Work Breakdown Structure (WBS) - STUDENT VERSION

Fraud Detection, Prevention and Reporting System for Malicious Websites

Project: Anti-Phishing System
Course: Comp1682 - Final Project
Developer: Solo Student Project
Version: 2.0 - Simplified & Realistic
Date: November 22, 2025

⚠️ Student Project Scope

Reality Check:

- ✅ Solo developer (you)
- ✅ Limited time (semester deadline)
- ✅ Academic project (not production system)
- ✅ Focus: Demonstrate understanding & core functionality
- ❌ No need for: Full production deployment, extensive testing, post-launch support

Goal: Build a **working prototype** that demonstrates the anti-phishing concept with core features

---

WBS Structure Overview

```
Level 0: Project Title
    ↓
Level 1: Major Phases (4 PHASES - Simplified from 7)
    ↓
Level 2: Deliverables
    ↓
Level 3: Essential Tasks Only
```

Simplified Work Breakdown Structure

0.0 FRAUD DETECTION, PREVENTION AND REPORTING SYSTEM FOR MALICIOUS WEBSITES

1.0 DESIGN & PLANNING ✅ COMPLETED

1.1 Requirements & Design (Already Done)

- **1.1.1** ✅ Requirements gathering (from project brief)
- **1.1.2** ✅ Database design (18 PostgreSQL + 4 DynamoDB tables)
- **1.1.3** ✅ Algorithm design (URL similarity detection)
- **1.1.4** ✅ ERD diagram creation
- **1.1.5** ✅ Wireframes (from project doc)

Status: ✅ **COMPLETE**
Deliverables:

- ✅ Database Design Document
- ✅ Algorithm Design Document
- ✅ ERD Diagram
- ✅ Data Dictionary

Duration: Already completed

2.0 DEVELOPMENT - Core Implementation

2.1 Environment Setup (Week 1)

- **2.1.1** Install required software
  - Node.js and npm
  - PostgreSQL
  - VS Code
  - Git
- **2.1.2** Create Git repository
- **2.1.3** Set up project folders
  - /backend (Node.js)
  - /frontend (React)
  - /database (SQL scripts)

2.2 Database Setup (Week 1-2)

- **2.2.1** Create local PostgreSQL database

- **2.2.2** Execute database_schema.sql (creates all 18 tables)
- **2.2.3** Create default admin user
- **2.2.4** Import sample data
  - 50-100 known phishing URLs
  - 20-30 legitimate brand domains
  - 5-10 test users
- **2.2.5** Test database queries

Optional AWS:

- Set up RDS PostgreSQL (if you want cloud deployment)
- Create DynamoDB tables (if time permits)
- **Note**: Can skip AWS entirely and use local PostgreSQL for demo

2.3 Backend API - Core Features (Week 3-5)

- **2.3.1** Set up Express.js server
- **2.3.2** Configure database connection (pg library)
- **2.3.3** Implement authentication (simple version)
  - User registration
  - User login with JWT
  - Basic role checking (admin/moderator/user)
- **2.3.4** Implement URL detection algorithm ⭐ **CORE FEATURE**
  - URL parsing and normalization
  - Levenshtein similarity function
  - Jaro-Winkler similarity function
  - Domain analysis
  - Subdomain analysis
  - Scoring system (weighted aggregation)
  - Classification (safe/suspicious/dangerous)
- **2.3.5** Implement essential API endpoints
  - POST /api/check-url (URL checking) ⭐ **MOST IMPORTANT**

    - GET /api/check-history (user's scan history)
    - POST /api/reports (submit phishing report)
    - GET /api/reports (list reports for moderators)
    - PUT /api/reports/:id/status (approve/reject reports)
    - GET /api/dashboard/stats (basic statistics)
- **2.3.6** Add basic error handling
- **2.3.7** Test API with Postman

Skip for Student Project:

- ❌ Google Safe Browsing API (optional, can mention as "future work")
- ❌ Threat feed integration (optional)
- ❌ Complex caching (local variables are fine)
- ❌ Rate limiting (not needed for demo)

2.4 Frontend Web Application (Week 6-8)

- **2.4.1** Set up React project (Create React App)
- **2.4.2** Create basic layout
    - Header with navigation
    - Footer
    - Simple responsive design
- **2.4.3** Implement authentication pages
    - Login page
    - Register page (simple form)
- **2.4.4** Implement URL check page ⭐ **CORE FEATURE**
    - URL input form
    - "Check URL" button
    - Results display showing:
        - Overall score and classification
        - Component breakdown (domain, subdomain, path scores)
        - Color-coded result (green/orange/red)

    - Scan history table
- **2.4.5** Implement report submission page
    - Simple form to report suspicious URL
    - Show auto-scan result before submission
- **2.4.6** Implement basic dashboard
    - Statistics cards (total scans, threats detected)
    - Recent scans table
    - Simple charts (optional - can use Chart.js)
- **2.4.7** Implement moderator page (simple)
    - List of pending reports
    - Approve/Reject buttons
    - View report details
- **2.4.8** Basic styling with CSS or Bootstrap
    - Clean, professional look
    - Mobile-responsive (basic)

Skip for Student Project:

- ❌ Complex animations
- ❌ Advanced state management (Redux)
- ❌ Internationalization (can just use English or Vietnamese)
- ❌ Educational content pages (can mention as "future work")
- ❌ Quiz functionality
- ❌ Advanced dashboard features

2.5 Browser Extension - Basic Version (Week 9-10) OPTIONAL

Note: Only if you have time. Can be skipped and mentioned as "future work"

- **2.5.1** Create manifest.json for Chrome
- **2.5.2** Simple background script that checks URL
- **2.5.3** Basic popup showing site status
- **2.5.4** Alert banner on suspicious sites

- **2.5.5** Test on Chrome

Or Skip Entirely: Focus on web application, mention extension as planned feature

2.6 Sample Data & Demo Preparation (Week 11)

- **2.6.1** Create realistic test data
  - Real phishing URL examples (from PhishTank)
  - Real legitimate URLs
  - Sample user accounts
  - Sample reports
- **2.6.2** Test all user flows
- **2.6.3** Take screenshots for documentation
- **2.6.4** Record demo video (5-10 minutes)
- **2.6.5** Fix critical bugs only

Deliverables:

- ✅ Working Backend API
- ✅ Working Web Application
- ✅ Populated Database
- ✅ Demo Video
- ⚠️ Browser Extension (optional)

Duration: 11 weeks total

3.0 TESTING - Essential Only

3.1 Basic Testing (Week 11-12)

- **3.1.1** Manual testing of core features
  - URL checking works correctly
  - Users can register and login
  - Reports can be submitted
  - Moderators can approve/reject
- **3.1.2** Algorithm accuracy testing
  - Test with 20-30 known phishing URLs (should detect most)

    - Test with 20-30 legitimate URLs (should NOT flag)
    - Calculate rough accuracy percentage
    - Document results
- **3.1.3** Fix critical bugs only
    - Crashes
    - Broken core features
    - Security issues (SQL injection, XSS basics)
- **3.1.4** Browser compatibility
    - Test on Chrome
    - Test on Firefox
    - Fix major issues only

Skip for Student Project:

- ❌ Unit tests (too time-consuming for student project)
- ❌ Integration tests
- ❌ Automated testing
- ❌ Performance testing
- ❌ Load testing
- ❌ Security penetration testing
- ❌ User acceptance testing with real users

Deliverables:

- ✅ Test results document (simple spreadsheet)
- ✅ Bug fix log

Duration: 1-2 weeks

4.0 DOCUMENTATION & SUBMISSION

4.1 Project Report (Week 13-14)

- **4.1.1** Write final report (40-60 pages is fine)
    - **Chapter 1: Introduction**
        - Background and motivation

    - Problem statement
    - Objectives
    - Project scope
- **Chapter 2: Literature Review**
    - Anti-phishing techniques
    - Existing solutions
    - Technology overview
- **Chapter 3: System Design**
    - System architecture diagram (simple)
    - Database design (reference your ERD)
    - Algorithm design (reference your algorithm doc)
    - User interface design (wireframes)
- **Chapter 4: Implementation**
    - Technology stack used
    - Database implementation
    - Backend implementation
    - Frontend implementation
    - Key code snippets (algorithm parts)
- **Chapter 5: Testing & Results**
    - Testing methodology
    - Test results (accuracy percentages)
    - Screenshots of working system
- **Chapter 6: Conclusion**
    - What was achieved
    - Challenges faced
    - Future work
- **References**
- **Appendices** (optional)

- Database schema
- API endpoints list
- User manual (brief)

4.2 Presentation Preparation (Week 14)

- **4.2.1** Create PowerPoint (15-20 slides)
  - Title slide
  - Problem statement (1-2 slides)
  - Solution overview (1 slide)
  - System architecture (1 slide)
  - Database design (1 slide - show ERD)
  - Algorithm explanation (2-3 slides)
  - Demo screenshots (3-4 slides)
  - Results (1 slide - accuracy percentages)
  - Challenges & solutions (1 slide)
  - Future work (1 slide)
  - Conclusion & Q&A
- **4.2.2** Prepare live demo
  - Demo scenario script
  - Test demo flow multiple times
  - Prepare backup (video recording in case live demo fails)
- **4.2.3** Practice presentation (aim for 15-20 minutes + Q&A)

4.3 Code Submission (Week 14)

- **4.3.1** Clean up code
  - Remove console.logs
  - Add comments to complex parts
  - Remove unused files
- **4.3.2** Create README.md for GitHub
  - Project description

    - Features implemented
    - Technology stack
    - Installation instructions
    - How to run locally
    - Screenshots
- **4.3.3** Push to GitHub
- **4.3.4** Create submission package
    - Final report PDF
    - Presentation slides
    - Source code (GitHub link or ZIP)
    - Demo video link
    - Database backup (.sql file)

4.4 Final Submission (Week 14-15)

- **4.4.1** Review submission requirements
- **4.4.2** Double-check all files
- **4.4.3** Submit to university system
- **4.4.4** Present to instructor/committee

Deliverables:

- ✅ Final Project Report (40-60 pages)
- ✅ Presentation Slides
- ✅ Source Code (GitHub)
- ✅ Demo Video
- ✅ Database Backup

Duration: 2-3 weeks

SIMPLIFIED PROJECT TIMELINE

Phase	Duration	Tasks	Priority
1. Design ✅	Already done	Database, Algorithm, ERD	DONE

Phase	Duration	Tasks	Priority
2. Development	11 weeks	Setup → Backend → Frontend → (Extension)	HIGH
3. Testing	1-2 weeks	Manual testing, bug fixes	MEDIUM
4. Documentation	2-3 weeks	Report, presentation, submission	HIGH

Total Duration: ~14-16 weeks (3.5-4 months)

MINIMUM VIABLE PROJECT (If Time Is VERY Tight)

If you're really short on time, focus on **absolute essentials**:

Must Have (Core Requirements):

1. ✅ **Database** with at least 10 key tables
2. ✅ **URL checking algorithm** (the main innovation)
3. ✅ **Backend API** with check-url endpoint working
4. ✅ **Simple web UI** to input URL and see results
5. ✅ **Basic authentication** (login/register)
6. ✅ **Simple dashboard** showing statistics
7. ✅ **Final report** documenting everything
8. ✅ **Demo** showing it works

Nice to Have (If Time Permits):

- ⚠️ Full 18 tables (can start with 10-12 core tables)
- ⚠️ Reporting system
- ⚠️ Moderator workflow
- ⚠️ Educational content
- ⚠️ Browser extension
- ⚠️ AWS deployment (can demo locally)
- ⚠️ Advanced UI features

Can Skip Entirely:

- ❌ Post-launch support
- ❌ Automated testing

- ❌ CI/CD pipeline
- ❌ Performance optimization
- ❌ Production deployment
- ❌ User manual
- ❌ Google Safe Browsing integration (mention as future work)

RESOURCE REQUIREMENTS (Realistic)

Time Investment Per Week:

- **Light weeks**: 10-15 hours/week
- **Heavy weeks**: 20-25 hours/week
- **Final weeks**: 30+ hours/week (documentation)

Software & Tools (All Free):

- **Development**: VS Code, Git, Node.js, PostgreSQL
- **Design**: Draw.io (for diagrams)
- **Testing**: Postman
- **Documentation**: Microsoft Word or Google Docs
- **Presentation**: PowerPoint or Google Slides

Budget:

- **$0** if running everything locally
- **$20-50 total** if deploying to AWS for 2-3 months (can use Free Tier)

Recommendation: Start local, deploy to AWS only if you have time

---

DELIVERABLES CHECKLIST - STUDENT VERSION

Essential Deliverables (Must Have):

- [ ] ✅ Database schema (database_schema.sql)
- [ ] ✅ ERD diagram (already have)
- [ ] ✅ Algorithm design document (already have)
- [ ] Working backend with URL checking
- [ ] Working frontend web application
- [ ] Final project report (40-60 pages)

- [ ] Presentation slides (15-20 slides)
- [ ] Demo video or live demo
- [ ] GitHub repository with code

Optional Deliverables (Nice to Have):

- [ ] Browser extension
- [ ] AWS cloud deployment
- [ ] Test results documentation
- [ ] User manual
- [ ] API documentation

Skip Entirely:

- ❌ Automated test suite
- ❌ CI/CD configuration
- ❌ Production monitoring setup
- ❌ Maintenance plan
- ❌ User acceptance testing report

WEEKLY BREAKDOWN - REALISTIC SCHEDULE

✅ **Weeks 1-2: Already Done**

- Database design complete
- Algorithm design complete
- ERD complete

Week 3: Environment Setup

- Day 1-2: Install all software, create Git repo
- Day 3-4: Set up PostgreSQL database
- Day 5-7: Execute schema, import sample data

Week 4: Backend - Authentication & Setup

- Day 1-3: Express.js setup, database connection
- Day 4-7: User registration, login, JWT

Week 5: Backend - Core Algorithm

- Day 1-3: Implement similarity functions (Levenshtein, Jaro-Winkler)

- Day 4-5: Implement URL parsing and normalization
- Day 6-7: Implement scoring system

Week 6: Backend - API Endpoints

- Day 1-3: URL check endpoint
- Day 4-5: Reports endpoints
- Day 6-7: Dashboard endpoints, testing with Postman

