AgencyFlow: Cetak Biru Teknis Full-Stack dengan Next.js dan PostgreSQL

Laporan ini menyajikan cetak biru teknis yang komprehensif untuk pengembangan platform Software as a Service (SaaS) "AgencyFlow". Dokumen ini dirancang untuk menjadi panduan bagi tim pengembang dalam membangun aplikasi yang skalabel, dapat dipelihara, dan aman, berdasarkan tumpukan teknologi modern yang terdiri dari Next.js untuk front-end dan back-end, serta PostgreSQL sebagai database. Analisis ini mencakup desain arsitektur, skema database, alur pengguna, dan praktik terbaik implementasi.

Bagian 1: Arsitektur Proyek Fondasional untuk AgencyFlow

Bagian ini menetapkan tulang punggung struktural aplikasi. Pilihan arsitektur yang dibuat di sini sangat penting untuk memastikan kemudahan pemeliharaan, skalabilitas, dan produktivitas pengembang dalam jangka panjang, terutama untuk platform SaaS multifaset seperti AgencyFlow.

1.1. Struktur Folder Berbasis Domain yang Skalabel

Prinsip inti dari arsitektur proyek AgencyFlow adalah adopsi struktur berbasis domain (atau berbasis fitur), yang beralih dari pengorganisasian tradisional berbasis tipe (misalnya, folder /components, /hooks di root). Pendekatan ini merupakan praktik terbaik modern untuk aplikasi berskala besar karena meningkatkan modularitas, menempatkan kode yang saling terkait berdekatan, dan mengurangi beban kognitif bagi pengembang.[1] Setiap fitur inti AgencyFlow, seperti

projects, invoicing, dan clients, akan menjadi modul mandiri yang kohesif.

Detail Implementasi:

- **Direktori src/:** Semua kode aplikasi akan ditempatkan di dalam direktori src/ untuk menjaga kebersihan direktori root dan memisahkan kode sumber dari file konfigurasi seperti next.config.js dan package.json.[2]
- **Direktori Fitur (src/features/):** Direktori ini akan menjadi pusat logika domain aplikasi. Setiap domain bisnis utama akan memiliki foldernya sendiri, misalnya: src/features/projects/, src/features/invoicing/, dan src/features/auth/.
- **Struktur Internal Fitur:** Di dalam setiap folder fitur, file akan diorganisir berdasarkan jenisnya: components/, hooks/, services/, utils/, dan types/. Struktur ini menempatkan semua kode yang terkait dengan satu domain di lokasi yang sama, membuatnya lebih mudah untuk dikelola, dipahami, dan direfaktor di masa depan.[2]
- **Direktori Bersama (src/shared/):** Sebuah direktori src/shared/ akan dibuat untuk menampung komponen UI yang benar-benar global (Button, Modal), hook kustom (use-local-storage), dan utilitas (lib/) yang digunakan di berbagai fitur. Ini mencegah duplikasi kode yang tidak perlu sambil mempertahankan batasan yang jelas antar modul.

Pilihan struktur berbasis domain ini bukan sekadar preferensi organisasi; ini adalah keputusan strategis yang secara langsung memungkinkan penskalaan tim di masa depan. Visi jangka panjang AgencyFlow adalah menjadi platform utama di Asia Tenggara, yang menyiratkan pertumbuhan tim pengembangan yang signifikan.[4] Tim-tim ini sering kali dibagi berdasarkan area produk (misalnya, tim "Keuangan & Penagihan", tim "Manajemen Proyek Inti"). Struktur berbasis domain (

src/features/invoicing/, src/features/projects/) menciptakan batasan kepemilikan kode yang jelas. Tim "Keuangan" akan bekerja terutama di dalam folder fitur mereka, meminimalkan konflik dan gangguan dengan tim lain. Dengan demikian, pilihan arsitektur ini adalah langkah fundamental dalam membangun organisasi rekayasa yang dapat menskalakan upayanya secara efisien.

