-

1. Node.js

```bash
#                Node.js (       LTS       )
#          https://nodejs.org/
#
node --version
npm --version
```

2. PostgreSQL

```bash
# Windows:
# https://www.postgresql.org/download/

# macOS:        Homebrew
brew install postgresql

# Ubuntu/Debian
sudo apt-get install postgresql postgresql-contrib

#
# Windows:
# macOS: brew services start postgresql
# Linux: sudo systemctl start postgresql
```

3.

Visual Studio Code

- ES7+ React/Redux/React-Native snippets
- Prettier - Code formatter
- ESLint
- Prisma

1.

```bash
mkdir system-dev-management
cd system-dev-management
```

2.

```bash
#
mkdir backend
cd backend

#          Node.js
npm init -y

#
npm install express cors helmet morgan dotenv
npm install prisma @prisma/client
npm install bcryptjs jsonwebtoken
npm install --save-dev nodemon @types/node

#
mkdir src
mkdir src/controllers
mkdir src/models
mkdir src/routes
mkdir src/middleware
mkdir src/utils
mkdir prisma
```

3.

```bash
```

```
#
cd ..

#       React
npx create-react-app frontend
cd frontend

#
npm install axios react-router-dom
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

4. Tailwind CSS

javascript

```
// frontend/tailwind.config.js
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    "./src/**/*.{js,jsx,ts,tsx}",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

css

```
/* frontend/src/index.css */
@tailwind base;
@tailwind components;
@tailwind utilities;
```

1.

bash

```
# backend/.env
DATABASE_URL="postgresql://username:password@localhost:5432/system_dev_management"
JWT_SECRET="your-secret-key-here"
PORT=5000
NODE_ENV=development
```

2. Package.json

json

```
{
  "name": "system-dev-management-backend",
  "version": "1.0.0",
  "scripts": {
    "start": "node src/server.js",
    "dev": "nodemon src/server.js",
    "db:generate": "prisma generate",
    "db:push": "prisma db push",
    "db:studio": "prisma studio"
  }
}
```

3.

javascript

```
// backend/src/server.js
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
const morgan = require('morgan');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 5000;

//
app.use(helmet());
app.use(cors());
app.use(morgan('combined'));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

//
app.get('/api/health', (req, res) => {
  res.json({
    status: 'OK',
    message: 'Server is running',
    timestamp: new Date().toISOString()
  });
});

//
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({
    error: 'Something went wrong!',
    message: process.env.NODE_ENV === 'development' ? err.message : 'Internal Server Error'
  });
});

// 404
app.use('*', (req, res) => {
  res.status(404).json({ error: 'Route not found' });
});

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

```
  console.log(`Health check: http://localhost:${PORT}/api/health`);
});
```

1. Prisma Schema

prisma

```
// backend/prisma/schema.prisma
generator client {
  provider = "prisma-client-js"
}

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

model User {
  userId    Int      @id @default(autoincrement()) @map("user_id")
  username  String   @unique @db.VarChar(50)
  email     String   @unique @db.VarChar(100)
  passwordHash String @map("password_hash") @db.VarChar(255)
  createdAt DateTime @default(now()) @map("created_at")
  updatedAt DateTime @updatedAt @map("updated_at")

  //
  createdDictionaries DataDictionary[] @relation("DictionaryCreator")
  updatedDictionaries DataDictionary[] @relation("DictionaryUpdater")
  createdKnowledge    KnowledgeBase[]  @relation("KnowledgeCreator")
  updatedKnowledge    KnowledgeBase[]  @relation("KnowledgeUpdater")
  createdSystems      ApiSystem[]     @relation("SystemCreator")
  updatedSystems      ApiSystem[]     @relation("SystemUpdater")
  createdCategories   FunctionCategory[] @relation("CategoryCreator")
  updatedCategories   FunctionCategory[] @relation("CategoryUpdater")
  createdComponents   ApiComponent[]   @relation("ComponentCreator")
  updatedComponents   ApiComponent[]   @relation("ComponentUpdater")

  @@map("users")
}

model DataDictionary {
  dictId       Int      @id @default(autoincrement()) @map("dict_id")
  name         String   @db.VarChar(100)
  abbreviation String?  @db.VarChar(20)
  fullName     String   @map("full_name") @db.VarChar(200)
  dataType     String?  @map("data_type") @db.VarChar(50)
  description  String?  @db.Text
  isActive     Boolean  @default(true) @map("is_active")
  version      String   @default("1.0") @db.VarChar(10)
  remarks      String?  @db.Text
```

