Answering this prompt to build a complex Backend-as-a-Service (BaaS) from scratch is a significant undertaking. A full, production-ready implementation of a PocketBase-like application would require thousands of lines of code and extensive security considerations, which is beyond the scope of a single response.
I can, however, provide a solid, scalable foundation for this project. This includes a modular folder structure, a complete Hono/TypeScript/PostgreSQL setup with Drizzle ORM, user authentication (with JWT), and a basic CRUD implementation for a 'notes' collection. This gives you the core building blocks and best practices needed to expand the project with the other features you requested (OAuth, WebSockets, file storage, etc.).

Overview

A scalable Backend-as-a-Service (BaaS) foundation built with Hono, TypeScript, and PostgreSQL. It includes user authentication and a CRUD API for managing data records.

Folder Structure

```
/hono-baas-backend
├── .dockerignore
├── .env
├── .env.example
├── .eslintrc.cjs
├── .gitignore
├── .prettierrc
├── Dockerfile
├── README.md
├── package.json
├── tsconfig.json
├── drizzle.config.ts
└── src
    ├── index.ts
    ├── routes
    │   ├── auth.route.ts
    │   ├── notes.route.ts
    │   └── index.ts
    ├── controllers
    │   ├── auth.controller.ts
    │   └── notes.controller.ts
    ├── services
    │   ├── auth.service.ts
    │   └── notes.service.ts
    ├── db
    │   ├── index.ts
    │   ├── schema.ts
    │   └── migrate.ts
    ├── middleware
    │   └── auth.middleware.ts
    └── utils
```

```
        ├── jwt.ts
        └── logger.ts
```

Install Commands

```
# Initialize a new npm project and install dependencies
mkdir hono-baas-backend
cd hono-baas-backend
npm init -y
npm install hono @hono/zod-validator @neondatabase/serverless
drizzle-orm zod bcryptjs jsonwebtoken dotenv
npm install -D typescript tsx @types/node @types/bcryptjs
@types/jsonwebtoken eslint prettier eslint-config-prettier
eslint-plugin-prettier eslint-plugin-import eslint-plugin-node
eslint-plugin-promise drizzle-kit
npx tsx src/db/migrate.ts

# Generate Drizzle migration file after schema changes
npx drizzle-kit generate:pg

# Run the project locally
npm run dev

# Build the project for production
npm run build

# Run the project in production
npm run start
```

Configuration Files

package.json

```
{
  "name": "hono-baas-backend",
  "version": "1.0.0",
  "description": "A scalable Backend-as-a-Service foundation with
Hono, TypeScript, and PostgreSQL.",
  "main": "index.js",
  "scripts": {
    "dev": "tsx watch src/index.ts",
    "start": "node dist/index.js",
    "build": "tsc",
    "lint": "eslint . --ext .ts",
    "prettier": "prettier --write .",
    "migrate": "tsx src/db/migrate.ts"
  },
```

```
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "@types/bcryptjs": "^2.4.6",
    "@types/jsonwebtoken": "^9.0.5",
    "@types/node": "^20.11.19",
    "drizzle-kit": "^0.20.14",
    "eslint": "^8.56.0",
    "eslint-config-prettier": "^9.1.0",
    "eslint-plugin-import": "^2.29.1",
    "eslint-plugin-node": "^11.1.0",
    "eslint-plugin-prettier": "^5.1.3",
    "eslint-plugin-promise": "^6.1.1",
    "prettier": "^3.2.5",
    "tsx": "^4.7.1",
    "typescript": "^5.3.3"
  },
  "dependencies": {
    "@hono/zod-validator": "^0.1.11",
    "@neondatabase/serverless": "^0.8.1",
    "bcryptjs": "^2.4.3",
    "dotenv": "^16.4.5",
    "drizzle-orm": "^0.30.1",
    "hono": "^4.0.5",
    "jsonwebtoken": "^9.0.2",
    "zod": "^3.22.4"
  }
}
```

.env and .env.example

```
DATABASE_URL="your_postgresql_connection_string"
JWT_SECRET="your_secure_jwt_secret"
PORT=3000
```

tsconfig.json

```
{
  "compilerOptions": {
    "target": "es2021",
    "module": "commonjs",
    "rootDir": "./src",
    "outDir": "./dist",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true
  },
  "include": ["src/**/*.ts"],
```

```
  "exclude": ["node_modules"]
}
```

drizzle.config.ts

```
import 'dotenv/config';
import { defineConfig } from 'drizzle-kit';

export default defineConfig({
  schema: './src/db/schema.ts',
  out: './drizzle',
  dialect: 'postgresql',
  dbCredentials: {
    url: process.env.DATABASE_URL!,
  },
  verbose: true,
  strict: true,
});
```