Berikut adalah diagram visual dari struktur folder yang diusulkan:

```
/
├──.env.local
├── next.config.js
├── package.json
├── prisma/
│   └── schema.prisma
```

```
├── public/
│   └── ... (aset statis seperti gambar dan font)
└── src/
    ├── app/
    │   ├── (marketing)/
    │   │   ├── layout.tsx
    │   │   └── page.tsx      // Landing page
    │   ├── (auth)/
    │   │   ├── login/
    │   │   │   └── page.tsx
    │   │   └── layout.tsx
    │   ├── (app)/
    │   │   ├── dashboard/
    │   │   │   └── page.tsx
    │   │   ├── projects/
    │   │   │   ├── [projectId]/
    │   │   │   │   └── page.tsx
    │   │   │   └── page.tsx
    │   │   └── layout.tsx    // Layout utama aplikasi dengan sidebar/header
    │   ├── api/
    │   │   ├── projects/
    │   │   │   └── route.ts
    │   │   └── auth/
    │   │       └── [...nextauth]/
    │   │           └── route.ts
    │   └── layout.tsx        // Root layout
    ├── features/
    │   ├── projects/
    │   │   ├── components/   // Komponen spesifik untuk proyek (misal: KanbanBoard.tsx)
    │   │   ├── hooks/        // Hook spesifik untuk proyek (misal: useProjectTasks.ts)
    │   │   └── services/     // Logika bisnis sisi server (misal: createTask.ts)
    │   ├── invoicing/
    │   │   └── ... (struktur serupa)
    │   └── auth/
    │       └── ... (struktur serupa)
    └── shared/
        ├── components/
        │   └── ui/           // Komponen UI generik (misal: Button.tsx, Modal.tsx)
        ├── hooks/            // Hook global (misal: use-local-storage.ts)
        ├── lib/
        │   ├── db.ts         // Instansiasi singleton Prisma Client
        │   └── utils.ts      // Fungsi utilitas umum
        └── types/
```

```
└── index.ts      // Tipe data global
```

1.2. Strategi Routing dan Layout dengan App Router

Direktori src/app/ akan digunakan secara eksklusif untuk routing dan menyusun antarmuka pengguna (UI) dari modul-modul fitur, sesuai dengan praktik terbaik yang dianjurkan.[1] Penggunaan Next.js Route Groups akan menjadi kunci untuk mengorganisir berbagai bagian aplikasi tanpa memengaruhi struktur URL.

- **Grup (marketing):** Untuk halaman yang menghadap publik seperti halaman arahan (landing page), harga, dan tentang kami. Halaman-halaman ini akan berbagi layout pemasaran yang umum.
- **Grup (auth):** Didedikasikan untuk halaman otentikasi seperti masuk, daftar, dan lupa kata sandi. Grup ini akan menggunakan layout yang sederhana dan terpusat.
- **Grup (app):** Ini adalah area aplikasi utama untuk pengguna yang sudah terotentikasi. Grup ini akan memiliki layout persisten yang mencakup sidebar, header, dan area konten utama. Semua rute fitur inti (/dashboard, /projects/[id], /clients) akan berada di dalam grup ini.

Struktur ini secara bersih memisahkan kepentingan pemasaran publik, otentikasi pengguna, dan logika aplikasi inti, yang merupakan praktik kunci untuk routing yang dapat dipelihara.[2]

1.3. Strategi Rendering Hibrida

Untuk mengoptimalkan kinerja dan pengalaman pengguna, AgencyFlow akan menerapkan strategi rendering hibrida, memanfaatkan kekuatan penuh dari framework Next.js.

- **Static Site Generation (SSG):** Diterapkan untuk semua halaman dalam grup (marketing). Halaman-halaman ini kaya akan konten dan jarang berubah, menjadikannya kandidat ideal untuk di-render saat waktu build. Ini menghasilkan kecepatan pemuatan maksimum dan manfaat SEO yang optimal. Permintaan fetch untuk konten ini akan menggunakan pengaturan cache default { cache: 'force-cache' }.[5]
- **Server-Side Rendering (SSR) / Dynamic Rendering:** Digunakan untuk dasbor utama ((app)/dashboard) dan halaman detail proyek ((app)/projects/[id]). Halaman-halaman ini menampilkan data real-time yang spesifik untuk pengguna. Caching akan dinonaktifkan dengan menggunakan { cache: 'no-store' } dalam panggilan fetch atau dengan menggunakan fungsi dinamis seperti cookies() atau headers() untuk memastikan data

selalu segar pada setiap permintaan.[6]

- **Client-Side Rendering (CSR):** Diterapkan untuk komponen yang sangat interaktif, seperti papan Kanban (tampilan seperti Trello yang dijelaskan dalam dokumen ide proyek [4]). Halaman utama dapat di-render di server dengan kerangka pemuatan (loading skeleton), dan papan Kanban itu sendiri akan menjadi Client Component ('use client') yang mengambil datanya menggunakan pustaka seperti SWR atau React Query. Ini memungkinkan interaksi yang mulus seperti seret dan lepas (drag-and-drop) dan pembaruan UI optimis tanpa perlu memuat ulang seluruh halaman.[6]

1.4. Desain Lapisan API dengan Route Handlers

Lapisan API akan dibangun menggunakan Next.js Route Handlers yang ditempatkan di src/app/api/. Desainnya akan mengikuti prinsip RESTful untuk kejelasan dan prediktabilitas.

