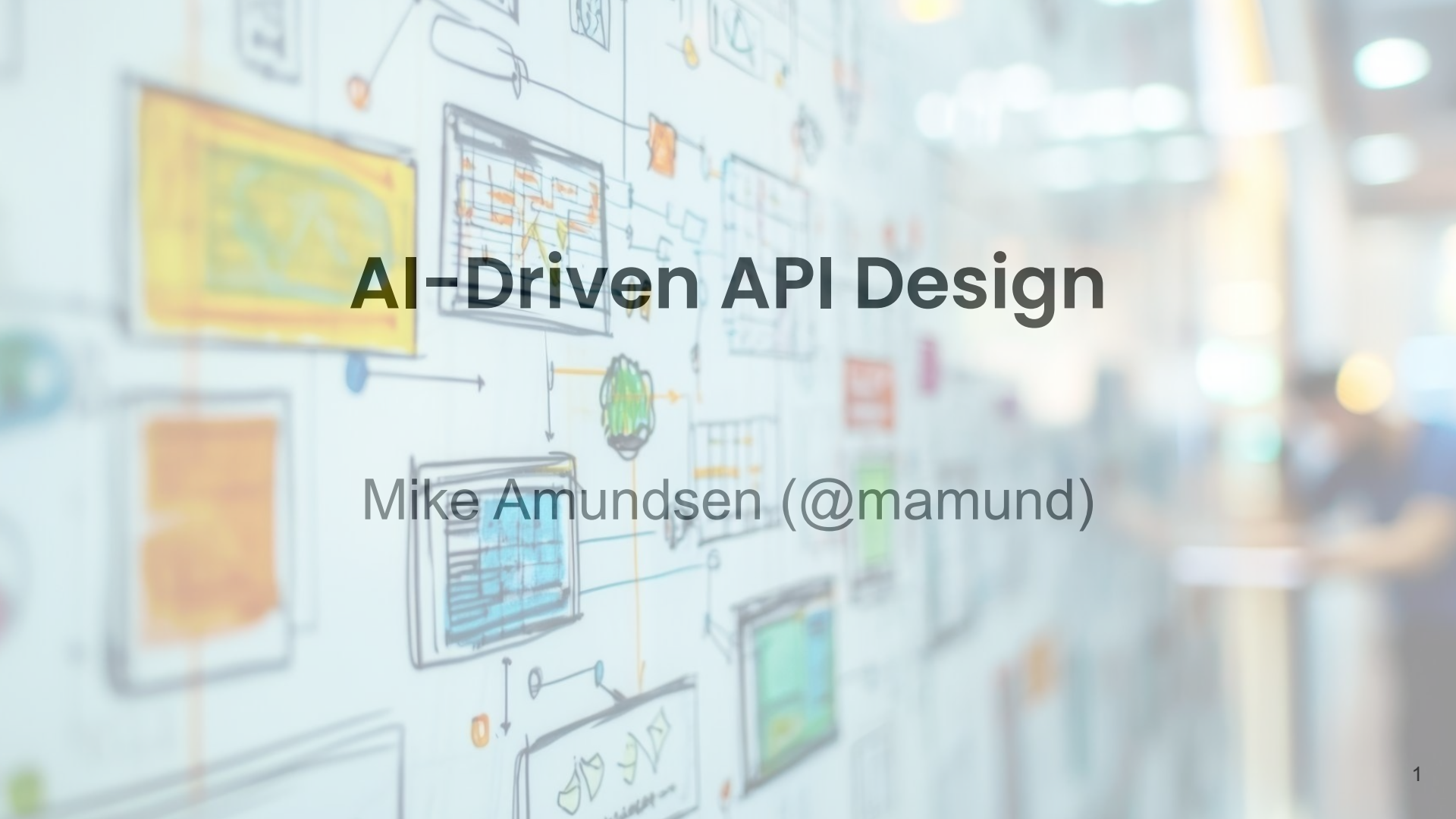


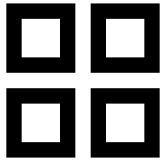
AI-Driven API Design

Mike Amundsen (@mamund)


Mike Amundsen
@mamund

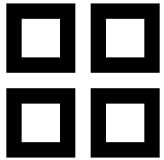
Introduction

Course schedule



- Course Introduction
- API Design Basics
- AI Prompt Basics
- Authoring API Stories
- *BREAK (5min)*
- Generating API Descriptions
- Generating API Documentation
- *BREAK (5min)*
- Generating OpenAPI Documents
- Generating Running API Prototypes
- *BREAK (5min)*
- 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/2025-12-apiconf-ai-driven-api-design

