify(flowData),
    { headers: { 'Content-Type': 'application/json' } }
  );

  check(response, {
    'custom node processed': (r) => r.status === 200,
    'response time < 500ms': (r) => r.timings.duration < 500,
  });
}
```

Deployment Guide

Production Build

```
# Create optimized production build
npm run build

# Test production build locally
npm run preview

# Verify build artifacts
ls -la dist/
```

Docker Deployment

```
# Dockerfile
FROM node:18-alpine AS builder

WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
```

```
COPY . .
RUN npm run build

FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

```
# docker-compose.yml
version: '3.8'
services:
  ussd-editor:
    build: .
    ports:
      - "80:80"
    environment:
      - VITE_API_BASE_URL=https://api.production.com
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf

  git-workflow:
    build: ./git-workflow-server
    ports:
      - "3001:3001"
    volumes:
      - ./workflow-data:/app/data
```

Environment Configuration

```
# Production environment variables
VITE_APP_TITLE="USSD Flow Editor - Production"
VITE_GIT_WORKFLOW_URL="https://workflow.production.com"
VITE_API_BASE_URL="https://api.production.com"
VITE_ENABLE_DEBUG=false
```

CI/CD Pipeline

```
# .github/workflows/deploy.yml
name: Deploy to Production

on:
  push:
    branches: [main]

jobs:
```

```
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: '18'
      - run: npm ci
      - run: npm run lint
      - run: npm run test
      - run: npm run build

  deploy:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Deploy to production
        run: |
          docker build -t ussd-editor:latest .
          docker push ${{ secrets.DOCKER_REGISTRY }}/ussd-editor:latest
```

Troubleshooting

Common Issues

React Flow Canvas Issues

```
// Issue: Nodes not rendering correctly
// Solution: Ensure nodeTypes are properly registered

// Check node type registration
const nodeTypes = useMemo(() => ({
  start: StartNode,
  menu: MenuNode,
  custom: CustomNode,
}), []);

// Ensure proper CSS imports
import 'reactflow/dist/style.css';
import './styles/custom-nodes.css';
```

State Synchronization Problems

```
// Issue: Configuration not saving
// Solution: Verify state update pattern

// Incorrect - direct mutation
```

```
node.data.config.property = newValue;

// Correct - immutable update
setNodes(nodes =>
  nodes.map(n =>
    n.id === nodeId
      ? { ...n, data: { ...n.data, config: { ...n.data.config, property: newValue
} }}
      : n
  )
);
```

Performance Issues

```
// Issue: Slow rendering with many nodes
// Solution: Implement optimization strategies

// Use React.memo for node components
export default memo(CustomNode);

// Optimize handle rendering
const handles = useMemo(() =>
  generateHandles(config.transitions),
  [config.transitions]
);

// Debounce expensive operations
const debouncedSave = useMemo(
  () => debounce(saveToStorage, 1000),
  []
);
```

Export/Import Issues

```
// Issue: Data loss during export
// Solution: Validate export format

const validateExport = (exportData) => {
  const required = ['id', 'type', 'transitions'];

  return exportData.every(node =>
    required.every(field => field in node)
  );
};

// Use in export process
const exported = exportToFlowFormat(nodes, edges);
if (!validateExport(exported)) {
```

```
    throw new Error('Export validation failed');
  }
```

Debug Utilities

```
// src/utils/debugUtils.js
export const debugFlow = (nodes, edges) => {
  console.group('Flow Debug Information');
  console.log('Nodes:', nodes.length);
  console.log('Edges:', edges.length);

  // Check for common issues
  const orphanedNodes = nodes.filter(node =>
    !edges.some(edge => edge.source === node.id || edge.target === node.id)
  );

  if (orphanedNodes.length > 0) {
    console.warn('Orphaned nodes found:', orphanedNodes);
  }

  // Validate transitions
  nodes.forEach(node => {
    const nodeEdges = edges.filter(edge => edge.source === node.id);
    const configTransitions = Object.keys(node.data.config?.transitions || {});

    if (nodeEdges.length !== configTransitions.length) {
      console.warn(`Transition mismatch for node ${node.id}`);
    }
  });

  console.groupEnd();
};

// Enable debug mode
if (import.meta.env.VITE_ENABLE_DEBUG) {
  window.debugFlow = debugFlow;
}
```

This developer guide provides comprehensive information for working with the USSD Flow Editor codebase, from initial setup through advanced feature development and deployment.