CyberSecure AI - Wireframe UI/UX Plan & Replit.com Development Guide

CyberSecure AI - Wireframe UI/UX Plan & Replit.com Development Guide

Table of Contents

Part 1: UI/UX Wireframe Plan

1. Core Design Principles

Brand Identity & Visual Language

The CyberSecure AI platform will follow these design principles:

Color Scheme	Typography	Visual Style
Primary: Midnight Blue (#0D3B66)Secondary: Spring Green (#FAF0CA)Warning: Dandelion (#F4D35E)Interactive: Neon Carrot (#EE964B)Critical: Red Orange (#F95738)	Primary: Work Sans font familyHeaders: Bold, clean with proper hierarchyBody: Regular weight for readabilityMonospace: For code and technical details	High-tech environments with abstract digital gridsSharp, clean interfaces with digital overlaysGlowing effects for active/important elementsAI-inspired visualizations with blue/orange highlights

Accessibility & Security Requirements

- Dark mode support for reduced eye strain
- Colorblind-friendly visualization schemes for threat severity
- Clear visual indicators for secure/insecure states
- Anti-phishing design patterns
- WCAG 2.1 AA compliance
- Role-based visual indicators

2. User Roles & Personas

Education Sector

Role	Primary Needs	Key Dashboard Elements
Students	Personal security statusPrivacy controls	Account security dashboardPrivacy settingsOwn data visibility only
Faculty	Class data protectionThreat alerts	Classroom security overviewStudent data protection statusClass-specific data visibility
IT Administrators	Full security operationsSystem management	Threat detectionIncident responseInstitution-wide visibility
Compliance Officers	Regulatory complianceAudit tracking	FERPA compliance statusAudit trailsViolation reports

Government Sector

Role	Primary Needs	Key Dashboard Elements
Citizens	Personal account securityService status	Account security dashboardService status indicatorsPersonal data visibility only
Department Staff	Departmental securityAlerts and notifications	Department security dashboardSystem status indicatorsDepartment-specific data

Security Officers	Comprehensive securityIncident management	Threat hunting toolsIncident management consoleForensic analysis
Executives	High-level security metricsStrategic overview	Executive dashboardRisk assessmentOrganization-wide summaries

3. Key Screens & Components

1. Login & Authentication Flow

Key Components:

- Multi-factor authentication interface with biometric options
- Progressive security based on access level requirements
- Account recovery process with security verification
- Anti-phishing visual indicators
- Role-based login paths

2. Main Dashboard (Role-Based)

Key Components:

- Security status overview with threat level indicators
- Real-time alert feed with severity classification
- Quick action buttons for common security tasks
- Role-specific metrics and visualization widgets
- System health indicators

3. Threat Detection & Monitoring

Key Components:

- Interactive threat map showing attack origins
- Behavioral analysis visualizations
- Anomaly detection indicators
- ML-powered threat classification display
- Historical trend analysis charts

4. Incident Response Interface

Key Components:

- Incident timeline visualization
- Response workflow management tools
- Evidence collection and documentation interface
- Automated response action controls
- Collaboration workspace for security teams

5. Compliance Management Dashboard

Key Components:

- Regulatory framework status indicators (FERPA, FISMA, FedRAMP, CIPA)
- Compliance score visualizations
- Control implementation status
- Audit trail and reporting tools
- Remediation tracking interface

6. Admin Control Panel

Key Components:

- User management interface with role assignment
- Policy configuration tools
- System-wide settings and controls
- AI model configuration options
- Integration management for external systems

7. AI Configuration Interface

Key Components:

- ML model training controls
- Threshold configuration for detection sensitivity
- Behavioral baseline management
- Model performance metrics
- Custom rule creation interface

4. User Flows

Threat Detection to Response Workflow

1. System detects potential threat (Automated)
2. Alert generated with severity classification
3. Security analyst reviews alert details
4. Analyst initiates investigation
5. Evidence collection and analysis

6. Response action selection and execution
7. Incident documentation and resolution
8. Post-incident review and improvement