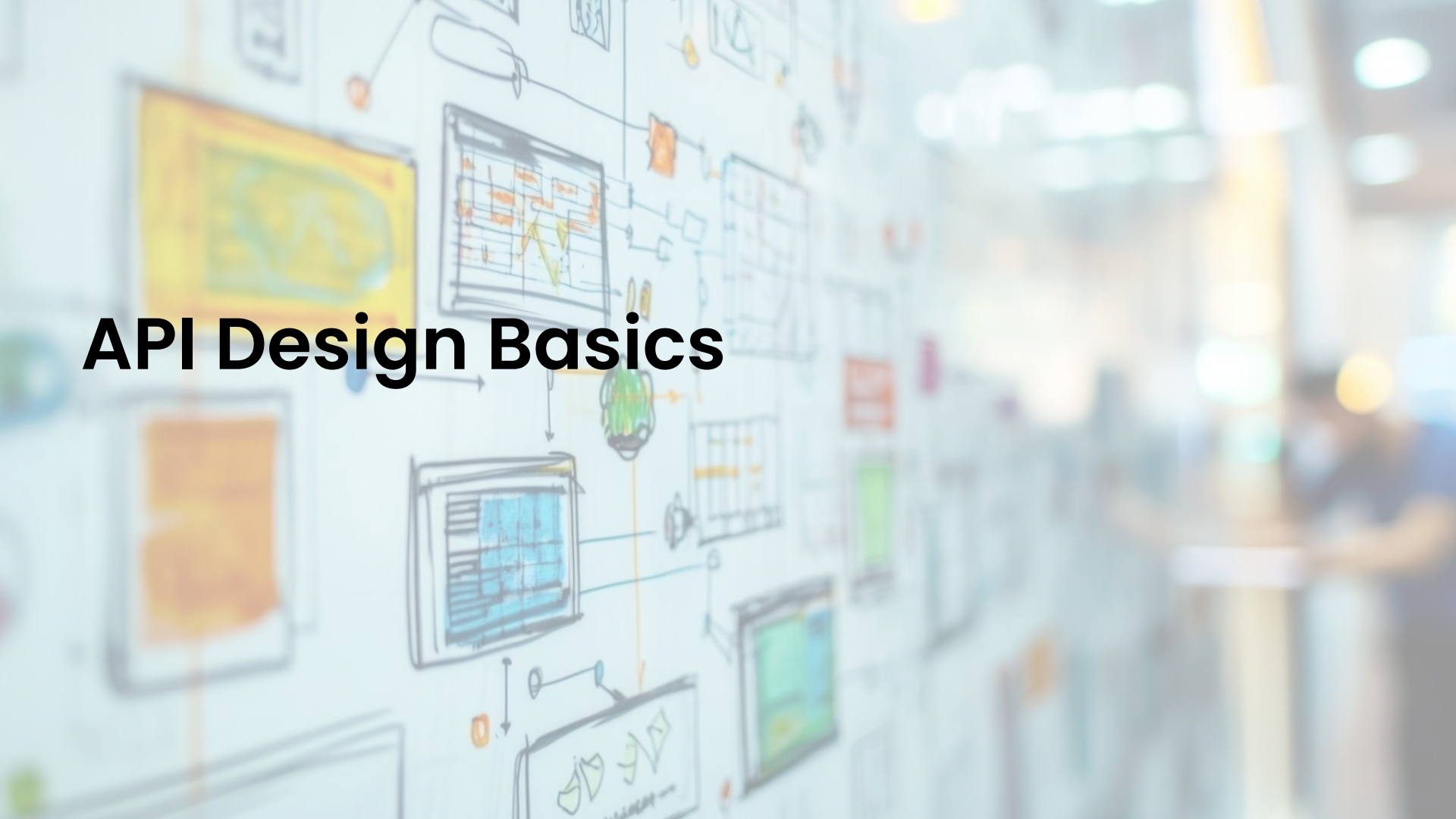
https://b.mamund.com/4oVcet8

Poll: AI Awareness

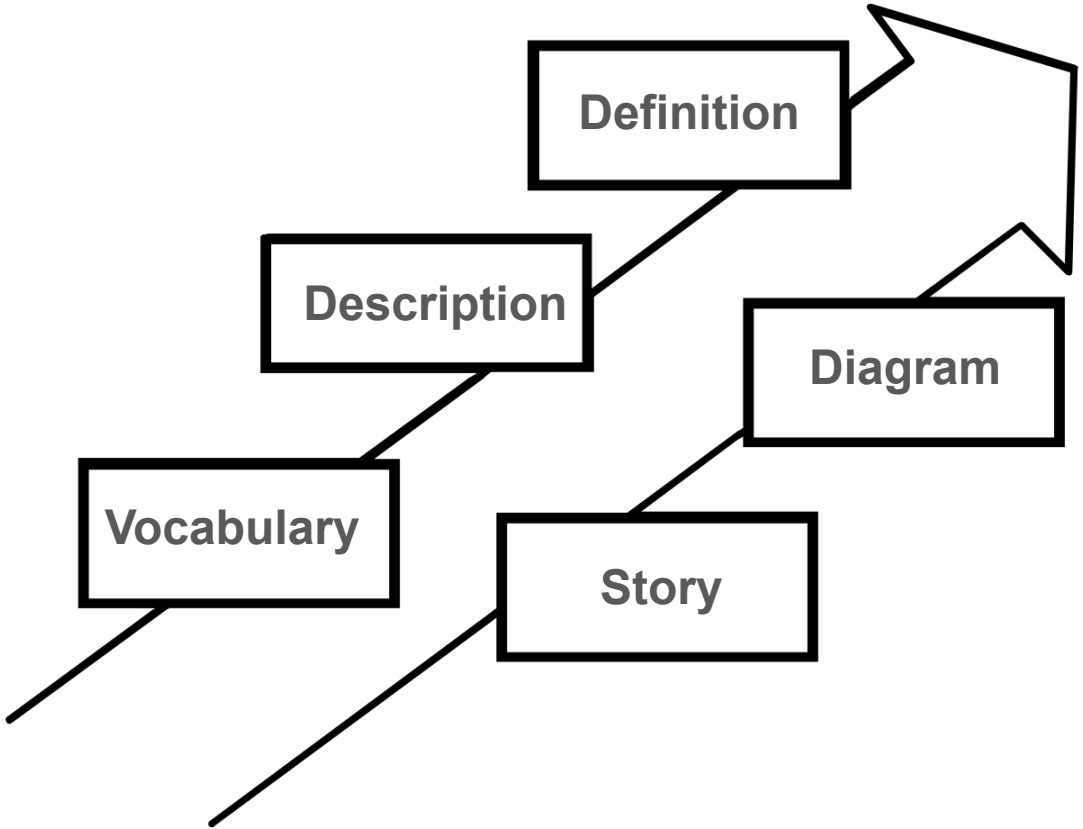
How would you characterize your "AI Awareness" level?

- **Novice**: Just starting to explore AI tools
- **Advanced Beginner**: I know my way around
- **Competent**: I can use AI to get things done
- **Proficient**: I am comfortable experimenting with advanced AI tools
- **Expert**: I am able to help others learn to use AI.

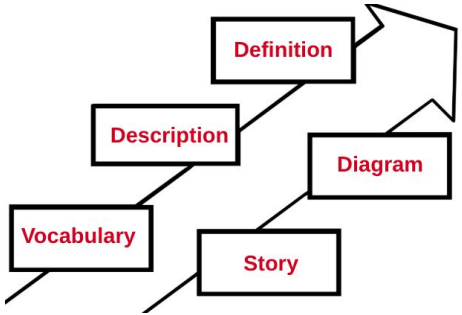
API Design Basics



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

https://www.youtube.com/watch?v=DZIG9U4AmEE

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

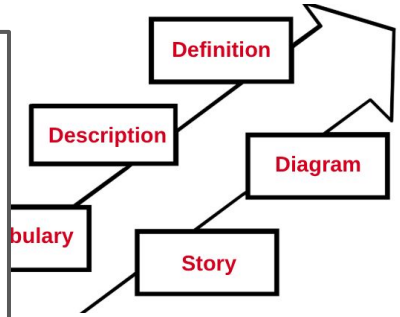
In this first pass at the application, we need to keep track of the following data properties:

- **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]

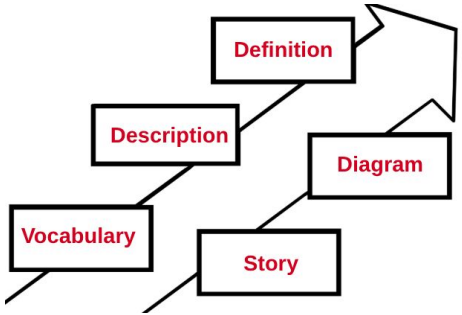
Resources

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



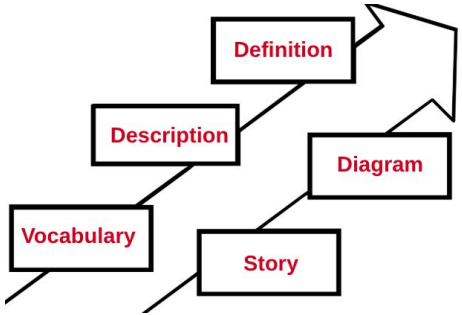
- All the "magic words" related to the domain
- Not just the properties:
  - `id`, `givenName`, etc.
- Also :
  - Actions (`create`, `approve`, `save`, etc.)
  - Enumerators (`status.active`, `status.inactive`)
  - Containers (`item.id`, `item.givenName`, etc.)
  - Resources/Topics (`home`, `collection`,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

API Design Basics: Vocabulary

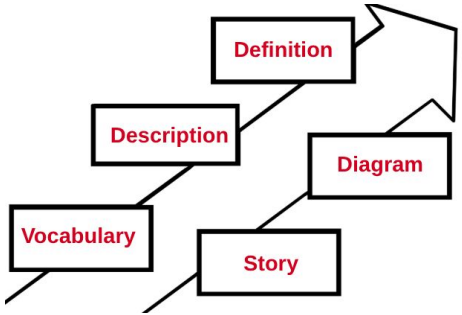
Semantic Descriptors

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

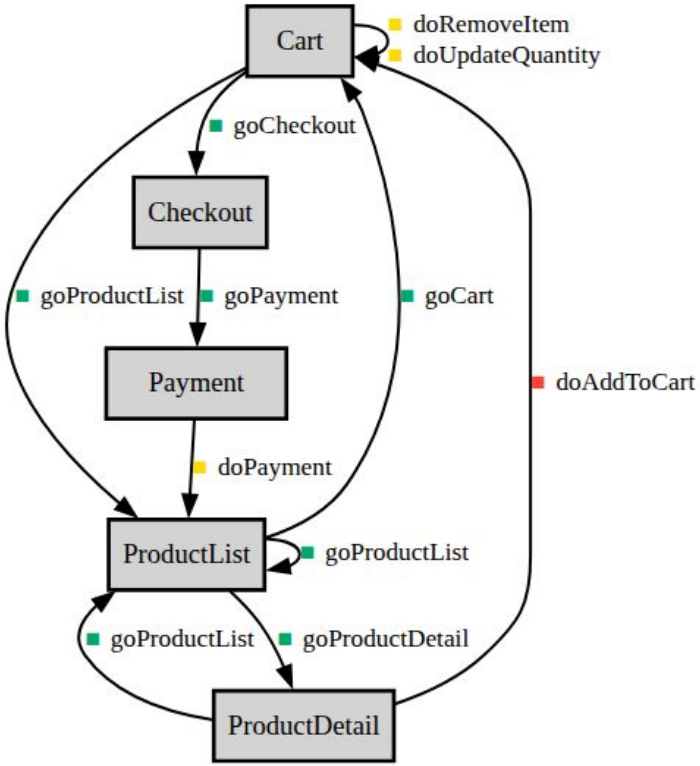


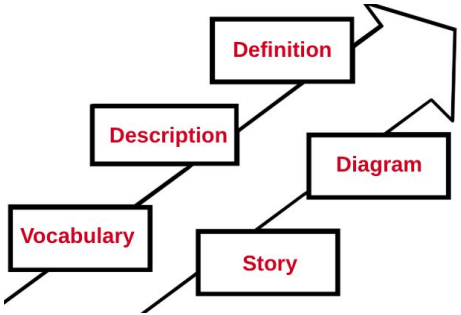
API Design Basics: Diagram



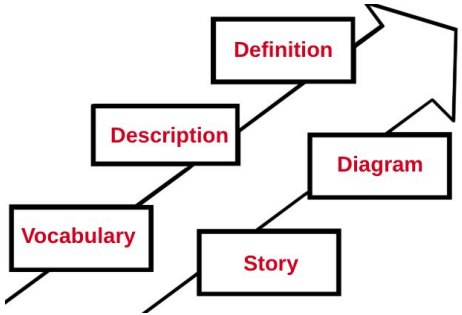
- Visual representation of the API
- Not ..
  - a state diagram
  - a flow diagram
- An *interaction* diagram

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

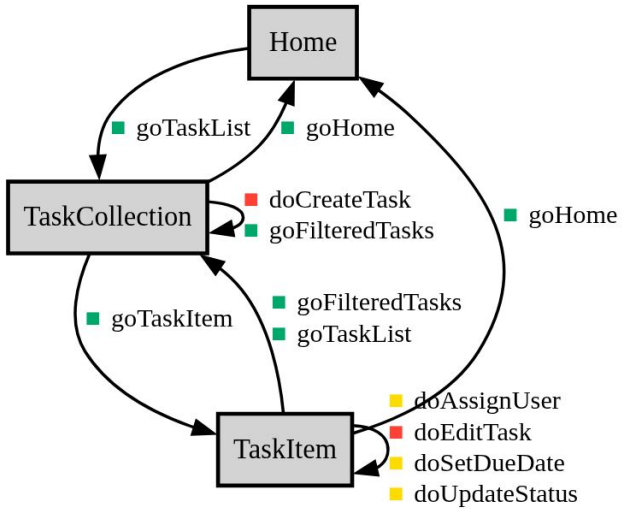
app-state-diagram

JSON ✅ Downloa


```
{
  "$schema": "https://alps-io.github.io/schemas/alps.json",
  "alps": {
    "version": "1.0",
    "title": "Task Management API",
    "doc": {
      "format": "text",
      "value": "ALPS profile for managing task records including required fields
    },
    "descriptor": [
      {
        "id": "id",
        "type": "semantic",
        "title": "Id",
        "doc": {
          "format": "text",
          "value": "UUID of the task"
        },
        "tag": "task-management"
      },
      {
        "id": "title",
        "type": "semantic",
        "title": "Title",
        "doc": {
          "format": "text",
          "value": "Title of the task"
        },
        "tag": "task-management"
      },
      {
        "id": "description",
        "type": "semantic",
        "title": "Description",
        "doc": {
          "format": "text",
          "value": "Detailed task description"
        },
        "tag": "task-management"
      },
      {
```


