GUI Configuration Tool Specification

Overview

Create a user-friendly graphical interface for configuring Audityzer projects, targeting non-CLI users and providing visual project setup capabilities. This tool will complement the CLI while making Audityzer accessible to a broader audience.

Technology Stack

Frontend Framework: Electron + React

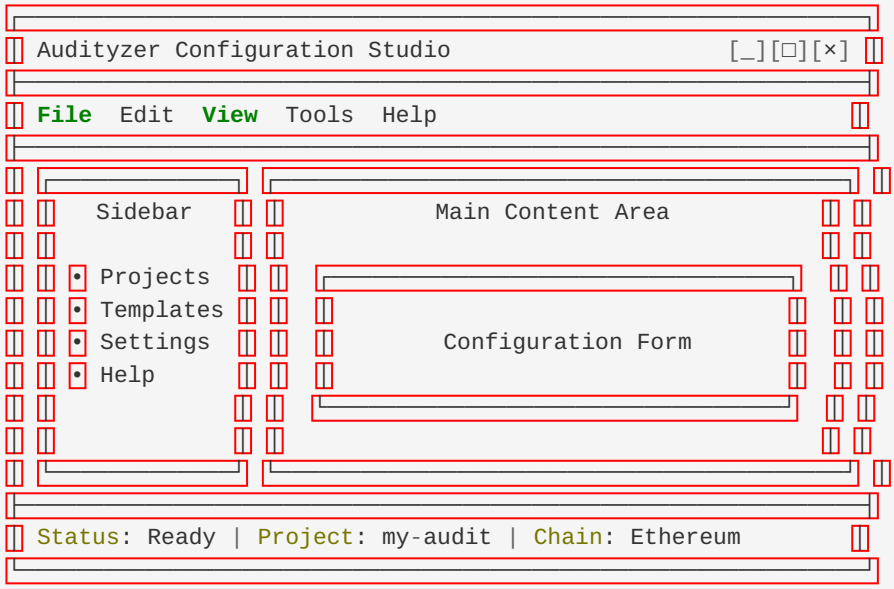
- **Electron**: Cross-platform desktop application
- **React**: Component-based UI framework
- **TypeScript**: Type safety and better developer experience
- **Tailwind CSS**: Utility-first styling
- **React Hook Form**: Form management
- **Zustand**: State management

Backend Integration

- **Node.js**: Backend services
- **IPC**: Electron main/renderer communication
- **File System**: Direct project file manipulation
- **CLI Integration**: Execute Audityzer commands

User Interface Design

Main Application Layout



Welcome Screen

```
const WelcomeScreen = () => (
  <div className="flex flex-col items-center justify-center h-full bg-gradient-to-br
from-blue-50 to-indigo-100">
    <div className="text-center max-w-2xl">
      <img src="/logo.svg" alt="Audityzer" className="w-24 h-24 mx-auto mb-6" />
      <h1 className="text-4xl font-bold text-gray-900 mb-4">
        Welcome to Audityzer Studio
      </h1>
      <p className="text-xl text-gray-600 mb-8">
        Visual configuration tool for Web3 security testing
      </p>

      <div className="grid grid-cols-1 md:grid-cols-3 gap-6">
        <ActionCard
          icon={<PlusIcon />}
          title="New Project"
          description="Create a new security testing project"
          onClick={() => navigate('/new-project')}
        />
        <ActionCard
          icon={<FolderIcon />}
          title="Open Project"
          description="Open an existing Audityzer project"
          onClick={() => openProject()}
        />
        <ActionCard
          icon={<TemplateIcon />}
          title="Browse Templates"
          description="Explore pre-built project templates"
          onClick={() => navigate('/templates')}
        />
      </div>
    </div>
  </div>
);
```

