



AI-Driven API Design

Mike Amundsen (@mamund)

<https://github.com/mamund/2026-08-ai-driven-workshop>

<https://b.mamund.com/4gETA7f>



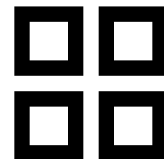
Mike Amundsen
@mamund

Introduction

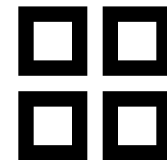
The background is a blurred image of a whiteboard in a meeting room. The whiteboard is covered with various hand-drawn diagrams, charts, and notes. There are several rectangular boxes, some containing text and others with graphical elements like arrows and small icons. A person is visible in the background, standing near the whiteboard, but they are out of focus. The overall scene suggests a collaborative work environment or a presentation.

Course schedule

- Course Introduction
- API Design Basics
- AI Prompt Basics
- Authoring API Stories
- Generating API Descriptions
- Generating API Documentation
- Generating OpenAPI Documents
- Generating Running API Prototypes
- Generating API Tests
- Generating Security Profiles
- Course Review



Plans for the day



- Review basics of AI prompting and API design
- Explore API Stories
- Learn API Description with ALPS
- Generate Design Assets with AI

Play-along demos

- I will demonstrate each lesson
- You can play-along with your own AI chatbot
- Your results will vary - that's a feature!



<https://github.com/mamund/2026-08-ai-driven-workshop>
<https://b.mamund.com/4gETA7f>

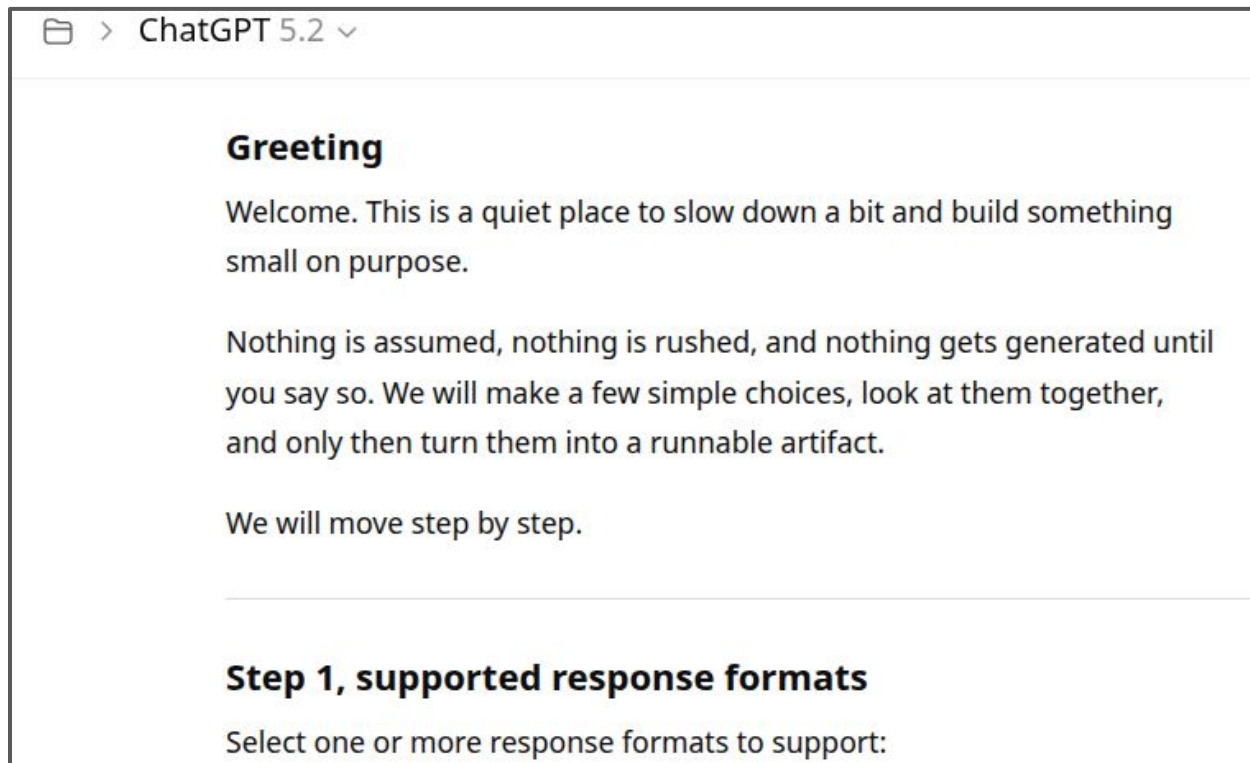


Hello, Coach!

Hello, Coach!

- AI chatbots are often used as an “answer machine”
- AI can also be used as a workbench tool
- You can design AI tools to coach you through a process

Hello, Coach!



Step 2, supported languages

Select one or more languages to support:

- English (en), always the default
- German (de)
- French (fr)
- Spanish (es)

Rules:

- If English is not selected, the service still defaults to English
- Unsupported requested languages fall back to English

Please reply with one or more of:

en

de

fr

es

Reflection

Here is what we have designed so far:

- Supported formats: JSON
- Supported languages: English and German
- Default format: JSON
- Default language: English
- Endpoints to be created:
 - GET /
 - GET /hello with optional name query argument
- / will:
 - Ignore query parameters
 - Support format and language negotiation
 - Include a discoverable link to /hello in its JSON representation
- /hello will:
 - Accept optional name
 - Fall back to a default greeting if name is missing or empty
 - Honor format and language negotiation

Nothing has been generated yet.

Please confirm that this matches your intent.

You may adjust anything now.



Open + server_en_fr_es.js storage ~/Desktop/Dropbox/Private/Projec...ign/coaches/2026-02-08-hello-coach/output Save ⚙️ - + ×

```
1/**
2 * Hello, coach
3 * A tiny Node.js HTTP service that demonstrates:
4 * - orientation through a root resource
5 * - representation negotiation via Accept
6 * - language negotiation via Accept-Language
7 * - safe defaults when headers are missing or unsupported
8 *
9 * Endpoints:
10 * - GET /
11 * - GET /hello
12 *
13 * Run:
14 *   node server.js
15 *
16 * Then try:
17 *   curl -H "Accept: application/json" -H "Accept-Language: en" http://localhost:3000/hello
18 *   curl -H "Accept: text/plain" -H "Accept-Language: fr" http://localhost:3000/hello
19 *   curl -H "Accept: text/plain" -H "Accept-Language: es" http://localhost:3000/hello
20 */
21
22 const http = require('http');
23 const { URL } = require('url');
24
25 const PORT = process.env.PORT ? Number(process.env.PORT) : 3000;
26
27 // Supported negotiation options
```

JavaScript ▾ Tab Width: 2 ▾ Ln 1, Col 1 ▾ INS

Synthesis

What was produced:

- A single file, `server.js`, implementing a minimal Node.js HTTP service.

What it demonstrates:

- A stable entry point at `GET /` that exposes a discoverable link to `GET /hello`
- A second resource, `GET /hello`, with an optional `name` query input
- Language negotiation via `Accept-Language`, supporting `en` and `de`, defaulting to `en`
- Safe defaults when the client provides missing or unsupported inputs

Hello, Coach! – Exercise

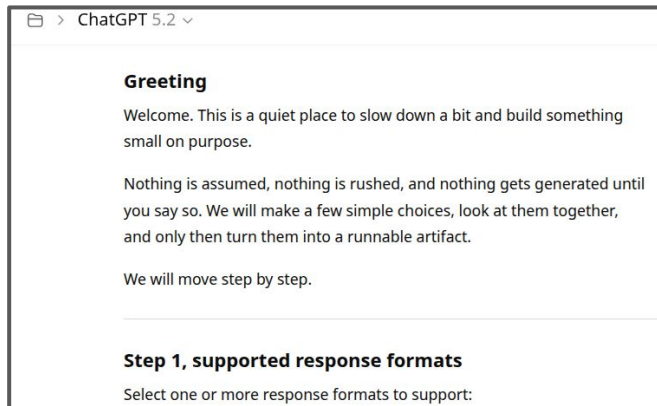
- Download your copy of the “Hello Coach” file from the repo
- Load it into your local chatbot
- Work through the coach to generate your [server.js](#)
- Optionally, test it on the command line
 - `node server.js`

```
Terminal - mca@mamund-ws: ~/.../2026-02-19-design-navigation
File Edit View Terminal Tabs Help
mca@mamund-ws:~/.../2026-02-19-design-navigation$ node server.js
Hello, coach listening on http://localhost:3000
█
```

```
Terminal - mca@mamund-ws: ~/.../2026-02-19-design-navigation
File Edit View Terminal Tabs Help
mca@mamund-ws:~/.../2026-02-19-design-navigation$ curl localhost:3000/
{
  "message": "This is the entry point. Follow the link to /hello.",
  "links": {
    "hello": "/hello",
    "helloWithName": "/hello?name=Deena"
  }
}
mca@mamund-ws:~/.../2026-02-19-design-navigation$ curl localhost:3000/hello
{
  "greeting": "Hello.",
  "language": "en",
mca@mamund-ws:~/.../2026-02-19-design-navigation$ curl localhost:3000/hello?name=Mike
{
  "greeting": "Hello, Mike.",
  "language": "en",
  "name": "Mike"
}
mca@mamund-ws:~/.../2026-02-19-design-navigation$ █
```

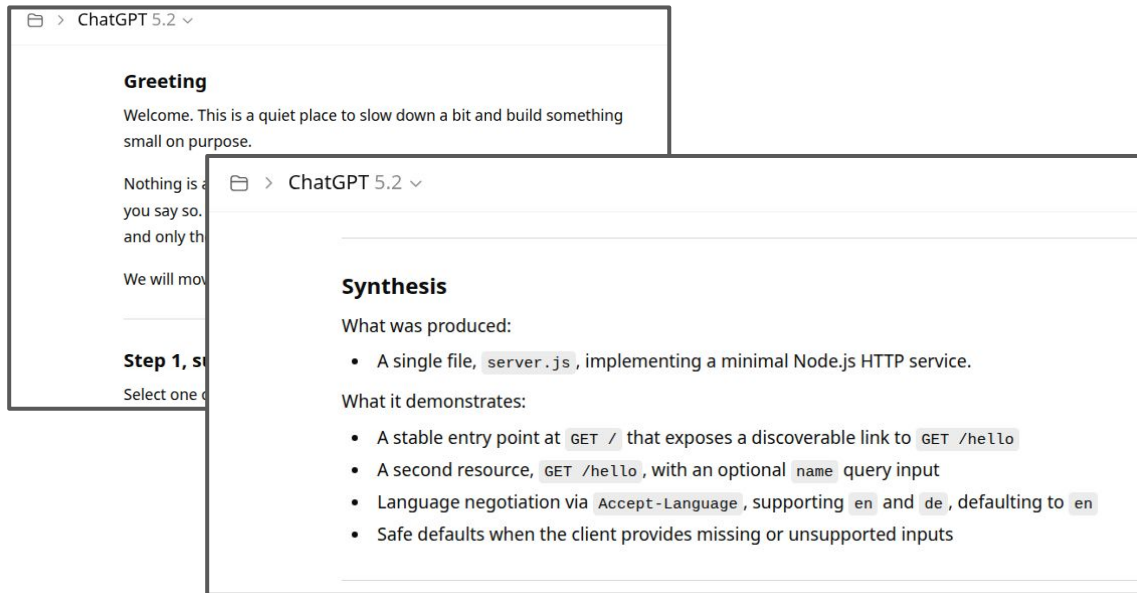
Hello, Coach!

- We'll be focused on using AI as a workbench tool



Hello, Coach!

- We'll be focused on using AI as a workbench tool



Hello, Coach!