Task Management API

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

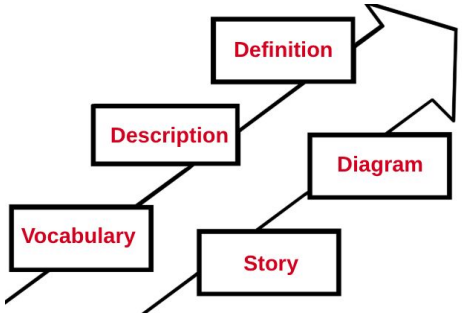


Definition

Diagram

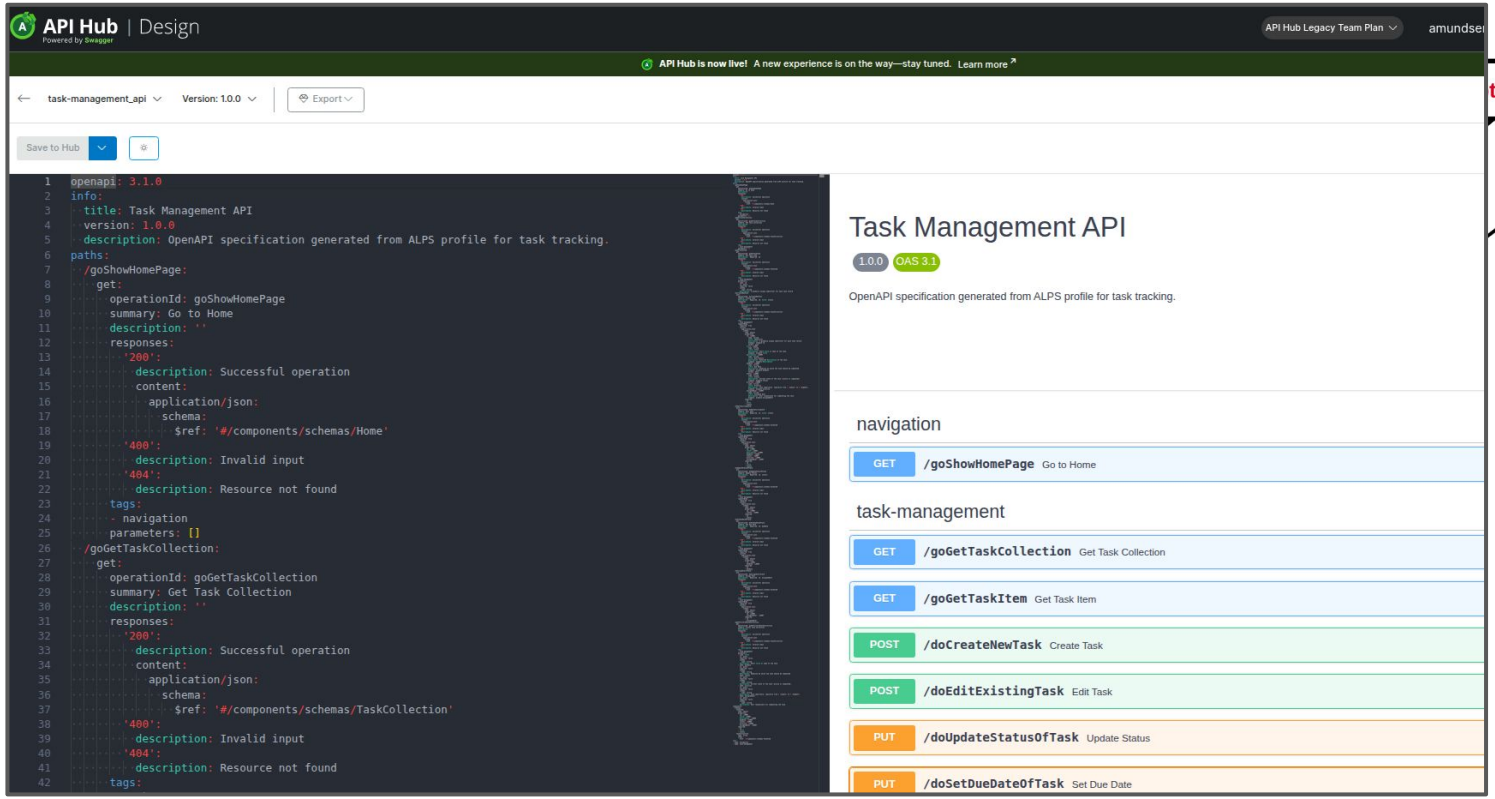
ory

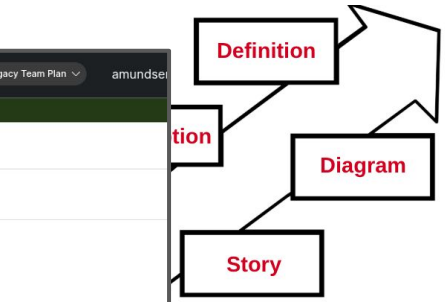
API Design Basics: Definition



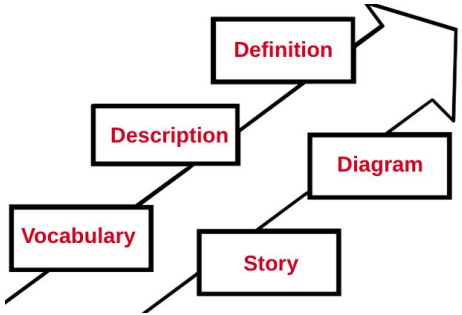
- API definitions translate design into implementation
- API definitions are implementation-specific
  - OpenAPI (HTTP)
  - AnsycAPI (Event-Driven)
  - ProtoBuf (gRPC)
  - SDL (GraphQL)
- Usually API definitions power "generators"
  - Code
  - Diagrams
  - Documentation

