spur-express

- [spur-express github repo](#)

Journey

Setup Github Repo: [spur-express github repo](#)

---

chore: bootstrap Express + TypeScript backend with health check and gitignore

- Project initialization

```
mkdir backend frontend
cd backend
npm init -y
npm install express cors dotenv
npm install -D typescript ts-node-dev @types/node @types/express
@types/cors
npx tsc --init
```

- tsconfig.json: Configured TypeScript for a stable Node.js backend with strict type safety and a clear source-to-build output flow.

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "rootDir": "src",
    "outDir": "dist",
    "strict": true,
    "esModuleInterop": true
  }
}
```

- src/app.ts: Created a minimal Express app with CORS and JSON middleware, plus a /health endpoint to verify server liveness.

```
import express from "express";
import cors from "cors";

const app = express();

app.use(cors());
```

```
app.use(express.json());

app.get("/health", (_, res) => {
  res.json({ status: "ok" });
});

export default app;
```

- src/server.ts: Separated server startup from app configuration and enabled environment variable loading for deployment readiness.

```
import app from "./app";
import dotenv from "dotenv";

dotenv.config();

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {
  console.log(`spur-express backend running on port ${PORT}`);
});
```

- package.json: Added development, build, and production scripts to support local development and deployment on Render.

```
"scripts": {
  "dev": "ts-node-dev --respawn src/server.ts",
  "build": "tsc",
  "start": "node dist/server.js"
}
```

- Started the backend in watch mode to ensure the server runs correctly during development.

```
npm run dev
```

- Health check validation

```
http://localhost:3000/health
```

- Confirmed the backend is running successfully and responding to requests.

```
{ "status": "ok" }
```

feat: connect backend to Supabase Postgres and verify db health

- init supabase project 'spur-express'
  - db URI
    - conn. str. : `postgresql://postgres:`
      `<PASSWORD>@db.xxxxx.supabase.co:5432/postgres`
    - stored at `.env`

```
DATABASE_URL=postgresql://postgres:YOUR_PASSWORD@db.xxxxx.supabase.co:5432/
postgres
PORT=3000
```

- packs

```
npm install pg uuid
npm install -D @types/pg
```

- backend/src/db/index.ts: db client using pool, manager that keeps a small set of open database connections and reuses them instead of opening a new one for every query.

```typescript
import { Pool } from "pg";

if (!process.env.DATABASE_URL) {
  throw new Error("DATABASE_URL is missing");
}

export const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
  ssl: { rejectUnauthorized: false }, // REQUIRED for Supabase
});
```

- supabase sql query init:
  - conversations represents one chat session (thread) so all messages are grouped correctly.
  - messages stores each user/AI message with sender, order, and automatic cleanup via ON DELETE CASCADE.

```sql
CREATE TABLE IF NOT EXISTS conversations (
  id UUID PRIMARY KEY,
  created_at TIMESTAMP DEFAULT now()
);

CREATE TABLE IF NOT EXISTS messages (
```

/

```
  id UUID PRIMARY KEY,
  conversation_id UUID REFERENCES conversations(id) ON DELETE CASCADE,
  sender TEXT CHECK (sender IN ('user', 'ai')),
  text TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT now()
);
```

- Modify src/app.ts: Verify DB connection from Express

```
import express from "express";
import cors from "cors";
import { pool } from "./db";

const app = express();

app.use(cors());
app.use(express.json());

app.get("/health", (_, res) => {
  res.json({ status: "ok" });
});
app.get("/db-health", async (_, res) => {const result = await
pool.query("SELECT now()");
  res.json({ db: "connected", time: result.rows[0] });
});

export default app;
```

- Modify src/server.ts: Move dotenv.config() to the TOP, Node evaluates imports first, so dotenv.config() must run before importing anything

```
import dotenv from "dotenv"; // Node evaluates imports first, so
dotenv.config() must run before importing anything that reads process.env.

dotenv.config();

import app from "./app";

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {
  console.log(`spur-express backend running on port ${PORT}`);
});
```

- run & check

```
npm run dev
```

```
http://localhost:3000/db-health
```

```
{"db":"connected","time":{"now":"2025-12-18T22:20:54.578Z"}}
```

---

feat: implement chat message API with session-based conversations

- init: Creates folders and files to separate routes, controllers, and services so chat logic stays clean and organized.