- We'll be focused on using AI as a workbench tool

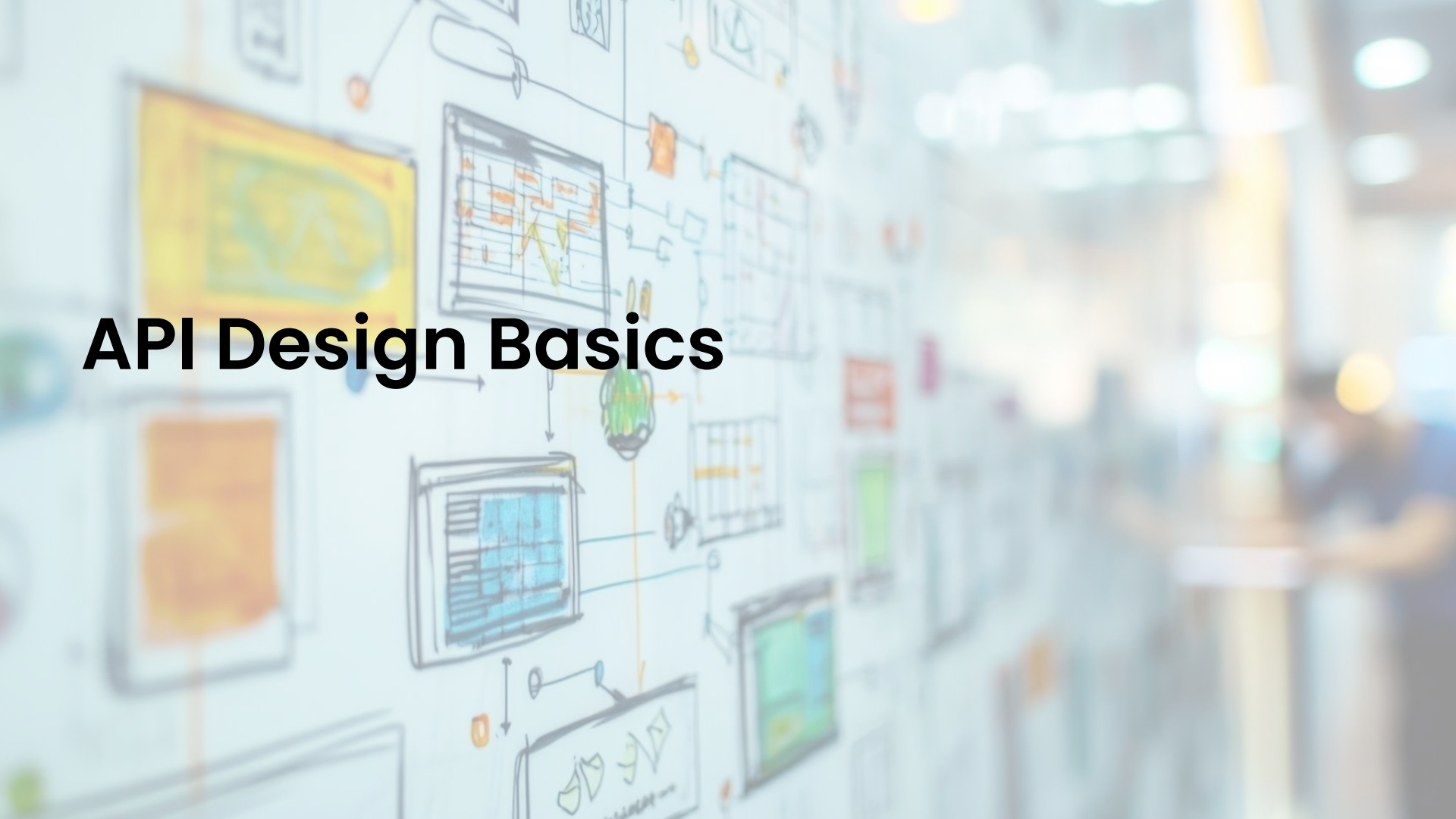
The image is a collage of screenshots illustrating the development of a simple Node.js HTTP service using AI. It includes:

- ChatGPT 5.2 Interface:** Two overlapping windows showing the chat interface. The top window displays a "Greeting" section with the text: "Welcome. This is a quiet place to slow down a bit and build something small on purpose." and "Nothing is a... you say so... and only th... We will mov...". The bottom window displays a "Synthesis" section with the text: "What was produced:" followed by a bullet point: "A single file, `server.js`, implementing a minimal Node.js HTTP service." and "What it demonstrates:" followed by a bullet point: "A stable entry point at `GET /` that exposes a discoverable link to `GET /hello`".
- Terminal Output:** A screenshot of a terminal window showing the execution of a Node.js server. The prompt is `mca@mamund-ws:~/2026-02-19-design-navigation`. The command `node server.js` is executed, resulting in the output: "Hello, coach listening on http://localhost:3000".
- API Interactions:** Two more terminal screenshots showing `curl` commands and their JSON responses. The first `curl` command is `curl localhost:3000/`, which returns a JSON object with a "message" and "links" array. The second `curl` command is `curl localhost:3000/hello`, which returns a JSON object with "greeting", "language", and "name" fields. A third `curl` command is `curl localhost:3000/hello?name=Mike`, which returns a similar JSON object with the name "Mike".

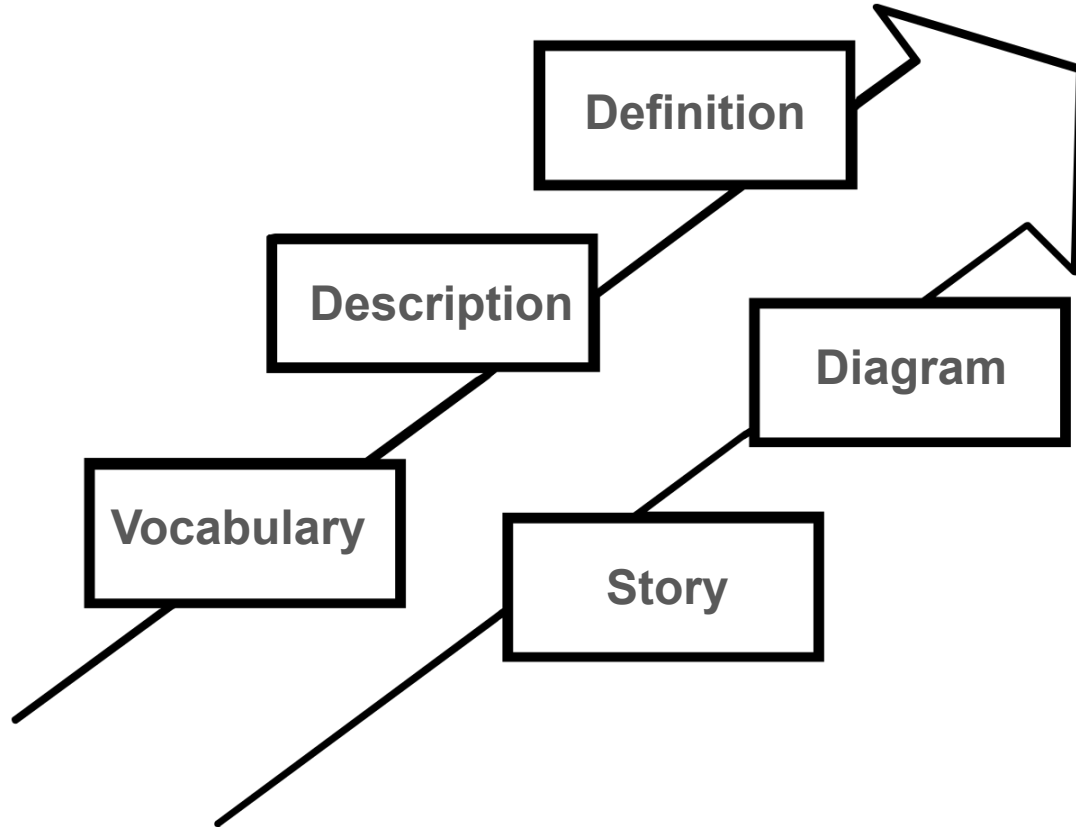


Let's get started ...

API Design Basics

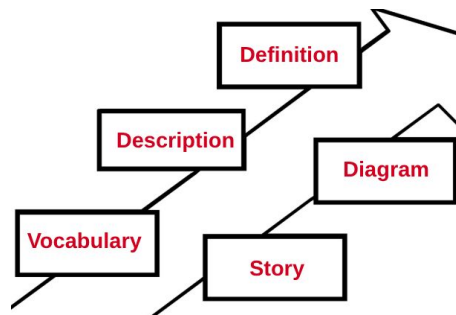


API Design Basics : Five Assets



API Design Basics: Story

- APIs start with a story
 - "We need..."
 - "Our customers requested..."
 - "I have an idea..."
- Stories are shared understanding
 - Our brains are wired for stories, not data
 - Stories are accessible
 - Stories are repeatable



API Design Basics: Story

Task Management

Purpose

We need to track 'Task' records in order to improve both timeliness and accuracy of customer follow-up activity.

Data Properties

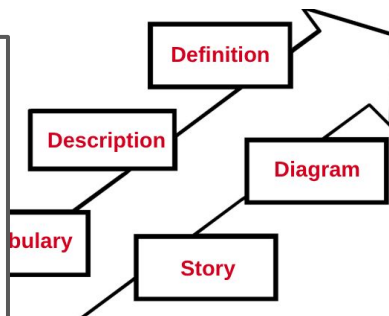
In this first pass at the application, we need to keep track of the following data properties:

- **id** : a globally unique value for each record [uuid]
- **title** : the text content of the record (the title of the task) [string]
- **description** : the description of the record (the task details) [text]
- **dueDate** : the date the record is due to be completed [date]
- **status** : Indicates the status of the record (active, completed) [enumerated string]
- **priority** : the priority of the task (a number between 1 and 5) [enumerated number]
- **assignedUser** : the user assigned to handle the task (a simple name string) [string]

Resources

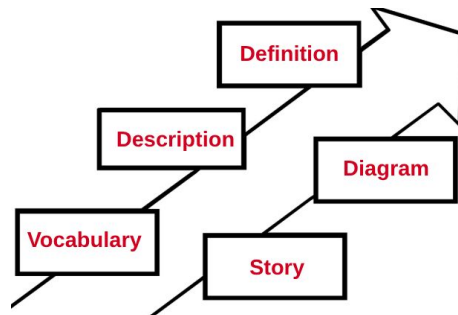
The following are resources, or states, of the API. Each state has one or more possible Actions.

- **Home** : The home or landing page of the API. Users typically start here.
 - Actions: **GetTaskList**, **GetFilteredTaskList**, **ShowHomePage**
- **TaskCollection** : The list of tasks in the system are displayed here.
 - Actions: **ShowHomePage**, **GetTaskList**, **GetFilteredTaskList**, **CreateNewTask**, **GetTaskItem**
- **TaskItem** : This resource shows a single task (selected from the TaskCollection)
 - Actions: **EditExistingTask**, **UpdateStatusOfTask**, **SetDueDateOfTask**, **AssignUserToTask**, **GetTaskList**, **GetFilteredTaskList**, **ShowHomePage**



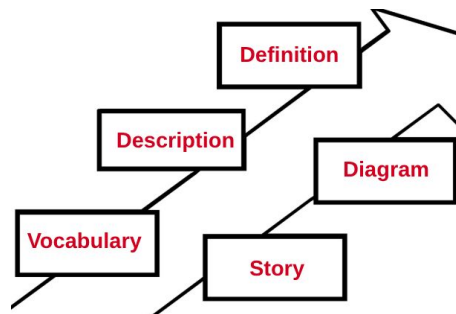
API Design Basics: Vocabulary

- All the "magic words" related to the domain
- Not just the properties:
 - `id`, `givenName`, `etc.`
- Also :
 - **Actions** (`create`, `approve`, `save`, `etc.`)
 - **Enumerators** (`status.active`, `status.inactive`)
 - **Containers** (`item.id`, `item.givenName`, `etc.`)
 - **Resources/Topics** (`home`, `collection`, `item`)



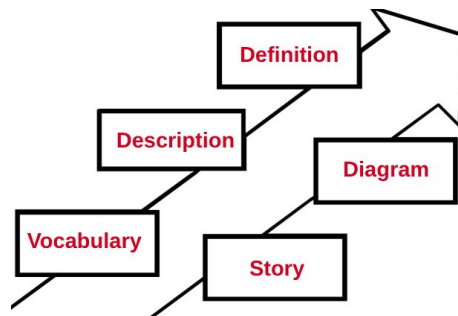
API Design Basics: Vocabulary

Semantic Descriptors				
Q Search semantic descriptors...				
Type	ID	Title	Contained	Extra Info
☐	assignedUser	AssignedUser		tag: task-management doc: User assigned to the task
☐	description	Description		tag: task-management doc: Detailed task description
☒	doAssignUser	AssignUser	☐ id ☐ assignedUser	tag: task-management rt: TaskItem doc: Required: id, assignedUser
☒	doCreateTask	CreateTask	☐ id ☐ title ☐ description ☐ dueDate ☐ status ☐ priority ☐ assignedUser	tag: task-management rt: TaskCollection doc: Required: id, title, status
☒	doEditTask	EditTask	☐ id ☐ title ☐ description ☐ dueDate ☐ status ☐ priority ☐ assignedUser	tag: task-management rt: TaskItem doc: Required: id, title, status
☒	doSetDueDate	SetDueDate	☐ id ☐ dueDate	tag: task-management rt: TaskItem