Dockerfile

```
# Stage 1: Build the application
FROM node:20-alpine AS build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

# Stage 2: Run the application
FROM node:20-alpine AS production
WORKDIR /app
COPY --from=build /app/dist ./dist
COPY --from=build /app/node_modules ./node_modules
COPY --from=build /app/package*.json ./
COPY --from=build /app/.env .env
EXPOSE 3000
CMD ["npm", "start"]
```

.gitignore

```
node_modules/
dist/
.env
drizzle/
```

.prettierrc

```
{
  "semi": true,
  "trailingComma": "es5",
  "singleQuote": true,
```

```
  "printWidth": 80,
  "tabWidth": 2
}
```

.eslintrc.cjs
```
module.exports = {
  root: true,
  parser: '@typescript-eslint/parser',
  plugins: ['@typescript-eslint'],
  extends: [
    'eslint:recommended',
    'plugin:@typescript-eslint/eslint-recommended',
    'plugin:@typescript-eslint/recommended',
    'prettier',
  ],
  rules: {},
};
```

Code Files

src/index.ts
```
import 'dotenv/config';
import { Hono } from 'hono';
import { logger as honoLogger } from 'hono/logger';
import authRoute from './routes/auth.route';
import notesRoute from './routes/notes.route';
import { logger } from './utils/logger';

const app = new Hono();

// Middleware
app.use(honoLogger());

// Routes
app.route('/auth', authRoute);
app.route('/notes', notesRoute);

// Basic health check
app.get('/', (c) => {
  return c.json({ message: 'Hono BaaS API is running!' });
});

app.onError((err, c) => {
  logger.error('An error occurred:', err);
  return c.json({ error: 'Internal Server Error' }, 500);
});
```

```
const port = Number(process.env.PORT) || 3000;
logger.info(`Server is running on port ${port}`);

export default {
  port,
  fetch: app.fetch,
};
```

src/db/index.ts

```
import { neon } from '@neondatabase/serverless';
import { drizzle } from 'drizzle-orm/neon-http';
import * as schema from './schema';

const sql = neon(process.env.DATABASE_URL!);
export const db = drizzle(sql, { schema });
```

src/db/schema.ts

```
import { pgTable, text, timestamp, uuid } from 'drizzle-orm/pg-core';

export const users = pgTable('users', {
  id: uuid('id').defaultRandom().primaryKey(),
  email: text('email').unique().notNull(),
  passwordHash: text('password_hash').notNull(),
  createdAt: timestamp('created_at').defaultNow().notNull(),
});

export const notes = pgTable('notes', {
  id: uuid('id').defaultRandom().primaryKey(),
  title: text('title').notNull(),
  content: text('content'),
  userId: uuid('user_id').notNull().references(() => users.id),
  createdAt: timestamp('created_at').defaultNow().notNull(),
});
```

src/db/migrate.ts

```
import 'dotenv/config';
import { neon } from '@neondatabase/serverless';
import { drizzle } from 'drizzle-orm/neon-http';
import { migrate } from 'drizzle-orm/neon-http/migrator';
import { logger } from '../utils/logger';

async function main() {
  if (!process.env.DATABASE_URL) {
    throw new Error('DATABASE_URL is not set.');
  }
  const sql = neon(process.env.DATABASE_URL);
  const db = drizzle(sql);
```

```
  logger.info('Starting database migration...');

  try {
    await migrate(db, { migrationsFolder: './drizzle' });
    logger.info('Migration finished successfully!');
  } catch (error) {
    logger.error('Migration failed:', error);
    process.exit(1);
  } finally {
    process.exit(0);
  }
}

main();
```

src/routes/index.ts

```
import { Hono } from 'hono';
import authRoute from './auth.route';
import notesRoute from './notes.route';

const app = new Hono();

app.route('/auth', authRoute);
app.route('/notes', notesRoute);

export default app;
```

src/routes/auth.route.ts

```
import { Hono } from 'hono';
import { zValidator } from '@hono/zod-validator';
import { z } from 'zod';
import { register, login } from '../controllers/auth.controller';

const authRoute = new Hono();

const registerSchema = z.object({
  email: z.string().email(),
  password: z.string().min(8),
});

const loginSchema = z.object({
  email: z.string().email(),
  password: z.string().min(8),
});

authRoute.post('/register', zValidator('json', registerSchema),
register);
authRoute.post('/login', zValidator('json', loginSchema), login);
```

```
export default authRoute;
```