API Design Basics: Definition

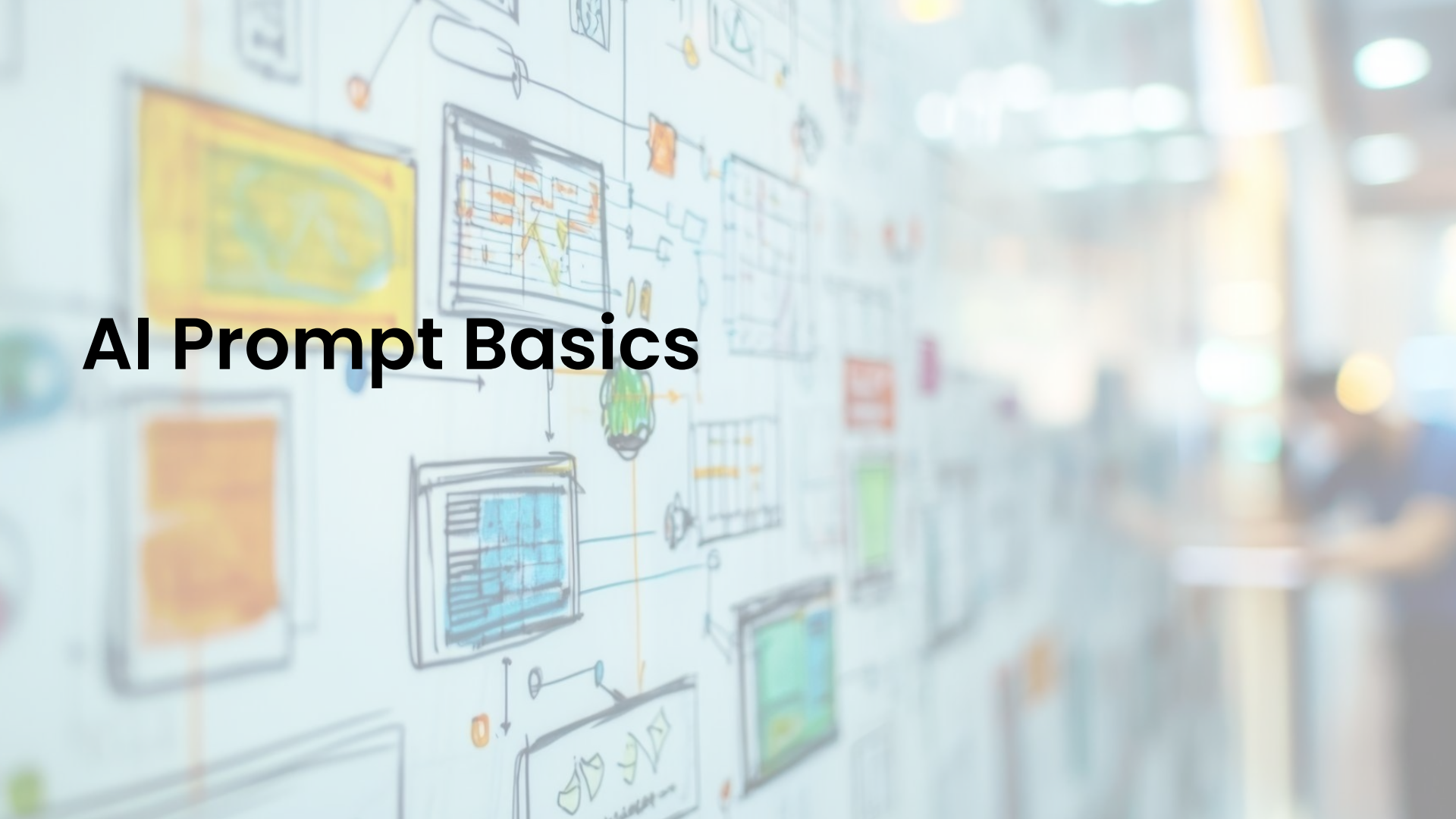




API Design Basics: Summary



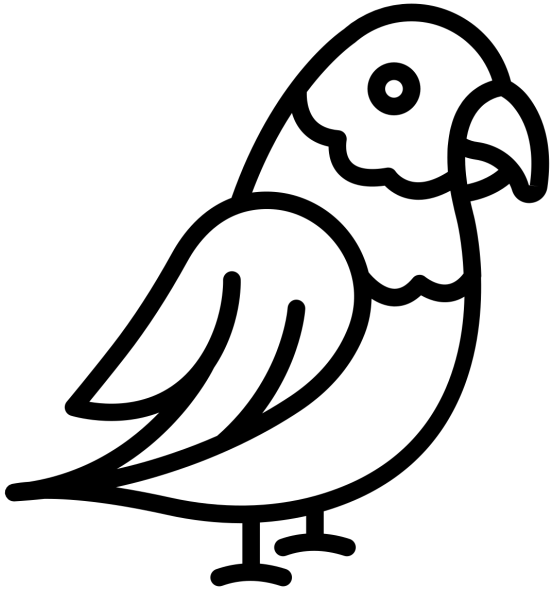
- Story
  - Stories are shared understanding
- Vocabulary
  - All the "magic words" related to the API domain
- Diagram
  - Visual representation of the API
- Description
  - The complete design in one place, in one voice.
- Definition
  - Translate design into implementation



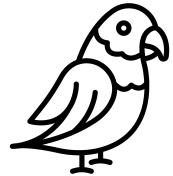
AI Prompt Basics

AI Prompt Basics

- Conversation
- Context
- Memory
- Tips



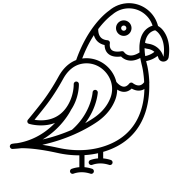
AI Prompt Basics : Conversation



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

"On the Dangers of Stochastic Parrots" : Bender, Gebru, McMillan-Major, and Mitchell

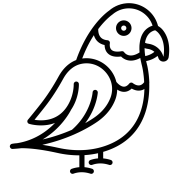
AI Prompt Basics : Context



- 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

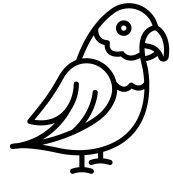
"On the Dangers of Stochastic Parrots" : Bender, Gebru, McMillan-Major, and Mitchell

AI Prompt Basics : Memory



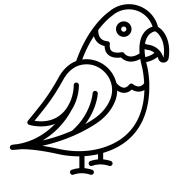
- Each conversation is a separate session
- That session is the only "memory" it has
- They won't remember anything for next time

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***

https://www.youtube.com/watch?app=desktop&v=UUlhqPBk0GA

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

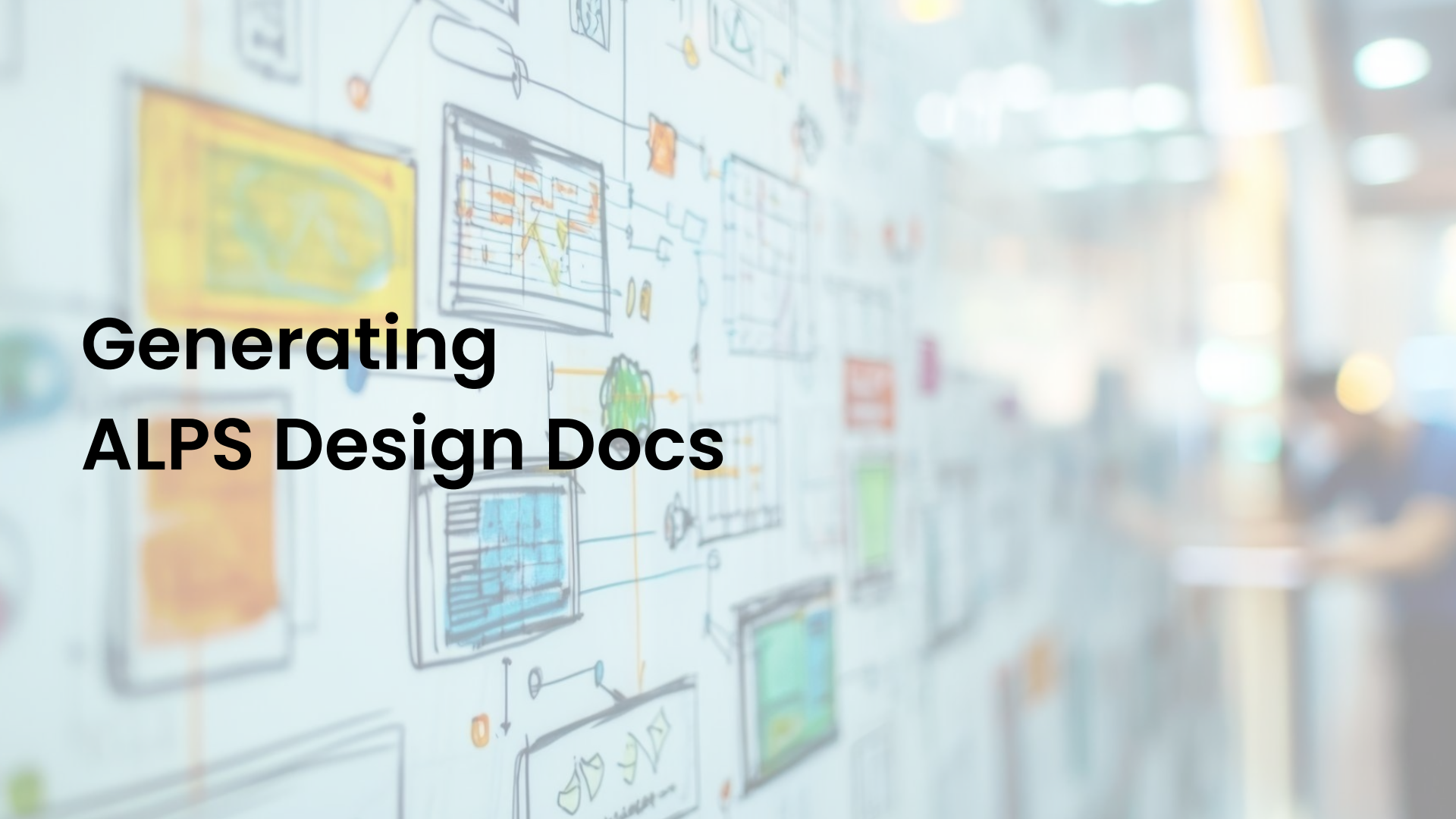
Authoring API Stories : Summary

- 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

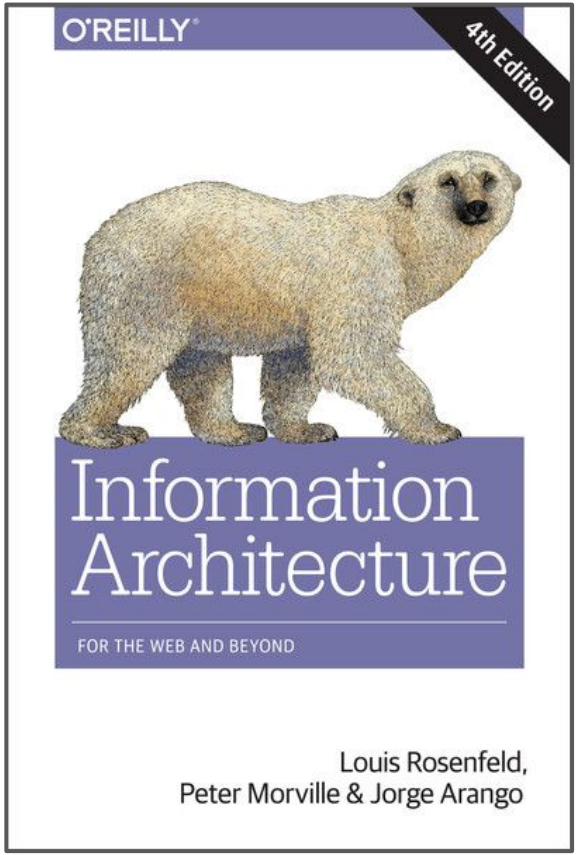
Exercise: Generating Designs with API Stories

