

Priority Implementation Roadmap

FAZA 1: Kritična Infrastruktura (Danas - 7 Dana)

Dan 1-2: Core Encryption Setup

```
bash

# 1. Instaliraj dependencies
npm install jsonwebtoken speakeasy qrcode bcrypt uuid node-forge express-rate-limit helmet
npm install --save-dev @types/jsonwebtoken @types/speakeasy @types/qrcode @types/bcrypt @types/uuid @types/

# 2. Generiraj encryption keys
node -e "console.log('DIGITAL_KEY_ENCRYPTION_MASTER=' + require('crypto').randomBytes(32).toString('hex'))"
node -e "console.log('VAULT_MASTER_KEY=' + require('crypto').randomBytes(32).toString('hex'))"
node -e "console.log('JWT_SECRET=' + require('crypto').randomBytes(64).toString('hex'))"
```

Dan 3-4: Database Schema

```
sql
```

-- Prioritetne tabele za immediate implementation

```
CREATE TABLE digital_keys (  
  id SERIAL PRIMARY KEY,  
  key_id VARCHAR(64) UNIQUE NOT NULL,  
  encrypted_key TEXT NOT NULL,  
  key_fingerprint VARCHAR(32) NOT NULL,  
  product_id INTEGER,  
  user_id INTEGER,  
  created_at TIMESTAMP DEFAULT NOW(),  
  is_active BOOLEAN DEFAULT TRUE,  
  access_count INTEGER DEFAULT 0  
);
```

```
CREATE TABLE download_tokens (  
  id SERIAL PRIMARY KEY,  
  token VARCHAR(64) UNIQUE NOT NULL,  
  resource_id VARCHAR(100) NOT NULL,  
  user_id INTEGER,  
  created_at TIMESTAMP DEFAULT NOW(),  
  expires_at TIMESTAMP NOT NULL,  
  download_count INTEGER DEFAULT 0,  
  is_consumed BOOLEAN DEFAULT FALSE  
);
```

```
CREATE TABLE security_audit_logs (  
  id SERIAL PRIMARY KEY,  
  category VARCHAR(50) NOT NULL,  
  event VARCHAR(100) NOT NULL,  
  user_id INTEGER,  
  details JSONB DEFAULT '{}',  
  timestamp TIMESTAMP DEFAULT NOW(),  
  success BOOLEAN DEFAULT TRUE  
);
```

Dan 5-7: Minimal Viable Security (MVP)

Implementiraj samo core komponente:

Minimal Security Middleware:

typescript

```
// minimal-security.middleware.ts
```

```
import { Request, Response, NextFunction } from 'express';
```

```
import crypto from 'crypto';
```

```
import jwt from 'jsonwebtoken';
```

```
export class MinimalSecurityFramework {
```

```
  // 1. Basic encryption for license keys
```

```
  static encryptLicenseKey(key: string): string {
```

```
    const algorithm = 'aes-256-gcm';
```

```
    const secretKey = Buffer.from(process.env.DIGITAL_KEY_ENCRYPTION_MASTER!, 'hex');
```

```
    const iv = crypto.randomBytes(16);
```

```
    const cipher = crypto.createCipher(algorithm, secretKey);
```

```
    let encrypted = cipher.update(key, 'utf8', 'hex');
```

```
    encrypted += cipher.final('hex');
```

```
    const tag = cipher.getAuthTag();
```

```
    return `${iv.toString('hex')}:${tag.toString('hex')}:${encrypted}`;
```

```
  }
```

```
  static decryptLicenseKey(encryptedKey: string): string {
```

```
    const algorithm = 'aes-256-gcm';
```

```
    const secretKey = Buffer.from(process.env.DIGITAL_KEY_ENCRYPTION_MASTER!, 'hex');
```

```
    const [ivHex, tagHex, encrypted] = encryptedKey.split(':');
```

```
    const iv = Buffer.from(ivHex, 'hex');
```

```
    const tag = Buffer.from(tagHex, 'hex');
```

```
    const decipher = crypto.createDecipher(algorithm, secretKey);
```

```
    decipher.setAuthTag(tag);
```

```
    let decrypted = decipher.update(encrypted, 'hex', 'utf8');
```

```
    decrypted += decipher.final('utf8');
```

```
    return decrypted;
```

```
  }
```

```
  // 2. Basic JWT with scopes
```

```
  static generateToken(userId: string, scopes: string[]): string {
```

```
    return jwt.sign(
```

```
      { sub: userId, scopes, iat: Math.floor(Date.now() / 1000) },
```

```
      process.env.JWT_SECRET!,
```

```
      { expiresIn: '1h' } 
```

```
    );
```

```
  }
```

```

static validateToken(token: string, requiredScopes: string[] = []): any {
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET!) as any;
    const userScopes = decoded.scopes || [];
    const hasRequiredScopes = requiredScopes.every(scope => userScopes.includes(scope));

    return hasRequiredScopes ? decoded : null;
  } catch {
    return null;
  }
}