- **Organisasi Rute:** Rute akan diorganisir berdasarkan sumber daya, misalnya, src/app/api/projects/[projectId]/tasks/route.ts.
- **Metode HTTP:** Penggunaan metode HTTP (GET, POST, PUT, DELETE) akan secara ketat mewakili operasi CRUD (Create, Read, Update, Delete).
- **Pemisahan Tanggung Jawab:** Logika di dalam Route Handlers akan dibuat ramping. Tanggung jawab utamanya adalah: (1) pemeriksaan otentikasi dan otorisasi, (2) validasi input menggunakan Zod, dan (3) memanggil fungsi layanan dari modul fitur (misalnya, src/features/projects/services/create-task.ts) yang berisi logika bisnis inti. Pemisahan tanggung jawab ini sangat penting untuk kemudahan pengujian dan pemeliharaan.

Bagian 2: Arsitektur Database dan Desain Skema

Bagian ini merinci lapisan data, yang merupakan jantung dari AgencyFlow. Skema yang dirancang dengan baik sangat penting untuk mendukung fitur-fitur aplikasi, memastikan integritas data, dan memungkinkan skalabilitas di masa depan.

2.1. Pemilihan ORM: Rekomendasi Prisma untuk Kecepatan dan Keamanan Tipe

Berdasarkan analisis mendalam terhadap Prisma dan Drizzle, **Prisma** sangat direkomendasikan sebagai Object-Relational Mapper (ORM) untuk proyek ini.[7]

Justifikasi:

- **Kecepatan Pengembangan:** Tujuan utama AgencyFlow saat ini adalah membangun Minimum Viable Product (MVP) dan memvalidasi ide.[4] API Prisma yang intuitif dan tingkat abstraksi yang tinggi untuk kueri dan relasi secara drastis mempercepat pengembangan dibandingkan dengan sintaks Drizzle yang lebih mirip SQL.[7] Ini adalah faktor terpenting pada tahap awal.
- **Kurva Pembelajaran:** Prisma lebih mudah dipelajari oleh tim dengan tingkat keahlian SQL yang bervariasi. Skema deklaratif dan klien yang dihasilkan secara otomatis memberikan pengalaman pengembang yang lebih ramah.[7]
- **Ekosistem dan Alat Bantu:** Prisma memiliki ekosistem yang matang, termasuk Prisma Studio, sebuah antarmuka grafis untuk berinteraksi dengan database, yang sangat berharga untuk pengembangan dan debugging.[7]
- **Keamanan Tipe (Type Safety):** Meskipun keduanya aman secara tipe, klien yang dihasilkan secara otomatis oleh Prisma berdasarkan file schema.prisma menyediakan lingkungan yang sangat kuat dan mudah digunakan.

2.2. Skema PostgreSQL Komprehensif dalam Prisma DSL

Seluruh skema database akan didefinisikan dalam satu file otoritatif, prisma/schema.prisma. File ini akan berfungsi sebagai satu-satunya sumber kebenaran (single source of truth) untuk model data, dari mana migrasi dan klien yang aman secara tipe akan dihasilkan.[10]

Untuk memberikan gambaran umum yang mudah dicerna sebelum menyajikan skema lengkap, tabel berikut merangkum model data inti AgencyFlow.

Model	Deskripsi	Relasi Kunci
Organization	Entitas root untuk satu agensi (tenant). Semua data lain terikat pada sebuah organisasi.	Memiliki banyak User, Project, Client, Invoice.
User	Akun pengguna individu (anggota agensi atau	Milik Organization, memiliki satu Role.

	kontak klien).	
Role	Mendefinisikan izin pengguna (misalnya, ADMIN, MEMBER, CLIENT).	Memiliki banyak User.
Client	Pelanggan eksternal dari agensi.	Milik Organization, memiliki banyak Project, Invoice.
Project	Entitas pusat untuk mengelola pekerjaan, berisi tugas dan milestone.	Milik Client dan Organization, memiliki banyak Task, Invoice.
Task	Item pekerjaan individual dalam sebuah proyek. Dapat ditugaskan kepada pengguna.	Milik Project, dapat ditugaskan kepada User.
Quotation	Proposal atau penawaran harga pra-proyek yang dikirim ke klien.	Milik Client, dapat diubah menjadi Project.
Invoice	Dokumen keuangan untuk menagih klien berdasarkan pekerjaan proyek.	Milik Project dan Client.

