🚀 Kolab Project

Understanding Your Folder Structure

A beginner-friendly guide to understanding where your code lives and where it actually runs

Next.js + AWS Architecture Explained Simply

FIU INIT Build Program • Fall 2025

📚 Introduction: The Big Picture

Think of your project like a restaurant:

- **Frontend (Next.js):** The dining area where customers sit and order
- **Backend (AWS Lambda):** The kitchen where food is prepared
- **Database (DynamoDB):** The storage room with all ingredients
- **Vercel:** The restaurant building that houses the dining area
- **AWS:** The kitchen facility and storage warehouse

🎯 Three Key Places Your Code Runs

1. User's Browser

What runs here: React components, buttons, forms, visual elements

Example: When a user types a message and clicks "Send"

2. Vercel Servers

What runs here: Page generation, API routes, initial HTML serving

Example: When someone visits kolab.vercel.app, Vercel sends them the webpage

3. AWS Cloud

What runs here: Database queries, file storage, authentication, business logic

Example: When a message is saved to the database or a user logs in

💡 **Key Concept:** Your code is split between these three places. Understanding where each piece runs helps you know where to look when debugging!

📁 Your Complete Project Structure

```
kolab/
├── frontend/ ← Your Next.js app (runs on Vercel + Browser)
│   ├── app/ ← Pages and routes
│   ├── components/ ← Reusable UI pieces
│   ├── lib/ ← Helper functions and AWS config
│   ├── public/ ← Images, icons,fonts
│   └── styles/ ← CSS styling
├── backend/ ← AWS Lambda functions (runs on AWS)
│   ├── resolvers/ ← Custom business logic
│   ├── triggers/ ← Authentication hooks
│   └── utils/ ← Shared helper functions
├── amplify/ ← AWS configuration (doesn't "run" - just config)
│   └── backend/ ← Defines your AWS infrastructure
│       ├── api/ ← GraphQL API   setup
│       ├── auth/ ← User authentication setup
│       ├── function/ ← Lambda function configs
│       └── storage/ ← File storage setup
└── docs/ ← Documentation (doesn't run - just reference)
    ├── architecture.md
    └── setup-guide.md
```

🎨 Color Coding Guide

Throughout this guide, we'll use these colors:

🟩 GREEN = Browser	⬛ BLACK = Vercel	🟧 ORANGE = AWS
Code that runs on the user's computer in their web browser	Code that runs on Vercel's servers when building or serving pages	Code that runs on Amazon Web Services cloud infrastructure

🎨 Frontend Folder (frontend/)

This is where **everything the user sees and interacts with** lives. It's your Next.js application.

📄 app/ - Your Pages and Routes

app/(auth)/login/page.tsx

Runs on: Vercel Then in: Browser

What it does: Creates the login page at `/login`

When it runs:

1. User visits kolab.vercel.app/login
2. Vercel builds the HTML and sends it to the user
3. React code runs in the user's browser to make the page interactive

```
// Example: app/(auth)/login/page.tsx
export default function
LoginPage() {
// This code runs in the USER'S BROWSER
return ( <div> <h1>Login to Kolab</h1> <input
placeholder="Email"
/> <button>Sign In</button> </div> ); }
```

🔧 app/api/upload/route.ts

Runs on: Vercel Servers

What it does: Creates an API endpoint at `/api/upload`

When it runs: When your frontend code calls this API

```
// Example: app/api/upload/route.ts
```

```
export async function
POST(request) {
// This code runs on VERCEL'S SERVERS
const file =
await request.formData();

// Can talk to AWS from here
await
uploadToS3(file);

return Response.json({ success:
true }); }
```

🤔 **When to use API routes:** Use them when you need to keep secrets (API keys) hidden from users, or when you need to do server-side processing before talking to AWS.

🧩 Components Folder (frontend/components/)

Components are **reusable pieces of your user interface**. They're like LEGO blocks you can use to build your pages.