Week 7: Frontend - Setup & Auth

- Day 1-2: React setup, routing
- Day 3-5: Login and register pages
- Day 6-7: Basic layout (header, footer)

Week 8: Frontend - Main Features

- Day 1-3: URL check page with results display
- Day 4-5: Scan history
- Day 6-7: Report submission page

Week 9: Frontend - Dashboard & Polish

- Day 1-3: Dashboard with statistics
- Day 4-5: Moderator page
- Day 6-7: Styling, bug fixes

Week 10: Integration & Testing

- Day 1-3: Connect frontend to backend
- Day 4-5: Test all features end-to-end
- Day 6-7: Fix critical bugs

Week 11: Final Testing & Demo Prep

- Day 1-3: Algorithm accuracy testing
- Day 4-5: Prepare demo data
- Day 6-7: Record demo video, screenshots

Week 12-13: Documentation - Report Writing

- Day 1-4: Write chapters 1-3 (intro, literature, design)
- Day 5-7: Write chapters 4-5 (implementation, testing)

- Day 8-10: Write conclusion, format, proofread
- Day 11-14: Revisions, final touches

Week 14: Presentation & Submission

- Day 1-3: Create presentation slides
- Day 4-5: Practice presentation
- Day 6-7: Final submission, presentation

Total: 14 weeks = ~3.5 months

RISK MANAGEMENT - STUDENT VERSION

Risk	What to Do
Running out of time	Cut browser extension, skip AWS deployment, simplify UI
Algorithm not accurate enough	Document limitations, suggest improvements in "Future Work"
Database too complex	Reduce from 18 to 12 tables (keep core ones only)
Frontend taking too long	Use Bootstrap templates, focus on functionality over design
Don't know how to code something	Google, YouTube tutorials, ask AI assistants, check Stack Overflow
Demo fails during presentation	Always have backup demo video ready

---

SUCCESS CRITERIA - REALISTIC

Minimum to Pass:

- ✅ Working URL checking feature (can demo)
- ✅ Database with core tables created
- ✅ Algorithm implemented (even if not 100% accurate)
- ✅ Basic web interface
- ✅ Complete project report
- ✅ Successful presentation

For Good Grade:

- ✅ All above +
- ✅ Working reporting system
- ✅ Moderator workflow

- ✅ Clean, professional UI
- ✅ 70%+ algorithm accuracy
- ✅ Well-structured code
- ✅ Comprehensive documentation

For Excellent Grade:

- ✅ All above +
- ✅ Browser extension working
- ✅ 85%+ algorithm accuracy
- ✅ AWS deployment
- ✅ Impressive demo
- ✅ Publication-quality documentation

FINAL TIPS FOR SOLO STUDENT

1. **Start Simple**: Get basic version working first, add features later
2. **Prioritize Core Feature**: URL checking algorithm is your main innovation
3. **Don't Overthink**: "Done is better than perfect" for student project
4. **Use What You Know**: Stick to technologies you're comfortable with
5. **Copy-Paste Wisely**: Use Bootstrap, templates, but understand the code
6. **Document As You Go**: Don't wait until the end to write documentation
7. **Test Often**: Test after each feature, easier than debugging later
8. **Ask for Help**: Use Stack Overflow, YouTube, AI assistants
9. **Have Backup Plans**: Demo video if live demo fails, localhost if AWS fails
10. **Take Breaks**: Avoid burnout, pace yourself

Remember: This is a **student project**, not a production system. Your goal is to **demonstrate understanding** of:

- Database design
- Algorithm development

- Full-stack web development
- Cloud computing concepts (even if just theoretically)

You don't need:

- ❌ 100% test coverage
- ❌ Production-grade security
- ❌ Perfect UI/UX
- ❌ Scalability to millions of users
- ❌ 24/7 uptime

You DO need:

- ✅ Working prototype
- ✅ Core features functional
- ✅ Good documentation
- ✅ Successful demo
- ✅ Understanding of concepts

Good luck with your project! 🎓🚀

Next Step: Start Week 3 - Environment Setup!

End of Simplified WBS for Student Project

APPENDIX: What We Cut (Can Mention as "Future Work")

These features can be mentioned in your "Future Work" section of the report:

From Original Plan - Now Optional/Future:

- ✅ Google Safe Browsing API integration → "Future enhancement"
- ✅ Threat feed integration → "Future enhancement"
- ✅ Educational content pages → "Planned feature"
- ✅ Quiz functionality → "Planned feature"
- ✅ Automated testing suite → "Would improve quality assurance"
- ✅ CI/CD pipeline → "For production deployment"

- ✅ Post-launch support → "Not applicable to academic project"
- ✅ DynamoDB tables → "Optional cloud optimization"
- ✅ Full AWS production deployment → "Can demo locally"

This is **completely acceptable** for a student project. Mention these as "limitations" or "future work" in your conclusion.

Document Version: 2.0 - Student Simplified
Last Updated: November 22, 2025
Status: Ready for Implementation

Current Phase: Week 3 - Start Development! 🚀

2.1 Architecture Design

- **2.1.1** Design overall system architecture
  - **2.1.1.1** Define system components
  - **2.1.1.2** Identify component interactions
  - **2.1.1.3** Create architecture diagrams
- **2.1.2** Design data flow between components
- **2.1.3** Design security architecture
  - **2.1.3.1** Authentication mechanism
  - **2.1.3.2** Authorization model (RBAC)
  - **2.1.3.3** Data encryption strategy
- **2.1.4** Design AWS cloud architecture
  - **2.1.4.1** Network topology (VPC, subnets)
  - **2.1.4.2** Service integration plan
  - **2.1.4.3** Deployment architecture

2.2 Database Design

- **2.2.1** Design database schema
  - **2.2.1.1** User management tables (2 tables)
  - **2.2.1.2** URL detection tables (4 tables)
  - **2.2.1.3** Third-party integration tables (4 tables)
  - **2.2.1.4** Community reporting tables (2 tables)

        - **2.2.1.5** Educational content tables (3 tables)
        - **2.2.1.6** Browser extension tables (2 tables)
        - **2.2.1.7** System configuration tables (1 table)
    - **2.2.2** Create Entity Relationship Diagram (ERD)
    - **2.2.3** Define indexes for performance optimization
    - **2.2.4** Design DynamoDB tables (4 tables)
        - **2.2.4.1** RealTimeScans table
        - **2.2.4.2** LiveDashboard table
        - **2.2.4.3** ExtensionSessions table
        - **2.2.4.4** ApiRateLimits table
    - **2.2.5** Create data dictionary
    - **2.2.6** Write SQL DDL scripts

2.3 Algorithm Design

- **2.3.1** Design URL parsing and normalization logic
- **2.3.2** Design similarity metrics
    - **2.3.2.1** Levenshtein distance algorithm
    - **2.3.2.2** Jaro-Winkler similarity
    - **2.3.2.3** Token-based similarity
    - **2.3.2.4** Longest common substring
- **2.3.3** Design component analysis modules
    - **2.3.3.1** Domain similarity analysis
    - **2.3.3.2** Subdomain analysis
    - **2.3.3.3** Path analysis
    - **2.3.3.4** Query parameter analysis
    - **2.3.3.5** Heuristic checks
- **2.3.4** Design scoring and classification system
    - **2.3.4.1** Define component weights
    - **2.3.4.2** Define threshold values