Berikut adalah skema lengkap dalam Prisma DSL yang dirancang untuk mendukung fitur MVP AgencyFlow:

Code snippet

```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

generator client {
  provider = "prisma-client-js"
}
```

```
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

// ====================================
// Core Tenancy and User Models
// ====================================

model Organization {
  id        String    @id @default(cuid())
  name      String
  createdAt DateTime  @default(now())
  updatedAt DateTime  @updatedAt

  users     User
  clients   Client
  projects  Project
  invoices  Invoice
  quotations Quotation
  services  Service
}

model User {
  id           String    @id @default(cuid())
  email        String    @unique
  name         String?
  passwordHash  String
  createdAt     DateTime  @default(now())
  updatedAt     DateTime  @updatedAt

  organizationId String
  organization   Organization @relation(fields: [organizationId], references: [id], onDelete:
Cascade)

  roleId String
  role   Role   @relation(fields: [roleId], references: [id])

  assignedTasks Task
}

model Role {
```

```
  id          String   @id @default(cuid())
  name        UserRole @unique // ADMIN, PROJECT_MANAGER, MEMBER, CLIENT
  users       User
}

enum UserRole {
  ADMIN
  PROJECT_MANAGER
  MEMBER
  CLIENT
}

// ==================================
// Client and Project Management Models
// ==================================

model Client {
  id        String    @id @default(cuid())
  name      String
  email     String    @unique
  phone     String?
  address   String?
  createdAt DateTime  @default(now())
  updatedAt DateTime  @updatedAt

  organizationId String
  organization   Organization @relation(fields: [organizationId], references: [id], onDelete:
Cascade)

  projects   Project
  quotations Quotation
  invoices   Invoice
}

model Project {
  id          String        @id @default(cuid())
  name        String
  description String?
  status      ProjectStatus @default(PLANNING)
  startDate   DateTime?
  endDate     DateTime?
  createdAt   DateTime      @default(now())
  updatedAt   DateTime      @updatedAt
```

```
  organizationId String
  organization   Organization @relation(fields: [organizationId], references: [id], onDelete:
Cascade)

  clientId String
  client   Client @relation(fields: [clientId], references: [id])

  tasks    Task
  invoices Invoice
}

enum ProjectStatus {
  PLANNING
  ON_GOING
  ON_HOLD
  COMPLETED
  CANCELLED
}

model Task {
  id          String     @id @default(cuid())
  title       String
  description String?
  status      TaskStatus @default(TO_DO)
  priority    Priority   @default(MEDIUM)
  dueDate     DateTime?
  createdAt   DateTime   @default(now())
  updatedAt   DateTime   @updatedAt

  projectId String
  project   Project @relation(fields: [projectId], references: [id], onDelete: Cascade)

  assigneeId String?
  assignee   User?   @relation(fields: [assigneeId], references: [id])
}

enum TaskStatus {
  TO_DO
  IN_PROGRESS
  IN_REVIEW
  DONE
}
```

```
enum Priority {
  LOW
  MEDIUM
  HIGH
}

// ==================================
// Financial and Pre-Project Models
// ==================================

model Quotation {
  id          String         @id @default(cuid())
  status      QuotationStatus @default(DRAFT)
  totalAmount  Decimal
  issueDate    DateTime
  expiryDate   DateTime?
  createdAt    DateTime       @default(now())
  updatedAt    DateTime       @updatedAt

  organizationId String
  organization   Organization    @relation(fields: [organizationId], references: [id], onDelete:
Cascade)

  clientId String
  client   Client         @relation(fields: [clientId], references: [id])

  items    QuotationItem
}

model QuotationItem {
  id          String  @id @default(cuid())
  description String
  quantity    Int
  unitPrice   Decimal

  quotationId String
  quotation   Quotation @relation(fields: [quotationId], references: [id], onDelete: Cascade)

  // Link to a reusable service for easy project creation
  serviceId   String?
  service     Service? @relation(fields: [serviceId], references: [id])
}
```

```
enum QuotationStatus {
  DRAFT
  SENT
  VIEWED
  APPROVED
  REJECTED
}

model Invoice {
  id            String        @id @default(cuid())
  invoiceNumber String
  status        InvoiceStatus @default(DRAFT)
  totalAmount   Decimal
  issueDate     DateTime
  dueDate       DateTime
  paidDate      DateTime?
  createdAt     DateTime      @default(now())
  updatedAt     DateTime      @updatedAt

  organizationId String
  organization   Organization  @relation(fields: [organizationId], references: [id], onDelete:
Cascade)

  clientId String
  client   Client       @relation(fields: [clientId], references: [id])

  projectId String
  project   Project      @relation(fields: [projectId], references: [id])
}

enum InvoiceStatus {
  DRAFT
  SENT
  PAID
  OVERDUE
  CANCELLED
}

// ====================================
// Service Library Model (Bank Item & Jasa)
// ====================================
```

```
model Service {
  id          String  @id @default(cuid())
  name        String
  description  String?
  defaultPrice Decimal
  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt

  organizationId String
  organization   Organization @relation(fields: [organizationId], references: [id], onDelete: Cascade)

  // Used in many quotation items
  quotationItems QuotationItem
}
```

2.3. Pemodelan Relasi Kompleks dan Multi-Tenancy

- **Multi-Tenancy:** Arsitektur yang diimplementasikan adalah multi-tenant dengan satu database. Setiap tabel kunci (misalnya, Project, Client, Task) memiliki foreign key organizationId yang wajib diisi. Ini memastikan isolasi data yang ketat antara agensi yang berbeda yang menggunakan platform.[11]
- **Peran & Izin Pengguna:** Model User memiliki relasi ke model Role. Model Role berisi enum (ADMIN, PROJECT_MANAGER, MEMBER, CLIENT) yang akan menentukan hak akses di seluruh aplikasi. Ini adalah implementasi Role-Based Access Control (RBAC) yang sederhana namun kuat.

