📚 AWIBI MEDTECH - Project Structure Guide

Version: 1.0.0
Date: July 9, 2025
Author: Manus AI
Status: Draft

🎯 Purpose of this Document

This guide is designed to provide a comprehensive, layman-friendly explanation of the AWIBI MEDTECH application's project structure. Understanding the organization of a project is the foundational step for any developer, especially beginners. This document will walk you through the 'why' behind each directory and file, explaining their purpose, the logic behind their placement, and how they contribute to the overall functionality and maintainability of the application. By the end of this guide, you will not only know 'what' each part of the project is, but 'why' it is structured that way, enabling you to confidently navigate, modify, and build upon this codebase.

🏗️ Overall Project Architecture

The AWIBI MEDTECH application follows a typical full-stack architecture, separating the frontend (client-side) and backend (server-side) into distinct directories. This separation of concerns is a fundamental principle in modern web development, promoting modularity, scalability, and independent development of each part. [1]

```
awibi-medtech/
├── frontend/                # Contains the React.js client-side application
├── backend/                 # Contains the Node.js/Express server-side
application
├── docs/                    # Comprehensive documentation for the project
├── README.md                # General project overview and setup instructions
├── DEPLOYMENT-GUIDE.md      # High-level deployment instructions
├── PROJECT-SUMMARY.md       # A brief summary of the project
└── todo.md                  # Task tracking and progress
```

Why this Structure?

This top-level structure is chosen for several key reasons:

1. **Separation of Concerns:** By dividing the frontend and backend, we ensure that changes in one part of the application do not directly impact the other. This allows for independent development, testing, and deployment.
2. **Scalability:** Each part can be scaled independently. For example, if the frontend experiences high traffic, it can be scaled without affecting the backend, and vice-versa.
3. **Team Collaboration:** Different teams or developers can work on the frontend and backend simultaneously without significant conflicts.
4. **Technology Flexibility:** It allows for different technologies to be used for the frontend and backend. Here, React.js is used for the frontend and Node.js/Express for the backend, but they could easily be swapped out for other frameworks if needed.
5. **Deployment Simplicity:** Platforms like Vercel (for frontend) and Render (for backend) are designed to deploy such separated applications efficiently.

📁 Frontend Directory (`awibi-medtech/frontend/`)

This directory houses the entire client-side application, built using React.js. It's what users interact with directly in their web browsers. The `awibi-medtech-frontend` subdirectory contains the actual React project.

```
frontend/awibi-medtech-frontend/
├── public/                 # Static assets (images, favicon, index.html
template)
├── src/                    # Source code for the React application
│   ├── components/         # Reusable UI components
│   ├── contexts/           # React Context API for global state management
│   ├── data/               # Mock data or static data for the application
│   ├── lib/                # Utility functions and API client setup
│   ├── pages/              # Top-level components representing different
views/pages
│   ├── App.jsx             # Main application component and routing setup
│   ├── App.css             # Global CSS styles
│   └── main.jsx            # Entry point of the React application
├── .env                    # Environment variables for development
├── .env.production         # Environment variables for production
├── package.json            # Project metadata and dependencies
├── vite.config.js          # Vite build tool configuration
└── tailwind.config.js      # Tailwind CSS configuration
```

Why this Structure for Frontend?

This structure is typical for a modern React application, especially one built with Vite. It promotes:

1. **Modularity:** Breaking down the UI into smaller, reusable `components` makes the codebase easier to manage and understand. For example, a `Button` component can be used across multiple pages.
2. **Clear Separation of Concerns:** `pages` define the overall layout and functionality of a view, while `components` are the building blocks within those pages. `contexts` handle global state, `data` provides static information, and `lib` contains shared utilities.
3. **Maintainability:** When a bug needs to be fixed or a new feature added, it's easy to locate the relevant files. For instance, if there's an issue on the Dashboard, you'd look in `pages/DashboardPage.jsx` and its associated components.
4. **Scalability:** As the application grows, new components, pages, or contexts can be added without disrupting existing structures.

📁 Backend Directory (`awibi-medtech/backend/`)

This directory contains the Node.js and Express.js server-side application, responsible for handling API requests, interacting with the database, and managing business logic.

It's the brain of the application, providing data and services to the frontend.

```
backend/
├── config/                 # Configuration files (database connection,
Passport.js setup)
├── middleware/             # Express middleware for authentication,
authorization, security
├── models/                 # Mongoose models defining database schemas
├── routes/                 # API routes defining endpoints and their handlers
├── .env                    # Environment variables for development
├── .env.production         # Environment variables for production
├── package.json            # Project metadata and dependencies
├── server-final-production.js # Main server entry point for production
└── server-test.js          # A simplified server for testing purposes
```

Why this Structure for Backend?