Core Features

1. Project Creation Wizard

Step 1: Project Type Selection

```
const ProjectTypeStep = ({ onNext, onBack }) => {
  const [selectedType, setSelectedType] = useState('');

  const projectTypes = [
    {
      id: 'defi',
      name: 'DeFi Protocol',
      description: 'Test lending, DEX, and yield farming protocols',
      icon: <DeFiIcon />,
      features: ['Flash loan testing', 'Oracle manipulation', 'Reentrancy detection']
    },
    {
      id: 'aa',
      name: 'Account Abstraction',
      description: 'Test ERC-4337 wallets and infrastructure',
      icon: <AAIcon />,
      features: ['UserOp validation', 'Bundler testing', 'Paymaster security']
    },
    {
      id: 'bridge',
      name: 'Cross-Chain Bridge',
      description: 'Test bridge security and cross-chain protocols',
      icon: <BridgeIcon />,
      features: ['Bridge validation', 'Cross-chain security', 'Asset locking']
    },
    {
      id: 'wallet',
      name: 'Wallet Application',
      description: 'Test wallet security and user interactions',
      icon: <WalletIcon />,
      features: ['Transaction security', 'Key management', 'UI/UX testing']
    }
  ];

  return (
    <div className="space-y-6">
      <div className="text-center">
        <h2 className="text-2xl font-bold text-gray-900">Choose Project Type</h2>
        <p className="text-gray-600 mt-2">Select the type of protocol you want to test<
/p>
      </div>

      <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
        {projectTypes.map((type) => (
          <ProjectTypeCard
            key={type.id}
            type={type}
            selected={selectedType === type.id}
            onSelect={() => setSelectedType(type.id)}
          />
        ))}
      </div>

      <div className="flex justify-between">
        <Button variant="outline" onClick={onBack}>Back</Button>
        <Button
          onClick={() => onNext({ projectType: selectedType })}
          disabled={!selectedType}
```

```
        >
          Next
        </Button>
      </div>
    </div>
  );
};
```

Step 2: Basic Configuration

```
const BasicConfigStep = ({ data, onNext, onBack }) => {
  const { register, handleSubmit, watch, formState: { errors } } = useForm({
    defaultValues: data
  });

  const projectType = watch('projectType');

  return (
    <form onSubmit={handleSubmit(onNext)} className="space-y-6">
      <div className="grid grid-cols-1 md:grid-cols-2 gap-6">
        <FormField
          label="Project Name"
          error={errors.projectName}
          required
        >
          <input
            {...register('projectName', { required: 'Project name is required' })}
            className="input"
            placeholder="my-audit-project"
          />
        </FormField>

        <FormField
          label="Primary Blockchain"
          error={errors.chain}
        >
          <select {...register('chain')} className="select">
            <option value="ethereum">Ethereum</option>
            <option value="polygon">Polygon</option>
            <option value="arbitrum">Arbitrum</option>
            <option value="optimism">Optimism</option>
            <option value="base">Base</option>
          </select>
        </FormField>

        <FormField
          label="Test Wallet"
          error={errors.wallet}
        >
          <select {...register('wallet')} className="select">
            <option value="metamask">MetaMask</option>
            <option value="coinbase">Coinbase Wallet</option>
            <option value="walletconnect">WalletConnect</option>
            <option value="mock">Mock/Simulation</option>
          </select>
        </FormField>

        <FormField
          label="Output Format"
          error={errors.outputFormat}
        >
          <select {...register('outputFormat')} className="select">
            <option value="json">JSON</option>
            <option value="html">HTML</option>
            <option value="markdown">Markdown</option>
          </select>
        </FormField>
      </div>
```

```
      {projectType === 'aa' && (
        <AAConfigSection register={register} errors={errors} />
      )}

      <div className="flex justify-between">
        <Button variant="outline" onClick={onBack}>Back</Button>
        <Button type="submit">Next</Button>
      </div>
    </form>
  );
};
```

Step 3: Test Configuration

```
const TestConfigStep = ({ data, onNext, onBack }) => {
  const [selectedTests, setSelectedTests] = useState(data.tests || []);

  const availableTests = getAvailableTests(data.projectType);

  const toggleTest = (testId) => {
    setSelectedTests(prev =>
      prev.includes(testId)
        ? prev.filter(id => id !== testId)
        : [...prev, testId]
    );
  };

  return (
    <div className="space-y-6">
      <div className="text-center">
        <h2 className="text-2xl font-bold text-gray-900">Select Security Tests</h2>
        <p className="text-gray-600 mt-2">Choose which vulnerability types to test for<
/p>
      </div>

      <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
        {availableTests.map((test) => (
          <TestCard
            key={test.id}
            test={test}
            selected={selectedTests.includes(test.id)}
            onToggle={() => toggleTest(test.id)}
          />
        ))}
      </div>

      <div className="bg-blue-50 border border-blue-200 rounded-lg p-4">
        <h3 className="font-semibold text-blue-900 mb-2">Recommended Tests</h3>
        <p className="text-blue-700 text-sm">
          Based on your project type, we recommend: {getRecommen-
dedTests(data.projectType).join(', ')}
        </p>
      </div>

      <div className="flex justify-between">
        <Button variant="outline" onClick={onBack}>Back</Button>
        <Button
          onClick={() => onNext({ ...data, tests: selectedTests })}
          disabled={selectedTests.length === 0}
        >
          Next
        </Button>
      </div>
    </div>
  );
};
```

