Agentic AI

Open Source by WrtnLabs
2025-05-22

Introduction

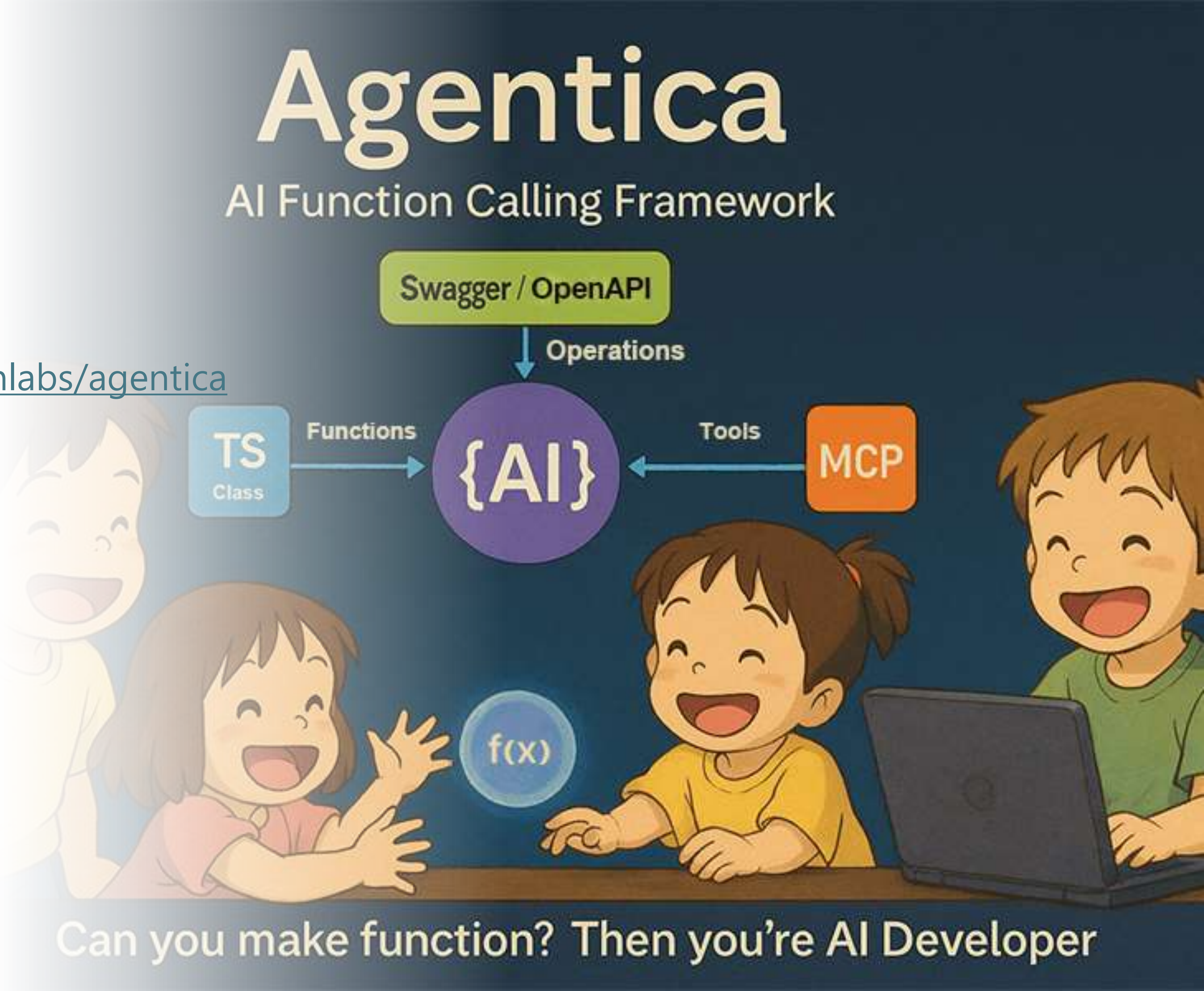
- https://github.com/wrtnlabs

- WrtnLabs in Wrtn Technologies
- Open-Source Development Team
- Researching Viral Coding with Compiler Skills

Agentica

- https://github.com/wrtnlabs/agentica
- AI Function Calling

- Easy
- Scalable
- Flexible
- Mass Productive


Agentica
AI Function Calling Framework
Swagger / OpenAPI
Operations
TS
Class
Functions
{AI}
Tools
MCP
f(x)
Can you make function? Then you're AI Developer

AutoBE

- https://github.com/wrtnlabs/autobe
- Backend Automation Tool
- Viral Coding

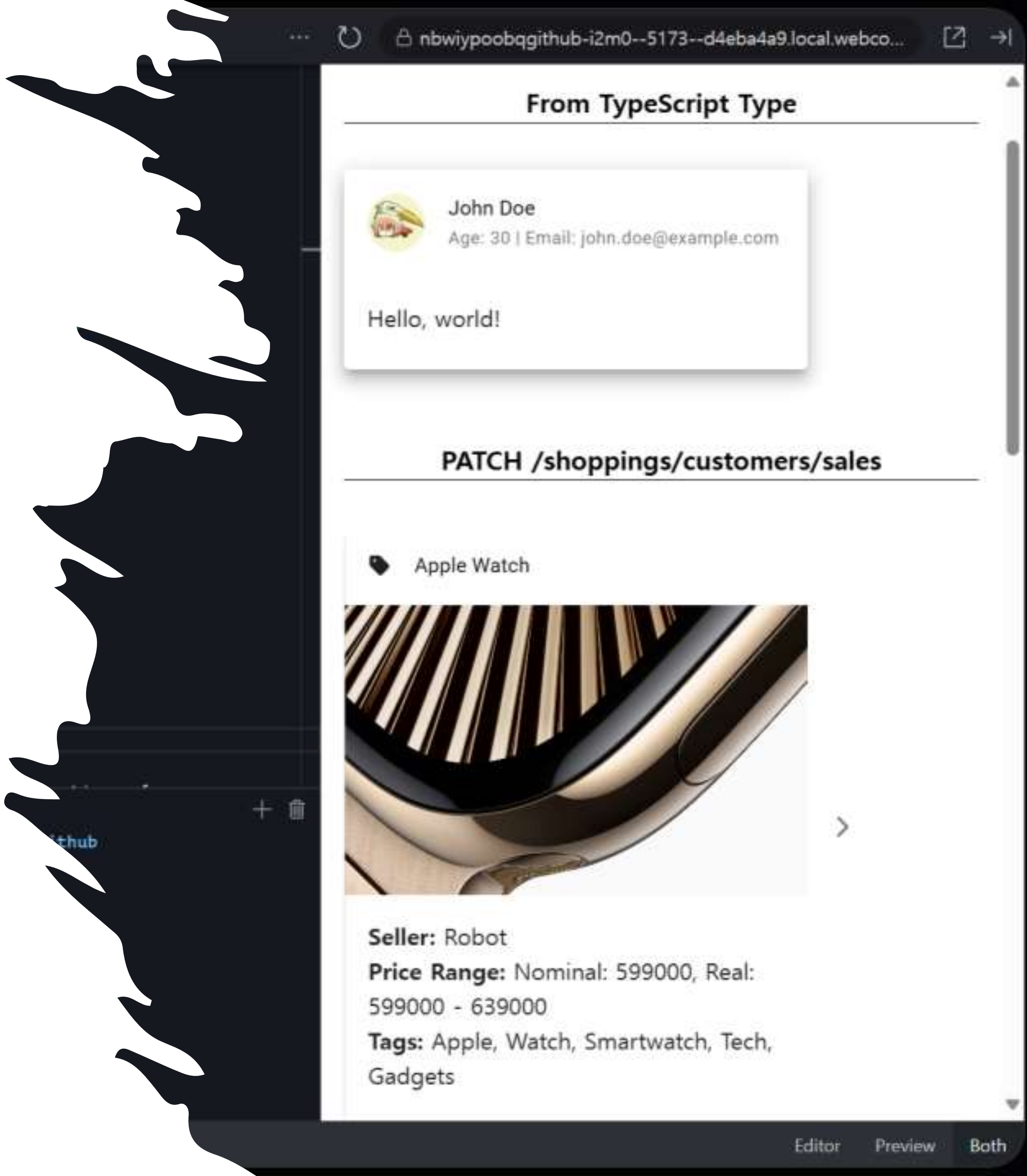
- Debate Requirements with AI
- Then AI makes Backend Program

- TypeScript
- Prisma
- NestJS / Nestia

AutoView

- https://github.com/wrtnlabs/autoview
- Frontend Automation Tool
- Type to Frontend Code

- TypeScript Type
- OpenAPI Operation Schema
- AI Function Calling Schema



From TypeScript Type

John Doe

Age: 30 | Email: john.doe@example.com

Hello, world!

PATCH /shoppings/customers/sales

Apple Watch

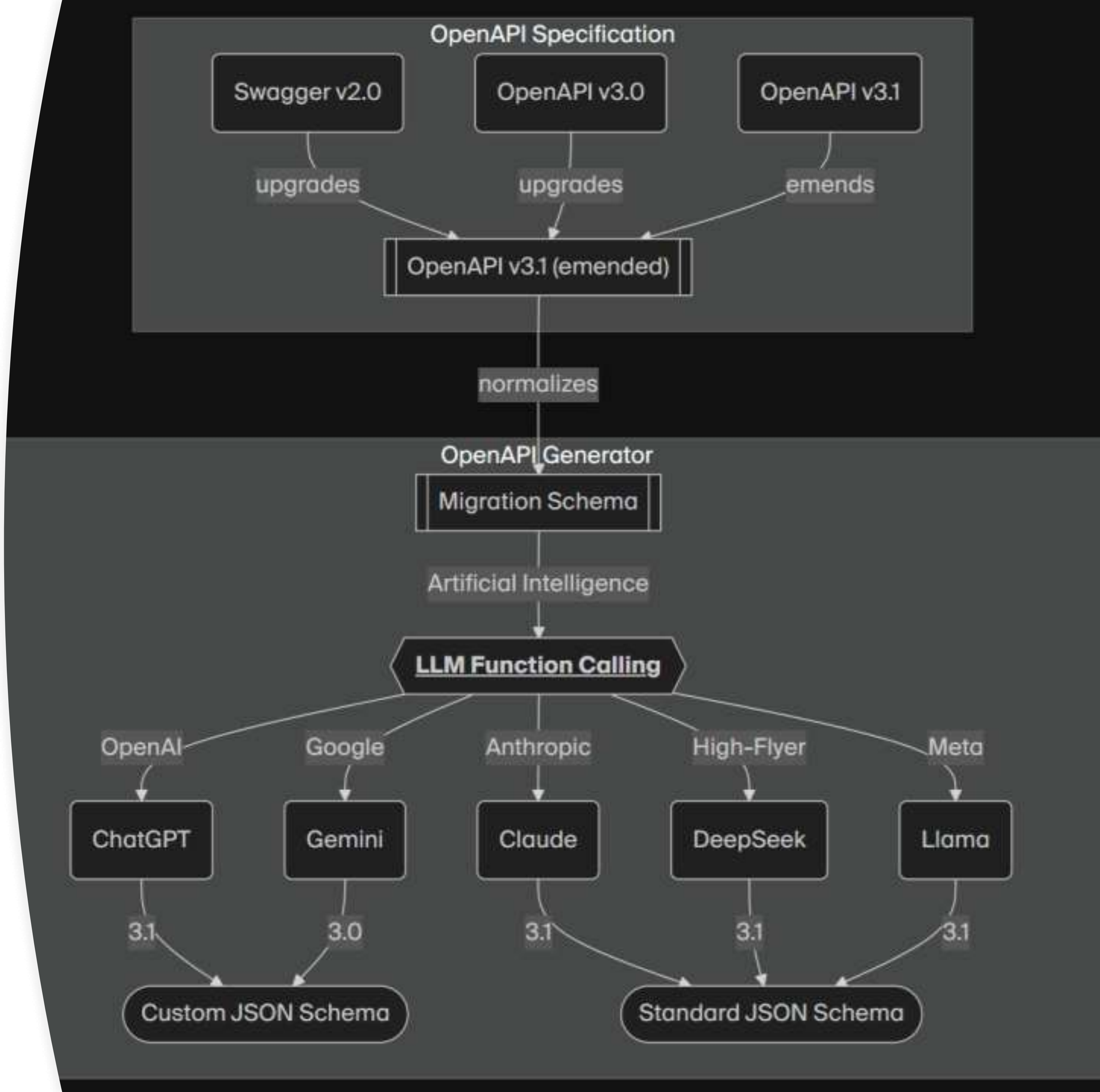
Seller: Robot

Price Range: Nominal: 599000, Real: 599000 - 639000

Tags: Apple, Watch, Smartwatch, Tech, Gadgets

Fundamental

- JSON Schema Specification
- Compiler Skills
- Documentation



Index

Agentica

- AI Function Calling
- JSON Schema Specification
- Compiler Driven Development

AutoBE

- Backend Automation Tool
- Spiral Waterfall Model
- Compiler Feedback

AutoView

- Frontend Automation Tool
- Principles
- Blueprint

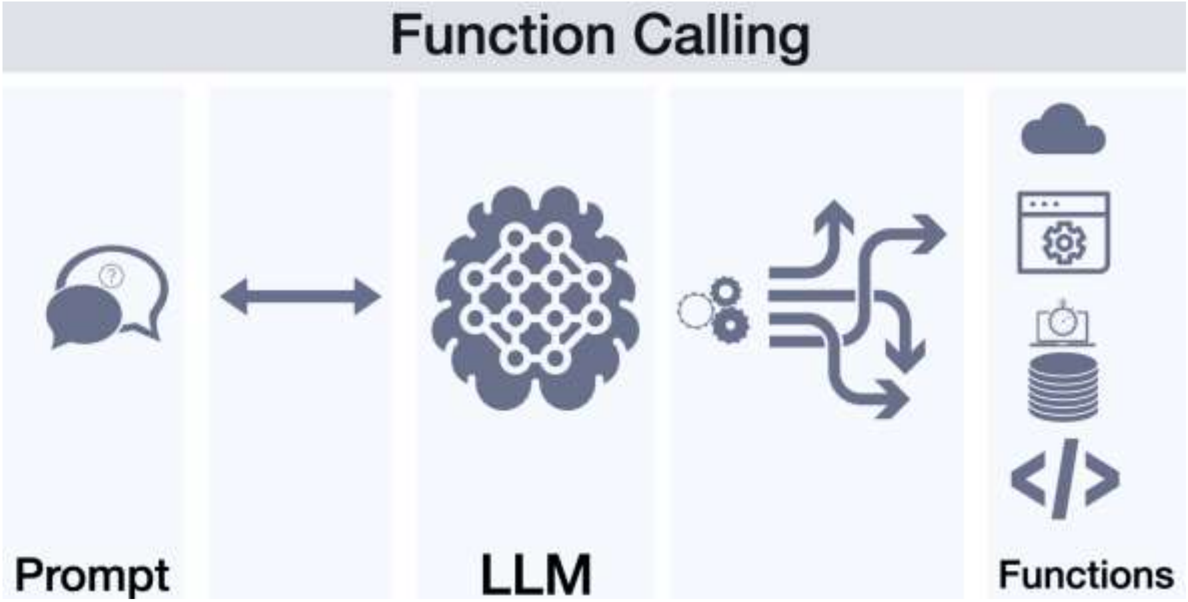
Agentica
AI Function Calling Framework
Swagger / OpenAPI
Operations
TS
Class
Functions
{AI}
Tools
MCP
f(x)
Can you make function? Then you're AI Developer

Agentica

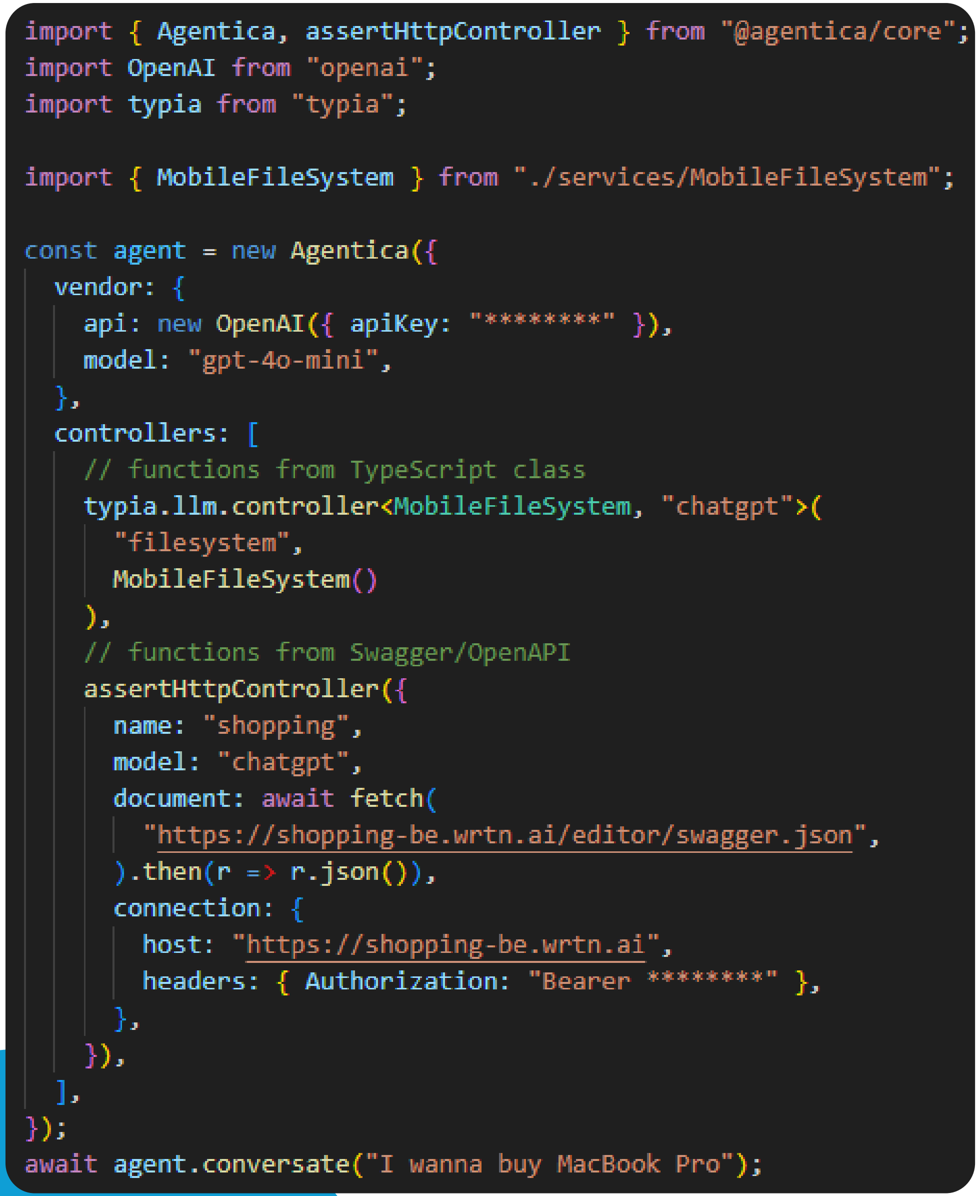
AI Function Calling
JSON Schema Specification
Compiler Driven Development

AI Function Callinig

- AI selects proper function
- Fill arguments of the function
- Bu analyzing chatting histories



AI Function Calling



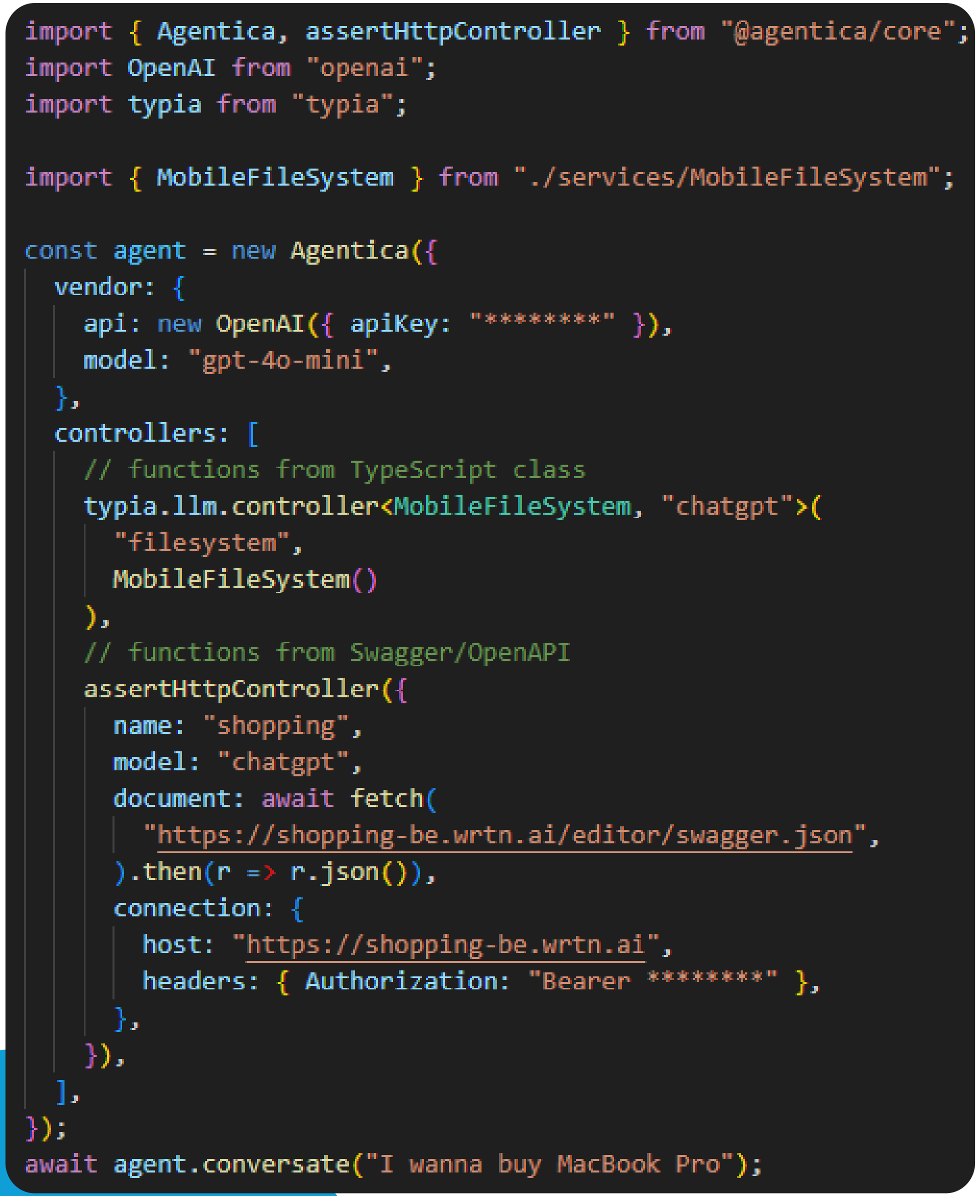
```
import { Agentica, assertHttpController } from "@agentica/core";
import OpenAI from "openai";
import typia from "typia";

import { MobileFileSystem } from "./services/MobileFileSystem";

const agent = new Agentica({
  vendor: {
    api: new OpenAI({ apiKey: "********" }),
    model: "gpt-4o-mini",
  },
  controllers: [
    // functions from TypeScript class
    typia.llm.controller<MobileFileSystem, "chatgpt">(
      "filesystem",
      MobileFileSystem()
    ),
    // functions from Swagger/OpenAPI
    assertHttpController({
      name: "shopping",
      model: "chatgpt",
      document: await fetch(
        "https://shopping-be.wrtn.ai/editor/swagger.json",
      ).then(r => r.json()),
      connection: {
        host: "https://shopping-be.wrtn.ai",
        headers: { Authorization: "Bearer ********" },
      },
    }),
  ],
});
await agent.conversate("I wanna buy MacBook Pro");
```

