**Initial Setup:**

* Created React app with create-react-app
* Set up backend with Express.js and SQLite
* Decided on Electron for desktop app functionality
* Chose styled-components for CSS-in-JS styling

**Key Decisions Made:**

* SQLite over PostgreSQL for simplicity and Electron compatibility
* JWT authentication over sessions for stateless API
* File-based uploads over cloud storage for MVP
* Python email service for complex email operations

**ARCHITECTURE DECISIONS & RATIONALE**

**Frontend Architecture:**

**javascript**

// src/services/api.js - Line 1-6

*const* API\_BASE\_URL = 'http://localhost:5000/api';

*class* ApiService {

*constructor*() {

        this.token = localStorage.getItem('monexa\_token');

    }

**Reason:**

* **Singleton pattern** for API service to avoid multiple instances
* **localStorage** for token persistence across browser sessions
* **Class-based** structure for better organization and method grouping

**Backend Architecture:**

**javascript**

// backend/server.js - Main server setup

*const* app = express();

*const* PORT = process.env.PORT || 5000;

// Security middleware

app.use(helmet());

app.use(compression());

**Security :**

* **Helmet.js** for security headers
* **CORS** configuration for cross-origin requests
* **Rate limiting** to prevent abuse
* **Input validation** with express-validator

**AUTHENTICATION SYSTEM DEVELOPMENT**

**JWT Implementation:**

**javascript**

// backend/middleware/auth.js - Line 7-13

*const* generateToken = (*userId*, *sessionId*) *=>* {

    return jwt.sign(

        { userId, sessionId },

        JWT\_SECRET,

        { expiresIn: '30d' }

    );

};

**Development :**

* **30-day expiration** for better UX (users don't need to login frequently)
* **Session tracking** in database for security and logout functionality
* **2FA support** using Speakeasy library for enhanced security

**Session Management I/S:**

**1: Session Timeout**

javascript

// src/contexts/AuthContext.js - Line 33-48

useEffect(() *=>* {

*const* checkSessionTimeout = () *=>* {

*const* now = Date.now();

*const* timeoutMs = sessionTimeout \* 60 \* 1000;

*const* timeSinceActivity = now - lastActivity;

        if (user && timeSinceActivity > timeoutMs && !isSessionLocked) {

            lockSession();

        }

    };

*const* interval = setInterval(checkSessionTimeout, checkInterval);

    return () *=>* clearInterval(interval);

}, [user, lastActivity, sessionTimeout, isSessionLocked]);

**Issue:**

* Initially set timeout to 30 minutes, but users complained about frequent logouts
* Activity tracking wasn't comprehensive enough

**fix:**

* Added more event listeners: ['mousedown', 'mousemove', 'keypress', 'scroll', 'touchstart', 'click']
* Made timeout configurable per user
* Added session lock screen instead of immediate logout

**MEDIA PLAYER DEVELOPMENT CHALLENGES**

**The YouTube Music Inspiration:**

I wanted to create a media player that felt familiar to users, so I studied YouTube Music's interface extensively.**Key Features Implemented:**

javascript

// src/components/MediaPlayer.js - Line 17-27

*const* [isMaximized, setIsMaximized] = useState(false);

*const* [isPlaying, setIsPlaying] = useState(false);

*const* [currentTime, setCurrentTime] = useState(0);

*const* [duration, setDuration] = useState(0);

*const* [volume, setVolume] = useState(0.7);

*const* [isMuted, setIsMuted] = useState(false);

*const* [isLoading, setIsLoading] = useState(true);

*const* [error, setError] = useState(null);

*const* [showControls, setShowControls] = useState(true);

*const* [mediaUrl, setMediaUrl] = useState(null);

*const* [showMinimizedControls, setShowMinimizedControls] = useState(false);

**Major Media Player Issues:**

**1: File Authentication**

javascript

// src/components/MediaPlayer.js - Line 52-56

*const* response = await fetch(`http://localhost:5000/api/documents/${document.id}/stream`, {

    headers: {

        'Authorization': `Bearer ${localStorage.getItem('monexa\_token')}`

    }

});

**Problem:** Media files weren't loading due to missing authentication headers

**Solution:** Added JWT token to all media streaming requests

**2: Memory Management**

javascript

// src/components/MediaPlayer.js - Line 81-85

return () *=>* {

    if (mediaUrl) {

        URL.revokeObjectURL(mediaUrl);

    }

};

**Problem:** Memory leaks from blob URLs not being cleaned up

**Solution:** Added cleanup function to revoke object URLs when component unmounts

**3: Window State Management:** The minimized player was tricky to implement. I wanted it to feel like YouTube Music's minimized player.