💬 components/chat/MessageList.tsx

Runs in: User's Browser

What it does: Displays a list of messages in the chat

Why it's separate: You can use this component on multiple pages

```
// Example: components/chat/MessageList.tsx
export function
MessageList({
messages }) {
// This runs in the USER'S BROWSER
return ( <div> {messages.map(msg
=> ( <div
key={msg.id}> <p>{msg.content}</p> </div> ))} </div> ); }

// Used in your page like this:
// <MessageList messages={chatMessages} />
```

📂 Typical Component Organization

```
components/
```

```
    ├── ui/ ← Generic components used everywhere
    │   ├──Button.tsx ← Custom button styling
        ├── Input.tsx ← Custom input fields
    │   └── Modal.tsx ← Popup windows
    ├── chat/ ← Chat-specific components
    │   ├── MessageList.tsx
    │   ├── MessageInput.tsx
    │   └── ChatSidebar.tsx
    └──whiteboard/ ← Whiteboard-specific components
        └── Canvas.tsx
```

💡 **Think of it like this:** If you copy-paste the same code in multiple places, it should probably be a component!

🌉 Lib Folder (frontend/lib/) - The Bridge to AWS

This folder is **the bridge between your frontend and AWS**. It contains helper functions and AWS configuration.

⚙️ lib/amplify-config.ts

Runs in: User's Browser

What it does: Tells your app how to connect to AWS services

```
// Example: lib/amplify-config.ts
export const amplifyConfig = {
Auth: {
Cognito: {
userPoolId:
'us-east-1_xxxxx',
// Your AWS Cognito
userPoolClientId:
'xxxxx', } },
API: {
GraphQL: {
endpoint:
'https://xxxxx.appsync-api...',
// Your AWS AppSync
region:
'us-east-1', } } };
```

📝 lib/graphql/queries.ts

Runs in: User's Browser

What it does: Contains all your data-fetching queries

```
// Example: lib/graphql/queries.ts
```

```
export const getMessages =
` query GetMessages($groupId: ID!) { listMessages(groupId:
  $groupId) { items { id content sender timestamp } } } `;

// Used in your components like this:
// const messages = await client.graphql({ query: getMessages
  });
```

🔑 **Important:** The queries are defined here, but they're executed in the user's browser and sent directly to AWS AppSync. Vercel isn't involved in this data transfer!

⚙️ Backend Folder (backend/)

This folder contains **AWS Lambda functions** - code that runs in Amazon's cloud, not on Vercel or in the browser.

🔒 backend/triggers/preSignUp.ts

Runs on: AWS Lambda

What it does: Checks if an email is valid BEFORE letting someone sign up

When it runs: AWS Cognito automatically calls this when someone tries to register

```
// Example: backend/triggers/preSignUp.ts
export const
handler =
async (event) => {
// This runs on AWS LAMBDA (Amazon's servers)
const email =
event.request.userAttributes.email;

// Only allow FIU emails
if (!email.endsWith('@fiu.edu')) {
throw new
Error('Only FIU emails allowed!'); }

return event;
// Allow sign-up to continue };
```

💬 backend/resolvers/sendMessage.ts

Runs on: AWS Lambda

What it does: Custom logic for sending messages (like filtering bad words)

When it runs: AWS AppSync calls this when someone sends a message

```
// Example: backend/resolvers/sendMessage.ts
export const
```

```
handler =
async (event) => {
// This runs on AWS LAMBDA
const message = event.arguments.input;

// Custom business logic
const cleanContent =
filterBadWords(message.content);

// Save to database
await
saveToDatabase({ ...message,
content: cleanContent });

return {
success:
true }; };
```

⚡ **Key Difference:** Lambda functions run on AWS's computers, not on Vercel. They're triggered by AWS services (Cognito, AppSync) automatically!

🏗️ Amplify Folder (amplify/)

This folder **doesn't contain code that "runs"** - it's configuration that tells AWS how to set up your infrastructure.

📋 amplify/backend/api/kolab/schema.graphql

What it is: A blueprint for your database structure
What happens: When you run `amplify push`, AWS reads this and creates:

- DynamoDB tables
- GraphQL API endpoints
- Security rules

```
// Example: schema.graphql
type
Message @model {
id: ID!
content:
String!
sender:
String!
groupId:
ID!
timestamp:
AWSDateTime! }

// When you run 'amplify push', AWS automatically creates:
// - A DynamoDB table called "Message"
// - GraphQL queries to fetch messages
// - GraphQL mutations to create/update/delete messages
```

🔧 What's in amplify/backend/?

```
amplify/backend/ ├──
api/
← Defines your GraphQL API structure
├── auth/
← Configures Cognito (user authentication)
├── function/
← Lists your Lambda functions └──
storage/
← Configures S3 (file storage)
```

💡 **Think of it as:** A recipe book that tells AWS how to cook (set up) your backend infrastructure. The actual cooking happens when you run `amplify push`.