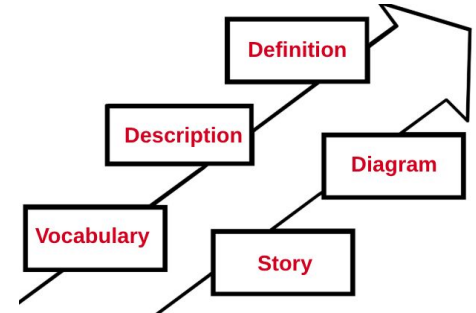
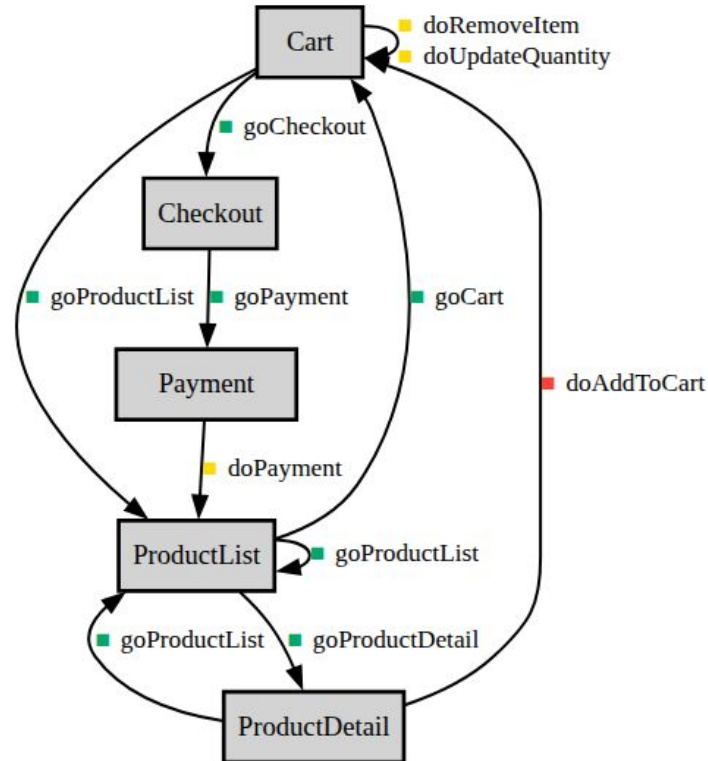


API Design Basics: Diagram

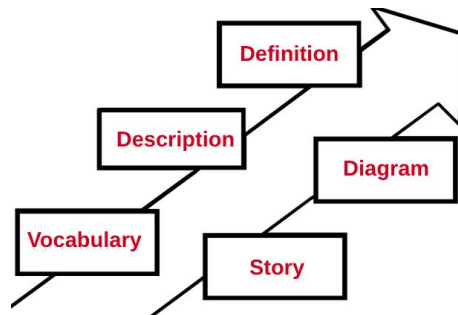
- Visual representation of the API
- Not ..
 - a state diagram
 - a flow diagram
- An *interaction* diagram



API Design Basics: Diagram



API Design Basics: Description



- Why "Descriptions"?
 - The complete design in one place, in one voice.
- Who uses descriptions?
 - Architects, Designers, Developers, Test, Security, etc.
- Time Traveling w/ descriptions
 - Options for the future, replays for the past
- Interaction-centric and Operation-centric

API Design Basics: Description

app-state-diagram

JSON



Download

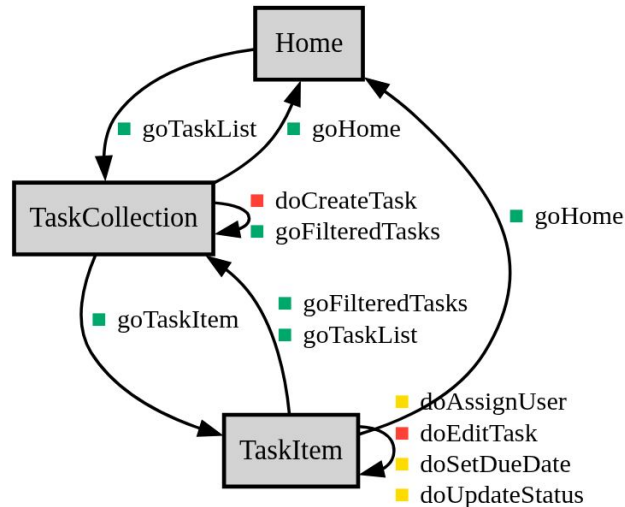
Definition

Diagram

ory

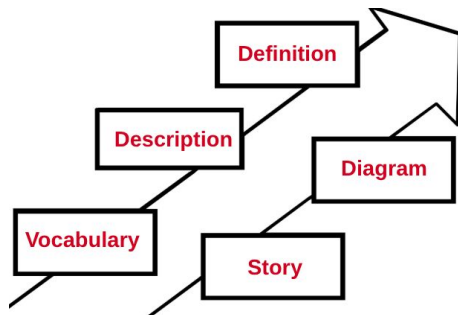
Task Management API

ALPS profile for managing task records including required fields on actions.



API Design Basics: Definition

- API definitions translate design into implementation
- API definitions are implementation-specific
 - OpenAPI (HTTP)
 - AnsyncAPI (Event-Driven)
 - ProtoBuf (gRPC)
 - SDL (GraphQL)
- Usually API definitions power "generators"
 - Code
 - Diagrams
 - Documentation



API Design Basics: Definition

API Hub | Design
Powered by Swagger

API Hub is now live! A new experience is on the way—stay tuned. [Learn more](#)

task-management_api Version: 1.0.0 Export

Save to Hub

```
1 openapi: 3.1.0
2 info:
3   title: Task Management API
4   version: 1.0.0
5   description: OpenAPI specification generated from ALPS profile for task tracking.
6 paths:
7   /goShowHomePage:
8     get:
9       operationId: goShowHomePage
10      summary: Go to Home
11      description: ''
12      responses:
13        '200':
14          description: Successful operation
15          content:
16            application/json:
17              schema:
18                $ref: '#/components/schemas/Home'
19        '400':
20          description: Invalid input
21        '404':
22          description: Resource not found
23      tags:
24        - navigation
25      parameters: []
26   /goGetTaskCollection:
27     get:
28       operationId: goGetTaskCollection
29       summary: Get Task Collection
30       description: ''
31       responses:
32        '200':
33          description: Successful operation
34          content:
35            application/json:
36              schema:
37                $ref: '#/components/schemas/TaskCollection'
38        '400':
39          description: Invalid input
40        '404':
41          description: Resource not found
42      tags:
```

Task Management API

1.0.0 OAS 3.1

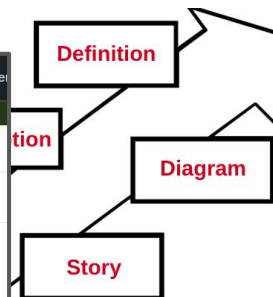
OpenAPI specification generated from ALPS profile for task tracking.

navigation

GET	/goShowHomePage	Go to Home
-----	-----------------	------------

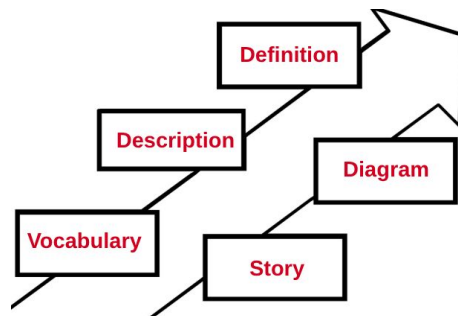
task-management

GET	/goGetTaskCollection	Get Task Collection
GET	/goGetTaskItem	Get Task Item
POST	/doCreateNewTask	Create Task
POST	/doEditExistingTask	Edit Task
PUT	/doUpdateStatusOfTask	Update Status
PUT	/doSetDueDateOfTask	Set Due Date

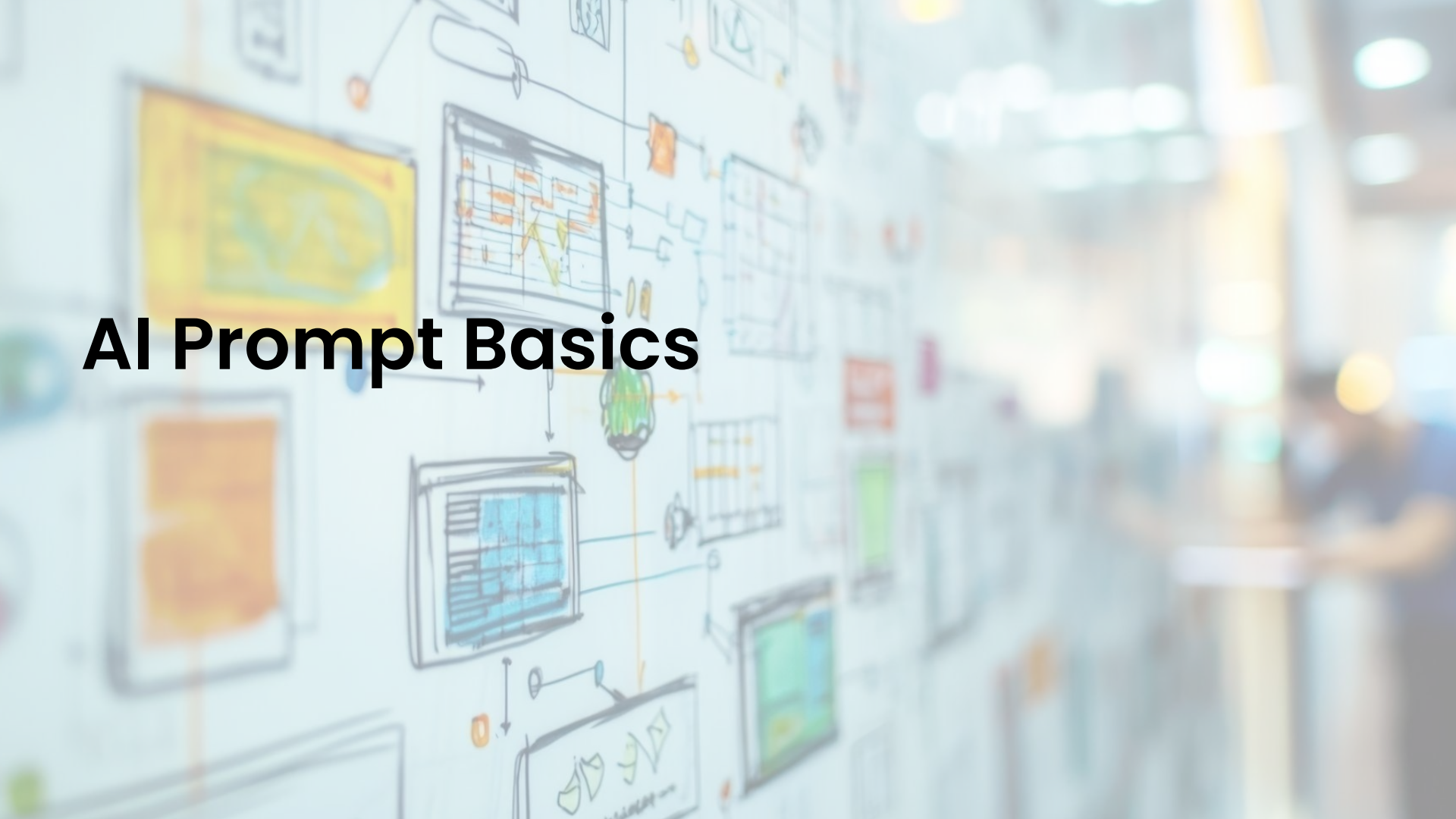


API Design Basics: Summary

- Story
 - Stories are shared understanding
- Vocabulary
 - All the "magic words" related to the API domain
- Diagram
 - Visual representation of the API
- Description
 - The complete design in one place, in one voice.
- Definition
 - Translate design into implementation

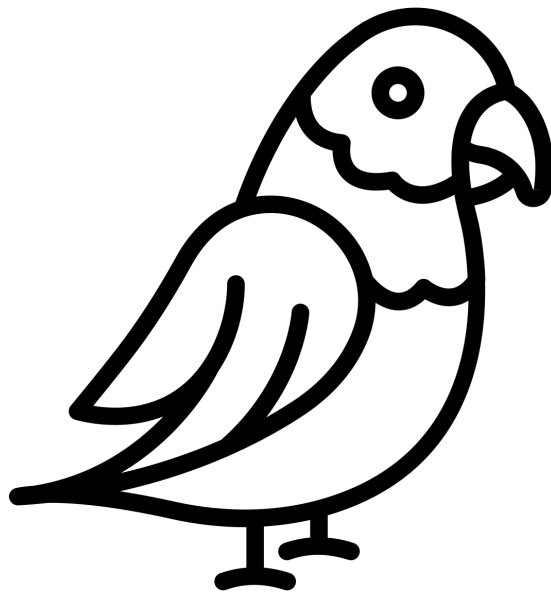


AI Prompt Basics

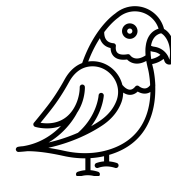
The background is a blurred image of a whiteboard in a meeting room. The whiteboard is covered with various hand-drawn diagrams, charts, and sketches. There are several rectangular boxes, some containing text or small drawings, and many lines connecting them, suggesting a flowchart or a complex diagram. The colors used in the drawings include yellow, orange, blue, and green. The overall scene is out of focus, with the foreground elements being sharper than the background. The text 'AI Prompt Basics' is overlaid on the left side of the image, in a bold, black, sans-serif font.