        - **2.3.4.3** Create classification rules
    - **2.3.5** Document algorithm with pseudocode
    - **2.3.6** Create algorithm flowcharts

2.4 API Design

- **2.4.1** Define RESTful API structure
- **2.4.2** Design API endpoints
    - **2.4.2.1** Authentication endpoints
    - **2.4.2.2** URL check endpoints
    - **2.4.2.3** Reporting endpoints
    - **2.4.2.4** User management endpoints
    - **2.4.2.5** Dashboard endpoints
    - **2.4.2.6** Educational content endpoints
- **2.4.3** Define request/response schemas (JSON)
- **2.4.4** Design error handling
- **2.4.5** Create API documentation (OpenAPI/Swagger)
- **2.4.6** Define rate limiting strategy

2.5 UI/UX Design

- **2.5.1** Create wireframes
    - **2.5.1.1** Home page wireframe
    - **2.5.1.2** URL check page wireframe
    - **2.5.1.3** Report submission page wireframe
    - **2.5.1.4** Dashboard wireframe
    - **2.5.1.5** Education page wireframe
- **2.5.2** Design user interface mockups
- **2.5.3** Create user flow diagrams
- **2.5.4** Design browser extension interface
- **2.5.5** Design mobile-responsive layouts
- **2.5.6** Create style guide and design system

2.6 Sequence Diagrams

- **2.6.1** URL checking workflow
- **2.6.2** User registration workflow
- **2.6.3** Report submission workflow
- **2.6.4** Moderation workflow
- **2.6.5** Browser extension alert workflow

Deliverables:

- ✅ System Architecture Document
- ✅ Database Design Document (18 PostgreSQL + 4 DynamoDB tables)
- ✅ Algorithm Design Document (200+ pages)
- ✅ API Specification Document
- ✅ UI/UX Design Package
- ✅ ERD Diagram
- ✅ Sequence Diagrams

Duration: 3-4 weeks

3.0 DEVELOPMENT

3.1 Development Environment Setup

- **3.1.1** Set up version control (Git repository)
- **3.1.2** Configure development tools
  - **3.1.2.1** Install Node.js and npm
  - **3.1.2.2** Install PostgreSQL
  - **3.1.2.3** Install code editor (VS Code)
  - **3.1.2.4** Install AWS CLI
- **3.1.3** Set up local PostgreSQL database
- **3.1.4** Configure AWS account and services
- **3.1.5** Set up project structure
- **3.1.6** Configure environment variables

- **3.1.7** Set up CI/CD pipeline (optional)

3.2 Database Implementation

- **3.2.1** Create PostgreSQL database
- **3.2.2** Execute SQL DDL scripts
  - **3.2.2.1** Create all 18 tables
  - **3.2.2.2** Create 26 indexes
  - **3.2.2.3** Create 2 triggers
  - **3.2.2.4** Create 3 views
- **3.2.3** Set up database users and permissions
- **3.2.4** Seed initial data
  - **3.2.4.1** Create default admin user
  - **3.2.4.2** Populate system settings
  - **3.2.4.3** Import known phishing patterns
  - **3.2.4.4** Import legitimate brand list
- **3.2.5** Create DynamoDB tables
  - **3.2.5.1** Deploy CloudFormation stack
  - **3.2.5.2** Configure GSIs
  - **3.2.5.3** Enable TTL
  - **3.2.5.4** Enable point-in-time recovery
- **3.2.6** Test database connections
- **3.2.7** Verify data integrity

3.3 Backend API Development

- **3.3.1** Set up Express.js project structure
- **3.3.2** Configure database drivers
  - **3.3.2.1** Install and configure pg (PostgreSQL)
  - **3.3.2.2** Install and configure AWS SDK (DynamoDB)
  - **3.3.2.3** Set up connection pooling
- **3.3.3** Implement authentication system
  - **3.3.3.1** User registration endpoint

    - **3.3.3.2** User login endpoint
    - **3.3.3.3** Password hashing (bcrypt)
    - **3.3.3.4** JWT token generation
    - **3.3.3.5** Token validation middleware
- **3.3.4** Implement URL detection algorithm
    - **3.3.4.1** URL parsing module
    - **3.3.4.2** URL normalization module
    - **3.3.4.3** Levenshtein distance function
    - **3.3.4.4** Jaro-Winkler similarity function
    - **3.3.4.5** Domain analysis module
    - **3.3.4.6** Subdomain analysis module
    - **3.3.4.7** Path analysis module
    - **3.3.4.8** Query parameter analysis module
    - **3.3.4.9** Heuristic checks module
    - **3.3.4.10** Scoring aggregation module
    - **3.3.4.11** Classification module
- **3.3.5** Implement Google Safe Browsing API integration
    - **3.3.5.1** API client setup
    - **3.3.5.2** Cache implementation
    - **3.3.5.3** Error handling
- **3.3.6** Implement threat feed integration
    - **3.3.6.1** Feed import scheduler
    - **3.3.6.2** Feed parsing logic
    - **3.3.6.3** Feed matching algorithm
- **3.3.7** Implement URL check endpoints
    - **3.3.7.1** POST /api/check-url
    - **3.3.7.2** GET /api/check-history
    - **3.3.7.3** GET /api/url/:id

- **3.3.8** Implement reporting system
  - **3.3.8.1** POST /api/reports (submit report)
  - **3.3.8.2** GET /api/reports (list reports)
  - **3.3.8.3** GET /api/reports/:id (report details)
  - **3.3.8.4** PUT /api/reports/:id (update status)
  - **3.3.8.5** POST /api/reports/:id/feedback (add feedback)
- **3.3.9** Implement user management endpoints
  - **3.3.9.1** GET /api/users (list users - admin only)
  - **3.3.9.2** GET /api/users/:id (user profile)
  - **3.3.9.3** PUT /api/users/:id (update profile)
  - **3.3.9.4** DELETE /api/users/:id (delete user)
  - **3.3.9.5** PUT /api/users/:id/role (change role - admin only)
- **3.3.10** Implement dashboard endpoints
  - **3.3.10.1** GET /api/dashboard/stats (statistics)
  - **3.3.10.2** GET /api/dashboard/recent-threats
  - **3.3.10.3** GET /api/dashboard/moderator-queue
- **3.3.11** Implement educational content endpoints
  - **3.3.11.1** GET /api/education (list content)
  - **3.3.11.2** GET /api/education/:id (content details)
  - **3.3.11.3** POST /api/education (create - admin only)
  - **3.3.11.4** POST /api/quiz/:id/attempt (submit quiz)
- **3.3.12** Implement error handling middleware
- **3.3.13** Implement logging system
- **3.3.14** Implement rate limiting
- **3.3.15** Write API documentation

3.4 Frontend Web Application Development

- **3.4.1** Set up React.js project
  - **3.4.1.1** Initialize with Create React App
  - **3.4.1.2** Install dependencies

