4-Week Development Schedule

Media Authentication & Watermarking Platform - Mini Project

Team Size: 4 Members
Timeline: ~25 Days (3.5 weeks)
Goal: Working MVP for Project Presentation

🎯 MVP SCOPE - What to Build

✅ MUST HAVE (Core Demo Features)

1. **Basic Image Upload & Download** (Frontend + Backend)
2. **Simplified Watermark Embedding** (Basic DWT + DCT)
3. **Watermark Extraction & Verification** (Extract embedded ID)
4. **Simple JWT Authentication** (Login/Register)
5. **PostgreSQL Database** (Users + Image Metadata)
6. **Basic UI Dashboard** (Upload, Extract, View Results)

⚠️ SIMPLIFIED (Reduce Complexity)

- **Skip QIM** - Use simpler LSB (Least Significant Bit) embedding instead
- **Skip Perceptual Fingerprinting** - Use simple SHA-256 hash comparison
- **Skip API Keys & Organizations** - Focus on user authentication only
- **Skip S3/Redis** - Store images temporarily on local filesystem
- **Skip Rate Limiting** - Basic validation only
- **Skip Kubernetes** - Use Docker Compose for deployment

❌ SKIP FOR NOW (Post-Presentation Enhancements)

- Advanced tamper detection scoring
- Bulk processing for organizations
- Comprehensive audit logging
- Admin panel
- API key management
- Horizontal scaling infrastructure

📅 WEEK-BY-WEEK BREAKDOWN

Week 1: Foundation & Core Backend (Days 1-7)

Team Assignment:

- **Person 1 & 2:** Backend Setup + Database
- **Person 3:** Frontend Setup + Basic UI
- **Person 4:** Research & Watermarking Algorithm

Deliverables:

Days 1-2: Project Setup

- [ ] Initialize Git repository
- [ ] Setup Go project structure (simplified version)
- [ ] Setup React project with Vite
- [ ] Create Docker Compose file (PostgreSQL + Backend + Frontend)
- [ ] Database schema design (Users, Images only)

Days 3-5: Core Backend

- [ ] PostgreSQL connection & migrations
- [ ] User model & authentication (JWT)
- [ ] Login/Register endpoints
- [ ] Image upload endpoint (save to `/tmp/uploads`)
- [ ] Basic file validation (MIME type, size)

Days 6-7: Watermarking Research & Prototype

- [ ] Implement basic DWT in Go (use library: `gonum` or similar)
- [ ] Implement basic DCT (8x8 blocks)
- [ ] Test LSB embedding in frequency domain
- [ ] Create standalone test script for embedding/extraction

Frontend Progress:

- [ ] Basic React setup with routing
- [ ] Login/Register pages
- [ ] Dashboard layout

Week 2: Watermarking Engine Integration (Days 8-14)

Team Assignment:

- **Person 1:** Embedding pipeline integration

- **Person 2:** Extraction pipeline integration
- **Person 3 & 4:** Frontend development

Deliverables:

Days 8-10: Embedding Pipeline

- [ ] Integrate DWT/DCT into backend API
- [ ] Create `/api/embed` endpoint
- [ ] Accept image + return watermarked image
- [ ] Generate unique Image ID (UUID)
- [ ] Store metadata in PostgreSQL (Image_ID, User_ID, Hash, Timestamp)
- [ ] Return watermarked image as download

Days 11-13: Extraction Pipeline

- [ ] Create `/api/extract` endpoint
- [ ] Accept watermarked image
- [ ] Extract embedded ID using inverse DWT/DCT
- [ ] Verify ID exists in database
- [ ] Return authenticity result (Valid/Invalid/Tampered)

Days 14: Testing & Bug Fixes

- [ ] Test embedding with different image formats (PNG, JPEG)
- [ ] Test extraction accuracy
- [ ] Handle edge cases (corrupted images, wrong format)

Frontend Progress:

- [ ] Image upload component with drag-and-drop
- [ ] Embed page (upload → process → download)
- [ ] Extract page (upload → verify → show results)
- [ ] Loading states and error handling

Week 3: Frontend Integration & Polish (Days 15-21)

Team Assignment:

- **Person 1 & 2:** Backend refinements + API testing
- **Person 3 & 4:** Frontend-Backend integration + UI/UX

Deliverables:

Days 15-17: Frontend-Backend Connection

- [ ] Connect login/register forms to backend
- [ ] Implement JWT storage (localStorage or httpOnly cookies)
- [ ] Connect embed page to `/api/embed`
- [ ] Connect extract page to `/api/extract`
- [ ] Display processing results (Image ID, timestamp, status)