AI Prompt Basics

- Conversation
- Context
- Memory
- Tips

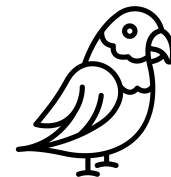


AI Prompt Basics : Conversation



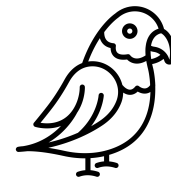
- You're interacting in "plain english"
- Responses are statistical estimates
- You're results will vary - that's a feature

AI Prompt Basics : Context



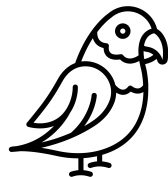
- The context window is the limit of your bot
- Your inputs and outputs fills the window
- You can save context and load it next time
- Ask your bot how much context window is left

AI Prompt Basics : Memory



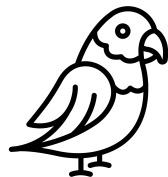
- Each conversation is a separate session
- That session is the only "memory" it has
- They won't remember anything for next time

AI Prompt Basics : Tips



- "What are my options?"
- "Do you have all you need?"
- "Ask me questions until you are confident you can perform the task."
- "What do I need to repeat this in a new conversation tomorrow?"

AI Prompt Basics : Summary



- Conversation
 - Responses are statistical estimates
- Context
 - The context window is the limit of your bot
- Memory
 - They won't remember anything for next time
- Tips
 - Ask lots of questions

Authoring API Stories



Authoring API Stories



- Stories are shared understanding
- Parts of an API Story
- Iterating on your API Story
- ***Exercise: The Task Management API Story***

Authoring API Stories : Shared Understanding



- Our brains are wired for stories
- Stories are accessible to all
- Stories are repeatable by everyone

Authoring API Stories : Story Parts

- Purpose
- Data Properties
- Resources
- Actions
- Rules



Authoring API Stories : Story Parts



Task Management

Purpose

We need to track 'Task' records in order to improve both timeliness and accuracy of customer follow-up activity.

Authoring API Stories : Story Parts



Data Properties

In this first pass at the application, we need to keep track of the following data properties:

- **id** : a globally unique value for each record [uuid]
- **title** : the text content of the record (the title of the task) [string]
- **description** : the description of the record (the task details) [text]
- **dueDate** : the date the record is due to be completed [date]
- **status** : Indicates the status of the record (active, completed) [enumerated string]
- **priority** : the priority of the task (a number between 1 and 5) [enumerated number]
- **assignedUser** : the user assigned to handle the task (a simple name string) [string]

Authoring API Stories : Story Parts



Resources

The following are resources, or states, of the API. Each state has one or more possible Actions.

- **Home** : The home or landing page of the API. Users typically start here.
 - Actions: **GetTaskList**, **GetFilteredTaskList**, **ShowHomePage**
- **TaskCollection** : The list of tasks in the system are displayed here.
 - Actions: **ShowHomePage**, **GetTaskList**, **GetFilteredTaskList**, **CreateNewTask**, **GetTaskItem**
- **TaskItem** : This resource shows a single task (selected from the TaskCollection)
 - Actions: **EditExistingTask**, **UpdateStatusOfTask**, **SetDueDateOfTask**, **AssignUserToTask**, **GetTaskList**, **GetFilteredTaskList**, **ShowHomePage**

Authoring API Stories : Story Parts



Actions

This edition of the application needs to support the following operations. Each action has zero or more input properties and always has one return value (to another state). Each action is also marked as either Safe (GET), Unsafe (POST), Idempotent (PUT/PATCH), or Delete (DELETE)

- **ShowHomePage** : Use this action to display the `home` resource
 - Inputs: None
 - Returns: **Home**
 - Type: Safe
- **GetTaskCollection** : Use this action to return a list of all Task records in the system
 - Inputs: None
 - Returns: **TaskCollection**
 - Type: Safe
- **GetTaskItem**: Use this action to get a single existing task record.
 - Inputs: id
 - Required: id
 - Returns: **TaskItem**
 - Type: Safe
- **CreateNewTask** : Use this action to add a new Task record to the system
 - Inputs: id, title, description, dueDate, status, priority, assignedUser

Authoring API Stories : Story Parts



Rules

- When executing **CreateNewTask**, the client should supply a unique `id` value.

Authoring API Stories : Story Parts



- Purpose
 - Sentence or two
- Data Properties
 - All data fields (`id`, `givenName`, etc.)
- Resources
 - All "resting states" (`home`, `list`, `item`)
- Actions
 - All actions (`save`, `share`, `filter`, etc.)
- Rules
 - Constraints (`id` must be unique, etc.)

Authoring API Stories : Iterating

- This is a multi-pass loop
- Get something now, improve later
- Know when to stop



Authoring API Stories : Exercise



Task Management

Purpose

We need to track 'Task' records in order to improve both timeliness and accuracy of customer follow-up activity.

Data Properties

In this first pass at the application, we need to keep track of the following data properties:

- **id** : a globally unique value for each record [uuid]
- **title** : the text content of the record (the title of the task) [string]
- **description** : the description of the record (the task details) [text]
- **dueDate** : the date the record is due to be completed [date]
- **status** : Indicates the status of the record (active, completed) [enumerated string]
- **priority** : the priority of the task (a number between 1 and 5) [enumerated number]
- **assignedUser** : the user assigned to handle the task (a simple name string) [string]

Resources

Authoring API Stories : Summary



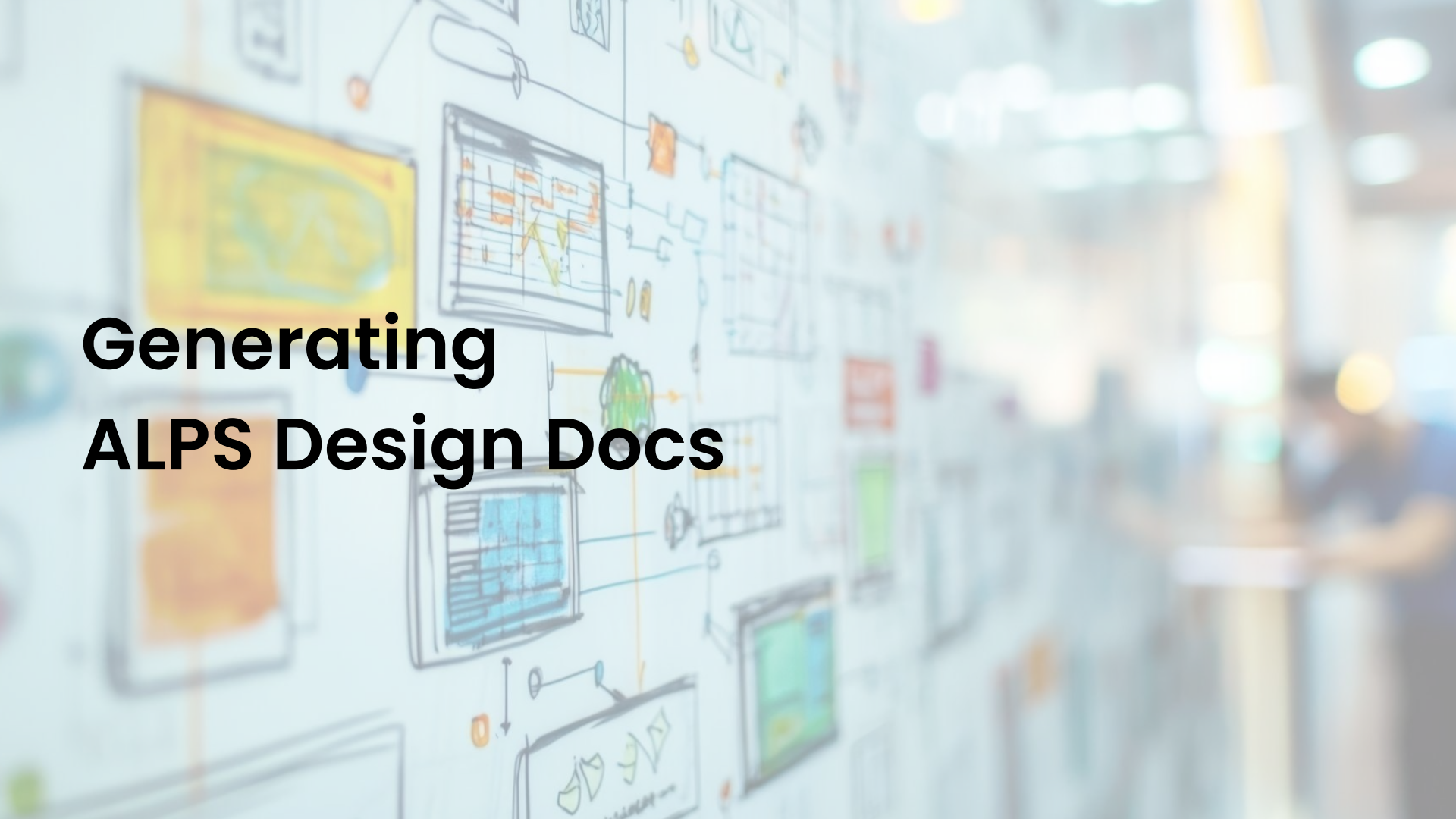
- Stories are shared understanding
 - Our brains are wired for stories
- Parts of an API Story
 - Purpose, Data Properties, Resources, Actions, Rules
- Iterating on your API Story
 - This is a multi-pass loop

AI-Driven API Design Stack

Authoring API Stories

Break





Generating ALPS Design Docs

Generating ALPS Design Docs

- Information Architecture
- Application-Level Profile Semantics
- ALPS Basics



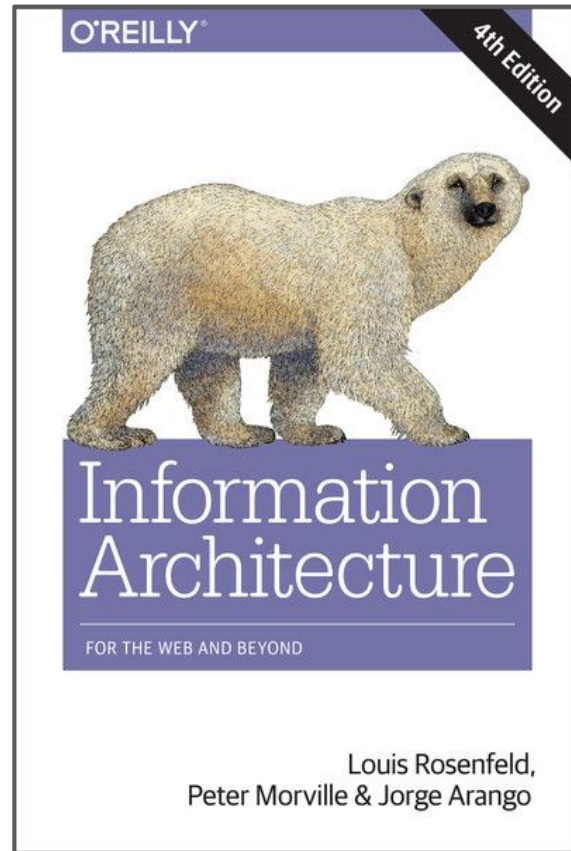
Exercise: Generating Designs with API Stories