Salah satu proses bisnis paling kritis dalam AgencyFlow adalah konversi "Quotation-to-Project".[4] Implementasi naif yang menghubungkan

QuotationItem langsung ke Task akan rapuh. Solusi yang lebih dalam dan skalabel adalah dengan memperkenalkan model abstrak Service. Sebuah Service adalah templat yang dapat digunakan kembali untuk item pekerjaan (misalnya, "Desain UI/UX - 40 jam") yang ada di "Bank Item & Jasa".[4]

Dengan menciptakan model Service, sebuah sumber kebenaran tunggal ditetapkan. QuotationItem menjadi *turunan* dari Service dengan harga spesifik untuk sebuah proposal. Task menjadi *turunan* dari Service yang sama yang dijadwalkan untuk produksi. Abstraksi ini menyederhanakan "konversi cerdas" menjadi sebuah loop sederhana: untuk setiap QuotationItem, temukan Service induknya, dan buat Task baru menggunakan templat dari

Service tersebut. Ini membuat sistem lebih kuat, dapat dipelihara, dan secara langsung memungkinkan fitur "Bank Item & Jasa" yang diinginkan.

Bagian 3: Pengalaman Pengguna, Alur, dan Penyempurnaan Fitur

Bagian ini menerjemahkan visi produk dari dokumen ide proyek menjadi perjalanan pengguna yang nyata dan mengidentifikasi peluang untuk meningkatkan pengalaman pengguna, memastikan produk akhir tidak hanya fungsional tetapi juga intuitif dan efisien.

3.1. Diagram Alur Pengguna Inti

Visualisasi alur pengguna yang paling krusial akan memastikan pemahaman bersama tentang alur aplikasi sebelum kode ditulis. Diagram ini akan dibuat menggunakan Mermaid.js untuk kemudahan integrasi ke dalam dokumentasi.

1. Alur Onboarding Klien & Inisiasi Proyek:

Code snippet

```
graph TD
    A --> B{Kirim ke Klien};
    B --> C[Klien Menerima Notifikasi & Melihat di Portal];
    C --> D{Klien Menyetujui Quotation};
    D --> E;
```

2. Alur Manajemen Tugas Tim Internal:

Code snippet

```
graph TD
    A --> B;
    B --> C;
    C --> D;
    D --> E;
```

3. Alur Umpan Balik & Persetujuan Klien:

```
Code snippet
```

```
graph TD
    A --> B[Klien Menerima Notifikasi];
    B --> C[Klien Melihat Mockup di Portal Klien];
    C --> D{Memberikan Komentar Umpan Balik};
    D --> E;
```

3.2. Rekomendasi Peningkatan UX

Di luar fitur dasar, beberapa peningkatan UX berdampak tinggi dapat diimplementasikan untuk menciptakan produk yang lebih "lengket" dan profesional.

- **Notifikasi Real-Time:** Menggunakan layanan seperti WebSockets atau solusi serverless (misalnya, Pusher, Ably) untuk memberikan notifikasi instan untuk peristiwa penting (komentar klien baru, penugasan tugas, pembayaran invoice). Ini meningkatkan nuansa kolaboratif platform.
- **Pembaruan UI Optimis:** Ketika pengguna melakukan tindakan seperti memindahkan tugas di papan Kanban, perbarui UI secara instan *sebelum* panggilan API selesai. Kembalikan ke kondisi semula hanya jika panggilan API gagal. Ini membuat aplikasi terasa jauh lebih cepat dan responsif.
- **Palet Perintah (Command Palette):** Implementasikan palet perintah global (dapat diakses melalui Cmd+K atau Ctrl+K) yang memungkinkan pengguna mahir untuk menavigasi, membuat proyek/tugas baru, atau mencari klien dengan cepat tanpa menggunakan mouse. Ini adalah fitur umum dalam alat produktivitas modern dan sangat meningkatkan efisiensi.