**Final Implementation:**

* **Windowed Mode**: 800x600px with full controls
* **Minimized Mode**: 300x230px with hover controls
* **Fullscreen Mode**: Full viewport experience

**FINANCIAL SYSTEM DEVELOPMENT**

**Database Schema Design:**

**sql**

-- backend/database/schema.sql - Line 52-67

CREATE TABLE IF NOT EXISTS transactions (

    id *TEXT* PRIMARY KEY,

    user\_id *TEXT* NOT NULL,

    type *TEXT* NOT NULL, -- income, expense, transfer

    category *TEXT* NOT NULL, -- beat\_sale, subscription, withdrawal, etc.

    title *TEXT* NOT NULL,

    description *TEXT*,

    amount *DECIMAL*(10,2) NOT NULL,

    currency *TEXT* DEFAULT 'USD',

    status *TEXT* DEFAULT 'completed', -- pending, completed, failed, cancelled

    reference *TEXT*,

    card\_id *TEXT*,

    created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,

    FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE,

    FOREIGN KEY (card\_id) REFERENCES cards(id) ON DELETE SET NULL

);

**Design Decisions:**

* **DECIMAL(10,2)** for precise financial calculations
* **CASCADE deletes** for data integrity
* **Status tracking** for transaction lifecycle
* **Reference field** for external transaction IDs

**Transaction Creation API:**

**javascript**

// backend/routes/transactions.js - Line 45-55

*const* transactionId = uuidv4();

await database.run(

    `INSERT INTO transactions (id, user\_id, type, amount, category, title, description, status)

     VALUES (?, ?, ?, ?, ?, ?, ?, ?)`,

    [transactionId, req.user.id, type, amount, category, title, description, status]

);

**UUID over auto-increment:**

* **Security**: Can't guess transaction IDs
* **Distributed**: Works well with multiple instances
* **URL-friendly**: Can be used in URLs without exposing sequence

**VAULT SYSTEM DEVELOPMENT**

**The Vault Concept:**

I wanted to create a system where music producers could organize their content in different "vaults" - each with its own purpose and security level.

**Beat Lock Vault :**

**javascript**

// src/pages/vaults/BeatLockVault.js - Line 45-55

*const* handleAuth = async (*e*) *=>* {

    e.preventDefault();

    setIsLoading(true);

    try {

        if (password === 'beatlock123') { // Demo password

            setIsAuthenticated(true);

            toast.success('Vault unlocked successfully!');

        } else {

            toast.error('Invalid password');

        }

    } finally {

        setIsLoading(false);

    }

};

**Development :**

* **Demo password** for testing - in production, this would be user-specific
* **Loading states** for better UX during authentication
* **Toast notifications** for immediate feedback

**Vault Authentication Overlay:**

**javascript**

// src/pages/vaults/BeatLockVault.js - Line 32-44

*const* AuthOverlay = styled(motion.div)`

  position: fixed;

  top: 0;

  left: 0;

  right: 0;

  bottom: 0;

  background: url('/images/userCheck.jpg') center/contain no-repeat;

  background-color: #f8fafc;

  z-index: 1000;

  display: flex;

  align-items: center;

  justify-content: center;

`;

**Design Choice:** Used background images to create immersive vault experiences

**Challenge:** Finding the right balance between security and UX

**EMAIL SYSTEM DEVELOPMENT**

**Hybrid Email Architecture:**

I decided to use a hybrid approach - Node.js for basic operations and Python for complex email processing.

javascript

// backend/services/emailService.js - Line 85-110

async sendBeatEmail(userEmail, userName, subject, message, attachmentData = null) {

    try {

*const* result = await this.\_callPythonServiceWithTempFile('send\_beat\_email', {

            userEmail, userName, subject, message, attachmentData

        });

        return result;

    } catch (error) {

        console.error('Email service error:', error);

        throw error;

    }

}

**Python for email:**

* **Better libraries** for email processing (email, smtplib)
* **Template processing** capabilities
* **File handling** for attachments
* **Error handling** more robust