Compliance Violation Workflow

1. System detects compliance violation (Automated)
2. Compliance officer receives notification
3. Officer reviews violation details
4. Investigation initiated to determine cause
5. Remediation plan created
6. Implementation of corrective actions
7. Verification of compliance restoration
8. Documentation and reporting

User Onboarding Flow

1. Initial account creation
2. Role assignment and permission configuration
3. Security profile setup
4. MFA configuration
5. Dashboard orientation
6. Feature introduction based on role
7. Initial security assessment
8. Personalization options

5. Responsive Design Considerations

Device Support Matrix

Device Type	Screen Size	Layout Adaptation	Feature Availability

Desktop	1920×1080+	Full dashboard layout	Complete feature set
Laptop	1366×768+	Optimized dashboard with collapsible panels	Complete feature set
Tablet	768×1024	Condensed sidebar, touch-optimized	Core security functions
Mobile	375×667	Single-column layout, essential features	Emergency response only
Large Display	2560×1440+	Extended dashboard, multiple panels	SOC operations center

Mobile-First Considerations

- Touch-friendly interface elements
- Simplified navigation for smaller screens
- Progressive disclosure of complex features
- Critical alerts and functions prioritized on mobile
- Offline capabilities for essential security functions

Part 2: Replit.com Development Guide

1. Setting Up Your Replit Environment

Creating Your CyberSecure AI Project

1. Sign up or log in to Replit.com
2. Create a new Repl using "Create a Repl" button
3. Select the appropriate template:

- For frontend: Choose "React.js" template
- For backend: Choose "Node.js" or "Python" based on your preference

4. Name your project "CyberSecure-AI" or appropriate name
5. Configure your Repl settings for team collaboration if needed

Environment Configuration

Use Replit's Secrets (environment variables) to store sensitive information:

```
# Required secrets to configure in Replit
API_KEY=your_api_key
DB_CONNECTION_STRING=your_db_connection
AUTH_SECRET=your_auth_secret
AI_MODEL_ENDPOINT=your_ai_endpoint
```

Project Structure

```
CyberSecure-AI/
├── frontend/           # React frontend
│   ├── public/         # Static assets
│   ├── src/            # React source code
│   │   ├── components/   # UI components
│   │   ├── pages/        # Page layouts
│   │   ├── contexts/     # React contexts
│   │   ├── hooks/        # Custom hooks
│   │   ├── services/     # API services
│   │   ├── utils/        # Utility functions
│   │   ├── styles/       # CSS/SCSS files
│   │   └── App.js        # Main component
├── backend/            # Node.js/Python backend
│   ├── controllers/      # Request handlers
│   ├── models/         # Data models
│   ├── routes/         # API routes
│   ├── services/       # Business logic
│   ├── utils/          # Utility functions
│   ├── middleware/       # Custom middleware
│   └── server.js       # Entry point
├── ai/                 # AI models and scripts
│   ├── models/         # Trained models
│   ├── training/       # Training scripts
```

```
│   └── inference/      # Inference scripts
└── docs/               # Documentation
```

2. Development Stages on Replit

Stage 1: Frontend Prototyping (1-2 weeks)

Objectives:

- Set up React.js project structure
- Implement core UI components based on wireframes
- Create responsive layouts for different devices
- Establish design system with color schemes and typography

Replit Implementation Steps:

1. Use Replit's React template to create your frontend project
2. Install required dependencies:

```
npm install react-router-dom @mui/material @emotion/react @emotion/styled
npm install chart.js react-chartjs-2 d3 axios
npm install react-hook-form zod @hookform/resolvers
```

3. Set up routing structure using React Router:

```
// src/App.js
import { BrowserRouter, Routes, Route } from 'react-router-dom';
import Dashboard from './pages/Dashboard';
import Login from './pages/Login';
import ThreatDetection from './pages/ThreatDetection';
import IncidentResponse from './pages/IncidentResponse';
import Compliance from './pages/Compliance';
import AdminPanel from './pages/AdminPanel';
import AIConfiguration from './pages/AIConfiguration';
```

```
function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/login" element={<Login />} />
        <Route path="/dashboard" element={<Dashboard />} />
        <Route path="/threat-detection" element={<ThreatDetection />} />
        <Route path="/incident-response" element={<IncidentResponse />} />
        <Route path="/compliance" element={<Compliance />} />
        <Route path="/admin" element={<AdminPanel />} />
        <Route path="/ai-config" element={<AIConfiguration />} />
        <Route path="/" element={<Login />} />
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

4. Create reusable UI components based on wireframes:

```
// src/components/SecurityDashboard.js
import React from 'react';
import { Card, CardContent, Typography, Grid, Box } from '@mui/material';
import { Doughnut } from 'react-chartjs-2';

const SecurityDashboard = ({ threatLevel, alerts, systemHealth }) => {
  // Chart data setup for threat level visualization
  const threatData = {
    labels: ['Critical', 'High', 'Medium', 'Low'],
    datasets: [{
      data: [threatLevel.critical, threatLevel.high, threatLevel.medium, threatLev
el.low],
      backgroundColor: ['#F95738', '#EE964B', '#F4D35E', '#FAF0CA'],
      borderColor: '#0D3B66',
```

```
    }]
  };

  return (
    <Box sx={{ p: 3 }}>
      <Typography variant="h4" gutterBottom>Security Dashboard</Typography>
      <Grid container spacing={3}>
        <Grid item xs={12} md={4}>
          <Card>
            <CardContent>
              <Typography variant="h6">Threat Level</Typography>
              <Doughnut data={threatData} />
            </CardContent>
          </Card>
        </Grid>
        {/* Additional dashboard components would go here */}
      </Grid>
    </Box>
  );
};

export default SecurityDashboard;
```

Stage 2: Backend API Development (2-3 weeks)

Objectives:

- Set up Node.js/Express or Python/Flask backend
- Implement authentication and authorization
- Create core API endpoints for security features
- Set up database integration

Replit Implementation Steps:

1. Create a new Repl for the backend using Node.js template

2. Install backend dependencies:

```
npm install express cors helmet jsonwebtoken bcrypt mongoose dotenv
npm install express-validator morgan winston
```

3. Create basic server setup:

```
// server.js
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
const mongoose = require('mongoose');
require('dotenv').config();

// Import routes
const authRoutes = require('./routes/auth');
const threatRoutes = require('./routes/threats');
const incidentRoutes = require('./routes/incidents');
const complianceRoutes = require('./routes/compliance');

// Create Express app
const app = express();

// Middleware
app.use(helmet()); // Security headers
app.use(cors());
app.use(express.json());

// Connect to MongoDB
mongoose.connect(process.env.DB_CONNECTION_STRING)
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.error('MongoDB connection error:', err));

// Routes
app.use('/api/auth', authRoutes);
```

```
app.use('/api/threats', threatRoutes);
app.use('/api/incidents', incidentRoutes);
app.use('/api/compliance', complianceRoutes);

// Error handler
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something broke!');
});

// Start server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

4. Implement authentication with JWT:

```
// routes/auth.js
const express = require('express');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const { check, validationResult } = require('express-validator');
const User = require('../models/User');

const router = express.Router();

// Register user
router.post(
  '/register',
  [
    check('email', 'Please include a valid email').isEmail(),
    check('password', 'Password must be 6 or more characters').isLength({ mi
n: 6 }),
    check('role', 'Role is required').not().isEmpty()
```

```
  ],
  async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    const { email, password, role } = req.body;

    try {
      // Check if user exists
      let user = await User.findOne({ email });
      if (user) {
        return res.status(400).json({ msg: 'User already exists' });
      }

      // Create new user
      user = new User({
        email,
        password,
        role
      });

      // Hash password
      const salt = await bcrypt.genSalt(10);
      user.password = await bcrypt.hash(password, salt);

      // Save user
      await user.save();

      // Create JWT
      const payload = {
        user: {
          id: user.id,
          role: user.role
        }
```

```
      };

      jwt.sign(
        payload,
        process.env.JWT_SECRET,
        { expiresIn: '1h' },
        (err, token) => {
          if (err) throw err;
          res.json({ token });
        }
      );
    } catch (err) {
      console.error(err.message);
      res.status(500).send('Server error');
    }
  }
);

// Login user
router.post(
  '/login',
  [
    check('email', 'Please include a valid email').isEmail(),
    check('password', 'Password is required').exists()
  ],
  async (req, res) => {
    // Similar implementation to register but for login
    // ...
  }
);

module.exports = router;
```