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

Generating ALPS Design Docs

Information Architecture

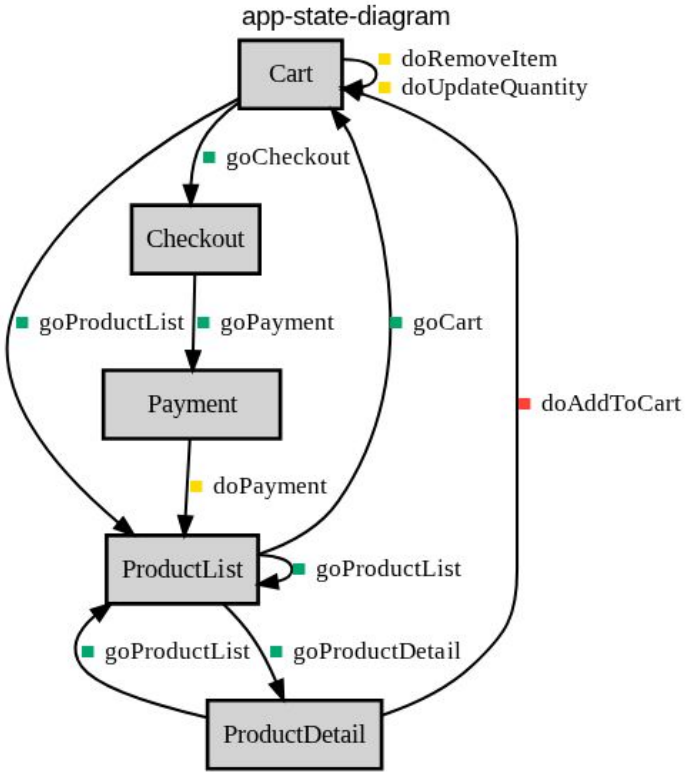
- Content, structure, and intent
- Richard Saul Wurman, Peter Morville, Lou Rosenfeld, and Don Norman
- Used in library science and web UI design
- Ontology, Taxonomy, Choreography



Generating ALPS Design Docs

Application-Level Profile Semantics

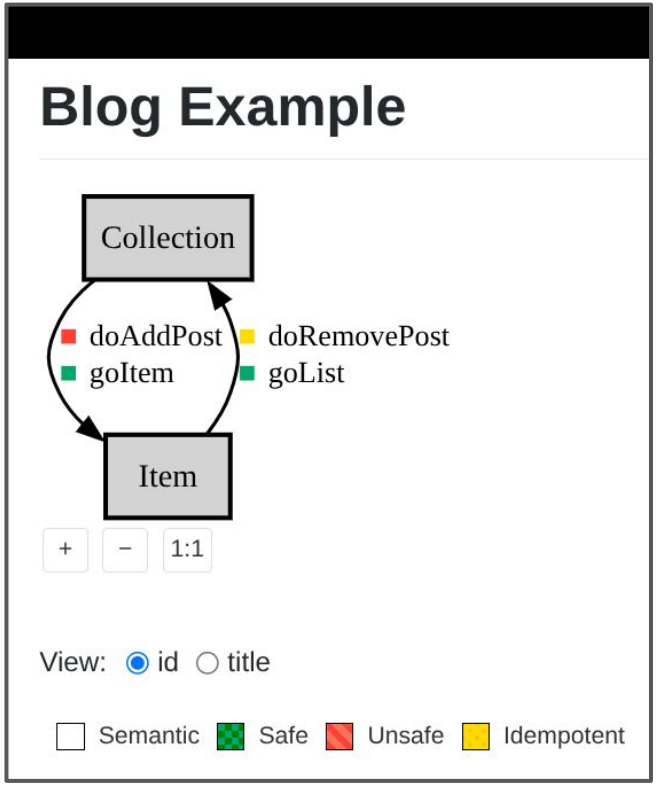
- ALPS is a format that expresses application-level meaning and structure
- Focus on Actions and Resources
- Supporting RESTful Design



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"}]},
```

Generating ALPS Design Docs

ALPS Basics

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

```
{"id": "Collection", "title": "Article List", "descriptor": [
    {"href": "#BlogPost"}, {"href": "#goItem"}, {"href": "#doAddPost"}]},
{"id": "Item", "title": "Article", "descriptor": [
    {"href": "#BlogPost"}, {"href": "#doRemovePost"}, {"href": "#goList"}]},
```

Generating ALPS Design Docs

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

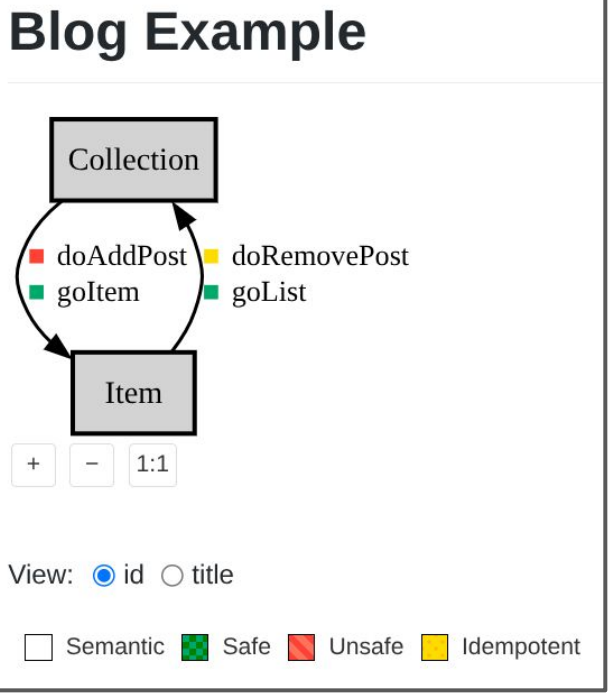
```
{"type": "safe", "id": "goList", "rt": "#Collection", "title": "View Article List"},
{"type": "safe", "id": "goItem", "rt": "#Item", "title": "View Article",
    "descriptor": [{"href": "#id"}]},
{"type": "unsafe", "id": "doAddPost", "rt": "#Item", "title": "Add Blog Article",
    "descriptor": [{"href":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

Generating ALPS Design Docs

ALPS Basics

app-state-diagram

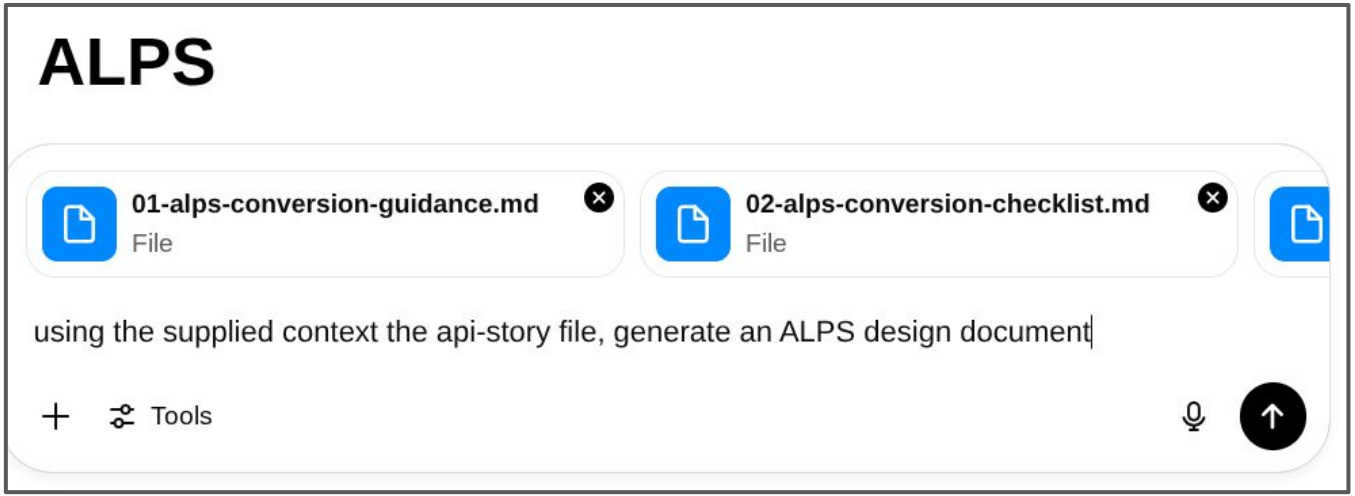
```
{
    "$schema": "https://alps-io.github.io/schemas/alps.json",
    "alps": {
        "title": "Blog Example",
        "descriptor": [
            {"id": "id", "title": "id"},
            {"id": "articleBody", "title": "Content"},
            {"id": "dateCreated", "title": "Creation Date"},
    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

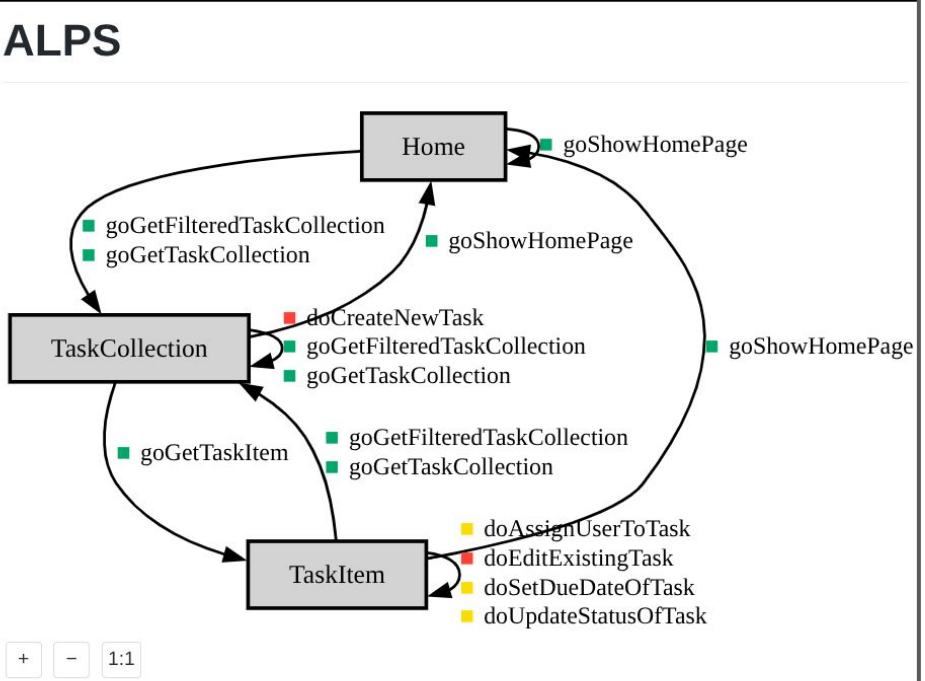
Exercise : Generating Designs with API Stories

- Supply Context
- Generate ALPS
- Validate Output



app-state-diagram JSON ✅ Download