<https://www.app-state-diagram.com/manuals/1.0/en/index.html>

Generating ALPS Design Docs

Information Architecture

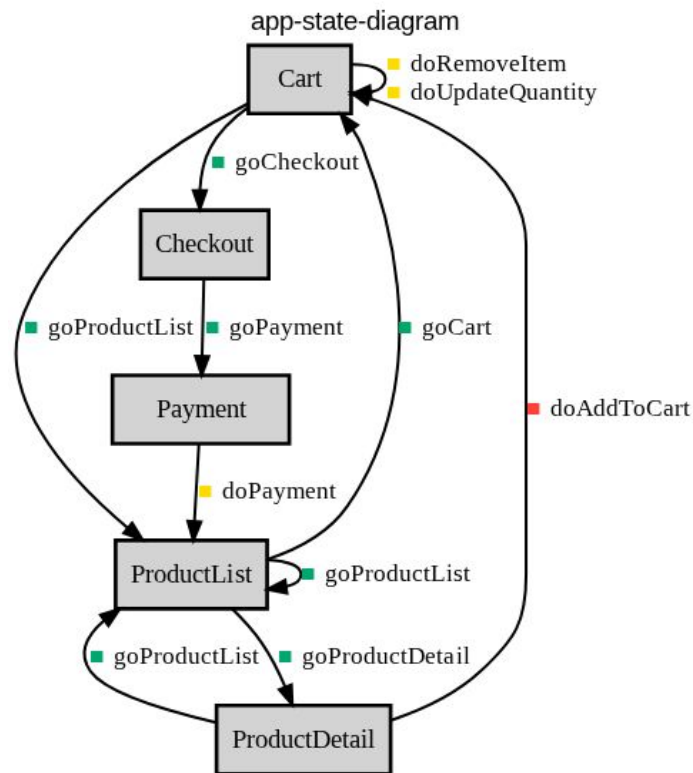
- Content, structure, and intent
- Richard Saul Wurman, Peter Morville, Lou Rosenfeld, and Don Norman
- Used in library science and web UI design
- Ontology, Taxonomy, Choreography



Generating ALPS Design Docs

Application-Level Profile Semantics

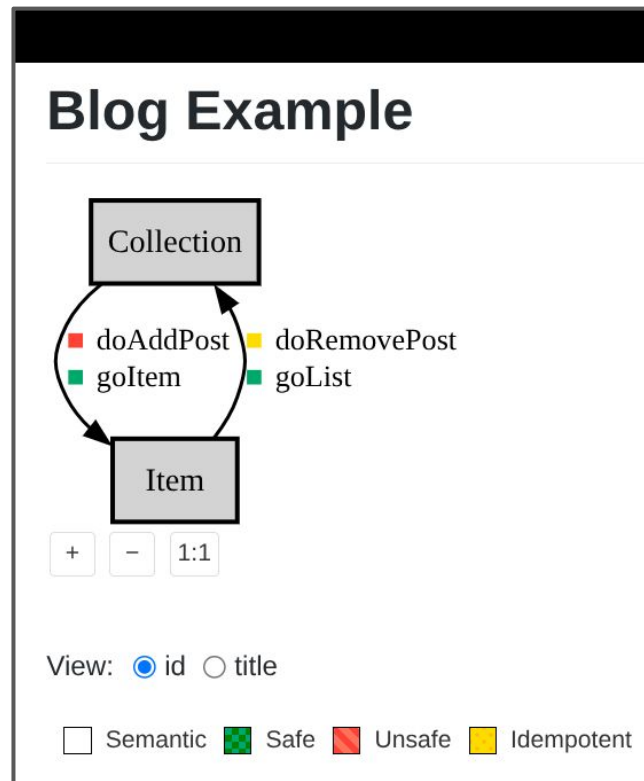
- ALPS is a format that expresses application-level meaning and structure
- Focus on Actions and Resources
- Supporting RESTful Design



Generating ALPS Design Docs

ALPS Basics

- Establish Meaning
- Identify Resources
- Describe Actions



Generating ALPS Design Docs

ALPS Basics

- **Establish Meaning**
- Identify Resources
- Describe Actions



```
{  
  "$schema": "https://alps-io.github.io/schemas/alps.json",  
  "alps": {  
    "title": "Blog Example",  
    "descriptor": [  
      {"id": "id", "title": "id"},  
      {"id": "articleBody", "title": "Content"},  
      {"id": "dateCreated", "title": "Creation Date"},  
      {"id": "BlogPost", "title": "Article", "descriptor": [  
        {"href": "#id"}, {"href": "#dateCreated"}, {"href": "#articleBody"}  
      ]  
    }  
  ]  
}
```

-

[illegible]

Generating ALPS Design Docs

ALPS Basics

- Establish Meaning
- Identify Resources
- **Describe Actions**



```
{
  "type": "safe", "id": "goList", "rt": "#Collection", "title": "View Article List",
  "type": "safe", "id": "goItem", "rt": "#Item", "title": "View Article",
    "descriptor": [{"href": "#id"}]},
  "type": "unsafe", "id": "doAddPost", "rt": "#Item", "title": "Add Blog Article",
    "descriptor": [{"href": "#BlogPost"}]},
  "type": "idempotent", "id": "doRemovePost", "rt": "#Collection",
    "title": "Remove Blog Article", "descriptor": [{"href": "#id"}]}
```

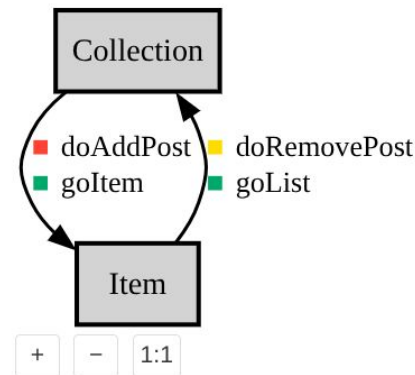
Generating ALPS Design Docs

ALPS Basics

app-state-diagram

```
1 {
2   "$schema": "https://alps-io.github.io/schemas/alps.json",
3   "alps": {
4     "title": "Blog Example",
5     "descriptor": [
6       {"id": "id", "title": "id"},
7       {"id": "articleBody", "title": "Content"},
8       {"id": "dateCreated", "title": "Creation Date"},
9       {"id": "BlogPost", "title": "Article", "descriptor": [
10        {"href": "#id"}, {"href": "#dateCreated"}, {"href": "#articleBody"}]},
11       {"id": "Collection", "title": "Article List", "descriptor": [
12        {"href": "#BlogPost"}, {"href": "#goItem"}, {"href": "#doAddPost"}]},
13       {"id": "Item", "title": "Article", "descriptor": [
14        {"href": "#BlogPost"}, {"href": "#doRemovePost"}, {"href": "#goList"}]},
15       {"type": "safe", "id": "goList", "rt": "#Collection", "title": "View Article List"},
16       {"type": "safe", "id": "goItem", "rt": "#Item", "title": "View Article",
17        "descriptor": [{"href": "#id"}]},
18       {"type": "unsafe", "id": "doAddPost", "rt": "#Item", "title": "Add Blog Article",
19        "descriptor": [{"href": "#BlogPost"}]},
20       {"type": "idempotent", "id": "doRemovePost", "rt": "#Collection",
21        "title": "Remove Blog Article", "descriptor": [{"href": "#id"}]}
22     ]
23   }
24 }
```

Blog Example



View: ☒ id ☐ title

☐ Semantic ☒ Safe ☒ Unsafe ☒ Idempotent

Generating ALPS Design Docs

Exercise : Generating Designs with API Stories

- Supply Context
- Generate ALPS
- Validate Output



ALPS

 01-alps-conversion-guidance.md 

 02-alps-conversion-checklist.md 



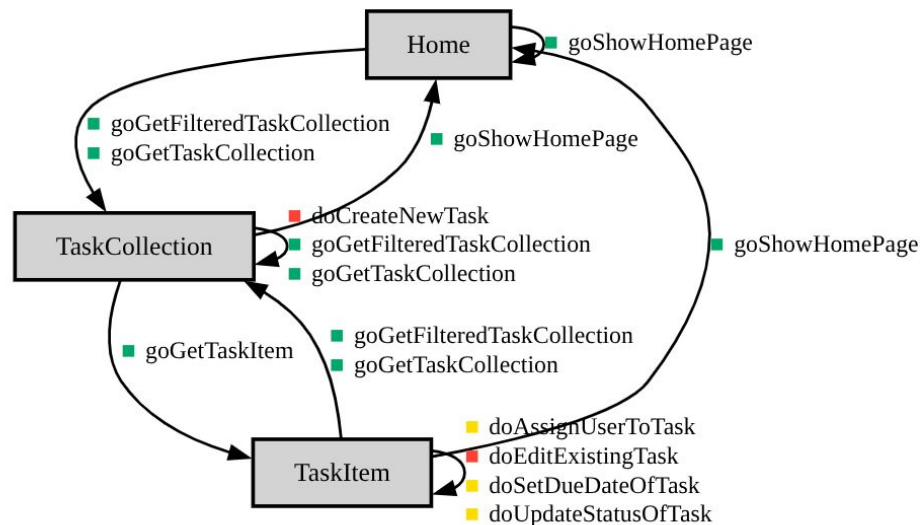
using the supplied context the api-story file, generate an ALPS design document|

  Tools

```
1 {  
2   "$schema": "https://alps-io.github.io/schemas/alps.json",  
3   "alps": {  
4     "version": "1.0",  
5     "descriptor": [  
6       {  
7         "id": "id",  
8         "type": "semantic",  
9         "title": "Identifier",  
10        "doc": {  
11          "value": "A globally unique identifier for each task record."  
12        },  
13        "tag": "task-management"  
14      },  
15      {  
16        "id": "title",  
17        "type": "semantic",  
18        "title": "Title",  
19        "doc": {  
20          "value": "Short title or name of the task."  
21        },  
22        "tag": "task-management"  
23      },  
24      {  
25        "id": "description",  
26        "type": "semantic",  
27        "title": "Description",  
28        "doc": {  
29          "value": "Detailed description of the task."  
30        },  
31        "tag": "task-management"
```

ALPS



+ - 1:1

Generating ALPS Design Docs

- Information Architecture
 - Structure: Ontology, Taxonomy, Choreography
- Application-Level Profile Semantics
 - RESTful: Focus on Actions and Resources
- ALPS Basics
 - Properties, resources, actions



AI-Driven API Design Stack

Generating API Descriptions

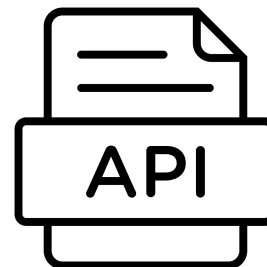
Authoring API Stories



Generating API Documentation

Generating API Documentation

- What are Docs?
- Who uses them?
- Basic Elements of API Docs



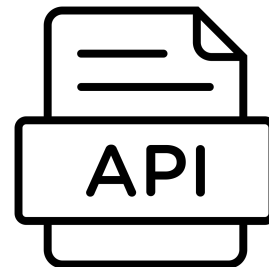
Exercise: Generate starter documentation from ALPS

<https://www.getambassador.io/blog/api-documentation-done-right-technical-guide>

Generating API Documentation

What are Docs?

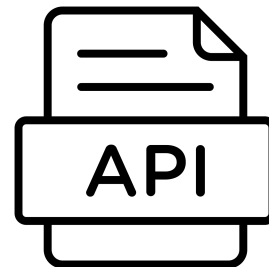
- Design Document
 - Intentions, scope, capabilities
- Implementation Plan
 - Inputs, outputs, states, protocols
- Operational Guide
 - Deployment, dependencies, monitoring, etc



Generating API Documentation

Who uses them?