Stage 3: AI Model Integration (2-3 weeks)

Objectives:

- Implement core AI/ML models for threat detection
- Set up model training and inference pipelines
- Create API endpoints for AI-powered features
- Implement basic behavioral analysis

Replit Implementation Steps:

1. Create AI model scripts using Python/TensorFlow:

```
# ai/models/threat_detection.py
import tensorflow as tf
import numpy as np

class ThreatDetectionModel:
    def __init__(self, model_path=None):
        if model_path:
            self.model = tf.keras.models.load_model(model_path)
        else:
            self.model = self._build_model()

    def _build_model(self):
        # Simple example model - would be more complex in production
        model = tf.keras.Sequential([
            tf.keras.layers.Dense(128, activation='relu', input_shape=(50,)),
            tf.keras.layers.Dropout(0.2),
            tf.keras.layers.Dense(64, activation='relu'),
            tf.keras.layers.Dropout(0.2),
            tf.keras.layers.Dense(4, activation='softmax')  # 4 classes: critical, hig
h, medium, low
        ])

        model.compile(
            optimizer='adam',
            loss='categorical_crossentropy',
            metrics=['accuracy']
```

```
        )

        return model

    def train(self, X_train, y_train, epochs=10, batch_size=32, validation_split=0.
2):
        history = self.model.fit(
            X_train, y_train,
            epochs=epochs,
            batch_size=batch_size,
            validation_split=validation_split
        )
        return history

    def predict(self, data):
        predictions = self.model.predict(data)
        return predictions

    def save_model(self, path):
        self.model.save(path)
```

2. Create API endpoints to use the AI models:

```
// routes/ai.js
const express = require('express');
const { spawn } = require('child_process');
const auth = require('../middleware/auth');
const router = express.Router();

// Endpoint to analyze threat data
router.post('/analyze-threat', auth, (req, res) => {
  const { data } = req.body;

  // Call Python script for prediction
  const python = spawn('python', ['./ai/inference/predict.py', JSON.stringify(da
```

```
ta)]);

  let predictionData = '';

  // Collect data from script
  python.stdout.on('data', (data) => {
    predictionData += data.toString();
  });

  // Handle end of script execution
  python.on('close', (code) => {
    if (code !== 0) {
      return res.status(500).json({ error: 'Prediction failed' });
    }

    try {
      const prediction = JSON.parse(predictionData);
      res.json(prediction);
    } catch (err) {
      res.status(500).json({ error: 'Failed to parse prediction data' });
    }
  });
});

module.exports = router;
```

Stage 4: Frontend-Backend Integration (1-2 weeks)

Objectives:

- Connect frontend to backend API endpoints
- Implement authentication flow
- Create data visualization with real data
- Set up real-time updates

Replit Implementation Steps:

1. Set up API service in the frontend:

```
// src/services/api.js
import axios from 'axios';

const API_URL = process.env.REACT_APP_API_URL || 'https://backend-url.repl.co/api';

// Create axios instance
const api = axios.create({
  baseURL: API_URL,
  headers: {
    'Content-Type': 'application/json'
  }
});

// Add JWT token to requests
api.interceptors.request.use(
  config => {
    const token = localStorage.getItem('token');
    if (token) {
      config.headers['Authorization'] = `Bearer ${token}`;
    }
    return config;
  },
  error => {
    return Promise.reject(error);
  }
);

// Authentication service
export const authService = {
  login: (credentials) => api.post('/auth/login', credentials),
  register: (userData) => api.post('/auth/register', userData),
  getCurrentUser: () => api.get('/auth/me')
```

```
};

// Threat detection service
export const threatService = {
  getThreats: () => api.get('/threats'),
  analyzeThreat: (data) => api.post('/ai/analyze-threat', { data })
};

// Incident response service
export const incidentService = {
  getIncidents: () => api.get('/incidents'),
  createIncident: (incident) => api.post('/incidents', incident),
  updateIncident: (id, updates) => api.put(`/incidents/${id}`, updates)
};

// Compliance service
export const complianceService = {
  getComplianceStatus: () => api.get('/compliance/status'),
  getFrameworks: () => api.get('/compliance/frameworks')
};

export default api;
```