    - **3.4.1.3** Configure routing (React Router)
    - **3.4.1.4** Set up state management (Context API or Redux)
- **3.4.2** Implement authentication pages
    - **3.4.2.1** Login page
    - **3.4.2.2** Registration page
    - **3.4.2.3** Password reset page
- **3.4.3** Implement home page
    - **3.4.3.1** Header component
    - **3.4.3.2** Hero section with real-time stats
    - **3.4.3.3** Feature highlights
    - **3.4.3.4** Call-to-action sections
    - **3.4.3.5** Footer component
- **3.4.4** Implement URL check page
    - **3.4.4.1** URL input form
    - **3.4.4.2** Auto-scan functionality
    - **3.4.4.3** Results display component
    - **3.4.4.4** Component-level breakdown display
    - **3.4.4.5** Share results feature
- **3.4.5** Implement report submission page
    - **3.4.5.1** Report form with validation
    - **3.4.5.2** Evidence upload (screenshots)
    - **3.4.5.3** Pre-scan display
    - **3.4.5.4** Submission confirmation
- **3.4.6** Implement education page
    - **3.4.6.1** Content listing
    - **3.4.6.2** Article viewer
    - **3.4.6.3** Video player
    - **3.4.6.4** Interactive quiz component

- **3.4.6.5** Audio guide player (accessibility)

- **3.4.7** Implement community feedback page
  - **3.4.7.1** Report listing
  - **3.4.7.2** Voting interface
  - **3.4.7.3** Comment system
  - **3.4.7.4** Report details view
- **3.4.8** Implement dashboard (role-specific)
  - **3.4.8.1** Statistics overview
  - **3.4.8.2** Recent activity feed
  - **3.4.8.3** User activity logs (admin)
  - **3.4.8.4** Moderator queue (moderator/admin)
  - **3.4.8.5** Charts and visualizations
- **3.4.9** Implement user profile page
  - **3.4.9.1** Profile information display
  - **3.4.9.2** Edit profile form
  - **3.4.9.3** Scan history
  - **3.4.9.4** Report history
- **3.4.10** Implement admin panel
  - **3.4.10.1** User management interface
  - **3.4.10.2** System settings editor
  - **3.4.10.3** Phishing database management
  - **3.4.10.4** Report moderation tools
- **3.4.11** Implement responsive design (mobile/tablet)
- **3.4.12** Implement error boundaries
- **3.4.13** Add loading states and spinners
- **3.4.14** Implement internationalization (i18n)
  - **3.4.14.1** Vietnamese language support
  - **3.4.14.2** English language support

- **3.4.15** Optimize performance
  - **3.4.15.1** Code splitting
  - **3.4.15.2** Lazy loading
  - **3.4.15.3** Image optimization

3.5 Browser Extension Development

- **3.5.1** Set up extension project structure
- **3.5.2** Create manifest.json configuration
- **3.5.3** Implement background script
  - **3.5.3.1** URL monitoring
  - **3.5.3.2** API communication
  - **3.5.3.3** Cache management
- **3.5.4** Implement content script
  - **3.5.4.1** Page analysis
  - **3.5.4.2** Alert injection
- **3.5.5** Implement popup interface
  - **3.5.5.1** Status display
  - **3.5.5.2** Manual scan button
  - **3.5.5.3** Quick report button
  - **3.5.5.4** Settings panel
- **3.5.6** Implement alert system
  - **3.5.6.1** Warning banner (suspicious)
  - **3.5.6.2** Block page (dangerous)
  - **3.5.6.3** Info notification (safe)
- **3.5.7** Implement offline functionality
  - **3.5.7.1** Local cache of known phishing patterns
  - **3.5.7.2** Offline detection logic
- **3.5.8** Implement auto-update mechanism
- **3.5.9** Create extension icons (16x16, 48x48, 128x128)
- **3.5.10** Build extension packages

- **3.5.10.1** Chrome extension (.crx)
- **3.5.10.2** Firefox extension (.xpi)
- **3.5.10.3** Edge extension

3.6 Educational Content Creation

- **3.6.1** Research Vietnamese phishing trends
- **3.6.2** Create educational articles
  - **3.6.2.1** "Recognizing phishing emails"
  - **3.6.2.2** "Common phishing tactics in Vietnam"
  - **3.6.2.3** "How to verify website authenticity"
  - **3.6.2.4** "Safe online banking practices"
  - **3.6.2.5** "What to do if you've been phished"
- **3.6.3** Create video tutorials
  - **3.6.3.1** Script writing
  - **3.6.3.2** Video recording/animation
  - **3.6.3.3** Vietnamese voiceover
  - **3.6.3.4** Video editing
- **3.6.4** Create infographics
  - **3.6.4.1** Design visual guides
  - **3.6.4.2** Translate to Vietnamese
- **3.6.5** Create interactive quizzes
  - **3.6.5.1** Beginner level quiz (10 questions)
  - **3.6.5.2** Intermediate level quiz (15 questions)
  - **3.6.5.3** Advanced level quiz (20 questions)
- **3.6.6** Create audio guides (for elderly/visually impaired)
  - **3.6.6.1** Script writing
  - **3.6.6.2** Vietnamese narration recording
  - **3.6.6.3** Audio editing
- **3.6.7** Upload content to AWS S3

- **3.6.8** Import content metadata to database

3.7 Data Collection & Preparation

- **3.7.1** Collect phishing URL datasets
  - **3.7.1.1** PhishTank dataset
  - **3.7.1.2** OpenPhish dataset
  - **3.7.1.3** APWG dataset
  - **3.7.1.4** Vietnamese-specific phishing reports
- **3.7.2** Collect legitimate URL datasets
  - **3.7.2.1** Alexa Top 1000
  - **3.7.2.2** Vietnamese popular websites
  - **3.7.2.3** Major brand domains
- **3.7.3** Clean and normalize data
- **3.7.4** Import data into database
  - **3.7.4.1** Import known phishing URLs
  - **3.7.4.2** Import legitimate brand list
  - **3.7.4.3** Import threat feed data
- **3.7.5** Verify data quality

Deliverables:

- ✅ Functional Backend API
- ✅ Functional Web Application
- ✅ Functional Browser Extension
- ✅ Educational Content Library
- ✅ Populated Database

Duration: 8-10 weeks

4.0 TESTING

4.1 Unit Testing

- **4.1.1** Write backend unit tests
  - **4.1.1.1** Algorithm component tests

    - **4.1.1.2** Similarity metric tests
    - **4.1.1.3** Database query tests
    - **4.1.1.4** API endpoint tests
- **4.1.2** Write frontend unit tests
    - **4.1.2.1** Component tests (React)
    - **4.1.2.2** Utility function tests
    - **4.1.2.3** State management tests
- **4.1.3** Write extension unit tests
- **4.1.4** Achieve 80%+ code coverage
- **4.1.5** Fix failing tests

4.2 Integration Testing

- **4.2.1** Test frontend-backend integration
    - **4.2.1.1** API call integration
    - **4.2.1.2** Authentication flow
    - **4.2.1.3** Data display accuracy
- **4.2.2** Test backend-database integration
    - **4.2.2.1** CRUD operations
    - **4.2.2.2** Transaction handling
    - **4.2.2.3** Trigger functionality
- **4.2.3** Test third-party API integration
    - **4.2.3.1** Google Safe Browsing API
    - **4.2.3.2** Threat feed APIs
- **4.2.4** Test browser extension integration
    - **4.2.4.1** Extension-backend communication
    - **4.2.4.2** Alert functionality
    - **4.2.4.3** Cache synchronization