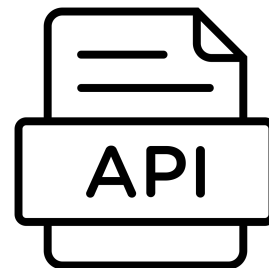
- Designers
 - What is should look like
- Programmers
 - How we should build it
- Customers
 - How we can use it
- Product Managers
 - How we can manage it



Generating API Documentation

Basic elements of API documentation

- Purpose
 - Short statement
- Data Properties
 - Name, type, description
- Resource States
 - Name, contents, links
- Action Transitions
 - Inputs, protocol details, outputs (states)
- Metadata
 - Errors, limits, security, etc.



Generating API Documentation

From ALPS design to HTML documentation



01-alps-to-html-step-by-step.md

File



02-alps-to-html-template.md

File



03-alp

File

using the context files and the task-management ALPS document, generate HTML document. Beusre to generate work working mermaid diagram, too.



Tools



Task Management API

Overview Data Fields States Transitions Diagram

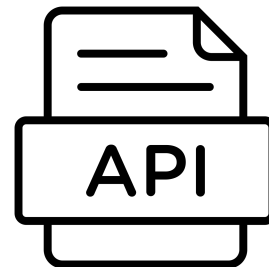
doSetDueDateOfTask	Set Due Date	idempotent	PUT	TaskItem	<pre>{ "id": "...", "dueDate": "..."} </pre>
doAssignUserToTask	Assign User	idempotent	PUT	TaskItem	<pre>{ "id": "...", "assignedUser": "..."} </pre>
goGetFilteredTaskCollection	Filter Task Collection	safe	GET	TaskCollection	?title=...&dueDate=...&status=...&priority=...&assignedUser=...

API State Diagram



Generating API Documentation : Summary

- What are Docs?
 - Design, implementation, operation
- Who uses them?
 - Developers, customers, operators
- Basic Elements of API Docs
 - Purpose, properties, resources, actions



AI-Driven API Design Stack

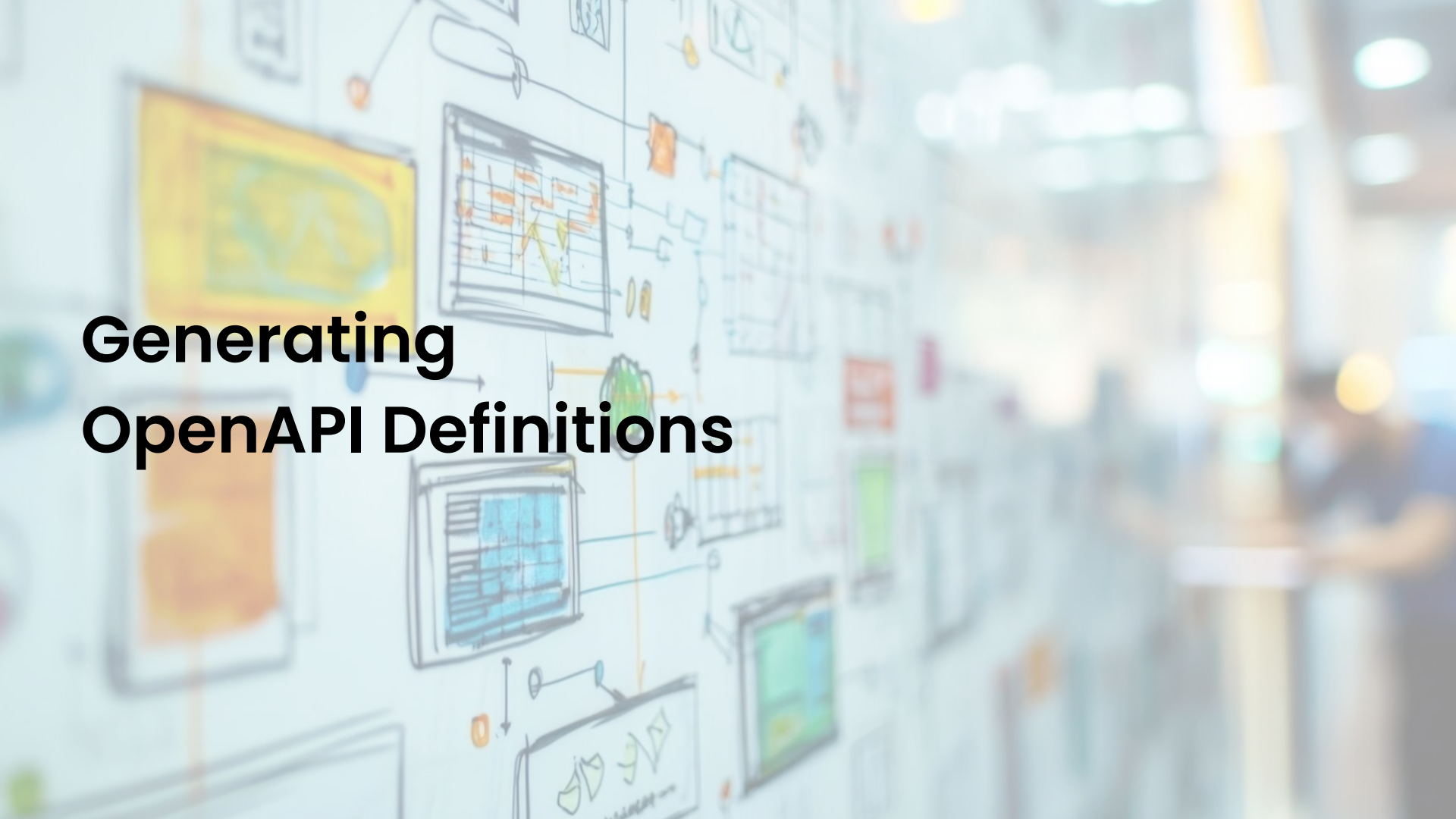
Generating API Documentation

Generating API Descriptions

Authoring API Stories

Break





Generating OpenAPI Definitions

Generating Open API Documents

Why Open API

- The OpenAPI Specifications provide a formal standard for describing HTTP APIs.
- Common way to describe implementations
- Common platform for tooling
- Shared practice for building



<https://www.openapis.org/what-is-openapi>

Generating Open API Documents

Who Uses it?

- **Designers**
 - Produces API specifications
- **Builders**
 - Produces working code
- **Testers**
 - Produces validation
- **Consumers**
 - Produces useful solutions



Generating Open API Documents

Open API Basics

- **Basic Info**
- Component Section
- Paths Section



```
1 openapi: 3.0.0
2 info:
3   version: '1.0'
4   title: Person API
5   description: An example API for the "Designing APIs with OpenAPI" workshop
6 servers:
7   # Added by API Auto Mocking Plugin
8   - description: SwaggerHub API Auto Mocking
9     url: https://virtserver.swaggerhub.com/amundsen/Person-API/1.0
10
11 paths: {}
12 components: {}
```

Generating Open API Documents

Open API Basics



- Basic Info
- **Component Section**
- Paths Section

```
parameters:  
  nameParam:  
    name: name  
    in: query  
    description:  
    required:  
    schema:  
      type: string  
    example:
```

```
PersonItem:  
  type: object  
  properties:  
    id:  
      type: string  
      description:  
      example: "q1"  
    name:  
      type: string  
      description:  
      example: "Morton Muffley"  
    email:  
      type: string  
      description: "Default email for"  
      example: "morton.muffley@example"
```

```
PersonCollection:  
  type: array  
  items:  
    $ref: '#/components/schemas/PersonItem'
```

```
responses:  
  GenericError:  
    description: "  
    content:  
      application/  
      schema:  
        $ref: "#"
```

```
requestBodies:  
  AddPerson:  
    description: "body for a new Person record"  
    required: true  
    content:  
      application/json:  
        schema:  
          $ref: "#/components/schemas/PersonItem"  
      application/x-www-urlencoded:  
        schema:  
          $ref: "#/components/schemas/PersonItem"
```

Generating Open API Documents

Open API Basics

- Basic Info
- Component Section
- **Paths Section**

```
### work in progress record ###  
/wip/{identifier}:  
  get:  
    operationId: wipItem  
    summary: Use this to return a single onboarding record  
    tags:  
      - onboarding  
  
    parameters:  
      - $ref: "#/components/parameters/identifier"  
      - $ref: "#/components/parameters/eTag"  
  
    responses:  
      '200':  
        $ref: "#/components/responses/reply"  
  
    default:  
      $ref: "#/components/responses/error"
```

Generating Open API Documents

From ALPS to OpenAPI



tasks-alps-profile.json.txt

Document

attached are the context files for generating OpenAPI from ALPS

✅ The OpenAPI 3.1 YAML specification has been generated from the ALPS profile, using all context files and mapping rules.

You can download it here:

 [tasks-openapi.yaml](#)

Let me know if you'd like a validation report, HTML rendering, or further transformations. [\[>\]](#)



Ask anything

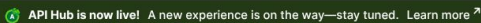
+ 🔧 Tools



Generating Open API Documents

From ALPS to OpenAPI

 **API Hub** | Design
Powered by Swagger



[task-management_api](#) | Version: 1.0.0 | [Export](#)

[Save to Hub](#) | [+](#) | [⚙](#)

```
1 openapi: 3.1.0
2 info:
3   title: Task Management API
4   version: 1.0.0
5   description: OpenAPI specification generated from ALPS
6     profile for task tracking.
7 paths:
8   /goShowHomePage:
9     get:
10       operationId: goShowHomePage
11       summary: Go to Home
12       description: ''
13       responses:
14         '200':
15           description: Successful operation
16           content:
17             application/json:
18               schema:
19                 $ref: '#/components/schemas/Home'
20         '400':
21           description: Invalid input
22         '404':
23           description: Resource not found
24       tags:
25         - navigation
26       parameters: []
27   /goGetTaskCollection:
28     get:
29       operationId: goGetTaskCollection
30       summary: Get Task Collection
31       description: ''
32       responses:
```

navigation

GET	/goShowHomePage	Go to Home
-----	-----------------	------------

task-management

GET	/goGetTaskCollection	Get Task Collection
GET	/goGetTaskItem	Get Task Item
POST	/doCreateNewTask	Create Task
POST	/doEditExistingTask	Edit Task
PUT	/doUpdateStatusOfTask	Update Status
PUT	/doSetDueDateOfTask	Set Due Date
PUT	/doAssignUserToTask	Assign User
GET	/goGetFilteredTaskCollection	Filter Task Collection



Generating Open API Documents : Summary



- Why OpenAPI
 - Common platform for describing HTTP APIs
- Who Uses it?
 - Designers, builders, testers, consumers
- OpenAPI Basics
 - Info, components, paths

AI-Driven API Design Stack

Generating OpenAPI

Generating API Documentation

Generating API Descriptions

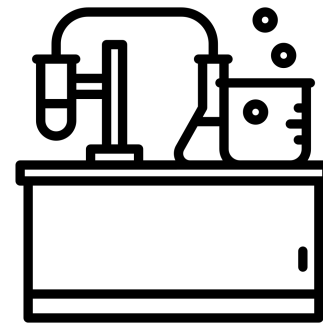
Authoring API Stories

Generating Running Prototypes

The background is a blurred image of a whiteboard or wall covered in various hand-drawn diagrams, charts, and sketches. The drawings include flowcharts, tables, and abstract shapes, all in various colors like yellow, orange, blue, and green. The overall scene suggests a creative or design process, possibly related to the text 'Generating Running Prototypes'.

Generating Running Prototypes

- What is a Prototype?
- Why use them?
- Prototype Essentials



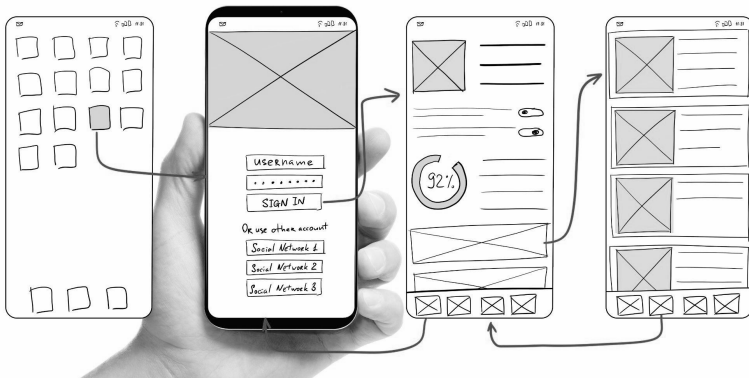
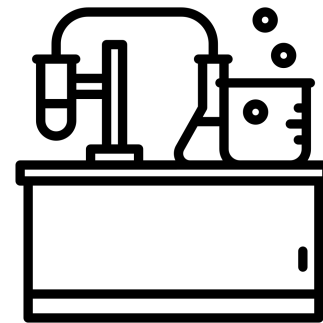
Exercise: Generate a NodeJS prototype from ALPS

Generating Running Prototypes

What is a prototype?