- https://github.com/wrtnlabs/agentica

- Agentic AI Framework
- Specialized in Function Calling
- And Doing Everythihng with FC

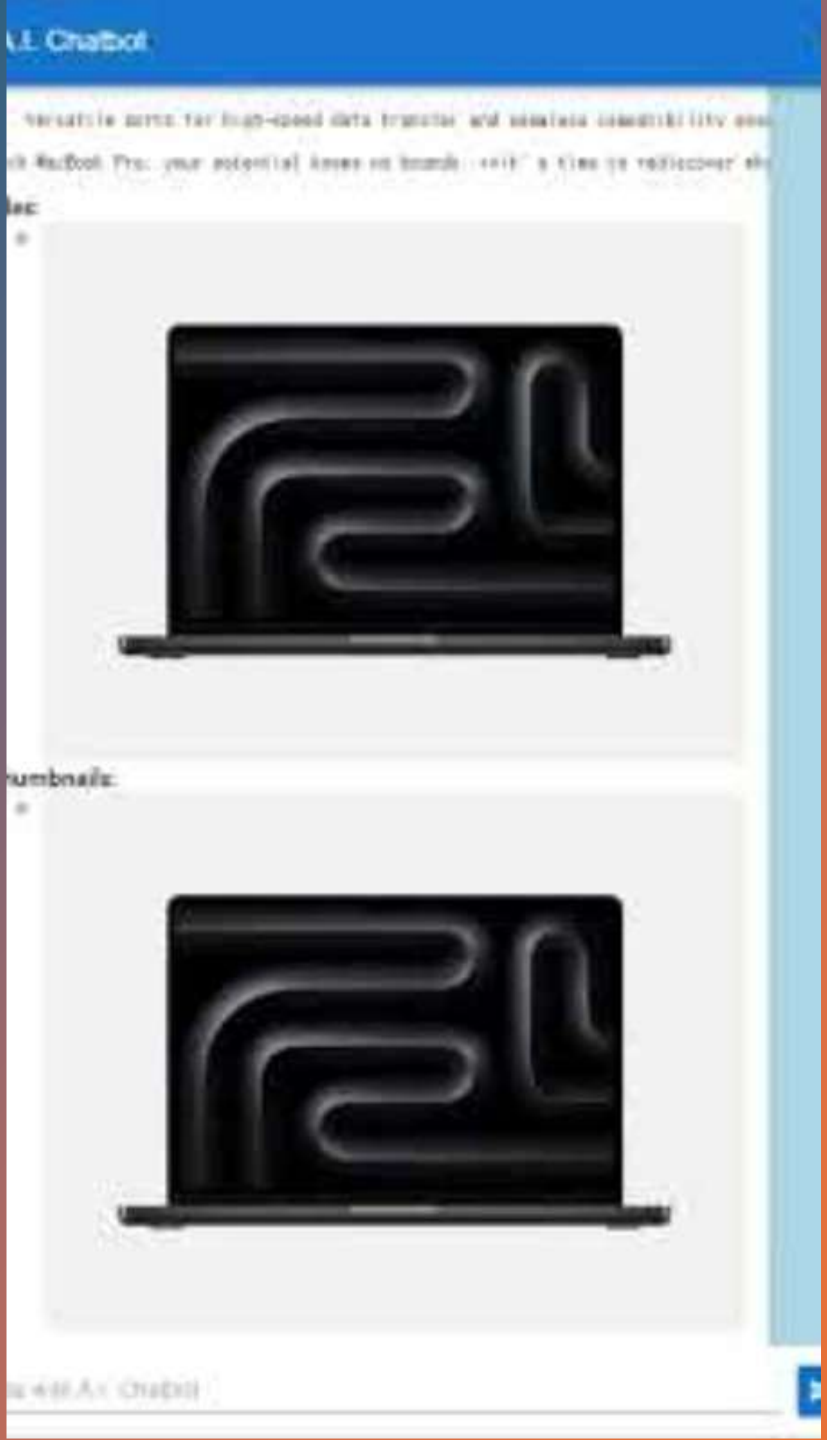
- TypeScript Class
- Swagger/OpenAPI Document
- MCP (Model Context Protocol)

AI Function Calling



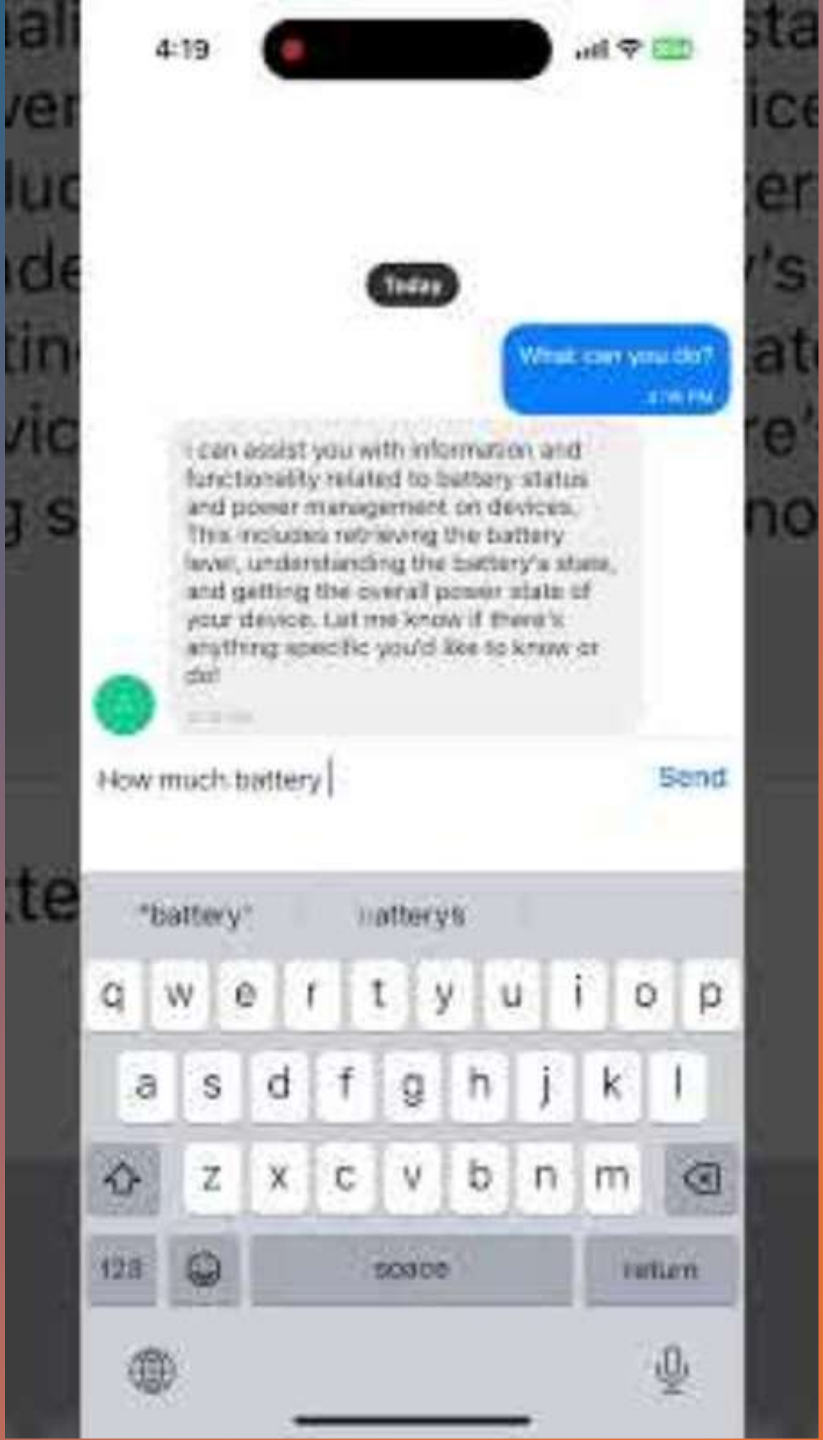
```
import { Agentica, assertHttpController } from "@agentica/core";
import OpenAI from "openai";
import typia from "typia";

import { MobileFileSystem } from "./services/MobileFileSystem";

const agent = new Agentica({
  vendor: {
    api: new OpenAI({ apiKey: "********" }),
    model: "gpt-4o-mini",
  },
  controllers: [
    // functions from TypeScript class
    typia.llm.controller<MobileFileSystem, "chatgpt">(
      "filesystem",
      MobileFileSystem()
    ),
    // functions from Swagger/OpenAPI
    assertHttpController({
      name: "shopping",
      model: "chatgpt",
      document: await fetch(
        "https://shopping-be.wrtn.ai/editor/swagger.json",
      ).then(r => r.json()),
      connection: {
        host: "https://shopping-be.wrtn.ai",
        headers: { Authorization: "Bearer ********" },
      },
    }),
  ],
});
await agent.conversate("I wanna buy MacBook Pro");
```

- The world's easiest
- Just by listing up functions
  - TypeScript Class
  - Swagger/OpenAPI Document
  - Model Context Protocol
- You can make any agent
  - Easy
  - Scalable
  - Flexible
  - Mass Productive



AI Function Calling

- Shopping Mall
- Enterprise features
  - 289 API functions
  - Product, order, payment, refund, review, coupon, deposit, withdrawal, delivery, and so on
- Agent just by a swagger file
- Working on 8b model (4o-mini)



What can you do?

I can assist you with information and functionality related to battery status and power management on devices. This includes retrieving the battery level, understanding the battery's state, and getting the overall power state of your device. Let me know if there's anything specific you'd like to know or do!

How much battery

AI Function Calling

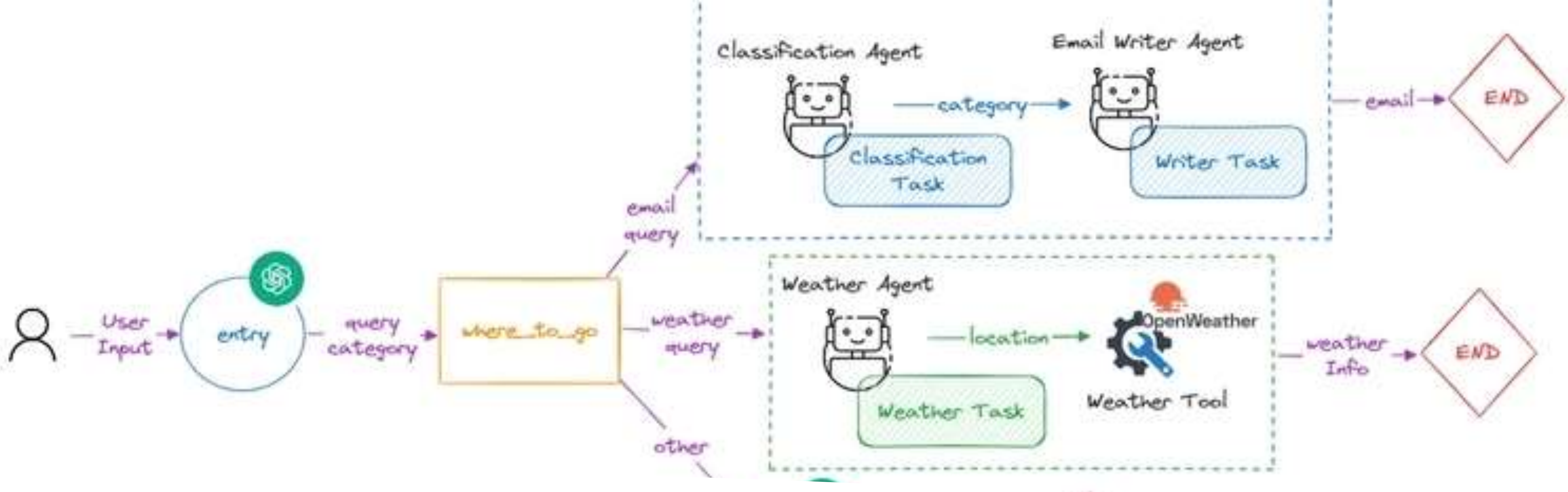
- React Native Application
- Function Call to Device API

- 20yo student
- Defeated Apple Siri
- Just by few minutes

Failure of Function Calling

- In 2023
- OpenAI Announced Function Calling
- Many people predicated that

- Function Calling will conquer the world
- Developers make only functions
- AI will do everything else



Failure of Function Calling

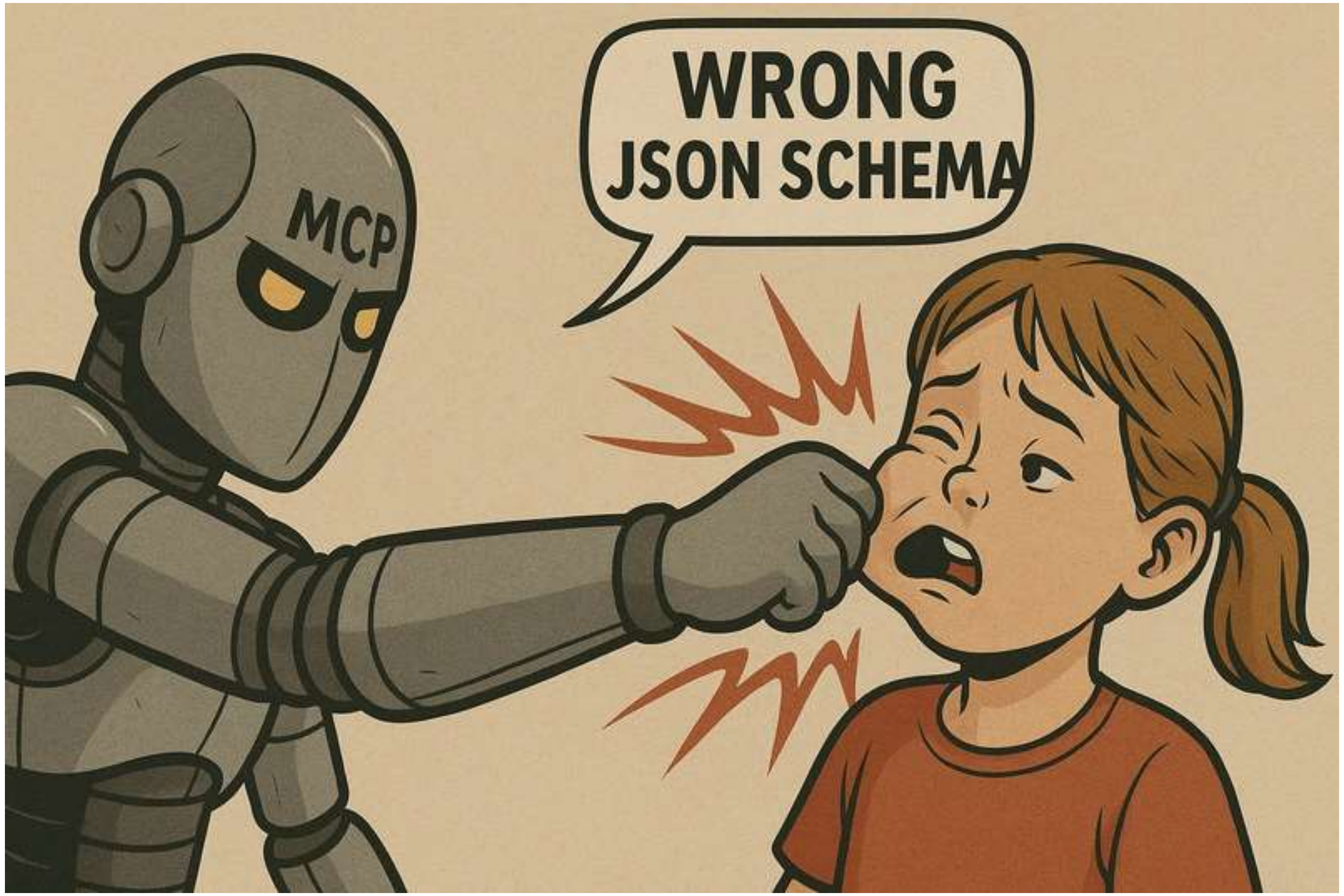
- However, in actually
- Workflow conquered the AI ecosystem

- Not general purposed
- Not Easy/Flexible/Scalable

JSON Schema

- One of the reason of failure
- Comes from JSON schema

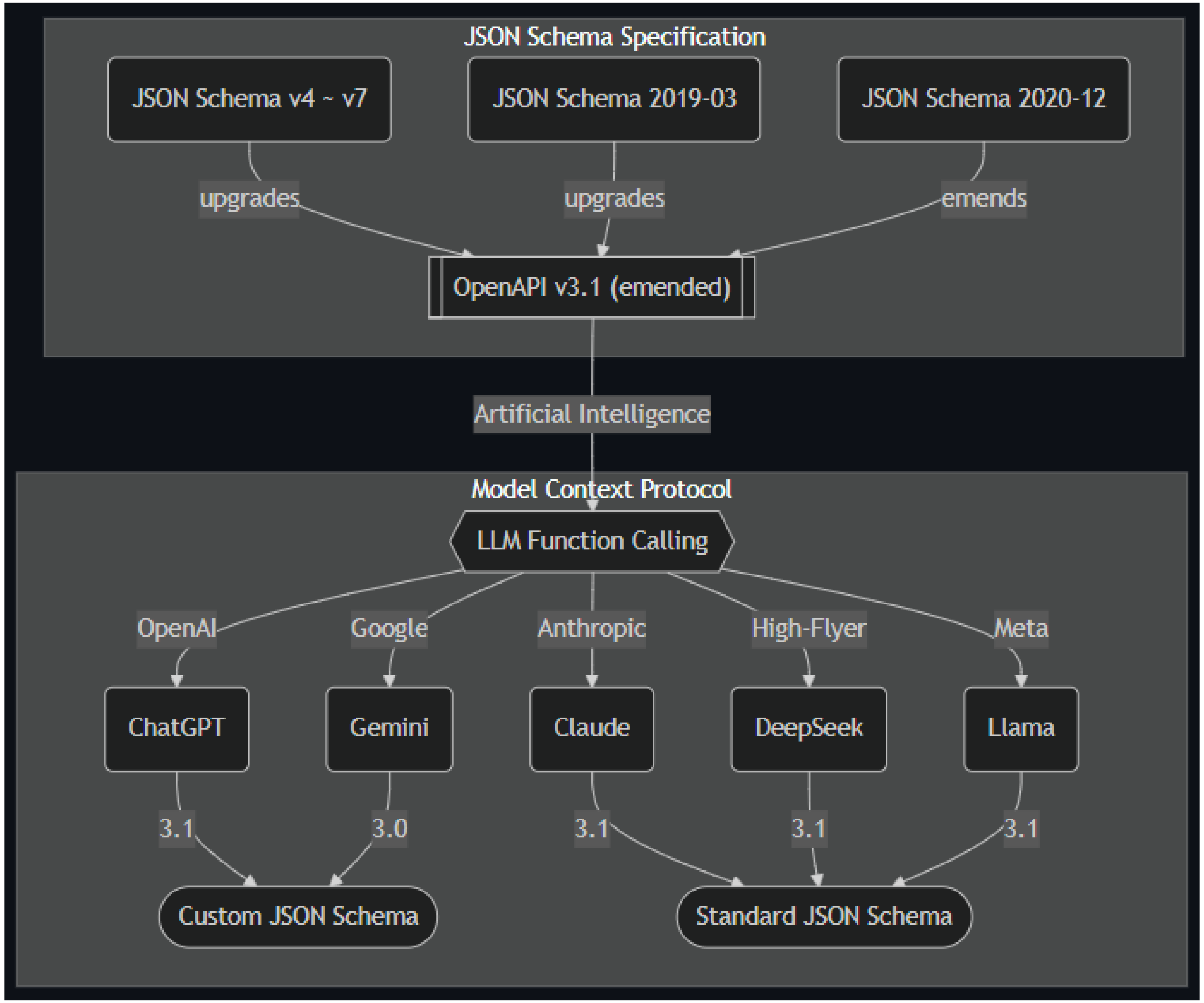
- Too much complex
- Too much ambiguous
- Too many specifications



JSON Schema

- JSON schema specifications
- Different across LLM vendors

- OpenAI does not follow
- Gemini does not follow
- Standard JSON schema spec



JSON Schema Specification

JSON Schema v4 ~ v7
JSON Schema 2019-03
JSON Schema 2020-12

upgrades
upgrades
emends

OpenAPI v3.1 (emended)