4.3 Algorithm Testing

- **4.3.1** Test against PhishTank dataset
    - **4.3.1.1** Calculate detection rate (target: 85%+)

    - **4.3.1.2** Analyze false negatives
- **4.3.2** Test against legitimate websites
    - **4.3.2.1** Calculate false positive rate (target: <5%)
    - **4.3.2.2** Analyze false positives
- **4.3.3** Test edge cases
    - **4.3.3.1** URL shorteners
    - **4.3.3.2** Punycode domains
    - **4.3.3.3** Data URIs
    - **4.3.3.4** Malformed URLs
- **4.3.4** Test performance
    - **4.3.4.1** Single URL analysis time (target: <100ms)
    - **4.3.4.2** Concurrent request handling
    - **4.3.4.3** Database query performance
- **4.3.5** Tune algorithm parameters
    - **4.3.5.1** Adjust component weights
    - **4.3.5.2** Adjust thresholds
    - **4.3.5.3** Re-test after adjustments

4.4 User Acceptance Testing (UAT)

- **4.4.1** Recruit test users (10-15 people)
    - **4.4.1.1** Elderly users (target demographic)
    - **4.4.1.2** Tech-savvy users
    - **4.4.1.3** Moderators/admins
- **4.4.2** Prepare test scenarios
    - **4.4.2.1** User registration and login
    - **4.4.2.2** URL checking
    - **4.4.2.3** Report submission
    - **4.4.2.4** Community feedback
    - **4.4.2.5** Educational content access

    - **4.4.2.6** Browser extension usage
- **4.4.3** Conduct UAT sessions
- **4.4.4** Collect feedback
    - **4.4.4.1** Usability feedback
    - **4.4.4.2** Feature requests
    - **4.4.4.3** Bug reports
- **4.4.5** Analyze results
- **4.4.6** Prioritize improvements
- **4.4.7** Implement critical fixes

4.5 Security Testing

- **4.5.1** Perform vulnerability assessment
    - **4.5.1.1** SQL injection testing
    - **4.5.1.2** XSS (Cross-Site Scripting) testing
    - **4.5.1.3** CSRF (Cross-Site Request Forgery) testing
    - **4.5.1.4** Authentication bypass testing
    - **4.5.1.5** Authorization testing
- **4.5.2** Test password security
    - **4.5.2.1** Verify bcrypt hashing
    - **4.5.2.2** Test password strength requirements
- **4.5.3** Test API security
    - **4.5.3.1** JWT token validation
    - **4.5.3.2** Rate limiting effectiveness
    - **4.5.3.3** Input validation
- **4.5.4** Test data encryption
    - **4.5.4.1** Data at rest encryption
    - **4.5.4.2** Data in transit encryption (HTTPS)
- **4.5.5** Penetration testing (optional)
- **4.5.6** Fix security vulnerabilities

4.6 Performance Testing

- **4.6.1** Load testing
  - **4.6.1.1** Simulate 100 concurrent users
  - **4.6.1.2** Simulate 1000 concurrent users
  - **4.6.1.3** Measure response times
- **4.6.2** Stress testing
  - **4.6.2.1** Test system limits
  - **4.6.2.2** Identify breaking points
- **4.6.3** Database performance testing
  - **4.6.3.1** Query optimization
  - **4.6.3.2** Index effectiveness
- **4.6.4** API endpoint performance testing
- **4.6.5** Browser extension performance testing
- **4.6.6** Optimize identified bottlenecks

4.7 Cross-Browser Testing

- **4.7.1** Test on Google Chrome
- **4.7.2** Test on Mozilla Firefox
- **4.7.3** Test on Microsoft Edge
- **4.7.4** Test on Safari (macOS)
- **4.7.5** Test on mobile browsers
  - **4.7.5.1** Chrome Mobile
  - **4.7.5.2** Safari Mobile
- **4.7.6** Fix browser-specific issues

4.8 Accessibility Testing

- **4.8.1** Test with screen readers
- **4.8.2** Test keyboard navigation
- **4.8.3** Test color contrast
- **4.8.4** Test font scaling
- **4.8.5** Test audio guide functionality

- **4.8.6** Ensure WCAG 2.1 compliance

Deliverables:

- ✅ Test Reports (unit, integration, UAT)
- ✅ Algorithm Performance Report
- ✅ Security Audit Report
- ✅ Performance Test Results
- ✅ Bug Fix Log

Duration: 3-4 weeks

5.0 DEPLOYMENT

5.1 AWS Infrastructure Setup

- **5.1.1** Configure VPC and subnets
- **5.1.2** Set up security groups
- **5.1.3** Configure IAM roles and policies
- **5.1.4** Set up RDS PostgreSQL instance
  - **5.1.4.1** Choose instance type
  - **5.1.4.2** Configure Multi-AZ deployment
  - **5.1.4.3** Set up automated backups
  - **5.1.4.4** Configure parameter groups
- **5.1.5** Set up DynamoDB tables
  - **5.1.5.1** Deploy CloudFormation stack
  - **5.1.5.2** Configure auto-scaling
  - **5.1.5.3** Enable point-in-time recovery
- **5.1.6** Set up S3 buckets
  - **5.1.6.1** Static website hosting bucket
  - **5.1.6.2** Educational content bucket
  - **5.1.6.3** Log storage bucket
- **5.1.7** Configure CloudFront CDN

- **5.1.8** Set up EC2 instances for backend
  - **5.1.8.1** Choose instance type
  - **5.1.8.2** Configure auto-scaling group
  - **5.1.8.3** Set up load balancer
- **5.1.9** Configure Route 53 DNS
- **5.1.10** Set up SSL/TLS certificates (AWS Certificate Manager)

5.2 Database Deployment

- **5.2.1** Migrate schema to RDS
  - **5.2.1.1** Execute DDL scripts
  - **5.2.1.2** Verify table creation
- **5.2.2** Import production data
  - **5.2.2.1** Import known phishing patterns
  - **5.2.2.2** Import legitimate brands
  - **5.2.2.3** Create admin users
- **5.2.3** Set up database backups
  - **5.2.3.1** Configure automated backups
  - **5.2.3.2** Test backup restoration
- **5.2.4** Configure read replicas (optional)
- **5.2.5** Test database connectivity

5.3 Backend Deployment

- **5.3.1** Build backend application
  - **5.3.1.1** Run production build
  - **5.3.1.2** Bundle dependencies
- **5.3.2** Configure environment variables
  - **5.3.2.1** Database credentials
  - **5.3.2.2** API keys
  - **5.3.2.3** AWS credentials
- **5.3.3** Deploy to EC2 instances
  - **5.3.3.1** Upload application code

        - **5.3.3.2** Install Node.js runtime
        - **5.3.3.3** Install dependencies
        - **5.3.3.4** Configure PM2 process manager
    - **5.3.4** Configure reverse proxy (Nginx)
    - **5.3.5** Set up HTTPS
    - **5.3.6** Test API endpoints
    - **5.3.7** Configure auto-restart on failure

