

ArchitectPro - Backend Architecture Documentation

Technology Stack

Core Framework

- **Next.js 14.2.28** - React framework with App Router
- **React 18.2.0** - UI library
- **TypeScript 5.2.2** - Type safety

Database & ORM

- **PostgreSQL** - Primary database
- **Prisma 6.7.0** - ORM (Object-Relational Mapping)
- **@prisma/client 6.7.0** - Type-safe database client

API Architecture

- **Next.js API Routes** - Serverless functions
 - **RESTful endpoints** - `/api/templates`, `/api/templates/[bhkType]`
-

Database Schema

Models Overview

1. Template Model (Primary)

Stores pre-designed floor plan templates.

```
model Template {
  id          String    @id @default(cuid())
  name        String
  bhkType     String    // "Studio", "1BHK", "2BHK", "3BHK", "4BHK", "5BHK+"
  propertyType String   @default("Apartment") // "Apartment", "Condo", "Villa", etc.
  description  String?
  roomsData   Json      // Array of rooms with dimensions, positions, colors
  doorsData   Json?     // Array of doors with swing directions
  windowsData Json?     // Array of windows with types
  fixturesData Json?    // Bathroom/kitchen fixtures
  furnitureData Json?   // Furniture items
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  @@index([bhkType])
  @@index([propertyType])
}
```

Fields:

- `id` : Unique identifier (CUID)
- `bhkType` : Layout configuration (Studio, 1BHK, 2BHK, 3BHK, 4BHK, 5BHK+)

- `propertyType` : Building classification (Apartment, Condo, Villa, Townhouse, Duplex, Penthouse, Bungalow)
- `roomsData` : JSON array containing room specifications
- `doorsData` : JSON array with door positions and types
- `windowsData` : JSON array with window configurations
- `fixturesData` : JSON array with bathroom/kitchen fixtures
- `furnitureData` : JSON array with furniture placements

Indexes:

- `bhkType` - Fast filtering by layout type
- `propertyType` - Fast filtering by property classification

2. Design Model

Stores user-created or modified designs.

```
model Design {
  id          String    @id @default(cuid())
  name        String
  bhkType     String
  templateId  String?
  roomsData   Json      // Current room configuration
  specifications Json    // Wall thickness, dimensions, etc.
  metadata    Json?     // Additional info
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  @@index([bhkType])
}
```

3. Configuration Model

Stores design specifications and settings.

```
model Configuration {
  id          String    @id @default(cuid())
  designId    String    @unique
  overallWidth Float     // in meters or feet
  overallHeight Float    // in meters or feet
  wallThickness Float    // in cm or inches
  ceilingHeight Float    // in meters or feet
  unit        String    @default("meters")
  drainageSpecs Json?    // Drainage system specifications
  sewageSpecs  Json?    // Sewage system specifications
  materialSpecs Json?    // Material specifications
  createdAt    DateTime  @default(now())
  updatedAt    DateTime  @updatedAt
}
```

4. UserPreference Model

Stores user preferences and defaults.

```
model UserPreference {
  id          String @id @default(cuid())
  userId      String? @unique
  defaultUnit String @default("meters")
  defaultBHK  String @default("2BHK")
  preferences Json?   // Additional preferences
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
}
```

API Endpoints

GET /api/templates

Description: Fetch templates with optional filtering.

Query Parameters:

- `bhkType` (optional): Filter by layout type (Studio, 1BHK, 2BHK, 3BHK, 4BHK, 5BHK+)
- `propertyType` (optional): Filter by property type (Apartment, Condo, Villa, etc.)

Response:

```
[
  {
    "id": "clxxxx",
    "name": "2 BHK West Facing",
    "bhkType": "2BHK",
    "propertyType": "Apartment",
    "description": "Modern 2 bedroom apartment",
    "roomsData": [...],
    "doorsData": [...],
    "windowsData": [...],
    "fixturesData": [...],
    "furnitureData": [...],
    "createdAt": "2024-01-01T00:00:00.000Z",
    "updatedAt": "2024-01-01T00:00:00.000Z"
  }
]
```

Error Response:

```
{
  "error": "Failed to fetch templates"
}
```

Status Codes:

- 200 : Success
- 500 : Server error

Data Structures

Room Object

```
{
  id: string;
  name: string;           // "Living Room", "Kitchen", etc.
  type: string;           // "living", "bedroom", "kitchen", "bathroom"
  x: number;              // X position in meters
  y: number;              // Y position in meters
  width: number;          // Width in meters
  height: number;         // Height in meters
  color: string;          // Hex color code
  floor: number;          // Floor level (0 = ground)
}
```

Door Object

```
{
  id: string;
  x: number;              // X position
  y: number;              // Y position
  width: number;          // Door width (typically 0.8-0.9m)
  height: number;         // Door height
  rotation: number;       // Rotation angle (0, 90, 180, 270)
  type: string;           // "single", "double", "sliding", "bifold"
  swingDirection: string; // "inward", "outward"
}
```