**Email Configuration Issues:**

**markdown**

**# backend/EMAIL\_SETUP.md - Line 125-141**

**1. "Authentication failed" error**

- **Solution**: Make sure you're using an App Password, not your regular password

- **For Gmail**: Generate a new App Password in Google Account settings

**2. "Connection timeout" error**

- **Solution**: Check your firewall settings

- **Solution**: Verify SMTP server and port are correct

**3. "Python not found" error**

- **Solution**: Make sure Python is installed and in your PATH

- **Solution**: Test with `python --version` in terminal

**Common Issues Encountered:**

* Gmail App Password requirements
* Python path issues on Windows
* SMTP connection timeouts
* Email going to spam folders

**STYLING & UI DEVELOPMENT**

**Global Styles Approach:**

**css**

/\* src/styles/GlobalStyles.css - Line 1-13 \*/

\* {

*margin*: 0;

*padding*: 0;

*box-sizing*: border-box;

}

html, body {

*height*: 100%;

*font-family*: 'Inter', -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', sans-serif;

*-webkit-font-smoothing*: antialiased;

*-moz-osx-font-smoothing*: grayscale;

}

**Design System Decisions:**

* **Inter font** for modern, clean typography
* **CSS reset** for consistent cross-browser behavior
* **Font smoothing** for better text rendering
* **Utility classes** for common patterns

**Toast Customizations:**

**css**

/\* src/styles/GlobalStyles.css - Line 134-153 \*/

.Toastify\_\_toast {

*border-radius*: 12px !important;

*font-family*: 'Inter', sans-serif !important;

}

.Toastify\_\_toast--success {

*background*: linear-gradient(135deg, #10b981, #059669) !important;

}

.Toastify\_\_toast--error {

*background*: linear-gradient(135deg, #ef4444, #dc2626) !important;

}

**custom toast styles:**

* **Brand consistency** with app design
* **Better visual hierarchy** with gradients
* **Improved readability** with custom fonts

**API SERVICE DEVELOPMENT**

**Generic Request Method:**

**javascript**

// src/services/api.js - Line 33-57

async request(endpoint, options = {}) {

*const* url = `${API\_BASE\_URL}${endpoint}`;

*const* config = {

        headers: this.getHeaders(),

        ...options,

    };

    try {

*const* response = await fetch(url, config);

*const* data = await response.json();

        if (!response.ok) {

            // Special handling for 2FA required response

            if (response.status === 401 && data.requires2FA) {

                return data; // Return the data instead of throwing error

            }

            throw new Error(data.error || `HTTP error! status: ${response.status}`);

        }

        return data;

    } catch (error) {

        console.error('API Request failed:', error);

        throw error;

    }

}

**Key Features:**

* **Generic method** for all HTTP requests
* **Automatic token handling** in headers
* **2FA special handling** for authentication flow
* **Consistent error handling** across all requests

**FormData Handling:**

**javascript**

// src/services/api.js - Line 73-102

async postFormData(endpoint, formData) {

    try {

        console.log('API postFormData called with endpoint:', endpoint);

        console.log('Token available:', !!this.token);

*const* response = await fetch(`${API\_BASE\_URL}${endpoint}`, {

            method: 'POST',

            headers: {

                ...(this.token && { Authorization: `Bearer ${this.token}` })

                // Don't set Content-Type for FormData, let the browser set it with boundary

            },

            body: formData

        });

        console.log('Upload response status:', response.status, response.statusText);

        // ... rest of the method

    } catch (error) {

        console.error('FormData request failed:', error);

        throw error;

    }

}

**Critical Learning:** Don't set Content-Type header for FormData - let the browser set it with the boundary parameter.

**ELECTRON INTEGRATION**

**Main Process Setup:**

**javascript**

// public/electron.js - Main window creation

*function* createWindow() {

*const* mainWindow = new BrowserWindow({

        width: 1200,

        height: 800,

        webPreferences: {

            nodeIntegration: false,

            contextIsolation: true,

            preload: path.join(\_\_dirname, 'preload.js')

        }

    });

}

**Security Considerations:**

* **nodeIntegration: false** for security
* **contextIsolation: true** to prevent context pollution
* **Preload script** for secure IPC communication

**Electron Build Configuration:**

**json**

// package.json - Line 39-54

"build": {

*"appId"*: "com.monexa.app",

*"productName"*: "Monexa",

*"directories"*: {

*"output"*: "dist"

    },

*"files"*: [

        "build/\*\*/\*",

        "node\_modules/\*\*/\*",

        "public/electron.js"