Artificial Intelligence

Model Context Protocol

LLM Function Calling

OpenAI
Google
Anthropic
High-Flyer
Meta

ChatGPT
Gemini
Claude
DeepSeek
Llama

3.1
3.0
3.1
3.1
3.1

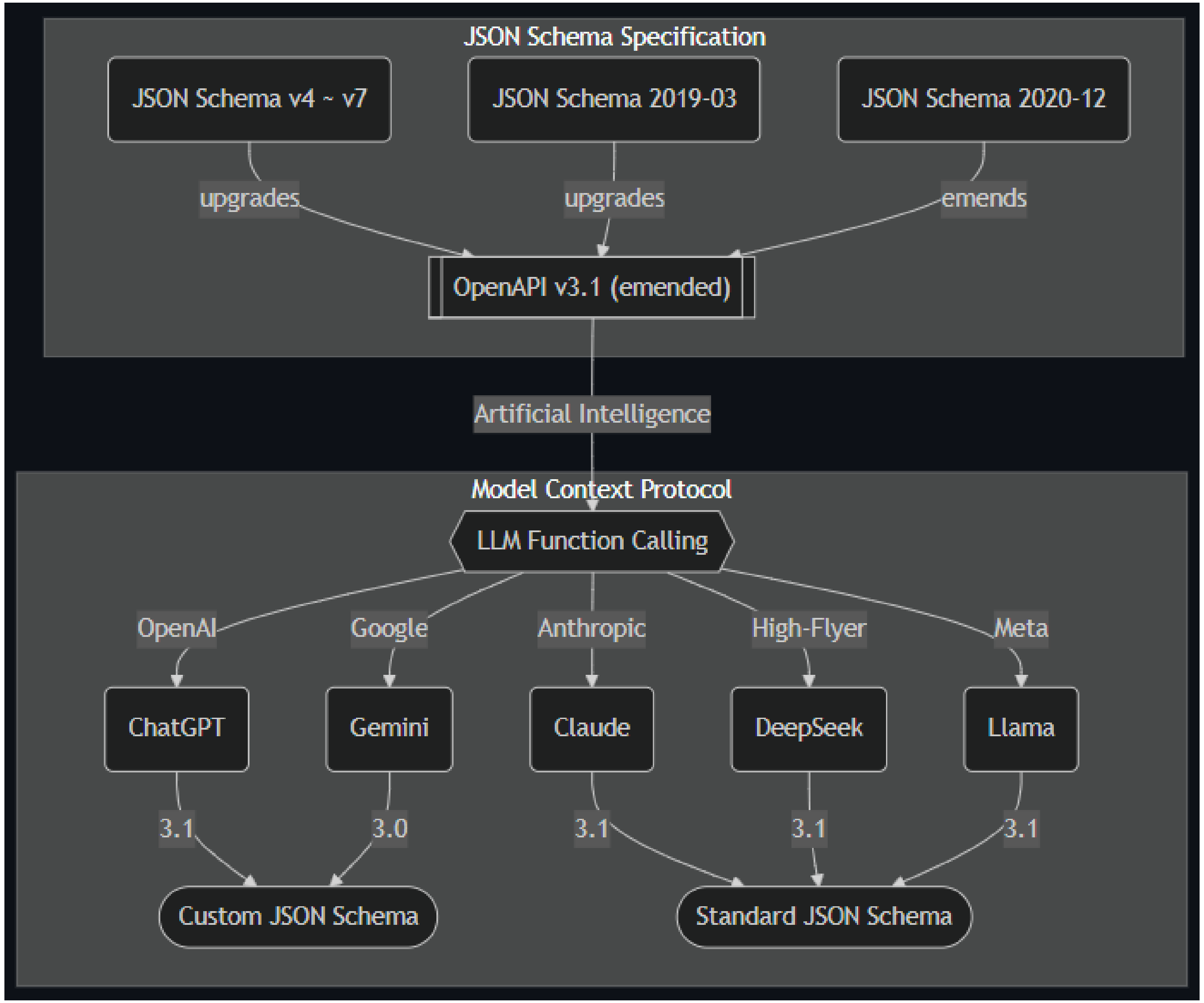
Custom JSON Schema
Standard JSON Schema

JSON Schema

- Claude follows JSON Schema 2020-12
- MCP SDK follows JSON schema v7

- No problem in Claude
- But not compatible with

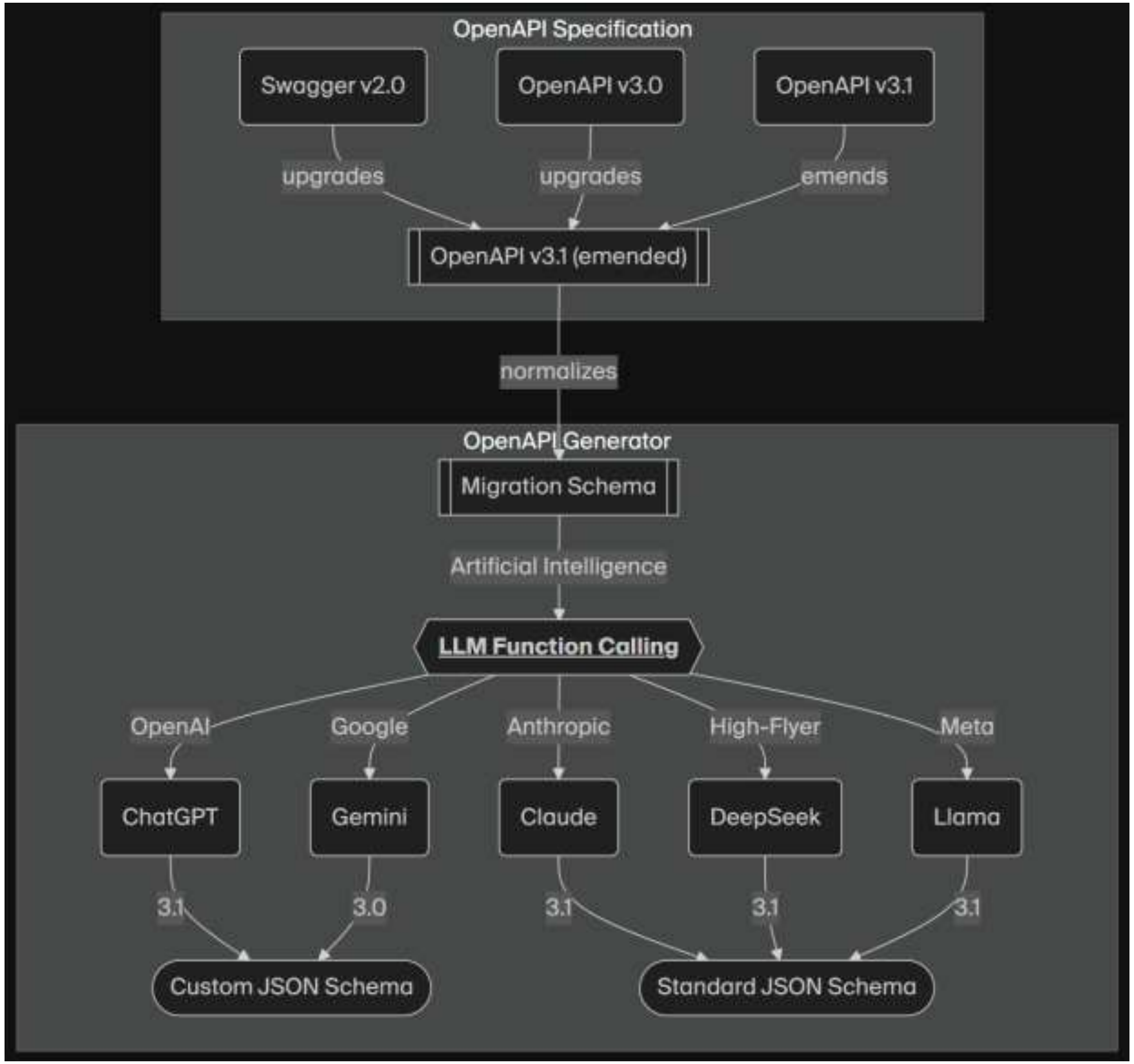
- OpenAI
- Gemini



JSON Schema

- OpenAPI is another world

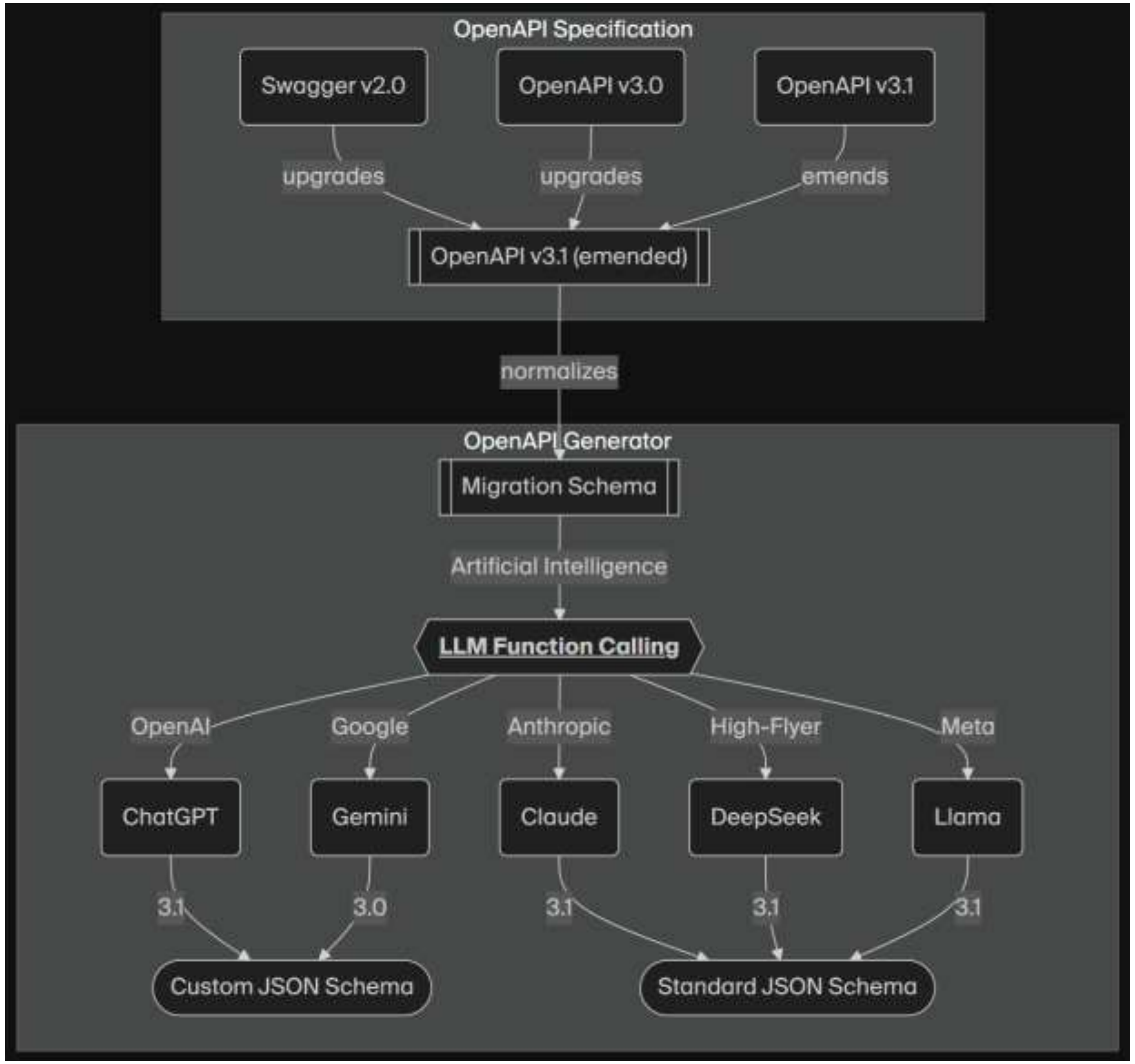
- A lot of versions
- A lot of duplicated expressions
- A little bit different with
  Standard JSON schema



JSON Schema

- Agentica overcomes
- By JSON schema converter

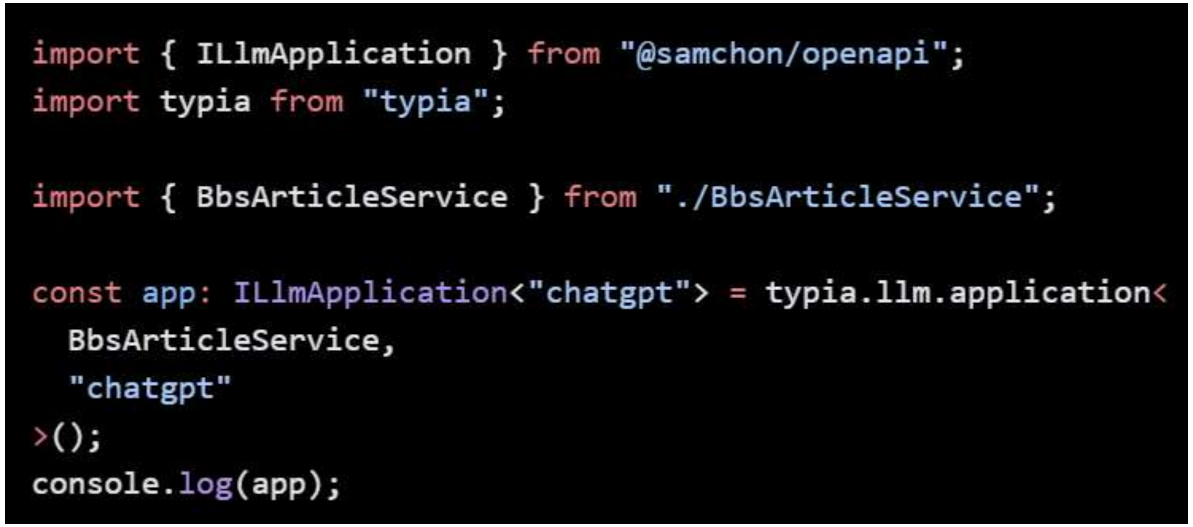
- Convert to a specific model
- bypassing OpenAPI v3.1



Compiler Driven Development

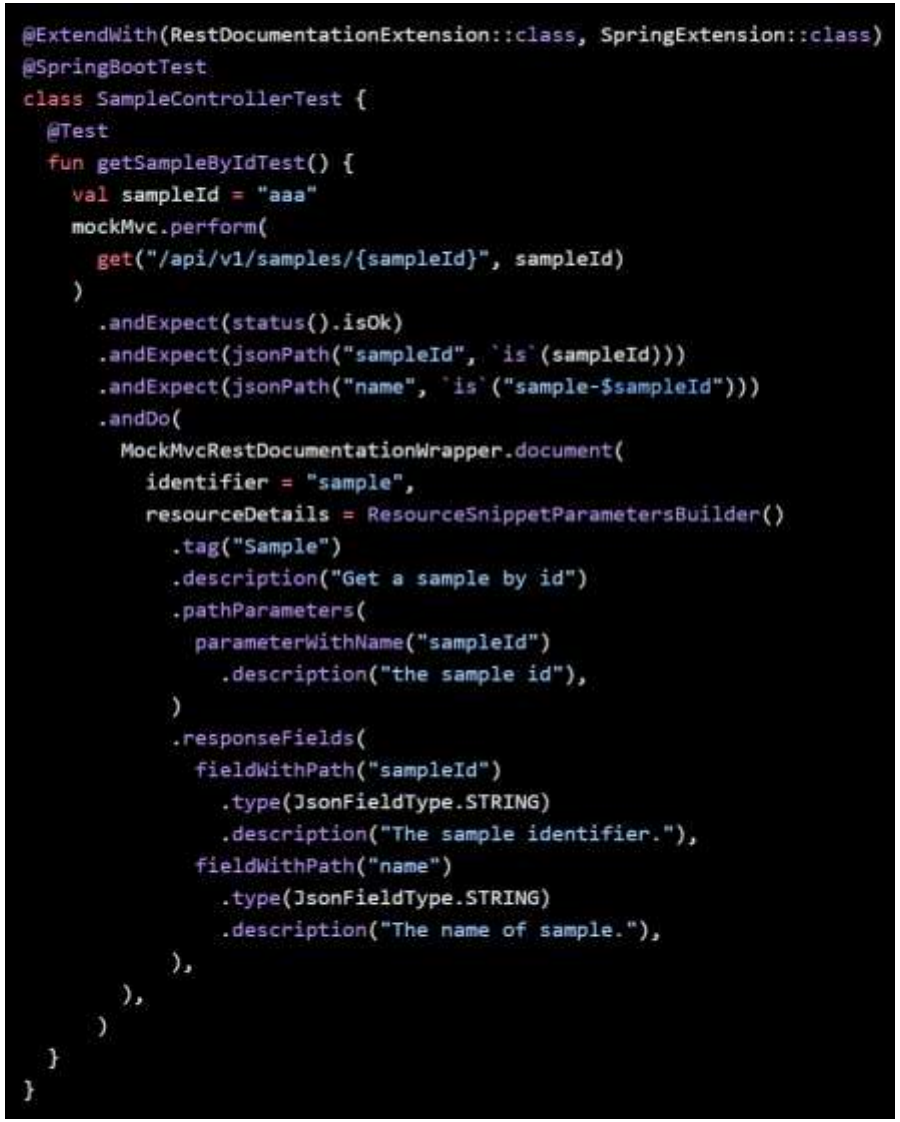
- The next reason of
- AI Function Calling Failure

- Absence of
- Compiler level supporting
- For AI Function Schema Crafting



```
import { ILlmApplication } from "@samchon/openapi";
import typia from "typia";

import { BbsArticleService } from "./BbsArticleService";

const app: ILlmApplication<"chatgpt"> = typia.llm.application<
  BbsArticleService,
  "chatgpt"
>();
console.log(app);
```

💻 *Playground Link*



```
@ExtendWith(RestDocumentationExtension::class, SpringExtension::class)
@SpringBootTest
class SampleControllerTest {
  @Test
  fun getSampleByIdTest() {
    val sampleId = "aaa"
    mockMvc.perform(
      get("/api/v1/samples/{sampleId}", sampleId)
    )
      .andExpect(status().isOk)
      .andExpect(jsonPath("sampleId", `is`(sampleId)))
      .andExpect(jsonPath("name", `is`("sample-$sampleId")))
      .andDo(
        MockMvcRestDocumentationWrapper.document(
          identifier = "sample",
          resourceDetails = ResourceSnippetParametersBuilder()
            .tag("Sample")
            .description("Get a sample by id")
            .pathParameters(
              parameterWithName("sampleId")
                .description("the sample id"),
            )
            .responseFields(
              fieldWithPath("sampleId")
                .type(JsonFieldType.STRING)
                .description("The sample identifier."),
              fieldWithPath("name")
                .type(JsonFieldType.STRING)
                .description("The name of sample."),
            ),
        ),
      )
  }
}
```

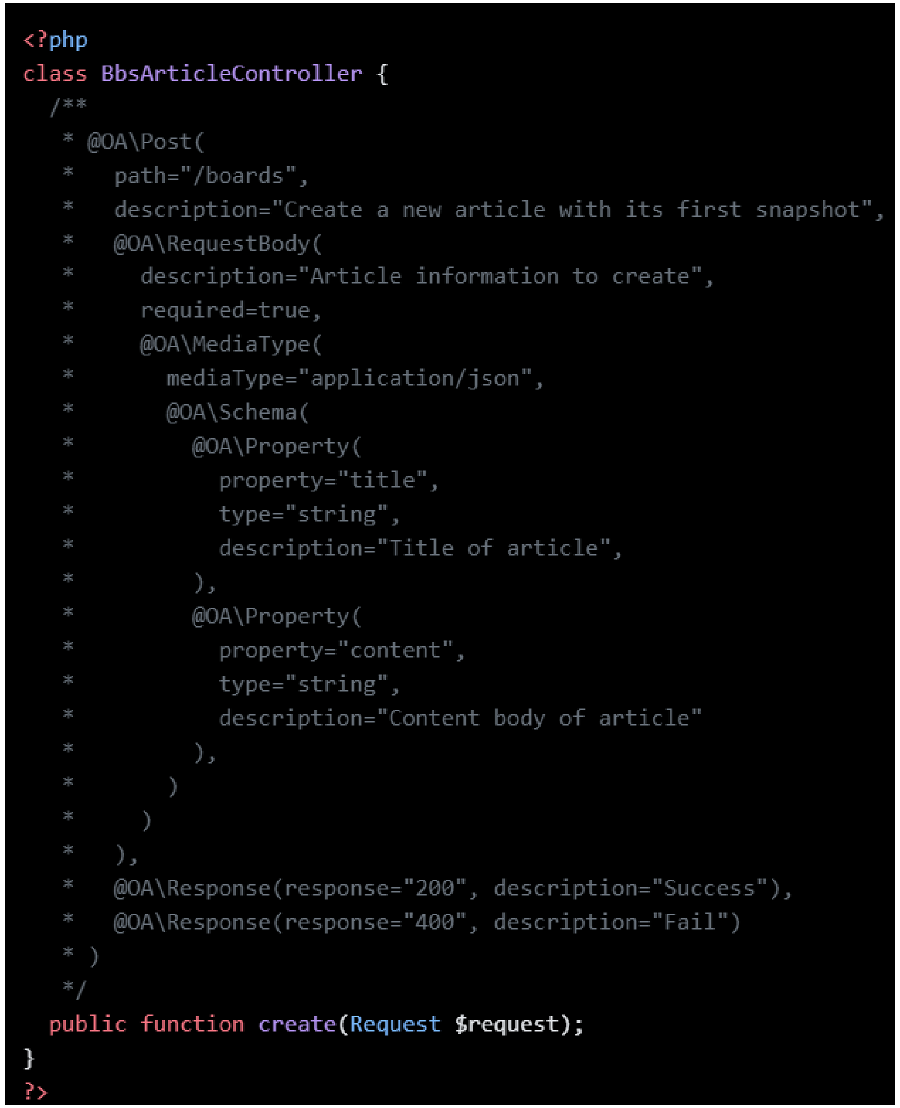
Compiler Driven Development

- Even in the OpenAPI Ecosystem
- Many BE developers
- Hand-craft JSON schema

- Don’t know how to transform
- OpenAPI schema to AI schema

- How they could accomplish
- AI Function Calling?

Compiler Driven Development



```
<?php
class BbsArticleController {
  /**
   * @OA\Post(
   *   path="/boards",
   *   description="Create a new article with its first snapshot",
   *   @OA\RequestBody(
   *     description="Article information to create",
   *     required=true,
   *     @OA\MediaType(
   *       mediaType="application/json",
   *       @OA\Schema(
   *         @OA\Property(
   *           property="title",
   *           type="string",
   *           description="Title of article",
   *         ),
   *         @OA\Property(
   *           property="content",
   *           type="string",
   *           description="Content body of article"
   *         ),
   *       )
   *     )
   *   ),
   *   @OA\Response(response="200", description="Success"),
   *   @OA\Response(response="400", description="Fail")
   * )
   */
  public function create(Request $request);
}
?>
```

- JSON schema specifications
- Different between OpenAPI versions
- Different between AI vendors

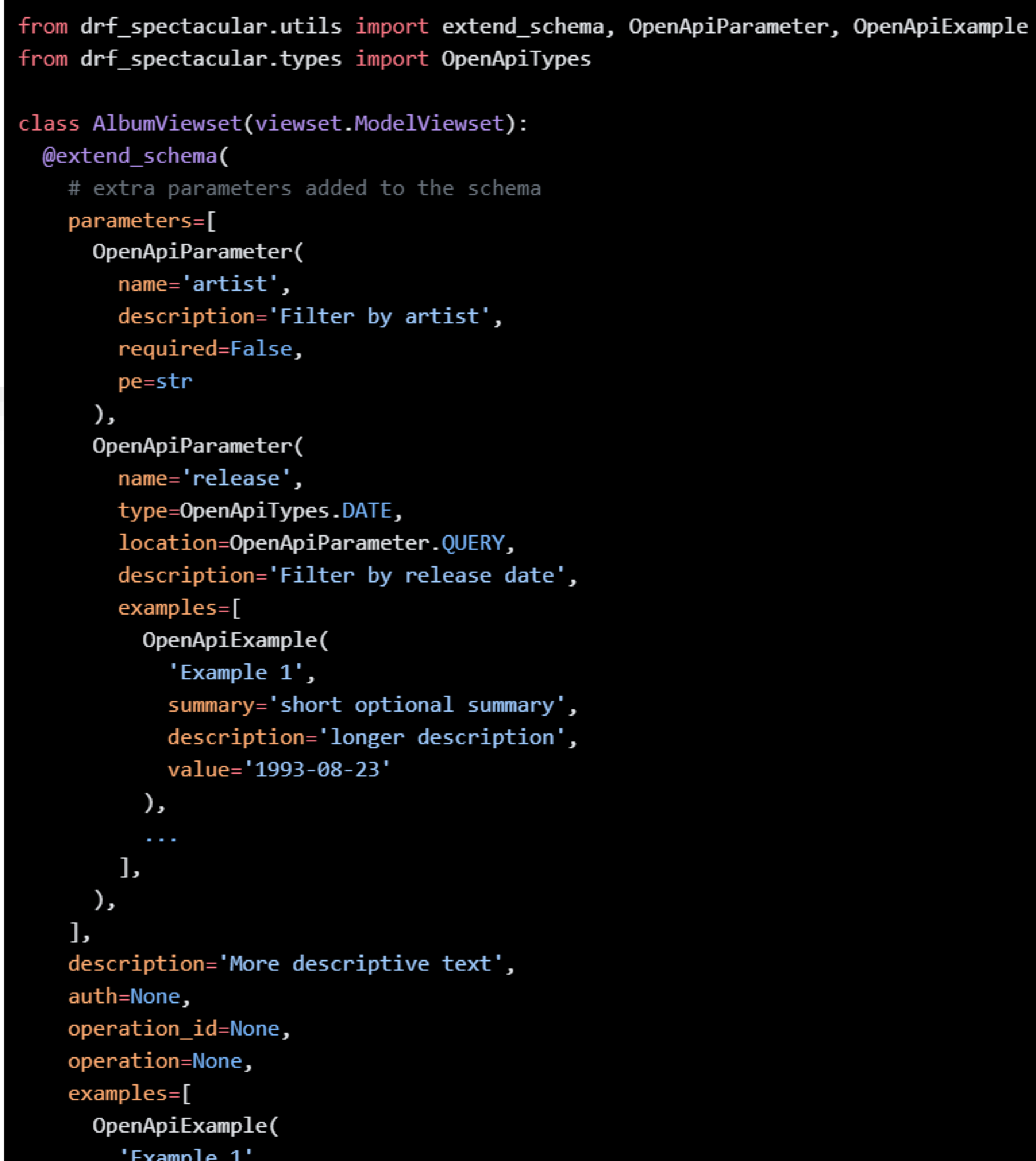
- Furthermore
- Many BE developers
- Hand crafted JSON schema