2. Implement authentication context:

```
// src/contexts/AuthContext.js
import React, { createContext, useState, useEffect } from 'react';
import { authService } from '../services/api';

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [currentUser, setCurrentUser] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
```

```
useEffect(() => {
  const token = localStorage.getItem('token');
  if (token) {
    authService.getCurrentUser()
      .then(response => {
        setCurrentUser(response.data);
      })
      .catch(err => {
        localStorage.removeItem('token');
        setError(err);
      })
      .finally(() => {
        setLoading(false);
      });
  } else {
    setLoading(false);
  }
}, []);

const login = async (email, password) => {
  try {
    const response = await authService.login({ email, password });
    localStorage.setItem('token', response.data.token);
    const userResponse = await authService.getCurrentUser();
    setCurrentUser(userResponse.data);
    return userResponse.data;
  } catch (err) {
    setError(err.response?.data?.message || 'Login failed');
    throw err;
  }
};

const logout = () => {
  localStorage.removeItem('token');
  setCurrentUser(null);
```

```
  };

  const value = {
    currentUser,
    loading,
    error,
    login,
    logout
  };

  return (
    <AuthContext.Provider value={value}>
      {children}
    </AuthContext.Provider>
  );
};
```

Stage 5: Advanced Features & Refinement (2-3 weeks)

Objectives:

- Implement real-time alerts and notifications
- Enhance AI model performance
- Add advanced visualizations
- Implement role-based access control
- Refine UI/UX based on testing

Replit Implementation Steps:

1. Implement real-time updates with Socket.io:

```
// Backend: server.js (additional code)
const http = require('http');
const socketIo = require('socket.io');

const server = http.createServer(app);
```

```
const io = socketIo(server, {
  cors: {
    origin: process.env.FRONTEND_URL,
    methods: ["GET", "POST"]
  }
});

// Socket.io middleware for authentication
io.use((socket, next) => {
  const token = socket.handshake.auth.token;
  if (!token) {
    return next(new Error('Authentication error'));
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    socket.user = decoded.user;
    next();
  } catch (err) {
    next(new Error('Authentication error'));
  }
});

// Socket.io connection handling
io.on('connection', (socket) => {
  console.log(`User connected: ${socket.user.id}`);

  // Join room based on user role
  socket.join(socket.user.role);

  // Handle disconnect
  socket.on('disconnect', () => {
    console.log(`User disconnected: ${socket.user.id}`);
  });
});
```

```
// Function to emit security alerts
const emitSecurityAlert = (alert) => {
  // Emit to specific roles based on alert severity
  if (alert.severity === 'critical') {
    io.to('admin').to('security').emit('security-alert', alert);
  } else {
    io.to(alert.targetRole).emit('security-alert', alert);
  }
};

// Export for use in other files
app.set('io', io);
app.set('emitSecurityAlert', emitSecurityAlert);

server.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

2. Frontend Socket.io integration:

```
// src/services/socket.js
import { io } from 'socket.io-client';

let socket;

export const initSocket = (token) => {
  socket = io(process.env.REACT_APP_SOCKET_URL, {
    auth: { token }
  });

  return socket;
};

export const getSocket = () => {
  if (!socket) {
```

```
    throw new Error('Socket not initialized');
  }
  return socket;
};

export const closeSocket = () => {
  if (socket) socket.close();
};
```

3. Implement role-based access control middleware:

```
// middleware/roleAuth.js
const roleAuth = (roles = []) => {
  if (typeof roles === 'string') {
    roles = [roles];
  }

  return (req, res, next) => {
    if (!req.user) {
      return res.status(401).json({ message: 'Unauthorized' });
    }

    if (roles.length && !roles.includes(req.user.role)) {
      return res.status(403).json({ message: 'Forbidden' });
    }

    next();
  };
};

module.exports = roleAuth;
```

Stage 6: Testing & Deployment (1-2 weeks)

Objectives:

- Implement comprehensive testing
- Set up CI/CD pipeline
- Deploy to Replit production environment
- Implement monitoring and logging

Replit Implementation Steps:

1. Set up testing with Jest:

```
npm install --save-dev jest supertest
```

```
// tests/auth.test.js
const request = require('supertest');
const app = require('../server');
const mongoose = require('mongoose');
const User = require('../models/User');

beforeAll(async () => {
  // Connect to test database
  await mongoose.connect(process.env.TEST_DB_CONNECTION_STRING);
});

afterAll(async () => {
  // Clean up and close connection
  await User.deleteMany({});
  await mongoose.connection.close();
});

describe('Auth API', () => {
  it('should register a new user', async () => {
    const res = await request(app)
      .post('/api/auth/register')
      .send({
        email: 'test@example.com',
        password: 'password123',
```

```
        role: 'user'
      });

    expect(res.statusCode).toEqual(200);
    expect(res.body).toHaveProperty('token');
  });

  it('should login a user', async () => {
    const res = await request(app)
      .post('/api/auth/login')
      .send({
        email: 'test@example.com',
        password: 'password123'
      });

    expect(res.statusCode).toEqual(200);
    expect(res.body).toHaveProperty('token');
  });
});
```

2. Deploy on Replit:

- Use Replit's built-in hosting features
- Set up environment variables in Replit Secrets
- Configure custom domain if needed

Part 3: Implementation Timeline

Project Timeline Overview

Phase	Duration	Key Milestones
Research & Planning	2 weeks	Requirements gathering, technology selection, architecture design
UI/UX Design	2 weeks	Wireframes, prototypes, user flow design

Frontend Development	3 weeks	Core UI components, responsive layouts, state management
Backend Development	4 weeks	API endpoints, authentication, database integration
AI Model Development	4 weeks	ML models for threat detection, behavioral analysis
Integration & Testing	3 weeks	Frontend-backend integration, comprehensive testing
Refinement & Optimization	2 weeks	Performance optimization, UI/UX refinement
Deployment & Launch	1 week	Production deployment, monitoring setup

Detailed Timeline

Month 1: Foundation

- Weeks 1-2: Research, planning, and initial UI/UX design
- Weeks 3-4: Frontend prototyping and basic component development

Month 2: Core Development

- Weeks 1-2: Backend API development and database setup
- Weeks 3-4: Authentication system and basic security features

Month 3: AI Integration

- Weeks 1-2: AI model development for threat detection
- Weeks 3-4: Integration of AI models with backend APIs

Month 4: Feature Completion

- Weeks 1-2: Frontend-backend integration and real-time features
- Weeks 3-4: Advanced features implementation and refinement

Month 5: Testing & Launch

- Weeks 1-2: Comprehensive testing and bug fixing

- Weeks 3-4: Performance optimization and production deployment

Resource Allocation

Role	Responsibilities	Allocation
UI/UX Designer	Wireframes, prototypes, user flows	Full-time for 2 months
Frontend Developer	React components, responsive design	Full-time for 4 months
Backend Developer	API development, database integration	Full-time for 4 months
AI/ML Engineer	Model development, integration	Full-time for 3 months
QA Engineer	Testing, bug tracking	Full-time for 2 months
DevOps Engineer	Deployment, CI/CD, monitoring	Part-time for 5 months

Delivery Milestones

- **Milestone 1:** UI/UX design approval (End of Month 1)
- **Milestone 2:** Functional prototype with core features (End of Month 2)
- **Milestone 3:** AI-powered threat detection integration (Mid-Month 3)
- **Milestone 4:** Complete feature set with testing (End of Month 4)
- **Milestone 5:** Production deployment and launch (End of Month 5)

Next Steps:

- Set up Replit.com development environment
- Create frontend prototypes based on wireframe designs
- Implement core UI components for key screens
- Begin backend API development for authentication and basic features