🔄 How Everything Connects: Complete Flow

Let's trace what happens when a user sends a message in your chat app.

Step 1: User Opens App

User types: `kolab.vercel.app`

VERCEL serves the Next.js app

↓

Step 2: App Loads in Browser

JavaScript from `frontend/` runs

USER'S BROWSER executes React components

↓

Step 3: User Types Message

Component: `components/chat/MessageInput.tsx`

Uses query from: `lib/graphql/mutations.ts`

USER'S BROWSER

↓

Step 4: Message Sent to AWS

Browser makes DIRECT request to AWS AppSync

(Vercel is NOT involved here!)

AWS APPSYNC

↓

Step 5: Custom Logic Runs (Optional)

Lambda function: `backend/resolvers/sendMessage.ts`

Filters bad words, validates content

AWS LAMBDA

↓

Step 6: Save to Database

Message stored in DynamoDB table

AWS DYNAMODB

↓

Step 7: Real-time Update

AppSync broadcasts to all connected users

AWS APPSYNC → ALL BROWSERS

↓

Step 8: UI Updates

Component: `components/chat/MessageList.tsx`

New message appears on screen

USER'S BROWSER

🎯 **Key Takeaway:** After the initial page load, your app talks DIRECTLY to AWS. Vercel's job is done after sending you the HTML/CSS/JavaScript!

📊 Quick Reference: What Runs Where?

Folder/File	Runs On	When It Runs
`app/(auth)/login/page.tsx`	Vercel + Browser	When user visits /login
`app/api/upload/route.ts`	Vercel	When API is called
`components/chat/MessageList.tsx`	Browser	Rendered on user's screen
`lib/amplify-config.ts`	Browser	On app initialization
`lib/graphql/queries.ts`	Browser	When fetching data
`backend/triggers/preSignUp.ts`	AWS Lambda	When user signs up
`backend/resolvers/sendMessage.ts`	AWS Lambda	When message is sent
`amplify/backend/api/schema.graphql`	*Configuration only*	Read by `amplify push`
`public/logo.png`	Vercel CDN	When image is requested

🎨 Visual Summary

USER'S BROWSER	VERCEL	AWS CLOUD
• React components	• Serves pages	• Lambda functions
• UI interactions	• API routes	• Database
• GraphQL queries	• Static assets	• Authentication

🎬 Common Scenarios Explained

Scenario 1: User Logs In

Step-by-step breakdown:

1. **User visits /login**
   - File: `app/(auth)/login/page.tsx`
   - Runs on: Vercel builds, then Browser displays
2. **User enters email/password and clicks "Sign In"**
   - Component: `components/auth/LoginForm.tsx`
   - Runs in: Browser
3. **Browser sends credentials to AWS Cognito**
   - Config from: `lib/amplify-config.ts`
   - Goes directly to: AWS Cognito
4. **Cognito checks if email ends with @fiu.edu**
   - Lambda: `backend/triggers/preSignUp.ts`
   - Runs on: AWS Lambda
5. **User receives JWT token and is logged in**
   - Token stored in: Browser memory
   - Used for all future AWS requests

Scenario 2: User Uploads a File

Step-by-step breakdown:

1. **User clicks "Upload" and selects file**
   - Component: `components/chat/FileUpload.tsx`
   - Runs in: Browser

2. **File sent to your API route (optional)**
    - File: `app/api/upload/route.ts`
    - Runs on: Vercel
    - Purpose: Validate file size, type, etc.
3. **File uploaded to AWS S3**
    - Storage: AWS S3
    - Config: `amplify/backend/storage/`
4. **File URL saved to database**
    - Mutation from: `lib/graphql/mutations.ts`
    - Saved to: AWS DynamoDB

Scenario 3: User Views Their Profile

Step-by-step breakdown:

1. **User clicks "Profile"**
    - Navigates to: `app/(dashboard)/profile/page.tsx`
    - Runs on: Vercel then Browser
2. **Page fetches user data**
    - Query from: `lib/graphql/queries.ts`
    - Sent to: AWS AppSync
3. **AppSync retrieves data from database**
    - Fetched from: AWS DynamoDB
4. **Data displayed on screen**
    - Component: `components/profile/ProfileCard.tsx`
    - Rendered in: Browser

🚀 Deployment: What Goes Where?