Compiler Driven Development

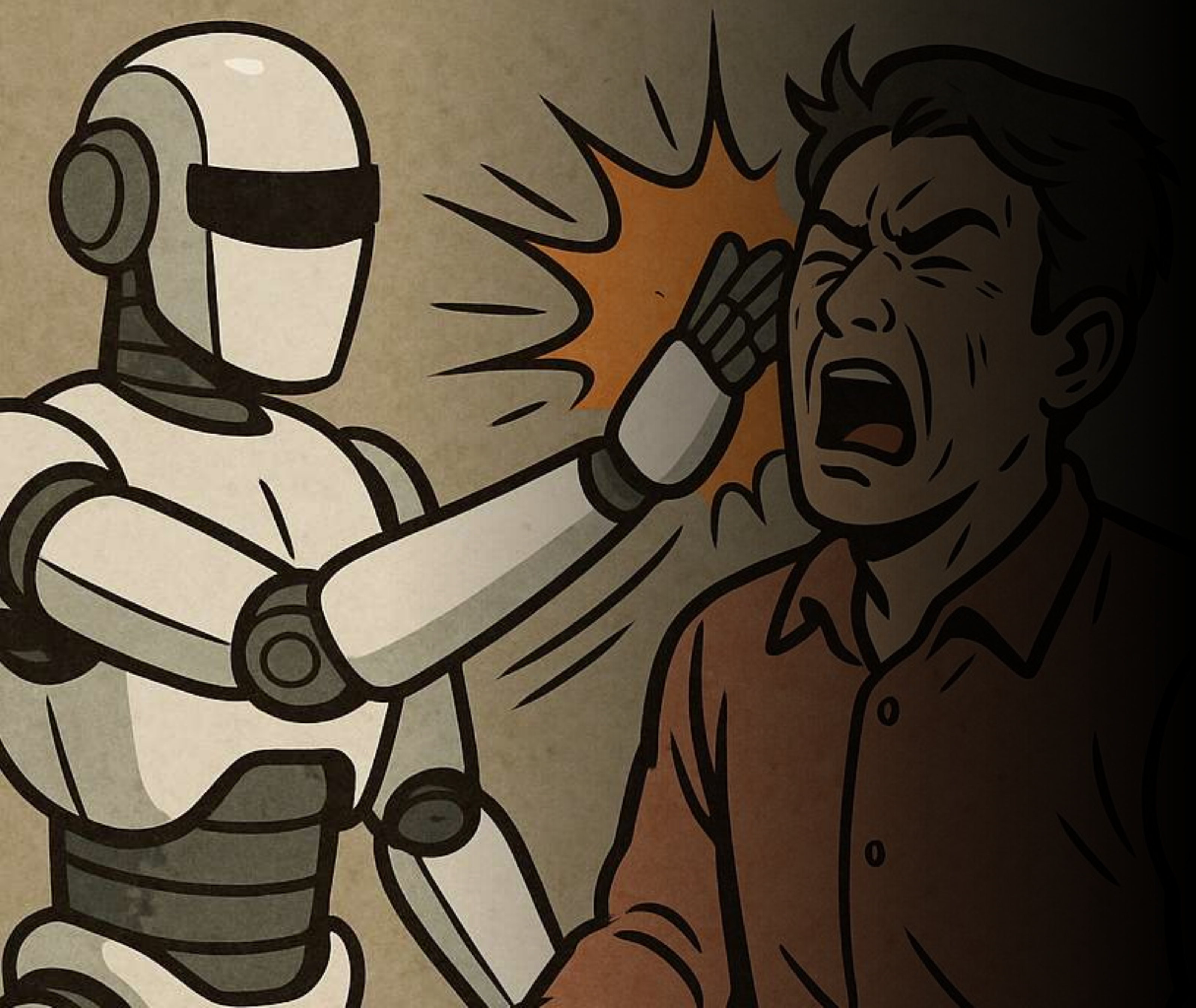
- Shopping Mall Backend
  - Source Code: 37,752 LOC
  - AI Schema LOC: 212,069 LOC

- Hand-written AI schema
- Much huger than source code

- Any mistake on hand-writing
- Breaks entire AI application
- It was the reason why...



```
from drf_spectacular.utils import extend_schema, OpenApiParameter, OpenApiExample
from drf_spectacular.types import OpenApiTypes

class AlbumViewset(viewset.ModelViewset):
  @extend_schema(
    # extra parameters added to the schema
    parameters=[
      OpenApiParameter(
        name='artist',
        description='Filter by artist',
        required=False,
        pe=str
      ),
      OpenApiParameter(
        name='release',
        type=OpenApiTypes.DATE,
        location=OpenApiParameter.QUERY,
        description='Filter by release date',
        examples=[
          OpenApiExample(
            'Example 1',
            summary='short optional summary',
            description='longer description',
            value='1993-08-23'
          ),
          ...
        ],
      ),
    ],
    description='More descriptive text',
    auth=None,
    operation_id=None,
    operation=None,
    examples=[
      OpenApiExample(
        'Example 1',
```

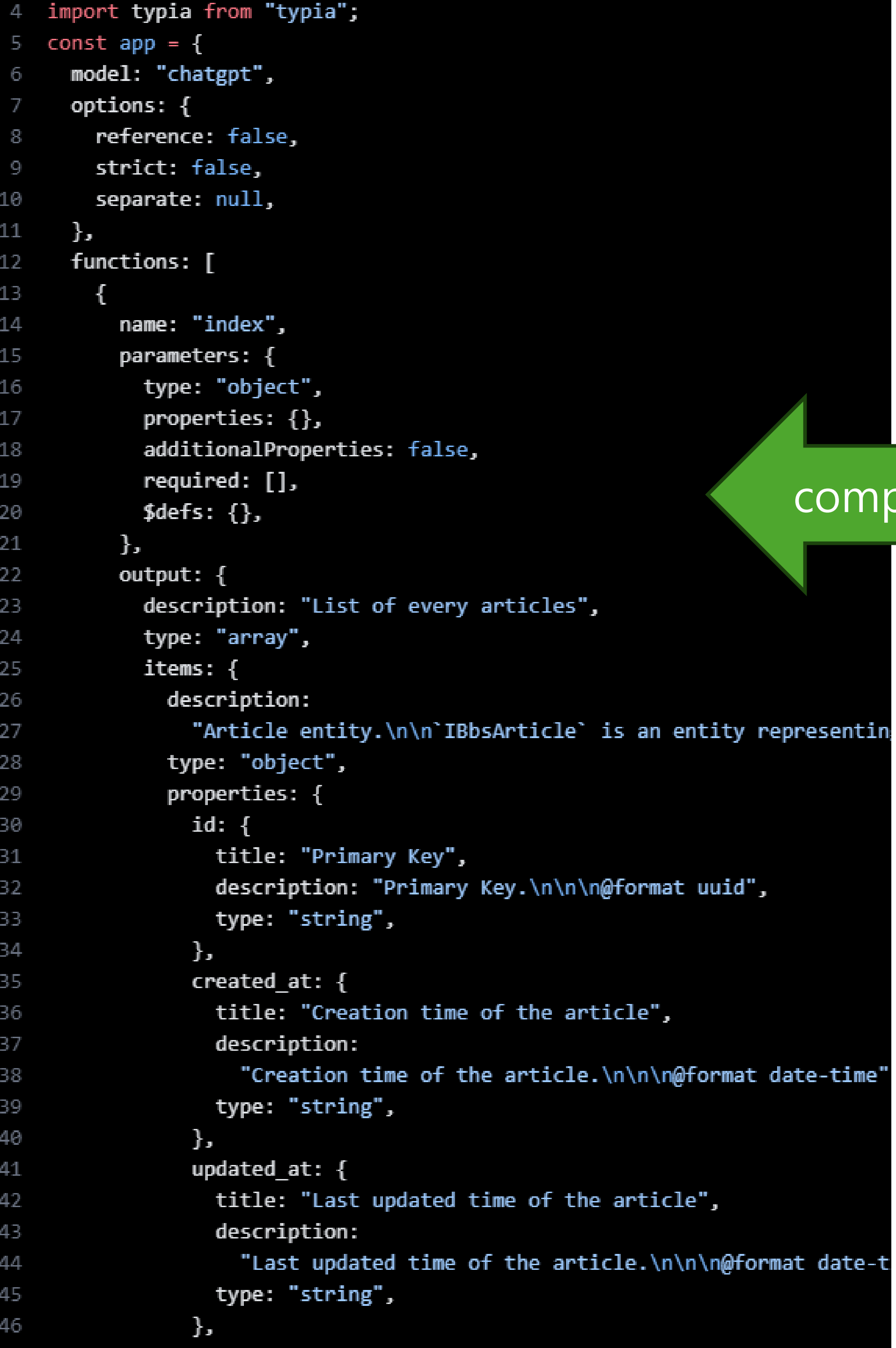


Compiler Driven Development

- Any mistake on
- Human made Schema

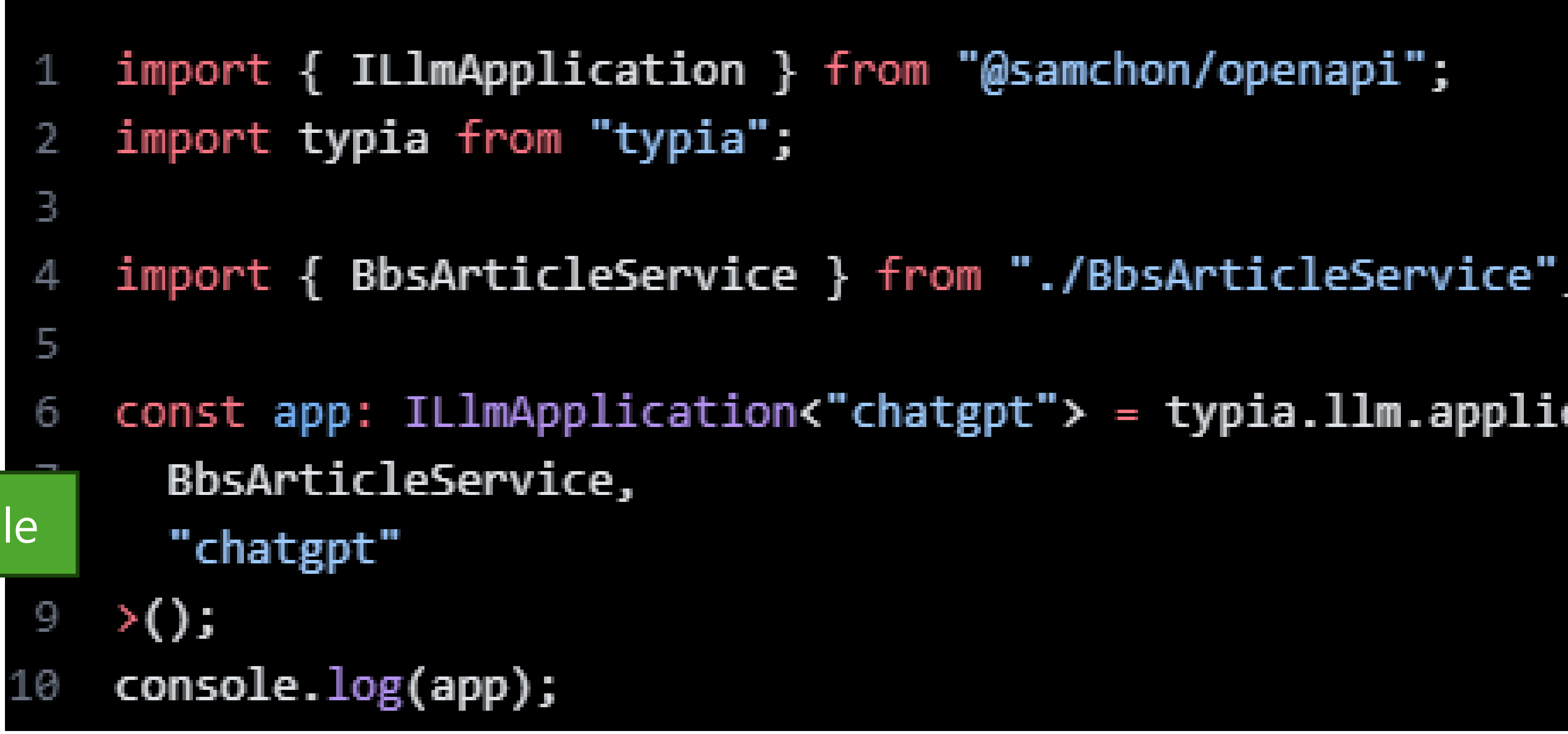
- Frontend developers forgiven
- But AI never forgives it
- Breaks entire AI application

- AI function schema
- Must be built by compiler
- We WrtnLabs did it



```
import typia from "typia";
const app = {
  model: "chatgpt",
  options: {
    reference: false,
    strict: false,
    separate: null,
  },
  functions: [
    {
      name: "index",
      parameters: {
        type: "object",
        properties: {},
        additionalProperties: false,
        required: [],
        $defs: {},
      },
      output: {
        description: "List of every articles",
        type: "array",
        items: {
          description:
            "Article entity.\n\n`IBbsArticle` is an entity representin
          type: "object",
          properties: {
            id: {
              title: "Primary Key",
              description: "Primary Key.\n\n\n@format uuid",
              type: "string",
            },
            created_at: {
              title: "Creation time of the article",
              description:
                "Creation time of the article.\n\n\n@format date-time"
              type: "string",
            },
            updated_at: {
              title: "Last updated time of the article",
              description:
                "Last updated time of the article.\n\n\n@format date-t
              type: "string",
            },
```

compile



```
import { ILlmApplication } from "@samchon/openapi";
import typia from "typia";

import { BbsArticleService } from "./BbsArticleService"

const app: ILlmApplication<"chatgpt"> = typia.llm.appli
  BbsArticleService,
  "chatgpt"
>();
console.log(app);
```

Compiler Driven Development

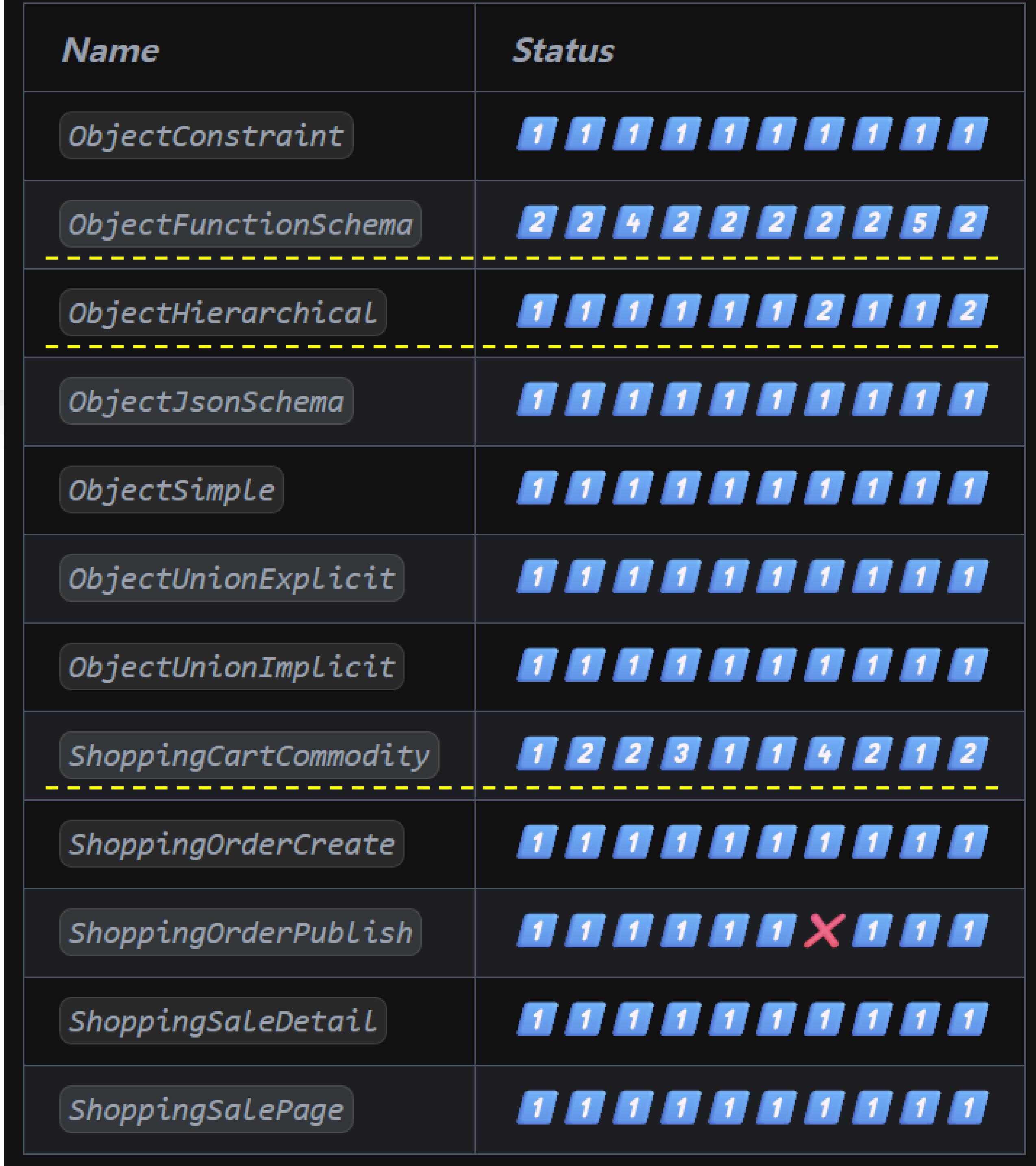
AI function schema
Must be crafted by
Compiler

No duplicated code
No hand-writing

Validation Feedback

- AI Function Calling sometimes
- Make wrong typed
- Parameter values

- When wrong typed value comes
- Entire AI application be broken

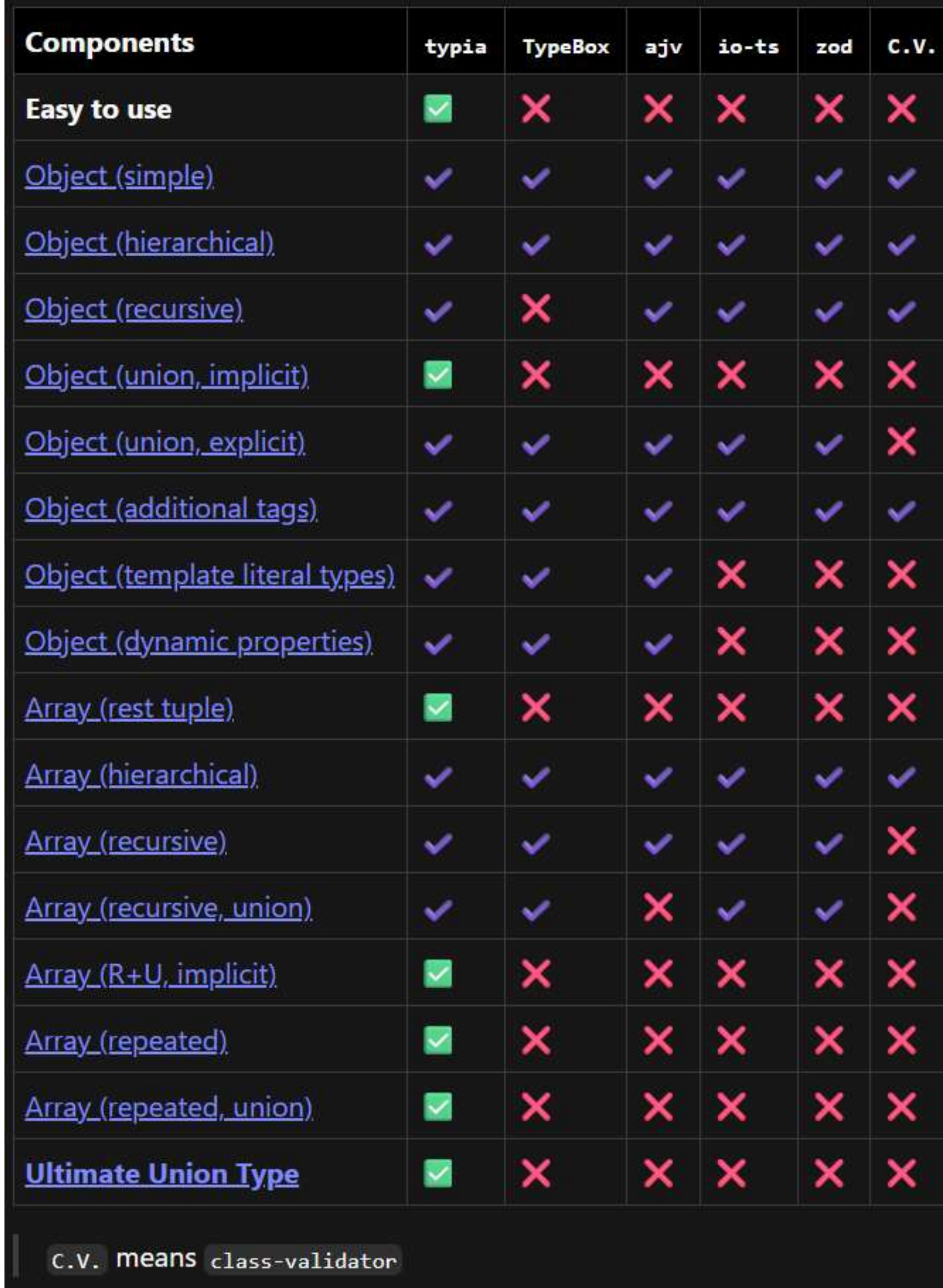


Name	Status									
`ObjectConstraint`	1	1	1	1	1	1	1	1	1	1
`ObjectFunctionSchema`	2	2	4	2	2	2	2	2	5	2
`ObjectHierarchical`	1	1	1	1	1	1	2	1	1	2
`ObjectJsonSchema`	1	1	1	1	1	1	1	1	1	1
`ObjectSimple`	1	1	1	1	1	1	1	1	1	1
`ObjectUnionExplicit`	1	1	1	1	1	1	1	1	1	1
`ObjectUnionImplicit`	1	1	1	1	1	1	1	1	1	1
`ShoppingCartCommodity`	1	2	2	3	1	1	4	2	1	2
`ShoppingOrderCreate`	1	1	1	1	1	1	1	1	1	1
`ShoppingOrderPublish`	1	1	1	1	1	1	❌	1	1	1
`ShoppingSaleDetail`	1	1	1	1	1	1	1	1	1	1
`ShoppingSalePage`	1	1	1	1	1	1	1	1	1	1

Validation Feedback

- Agentica corrects type error
- By delivering type error information
- To the AI agent

- The world's most accurate and detailed
- Error reporting via typia.validate<T>()



Components	typia	TypeBox	ajv	io-ts	zod	C.V.
Easy to use	✅	❌	❌	❌	❌	❌
Object (simple)	✔	✔	✔	✔	✔	✔
Object (hierarchical)	✔	✔	✔	✔	✔	✔
Object (recursive)	✔	❌	✔	✔	✔	✔
Object (union, implicit)	✅	❌	❌	❌	❌	❌
Object (union, explicit)	✔	✔	✔	✔	✔	❌
Object (additional tags)	✔	✔	✔	✔	✔	✔
Object (template literal types)	✔	✔	✔	❌	❌	❌
Object (dynamic properties)	✔	✔	✔	❌	❌	❌
Array (rest tuple)	✅	❌	❌	❌	❌	❌
Array (hierarchical)	✔	✔	✔	✔	✔	✔
Array (recursive)	✔	✔	✔	✔	✔	❌
Array (recursive, union)	✔	✔	❌	✔	✔	❌
Array (R+U, implicit)	✅	❌	❌	❌	❌	❌
Array (repeated)	✅	❌	❌	❌	❌	❌
Array (repeated, union)	✅	❌	❌	❌	❌	❌
Ultimate Union Type	✅	❌	❌	❌	❌	❌