5.4 Frontend Deployment

- **5.4.1** Build React application
    - **5.4.1.1** Run production build
    - **5.4.1.2** Optimize assets
    - **5.4.1.3** Generate source maps
- **5.4.2** Upload to S3 bucket
- **5.4.3** Configure S3 static website hosting
- **5.4.4** Configure CloudFront distribution
    - **5.4.4.1** Set up SSL certificate
    - **5.4.4.2** Configure caching rules
- **5.4.5** Update DNS records
- **5.4.6** Test website access
- **5.4.7** Configure custom error pages

5.5 Browser Extension Deployment

- **5.5.1** Prepare extension packages
    - **5.5.1.1** Chrome Web Store package
    - **5.5.1.2** Firefox Add-ons package
    - **5.5.1.3** Edge Add-ons package
- **5.5.2** Create developer accounts
    - **5.5.2.1** Chrome Web Store developer account
    - **5.5.2.2** Firefox Add-ons developer account

    - **5.5.2.3** Edge Add-ons developer account
- **5.5.3** Submit extensions for review
    - **5.5.3.1** Chrome Web Store submission
    - **5.5.3.2** Firefox Add-ons submission
    - **5.5.3.3** Edge Add-ons submission
- **5.5.4** Respond to review feedback
- **5.5.5** Publish approved extensions
- **5.5.6** Monitor installation metrics

5.6 Monitoring & Logging Setup

- **5.6.1** Configure CloudWatch
    - **5.6.1.1** Set up log groups
    - **5.6.1.2** Configure log streams
    - **5.6.1.3** Set up custom metrics
- **5.6.2** Create CloudWatch alarms
    - **5.6.2.1** High CPU usage alarm
    - **5.6.2.2** High memory usage alarm
    - **5.6.2.3** High error rate alarm
    - **5.6.2.4** Database connection errors alarm
- **5.6.3** Set up CloudWatch dashboard
    - **5.6.3.1** API request metrics
    - **5.6.3.2** Error rate graphs
    - **5.6.3.3** Database performance metrics
    - **5.6.3.4** User activity metrics
- **5.6.4** Configure SNS notifications
    - **5.6.4.1** Email notifications
    - **5.6.4.2** SMS notifications (optional)
- **5.6.5** Set up application logging
    - **5.6.5.1** Configure Winston logger

- **5.6.5.2** Send logs to CloudWatch

- **5.6.6** Configure AWS X-Ray for tracing (optional)

5.7 Post-Deployment Testing

- **5.7.1** Smoke testing
  - **5.7.1.1** Test critical user flows
  - **5.7.1.2** Verify API endpoints
  - **5.7.1.3** Test authentication
- **5.7.2** End-to-end testing in production
- **5.7.3** Performance testing in production
- **5.7.4** Security verification
  - **5.7.4.1** SSL certificate validation
  - **5.7.4.2** Security headers check
  - **5.7.4.3** Vulnerability scan
- **5.7.5** Monitor error logs
- **5.7.6** Verify monitoring and alerting

5.8 Launch Preparation

- **5.8.1** Create user documentation
  - **5.8.1.1** User guide (Vietnamese)
  - **5.8.1.2** FAQ document
  - **5.8.1.3** Video tutorials
- **5.8.2** Create admin documentation
  - **5.8.2.1** System administration guide
  - **5.8.2.2** Troubleshooting guide
  - **5.8.2.3** Backup/restore procedures
- **5.8.3** Prepare marketing materials
  - **5.8.3.1** Website announcement
  - **5.8.3.2** Social media posts
  - **5.8.3.3** Press release (optional)

- **5.8.4** Set up user support channels
  - **5.8.4.1** Email support
  - **5.8.4.2** FAQ page
  - **5.8.4.3** Contact form
- **5.8.5** Conduct final review
- **5.8.6** Go-live decision

Deliverables:

- ✅ Live Production System
- ✅ Monitoring Dashboard
- ✅ User Documentation
- ✅ Admin Documentation
- ✅ Deployment Guide

Duration: 2-3 weeks

6.0 POST-LAUNCH SUPPORT & MAINTENANCE

6.1 Monitoring & Performance

- **6.1.1** Monitor system health daily
- **6.1.2** Track key metrics
  - **6.1.2.1** API response times
  - **6.1.2.2** Error rates
  - **6.1.2.3** User activity
  - **6.1.2.4** Detection accuracy
- **6.1.3** Analyze CloudWatch logs
- **6.1.4** Review user feedback
- **6.1.5** Generate weekly performance reports

6.2 Bug Fixes & Updates

- **6.2.1** Monitor bug reports
- **6.2.2** Prioritize issues

- **6.2.3** Develop fixes
- **6.2.4** Test fixes
- **6.2.5** Deploy updates
- **6.2.6** Document changes in changelog

6.3 Database Maintenance

- **6.3.1** Perform weekly backups
- **6.3.2** Monitor database performance
- **6.3.3** Optimize slow queries
- **6.3.4** Clean up old data (per retention policy)
  - **6.3.4.1** Archive old activity logs
  - **6.3.4.2** Remove expired cache entries
- **6.3.5** Update phishing database
  - **6.3.5.1** Import new threat feeds
  - **6.3.5.2** Review community reports
  - **6.3.5.3** Update known patterns

6.4 User Support

- **6.4.1** Respond to user inquiries
- **6.4.2** Assist with technical issues
- **6.4.3** Moderate community reports
- **6.4.4** Handle false positive reports
- **6.4.5** Update FAQ based on common questions

6.5 Security Updates

- **6.5.1** Monitor security advisories
- **6.5.2** Update dependencies with security patches
- **6.5.3** Review and update security policies
- **6.5.4** Conduct periodic security audits
- **6.5.5** Update SSL certificates before expiry

6.6 Content Updates

- **6.6.1** Add new educational content
  - **6.6.1.1** New phishing examples
  - **6.6.1.2** Updated statistics
  - **6.6.1.3** New threat warnings
- **6.6.2** Update existing content
- **6.6.3** Translate new content to Vietnamese

6.7 Algorithm Tuning

- **6.7.1** Analyze detection performance
- **6.7.2** Collect false positive/negative cases
- **6.7.3** Adjust algorithm parameters
  - **6.7.3.1** Tune component weights
  - **6.7.3.2** Adjust thresholds
- **6.7.4** Test improved algorithm
- **6.7.5** Deploy algorithm updates

6.8 Feature Enhancements (Future)

- **6.8.1** Collect feature requests
- **6.8.2** Prioritize new features
- **6.8.3** Design enhancements
- **6.8.4** Develop new features
- **6.8.5** Test thoroughly
- **6.8.6** Deploy incrementally

Deliverables:

- ✅ Weekly Performance Reports
- ✅ Bug Fix Updates
- ✅ Updated Documentation
- ✅ Algorithm Improvements

Duration: Ongoing

7.0 PROJECT DOCUMENTATION & CLOSURE

7.1 Technical Documentation

- **7.1.1** Finalize system architecture document
- **7.1.2** Complete API documentation
- **7.1.3** Document database schema
- **7.1.4** Document algorithm design
- **7.1.5** Create deployment guide
- **7.1.6** Write developer handbook

7.2 User Documentation

- **7.2.1** Finalize user guide
- **7.2.2** Complete FAQ
- **7.2.3** Create video tutorials
- **7.2.4** Translate all documentation to Vietnamese