src/routes/notes.route.ts

```
import { Hono } from 'hono';
import { zValidator } from '@hono/zod-validator';
import { z } from 'zod';
import { authMiddleware } from '../middleware/auth.middleware';
import {
  createNote,
  getNotes,
  getNoteById,
  updateNote,
  deleteNote,
} from '../controllers/notes.controller';

const notesRoute = new Hono();

const createNoteSchema = z.object({
  title: z.string().min(1),
  content: z.string().optional(),
});

const updateNoteSchema = z.object({
  title: z.string().min(1).optional(),
  content: z.string().optional(),
});

notesRoute.use(authMiddleware);

notesRoute.post('/', zValidator('json', createNoteSchema),
createNote);
notesRoute.get('/', getNotes);
notesRoute.get('/:id', getNoteById);
notesRoute.put('/:id', zValidator('json', updateNoteSchema),
updateNote);
notesRoute.delete('/:id', deleteNote);

export default notesRoute;
```

src/controllers/auth.controller.ts

```
import { Context } from 'hono';
import { registerUser, loginUser } from '../services/auth.service';

export const register = async (c: Context) => {
  try {
    const body = await c.req.json();
    const result = await registerUser(body.email, body.password);
```

```
    return c.json(result, 201);
  } catch (error: any) {
    if (error.message === 'User already exists') {
      return c.json({ error: error.message }, 409);
    }
    return c.json({ error: 'Failed to register user' }, 500);
  }
};

export const login = async (c: Context) => {
  try {
    const body = await c.req.json();
    const result = await loginUser(body.email, body.password);
    if (!result) {
      return c.json({ error: 'Invalid credentials' }, 401);
    }
    return c.json(result);
  } catch (error) {
    return c.json({ error: 'Failed to log in' }, 500);
  }
};
```

src/controllers/notes.controller.ts

```
import { Context } from 'hono';
import {
  createNote as createNoteService,
  getNotes as getNotesService,
  getNoteById as getNoteByIdService,
  updateNote as updateNoteService,
  deleteNote as deleteNoteService,
} from '../services/notes.service';

export const createNote = async (c: Context) => {
  try {
    const user = c.get('user');
    const { title, content } = await c.req.json();
    const note = await createNoteService(title, content, user.id);
    return c.json(note, 201);
  } catch (error) {
    return c.json({ error: 'Failed to create note' }, 500);
  }
};

export const getNotes = async (c: Context) => {
  try {
    const user = c.get('user');
    const notes = await getNotesService(user.id);
    return c.json(notes);
```

```
  } catch (error) {
    return c.json({ error: 'Failed to fetch notes' }, 500);
  }
};

export const getNoteById = async (c: Context) => {
  try {
    const user = c.get('user');
    const noteId = c.req.param('id');
    const note = await getNoteByIdService(noteId, user.id);
    if (!note) {
      return c.json({ error: 'Note not found' }, 404);
    }
    return c.json(note);
  } catch (error) {
    return c.json({ error: 'Failed to fetch note' }, 500);
  }
};

export const updateNote = async (c: Context) => {
  try {
    const user = c.get('user');
    const noteId = c.req.param('id');
    const { title, content } = await c.req.json();
    const updatedNote = await updateNoteService(noteId, user.id, {
      title,
      content,
    });
    if (!updatedNote) {
      return c.json({ error: 'Note not found or you do not have
permission' }, 404);
    }
    return c.json(updatedNote);
  } catch (error) {
    return c.json({ error: 'Failed to update note' }, 500);
  }
};

export const deleteNote = async (c: Context) => {
  try {
    const user = c.get('user');
    const noteId = c.req.param('id');
    const result = await deleteNoteService(noteId, user.id);
    if (!result) {
      return c.json({ error: 'Note not found or you do not have
permission' }, 404);
    }
    return c.json({ message: 'Note deleted successfully' });
```

```
  } catch (error) {
    return c.json({ error: 'Failed to delete note' }, 500);
  }
};
```