> `C.V.` means `class-validator`

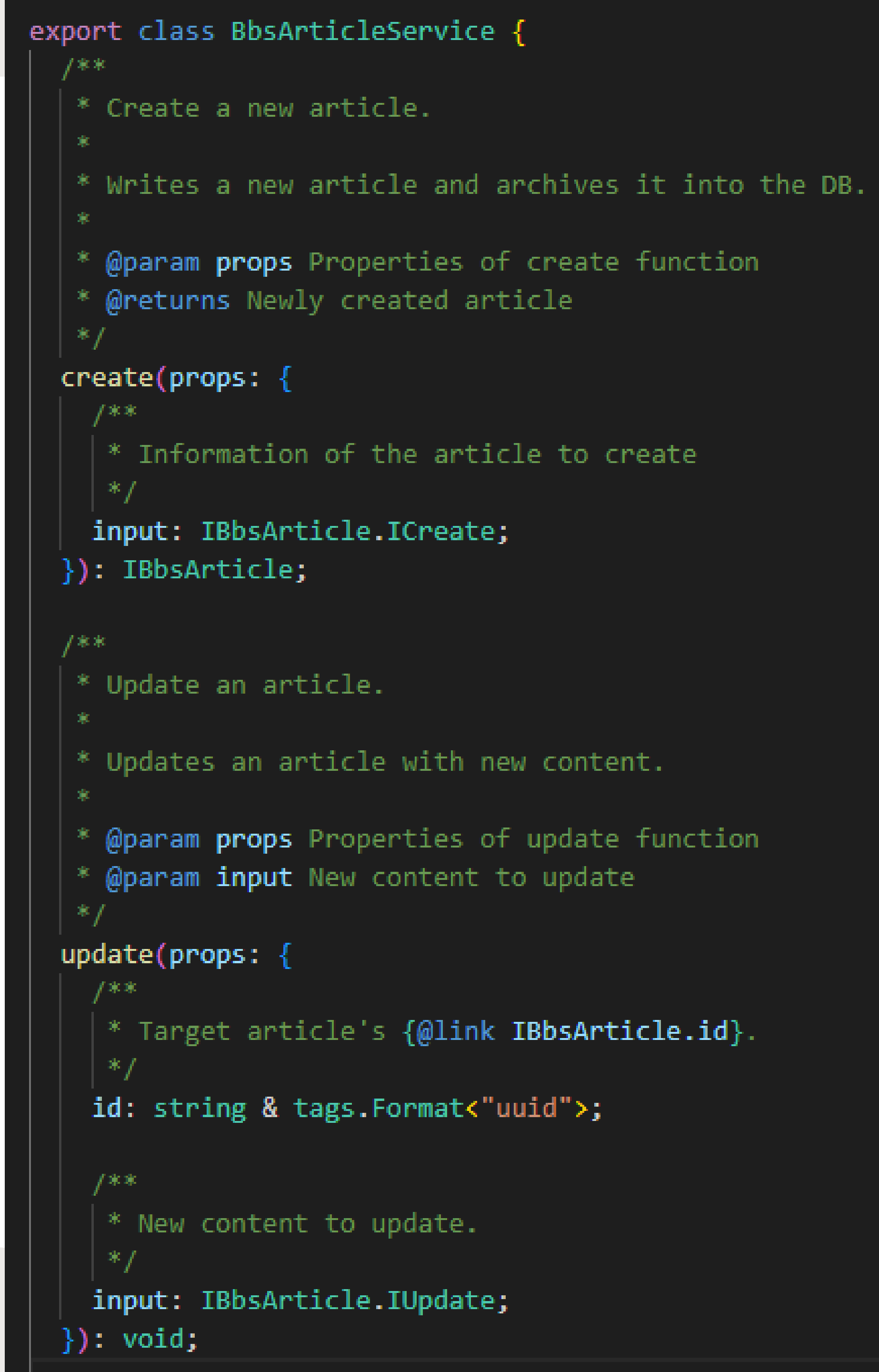
Validation Feedback

- Shopping Mall
- Enterprise features
    - 289 API functions
    - Product, order, payment, refund, review, coupon, deposit, withdrawal, delivery, and so on
- Agent just by a swagger file
- Working on 8b model (4o-mini)

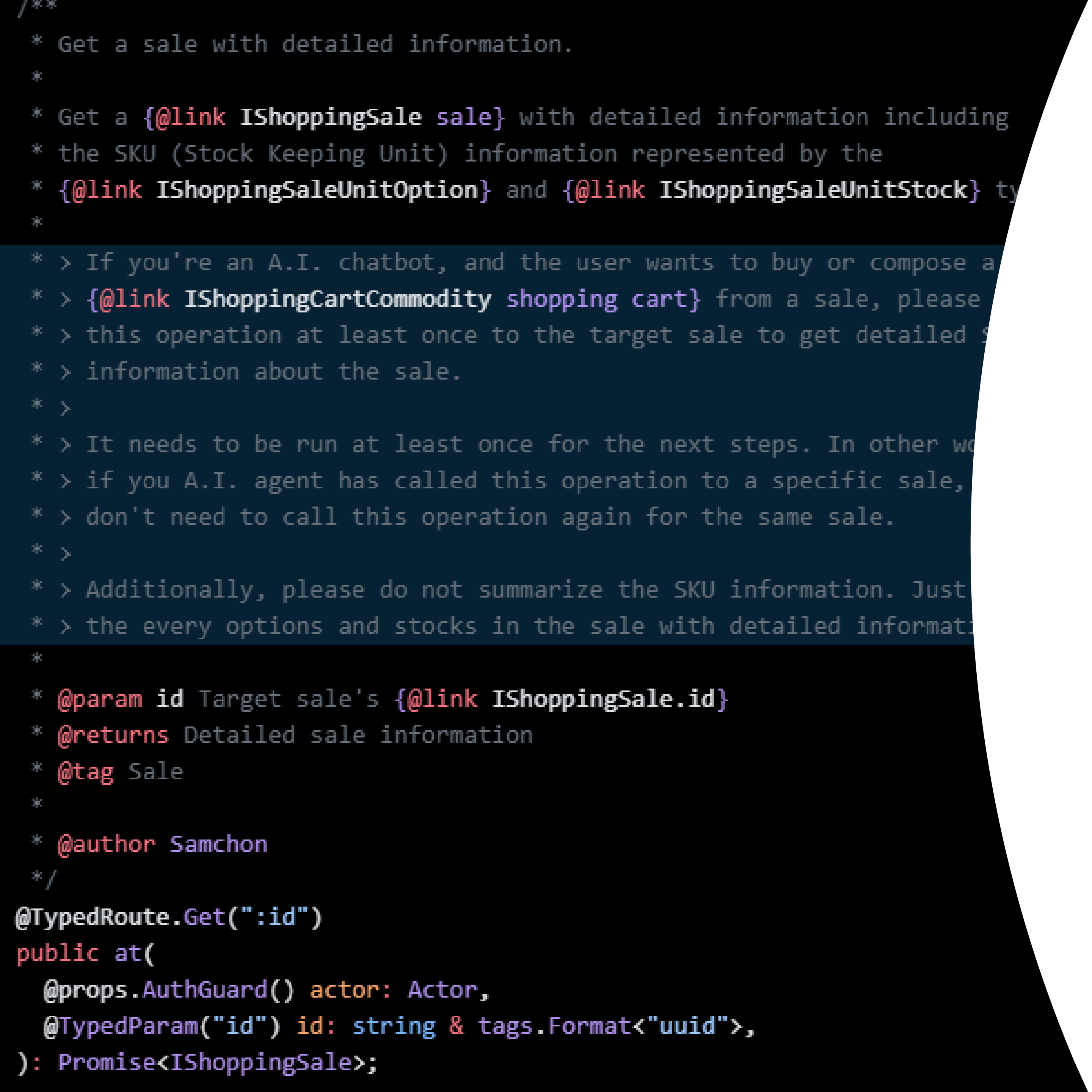
Document Driven Development

- Documentation to each function
- No workflow or graph nodes are required
- Take everything to LLM Function Calling

- Easy
- Scalable
- Flexible
- Mass Productive



```
export class BbsArticleService {
  /**
   * Create a new article.
   *
   * Writes a new article and archives it into the DB.
   *
   * @param props Properties of create function
   * @returns Newly created article
   */
  create(props: {
    /**
     * Information of the article to create
     */
    input: IBbsArticle.ICreate;
  }): IBbsArticle;

  /**
   * Update an article.
   *
   * Updates an article with new content.
   *
   * @param props Properties of update function
   * @param input New content to update
   */
  update(props: {
    /**
     * Target article's {@link IBbsArticle.id}.
     */
    id: string & tags.Format<"uuid">;

    /**
     * New content to update.
     */
    input: IBbsArticle.IUpdate;
  }): void;
```

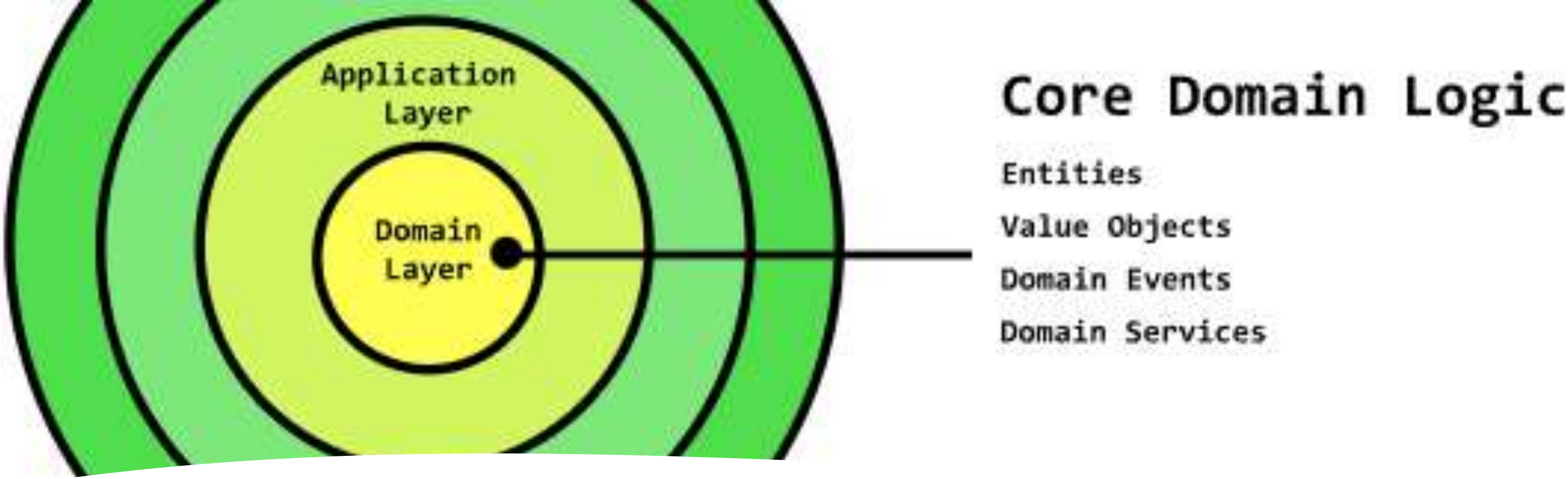


```
/**
 * Get a sale with detailed information.
 *
 * Get a {@link IShoppingSale sale} with detailed information including
 * the SKU (Stock Keeping Unit) information represented by the
 * {@link IShoppingSaleUnitOption} and {@link IShoppingSaleUnitStock} ty
 *
 * > If you're an A.I. chatbot, and the user wants to buy or compose a
 * > {@link IShoppingCartCommodity shopping cart} from a sale, please
 * > this operation at least once to the target sale to get detailed
 * > information about the sale.
 * >
 * > It needs to be run at least once for the next steps. In other wo
 * > if you A.I. agent has called this operation to a specific sale,
 * > don't need to call this operation again for the same sale.
 * >
 * > Additionally, please do not summarize the SKU information. Just
 * > the every options and stocks in the sale with detailed informati
 *
 * @param id Target sale's {@link IShoppingSale.id}
 * @returns Detailed sale information
 * @tag Sale
 *
 * @author Samchon
 */
@TypedRoute.Get(":id")
public at(
  @props.AuthGuard() actor: Actor,
  @TypedParam("id") id: string & tags.Format<"uuid">,
): Promise<IShoppingSale>;
```

Document Driven Development

- If there is a relationship between functions
- Write it on the description

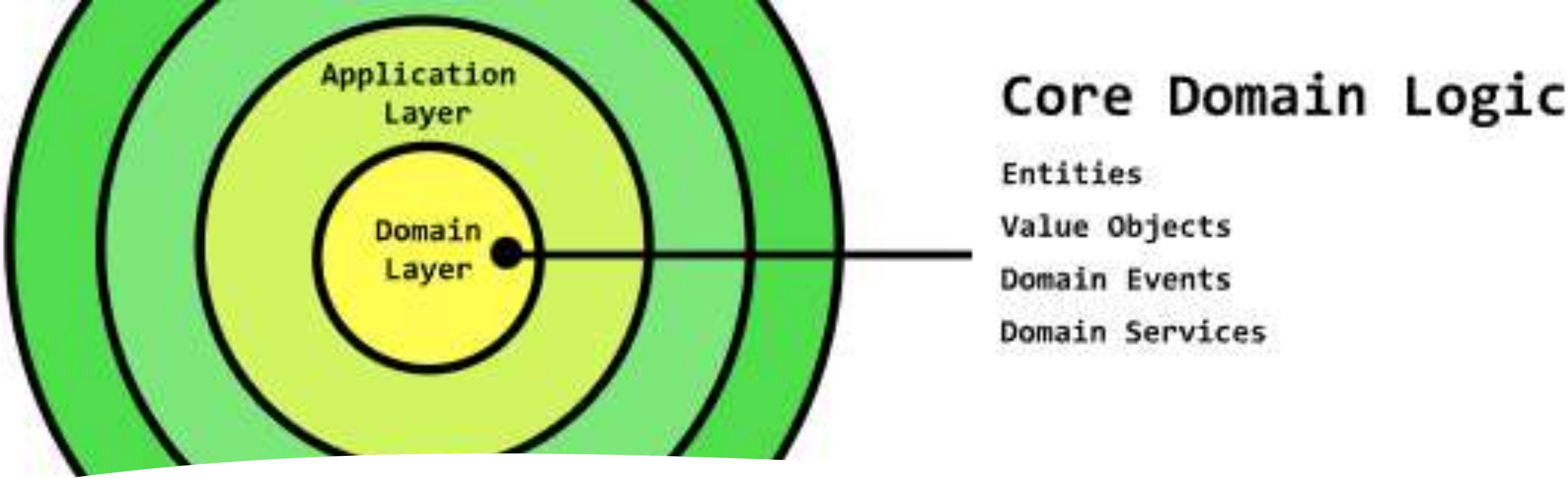
- LLM Function Calling
- It will do everything



Document Driven Development

- Describing functions independently
- So that separating complicate project to small domains

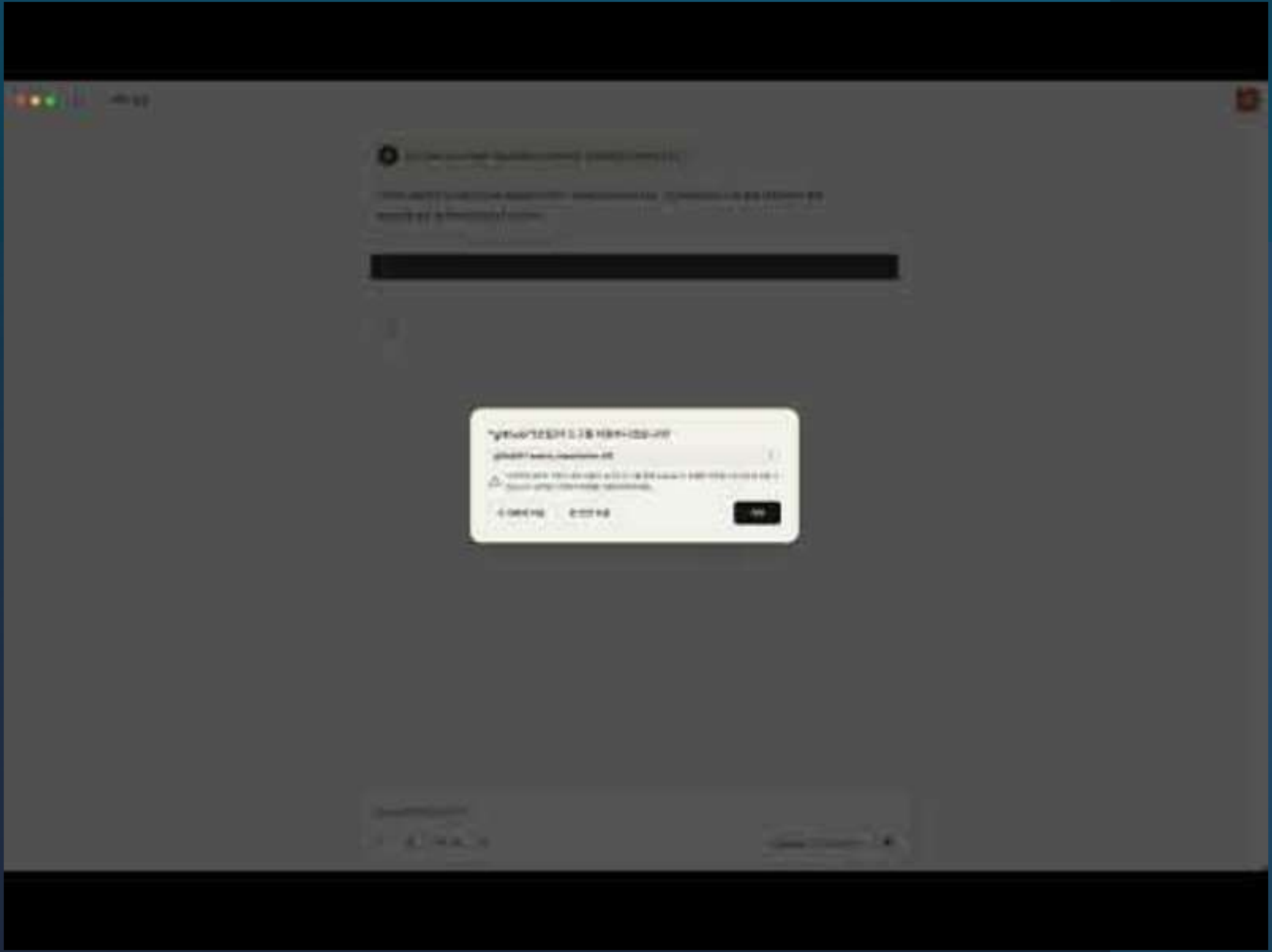
- Same with “Doman Driven Development” methodology
- Just a pun expressions in agent view

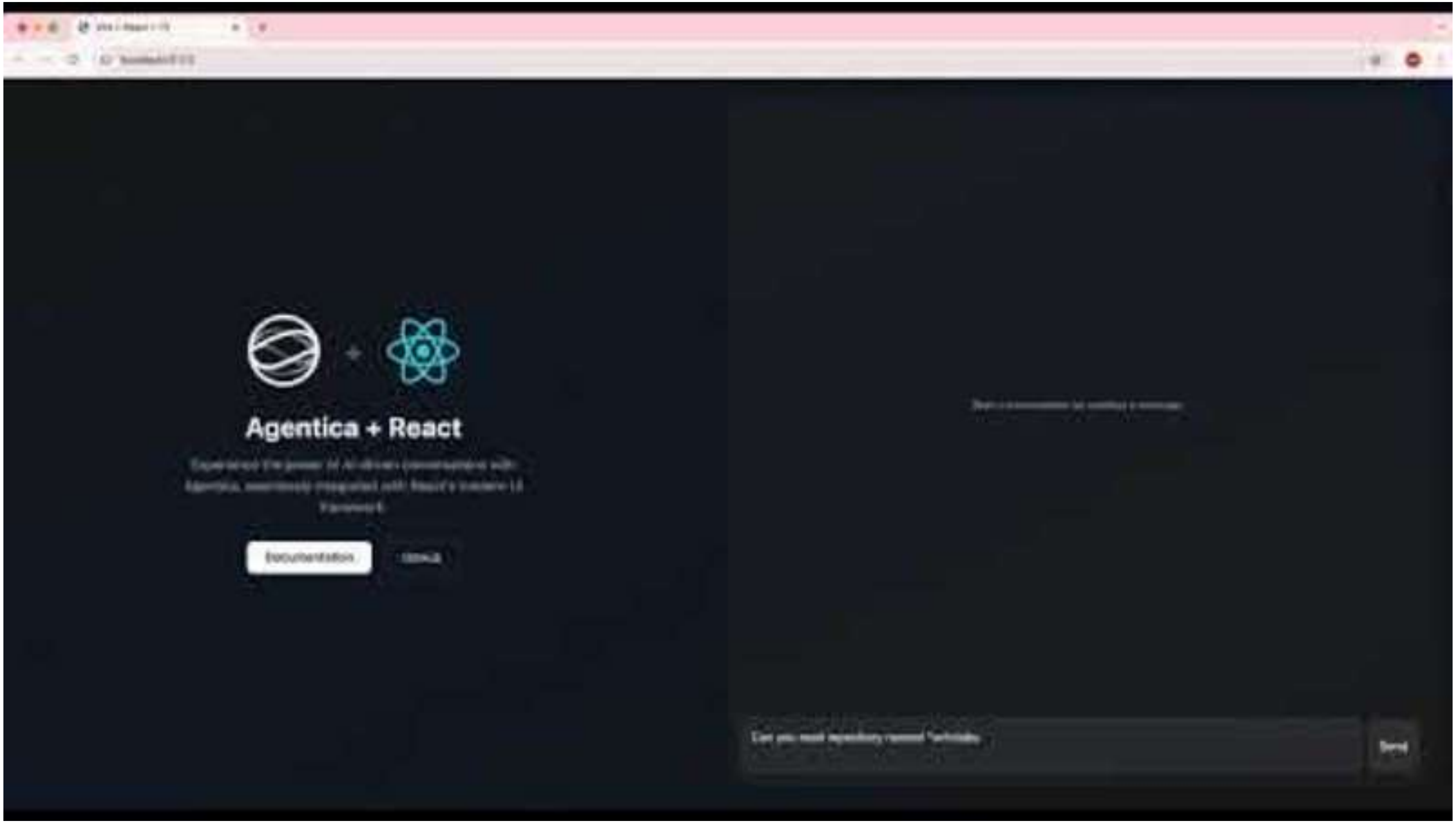


Document Driven Development

- Describing functions independently
- So that separating complicate project to small domains

- It makes Agent scalable, flexible and mass productive
- And accomplishes the Agentic AI