3.3. Penambahan Fitur Strategis untuk Pertumbuhan Pasca-MVP

Sejalan dengan visi jangka panjang, langkah-langkah pengembangan berikutnya setelah peluncuran MVP yang sukses dapat diidentifikasi.[4]

- **Integrasi Pihak Ketiga (Git, Slack):** Bagi agensi perangkat lunak, menghubungkan tugas proyek ke commit/branch Git adalah peningkatan alur kerja yang masif. Integrasi Slack untuk notifikasi adalah harapan standar untuk alat kolaboratif.
- **Pelaporan Keuangan Dasar:** Memanfaatkan data Invoice dan Project yang ada untuk menghasilkan laporan sederhana: "Pendapatan per Klien," "Pendapatan Berulang Bulanan (untuk retainer)," dan "Profitabilitas Proyek" (dengan membandingkan jumlah yang ditagih dengan waktu yang dilacak). Ini memberikan nilai bisnis langsung kepada pemilik agensi.
- **Integrasi Payment Gateway:** Seperti yang disebutkan dalam dokumen ide proyek, berintegrasi dengan Stripe, Midtrans, atau Xendit untuk memungkinkan klien membayar invoice langsung melalui portal klien adalah langkah penting menuju otomatisasi seluruh siklus hidup agensi.[4]

Bagian 4: Integrasi, Optimasi, dan Praktik Terbaik Keamanan

Bagian akhir ini memberikan panduan praktis dan langsung untuk membangun AgencyFlow, dengan fokus pada menghubungkan komponen, memastikan kinerja, dan menerapkan langkah-langkah keamanan yang kuat sejak hari pertama.

4.1. Menghubungkan Next.js ke PostgreSQL dengan Prisma

Koneksi database yang aman dan efisien sangat penting untuk aplikasi produksi.

- **Variabel Lingkungan:** String koneksi database akan disimpan dengan aman di file .env.local, yang tidak pernah dimasukkan ke dalam sistem kontrol versi.[12]
- **Prisma Client Singleton:** Sebuah instance singleton dari Prisma Client akan dibuat untuk menghindari kehabisan batas koneksi database di lingkungan serverless seperti Vercel. Ini adalah praktik terbaik yang kritis untuk aplikasi Next.js produksi dan melibatkan pembuatan file utilitas yang menginstansiasi atau mengembalikan instance Prisma Client

global yang ada.[14]

Berikut adalah cuplikan kode untuk pola singleton Prisma Client di src/shared/lib/db.ts:

TypeScript

```
import { PrismaClient } from '@prisma/client';

declare global {
  // allow global `var` declarations
  // eslint-disable-next-line no-var
  var prisma: PrismaClient | undefined;
}

export const prisma =
  global.prisma ||
  new PrismaClient({
    log: ['query'],
  });

if (process.env.NODE_ENV!== 'production') global.prisma = prisma;
```

4.2. Menerapkan Role-Based Access Control (RBAC) dengan Auth.js (NextAuth.js)

Auth.js akan digunakan untuk menangani otentikasi dan mengimplementasikan logika RBAC untuk melindungi halaman dan endpoint API.

1. **Konfigurasi:** Handler Auth.js akan diatur di src/app/api/auth/[...nextauth]/route.ts menggunakan Credentials provider untuk login email/password.
2. **Memperluas Sesi:** Objek callbacks dalam konfigurasi Auth.js akan digunakan. Callback jwt akan mengambil peran pengguna dari database saat login dan menambahkannya ke token JWT. Callback session kemudian akan mengekspos peran ini dari token ke objek sesi, membuatnya tersedia di seluruh aplikasi.[15]
3. **Melindungi Rute dengan Middleware:** File src/middleware.ts akan dibuat untuk memeriksa sesi yang valid pada semua rute di dalam grup (app). Middleware ini juga dapat melakukan pemeriksaan peran dasar, mengalihkan pengguna yang mencoba

mengakses area yang tidak sah.[15]

4. **Melindungi Endpoint API & Server Actions:** Di dalam setiap Route Handler atau Server Action, sesi saat ini akan diambil untuk melakukan pemeriksaan terperinci guna memastikan pengguna memiliki peran yang diperlukan sebelum mengeksekusi logika apa pun. Pemeriksaan sisi server ini adalah sumber kebenaran utama untuk otorisasi.[16]

Berikut adalah contoh cuplikan kode untuk konfigurasi callbacks di Auth.js:

TypeScript

```
// di dalam file konfigurasi Auth.js
//...
callbacks: {
  async jwt({ token, user }) {
    if (user) {
      // Saat login, 'user' object tersedia. Ambil peran dari database.
      // Asumsikan 'user' object dari authorize function sudah berisi 'role'.
      token.role = user.role;
    }
    return token;
  },
  async session({ session, token }) {
    if (session.user) {
      // Tambahkan peran dari token ke objek sesi
      session.user.role = token.role as string;
    }
    return session;
  },
},
//...
```

4.3. Strategi Optimasi Kinerja dan Caching

AgencyFlow adalah aplikasi yang sangat dinamis, sehingga strategi caching yang naif akan menyebabkan data basi. Revalidasi on-demand Next.js harus dimanfaatkan untuk memastikan konsistensi data.

