To build a web application for cryptocurrency arbitrage as described, you'll need to create a full-stack application with ReactJS for the frontend, Flask for the backend, MySQL for data storage, and Python for the arbitrage algorithm. The application will also utilize Redis for event handling and WebSockets for real-time updates.

Here is a step-by-step guide to achieving this:

### Step 1: Set Up the Development Environment

1. **Install Node.js and npm**:

Ensure you have Node.js and npm installed. You can download them from [nodejs.org](https://nodejs.org/).

2. **Install Python and Flask**:

Make sure Python is installed. You can download it from [python.org](https://www.python.org/). Then, install Flask using pip.

```bash
pip install flask flask-cors flask-socketio
```

3. **Install MySQL**:

Download and install MySQL from [mysql.com](https://www.mysql.com/). Ensure you have a MySQL server running.

4. **Install Redis**:

Download and install Redis from [redis.io](https://redis.io/).

### Step 2: Set Up the MySQL Database

1. **Create a Database and Tables**:

Use the following SQL commands to create a database and tables for storing cryptocurrency data:

```sql
   CREATE DATABASE crypto_arbitrage;

   USE crypto_arbitrage;

   CREATE TABLE exchanges (
     id INT AUTO_INCREMENT PRIMARY KEY,
     name VARCHAR(255) NOT NULL
   );

   CREATE TABLE cryptocurrencies (
     id INT AUTO_INCREMENT PRIMARY KEY,
     name VARCHAR(255) NOT NULL,
     symbol VARCHAR(10) NOT NULL
   );

   CREATE TABLE prices (
     id INT AUTO_INCREMENT PRIMARY KEY,
     exchange_id INT,
     cryptocurrency_id INT,
     price DECIMAL(18, 8),
     timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
     FOREIGN KEY (exchange_id) REFERENCES exchanges(id),
     FOREIGN KEY (cryptocurrency_id) REFERENCES cryptocurrencies(id)
   );
   ```

### Step 3: Backend Setup with Flask

1. **Create Flask Project Structure**:
   Create a directory structure for your Flask project.

```
flask-backend/
├── app.py
├── config.py
├── models.py
├── requirements.txt
├── static/
├── templates/
├── arbitrage.py
├── database.py
└── routes.py
```

2. **Install Required Libraries**:

Create a `requirements.txt` file and add the required libraries.

```txt
flask
flask-cors
flask-socketio
mysql-connector-python
redis
```

Install the dependencies:

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

3. **Configure the Database Connection**:

   Create a `config.py` file to store configuration settings.

   ```python
   import os

   class Config:
       SECRET_KEY = os.urandom(24)
       MYSQL_HOST = 'localhost'
       MYSQL_USER = 'root'
       MYSQL_PASSWORD = 'yourpassword'
       MYSQL_DB = 'crypto_arbitrage'
       MYSQL_CURSORCLASS = 'DictCursor'
   ```

4. **Database Setup**:

   Create a `database.py` file to set up the database connection.

   ```python
   from flask import Flask
   from flask_mysqldb import MySQL

   mysql = MySQL()

   def init_db(app):
       app.config['MYSQL_HOST'] = 'localhost'
       app.config['MYSQL_USER'] = 'root'
       app.config['MYSQL_PASSWORD'] = 'yourpassword'
       app.config['MYSQL_DB'] = 'crypto_arbitrage'
       app.config['MYSQL_CURSORCLASS'] = 'DictCursor'
       mysql.init_app(app)
   ```

5. **Create Models**:

   Create a `models.py` file to define database models.