    ],

*"win"*: {

*"target"*: "nsis",

*"icon"*: "public/icon.ico"

    }

}

**Build Process:**

1. React build creates build/ directory
2. Electron builder packages everything
3. NSIS installer for Windows distribution

**MAJOR BUGS & DEBUGGING SESSIONS**

**Bug 1: Media Player Not Loading Files**

**Issue:** Media player showed loading state indefinitely

**Debug:**

1. Checked network tab - 401 Unauthorized errors
2. Verified JWT token was being sent
3. Found token was expired
4. Implemented token refresh mechanism

**Solution:**

javascript

// Added token refresh in API service

async refreshSession() {

    return this.post('/auth/refresh');

}

**Bug 2: File Upload Failing**

**Issue:** File uploads would fail with "Upload failed" error

**Debug:**

1. Checked FormData construction
2. Verified multer configuration
3. Found Content-Type header conflict
4. Removed manual Content-Type setting

**Solution:**

javascript

// Removed Content-Type header for FormData

headers: {

    ...(this.token && { Authorization: `Bearer ${this.token}` })

    // Don't set Content-Type for FormData, let the browser set it with boundary

}

**Bug 3: Session Timeout Issues**

**Issue:** Users getting logged out unexpectedly

**Debug :**

1. Checked activity tracking events
2. Found missing event listeners
3. Verified timeout calculation
4. Added more comprehensive activity tracking

**Solution:**

javascript

// Added more event listeners

*const* events = ['mousedown', 'mousemove', 'keypress', 'scroll', 'touchstart', 'click'];

events.forEach(*event* *=>* {

    document.addEventListener(event, handleActivity, true);

});

**PERFORMANCE OPTIMIZATIONS**

**Database Optimizations:**

**sql**

-- backend/database/schema.sql - Line 227-249

CREATE INDEX IF NOT EXISTS idx\_users\_email ON users(email);

CREATE INDEX IF NOT EXISTS idx\_sessions\_user\_id ON user\_sessions(user\_id);

CREATE INDEX IF NOT EXISTS idx\_sessions\_token ON user\_sessions(token);

CREATE INDEX IF NOT EXISTS idx\_cards\_user\_id ON cards(user\_id);

CREATE INDEX IF NOT EXISTS idx\_transactions\_user\_id ON transactions(user\_id);

CREATE INDEX IF NOT EXISTS idx\_transactions\_type ON transactions(type);

CREATE INDEX IF NOT EXISTS idx\_transactions\_created\_at ON transactions(created\_at);

**Why these indexes:**

* **Email lookups** for authentication
* **Session validation** for security
* **User-specific queries** for data isolation
* **Date-based queries** for transaction history

**Frontend Optimizations:**

**javascript**

// Used React.memo for expensive components

*const* MediaPlayer = React.memo(({ *document*, *onClose*, *onDelete*, *onDownload* }) *=>* {

    // Component logic

});

// Used useCallback for event handlers

*const* handlePlay = useCallback(() *=>* {

    setIsPlaying(true);

    mediaRef.current.play();

}, []);

**SECURITY IMPLEMENTATION**

**Password Hashing:**

**javascript**

// backend/routes/auth.js - Password hashing

*const* saltRounds = 12;

*const* hashedPassword = await bcrypt.hash(password, saltRounds);

**Security Level:** 12 salt rounds for strong hashing

**Rate Limiting:**

**javascript**

// backend/server.js - Rate limiting

*const* rateLimit = require('express-rate-limit');

*const* limiter = rateLimit({

    windowMs: 15 \* 60 \* 1000, // 15 minutes

    max: 100 // limit each IP to 100 requests per windowMs

});

app.use('/api/', limiter);

**Protection:** 100 requests per 15 minutes per IP

**Input Validation:**

**javascript**

// backend/routes/auth.js - Input validation

router.post('/signup', [

    body('name').trim().isLength({ min: 2 }).withMessage('Name must be at least 2 characters'),

    body('email').isEmail().normalizeEmail().withMessage('Valid email required'),

    body('password').isLength({ min: 6 }).withMessage('Password must be at least 6 characters'),

    body('role').optional().trim()

], async (*req*, *res*) *=>* {

*const* errors = validationResult(req);

    if (!errors.isEmpty()) {

        return res.status(400).json({

            success: false,

            error: 'Validation failed',

            details: errors.array()