2. Visual Configuration Editor

Configuration Tree View

```
const ConfigurationEditor = ({ config, onChange }) => {
  const [expandedSections, setExpandedSections] = useState(['basic', 'testing']);

  const configSections = [
    {
      id: 'basic',
      title: 'Basic Configuration',
      icon: <SettingsIcon />,
      component: BasicConfigSection
    },
    {
      id: 'testing',
      title: 'Test Configuration',
      icon: <TestIcon />,
      component: TestConfigSection
    },
    {
      id: 'chains',
      title: 'Blockchain Networks',
      icon: <ChainIcon />,
      component: ChainConfigSection
    },
    {
      id: 'reporting',
      title: 'Reports & Output',
      icon: <ReportIcon />,
      component: ReportConfigSection
    },
    {
      id: 'advanced',
      title: 'Advanced Settings',
      icon: <AdvancedIcon />,
      component: AdvancedConfigSection
    }
  ];

  return (
    <div className="flex h-full">
      <div className="w-1/3 border-r border-gray-200 bg-gray-50">
        <ConfigurationTree
          sections={configSections}
          expanded={expandedSections}
          onToggle={setExpandedSections}
        />
      </div>

      <div className="flex-1 p-6">
        <ConfigurationForm
          config={config}
          onChange={onChange}
          activeSection={expandedSections[0]}
        />
      </div>
    </div>
  );
};
```

Real-time Configuration Preview

```
const ConfigurationPreview = ({ config }) => {
  const [previewMode, setPreviewMode] = useState('visual');

  return (
    <div className="bg-white border border-gray-200 rounded-lg">
      <div className="border-b border-gray-200 px-4 py-3">
        <div className="flex items-center justify-between">
          <h3 className="font-semibold text-gray-900">Configuration Preview</h3>
          <div className="flex space-x-2">
            <Button
              variant={previewMode === 'visual' ? 'primary' : 'outline'}
              size="sm"
              onClick={() => setPreviewMode('visual')}
            >
              Visual
            </Button>
            <Button
              variant={previewMode === 'code' ? 'primary' : 'outline'}
              size="sm"
              onClick={() => setPreviewMode('code')}
            >
              Code
            </Button>
          </div>
        </div>
      </div>

      <div className="p-4">
        {previewMode === 'visual' ? (
          <VisualConfigPreview config={config} />
        ) : (
          <CodeConfigPreview config={config} />
        )}
      </div>
    </div>
  );
};
```

3. Test Execution Interface

Test Runner Dashboard

```
const TestRunnerDashboard = ({ project }) => {
  const [isRunning, setIsRunning] = useState(false);
  const [testResults, setTestResults] = useState(null);
  const [logs, setLogs] = useState([]);

  const runTests = async () => {
    setIsRunning(true);
    setLogs([]);

    try {
      const result = await executeTests(project.config, {
        onLog: (log) => setLogs(prev => [...prev, log]),
        onProgress: (progress) => setProgress(progress)
      });

      setTestResults(result);
    } catch (error) {
      console.error('Test execution failed:', error);
    } finally {
      setIsRunning(false);
    }
  };

  return (
    <div className="space-y-6">
      <div className="flex items-center justify-between">
        <h2 className="text-2xl font-bold text-gray-900">Test Execution</h2>
        <div className="flex space-x-3">
          <Button
            variant="outline"
            onClick={() => runTests({ mock: true })}
            disabled={isRunning}
          >
            Mock Run
          </Button>
          <Button
            onClick={runTests}
            disabled={isRunning}
            className="flex items-center space-x-2"
          >
            {isRunning ? <SpinnerIcon /> : <PlayIcon />}
            <span>{isRunning ? 'Running...' : 'Run Tests'}</span>
          </Button>
        </div>
      </div>

      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        <TestProgress
          isRunning={isRunning}
          results={testResults}
          config={project.config}
        />

        <TestLogs
          logs={logs}
          isRunning={isRunning}
        />
      </div>
```

```
      {testResults && (
        <TestResults
          results={testResults}
          onExport={exportResults}
          onViewReport={viewReport}
        />
      )}
    </div>
  );
};
```