```
{
  "$schema": "https://alps-io.github.io/schemas/alps.json",
  "alps": {
    "version": "1.0",
    "descriptor": [
      {
        "id": "id",
        "type": "semantic",
        "title": "Identifier",
        "doc": {
          "value": "A globally unique identifier for each task record."
        },
        "tag": "task-management"
      },
      {
        "id": "title",
        "type": "semantic",
        "title": "Title",
        "doc": {
          "value": "Short title or name of the task."
        },
        "tag": "task-management"
      },
      {
        "id": "description",
        "type": "semantic",
        "title": "Description",
        "doc": {
          "value": "Detailed description of the task."
        },
        "tag": "task-management"
```

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

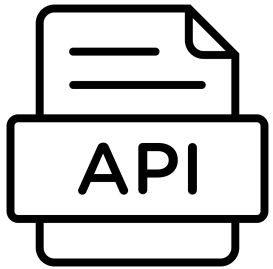
AI-Driven API Design Stack

Generating API Descriptions

Authoring API Stories

Generating API Documentation

Generating API Documentation



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

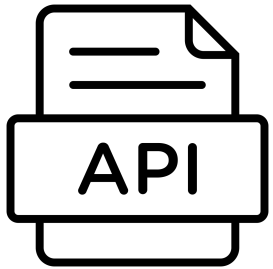
Exercise: Generate starter documentation from ALPS

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

Generating API Documentation

What are Docs?

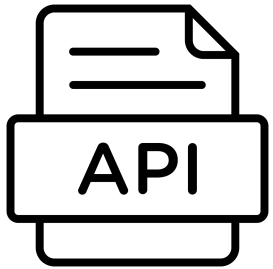
- Design Document
  - Intentions, scope, capabilities
- Implementation Plan
  - Inputs, outputs, states, protocols
- Operational Guide
  - Deployment, dependencies, monitoring, etc



Generating API Documentation

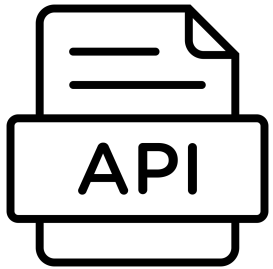
Who uses them?

- 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

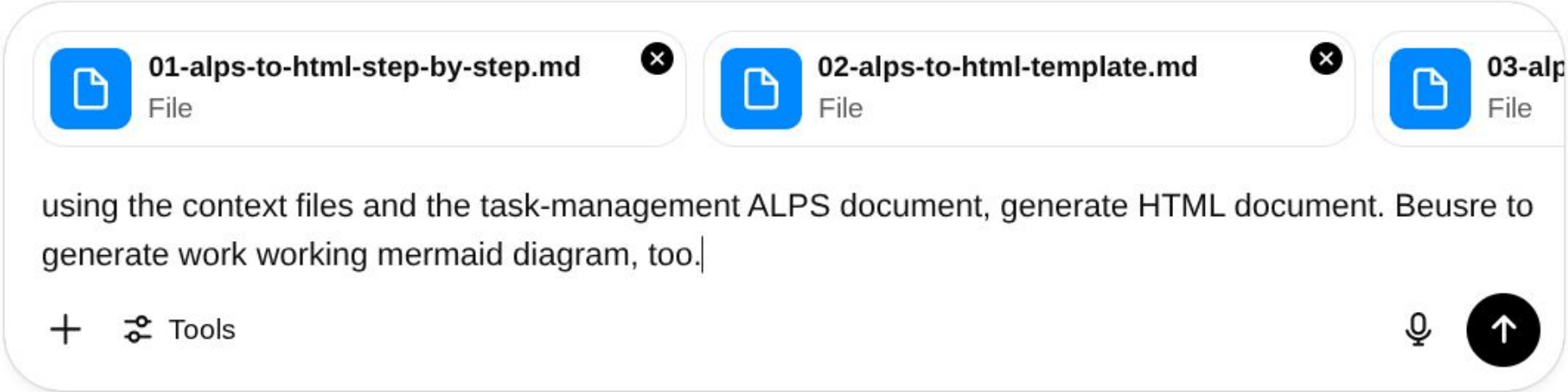
Basic elements of API documentation



- 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

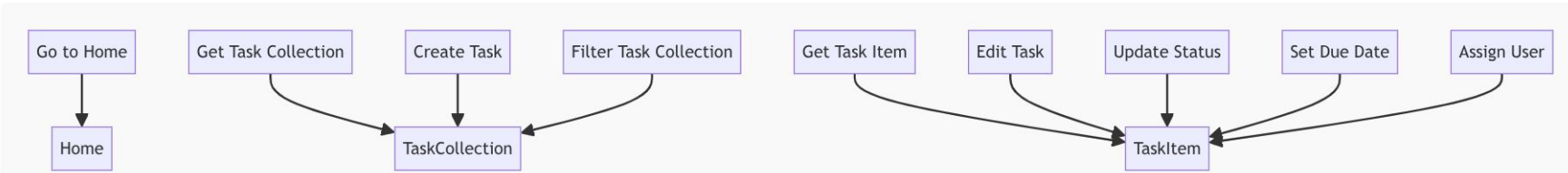


Task Management API

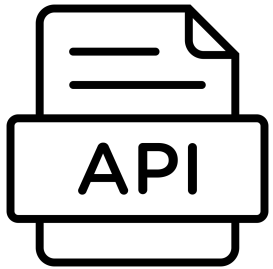
Overview Data Fields States Transitions Diagram

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

API State Diagram

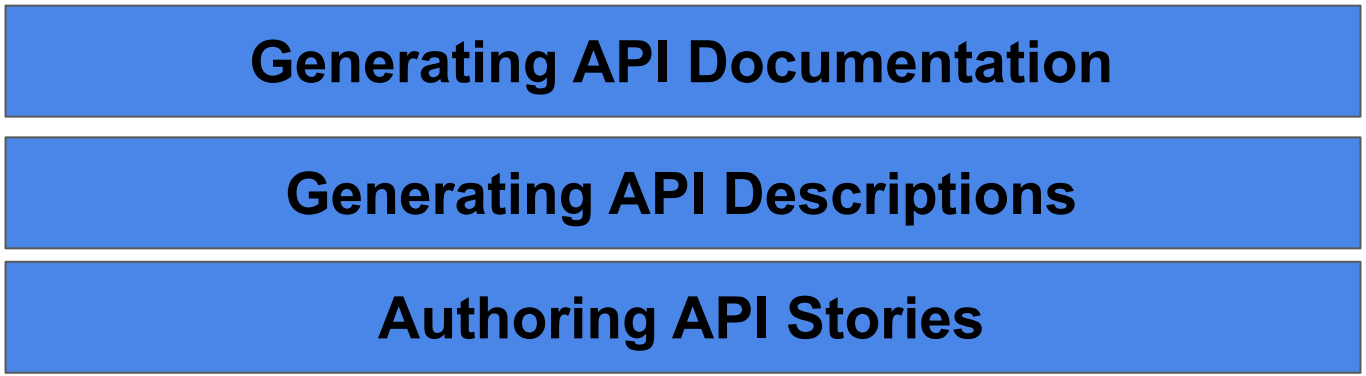


Generating API Documentation : Summary

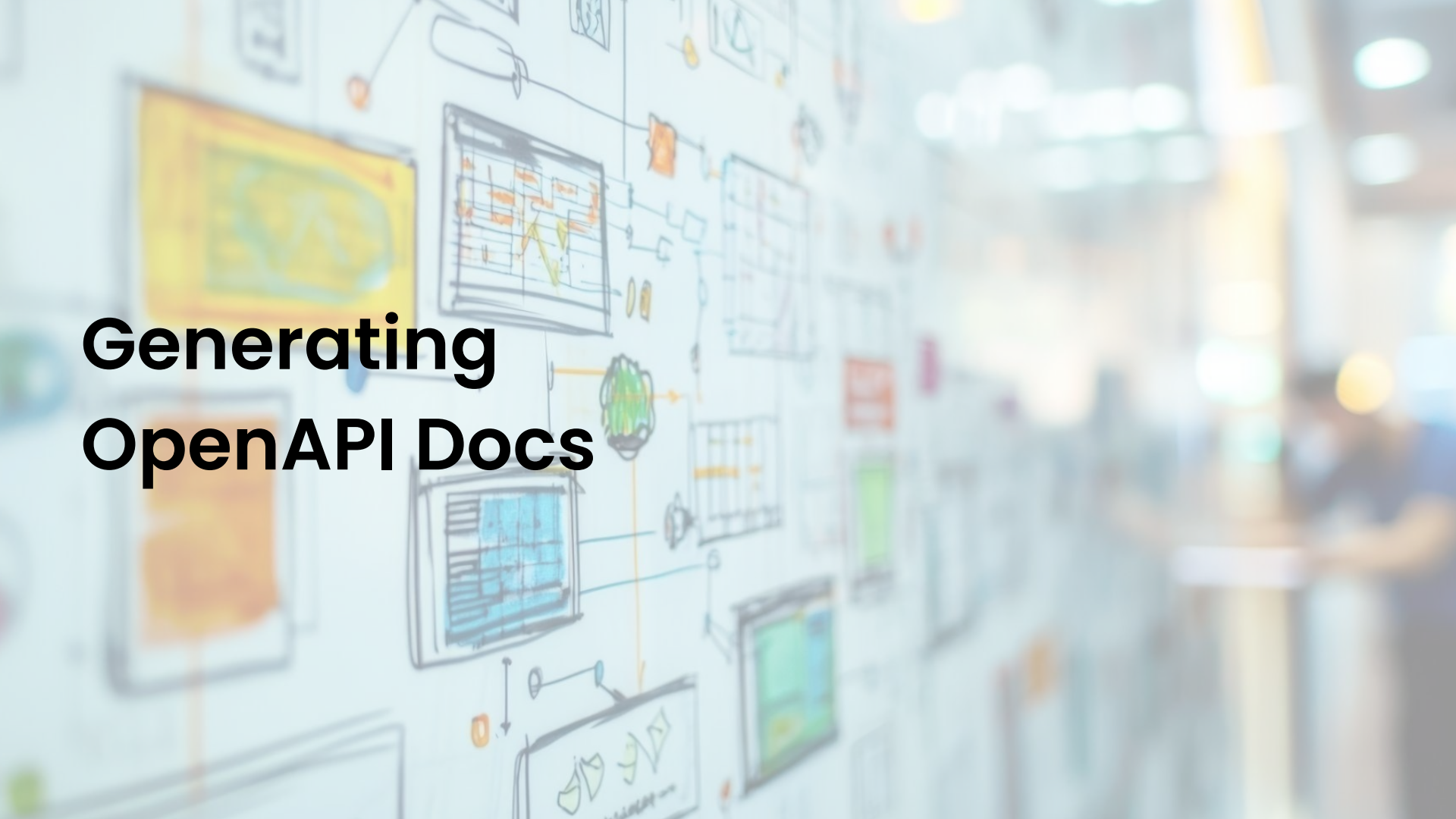


- What are Docs?
  - Design, implementation, operation
- Who uses them?
  - Developers, customers, operators
- Basic Elements of API Docs
  - Purpose, properties, resources, actions

AI-Driven API Design Stack



Break


Generating
OpenAPI Docs

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

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