- **Penandaan Cache Data (Cache Tagging):** Semua panggilan fetch dan kueri data sisi server (yang dibungkus dalam fungsi cache React) akan ditandai dengan tag cache yang relevan. Misalnya, mengambil tugas untuk sebuah proyek akan ditandai dengan ['tasks', 'project-123-tasks'].[5]
- **Revalidasi On-Demand dengan revalidateTag:** Server Actions akan dibuat untuk melakukan mutasi data (misalnya, membuat tugas, memperbarui proyek). Setelah mutasi database berhasil, action tersebut akan memanggil revalidateTag('tasks') atau revalidateTag('project-123-tasks'). Ini akan membersihkan data yang relevan dari Data Cache Next.js di seluruh aplikasi.[17]

Sebagai contoh praktis, ketika seorang anggota tim memindahkan tugas dari "In Progress" ke "Done" melalui Server Action, action tersebut akan (1) memperbarui status tugas di database dan (2) memanggil revalidateTag('tasks'). Ini memastikan bahwa dasbor manajer proyek, portal klien, dan bagian lain dari aplikasi yang menampilkan data tugas tersebut akan di-render ulang dengan informasi terbaru pada permintaan berikutnya.

4.4. Keamanan API dan Validasi Input dengan Zod

Prinsip utamanya adalah jangan pernah mempercayai input pengguna. Semua data yang masuk ke sistem melalui rute API atau Server Actions harus divalidasi secara ketat untuk memastikan integritas data dan mencegah kerentanan keamanan.

Implementasi dengan Zod:

1. **Definisikan Skema:** Untuk setiap endpoint API, skema Zod akan didefinisikan untuk menentukan bentuk, tipe, dan batasan data yang diharapkan.[19]
2. **Validasi di Awal:** Di awal setiap Route Handler atau Server Action, metode .safeParse() dari skema akan digunakan untuk memvalidasi data yang masuk.
3. **Tolak Permintaan Tidak Valid:** Jika validasi gagal, respons 400 Bad Request akan segera dikembalikan dengan pesan kesalahan terperinci yang berasal dari objek kesalahan Zod. Ini mencegah data tidak valid mencapai logika bisnis atau lapisan database.[20]

Berikut adalah contoh lengkap Route Handler untuk membuat proyek baru, termasuk skema Zod dan logika validasi:

TypeScript

```
// src/app/api/projects/route.ts

import { NextResponse } from 'next/server';
import { z } from 'zod';
import { prisma } from '@/shared/lib/db';
import { auth } from '@/auth'; // Asumsi auth.ts ada di root

const createProjectSchema = z.object({
  name: z.string().min(3, "Nama proyek harus memiliki minimal 3 karakter"),
  clientId: z.string().cuid("ID Klien tidak valid"),
  // tambahkan validasi lain sesuai kebutuhan
});

export async function POST(request: Request) {
  const session = await auth();

  // 1. Pemeriksaan Otorisasi
  if (!session |

| (session.user.role!== 'ADMIN' && session.user.role!== 'PROJECT_MANAGER')) {
    return NextResponse.json({ error: 'Tidak diizinkan' }, { status: 403 });
  }

  const json = await request.json();

  // 2. Validasi Input dengan Zod
  const validationResult = createProjectSchema.safeParse(json);

  if (!validationResult.success) {
    return NextResponse.json({ error: validationResult.error.flatten().fieldErrors }, { status: 400 });
  }

  const { name, clientId } = validationResult.data;
  const organizationId = session.user.organizationId; // Asumsi ID organisasi ada di sesi

  try {
    // 3. Logika Bisnis
    const newProject = await prisma.project.create({
      data: {
        name,
        clientId,
        organizationId,
```

```
    },
  });
  return NextResponse.json(newProject, { status: 201 });
 } catch (error) {
  return NextResponse.json({ error: 'Gagal membuat proyek' }, { status: 500 });
 }
}
```

Kesimpulan: Peta Jalan Bertahap Menuju Peluncuran

Cetak biru ini menguraikan arsitektur yang kuat dan modern untuk AgencyFlow, dengan memanfaatkan kekuatan Next.js dan PostgreSQL. Keputusan-keputusan kunci—seperti struktur berbasis domain, pemilihan Prisma sebagai ORM, implementasi RBAC dengan Auth.js, validasi input ketat dengan Zod, dan strategi caching dinamis dengan revalidasi on-demand—dirancang untuk mendukung pengembangan yang cepat, skalabilitas jangka panjang, dan keamanan yang solid.