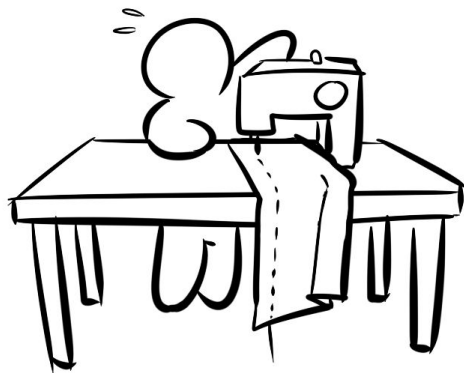
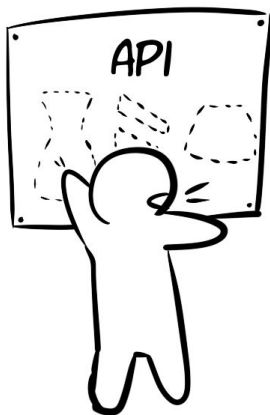
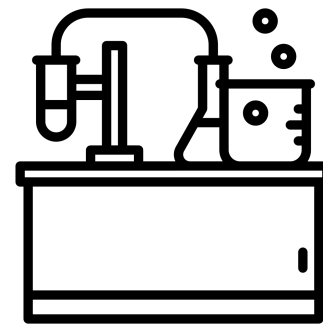
A first, typical or preliminary model of something, especially a machine, from which other forms are developed or copied.

-- Oxford Dictionary



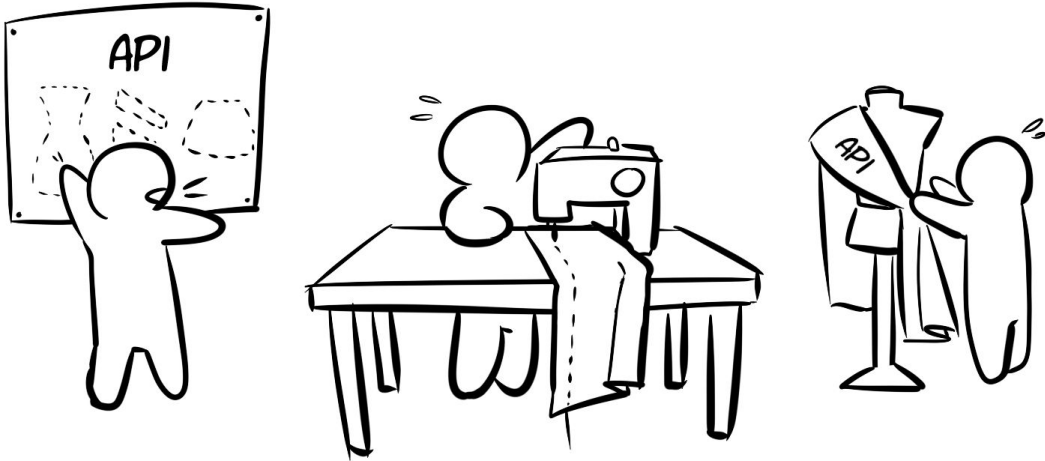
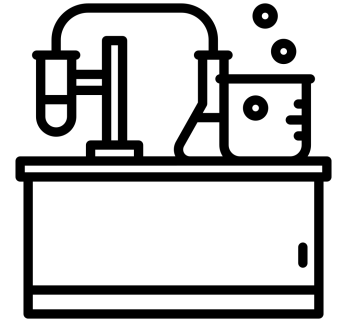
Generating Running Prototypes

- Inexpensive experiments
- Explore the details
- Prototypes are made to be tested



Generating Running Prototypes

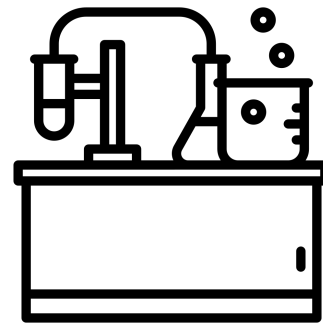
- Inexpensive experiments
- Explore the details
- *Prototypes are made to be tested*



Generating Running Prototypes

Prototype Essentials

- Running Code
- Rough Consensus
- Throw it away



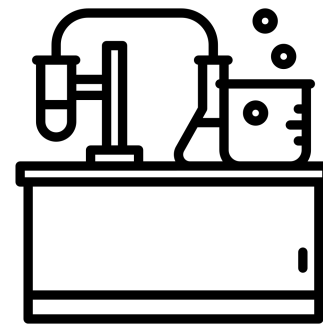
Code

Blame

130 lines (116 loc) · 4.26 KB

```
30  function generateLinks(person) {
92  const index = persons.findIndex(p => p.id === req.params.id);
93  if (index === -1) return res.status(404).json({ error: 'Person not found' });
94  persons[index] = { ...persons[index], ...person };
95  res.json({ ...persons[index], _links: generateLinks(persons[index]) });
96  });
97
98  app.put('/persons/:id/status', (req, res) => {
99  const { person } = req.body;
100  if (!person || !person.status) return res.status(400).json({ error: 'Missing required field: status' });
101  const record = persons.find(p => p.id === req.params.id);
102  if (!record) return res.status(404).json({ error: 'Person not found' });
103  record.status = person.status.toLowerCase();
104  if (!['active', 'inactive'].includes(record.status)) {
105  return res.status(400).json({ error: 'Invalid status value' });
```

Generating Running Prototypes



01-api-template-notes-v2.md

File



02-api-checklist-v2.md

File



02-api

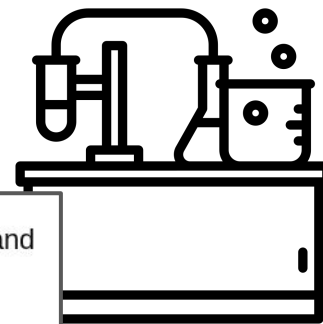
File

attached are all the context files and the ALPS document for generating a simple index.js server

+  Tools



Generating Running Prototypes



✅ The Node.js Express API server has been generated from the ALPS profile using the provided context and conventions. It includes:

- In-memory data
- Fully RESTful endpoints
- HAL-style `_links`
- Proper transition handling (GET, POST, PUT)
- Filtering and validation logic

You can download the complete `index.js` file here:

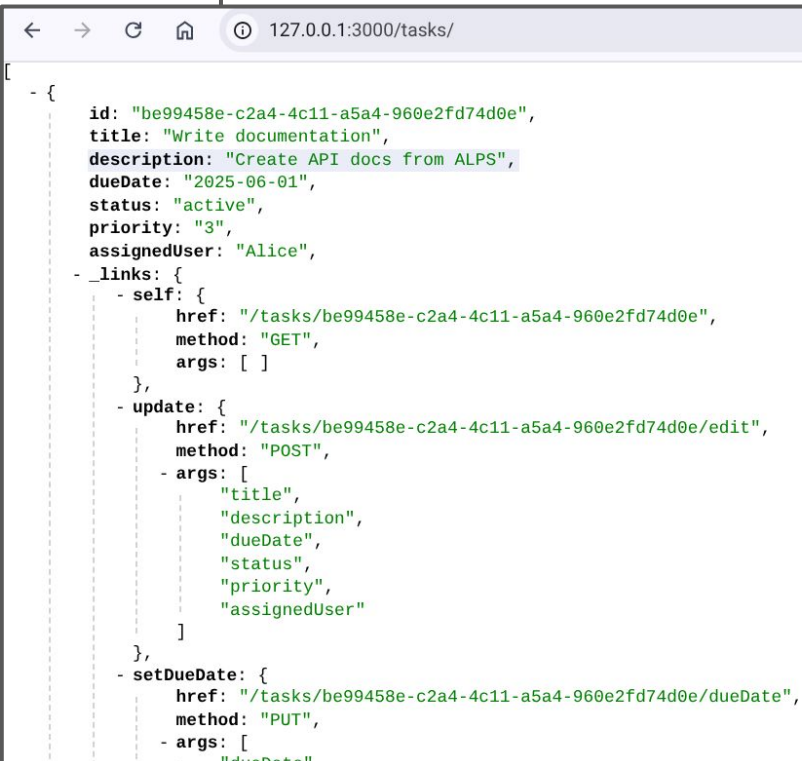
 [index.js](#)

Let me know if you'd like to test it, add more sample data, or wrap it in a Docker container. [\[>-\]](#)



Generating Running Prototypes

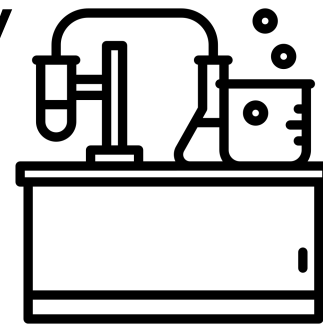
```
43 // Root endpoint
44 app.get('/', (req, res) => {
45   res.json({
46     _links: {
47       tasks: { href: "/tasks", method: "GET", args: [] },
48       create: { href: "/tasks", method: "POST", args: ["id"] }
49     }
50   });
51 });
52
53 // GET all tasks or filter
54 app.get('/tasks', (req, res) => {
55   const filters = req.query;
56   let filtered = tasks.filter(task =>
57     Object.entries(filters).every(([key, value]) =>
58       (task[key] || "").toLowerCase() === value.toLowerCase()
59     )
60   );
61   res.json(filtered.map(task => ({ ...task, _links: generateLinks(task) })));
62 });
63
64 // GET single task
65 app.get('/tasks/:id', (req, res) => {
66   const task = tasks.find(t => t.id === req.params.id);
67   if (!task) return res.status(404).json({ error: "Task not found" });
68   res.json({ ...task, _links: generateLinks(task) });
69 });
70
71 // POST create new task
72 app.post('/tasks', (req, res) => {
73   const input = req.body.task;
```



The screenshot shows a web browser window with the address bar displaying `127.0.0.1:3000/tasks/`. The browser content shows a JSON response for a task. The response is a JSON object with the following fields: `id`, `title`, `description`, `dueDate`, `status`, `priority`, `assignedUser`, and `_links`. The `_links` field contains a `self` link and an `update` link. The `self` link has a `href` of `/tasks/be99458e-c2a4-4c11-a5a4-960e2fd74d0e`, a `method` of `GET`, and `args` of `[]`. The `update` link has a `href` of `/tasks/be99458e-c2a4-4c11-a5a4-960e2fd74d0e/edit`, a `method` of `POST`, and `args` of `["title", "description", "dueDate", "status", "priority", "assignedUser"]`. The `setDueDate` link has a `href` of `/tasks/be99458e-c2a4-4c11-a5a4-960e2fd74d0e/dueDate`, a `method` of `PUT`, and `args` of `["dueDate"]`.

```
{
  id: "be99458e-c2a4-4c11-a5a4-960e2fd74d0e",
  title: "Write documentation",
  description: "Create API docs from ALPS",
  dueDate: "2025-06-01",
  status: "active",
  priority: "3",
  assignedUser: "Alice",
  _links: {
    self: {
      href: "/tasks/be99458e-c2a4-4c11-a5a4-960e2fd74d0e",
      method: "GET",
      args: [ ]
    },
    update: {
      href: "/tasks/be99458e-c2a4-4c11-a5a4-960e2fd74d0e/edit",
      method: "POST",
      args: [
        "title",
        "description",
        "dueDate",
        "status",
        "priority",
        "assignedUser"
      ]
    },
    setDueDate: {
      href: "/tasks/be99458e-c2a4-4c11-a5a4-960e2fd74d0e/dueDate",
      method: "PUT",
      args: [
        "dueDate"
      ]
    }
  }
}
```

Generating Running Prototypes : Summary



- What is a Prototype?
 - A preliminary model
- Why use them?
 - Prototypes are made to be tested
- Prototype Essentials
 - Rough consensus and running code

Exercise: Generate a NodeJS prototype from ALPS

AI-Driven API Design Stack

Generating Running API Prototypes

Generating OpenAPI

Generating API Documentation

Generating API Descriptions

Authoring API Stories

Break



Generating API Tests



Generating API Tests

- "Outside-In" testing
- Validating inputs/outputs
- Confirming Interface Behavior



Exercise: Generating API Tests from source code

Generating API Tests

Outside-In Testing

- Consumers can only "see" the API
- The aim is to test the interface, not the code
- Code may change, but the tests do not



Generating API Tests

Validating inputs/outputs

- Confirm the URL exists and responds as expected
- Validate the methods, input parameters
- Inspect the response bodies



Generating API Tests

Confirm interface behavior

- Validate expected results
- Happy-path tests (200 OK)
- Sad-path tests (400 Bad Request)



Generating API Tests

Exercise: Generates Tests from Code



00-nodejs-to-test.md

File



01-setup-tests.md

File



02-inc

File

use the supplied context documents including the attached index.js to generate a set of API tests using the Jest framework. Be sure to run a verification check list after you produce the HTML report.