When you deploy your app, different parts go to different places.

Deploying to Vercel

📦 What Gets Deployed to Vercel

```
vercel deploy

Deploying these folders: ✅ frontend/app/
→ Becomes your website ✅
frontend/components/
→ Bundled into pages ✅
frontend/lib/
→ Bundled into pages ✅
frontend/public/
→ Served as static files ✅
frontend/styles/
→ Compiled CSS

NOT deployed to Vercel: ❌ backend/
→ Goes to AWS instead ❌
amplify/
→ Just configuration ❌
docs/
→ Documentation only
```

Deploying to AWS

☁️ What Gets Deployed to AWS

```
amplify push

Deploying these items: ✅
backend/resolvers/
→ AWS Lambda functions ✅
backend/triggers/
→ AWS Lambda functions ✅
amplify/backend/api/
→ Creates AppSync API ✅
amplify/backend/auth/
→ Creates Cognito User Pool ✅
amplify/backend/storage/
→ Creates S3 Bucket

Creates in AWS: → DynamoDB tables (from schema.graphql) → Lambda
functions (from backend/) → AppSync GraphQL API → Cognito User Pool
S3 Bucket for files
```

💡 **Two Separate Deployments:** You deploy to Vercel and AWS separately. They work together but are hosted independently!

📋 Deployment Checklist

Before deploying:

- ✅ Run `amplify push` to deploy AWS resources
- ✅ Update `lib/amplify-config.ts` with production AWS endpoints
- ✅ Set environment variables in Vercel dashboard
- ✅ Run `vercel deploy` to deploy frontend

- ✅ Test authentication and database connections

🔍 Troubleshooting: Where to Look

When something goes wrong, knowing where to look saves hours of debugging!

Problem: Page Won't Load

Where to check:

- Vercel Dashboard - Check build logs
- `app/layout.tsx` - Check for syntax errors
- `next.config.js` - Check configuration
- Browser Console (F12) - Check for JavaScript errors

Problem: Login Not Working

Where to check:

- AWS Cognito Console - Check user pool settings
- `lib/amplify-config.ts` - Verify correct User Pool ID
- `backend/triggers/preSignUp.ts` - Check email validation logic
- Browser Network Tab (F12) - Check API calls to Cognito

Problem: Messages Not Sending

Where to check:

- AWS AppSync Console - Check API logs
- `lib/graphql/mutations.ts` - Verify mutation syntax
- `backend/resolvers/sendMessage.ts` - Check Lambda logs in CloudWatch

- AWS DynamoDB Console - Verify table exists and has correct permissions

Problem: Files Not Uploading

Where to check:

- AWS S3 Console - Check bucket permissions
- `app/api/upload/route.ts` - Check upload logic (if using API route)
- `amplify/backend/storage/` - Verify storage configuration
- Browser Console - Check for CORS errors

🎯 **Quick Debug Tip:**

- Frontend issues? Check Browser Console (F12)
- API issues? Check Vercel Logs
- Backend issues? Check AWS CloudWatch

✨ Best Practices for Organizing Your Code

1. Keep Secrets Safe

❌ **NEVER do this:**

```
// DON'T put secrets in your code! const API_KEY =
"sk-1234567890abcdef"; // ❌ BAD!
```

✅ **DO this instead:**

```
// Use environment variables const API_KEY =
process.env.OPENAI_API_KEY; // ✅ GOOD! // Create .env.local file
(never commit this!) OPENAI_API_KEY=sk-1234567890abcdef
```