   ```python
   from .database import mysql

   def get_exchanges():
       cursor = mysql.connection.cursor()
       cursor.execute("SELECT * FROM exchanges")
       exchanges = cursor.fetchall()
       cursor.close()
       return exchanges

   def get_cryptocurrencies():
       cursor = mysql.connection.cursor()
       cursor.execute("SELECT * FROM cryptocurrencies")
       cryptocurrencies = cursor.fetchall()
       cursor.close()
       return cryptocurrencies

   def get_prices():
       cursor = mysql.connection.cursor()
       cursor.execute("SELECT * FROM prices")
       prices = cursor.fetchall()
       cursor.close()
       return prices
   ```

6. **Arbitrage Algorithm**:

Create an `arbitrage.py` file to implement the arbitrage algorithm.

```python
def find_arbitrage_opportunities(prices):
    opportunities = []
    for crypto in prices:
        buy_prices = sorted(prices[crypto], key=lambda x: x['price'])
        sell_prices = sorted(prices[crypto], key=lambda x: x['price'], reverse=True)
        for buy in buy_prices:
            for sell in sell_prices:
                if sell['price'] > buy['price']:
                    profit = sell['price'] - buy['price']
                    if profit > 0:
                        opportunities.append({
                            'buy_exchange': buy['exchange'],
                            'sell_exchange': sell['exchange'],
                            'cryptocurrency': crypto,
                            'buy_price': buy['price'],
                            'sell_price': sell['price'],
                            'profit': profit
                        })
    return opportunities
```

7. **Create Flask Routes**:

   Create a `routes.py` file to define API routes.

```python
from flask import Flask, request, jsonify
from flask_cors import CORS
from flask_socketio import SocketIO
from .config import Config
from .database import init_db
from .models import get_exchanges, get_cryptocurrencies, get_prices
from .arbitrage import find_arbitrage_opportunities

app = Flask(__name__)
app.config.from_object(Config)
CORS(app)
socketio = SocketIO(app, cors_allowed_origins="*")

init_db(app)

@app.route('/exchanges', methods=['GET'])
def exchanges():
    return jsonify(get_exchanges())

@app.route('/cryptocurrencies', methods=['GET'])
def cryptocurrencies():
    return jsonify(get_cryptocurrencies())

@app.route('/prices', methods=['GET'])
def prices():
    prices = get_prices()
    return jsonify(prices)

@app.route('/arbitrage', methods=['POST'])
def arbitrage():
    prices = get_prices()
    opportunities = find_arbitrage_opportunities(prices)
    return jsonify(opportunities)
```

  if __name__ == '__main__':
    socketio.run(app, debug=True)

8. **Run the Flask Server**:
  Run the Flask server.

  ```bash
  python app.py
  ```

### Step 4: Frontend Setup with ReactJS

1. **Create React Project**:
  Create a new React project.

  ```bash
  npx create-react-app crypto-arbitrage
  cd crypto-arbitrage
  ```

2. **Install Tailwind CSS**:
  Install Tailwind CSS and its peer dependencies.

  ```bash
  npm install -D tailwindcss postcss autoprefixer
  npx tailwindcss init -p
  ```

3. **Configure Tailwind CSS**:
  Add the paths to all of your template files in your `tailwind.config.js` file.

```javascript
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    "./src/**/*.{js,jsx,ts,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

Add the Tailwind directives to your CSS file (`src/index.css`).

```css
@tailwind base;
@tailwind components;
@tailwind utilities;
```

4. **Create Components**:
   Create the necessary components (`CreateBotForm` and `Dashboard`) inside the `src/components` directory.

   **CreateBotForm.js**:
```jsx
import React, { useState, useEffect } from 'react';
import axios from 'axios';
```

```
const CreateBotForm = () => {
 const [exchanges, setExchanges] = useState([]);
 const [cryptocurrencies, setCryptocurrencies] = useState([]);
 const [selectedExchange, setSelectedExchange] = useState('');
 const [selectedCryptocurrency, setSelectedCryptocurrency] = useState('');
 const [investment, setInvestment] = useState(1000);
 const [maxOpenPositions, setMaxOpenPositions] = useState(20);
 const [riskLevel, setRiskLevel] = useState(0);
 const [takeProfit, setTakeProfit] = useState('');
 const [orderType, setOrderType] = useState('Auto');

 useEffect(() => {
  const fetchExchanges = async () => {
   const response = await axios.get('http://localhost:5000/exchanges');
   setExchanges(response.data);
  };

  const fetchCryptocurrencies = async () => {
   const response = await axios.get('http://localhost:5000/cryptocurrencies');
   setCryptocurrencies(response.data);
  };

  fetchExchanges();
  fetchCryptocurrencies();
 }, []);

 const handleSubmit = async (e) => {
  e.preventDefault();
  const botData = {
   exchange: selectedExchange,
   cryptocurrency: selectedCryptocurrency,
      investment,
      maxOpenPositions,
      riskLevel,
      takeProfit,
      orderType

