perplexity

# Complete Context-Engineered Prompt for Quantum Cryptography Tutorial Website

## Ready-to-Use Copy-Paste Prompt

```
Create a complete quantum cryptography tutorial website with the following exact specific

TECH STACK:
- Frontend: HTML5/CSS3/Vanilla JavaScript
- Backend: Python Flask
- Quantum Computing: Qiskit (small simulations) + QuTiP (large simulations)
- No frameworks, pure vanilla implementation

PROJECT REQUIREMENTS:
1. Two main educational sections:
   - Quantum Computing Basics with interactive visualizations
   - BB84 Quantum Key Distribution Protocol simulation
2. Interactive UI with drag-and-drop quantum gate builder
3. Real-time Bloch sphere visualizations using matplotlib
4. Complete BB84 protocol with Alice/Bob simulation
5. Base64 encoded image responses for quantum visualizations
6. CORS-enabled API for frontend-backend communication

EXACT FILE STRUCTURE NEEDED:
quantum-crypto-tutorial/
├── README.md
├── requirements.txt
├── run.py
├── frontend/
│   ├── index.html
│   ├── quantum-basics/
│   │   ├── quantum-intro.html
│   │   ├── qubits-tutorial.html
│   │   ├── quantum-gates.html
│   │   └── quantum-circuits.html
│   ├── bb84-protocol/
│   │   ├── bb84-tutorial.html
│   │   └── bb84-simulator.html
│   ├── css/
│   │   ├── styles.css
│   │   ├── quantum-visualizations.css
│   │   └── bb84-interface.css
│   ├── js/
│   │   ├── main.js
│   │   ├── quantum-visualizer.js
│   │   ├── bloch-sphere.js
│   │   └── bb84-simulator.js
│   └── assets/
```

```
│       └── images/
├── backend/
│   ├── app.py
│   ├── quantum_simulator/
│   │   ├── __init__.py
│   │   ├── circuit_simulator.py
│   │   ├── state_visualizer.py
│   │   └── advanced_simulator.py
│   ├── cryptography/
│   │   ├── __init__.py
│   │   ├── bb84_protocol.py
│   │   ├── quantum_channel.py
│   │   └── security_analysis.py
│   └── utils/
│       ├── __init__.py
│       ├── image_generator.py
│       └── data_validators.py
```

SPECIFIC FEATURES TO IMPLEMENT:

QUANTUM COMPUTING BASICS:
- Interactive circuit builder with drag-and-drop gates (H, X, Y, Z, CNOT)
- Real-time statevector calculation using Qiskit
- Bloch sphere visualization generation (matplotlib to base64)
- Qubit state superposition demonstrations
- Quantum gate effect visualization

BB84 PROTOCOL SIMULATION:
- Complete Alice/Bob bit generation and encoding
- Random basis selection for both parties
- Quantum channel transmission simulation
- Public basis comparison and key sifting
- Eavesdropping detection with error rate calculation
- Visual step-by-step protocol demonstration

API ENDPOINTS REQUIRED:
- GET /api/quantum-basics/<lesson> - Tutorial content delivery
- POST /api/simulate-circuit - Qiskit circuit simulation
- POST /api/advanced-simulation - QuTiP complex dynamics
- POST /api/bb84/simulate - Complete BB84 protocol execution
- GET /api/bb84/tutorial - BB84 educational content

TECHNICAL SPECIFICATIONS:
- Flask with CORS enabled for cross-origin requests
- Matplotlib non-interactive backend for server-side image generation
- JSON responses with base64 encoded visualizations
- Input validation for quantum circuit parameters
- Error handling for quantum simulation failures
- Responsive CSS design for educational content

CODE REQUIREMENTS:
- Complete, production-ready code with detailed comments
- Modular architecture for easy expansion
- Error handling and input validation
- Base64 image encoding utilities
- Quantum state visualization generators

```
- Interactive JavaScript components without external dependencies

DOCUMENTATION NEEDED:
- Complete setup and installation guide
- API endpoint documentation with examples
- File structure explanation
- Future development guidelines
- Testing and deployment instructions
- Dependency management documentation

OUTPUT FORMAT:
Provide each file with complete code implementation, proper file headers, and detailed co
```

## Project Setup and Running Guide

### Installation Steps

1. **Clone/Create Project Directory:**

```
mkdir quantum-crypto-tutorial
cd quantum-crypto-tutorial
```

2. **Install Python Dependencies:**

```
pip install -r requirements.txt
```

3. **Start Backend Server:**

```
python run.py
```

4. **Serve Frontend:**

```
# Option 1: Python HTTP Server
cd frontend
python -m http.server 3000

# Option 2: Open index.html directly in browser
# Navigate to frontend/index.html
```

5. **Access Application:**

- Frontend: `http://localhost:3000`
- Backend API: `http://localhost:5000`
- API Documentation: `http://localhost:5000/docs` (if implemented)

### Development Workflow

1. **Frontend Development:**
   - Edit HTML files in `frontend/` directories
   - Modify CSS in `frontend/css/`
   - Update JavaScript in `frontend/js/`
   - Refresh browser to see changes
2. **Backend Development:**
   - Modify Python files in `backend/`
   - Restart Flask server: `python run.py`
   - Test API endpoints using browser or Postman
3. **Quantum Algorithm Updates:**
   - Add new simulations in `backend/quantum_simulator/`
   - Create corresponding frontend interfaces
   - Update API routes in `backend/app.py`

### Project Structure Explanation

**Frontend Organization:**

- `index.html` - Main navigation and landing page
- `quantum-basics/` - Interactive quantum computing tutorials
- `bb84-protocol/` - Quantum key distribution simulation
- `css/` - Styling for responsive design and quantum visualizations
- `js/` - JavaScript modules for interactivity and API communication

**Backend Organization:**

- `app.py` - Flask application with API routes
- `quantum_simulator/` - Qiskit and QuTiP integration modules
- `cryptography/` - BB84 and quantum cryptography implementations
- `utils/` - Helper functions for image generation and validation

### Future Development Guidelines

**Adding New Quantum Protocols:**

1. Create protocol module in `backend/cryptography/`
2. Add API endpoint in `backend/app.py`
3. Create frontend interface following BB84 pattern
4. Update navigation in `index.html`

**Expanding Quantum Simulations:**

1. Add simulation logic in `backend/quantum_simulator/`
2. Create visualization components in `frontend/js/`
3. Add corresponding HTML templates
4. Update CSS for new UI components

**Performance Optimization:**

1. Implement caching for quantum simulations
2. Add lazy loading for heavy visualizations
3. Optimize matplotlib figure generation
4. Consider WebAssembly for client-side computations

## Testing Instructions

**Manual Testing:**

1. Navigate through all tutorial sections
2. Test quantum circuit builder functionality
3. Run BB84 protocol simulation with different parameters
4. Verify Bloch sphere visualizations display correctly
5. Check responsive design on mobile devices

**API Testing:**

```
# Test circuit simulation
curl -X POST http://localhost:5000/api/simulate-circuit \
  -H "Content-Type: application/json" \
  -d '{"num_qubits": 2, "gates": [{"type": "H", "qubit": 0}]}'

# Test BB84 protocol
curl -X POST http://localhost:5000/api/bb84/simulate \
  -H "Content-Type: application/json" \
  -d '{"num_bits": 100}'
```

This comprehensive prompt and guide provides everything needed to implement and maintain your quantum cryptography tutorial website. The context-engineered prompt includes all technical specifications, while the documentation ensures smooth development and future updates.