2. Component Organization

✅ **Good structure:**

```
components/ ├── ui/ ← Generic, reusable everywhere ├── chat/ ←
Feature-specific ├── whiteboard/ ← Feature-specific └── profile/ ←
Feature-specific
```

❌ **Poor structure:**

```
components/ ├── Button1.tsx ├── Button2.tsx ├── ChatThing.tsx ├──
```

```
RandomComponent.tsx ← What does this do?
```

3. File Naming Conventions

✅ **Good names:**

- `MessageList.tsx` - Clear what it does
- `useAuth.ts` - Custom hook for authentication
- `formatDate.ts` - Utility function

❌ **Bad names:**

- `Component1.tsx` - Too generic
- `utils.ts` - Too vague (what utilities?)
- `temp.tsx` - Should never be committed

4. Code Comments

✅ **Good comments:**

```
// Validate FIU email - only @fiu.edu allowed if
(!email.endsWith('@fiu.edu')) { throw new Error('Only FIU emails
allowed'); } // Fetch messages from last 24 hours const messages =
await getRecentMessages(Date.now() - 86400000);
```

❌ **Unnecessary comments:**

```
// Set x to 5 const x = 5; // ❌ Comment doesn't add value // Loop
through messages messages.forEach(msg => { ... }) // ❌ Code is
```

```
            self-explanatory
```

📋 Quick Reference Cheat Sheet

Common Commands

```
# Frontend (Next.js) Commands npm run dev
# Start development server npm run build
# Build for production npm run start
# Start production server vercel deploy
# Deploy to Vercel

# Backend (AWS) Commands
amplify init
# Initialize AWS Amplify amplify push
# Deploy to AWS amplify status
# Check deployment status amplify console
# Open AWS console

# Git Commands
git status # Check what changed git add .
# Stage all changes git commit -m
"msg"
# Commit with message git push
# Push to GitHub
```

Folder Quick Reference

Frontend Files

- `app/` - Pages
- `components/` - UI pieces
- `lib/` - Helpers
- `public/` - Images

Backend Files

- `backend/` - Lambda
- `amplify/` - Config
- `docs/` - Documentation

Where Things Run

React Components	Browser

API Routes	Vercel
Lambda Functions	AWS
GraphQL Queries	Browser → AWS
Database Queries	AWS

Environment Files

```
# .env.local (NEVER commit this!)
NEXT_PUBLIC_API_URL=https://your-api.com AWS_REGION=us-east-1
OPENAI_API_KEY=sk-xxxxx # Variables starting with NEXT_PUBLIC_ are #
accessible in browser (client-side) # Other variables are server-side
only
```

🔐 **Security Rule:** Never commit `.env.local` to Git! Always add it to `.gitignore`

🎓 Summary and Next Steps

What You Learned

1. Three Execution Environments

- User's Browser - UI components, user interactions
- Vercel Servers - Page building, API routes
- AWS Cloud - Database, authentication, business logic

2. Folder Structure Purpose

- `frontend/` - Everything users see and interact with
- `backend/` - Custom business logic in Lambda
- `amplify/` - Configuration for AWS infrastructure

3. Data Flow

User → Vercel (page load) → Browser → AWS (data) → Browser (display)

📚 Next Steps for Your Team

Week 1-2: Setup Phase

1. Each team member clones the repository
2. Set up AWS accounts and get credentials
3. Run `amplify init` to configure AWS
4. Run `npm install` to install dependencies

5. Start with small tasks in `components/`

Week 3-4: Core Features

1. Build authentication pages (`app/(auth)/`)
2. Create chat components (`components/chat/`)
3. Set up GraphQL queries (`lib/graphql/`)
4. Deploy to Vercel for first time

🆘 Getting Help

When you're stuck:

- Check this guide for where code runs
- Use browser console (F12) for frontend issues
- Check AWS CloudWatch for backend issues
- Ask team members - collaboration is key!
- Refer to official docs for Next.js and AWS Amplify

🎯 Remember

You've Got This!

Understanding where code runs is **75% of the battle**. Now that you know the structure, you can focus on building amazing features for Kolab!

Every expert was once a beginner who didn't give up 💪

Kolab Project - FIU INIT Build Program • Fall 2025
For questions or updates, refer to the project repository