```
  createdBy    Int     @map("created_by")
  updatedBy    Int?    @map("updated_by")
  createdAt    DateTime @default(now()) @map("created_at")
  updatedAt    DateTime @updatedAt @map("updated_at")

  //
  creator User  @relation("DictionaryCreator", fields: [createdBy], references: [userId])
  updater User? @relation("DictionaryUpdater", fields: [updatedBy], references: [userId])

  @@map("data_dictionary")
}

model KnowledgeBase {
  kbId       Int     @id @default(autoincrement()) @map("kb_id")
  category   String  @db.VarChar(50)
  code       String  @unique @db.VarChar(20)
  title      String  @db.VarChar(200)
  description String  @db.Text
  solution   String? @db.Text
  keywords   String? @db.VarChar(500)
  attachments String? @db.Text
  priority   String  @default("    ") @db.VarChar(10)
  status     String  @default("       ") @db.VarChar(20)
  createdBy  Int     @map("created_by")
  updatedBy  Int?    @map("updated_by")
  createdAt  DateTime @default(now()) @map("created_at")
  updatedAt  DateTime @updatedAt @map("updated_at")

  //
  creator User  @relation("KnowledgeCreator", fields: [createdBy], references: [userId])
  updater User? @relation("KnowledgeUpdater", fields: [updatedBy], references: [userId])

  @@map("knowledge_base")
}

model ApiSystem {
  systemId   Int     @id @default(autoincrement()) @map("system_id")
  systemCode  String  @unique @map("system_code") @db.VarChar(20)
  systemName  String  @map("system_name") @db.VarChar(100)
  systemType  String? @map("system_type") @db.VarChar(100)
  description String? @db.Text
  version    String? @db.VarChar(20)
  status     String  @default("          ") @db.VarChar(20)
  createdBy  Int     @map("created_by")
```

```
  updatedBy   Int?    @map("updated_by")
  createdAt   DateTime @default(now()) @map("created_at")
  updatedAt   DateTime @updatedAt @map("updated_at")

  //
  creator       User            @relation("SystemCreator", fields: [createdBy], references: [userId])
  updater       User?           @relation("SystemUpdater", fields: [updatedBy], references: [userId])
  categories    FunctionCategory[]
  apiComponents   ApiComponent[]

  @@map("api_systems")
}

model FunctionCategory {
  categoryId   Int     @id @default(autoincrement()) @map("category_id")
  systemId     Int     @map("system_id")
  categoryCode String   @map("category_code") @db.VarChar(50)
  categoryName String   @map("category_name") @db.VarChar(100)
  description  String?  @db.Text
  isActive     Boolean  @default(true) @map("is_active")
  createdBy    Int     @map("created_by")
  updatedBy    Int?    @map("updated_by")
  createdAt    DateTime @default(now()) @map("created_at")
  updatedAt    DateTime @updatedAt @map("updated_at")

  //
  system       ApiSystem     @relation(fields: [systemId], references: [systemId])
  creator      User          @relation("CategoryCreator", fields: [createdBy], references: [userId])
  updater      User?         @relation("CategoryUpdater", fields: [updatedBy], references: [userId])
  apiComponents ApiComponent[]

  @@map("function_categories")
}

model ApiComponent {
  apiId           Int     @id @default(autoincrement()) @map("api_id")
  systemId        Int     @map("system_id")
  categoryId      Int     @map("category_id")
  name            String   @db.VarChar(100)
  version         String   @db.VarChar(20)
  description     String?  @db.Text
  developer       String?  @db.VarChar(100)
  endpointPath    String?  @map("endpoint_path") @db.VarChar(500)
  parameters      String?  @db.Text
```

```
  returnDescription String?  @map("return_description") @db.Text
  dependencies     String?  @db.Text
  isActive         Boolean  @default(true) @map("is_active")
  createdBy        Int      @map("created_by")
  updatedBy        Int?     @map("updated_by")
  createdAt        DateTime @default(now()) @map("created_at")
  updatedAt        DateTime @updatedAt @map("updated_at")

  //
  system   ApiSystem       @relation(fields: [systemId], references: [systemId])
  category FunctionCategory @relation(fields: [categoryId], references: [categoryId])
  creator  User           @relation("ComponentCreator", fields: [createdBy], references: [userId])
  updater  User?          @relation("ComponentUpdater", fields: [updatedBy], references: [userId])

  @@map("api_components")
}
```

2.

```bash
#       Prisma
npx prisma generate

#       schema
npx prisma db push

#
npx prisma studio
```

1.

```bash
cd backend
npm run dev
```

http://localhost:5000/api/health

2.

```bash
cd frontend
npm start
```

http://localhost:3000

1. API
2.
3.
4.
5.