Untuk mengubah rencana arsitektur ini menjadi strategi pengembangan yang dapat ditindaklanjuti, peta jalan bertahap berikut direkomendasikan:

1. **Fase 1: Fondasi dan Otentikasi:** Siapkan struktur proyek, konfigurasikan Prisma dan database, dan implementasikan sistem otentikasi dan otorisasi (RBAC) lengkap menggunakan Auth.js.
2. **Fase 2: Fitur Manajemen Inti:** Bangun modul manajemen proyek dan tugas, termasuk papan Kanban interaktif dan detail tugas yang komprehensif untuk penggunaan internal.
3. **Fase 3: Portal Klien dan Kolaborasi:** Kembangkan portal klien, yang memungkinkan klien untuk melihat kemajuan proyek, memberikan umpan balik, dan mengakses dokumen.
4. **Fase 4: Alur Kerja Keuangan:** Implementasikan modul penagihan (invoicing) dan alur kerja krusial dari konversi penawaran (quotation) menjadi proyek.
5. **Fase 5: Peluncuran MVP dan Iterasi:** Luncurkan Minimum Viable Product (MVP) ke pengguna awal (misalnya, internal agensi) untuk mengumpulkan umpan balik dan mulai melakukan iterasi berdasarkan penggunaan nyata.

Dengan mengikuti cetak biru dan peta jalan ini, proyek AgencyFlow berada di jalur yang tepat untuk menjadi platform yang efisien, andal, dan berharga bagi agensi kreatif dan perangkat lunak.

Works cited

1. How do you structure your project files when using App Router? : r/nextjs - Reddit, accessed September 4, 2025, https://www.reddit.com/r/nextjs/comments/1jfr4q0/how_do_you_structure_your_project_files_when/
2. The Battle-Tested NextJS Project Structure I Use in 2025. | by ..., accessed September 4, 2025, https://medium.com/@burpdeepak96/the-battle-tested-nextjs-project-structure-i-use-in-2025-f84c4eb5f426
3. Sharing my go-to project structure for Next.js - colocation-first approach : r/nextjs - Reddit, accessed September 4, 2025, https://www.reddit.com/r/nextjs/comments/1kkpqtm/sharing_my_goto_project_structure_for_nextjs/
4. Ide Projek.pdf
5. Data Fetching, Caching, and Revalidating - Next.js, accessed September 4, 2025, https://nextjs.org/docs/14/app/building-your-application/data-fetching/fetching-caching-and-revalidating
6. Getting Started: Fetching Data - Next.js, accessed September 4, 2025, https://nextjs.org/docs/app/getting-started/fetching-data
7. Prisma vs. Drizzle: A Comprehensive Guide for Your NextJS Project ..., accessed September 4, 2025, https://dev.to/fabrikapp/prisma-vs-drizzle-a-comprehensive-guide-for-your-nextjs-project-1lfd
8. Best DB ORM for production : r/nextjs - Reddit, accessed September 4, 2025, https://www.reddit.com/r/nextjs/comments/1k9prpi/best_db_orm_for_production/
9. Prisma | Instant Postgres plus an ORM for simpler db workflows, accessed September 4, 2025, https://www.prisma.io/
10. Prisma Schema Overview | Prisma Documentation, accessed September 4, 2025, https://www.prisma.io/docs/orm/prisma-schema/overview
11. Multi Tenant SaaS Database Design With Prisma Using One Database - Stack Overflow, accessed September 4, 2025, https://stackoverflow.com/questions/79382275/multi-tenant-saas-database-design-with-prisma-using-one-database
12. How To Connect Next.js With Postgres SQL? - Saffron Tech, accessed September 4, 2025, https://www.saffrontech.net/blog/how-to-connect-nextjs-with-postgres-sql
13. App Router: Setting Up Your Database - Next.js, accessed September 4, 2025, https://nextjs.org/learn/dashboard-app/setting-up-your-database
14. Best Practices for Structuring a Full-Stack Next.js Project? · community · Discussion #151820, accessed September 4, 2025, https://github.com/orgs/community/discussions/151820
15. NextAuth.js Basics: Complete Role-Based Authentication Setup | by Robert Gouveia, accessed September 4, 2025, https://medium.com/@robertgeedev/nextauth-js-complete-role-based-authentication-setup-38774cc6f0c4
16. Role Based Access Control - Auth.js, accessed September 4, 2025,

https://authjs.dev/guides/role-based-access-control
17. Getting Started: Caching and Revalidating - Next.js, accessed September 4, 2025, https://nextjs.org/docs/app/getting-started/caching-and-revalidating
18. Guides: Caching | Next.js, accessed September 4, 2025, https://nextjs.org/docs/app/guides/caching
19. API Routes Validation - Makerkit, accessed September 4, 2025, https://makerkit.dev/docs/next-supabase/tutorials/api-routes-validation
20. How to validate Next.js API routes using Zod, accessed September 4, 2025, https://kirandev.com/nextjs-api-routes-zod-validation