```yaml
openapi: 3.0.0
info:
  version: '1.0'
  title: Person API
  description: An example API for the "Designing APIs with OpenAPI" workshop
servers:
  # Added by API Auto Mocking Plugin
  - description: SwaggerHub API Auto Mocking
    url: https://virtserver.swaggerhub.com/amundsen/Person-API/1.0

paths: {}
components: {}
```

Generating Open API Documents

Open API Basics

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

```
parameters:
  nameParam:
    name: name
    in: query
    descriptio
    required:
    schema:
      type: st
      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

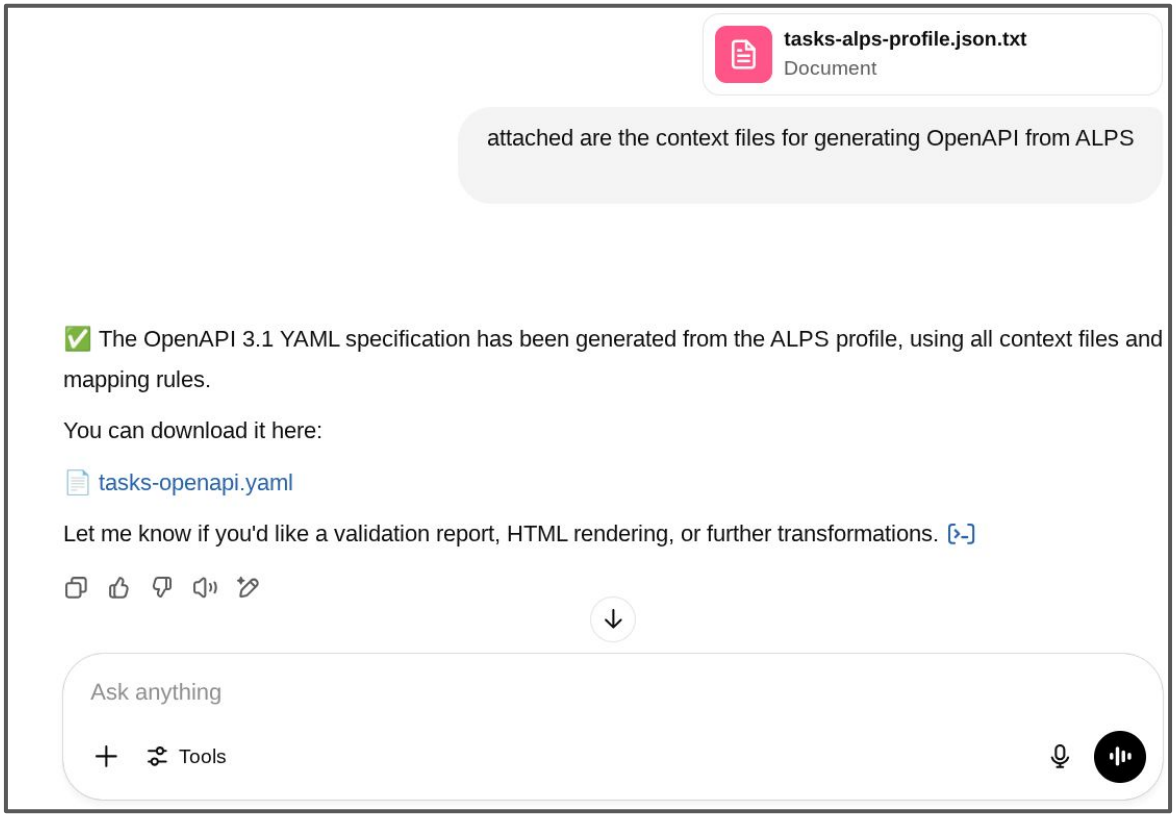
Open API Basics

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

```yaml
### 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"
```