This structure is a common and effective pattern for Node.js/Express applications, adhering to principles like:

1. **MVC-like Pattern (Model-View-Controller):** While Express is not strictly MVC, this structure loosely follows it:
   - **Models:** (`models/`) Define the structure of data and interact with the database.
   - **Views:** (Handled by the frontend in this case, as it's an API-only backend).
   - **Controllers/Routes:** (`routes/`) Define the API endpoints and the logic for handling requests and responses.
2. **Modularity:** Each directory has a specific responsibility. `middleware` handles cross-cutting concerns like authentication, `config` centralizes settings, and `routes` define API endpoints.
3. **Maintainability:** It's easy to find where a specific piece of logic resides. For example, if you need to change how users are authenticated, you'd look in `middleware/auth.js` or `config/passport.js`.
4. **Security:** Separating security-related logic into `middleware` ensures that security measures are applied consistently across relevant routes.
5. **Environment Management:** Dedicated `.env` files for different environments (`.env` for development, `.env.production` for production) ensure that sensitive information and configurations are managed securely and appropriately for each deployment context.

📄 Key Configuration Files

`package.json` (Frontend and Backend)

This file is present in both the `frontend/awibi-medtech-frontend` and `backend` directories. It's the heart of any Node.js project, serving as a manifest for the project. [2]

Purpose: - **Metadata:** Stores project information like name, version, description, and author. - **Dependencies:** Lists all third-party libraries (packages) that the project relies on. When you run `npm install`, npm reads this file and downloads all listed dependencies. - **Scripts:** Defines various command-line scripts that can be run using `npm run <script-name>`, such as `dev` (for development server), `build` (for creating production-ready code), or `start` (for running the production server).

Why it's Used: `package.json` ensures that everyone working on the project uses the same versions of libraries, preventing

compatibility issues. It also automates common tasks through scripts.

Example (Frontend `package.json` snippet):

```
{
  "name": "awibi-medtech-frontend",
  "version": "3.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-
warnings 0"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.8.1",
    "axios": "^1.6.0",
    "lucide-react": "^0.263.1"
  },
  "devDependencies": {
    "@vitejs/plugin-react": "^4.0.3",
    "vite": "^4.4.5",
    "tailwindcss": "^3.3.0",
    "autoprefixer": "^10.4.14",
    "postcss": "^8.4.24"
  }
}
```

Example (Backend `package.json` snippet):

```
{
  "name": "awibi-medtech-backend",
  "version": "3.0.0",
  "main": "server-final-production.js",
  "scripts": {
    "start": "node server-final-production.js",
    "dev": "nodemon server-final-production.js",
    "test": "node server-test.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "mongoose": "^7.5.0",
    "cors": "^2.8.5",
    "helmet": "^7.0.0",
    "bcryptjs": "^2.4.3",
    "jsonwebtoken": "^9.0.2",
    "passport": "^0.6.0",
    "passport-google-oauth20": "^2.0.0",
    "express-rate-limit": "^6.10.0",
    "dotenv": "^16.3.1",
    "express-validator": "^7.0.1",
    "express-mongo-sanitize": "^2.2.0"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

`.env` and `.env.production` Files (Frontend and Backend)

These files are crucial for managing environment-specific configurations, such as API keys, database connection strings, and other sensitive information. They are typically not committed to version control (e.g., Git) to prevent sensitive data from being exposed. [3]

Purpose: - **Security:** Keep sensitive data out of the public codebase. - **Flexibility:** Allow the application to behave differently based on the environment (development, testing, production) without changing the core code. - **Ease of Deployment:** Simplifies deployment by allowing environment variables to be set on the hosting platform (Vercel, Render) without modifying the application code.

Why they are Used: In development, you might connect to a local database and use less strict security settings. In production, you'll connect to a cloud database (like MongoDB Atlas) and use highly secure keys and settings. `.env` files allow you to switch between these configurations seamlessly.

Example (Frontend `.env` snippet):

```
# API Configuration
VITE_API_URL=http://localhost:5000

# Google OAuth Configuration
VITE_GOOGLE_CLIENT_ID=your-development-google-client-id
```

Example (Backend `.env` snippet):

```
# Application Configuration
NODE_ENV=development
PORT=5000

# Database Configuration
MONGODB_URI=mongodb://localhost:27017/awibi-medtech-dev

# JWT Configuration
JWT_SECRET=your-development-jwt-secret-key-minimum-32-characters
```

Example (Frontend `.env.production` snippet):

```
# API Configuration
VITE_API_URL=https://your-backend-app.onrender.com

# Google OAuth Configuration
VITE_GOOGLE_CLIENT_ID=your-production-google-client-id
```

Example (Backend `.env.production` snippet):

```
# Application Configuration
NODE_ENV=production
PORT=5000

# Database Configuration
MONGODB_URI=mongodb+srv://username:password@cluster.mongodb.net/awibi-medtech?
retryWrites=true&w=majority