7.3 Project Report

- **7.3.1** Write executive summary
- **7.3.2** Document project methodology
- **7.3.3** Present system architecture
- **7.3.4** Include algorithm design
- **7.3.5** Document implementation details
- **7.3.6** Present testing results
- **7.3.7** Include deployment information
- **7.3.8** Discuss challenges and solutions
- **7.3.9** Present project outcomes
- **7.3.10** Conclude with future recommendations

7.4 Presentation Preparation

- **7.4.1** Create PowerPoint presentation
  - **7.4.1.1** Introduction slides
  - **7.4.1.2** Problem statement

- **7.4.1.3** Solution overview
- **7.4.1.4** System architecture
- **7.4.1.5** Algorithm explanation
- **7.4.1.6** Demo screenshots/videos
- **7.4.1.7** Results and metrics
- **7.4.1.8** Conclusion and Q&A

- **7.4.2** Prepare live demonstration
- **7.4.3** Create presentation script
- **7.4.4** Rehearse presentation

7.5 Project Submission

- **7.5.1** Compile all deliverables
- **7.5.2** Review submission requirements
- **7.5.3** Organize documentation
- **7.5.4** Submit project report
- **7.5.5** Submit source code (GitHub repository)
- **7.5.6** Submit supporting documents

7.6 Project Presentation

- **7.6.1** Present to instructor/committee
- **7.6.2** Demonstrate system functionality
- **7.6.3** Answer questions
- **7.6.4** Address feedback

7.7 Post-Project Review

- **7.7.1** Conduct lessons learned session
- **7.7.2** Document successes
- **7.7.3** Document challenges
- **7.7.4** Identify improvement areas
- **7.7.5** Celebrate project completion

Deliverables:

- ✅ Complete Technical Documentation
- ✅ Complete User Documentation
- ✅ Final Project Report (100+ pages)
- ✅ Presentation Slides
- ✅ Project Demonstration

Duration: 2 weeks

PROJECT TIMELINE SUMMARY

Phase	Duration	Start	End
1.0 Project Initiation & Planning	2 weeks	Week 1	Week 2
2.0 System Design	3-4 weeks	Week 3	Week 6
3.0 Development	8-10 weeks	Week 7	Week 16
4.0 Testing	3-4 weeks	Week 17	Week 20
5.0 Deployment	2-3 weeks	Week 21	Week 23
6.0 Post-Launch Support	Ongoing	Week 24+	-
7.0 Documentation & Closure	2 weeks	Week 24	Week 25

Total Project Duration: ~25 weeks (6 months)

RESOURCE ALLOCATION

Human Resources

- **Project Lead/Developer**: 1 person (you)
- **Advisor/Supervisor**: 1 person
- **Test Users**: 10-15 people (for UAT)

Software & Tools

- **Development**: VS Code, Git, Node.js, React, PostgreSQL, Postman
- **Design**: Figma, Draw.io, Lucidchart
- **Cloud**: AWS (RDS, DynamoDB, EC2, S3, CloudFront, CloudWatch)
- **Testing**: Jest, React Testing Library, Postman
- **Project Management**: Notion, Trello, or Jira (optional)

Budget Estimate (AWS Costs)

- **RDS PostgreSQL**: $25-50/month
- **DynamoDB**: $5-10/month
- **EC2**: $20-40/month
- **S3 + CloudFront**: $5-10/month
- **Data Transfer**: $5-10/month
- **Total**: ~$60-120/month

Student Tip: Use AWS Free Tier (12 months) to minimize costs

RISK MANAGEMENT

Key Risks

Risk	Impact	Probability	Mitigation Strategy
Algorithm accuracy below target	High	Medium	Extensive testing, parameter tuning, collect more training data
AWS costs exceed budget	Medium	Low	Use free tier, monitor usage, optimize resources
Third-party API unavailable	Medium	Low	Implement local caching, fallback logic
Browser extension rejection	Medium	Low	Follow store guidelines, respond quickly to feedback
Database performance issues	High	Medium	Proper indexing, query optimization, connection pooling
Security vulnerabilities	High	Medium	Security testing, code review, follow best practices
Scope creep	Medium	High	Strict requirement prioritization, phase planning
Time delays	High	Medium	Buffer time in schedule, prioritize core features

---

SUCCESS CRITERIA

Technical Success Metrics

- ✅ Algorithm detection rate ≥ 85%
- ✅ False positive rate ≤ 5%
- ✅ API response time < 100ms
- ✅ System uptime ≥ 99%
- ✅ Database query time < 10ms
- ✅ Browser extension install rate > 100 users (post-launch)

Functional Success Metrics

- ✅ All 18 PostgreSQL tables implemented
- ✅ All 4 DynamoDB tables deployed
- ✅ All API endpoints functional
- ✅ All user roles implemented
- ✅ Browser extension published
- ✅ Educational content available

Project Success Metrics

- ✅ Project completed on time
- ✅ All deliverables submitted
- ✅ Successful final presentation
- ✅ Passing grade achieved

DELIVERABLES CHECKLIST

Documentation Deliverables

- [ ] Project Charter
- [ ] Requirements Document
- [ ] System Architecture Document
- [ ] Database Design Document (✅ Complete)
- [ ] Algorithm Design Document (✅ Complete)
- [ ] API Specification Document
- [ ] User Documentation
- [ ] Admin Documentation
- [ ] Final Project Report
- [ ] Presentation Slides

Technical Deliverables

- [ ] PostgreSQL Database (18 tables) (✅ Design Complete)
- [ ] DynamoDB Tables (4 tables) (✅ Design Complete)
- [ ] Backend API (Node.js + Express)
- [ ] Frontend Web Application (React)
- [ ] Browser Extension (Chrome/Firefox/Edge)
- [ ] Educational Content Library

- [ ] Test Suite (Unit + Integration)
- [ ] Deployment Scripts
- [ ] Source Code Repository (GitHub)

Visual Deliverables

- [ ] ERD Diagram (✅ Complete)
- [ ] System Architecture Diagram
- [ ] Sequence Diagrams
- [ ] UI Wireframes
- [ ] API Documentation
- [ ] CloudWatch Dashboard

NOTES

Current Status: ✅ Planning & Design phases completed

- Database design: 100% complete
- Algorithm design: 100% complete
- Next phase: Development (Backend API → Frontend → Extension)

Key Dependencies:

- AWS account setup required before deployment
- Google Safe Browsing API key needed for integration
- Test datasets need to be collected before algorithm testing

Communication:

- Weekly progress reports to advisor
- Documentation updates after each phase
- Code commits to GitHub repository daily

End of Work Breakdown Structure

Document Version: 1.0
Last Updated: November 22, 2025
Status: Ready for Implementation

How to Use This WBS

1. **Track Progress**: Check off tasks as completed
2. **Estimate Time**: Add your own time estimates for each task
3. **Assign Priorities**: Mark tasks as High/Medium/Low priority
4. **Identify Dependencies**: Note which tasks must be completed before others
5. **Update Regularly**: Keep this document current as project evolves
6. **Reference for Reports**: Use for progress reports and presentations

Next Action: Begin Phase 3.0 (Development) with environment setup!