,
    };

    try {
      const response = await axios.post('http://localhost:5000/create-bot', botData);
      alert(response.data.message);
    } catch (error) {
      console.error(error);
      alert('Error creating bot');
    }
  };

  return (
    <div className="max-w-md mx-auto bg-white p-6 rounded-lg shadow-md">
      <h2 className="text-2xl font-bold mb-4">Create AI Bot</h2>
      <form onSubmit={handleSubmit}>
        <div className="mb-4">
          <label className="block text-gray-700 text-sm font-bold mb-2">
            Exchange
          </label>
          <select
            value={selectedExchange}
            onChange={(e) => setSelectedExchange(e.target.value)}
            className="block w-full bg-white border border-gray-400 hover:border-gray-500 px-4 py-2
pr-8 rounded shadow leading-tight focus:outline-none focus:shadow-outline"
```

```
            >
              <option value="">Select Exchange</option>
              {exchanges.map((exchange) => (
                <option key={exchange.id} value={exchange.name}>
                  {exchange.name}
                </option>
              ))}
            </select>
          </div>
          <div className="mb-4">
            <label className="block text-gray-700 text-sm font-bold mb-2">
              Cryptocurrency
            </label>
            <select
              value={selectedCryptocurrency}
              onChange={(e) => setSelectedCryptocurrency(e.target.value)}
              className="block w-full bg-white border border-gray-400 hover:border-gray-500 px-4 py-2
pr-8 rounded shadow leading-tight focus:outline-none focus:shadow-outline"
            >
              <option value="">Select Cryptocurrency</option>
              {cryptocurrencies.map((crypto) => (
                <option key={crypto.id} value={crypto.name}>
                  {crypto.name}
                </option>
              ))}
            </select>
          </div>
          <div className="mb-4">
            <label className="block text-gray-700 text-sm font-bold mb-2">
              Investment Amount (USDT)
            </label>
```

```
      <input
        type="number"
        value={investment}
        onChange={(e) => setInvestment(e.target.value)}
        className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700
leading-tight focus:outline-none focus:shadow-outline"
      />
     </div>
     <div className="mb-4">
      <label className="block text-gray-700 text-sm font-bold mb-2">
        Max Open Positions
      </label>
      <input
        type="number"
        value={maxOpenPositions}
        onChange={(e) => setMaxOpenPositions(e.target.value)}
        className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700
leading-tight focus:outline-none focus:shadow-outline"
      />
     </div>
     <div className="mb-4">
      <label className="block text-gray-700 text-sm font-bold mb-2">
        Risk Level
      </label>
      <input
        type="range"
        min="0"
        max="10"
        value={riskLevel}
        onChange={(e) => setRiskLevel(e.target.value)}
        className="w-full"
      />
```