4. Report Visualization

Interactive Report Viewer

```
const ReportViewer = ({ results }) => {
  const [activeTab, setActiveTab] = useState('overview');

  const tabs = [
    { id: 'overview', label: 'Overview', icon: <OverviewIcon /> },
    { id: 'vulnerabilities', label: 'Vulnerabilities', icon: <VulnIcon /> },
    { id: 'performance', label: 'Performance', icon: <PerfIcon /> },
    { id: 'recommendations', label: 'Recommendations', icon: <RecommendIcon /> }
  ];

  return (
    <div className="bg-white border border-gray-200 rounded-lg">
      <div className="border-b border-gray-200">
        <nav className="flex space-x-8 px-6">
          {tabs.map((tab) => (
            <button
              key={tab.id}
              onClick={() => setActiveTab(tab.id)}
              className={`py-4 px-1 border-b-2 font-medium text-sm ${
                activeTab === tab.id
                  ? 'border-blue-500 text-blue-600'
                  : 'border-transparent text-gray-500 hover:text-gray-700'
              }`}
            >
              <div className="flex items-center space-x-2">
                {tab.icon}
                <span>{tab.label}</span>
              </div>
            </button>
          ))}
        </nav>
      </div>

      <div className="p-6">
        {activeTab === 'overview' && <OverviewTab results={results} />}
        {activeTab === 'vulnerabilities' && <VulnerabilitiesTab results={results} />}
        {activeTab === 'performance' && <PerformanceTab results={results} />}
        {activeTab === 'recommendations' && <RecommendationsTab results={results} />}
      </div>
    </div>
  );
};
```

Technical Implementation

Electron Main Process

```
// src/main/main.ts
import { app, BrowserWindow, ipcMain, dialog } from 'electron';
import { AudityzerService } from './services/AudityzerService';

class MainApplication {
  private mainWindow: BrowserWindow | null = null;
  private audityzerService: AudityzerService;

  constructor() {
    this.audityzerService = new AudityzerService();
    this.setupIPC();
  }

  private setupIPC() {
    // Project management
    ipcMain.handle('project:create', async (_, config) => {
      return await this.audityzerService.createProject(config);
    });

    ipcMain.handle('project:open', async () => {
      const result = await dialog.showOpenDialog(this.mainWindow!, {
        properties: ['openDirectory'],
        title: 'Select Audityzer Project'
      });

      if (!result.canceled) {
        return await this.audityzerService.loadProject(result.filePaths[0]);
      }
    });

    // Test execution
    ipcMain.handle('tests:run', async (_, config, options) => {
      return await this.audityzerService.runTests(config, options);
    });

    // Configuration management
    ipcMain.handle('config:validate', async (_, config) => {
      return await this.audityzerService.validateConfig(config);
    });
  }
}
```

Audityzer Service Integration

```typescript
// src/main/services/AudityzerService.ts
import { spawn } from 'child_process';
import { EventEmitter } from 'events';

export class AudityzerService extends EventEmitter {
  async createProject(config: ProjectConfig): Promise<ProjectResult> {
    const projectPath = await this.generateProjectStructure(config);
    await this.writeConfigFiles(projectPath, config);
    await this.installDependencies(projectPath);

    return {
      success: true,
      projectPath,
      config
    };
  }

  async runTests(config: ProjectConfig, options: TestOptions): Promise<TestResult> {
    return new Promise((resolve, reject) => {
      const args = this.buildCliArgs(config, options);
      const process = spawn('audityzer', args, {
        cwd: config.projectPath,
        stdio: ['pipe', 'pipe', 'pipe']
      });

      let output = '';
      let errors = '';

      process.stdout.on('data', (data) => {
        const chunk = data.toString();
        output += chunk;
        this.emit('test:log', { type: 'stdout', data: chunk });
      });

      process.stderr.on('data', (data) => {
        const chunk = data.toString();
        errors += chunk;
        this.emit('test:log', { type: 'stderr', data: chunk });
      });

      process.on('close', (code) => {
        if (code === 0) {
          resolve(this.parseTestResults(output));
        } else {
          reject(new Error(`Test execution failed: ${errors}`));
        }
      });
    });
  }

  private buildCliArgs(config: ProjectConfig, options: TestOptions): string[] {
    const args = ['run', config.target];

    if (config.chain) args.push('--chain', config.chain);
    if (config.wallet) args.push('--wallet', config.wallet);
    if (config.aa?.enabled) args.push('--aa');
    if (config.tests) args.push('--tests', ...config.tests);
    if (options.mock) args.push('--mock');
```

```
    if (options.verbose) args.push('--verbose');

    return args;
  }
}
```