```
mkdir -p src/routes src/controllers src/services
touch src/routes/chat.routes.ts
touch src/controllers/chat.controller.ts
touch src/services/chat.service.ts
# bali-king@war-machine:~/BaliGit/spur-express/backend$ tree src
# src
# ├── app.ts
# ├── controllers
# │   └── chat.controller.ts
# ├── db
# │   └── index.ts
# ├── routes
# │   └── chat.routes.ts
# ├── server.ts
# └── services
#     └── chat.service.ts

# 5 directories, 6 files
# bali-king@war-machine:~/BaliGit/spur-express/backend$
```

- src/services/chat.service.ts: Chat service (core logic)single responsibility, pure domain logic, LLM plugs in later with zero refactor, chat.service.ts holds all chat business logic and talks to the database without knowing anything about HTTP or Express.

```
import { pool } from "../db";
import { v4 as uuidv4 } from "uuid";

export async function createConversation(): Promise<string> {
  const id = uuidv4();
  await pool.query(
    "INSERT INTO conversations (id) VALUES ($1)",
```

```
      [id]
  );
  return id;
}

export async function saveMessage(
  conversationId: string,
  sender: "user" | "ai",
  text: string
) {
  await pool.query(
    `INSERT INTO messages (id, conversation_id, sender, text)
     VALUES ($1, $2, $3, $4)`,
    [uuidv4(), conversationId, sender, text]
  );
}

export async function getConversationHistory(conversationId: string) {
  const { rows } = await pool.query(
    `SELECT sender, text
     FROM messages
     WHERE conversation_id = $1
     ORDER BY created_at ASC`,
    [conversationId]
  );
  return rows;
}
```

- src/controllers/chat.controller.ts: Controller (request handling)
  - createConversation generates a new UUID and creates a fresh conversation record in the database.
  - saveMessage stores a single message linked to a conversation and tags who sent it (user or AI).
  - Imports chat services and keeps HTTP logic separate from business logic.
  - Extracts message and sessionId from the request body.
  - Validates input so empty or invalid messages are rejected early.
  - Creates a new conversation if no sessionId is provided.
  - Saves the user message to the database.
  - Generates a temporary AI reply as a placeholder for future LLM integration.
  - Saves the AI reply using the same message pipeline.
  - Fetches full conversation history in order.
  - Returns reply, sessionId, and history to the client.

```
import { Request, Response } from "express";
import {
  createConversation,
  saveMessage,
  getConversationHistory,
} from "../services/chat.service";
```

```typescript
export async function postMessage(req: Request, res: Response) {
  const { message, sessionId } = req.body;

  if (!message || typeof message !== "string" || !message.trim()) {
    return res.status(400).json({ error: "Message is required" });
  }

  const conversationId = sessionId || await createConversation();

  await saveMessage(conversationId, "user", message);

  // TEMP reply (LLM comes next phase)
  const reply = "Thanks! Your message has been received.";

  await saveMessage(conversationId, "ai", reply);

  const history = await getConversationHistory(conversationId);

  res.json({
    reply,
    sessionId: conversationId,
    history,
  });
}
```

- src/routes/chat.routes.ts: Route
    - Defines the chat message endpoint.
    - Connects the HTTP route to the controller.
    - Keeps routing logic thin and declarative.

```typescript
import { Router } from "express";
import { postMessage } from "../controllers/chat.controller";

const router = Router();

router.post("/message", postMessage);

export default router;
```

- Modify src/app.ts: enable chat route
    - Mounts chat routes under /chat namespace.
    - Keeps the main app clean and modular.

```typescript
import chatRoutes from "./routes/chat.routes";

app.use("/chat", chatRoutes);
```

- test it: Starts the development server with chat APIs enabled.

```
npm run dev
```

- curl: Sends a test chat message without a sessionId.Automatically creates a new conversation.