Window Object

```
{
  id: string;
  x: number;
  y: number;
  width: number;          // Window width
  height: number;         // Window height
  rotation: number;       // Rotation angle
  type: string;           // "fixed", "sliding", "casement", "bay"
}
```

Fixture Object

```
{
  id: string;
  x: number;
  y: number;
  width: number;
  height: number;
  rotation: number;
  type: string;           // "toilet", "sink", "shower", "bathtub", "kitchen-sink"
}
```

Furniture Object

```
{
  id: string;
  x: number;
  y: number;
  width: number;
  height: number;
  rotation: number;
  type: string;           // "bed", "sofa", "dining-table", "desk", "chair"
  label?: string;         // "Queen", "King", "Single" for beds
}
```



Data Flow

Template Loading

1. User selects BHK type and property type
2. Frontend calls `/api/templates?bhkType=2BHK&propertyType=Apartment`
3. API queries database with Prisma
4. Returns matching templates
5. Frontend renders floor plan with scaling

Dynamic Scaling

1. User adjusts dimensions in controls panel
2. Frontend calculates scale factors (scaleX, scaleY)
3. All elements (rooms, doors, windows, fixtures, furniture) scale proportionally
4. SVG re-renders with smooth CSS transitions



Environment Variables

Required environment variables:

```
DATABASE_URL="postgresql://user:password@host:5432/database"
```



Seeding

The database is seeded with professional templates using:

```
npm run prisma db seed
# or
yarn prisma db seed
```

Seed Script: `/scripts/seed_new.ts`

Templates Seeded:

- 2 BHK West Facing (Apartment) - 11m × 10m
- 3 BHK Modern Layout (Apartment) - 14m × 11m



Database Operations

Migrate Database

```
npx prisma migrate dev --name migration_name
```

Push Schema Changes

```
npx prisma db push
```

Generate Prisma Client

```
npx prisma generate
```

Open Prisma Studio (Database GUI)

```
npx prisma studio
```



Architecture Principles

1. Separation of Concerns

- Database logic in Prisma models
- Business logic in API routes
- UI logic in React components

2. Type Safety

- TypeScript for all code
- Prisma generates type-safe client
- Zod for runtime validation (if needed)

3. Performance

- Database indexes on frequently queried fields
- JSON fields for flexible nested data
- Server-side rendering for SEO

4. Scalability

- Serverless API routes
 - PostgreSQL for production reliability
 - JSON fields for schema flexibility
-

Dependencies

Core

- `@prisma/client` : 6.7.0
- `prisma` : 6.7.0
- `next` : 14.2.28
- `react` : 18.2.0
- `typescript` : 5.2.2

UI

- `@radix-ui/*` : Various UI components
- `lucide-react` : Icons
- `tailwindcss` : Styling

Utilities

- `date-fns` : Date formatting
- `clsx` : Conditional classes
- `zod` : Schema validation

Query Examples

Fetch All Templates

```
const templates = await prisma.template.findMany({
  orderBy: { createdAt: 'desc' }
});
```

Fetch by BHK Type

```
const templates = await prisma.template.findMany({
  where: { bhkType: '2BHK' },
  orderBy: { createdAt: 'desc' }
});
```

Fetch by Property Type and BHK

```
const templates = await prisma.template.findMany({
  where: {
    bhkType: '2BHK',
    propertyType: 'Villa'
  },
  orderBy: { createdAt: 'desc' }
});
```

Create New Template

```
const template = await prisma.template.create({
  data: {
    name: '4 BHK Penthouse',
    bhkType: '4BHK',
    propertyType: 'Penthouse',
    description: 'Luxury penthouse with terrace',
    roomsData: [...],
    doorsData: [...],
    windowsData: [...],
    fixturesData: [...],
    furnitureData: [...]
  }
});
```



Performance Considerations

1. Database Indexes

- `bhkType` index for fast filtering
- `propertyType` index for classification queries

2. JSON Fields

- Flexible for nested data
- Avoid complex queries on JSON content
- Use for read-heavy operations

3. API Response Size

- Templates include all data in single response
- Consider pagination for large datasets

4. Connection Pooling

- Prisma handles connection pooling automatically
- Configure in `schema.prisma` if needed



Troubleshooting

Prisma Client Not Found

```
npx prisma generate
```

Database Connection Issues

- Check `DATABASE_URL` in `.env`
- Verify PostgreSQL is running
- Check firewall/network settings

Migration Errors

```
npx prisma migrate reset # WARNING: Deletes all data
npx prisma db push      # Force schema sync
```

Seed Errors

```
npx prisma db seed --preview-feature
```



Additional Resources

- [Prisma Documentation](https://www.prisma.io/docs) (https://www.prisma.io/docs)
- [Next.js API Routes](https://nextjs.org/docs/api-routes/introduction) (https://nextjs.org/docs/api-routes/introduction)
- [PostgreSQL Documentation](https://www.postgresql.org/docs/) (https://www.postgresql.org/docs/)