```
        </div>
        <div className="mb-4">
          <label className="block text-gray-700 text-sm font-bold mb-2">
            Trade Take Profit (%)
          </label>
          <input
            type="number"
            value={takeProfit}
            onChange={(e) => setTakeProfit(e.target.value)}
            className="shadow appearance-none border rounded w-full py-2 px-3 text-gray-700
leading-tight focus:outline-none focus:shadow-outline"
          />
        </div>
        <div className="mb-4">
          <label className="block text-gray-700 text-sm font-bold mb-2">
            Order Type
          </label>
          <div className="flex">
            <label className="mr-4">
              <input
                type="radio"
                value="Auto"
                checked={orderType === 'Auto'}
                onChange={(e) => setOrderType(e.target.value)}
                className="mr-2"
              />
              Auto
            </label>
            <label className="mr-4">
              <input
                type="radio"
                value="Market"
                checked={orderType === 'Market'}
                onChange={(e) => setOrderType(e.target.value)}
                className="mr-2"
              />
              Market
            </label>
            <label>
              <input
                type="radio"
                value="Two Leg Entry"
                checked={orderType === 'Two Leg Entry'}
                onChange={(e) => setOrderType(e.target.value)}
                className="mr-2"
              />
              Two Leg Entry
            </label>
          </div>
        </div>
        <button
          type="submit"
          className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2 px-4 rounded
focus:outline-none focus:shadow-outline"
        >
          Create
        </button>
       </form>
</div>
);
};
```

export default CreateBotForm;

5. **Create the Dashboard Component**:
   Create a `Dashboard.js` file for displaying arbitrage opportunities.

```jsx
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const Dashboard = () => {
  const [opportunities, setOpportunities] = useState([]);

  useEffect(() => {
    const fetchOpportunities = async () => {
      const response = await axios.post('http://localhost:5000/arbitrage');
      setOpportunities(response.data);
    };

    fetchOpportunities();
  }, []);

  return (
    <div className="max-w-4xl mx-auto mt-6">
      <h2 className="text-2xl font-bold mb-4">Arbitrage Opportunities</h2>
      <table className="min-w-full bg-white">
        <thead>
          <tr>
            <th className="py-2">Buy Exchange</th>
            <th className="py-2">Sell Exchange</th>
            <th className="py-2">Cryptocurrency</th>
      <th className="py-2">Buy Price</th>
      <th className="py-2">Sell Price</th>
      <th className="py-2">Profit</th>
     </tr>
    </thead>
    <tbody>
     {opportunities.map((opportunity, index) => (
      <tr key={index}>
       <td className="py-2">{opportunity.buy_exchange}</td>
       <td className="py-2">{opportunity.sell_exchange}</td>
       <td className="py-2">{opportunity.cryptocurrency}</td>
       <td className="py-2">{opportunity.buy_price}</td>
       <td className="py-2">{opportunity.sell_price}</td>
       <td className="py-2">{opportunity.profit}</td>
      </tr>
     ))}
    </tbody>
   </table>
  </div>
 );
};

export default Dashboard;
```

6. **Set Up Routing**:
 Modify `App.js` to include routing for the components.

```jsx
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import CreateBotForm from './components/CreateBotForm';
import Dashboard from './components/Dashboard';
import './index.css';

const App = () => {
  return (
    <Router>
      <div className="container mx-auto p-4">
        <Switch>
          <Route path="/" exact component={CreateBotForm} />
          <Route path="/dashboard" component={Dashboard} />
        </Switch>
      </div>
    </Router>
  );
};

export default App;
```

7. **Run the React Application**:
   Start the React development server.

   ```bash
   npm start
   ```

### Step 5: Connect Frontend with Backend

1. **Enable CORS in Flask**:
   Ensure CORS is enabled in your Flask app.

```python
  from flask_cors import CORS
  app = Flask(__name__)
  CORS(app)
  ```

2. **Update React Axios Requests**:

  Ensure that all Axios requests in the React components point to the correct Flask backend URL (e.g., `http://localhost:5000`).

### Step 6: Implement Real-Time Updates with WebSockets

1. **Set Up Flask-SocketIO**:

  Modify `app.py` to include Flask-SocketIO for real-time updates.

  ```python
  from flask_socketio import SocketIO

  socketio = SocketIO(app, cors_allowed_origins="*")

  @socketio.on('connect')
  def handle_connect():
      print('Client connected')

  @socketio.on('disconnect')
  def handle_disconnect():
      print('Client disconnected')

  if __name__ == '__main__':
      socketio.run(app, debug=True)
  ```

2. **Integrate Socket.IO in React**:

Install `socket.io-client` in your React app.

```bash
npm install socket.io-client
```

Update your React components to use Socket.IO for real-time updates.