# JWT Configuration
JWT_SECRET=your-super-secure-production-jwt-secret-key-minimum-64-characters
```

`vite.config.js` (Frontend)

This file is the configuration hub for Vite, the build tool used by the React frontend. Vite is known for its speed and efficiency in development and production builds. [4]

Purpose: - **Build Process:** Configures how the React application is built for production, including output directory, sourcemaps, and minification. - **Development Server:** Sets up the development server, including the port and host. - **Plugins:** Integrates various plugins, such as `@vitejs/plugin-react` for React support. - **Optimization:** Allows for advanced optimizations like code splitting (`manualChunks`) to improve loading performance.

Why it's Used: Vite streamlines the development workflow by providing instant server start and hot module replacement (HMR), meaning changes to your code are reflected in the browser almost instantly without a full page reload. For production, it optimizes the build output for maximum performance.

Example (`vite.config.js` snippet):

```
 import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  build: {
    outDir: 'dist',
    sourcemap: false,
    minify: 'terser',
    rollupOptions: {
      output: {
        manualChunks: {
          vendor: ['react', 'react-dom'],
          router: ['react-router-dom'],
          ui: ['lucide-react']
        }
      }
    }
  },
  server: {
    port: 5173,
    host: true
  }
})
```

`server-final-production.js` (Backend)

This is the main entry point for the backend application in a production environment. It sets up the Express server, connects to the database, applies middleware, and defines the API routes.

Purpose: - **Server Initialization:** Starts the Node.js server using Express. - **Database Connection:** Establishes a connection to the MongoDB database. - **Middleware Application:** Applies various middleware functions for security (CORS, Helmet, rate limiting), request parsing, and authentication. - **Route Definition:** Integrates all the API routes, making the backend endpoints accessible. - **Error Handling:** Sets up global error handling to gracefully manage unexpected errors.

Why it's Used: This file acts as the central orchestrator for the backend. It ensures that all necessary components (database, security, routes) are properly initialized and configured before the server starts listening for requests. For production, it's crucial to have a dedicated entry point that applies all necessary optimizations and security measures.

Example (`server-final-production.js` snippet - simplified for structure explanation):

```javascript
 const express = require('express');
const dotenv = require('dotenv');
const connectDB = require('./config/database');
const cors = require('cors');
const helmet = require('helmet');
const rateLimit = require('express-rate-limit');
const passport = require('passport');

// Load environment variables
dotenv.config({ path: './.env.production' });

// Connect to database
connectDB();

const app = express();

// CORS Configuration
const corsOptions = {
  origin: function (origin, callback) {
    const allowedOrigins = process.env.ALLOWED_ORIGINS?.split(',') || [];
    if (!origin || allowedOrigins.includes(origin)) {
      callback(null, true);
    } else {
      callback(new Error('Not allowed by CORS'));
    }
  },
  credentials: true,
  methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS', 'PATCH'],
  allowedHeaders: ['Content-Type', 'Authorization', 'X-Requested-With',
'Accept', 'Origin'],
  optionsSuccessStatus: 200
};
app.use(cors(corsOptions));

// Security Middleware
app.use(helmet());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Rate Limiting
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // limit each IP to 100 requests per windowMs
  message: 'Too many requests from this IP'
});
app.use('/api/', limiter);

// Passport middleware
app.use(passport.initialize());
require('./config/passport')(passport);

// Import Routes
const authRoutes = require('./routes/auth');
const userRoutes = require('./routes/users');
const chapterRoutes = require('./routes/chapters');
const eventRoutes = require('./routes/events');
const badgeRoutes = require('./routes/badges');
const dashboardRoutes = require('./routes/dashboard');
const analyticsRoutes = require('./routes/analytics');

// Use Routes
```

```
app.use('/api/auth', authRoutes);
app.use('/api/users', userRoutes);
app.use('/api/chapters', chapterRoutes);
app.use('/api/events', eventRoutes);
app.use('/api/badges', badgeRoutes);
app.use('/api/dashboard', dashboardRoutes);
app.use('/api/analytics', analyticsRoutes);

// Health Check Endpoint
app.get('/health', (req, res) => {
  res.json({ status: 'OK', message: 'AWIBI MEDTECH API is running' });
});

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

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

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

📚 References

[1] Martin, R. C. (2002). *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall. [2] npm. (n.d.). *package.json*. Retrieved from https://docs.npmjs.com/cli/v7/configuring-npm/package-json [3] The Twelve-Factor App. (n.d.). *III. Config*. Retrieved from https://12factor.net/config [4] Vite. (n.d.). *Why Vite*. Retrieved from https://vitejs.dev/guide/why.html

---

Next: Frontend Development Guide