Agentica + React
Documentation
Send

AutoBE

Backend Automation Tool

Spiral Waterfall Model

Compiler Feedback

Backend Automation

NestJS + Prisma

- AI Viral Coding Chatbot
- Generating Backend Program

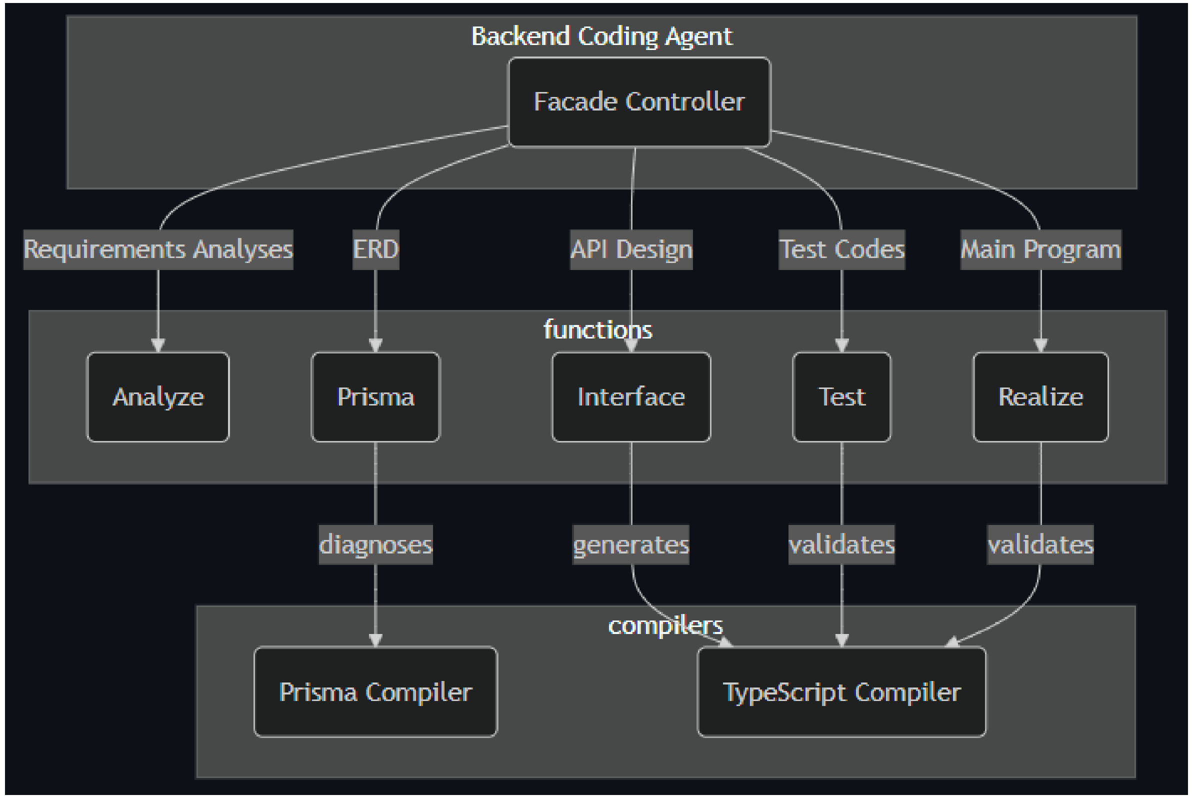
- With Spiral Waterfall Model
- Enhanced by Compiler Feedback
- Completed by Detailed Documentation

Spiral Waterfall Model

- Backend Coding Agent
- Cycling Waterfall Steps

- Developed by Agentica
- AI Function Calling

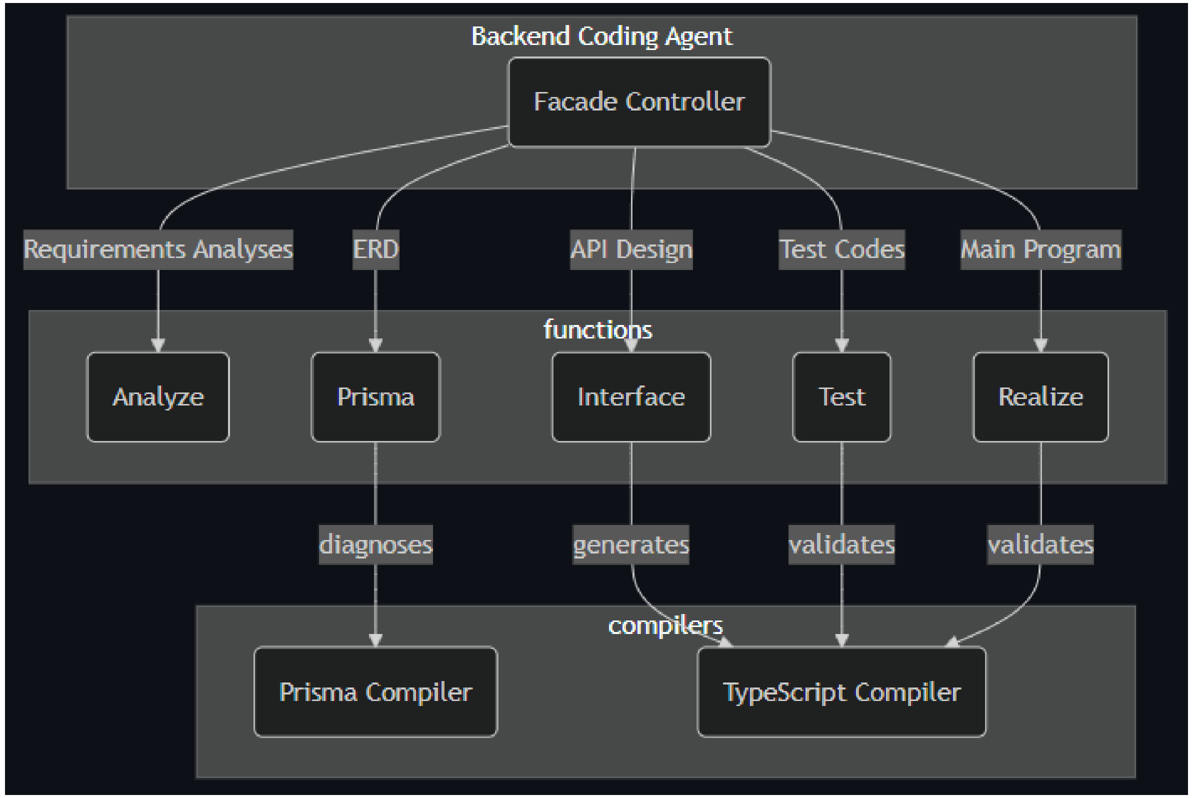
- Each Waterfall steps are
- Functional agent orchestrating



Spiral Waterfall Model

- Analyze Agent

- Human and AI debates requirements
- AI publishes an analyzed report
- Review agent helps detailing

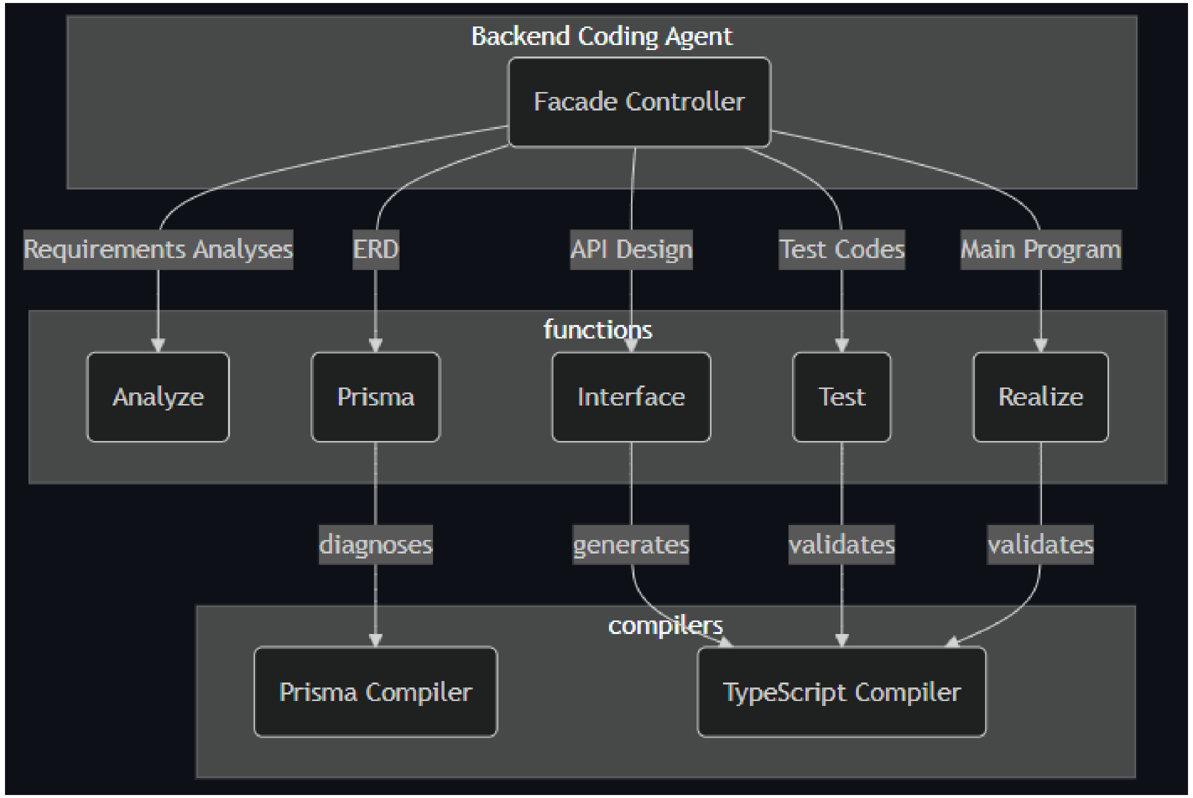


Spiral Waterfall Model

- Prisma Agent

- Transform Requirement Analyses
- To Prisma Schema Documents

- Validated by Prisma Compiler
- Reviewer agent helps detailing
- Fully Documented

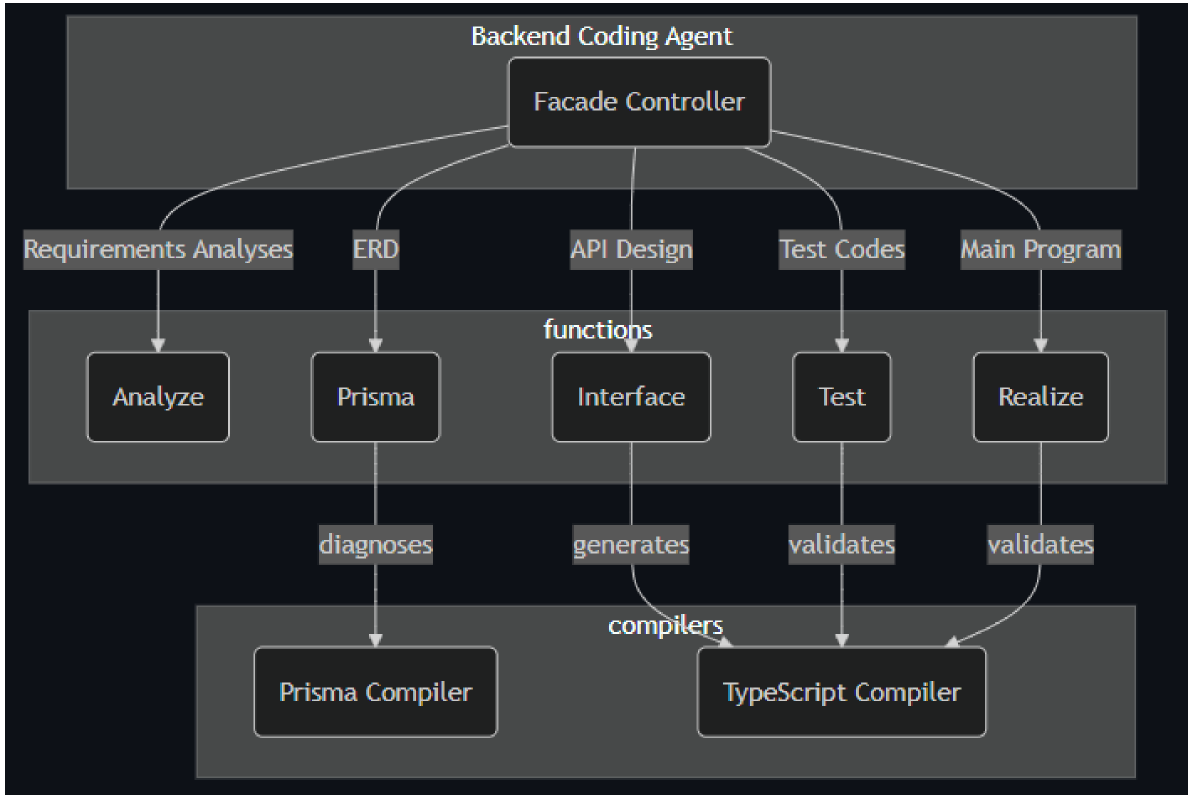


Spiral Waterfall Model

- Interface Agent

- Makes OpenAPI Document
- Transform OpenAPI Document
- To NestJS controllers, DTO interfaces

- Fully Documented

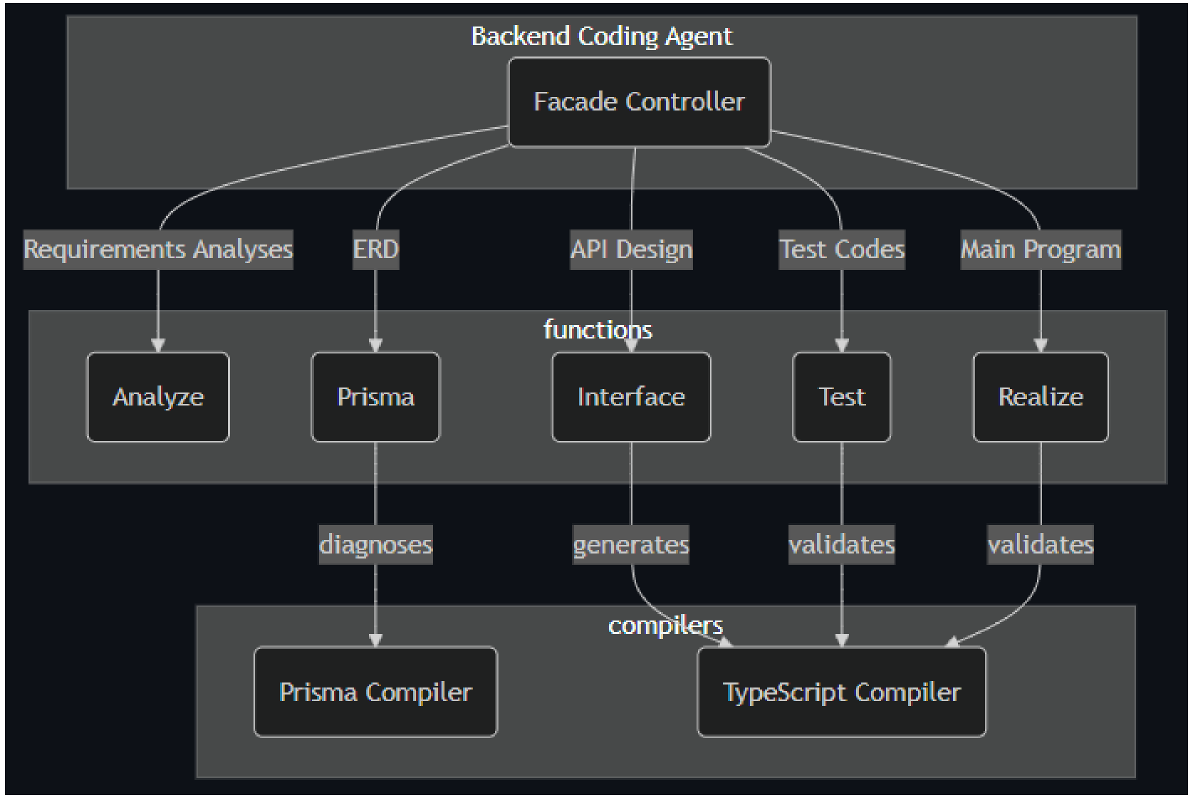


Spiral Waterfall Model

- Test Agent

- Make e2e test functions
- About RestfulAPI operations
- From nestia e2e generated functions

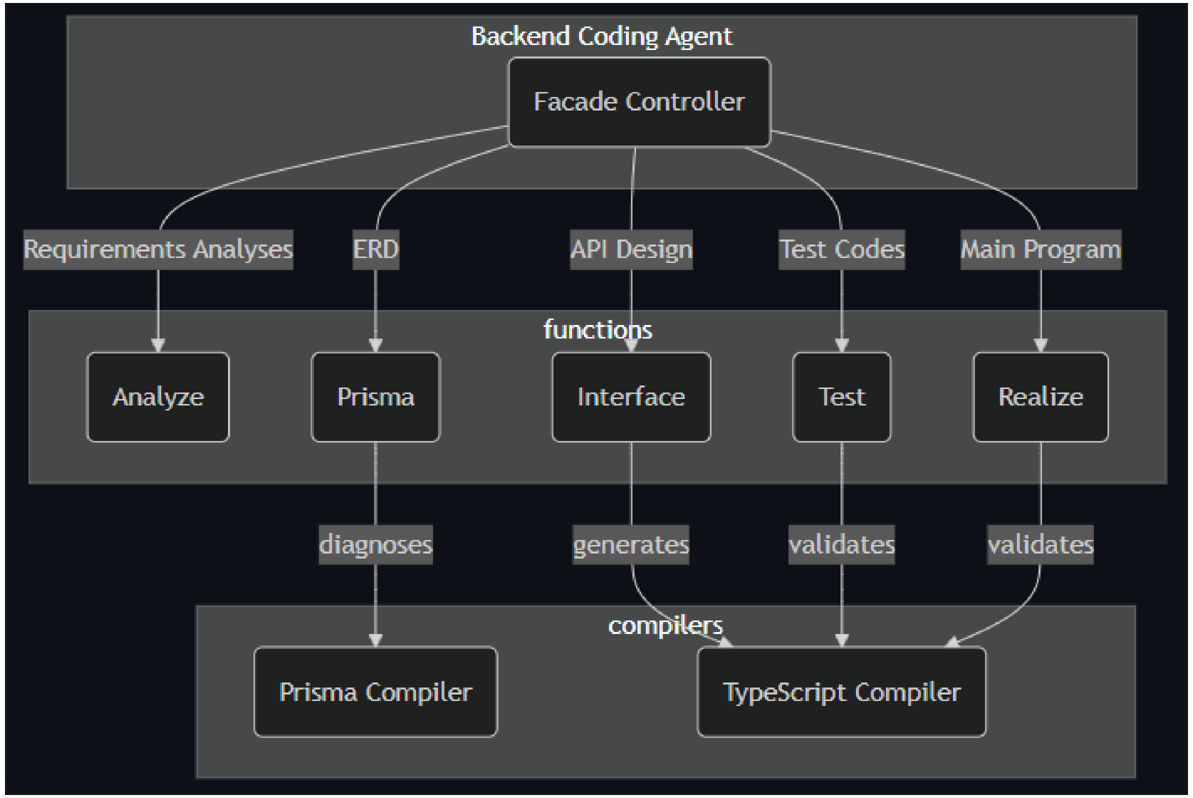
- Fully Documented



Spiral Waterfall Model

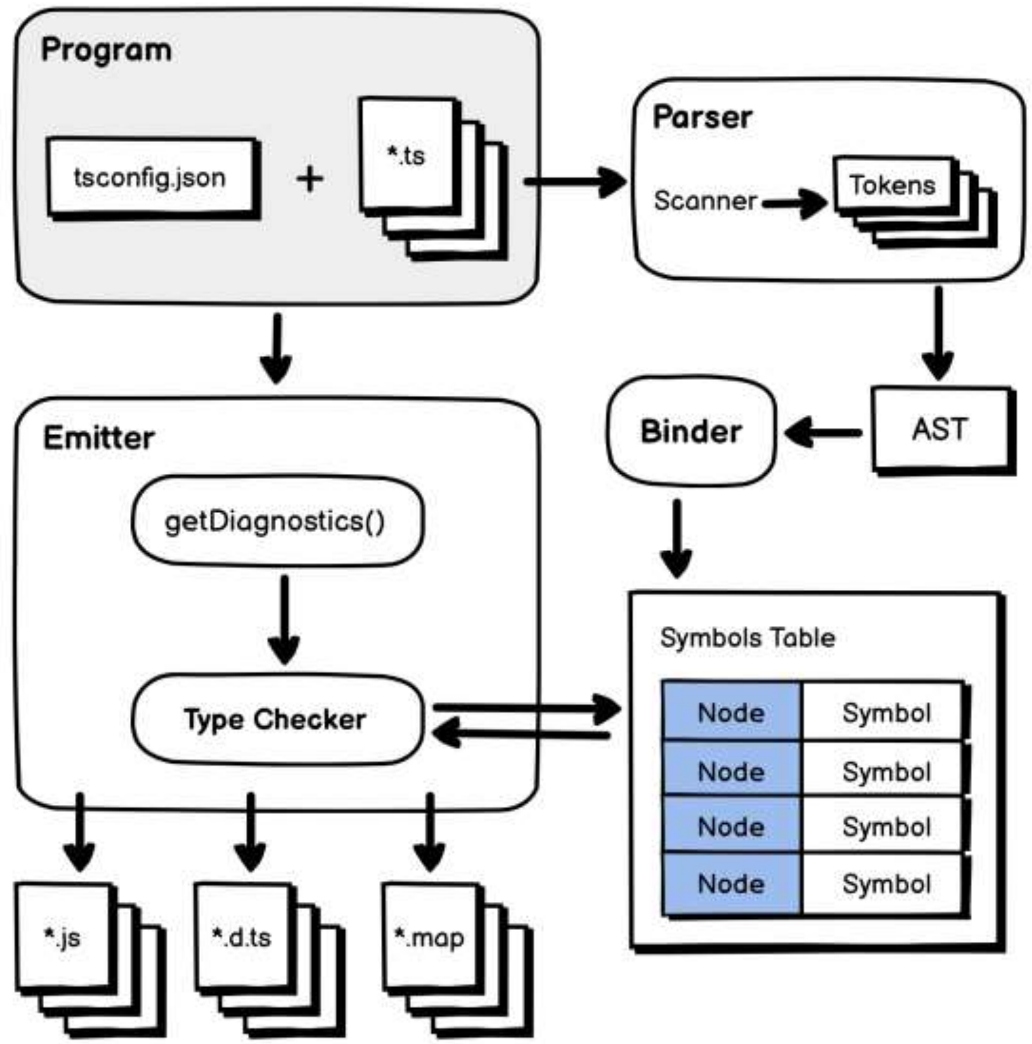
- Realize Agent

- Develop main source code
- Referencing
  - Requirements analyses
  - Prisma schema files
  - OpenAPI Interfaces
  - E2e Test Functions



Compiler Feedback

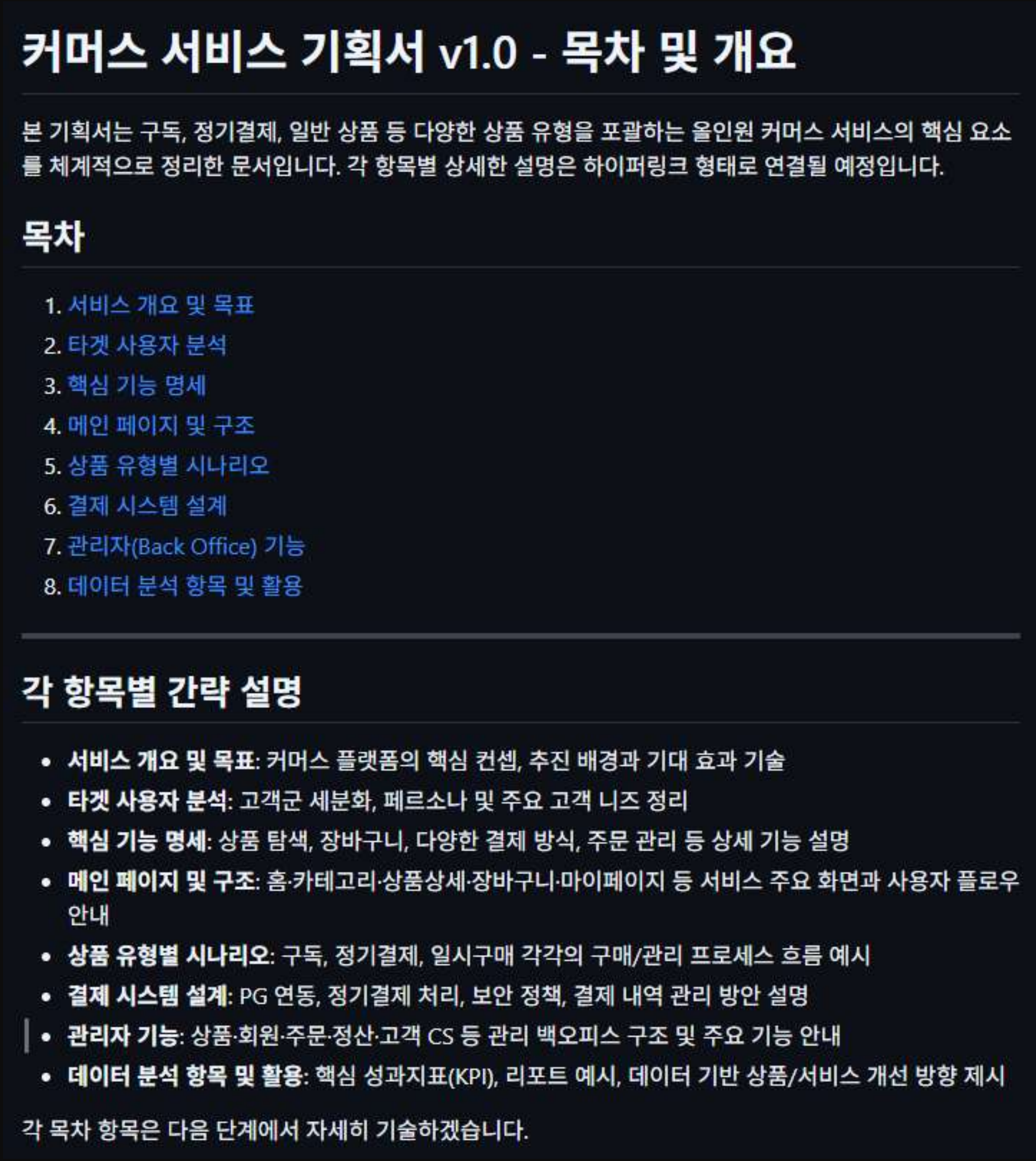
- There're two compilers
  - Prisma Compiler
  - TypeScript Compiler
- When AI makes wrong code
- Turn back to previous step
- With detailed error information



Demonstration

- Analyze Agent
- Complemented by Review Agent

- User Prompt
  - 구독, 정기결제, 일반 상품 등 다양한 상품 유형을 포함하는 커머스를 기획해줘



커머스 서비스 기획서 v1.0 - 목차 및 개요

본 기획서는 구독, 정기결제, 일반 상품 등 다양한 상품 유형을 포괄하는 올인원 커머스 서비스의 핵심 요소를 체계적으로 정리한 문서입니다. 각 항목별 상세한 설명은 하이퍼링크 형태로 연결될 예정입니다.