React Frontend State Management

```typescript
// src/renderer/store/projectStore.ts
import { create } from 'zustand';
import { persist } from 'zustand/middleware';

interface ProjectState {
  currentProject: Project | null;
  recentProjects: Project[];
  isLoading: boolean;

  // Actions
  setCurrentProject: (project: Project) => void;
  addRecentProject: (project: Project) => void;
  createProject: (config: ProjectConfig) => Promise<void>;
  openProject: (path: string) => Promise<void>;
  runTests: (options: TestOptions) => Promise<TestResult>;
}

export const useProjectStore = create<ProjectState>()(
  persist(
    (set, get) => ({
      currentProject: null,
      recentProjects: [],
      isLoading: false,

      setCurrentProject: (project) => set({ currentProject: project }),

      addRecentProject: (project) => set((state) => ({
        recentProjects: [
          project,
          ...state.recentProjects.filter(p => p.path !== project.path)
        ].slice(0, 10)
      })),

      createProject: async (config) => {
        set({ isLoading: true });
        try {
          const result = await window.electronAPI.project.create(config);
          set({ currentProject: result.project });
          get().addRecentProject(result.project);
        } finally {
          set({ isLoading: false });
        }
      },

      openProject: async (path) => {
        set({ isLoading: true });
        try {
          const project = await window.electronAPI.project.open(path);
          set({ currentProject: project });
          get().addRecentProject(project);
        } finally {
          set({ isLoading: false });
        }
      },

      runTests: async (options) => {
        const { currentProject } = get();
        if (!currentProject) throw new Error('No project selected');
```

```
        return await window.electronAPI.tests.run(currentProject.config, options);
      }
    }),
    {
      name: 'audityzer-projects',
      partialize: (state) => ({
        recentProjects: state.recentProjects
      })
    }
  )
);
```

Build & Distribution

Build Configuration

```
// electron-builder.config.js
module.exports = {
  appId: 'com.audityzer.studio',
  productName: 'Audityzer Studio',
  directories: {
    output: 'dist'
  },
  files: [
    'build/**/*',
    'node_modules/**/*',
    'package.json'
  ],
  mac: {
    category: 'public.app-category.developer-tools',
    icon: 'assets/icon.icns',
    target: [
      { target: 'dmg', arch: ['x64', 'arm64'] },
      { target: 'zip', arch: ['x64', 'arm64'] }
    ]
  },
  win: {
    icon: 'assets/icon.ico',
    target: [
      { target: 'nsis', arch: ['x64'] },
      { target: 'portable', arch: ['x64'] }
    ]
  },
  linux: {
    icon: 'assets/icon.png',
    target: [
      { target: 'AppImage', arch: ['x64'] },
      { target: 'deb', arch: ['x64'] },
      { target: 'rpm', arch: ['x64'] }
    ]
  }
};
```

Auto-Update Configuration

```
// src/main/updater.ts
import { autoUpdater } from 'electron-updater';

export class UpdaterService {
  constructor() {
    autoUpdater.checkForUpdatesAndNotify();

    autoUpdater.on('update-available', () => {
      // Notify user of available update
    });

    autoUpdater.on('update-downloaded', () => {
      // Prompt user to restart and install
    });
  }

  async checkForUpdates() {
    return await autoUpdater.checkForUpdates();
  }

  async downloadUpdate() {
    return await autoUpdater.downloadUpdate();
  }

  quitAndInstall() {
    autoUpdater.quitAndInstall();
  }
}
```

Success Metrics

User Experience Metrics

- **Setup Completion Rate**: Target 95% (vs 60% CLI)
- **Time to First Test**: Target 3 minutes (vs 15 minutes CLI)
- **User Satisfaction**: Target 4.5/5 stars
- **Support Ticket Reduction**: Target 50% reduction

Adoption Metrics

- **Downloads**: Target 10K+ in first quarter
- **Active Users**: Target 1K+ monthly active users
- **Retention**: Target 70% 30-day retention
- **Conversion**: Target 30% CLI adoption from GUI users

This GUI configuration tool will make Audityzer accessible to a much broader audience while maintaining the power and flexibility that advanced users expect. The visual interface will reduce barriers to entry and improve the overall developer experience.