        });

    }

    // ... rest of the handler

});

**RESPONSIVE DESIGN CHALLENGES**

**Sidebar Responsiveness:**

**javascript**

// src/components/Sidebar.js - Responsive behavior

*const* SidebarContainer = styled.div`

    width: ${*props* *=>* props.isCollapsed ? '80px' : '280px'};

    transition: width 0.3s ease;

@media (max-width: 768px) {

        width: ${*props* *=>* props.isCollapsed ? '0' : '280px'};

        position: fixed;

        z-index: 1000;

    }

`;

**Mobile Strategy:** Fixed positioning with overlay for mobile devices

**Media Player Responsiveness:**

**javascript**

// src/components/MediaPlayer.js - Responsive sizing

*const* PlayerContainer = styled.div`

    width: ${*props* *=>* props.isMinimized ? '300px' : '800px'};

    height: ${*props* *=>* props.isMinimized ? '230px' : '600px'};

    @media (max-width: 768px) {

        width: 100vw;

        height: 100vh;

        position: fixed;

        top: 0;

        left: 0;

    }

`;

**Mobile Strategy:** Full-screen player on mobile devices

**DEPLOYMENT PREPARATION**

**Environment Configuration:**

**javascript**

// backend/server.js - Environment handling

*const* PORT = process.env.PORT || 5000;

*const* NODE\_ENV = process.env.NODE\_ENV || 'development';

// Don't leak error details in production

*const* isDevelopment = process.env.NODE\_ENV === 'development';

res.status(err.status || 500).json({

    success: false,

    error: isDevelopment ? err.message : 'Internal server error',

    ...(isDevelopment && { stack: err.stack })

});

**Production Considerations:**

* **Error message sanitization** for security
* **Environment-specific configurations**
* **Logging levels** based on environment

**Build Scripts:**

**json**

// package.json - Build configuration

"scripts": {

*"start"*: "react-scripts start",

*"build"*: "react-scripts build",

*"electron"*: "electron .",

*"electron-dev"*: "concurrently \"npm start\" \"wait-on http://localhost:3000 && electron .\"",

*"electron-pack"*: "npm run build && electron-builder",

*"dist"*: "npm run build && electron-builder --publish=never"

}

**Build Process:**

1. npm run build - Creates production React build
2. electron-builder - Packages Electron app
3. dist - Creates distributable packages

**FUTURE ENHANCEMENTS PLANNED**

**Short Term (Next 2-3 months):**

* **Real-time notifications** with WebSocket integration
* **Advanced analytics** for financial insights
* **Mobile app** with React Native
* **Cloud storage** integration (AWS S3)

**Medium Term (6 months):**

* **AI-powered** content recommendations
* **Advanced security** with biometric authentication
* **Multi-currency** support
* **Advanced reporting** with charts and graphs

**Long Term (1 year):**

* **Microservices architecture** for scalability
* **GraphQL API** for flexible data fetching
* **Progressive Web App** features
* **Advanced caching** with Redis

**KEY LEARNINGS & INSIGHTS**

**Technical Learnings:**

1. **JWT vs Sessions:** JWT is great for stateless APIs but requires careful token management
2. **File Uploads:** FormData requires special handling - don't set Content-Type manually
3. **Electron Security:** Always use contextIsolation and preload scripts
4. **Database Design:** Indexes are crucial for performance, especially for user-specific queries

**UX Learnings:**

1. **Session Timeouts:** Users prefer longer sessions with activity-based tracking
2. **Media Players:** Familiar interfaces (YouTube Music style) reduce learning curve
3. **Vault System:** Users like organized content with different security levels
4. **Toast Notifications:** Immediate feedback is crucial for user confidence

**Development Process:**

1. **Start Simple:** Begin with basic functionality and iterate
2. **Security First:** Implement security measures from the beginning
3. **User Testing:** Get feedback early and often
4. **Documentation:** Keep detailed notes for future reference

**DEVELOPMENT TOOLS & WORKFLOW**

**Code Quality:**

* **ESLint** for code linting
* **Prettier** for code formatting
* **Consistent naming** conventions
* **Comprehensive comments** for complex logic
* **Postman** for API testing

**\*\* MASTER DEV IMPLEMENTATIONS:**

* **User feedback** collection and analysis
* **Performance optimization** based on real usage
* **Feature prioritization** based on user needs
* **Security audit** and penetration testing