목차

1. 서비스 개요 및 목표
2. 타겟 사용자 분석
3. 핵심 기능 명세
4. 메인 페이지 및 구조
5. 상품 유형별 시나리오
6. 결제 시스템 설계
7. 관리자(Back Office) 기능
8. 데이터 분석 항목 및 활용

각 항목별 간략 설명

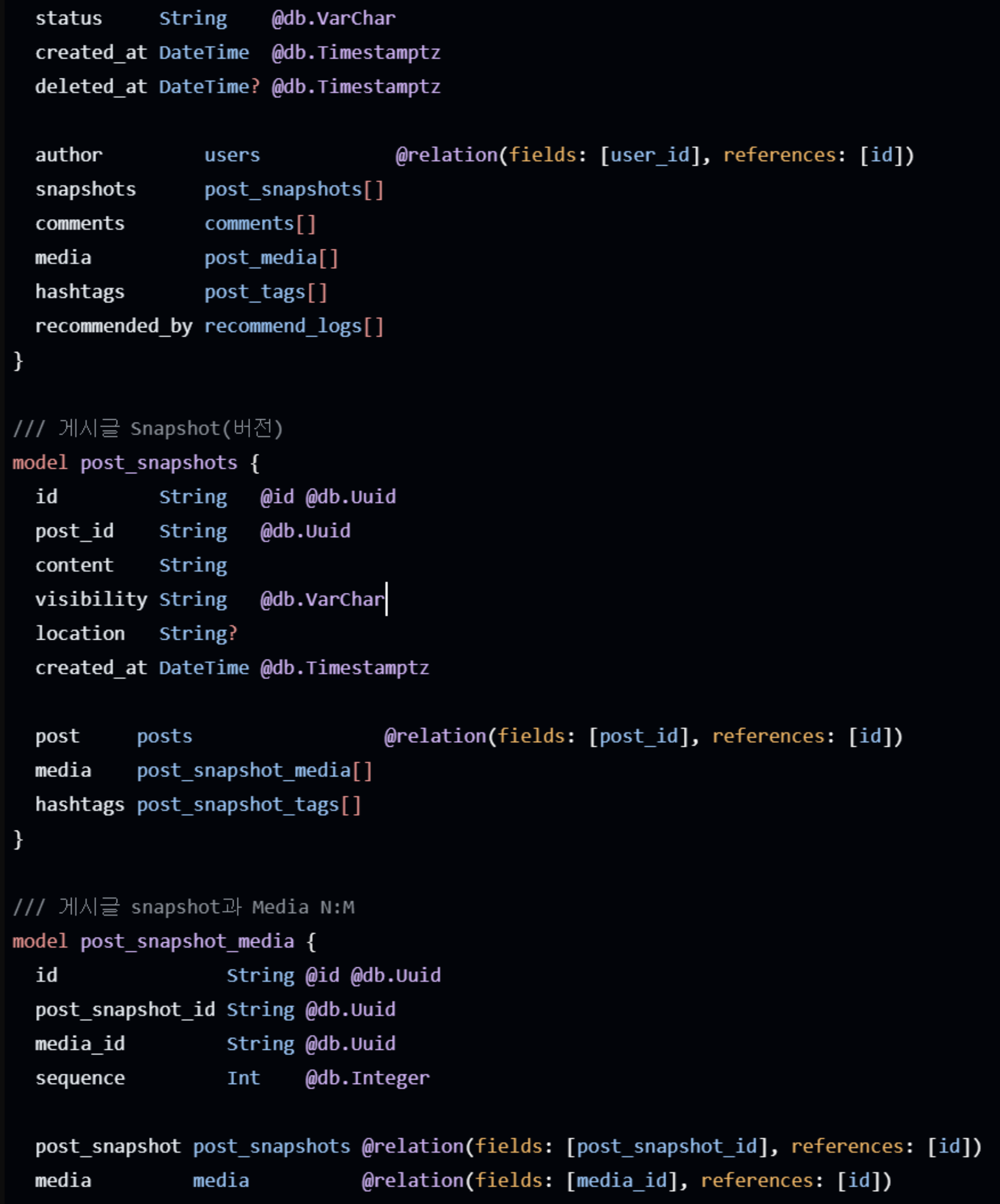
- **서비스 개요 및 목표**: 커머스 플랫폼의 핵심 컨셉, 추진 배경과 기대 효과 기술
- **타겟 사용자 분석**: 고객군 세분화, 페르소나 및 주요 고객 니즈 정리
- **핵심 기능 명세**: 상품 탐색, 장바구니, 다양한 결제 방식, 주문 관리 등 상세 기능 설명
- **메인 페이지 및 구조**: 홈·카테고리·상품상세·장바구니·마이페이지 등 서비스 주요 화면과 사용자 플로우 안내
- **상품 유형별 시나리오**: 구독, 정기결제, 일시구매 각각의 구매/관리 프로세스 흐름 예시
- **결제 시스템 설계**: PG 연동, 정기결제 처리, 보안 정책, 결제 내역 관리 방안 설명
- **관리자 기능**: 상품·회원·주문·정산·고객 CS 등 관리 백오피스 구조 및 주요 기능 안내
- **데이터 분석 항목 및 활용**: 핵심 성과지표(KPI), 리포트 예시, 데이터 기반 상품/서비스 개선 방향 제시

각 목차 항목은 다음 단계에서 자세히 기술하겠습니다.

Demonstration

- Prisma Agent
- Design about requirements

- Detailed by Review Agent
- Corrected by Prisma Compiler



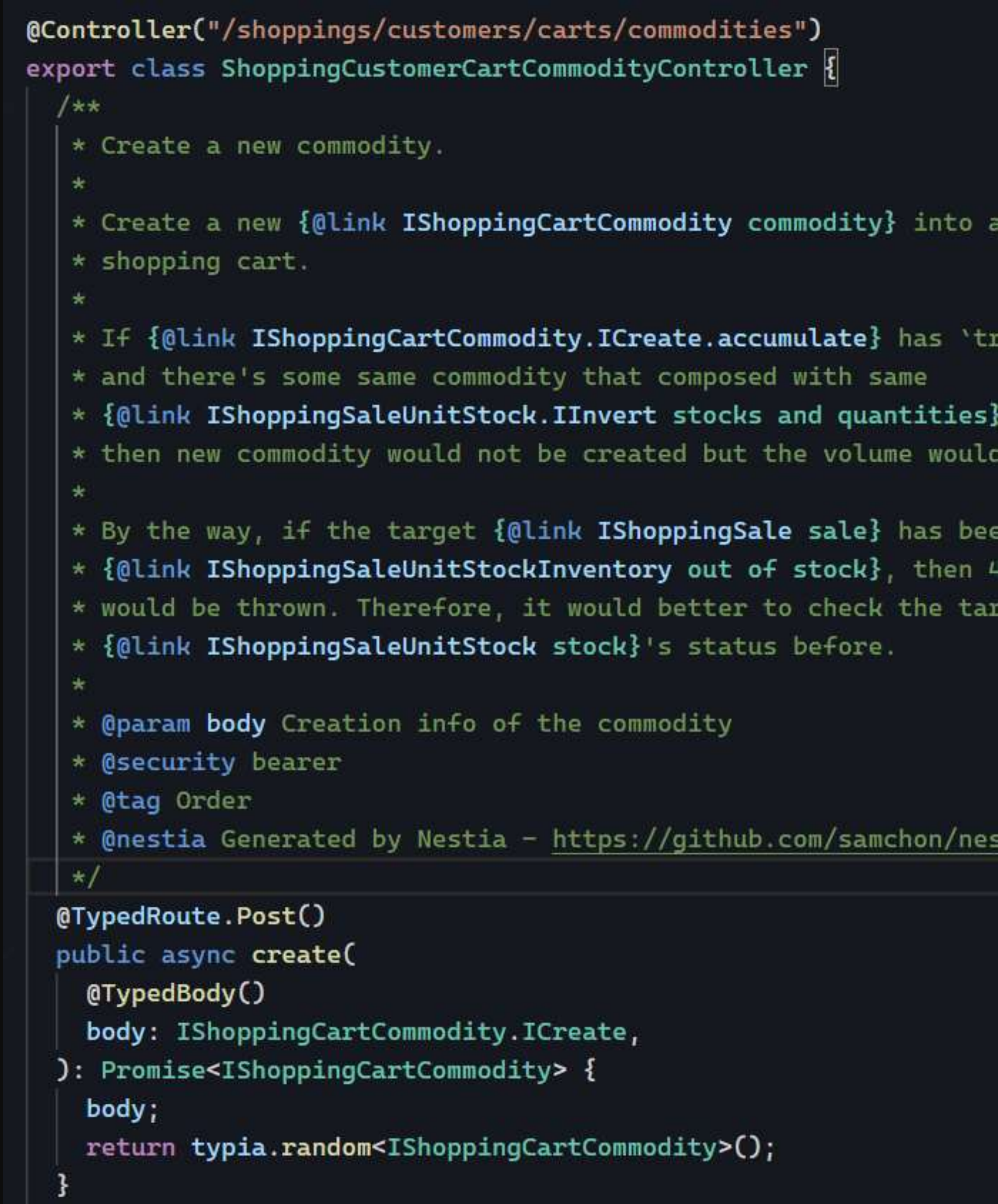
```
  status     String    @db.VarChar
  created_at DateTime  @db.Timestamptz
  deleted_at DateTime? @db.Timestamptz

  author         users            @relation(fields: [user_id], references: [id])
  snapshots      post_snapshots[]
  comments       comments[]
  media          post_media[]
  hashtags       post_tags[]
  recommended_by recommend_logs[]
}

/// 게시글 Snapshot(버전)
model post_snapshots {
  id         String   @id @db.Uuid
  post_id    String   @db.Uuid
  content    String
  visibility String   @db.VarChar
  location   String?
  created_at DateTime @db.Timestamptz

  post     posts                 @relation(fields: [post_id], references: [id])
  media    post_snapshot_media[]
  hashtags post_snapshot_tags[]
}

/// 게시글 snapshot과 Media N:M
model post_snapshot_media {
  id               String @id @db.Uuid
  post_snapshot_id String @db.Uuid
  media_id         String @db.Uuid
  sequence         Int    @db.Integer

  post_snapshot post_snapshots @relation(fields: [post_snapshot_id], references: [id])
  media         media          @relation(fields: [media_id], references: [id])
```

Demonstration

- Interface Agent

- Write OpenAPI Document
  - By analyzing requirements
  - Referencing Prisma schema
- Transform OpenAPI Document
- To NestJS Server

- Enhanced by Review Agent
- Validated by OpenAPI Validator
- Corrected by TypeScript Compiler



```
@Controller("/shoppings/customers/carts/commodities")
export class ShoppingCustomerCartCommodityController {
  /**
   * Create a new commodity.
   *
   * Create a new {@link IShoppingCartCommodity commodity} into a
   * shopping cart.
   *
   * If {@link IShoppingCartCommodity.ICreate.accumulate} has `tr
   * and there's some same commodity that composed with same
   * {@link IShoppingSaleUnitStock.IInvert stocks and quantities}
   * then new commodity would not be created but the volume woul
   *
   * By the way, if the target {@link IShoppingSale sale} has bee
   * {@link IShoppingSaleUnitStockInventory out of stock}, then
   * would be thrown. Therefore, it would better to check the tar
   * {@link IShoppingSaleUnitStock stock}'s status before.
   *
   * @param body Creation info of the commodity
   * @security bearer
   * @tag Order
   * @nestia Generated by Nestia - https://github.com/samchon/nes
   */
  @TypedRoute.Post()
  public async create(
    @TypedBody()
    body: IShoppingCartCommodity.ICreate,
  ): Promise<IShoppingCartCommodity> {
    body;
    return typia.random<IShoppingCartCommodity>();
  }
```

AutoView

Frontend Automation Tool

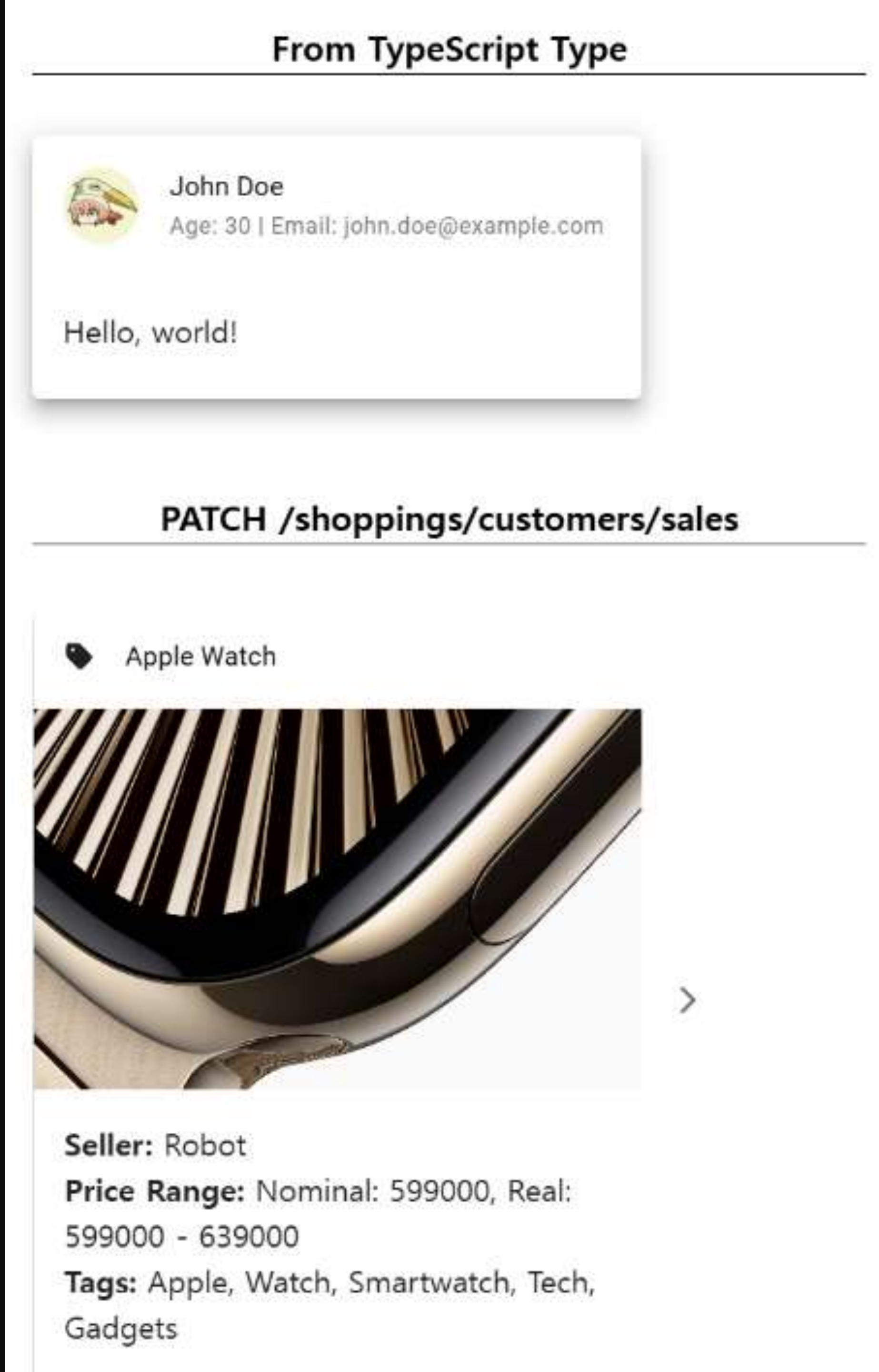
Principles

Blueprint

Frontend Automation

- Type to Frontend code
    - TypeScript Type
    - Swagger/OpenAPI Document

- Easy to Develop
- Mass Productive



From TypeScript Type

John Doe

Age: 30 | Email: john.doe@example.com

Hello, world!

PATCH /shoppings/customers/sales

Apple Watch

Seller: Robot

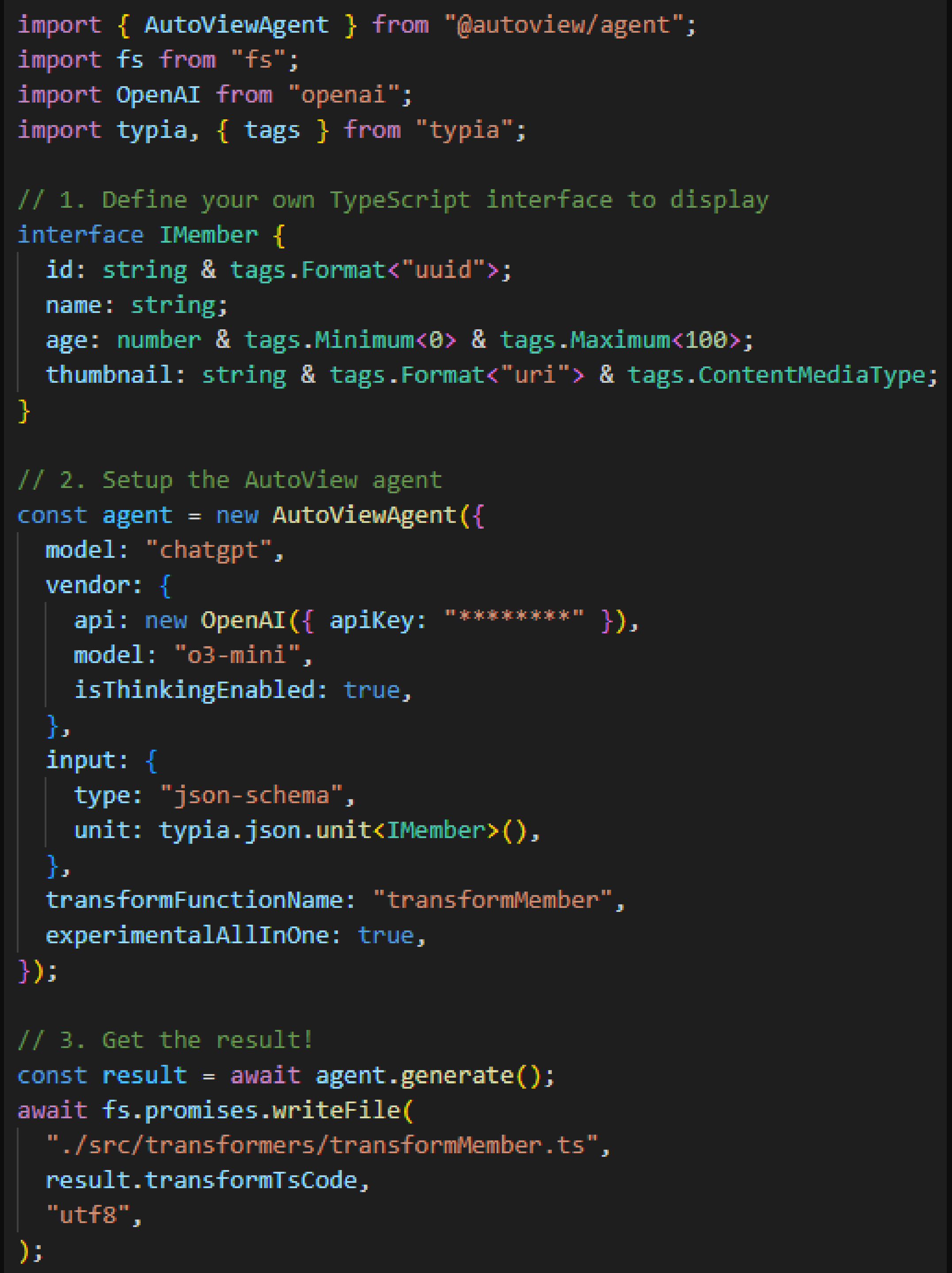
Price Range: Nominal: 599000, Real: 599000 - 639000