src/services/auth.service.ts

```
import { eq } from 'drizzle-orm';
import { db } from '../db';
import { users } from '../db/schema';
import * as bcrypt from 'bcryptjs';
import { generateToken } from '../utils/jwt';

export const registerUser = async (email: string, password: string) =>
{
  const existingUser = await db.query.users.findFirst({
    where: eq(users.email, email),
  });

  if (existingUser) {
    throw new Error('User already exists');
  }

  const passwordHash = await bcrypt.hash(password, 10);
  const [newUser] = await db.insert(users).values({ email,
passwordHash }).returning();
  const token = generateToken({ id: newUser.id, email: newUser.email
});

  return { token, user: { id: newUser.id, email: newUser.email } };
};

export const loginUser = async (email: string, password: string) => {
  const user = await db.query.users.findFirst({
    where: eq(users.email, email),
  });

  if (!user) {
    return null;
  }

  const isMatch = await bcrypt.compare(password, user.passwordHash);
  if (!isMatch) {
    return null;
  }

  const token = generateToken({ id: user.id, email: user.email });
  return { token, user: { id: user.id, email: user.email } };
};
```

src/services/notes.service.ts

```
import { eq, and } from 'drizzle-orm';
import { db } from '../db';
import { notes } from '../db/schema';

export const createNote = async (
  title: string,
  content: string | undefined,
  userId: string
) => {
  const [newNote] = await db
    .insert(notes)
    .values({
      title,
      content,
      userId,
    })
    .returning();
  return newNote;
};

export const getNotes = async (userId: string) => {
  return db.query.notes.findMany({
    where: eq(notes.userId, userId),
  });
};

export const getNoteById = async (noteId: string, userId: string) => {
  return db.query.notes.findFirst({
    where: and(eq(notes.id, noteId), eq(notes.userId, userId)),
  });
};

export const updateNote = async (
  noteId: string,
  userId: string,
  data: { title?: string; content?: string }
) => {
  const [updatedNote] = await db
    .update(notes)
    .set(data)
    .where(and(eq(notes.id, noteId), eq(notes.userId, userId)))
    .returning();
  return updatedNote;
};

export const deleteNote = async (noteId: string, userId: string) => {
```

```
  const [deletedNote] = await db
    .delete(notes)
    .where(and(eq(notes.id, noteId), eq(notes.userId, userId)))
    .returning();
  return deletedNote;
};
```

src/middleware/auth.middleware.ts

```
import { Context, Next } from 'hono';
import { verifyToken } from '../utils/jwt';

export const authMiddleware = async (c: Context, next: Next) => {
  const authHeader = c.req.header('Authorization');
  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return c.json({ error: 'Unauthorized' }, 401);
  }

  const token = authHeader.split(' ')[1];
  try {
    const user = verifyToken(token);
    c.set('user', user);
    await next();
  } catch (error) {
    return c.json({ error: 'Unauthorized' }, 401);
  }
};
```

src/utils/jwt.ts

```
import jwt from 'jsonwebtoken';

const JWT_SECRET = process.env.JWT_SECRET!;

export const generateToken = (payload: object) => {
  return jwt.sign(payload, JWT_SECRET, { expiresIn: '1h' });
};

export const verifyToken = (token: string) => {
  return jwt.verify(token, JWT_SECRET);
};
```

src/utils/logger.ts

```
import chalk from 'chalk';

type LogLevel = 'info' | 'warn' | 'error';

const log = (level: LogLevel, message: string, ...args: any[]) => {
  const timestamp = new Date().toISOString();
  switch (level) {
```

```
    case 'info':
      console.log(chalk.cyan(`[${timestamp}] [INFO]`), message,
...args);
      break;
    case 'warn':
      console.warn(chalk.yellow(`[${timestamp}] [WARN]`), message,
...args);
      break;
    case 'error':
      console.error(chalk.red(`[${timestamp}] [ERROR]`), message,
...args);
      break;
  }
};

export const logger = {
  info: (message: string, ...args: any[]) => log('info', message,
...args),
  warn: (message: string, ...args: any[]) => log('warn', message,
...args),
  error: (message: string, ...args: any[]) => log('error', message,
...args),
};
```

Run Instructions

1. **Set up your .env file:** Copy the contents of .env.example into a new file named .env and fill in your PostgreSQL connection string and a secure JWT secret.
2. **Install dependencies:** Run npm install in your project's root directory.
3. **Run database migration:** Execute npm run migrate to create the users and notes tables in your database.
4. **Development mode:** Start the development server with npm run dev. This uses tsx for hot-reloading.
5. **Production deployment:**
    - Build the project: npm run build
    - Run the production build: npm run start
    - Alternatively, use the provided Dockerfile to containerize your application for production:
        - Build the image: docker build -t hono-baas-backend .
        - Run the container: docker run -p 3000:3000 --env-file ./.env hono-baas-backend