```
curl -X POST http://localhost:3000/chat/message \
  -H "Content-Type: application/json" \
  -d '{"message":"Do you ship to USA?"}'
```

- postman

```
{
          "name": "http://localhost:3000/chat/message",
          "request": {
              "method": "POST",
              "header": [
                  {
                      "key": "Content-Type",
                      "value": "application/json",
                      "type": "text"
                  }
              ],
              "body": {
                  "mode": "raw",
                  "raw": "{\n  \"message\": \"Do you ship to USA?\"\n}",
                  "options": {
                      "raw": {
                          "language": "json"
                      }
                  }
              },
              "url": {
                  "raw": "http://localhost:3000/chat/message",
                  "protocol": "http",
                  "host": [
                      "localhost"
                  ],
                  "port": "3000",
                  "path": [
                      "chat",
                      "message"
                  ]
              }
          },
          "response": []
      }
```
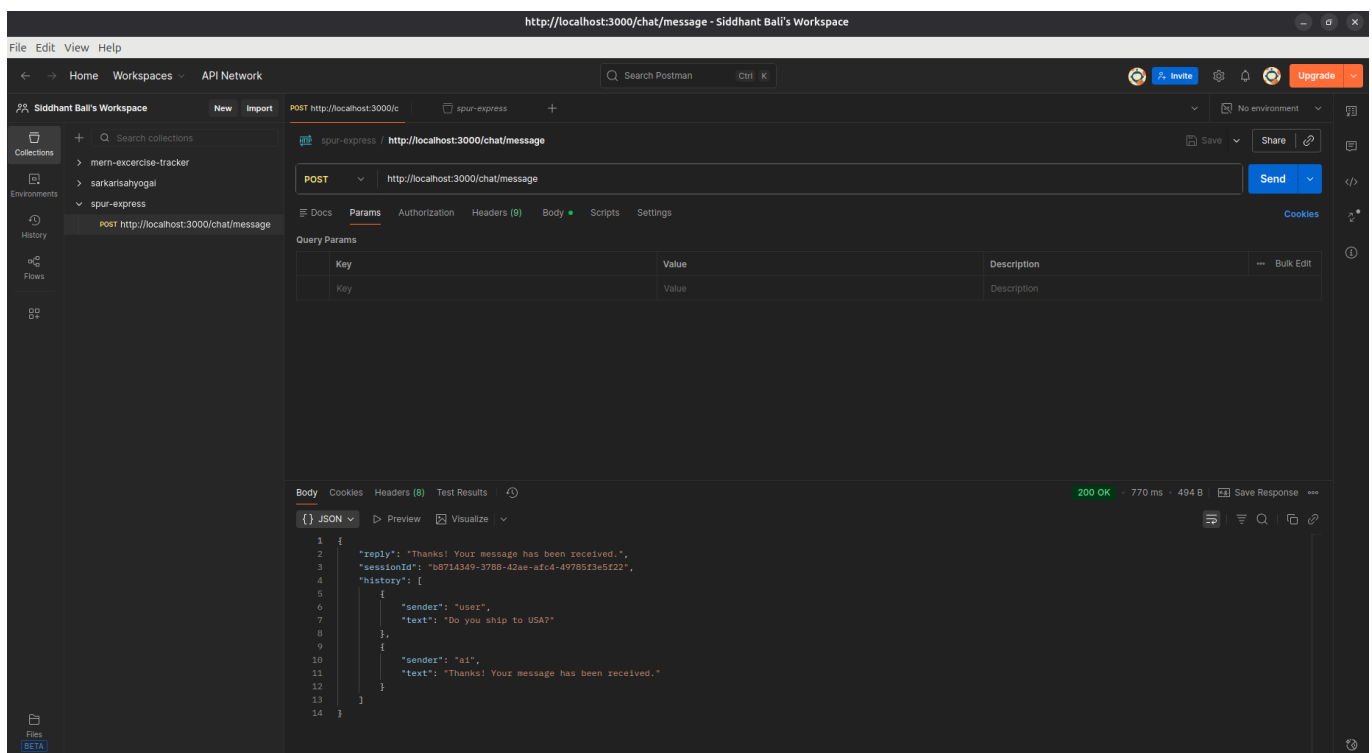
- output
  - Returns AI reply, sessionId, and full message history.
  - Confirms session-based conversation works end to end.