Tags: Apple, Watch, Smartwatch, Tech, Gadgets

Frontend Automation

- Frontend Code Generation
- From TypeScript Type

- If there're 100 TS Types
- 100 React codes generated

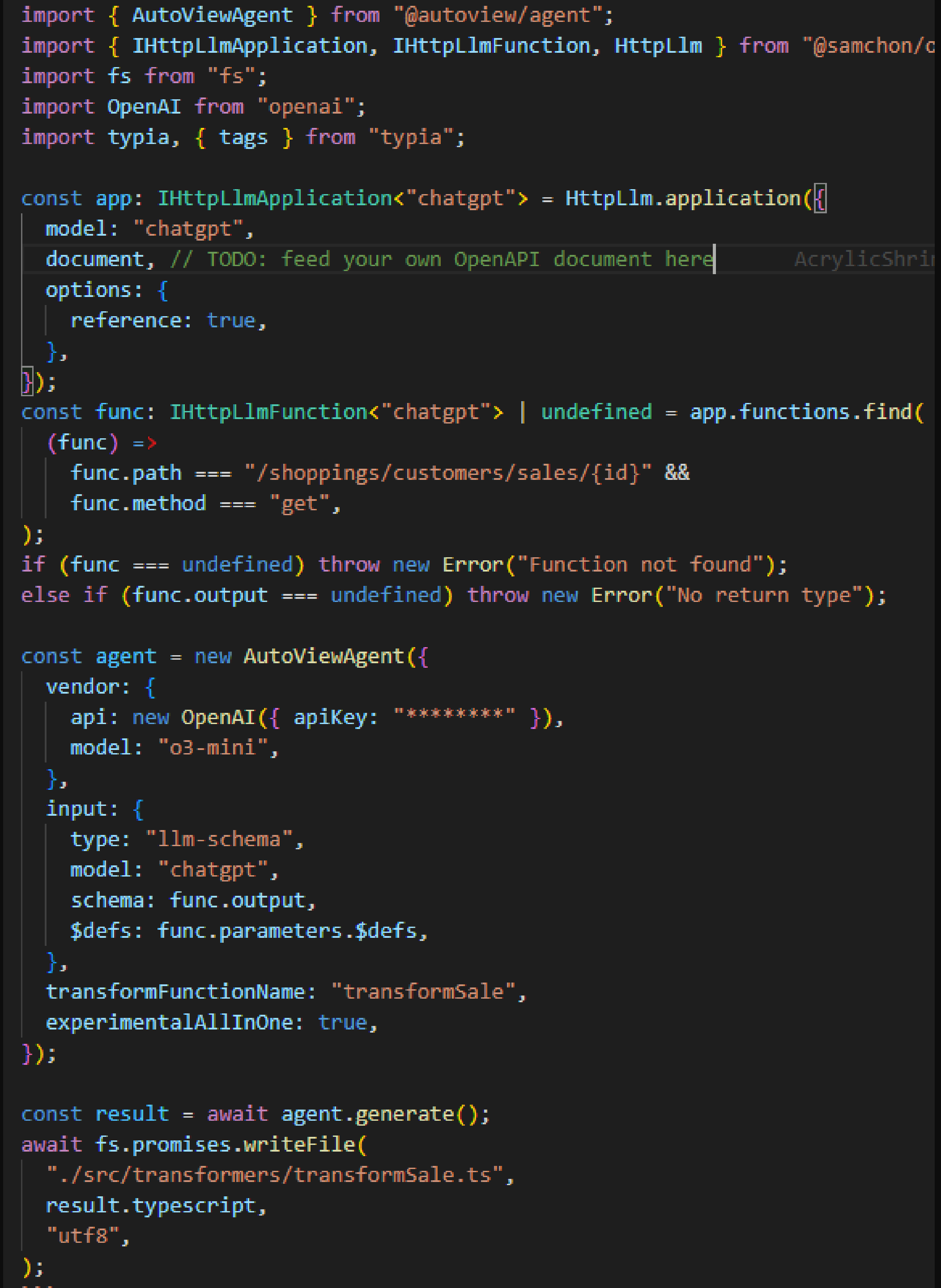


```typescript
import { AutoViewAgent } from "@autoview/agent";
import fs from "fs";
import OpenAI from "openai";
import typia, { tags } from "typia";

// 1. Define your own TypeScript interface to display
interface IMember {
  id: string & tags.Format<"uuid">;
  name: string;
  age: number & tags.Minimum<0> & tags.Maximum<100>;
  thumbnail: string & tags.Format<"uri"> & tags.ContentMediaType;
}

// 2. Setup the AutoView agent
const agent = new AutoViewAgent({
  model: "chatgpt",
  vendor: {
    api: new OpenAI({ apiKey: "********" }),
    model: "o3-mini",
    isThinkingEnabled: true,
  },
  input: {
    type: "json-schema",
    unit: typia.json.unit<IMember>(),
  },
  transformFunctionName: "transformMember",
  experimentalAllInOne: true,
});

// 3. Get the result!
const result = await agent.generate();
await fs.promises.writeFile(
  "./src/transformers/transformMember.ts",
  result.transformTsCode,
  "utf8",
);
```

Frontend Automation

- Frontend Code Generation
- From OpenAPI Document

- If 300 API functions are
- 300 React codes generated



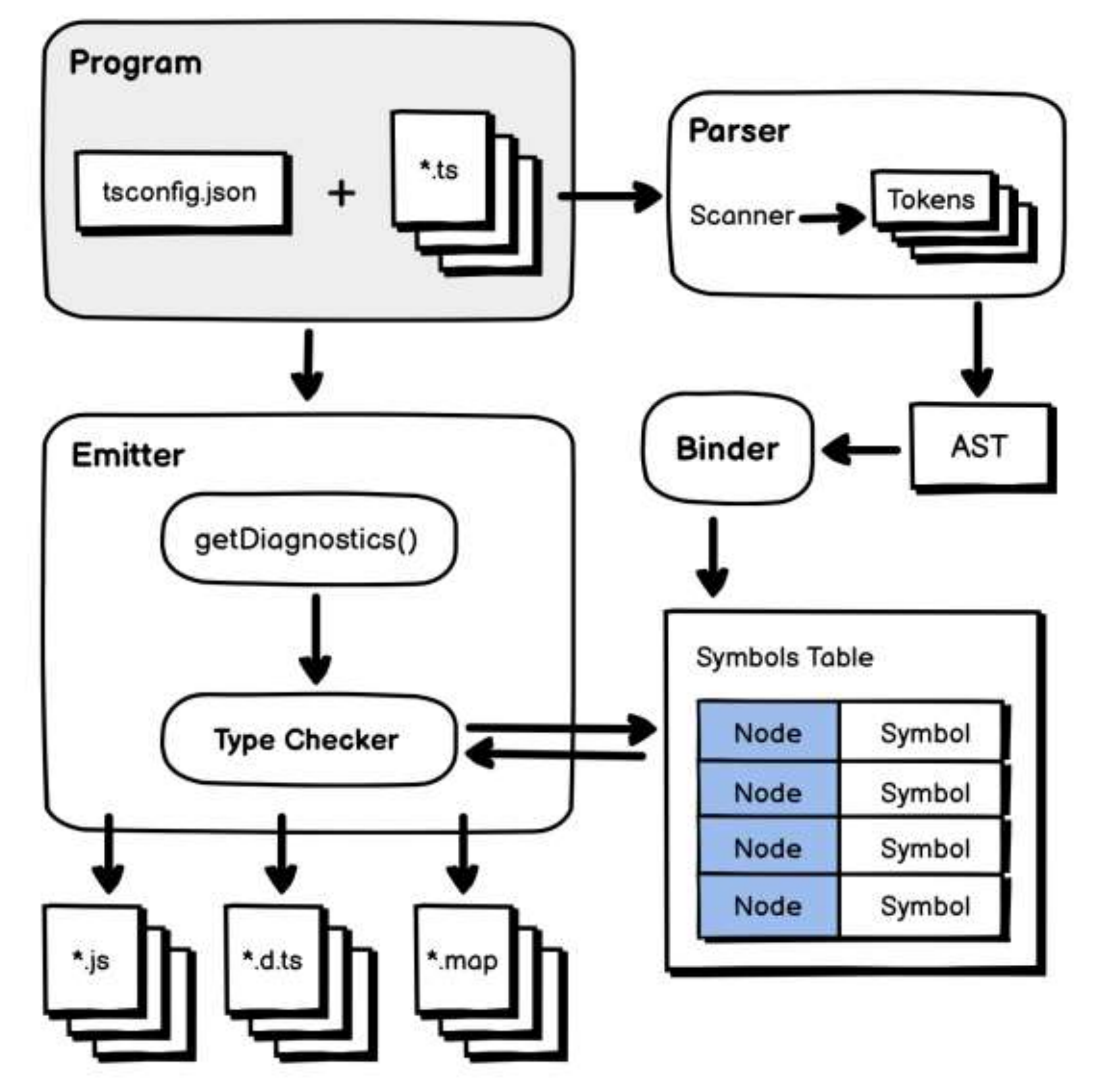
```
import { AutoViewAgent } from "@autoview/agent";
import { IHttpLlmApplication, IHttpLlmFunction, HttpLlm } from "@samchon/
import fs from "fs";
import OpenAI from "openai";
import typia, { tags } from "typia";

const app: IHttpLlmApplication<"chatgpt"> = HttpLlm.application({
  model: "chatgpt",
  document, // TODO: feed your own OpenAPI document here       AcrylicShri
  options: {
    reference: true,
  },
});
const func: IHttpLlmFunction<"chatgpt"> | undefined = app.functions.find(
  (func) =>
    func.path === "/shoppings/customers/sales/{id}" &&
    func.method === "get",
);
if (func === undefined) throw new Error("Function not found");
else if (func.output === undefined) throw new Error("No return type");

const agent = new AutoViewAgent({
  vendor: {
    api: new OpenAI({ apiKey: "********" }),
    model: "o3-mini",
  },
  input: {
    type: "llm-schema",
    model: "chatgpt",
    schema: func.output,
    $defs: func.parameters.$defs,
  },
  transformFunctionName: "transformSale",
  experimentalAllInOne: true,
});

const result = await agent.generate();
await fs.promises.writeFile(
  "./src/transformers/transformSale.ts",
  result.typescript,
  "utf8",
);
```

Principles

- Compiler Feedback

- AutoView generated TypeScript code
- Diagnosed by TypeScript compiler

- If there're some compilation errors
- Make agent to correct at next trial

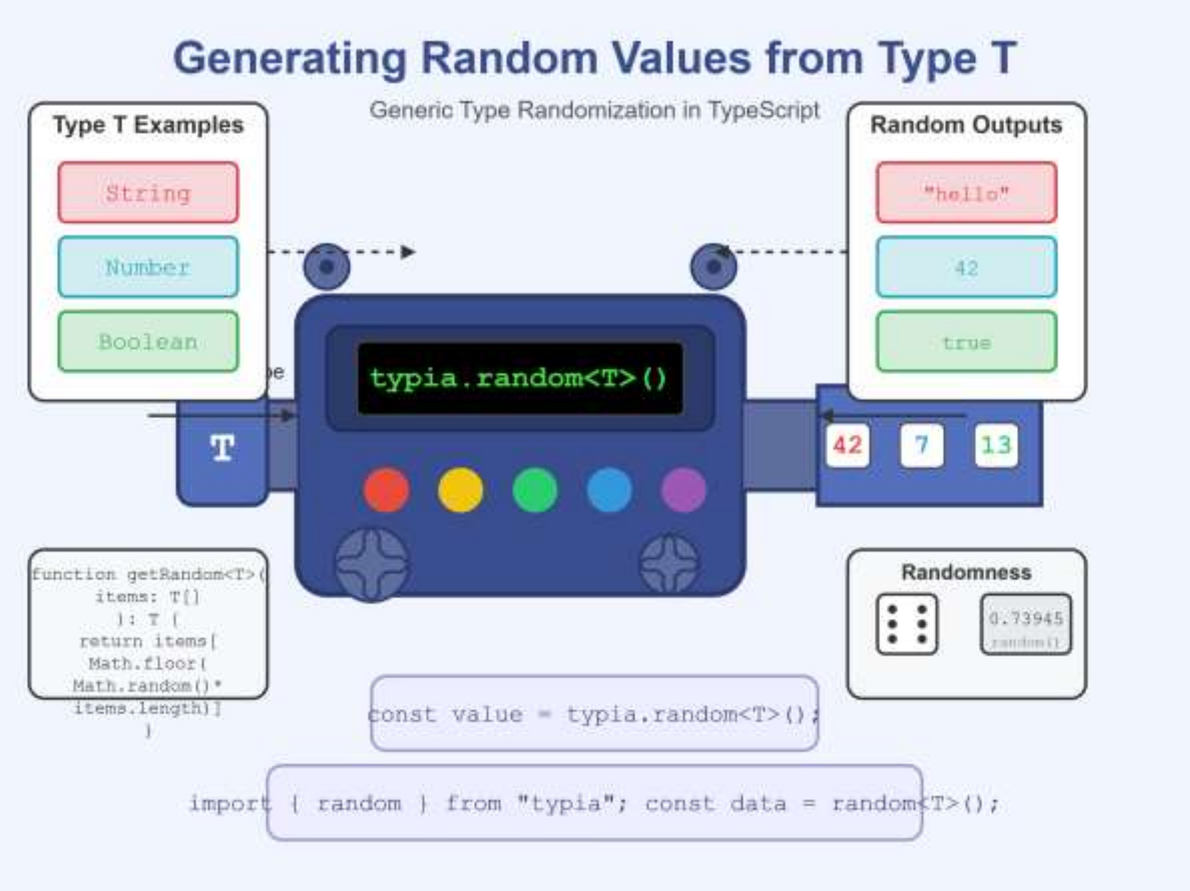


Principles

- Validation Feedback

- About target schema type T
- Generate random value
- By typia.random<T>() function

- And validate generated code
- Making proper UI component
- And let agent to correct it



```
function getRandom<T>(
  items: T[]
): T {
  return items[
    Math.floor(
      Math.random() *
      items.length)]
}
```

```
const value = typia.random<T>();
```

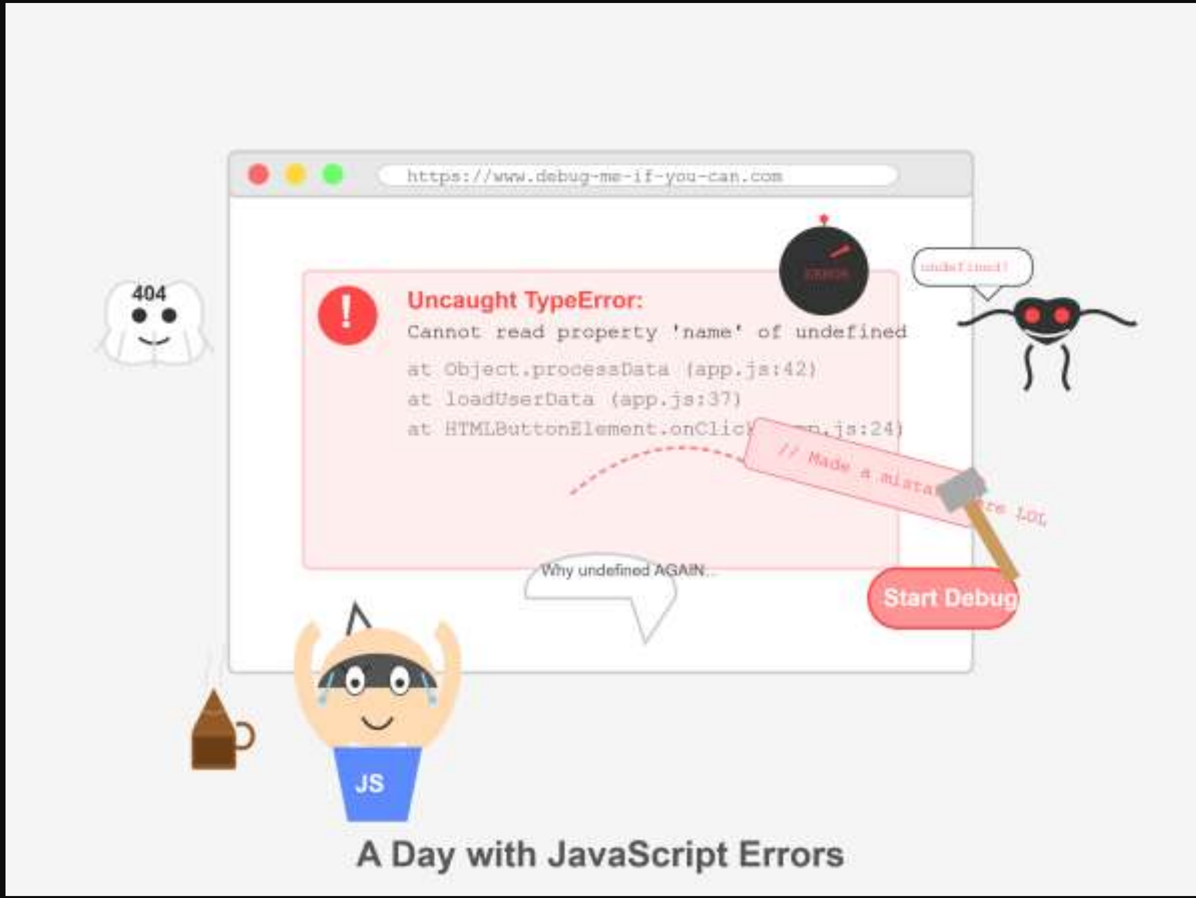
```
import { random } from "typia"; const data = random<T>();
```

Principles

- Exception Feedback

- About target schema type T
- Generate random value
- By typia.random<T>() function

- Test whether generated code
- Occurring exception or not
- And let agent to correct it

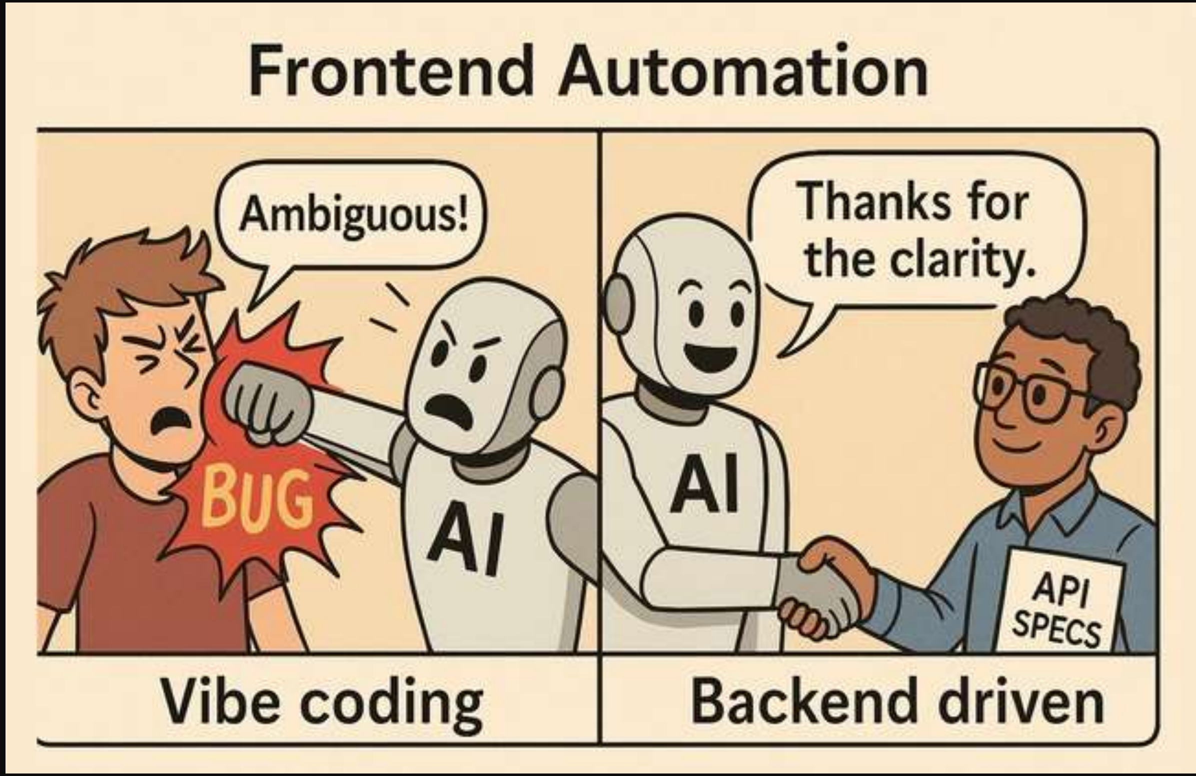


```
Uncaught TypeError:
Cannot read property 'name' of undefined
at Object.processData (app.js:42)
at loadUserData (app.js:37)
at HTMLButtonElement.onClic
```

A Day with JavaScript Errors

Blueprint

- Develop Backend Server
  - With Viral Coding
  - By AutoBE
- Create AI Chatbot
  - With Agentica
  - From swagger.json of above
- Make Frontend Application
  - With AutoView
  - From swagger.json of above

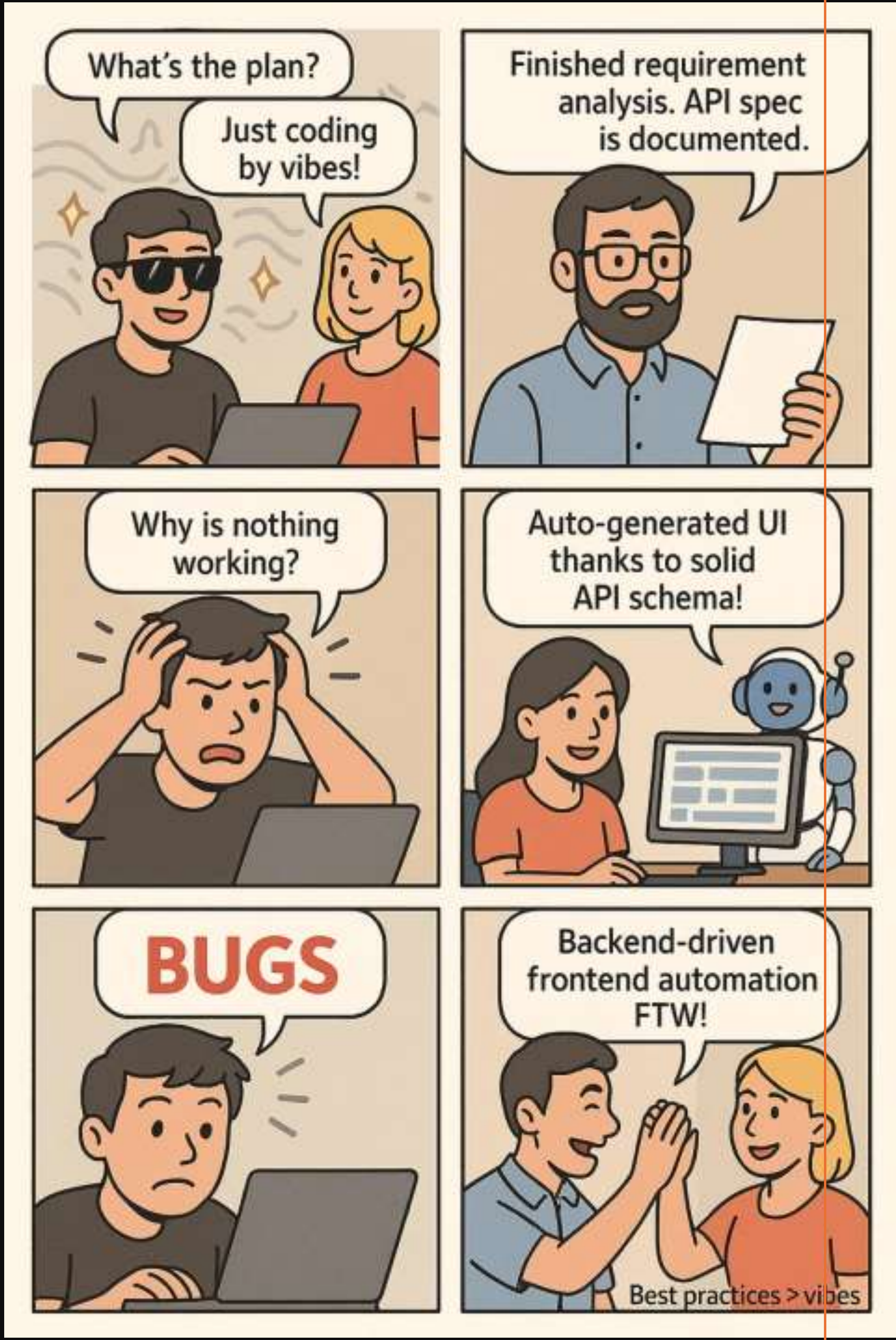


BluePrint

- Ultimate Viral Coding
- By Computer Science Skills

- Documentation
- JSON Schema Specification
- Compiler Feedback

- By WrtnLabs


What's the plan?
Just coding by vibes!
Finished requirement analysis. API spec is documented.
Why is nothing working?
Auto-generated UI thanks to solid API schema!
BUGS
Backend-driven frontend automation FTW!
Best practices > vibes

Q&A

2025-05-13

WrtnLabs