Days 18-20: UI/UX Polish

- [ ] Responsive design (mobile-friendly)
- [ ] Better image preview (before/after)
- [ ] Verification results display (Valid ✅ / Invalid ❌)
- [ ] Image history page (show user's uploaded images)
- [ ] Simple statistics (total images embedded, extracted)

Days 21: Integration Testing

- [ ] End-to-end testing (embed → extract workflow)
- [ ] Cross-browser testing
- [ ] Fix critical bugs
- [ ] Performance optimization (image processing speed)

Week 4: Final Testing & Presentation Prep (Days 22-25)

Team Assignment:

- **All Members:** Testing, Documentation, Presentation

Deliverables:

Days 22-23: System Testing & Demo Scenarios

- [ ] Prepare demo dataset (5-10 test images)
- [ ] Test complete workflow multiple times
- [ ] Verify tampering detection (modify watermarked image slightly)
- [ ] Create test accounts
- [ ] Docker deployment testing

Days 24-25: Documentation & Presentation

- [ ] README with setup instructions
- [ ] Architecture diagram (simplified)
- [ ] API documentation (Postman collection)
- [ ] Presentation slides:
  - Problem statement
  - System architecture

- Watermarking algorithm explanation
- Live demo walkthrough
- Challenges faced & solutions

☐ Prepare backup demo video (in case live demo fails)

☐ Rehearse presentation (each member knows their part)

🚀 DEPLOYMENT STRATEGY

Simple Docker Compose Deployment

yaml

```yaml
services:
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: watermark_db
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: password
    volumes:
      - pgdata:/var/lib/postgresql/data

  backend:
    build: ./backend
    ports:
      - "8080:8080"
    depends_on:
      - postgres
    environment:
      DB_HOST: postgres
      JWT_SECRET: your-secret-key

  frontend:
    build: ./frontend
    ports:
      - "3000:3000"
    depends_on:
      - backend
```

🎬 DEMO SCENARIO FOR PRESENTATION

Scenario 1: Basic Watermark Embedding (2 minutes)

1. Login to the platform
2. Upload an original image (e.g., company logo)
3. Click "Embed Watermark"
4. Show the processing status
5. Download the watermarked image
6. Explain: "The image looks identical but contains a hidden unique ID"

Scenario 2: Watermark Extraction & Verification (2 minutes)

1. Upload the watermarked image from Scenario 1
2. Click "Extract Watermark"
3. Show the extracted information:
   - Image ID: `abc-123-def-456`
   - Original Upload Date: `2024-02-01`
   - Uploaded By: `john@example.com`
   - Status: ✅ **Authentic**
4. Explain: "The system verified this is the original watermarked image"

Scenario 3: Tamper Detection (2 minutes)

1. Open the watermarked image in an image editor
2. Make a small modification (add text, crop slightly)
3. Upload the modified image to extraction
4. Show the result:
   - Status: ⚠️ **Tampered**
   - Reason: "Hash mismatch detected"
5. Explain: "The system detected unauthorized modifications"

📊 TASK DISTRIBUTION RECOMMENDATION

Backend Developers (Person 1 & 2)

- **Person 1:** Authentication, Database, API endpoints
- **Person 2:** Watermarking engine (DWT/DCT/LSB)

Frontend Developers (Person 3 & 4)

- **Person 3:** Core pages (Login, Dashboard, Embed)
- **Person 4:** Extract page, Image History, UI components

Shared Responsibilities:

- Daily standup (15 min) - Progress updates
- Code reviews before merging
- Testing each other's features
- Documentation

⚙️ SIMPLIFIED TECHNICAL STACK

Backend

- **Framework:** Go Fiber (lightweight, fast)
- **Database:** PostgreSQL (users, image_metadata tables only)
- **Image Processing:** `imaging` library (resize, format conversion)
- **Watermarking:** Custom implementation (DWT + DCT + LSB)
- **Auth:** JWT with `golang-jwt` library

Frontend

- **Framework:** React 18 + Vite
- **Styling:** Tailwind CSS (rapid development)
- **HTTP Client:** Axios
- **State Management:** React Context (no Redux needed)
- **File Upload:** react-dropzone

Deployment

- **Containerization:** Docker + Docker Compose
- **Storage:** Local filesystem (`/tmp/uploads` and `/tmp/outputs`)

🛠️ SIMPLIFIED WATERMARKING ALGORITHM

Embedding (LSB in Frequency Domain)