```

// 3. Basic one-time download tokens

```

private static downloadTokens = new Map<string, any>();

```

```

static generateDownloadToken(resourceId: string, userId: string): string {
  const token = crypto.randomBytes(32).toString('hex');
  const expiresAt = Date.now() + 15 * 60 * 1000; // 15 minutes

```

```

    this.downloadTokens.set(token, {
      resourceId,
      userId,
      expiresAt,
      used: false
    });

```

// Auto cleanup

```

    setTimeout(() => this.downloadTokens.delete(token), 15 * 60 * 1000);

```

```

    return token;
  }

```

```

static validateDownloadToken(token: string): { valid: boolean; resourceId?: string; userId?: string } {
  const tokenData = this.downloadTokens.get(token);

  if (!tokenData || tokenData.used || Date.now() > tokenData.expiresAt) {
    return { valid: false };
  }

```

```

  tokenData.used = true;
  this.downloadTokens.delete(token);

```

```

  return {
    valid: true,
    resourceId: tokenData.resourceId,
    userId: tokenData.userId
  };
}

```

```

}

// 4. Basic audit logging
static logSecurityEvent(event: string, details: any): void {
  console.log(`🔒 SECURITY: ${event}`, {
    timestamp: new Date().toISOString(),
    event,
    details
  });

  // Store in database when ready
  // db.query('INSERT INTO security_audit_logs (event, details) VALUES ($1, $2)', [event, details]);
}
}

// Quick middleware for immediate use
export const requireAuth = (requiredScopes: string[] = []) => {
  return (req: Request, res: Response, next: NextFunction) => {
    const authHeader = req.headers.authorization;
    if (!authHeader?.startsWith('Bearer ')) {
      return res.status(401).json({ error: 'Missing token' });
    }

    const token = authHeader.substring(7);
    const user = MinimalSecurityFramework.validateToken(token, requiredScopes);

    if (!user) {
      return res.status(403).json({ error: 'Invalid token or insufficient permissions' });
    }

    (req as any).user = user;
    next();
  };
};

export const protectDownload = (req: Request, res: Response, next: NextFunction) => {
  const { token } = req.params;
  const validation = MinimalSecurityFramework.validateDownloadToken(token);

  if (!validation.valid) {
    return res.status(401).json({ error: 'Invalid download token' });
  }

  (req as any).download = validation;
  next();
};

```

Quick Integration sa Postojećim Kodom:

typescript

// U vašem app.ts

```
import { MinimalSecurityFramework, requireAuth, protectDownload } from './middleware/minimal-security.middleware'
```

// Zaštiti postojeće rute

```
app.use('/api/licenses', requireAuth(['license:access']));
```

```
app.use('/api/admin', requireAuth(['admin:manage']));
```

// Test endpoints za immediate testing

```
app.post('/api/test/encrypt', (req, res) => {
```

```
  const { text } = req.body;
```

```
  const encrypted = MinimalSecurityFramework.encryptLicenseKey(text);
```

```
  res.json({ encrypted });
```

```
});
```

```
app.post('/api/test/decrypt', (req, res) => {
```

```
  const { encrypted } = req.body;
```

```
  try {
```

```
    const decrypted = MinimalSecurityFramework.decryptLicenseKey(encrypted);
```

```
    res.json({ decrypted });
```

```
  } catch (error) {
```

```
    res.status(400).json({ error: 'Decryption failed' });
```

```
  }
```

```
});
```

```
app.post('/api/test/token', (req, res) => {
```

```
  const { userId, scopes } = req.body;
```

```
  const token = MinimalSecurityFramework.generateToken(userId, scopes);
```

```
  res.json({ token });
```

```
});
```

```
app.get('/api/test/protected', requireAuth(['test:access']), (req, res) => {
```

```
  res.json({ message: 'Protected endpoint accessed', user: (req as any).user });
```

```
});
```

```
app.post('/api/test/download-token', requireAuth(), (req, res) => {
```

```
  const { resourceId } = req.body;
```

```
  const user = (req as any).user;
```

```
  const token = MinimalSecurityFramework.generateDownloadToken(resourceId, user.sub);
```

```
  res.json({ downloadToken: token, downloadUrl: `/api/test/download/${token}` });
```

```
});
```

```
app.get('/api/test/download/:token', protectDownload, (req, res) => {
```

```
  const download = (req as any).download;
```

```
  res.json({
```

```
    message: 'File downloaded',
```

```
    resourceId: download.resourceId,
```

```
    userId: download.userId
  });
});
```