```
{
    "reply": "Thanks! Your message has been received.",
    "sessionId": "b8714349-3788-42ae-afc4-49785f3e5f22",
    "history": [
        {
            "sender": "user",
            "text": "Do you ship to USA?"
        },
        {
            "sender": "ai",
            "text": "Thanks! Your message has been received."
        }
    ]
}
```


---

feat: integrate Gemini LLM for contextual chat replies

- Gemini LLM API init
  - gemini-2.5-flash
    - Text-out model
    - Fast & cheap
    - Fully supported for chat use cases
    - Exactly what Google recommends for general chat now

- gemini-2.5-flash-tts: text-to-speech (wrong)
- gemini-2.5-flash-native-audio-dialog: audio/live API (wrong)
- gemini-3-flash: newer, sometimes gated / unstable
- gemma-*: open models, not Gemini chat API
- robotics / preview models: not for chat

```
npm install @google/generative-ai
```

```
# backend/.env
GEMINI_API_KEY=your_api_key_here
```

- backend/src/services/llm.service.ts
  - LLM logic is isolated
  - Prompt is deterministic
  - FAQ knowledge is injected
  - Errors are handled gracefully
  - Easy to swap Gemini → OpenAI later

```typescript
import { GoogleGenerativeAI } from "@google/generative-ai";
// console.log("Gemini key loaded:", !!process.env.GEMINI_API_KEY);

if (!process.env.GEMINI_API_KEY) {
  throw new Error("GEMINI_API_KEY is missing");
}

const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);

const model = genAI.getGenerativeModel({
  model: "gemini-2.5-flash",
});

const SYSTEM_PROMPT = `
You are a helpful customer support agent for a small e-commerce store.

Store policies:
- Shipping: Ships in 3-5 business days. We ship to India and USA.
- Returns: 7-day return policy. Items must be unused.
- Support hours: Monday to Friday, 10am-6pm IST.

Answer clearly, concisely, and politely.
If you are unsure, say you don't know.
`;

export async function generateReply(
  history: { sender: string; text: string }[],
  userMessage: string
): Promise<string> {
```

```
    try {
      const conversation = history
        .map(m => `${m.sender}: ${m.text}`)
        .join("\n");

      const prompt = `
${SYSTEM_PROMPT}

Conversation so far:
${conversation}

User: ${userMessage}
AI:
`;

      const result = await model.generateContent(prompt);
      const response = result.response.text();

      return response.trim();
    } catch (error) {
      console.error("Gemini error:", error);
      return "Sorry, I'm having trouble responding right now. Please try
again later.";
    }
}
```

- Modify backend/src/controllers/chat.controller.ts: Replace dummy reply with Gemini
  - SDK usage
  - Prompt construction
  - History injection
  - Error handling
  - Guardrails

```
import { Request, Response } from "express";
import {
  createConversation,
  saveMessage,
  getConversationHistory,
} from "../services/chat.service";
import { generateReply } from "../services/llm.service";

export async function postMessage(req: Request, res: Response) {
  const { message, sessionId } = req.body;

  if (!message || typeof message !== "string" || !message.trim()) {
    return res.status(400).json({ error: "Message is required" });
  }

  const conversationId = sessionId || await createConversation();
```

```
    await saveMessage(conversationId, "user", message);

    // // TEMP reply (LLM comes next phase)
    // const reply = "Thanks! Your message has been received.";

    // fetch history BEFORE LLM call
    const history = await getConversationHistory(conversationId);

    const reply = await generateReply(history, message);

    await saveMessage(conversationId, "ai", reply);

    // fetch updated history (includes AI reply)
    const updatedHistory = await getConversationHistory(conversationId);

    res.json({
      reply,
      sessionId: conversationId,
      history: updatedHistory,
    });
}
```

- Restart Server and Same Postman Test

```
POST http://localhost:3000/chat/message
```

```
{
  "message": "Do you ship to USA?"
}
```

- Output
    - API endpoint works
    - Conversation created
    - User message saved
    - History fetched
    - generateReply() called
    - Error handling executed
    - AI reply saved
    - Response returned

```
{
    "reply": "Yes, we do! We ship to both USA and India.",
    "sessionId": "eab018be-16f8-43c6-bc4b-8303b2baeac7",
    "history": [
        {
```

```
            "sender": "user",
            "text": "Do you ship to USA?"
        },
        {
            "sender": "ai",
            "text": "Yes, we do! We ship to both USA and India."
        }
    ]
}
```