1. Convert image to YCbCr
2. Apply DWT to Y channel → Get HL sub-band
3. Divide HL into 8x8 blocks
4. Apply DCT to each block
5. Embed UUID bits into LSBs of mid-frequency coefficients
6. Apply Inverse DCT
7. Apply Inverse DWT
8. Convert back to RGB

Extraction

1. Convert image to YCbCr
2. Apply DWT to Y channel → Get HL sub-band
3. Divide HL into 8x8 blocks
4. Apply DCT to each block
5. Extract LSBs from mid-frequency coefficients
6. Reconstruct UUID
7. Query database for UUID
8. Compare image hash for tamper detection

Why LSB instead of QIM?

- Much simpler to implement
- Good enough for demo purposes
- Less robust but faster development
- Can explain limitation in presentation

📑 PRESENTATION STRUCTURE (15-20 minutes)

1. Introduction (2 min)

- Problem: Image authenticity in digital age
- Solution: Invisible watermarking for media authentication

2. System Architecture (3 min)

- Show simplified architecture diagram
- Explain three-tier design

- Highlight PostgreSQL database schema

3. Watermarking Algorithm (4 min)

- Explain DWT + DCT theory (with diagrams)
- Show embedding pipeline
- Show extraction pipeline
- Mention limitations (LSB vs QIM)

4. Live Demo (6 min)

- Scenario 1: Embed watermark
- Scenario 2: Extract and verify
- Scenario 3: Tamper detection

5. Technical Challenges (2 min)

- Image format compatibility
- Performance optimization
- Balancing invisibility vs robustness

6. Future Enhancements (2 min)

- QIM implementation
- Perceptual fingerprinting
- Organization API keys
- Cloud deployment (K8s)

7. Q&A (3 min)

✅ SUCCESS CRITERIA FOR PRESENTATION

- [ ] System runs without crashes during demo
- [ ] Embedding works for JPEG and PNG
- [ ] Extraction successfully retrieves embedded ID
- [ ] Tamper detection shows difference between original and modified
- [ ] UI is clean and professional
- [ ] Team can explain the algorithm confidently
- [ ] Backup demo video ready (just in case)

🚨 RISK MITIGATION

If Running Behind Schedule:

Week 2 Issues:

- Use pre-built DWT/DCT libraries instead of implementing from scratch
- Simplify to LSB in spatial domain (even simpler, skip frequency transform)

Week 3 Issues:

- Use basic HTML forms instead of React components
- Skip image history page
- Focus only on embed and extract pages

Week 4 Issues:

- Use pre-recorded demo video instead of live demo
- Simplify presentation to focus on algorithm explanation

Backup Plan (Absolute Minimum):

- Single-page React app
- Two buttons: "Embed" and "Extract"
- Backend API with just two endpoints
- SQLite instead of PostgreSQL
- No authentication (demo mode only)

💡 PRO TIPS FOR SUCCESS

1. **Start with the simplest version first** - Get something working end-to-end by Week 2
2. **Test frequently** - Don't wait until Week 4 to test integration
3. **Use existing libraries** - Don't reinvent DWT/DCT from scratch
4. **Version control discipline** - Commit working code daily
5. **Prepare backup demo** - Record a perfect run in case live demo fails
6. **Practice presentation** - Rehearse at least 3 times before final day
7. **Document as you go** - Don't leave README for the last day
8. **Focus on the story** - Explain WHY the project matters, not just HOW it works

---

📚 RECOMMENDED LIBRARIES

Go Backend

```bash
go get github.com/gofiber/fiber/v2
go get github.com/golang-jwt/jwt/v5
go get gorm.io/gorm
go get gorm.io/driver/postgres
go get github.com/disintegration/imaging
go get gonum.org/v1/gonum  # For DWT/DCT
```

React Frontend

```bash
npm install react-router-dom axios
npm install react-dropzone
npm install tailwindcss
npm install @headlessui/react  # For modals, dialogs
```

🎓 LEARNING RESOURCES (Quick References)

- **DWT Tutorial:** https://pywavelets.readthedocs.io/en/latest/
- **DCT Basics:** https://en.wikipedia.org/wiki/Discrete_cosine_transform
- **Digital Watermarking:** Search "LSB watermarking tutorial"
- **Go Fiber Docs:** https://docs.gofiber.io/
- **React Dropzone:** https://react-dropzone.js.org/

Good luck with your project! Focus on getting the core demo working first, then add polish. A simple working system is better than a complex broken one. 🚀