FAZA 2: Immediate Testing (Dan 8-10)

Test Plan:

```
bash

# 1. Test encryption
curl -X POST http://localhost:3000/api/test/encrypt \
  -H "Content-Type: application/json" \
  -d '{"text":"LICENSE-KEY-12345"}'

# 2. Test JWT generation
curl -X POST http://localhost:3000/api/test/token \
  -H "Content-Type: application/json" \
  -d '{"userId":"user123","scopes":["license:access","test:access"]}'

# 3. Test protected endpoint
curl -X GET http://localhost:3000/api/test/protected \
  -H "Authorization: Bearer YOUR_JWT_TOKEN"

# 4. Test download flow
curl -X POST http://localhost:3000/api/test/download-token \
  -H "Authorization: Bearer YOUR_JWT_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"resourceId":"license123"}'

curl -X GET http://localhost:3000/api/test/download/YOUR_DOWNLOAD_TOKEN
```

Environment Setup for Testing:

```
env

# .env.local for testing
NODE_ENV=development
DIGITAL_KEY_ENCRYPTION_MASTER=your-64-char-hex-key-here
JWT_SECRET=your-jwt-secret-here
DATABASE_URL=your-test-database-url

# Test with relaxed security
SECURITY_STRICT_MODE=false
ENABLE_DEBUG_LOGS=true
```


FAZA 3: Gradual Enhancement (Dan 11-21)

Week 2: Add Advanced Features

- Secure Key Vault implementation
- Rate limiting by user tiers
- Basic fraud detection
- Admin 2FA setup

Week 3: Full Integration

- Update all existing routes
- Frontend security integration
- Comprehensive audit logging
- Automated security tasks

Immediate Success Metrics

Po završetku Faze 1 (7 dana), trebali biste imati:

- ✔ **Functional AES-256 encryption** za license keys
- ✔ **Working JWT authentication** s scope validation
- ✔ **One-time download tokens** fully operational
- ✔ **Basic audit logging** za security events
- ✔ **Test endpoints** za verification
- ✔ **Database schema** ready za production

Critical Success Factors

1. Environment Security

```
bash

# Generate strong keys
DIGITAL_KEY_ENCRYPTION_MASTER=$(openssl rand -hex 32)
JWT_SECRET=$(openssl rand -hex 64)
VAULT_MASTER_KEY=$(openssl rand -hex 32)
```

2. Database Security

```
sql

-- Ensure proper permissions
GRANT SELECT, INSERT, UPDATE, DELETE ON digital_keys TO app_user;
GRANT SELECT, INSERT, DELETE ON download_tokens TO app_user;
GRANT INSERT ON security_audit_logs TO app_user;
```

3. Error Handling

typescript

```
// Never expose encryption errors to client
try {
  const decrypted = MinimalSecurityFramework.decryptLicenseKey(encrypted);
  return decrypted;
} catch (error) {
  MinimalSecurityFramework.logSecurityEvent('DECRYPTION_FAILED', {
    error: 'Decryption attempt failed',
    timestamp: Date.now()
  });
  throw new Error('Invalid license key format');
}
```



Quick Win Strategy

Day 1: Setup + Basic encryption testing

Day 2: JWT implementation + testing

Day 3: Download tokens + basic audit

Day 4: Integration s postojećim rutama

Day 5: Frontend integration testing

Day 6: Security testing + bug fixes

Day 7: Documentation + deployment prep

Ovaj approach omogućava vam da **odmah počnete s osnovnom security funkcionalnosti** dok gradite prema full enterprise solution.

Želite li da počnemo s **minimal implementation** za brzo testiranje ili idemo direktno na **full enterprise solution**?