+  Tools



Generating API Tests

Confirm test file

```
41 // @alps-route:GET /tasks
42 // @link-check
43 test('GET /tasks returns tasks with links', async () => {
44   const res = await request(app).get('/tasks');
45   expect(res.statusCode).toBe(200);
46   expect(Array.isArray(res.body)).toBe(true);
47   expect(res.body[0]._links).toBeDefined();
48 });
49
50 // @alps-route:GET /tasks/:id
51 // @error-case
52 test('GET /tasks/:id returns 404 for invalid id', async () => {
53   const res = await request(app).get('/tasks/invalid');
54   expect(res.statusCode).toBe(404);
55 });
56
57 // @alps-route:POST /tasks
58 test('POST /tasks creates task', async () => {
59   const newTask = {
60     id: uuidv4(),
61     title: "New",
62     description: "Test",
63     dueDate: "2025-06-15",
64     status: "active",
65     priority: "1",
66     assignedUser: "Test"
67   };
68   const res = await request(app).post('/tasks').send({ task: newTask });
69   expect(res.statusCode).toBe(201);
70   expect(res.body._links).toBeDefined();
71 });
72
```

Generating API Tests

Modify running code

```
1 # 🛠️ API Test Setup Guide for ALPS-Based Express Server
2
3 This guide walks you through all the necessary steps to prepare and run Jest
4 generated from an ALPS document.
5 ---
6
7 ## ✅ Prerequisites
8
9 Make sure the following are installed:
10
11 - **Node.js** (v18+)
12 - **npm** (v9+)
13 - **Jest** and **Supertest**
14
15 ---
16
17 ## 📁 Project Structure
18
19 Expected folder layout:
20
21 ```
22 /project-root
23 |   index.js           ← Express API server
24 |   test/
25 |   |   api.test.js    ← Jest test suite
26 |   |   package.json
27 |   ```
28
29 ---
30
31 ## 1. 🛠️ Modify `index.js` for Testing Support
32
33 Add a test hook to reset in-memory data. Just after defining `let tasks = [
```

Generating API Tests

Run test suite

```
mca@mamund-ws:.../task-management$ npm test

> task-management@1.0.0 test
> jest

PASS test/api.test.js
  Task API
    ✓ GET / returns HAL links (29 ms)
    ✓ GET /tasks returns tasks with links (4 ms)
    ✓ GET /tasks/:id returns 404 for invalid id (3 ms)
    ✓ POST /tasks creates task (603 ms)
    ✓ POST /tasks/:id/edit modifies a task (6 ms)
    ✓ PUT /tasks/:id/status changes status (5 ms)
    ✓ PUT /tasks/:id/dueDate sets dueDate (8 ms)
    ✓ PUT /tasks/:id/assign sets assignedUser (5 ms)

Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        3.397 s
Run all test suites
```

Generating API Tests : Summary

- "Outside-In" testing
 - Test the interface, not the code
- Validating inputs/outputs
 - URLs, methods, inputs, response bodies
- Confirming Interface Behavior
 - Happy path (200 OK) & sad path (400 Bad Request)



AI-Driven API Design Stack

Generating API Tests

Generating Running API Prototypes

Generating OpenAPI

Generating API Documentation

Generating API Descriptions

Authoring API Stories



Generating Security Profiles

Generating Security Profiles

Retro-fitting the process

- API Security Basics
- RBAC to the rescue
- Modifying Existing Assets



Exercise: Update Story, Generate Profile, regenerate, other assets

Generating Security Profiles

API Security Basics

- Encryption
 - HTTPS
- API Key
 - Controls access to the API
- Authentication
 - Identifies user
- Authorization
 - Defines access control



Generating Security Profiles

RBAC to the rescue



- Role-Based Access Control
 - RBAC
- Define Roles
 - anon, user, admin, etc
- Assign users to one or more roles
 - user:mike, roles:user, admin
- Secure API resources and actions
 - /tasks/, /tasks/:id, addTask, updateStatus, etc.

Generating Security Profiles

Exercise: Retrofit process



- Update API Story
 - Define roles, apply to resources & actions
- Generate Profile Report
 - New generation action
- Regenerate Assets
 - HTML documentation
 - ALPS
 - OpenAPI
 - NodeJS
 - Tests

Generating Security Profiles

Exercise: Update API Story



```
* AllowedRoles: user, admin
* **SetDueDateOfTask** : Use this action to set the `dueDate` of a single r
* Inputs: id, dueDate
* Required: id, dueDate
* Returns: **TaskItem**
* Type: Idempotent
* AllowedRoles: admin
* **AssignUserToTask** : Use this action to update the name of the `assigne
* Inputs: id, assignedUser
* Required: id, assignedUser
* Returns: **TaskItem**
* Type: Idempotent
* AllowedRoles: admin
* **GetFilteredTaskCollection** : Use this action to filter the list of tas
`priority` and/or `assignedUser`
* Inputs: title, dueDate,status, priority
* Returns: **TaskCollection**
* Type: Safe
* AllowedRoles: user, admin
```

Roles

These are the RBAC security roles for this API.

- * anon : Unauthenticated visitor (read-only, limited views)
- * user : Authenticated end-user with basic task interaction privileges
- * admin : Elevated privileges for managing tasks and assignments

Generating Security Profiles

Run security context process



00-rbac-system-prompt.md

File



01-rbac-style-guide.md

File



02-rbac-style-guide.md

File

using the task-api-story-security.md document, generate an API Security Profile Report



Generating Security Profiles

Run security context process

**00-rbac-system-prompt.md**
File



**01-rbac**
File

using the task-api-story-security.md document, generate

+

Security Profile

Declared Roles

Role	Description
anon	Anonymous (unauthenticated user)
user	Authenticated standard user
admin	Authenticated administrative user

Resource Access Matrix

Resource	anon	user	admin
Home	✓	✓	✓
TaskCollection		✓	✓
TaskItem		✓	✓

Action Access Matrix

Action	Type	anon	user	admin
ShowHomePage	Safe	✓	✓	✓
GetTaskCollection	Safe		✓	✓
GetTaskItem	Safe			

Generating Security Profiles

Retro-fitting the process



- API Security Basics
 - Encryption, API Key, Authentication, Authorization
- RBAC to the rescue
 - Role-based access control
- Modifying Existing Assets
 - Generate new report, backfill existing assets

AI-Driven API Design Stack

Generating Security Profiles

Generating API Tests

Generating Running API Prototypes

Generating OpenAPI

Generating API Documentation

Generating API Descriptions

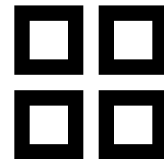
Authoring API Stories

Course Review



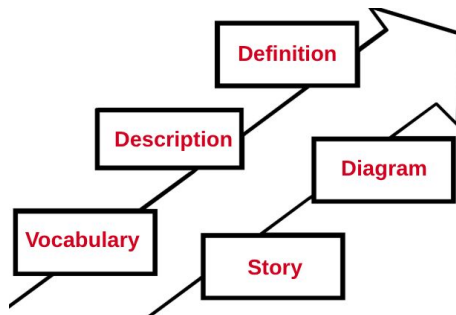
Course Review

- API Design Basics
- AI Prompt Basics
- Authoring API Stories
- Generating API Descriptions
- Generating API Documentation
- Generating OpenAPI Documents
- Generating Running API Prototypes
- Generating API Tests
- Generating Security Profiles

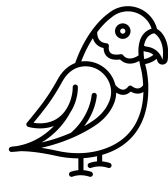


API Design Basics: Review

- Story
 - Stories are shared understanding
- Vocabulary
 - All the "magic words" related to the API domain
- Diagram
 - Visual representation of the API
- Description
 - The complete design in one place, in one voice.
- Definition
 - Translate design into implementation



AI Prompt Basics : Review



- Conversation
 - Responses are statistical estimates
- Context
 - The context window is the limit of your bot
- Memory
 - They won't remember anything for next time
- Tips
 - Ask lots of questions

Authoring API Stories : Review



- Stories are shared understanding
 - Our brains are wired for stories
- Parts of an API Story
 - Purpose, Data Properties, Resources, Actions, Rules
- Iterating on your API Story
 - This is a multi-pass loop

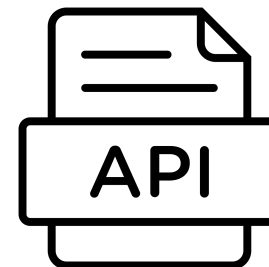
Generating ALPS Design Docs : Review

- Information Architecture
 - Structure: Ontology, Taxonomy, Choreography
- Application-Level Profile Semantics
 - RESTful: Focus on Actions and Resources
- ALPS Basics
 - Properties, resources, actions



Generating API Documentation : Review

- What are Docs?
 - Design, implementation, operation
- Who uses them?
 - Developers, customers, operators
- Basic Elements of API Docs
 - Purpose, properties, resources, actions



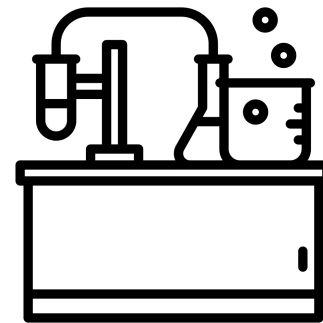
Generating Open API Documents : Review



- Why OpenAPI
 - Common platform for describing HTTP APIs
- Who Uses it?
 - Designers, builders, testers, consumers
- OpenAPI Basics
 - Info, components, paths

Generating Running Prototypes : Review

- What is a Prototype?
 - A preliminary model
- Why use them?
 - Prototypes are made to be tested
- Prototype Essentials
 - Rough consensus and running code



Generating API Tests : Review

- "Outside-In" testing
 - Test the interface, not the code
- Validating inputs/outputs
 - URLs, methods, inputs, response bodies
- Confirming Interface Behavior
 - Happy path (200 OK) & sad path (400 Bad Request)



Generating Security Profiles : Review

Retro-fitting the process



- API Security Basics
 - Encryption, API Key, Authentication, Authorization
- RBAC to the rescue
 - Role-based access control
- Modifying Existing Assets
 - Generate new report, backfill existing assets

Finally ...



Andrej Karpathy ✓

@karpathy



There's a new kind of coding I call "vibe coding", where you fully give in to the vibes, embrace exponentials, and forget that the code even exists. It's possible because the LLMs (e.g. Cursor Composer w Sonnet) are getting too good. Also I just talk to Composer with SuperWhisper so I barely even touch the keyboard. I ask for the dumbest things like "decrease the padding on the sidebar by half" because I'm too lazy to find it. I "Accept All" always, I don't read the diffs anymore. When I get error messages I just copy paste them in with no comment, usually that fixes it. The code grows beyond my usual comprehension, I'd have to really read through it for a while. Sometimes the LLMs can't fix a bug so I just work around it or ask for random changes until it goes away. It's not too bad for throwaway weekend projects, but still quite amusing. I'm building a project or webapp, but it's not really coding - I just see stuff, say stuff, run stuff, and copy paste stuff, and it mostly works.

6:17 PM · Feb 2, 2025 · **4.9M** Views



1.3K



5K



29K



15K



Andrej Karpathy

"Ultimately, vibe coding full web apps today is kind of messy and not a good idea for anything of actual importance."

— Andrej Karpathy



Q&A

The background is a blurred image of a whiteboard in a meeting room. The whiteboard is covered with various hand-drawn diagrams, charts, and notes. There are several rectangular boxes, some containing text, and others containing diagrams. A prominent diagram in the center shows a flowchart with arrows connecting different components. There are also some circular diagrams and a large rectangular box on the left side. The overall scene suggests a collaborative workspace or a meeting where ideas are being discussed and recorded.



AI-Driven API Design

Mike Amundsen (@mamund)

<https://github.com/mamund/2026-08-ai-driven-workshop>

<https://b.mamund.com/4gETA7f>