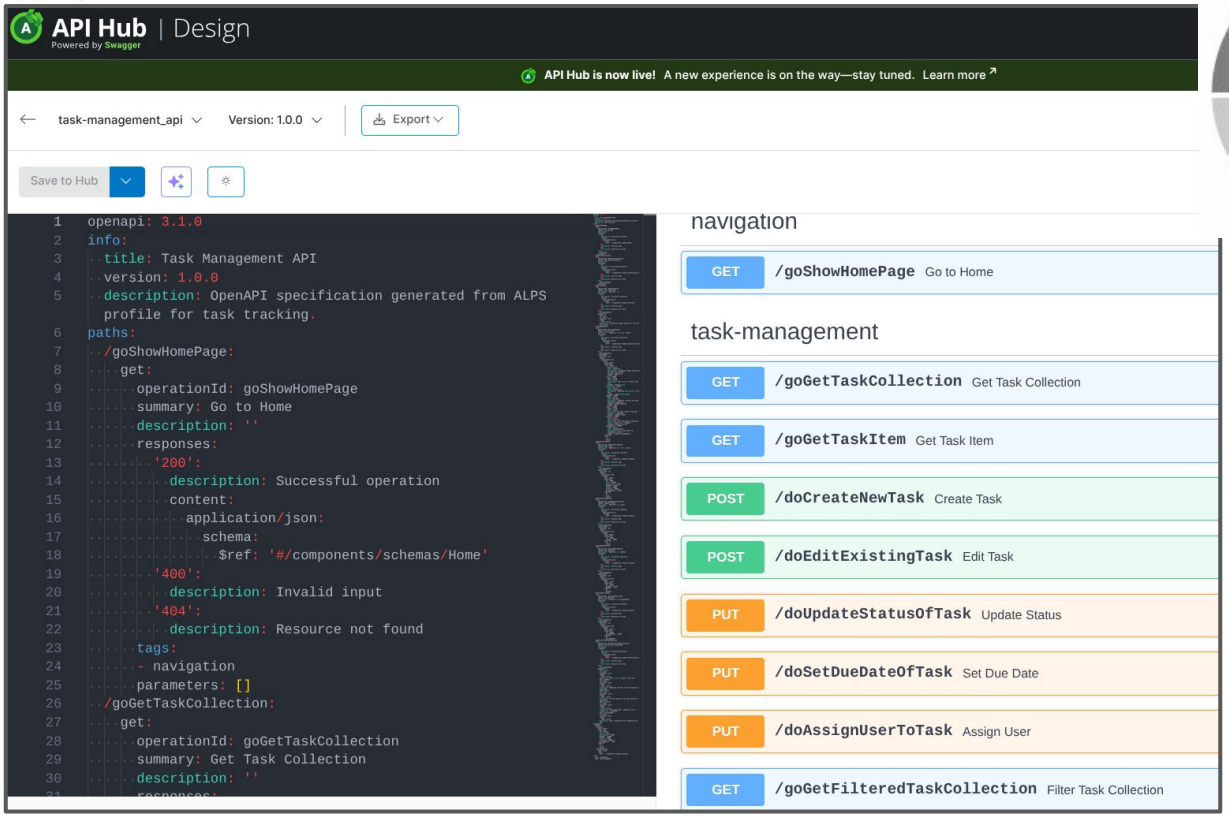
Generating Open API Documents

From ALPS to OpenAPI



Generating Open API Documents

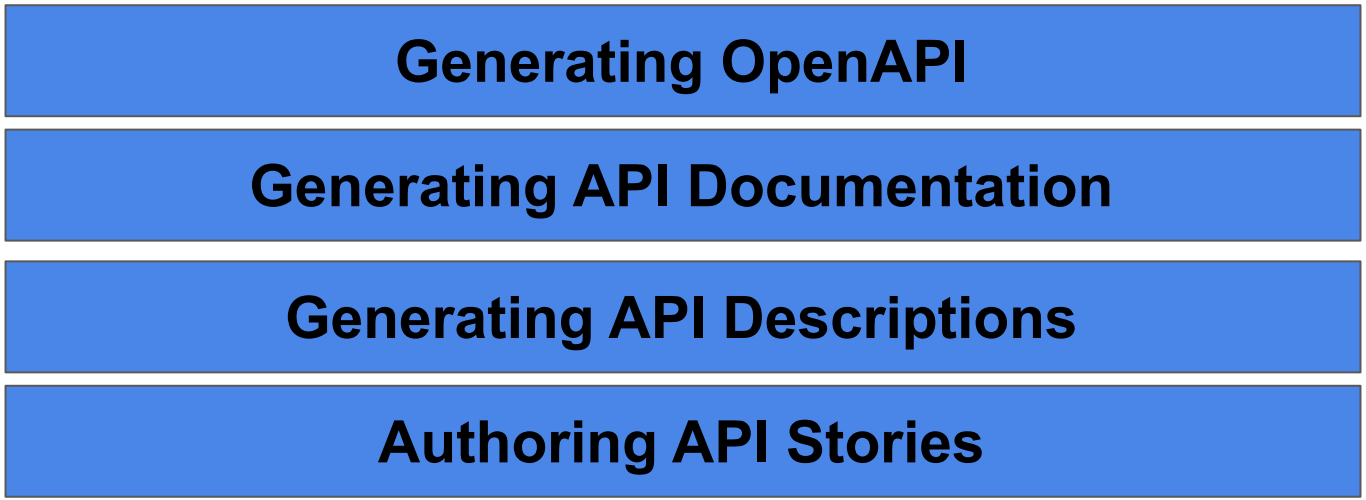
From ALPS to OpenAPI



Generating Open API Documents : Summary

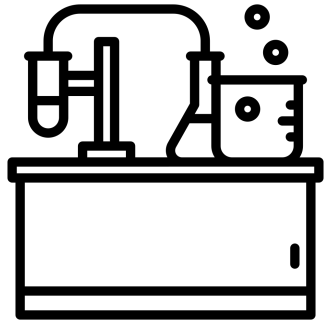
- Why OpenAPI
  - Common platform for describing HTTP APIs
- Who Uses it?
  - Designers, builders, testers, consumers
- OpenAPI Basics
  - Info, components, paths

AI-Driven API Design Stack




Generating
Running
Prototypes

Generating Running Prototypes

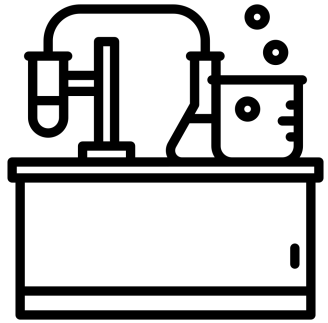


- What is a Prototype?
- Why use them?
- Prototype Essentials

Exercise: Generate a NodeJS prototype from ALPS

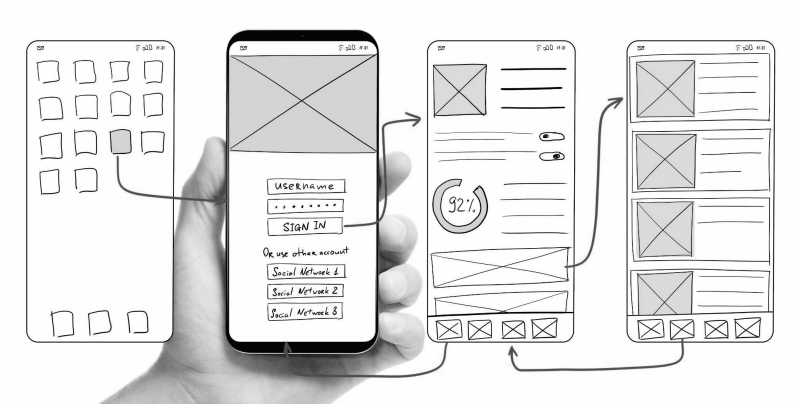
Generating Running Prototypes

What is a prototype?

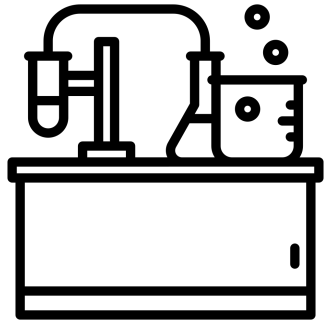


A first, typical or preliminary model of something, especially a machine, from which other forms are developed or copied.

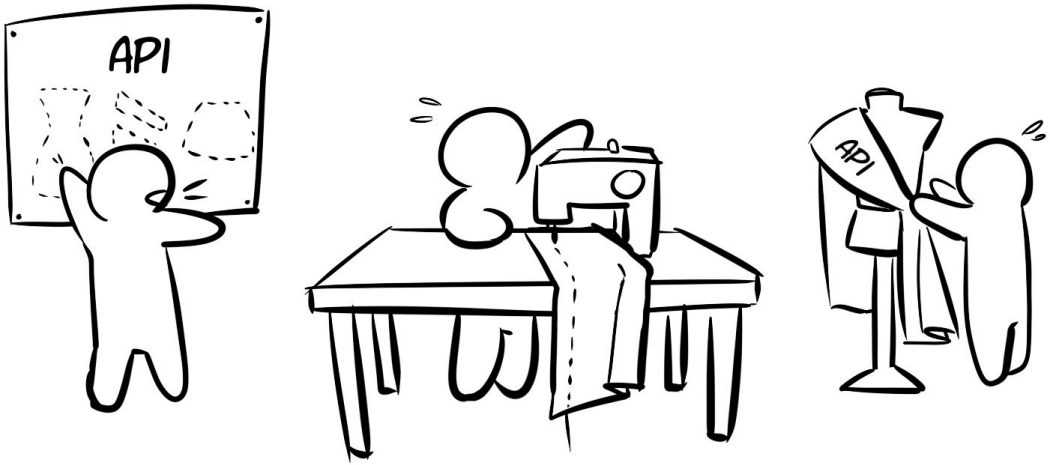
-- Oxford Dictionary



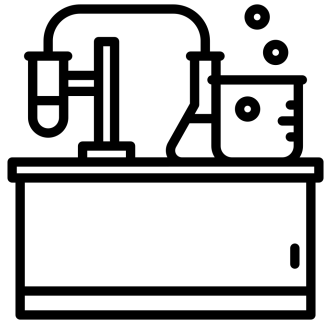
Generating Running Prototypes



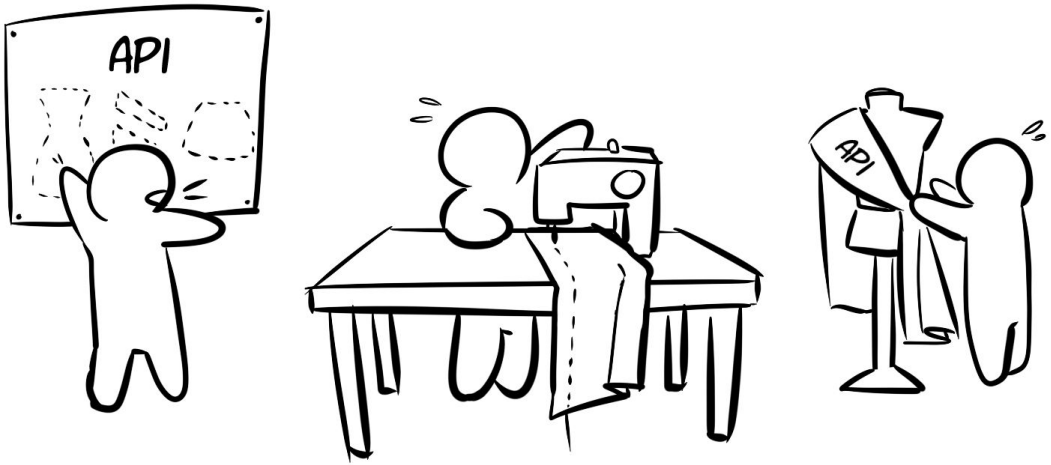
- Inexpensive experiments
- Explore the details
- Prototypes are made to be tested



Generating Running Prototypes

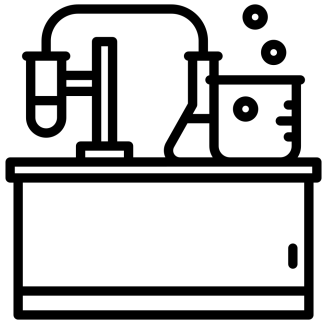


- Inexpensive experiments
- Explore the details
- ***Prototypes are made to be tested***



Generating Running Prototypes

Prototype Essentials

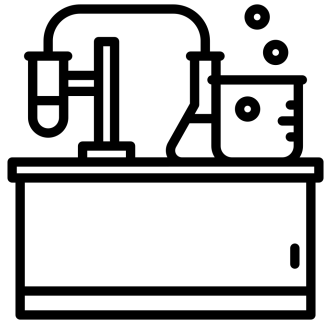


- Running Code
- Rough Consensus
- Throw it away

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

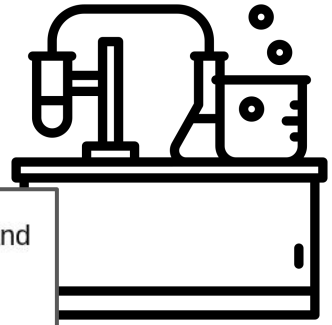
02-api-checklist-v2.md
File

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.

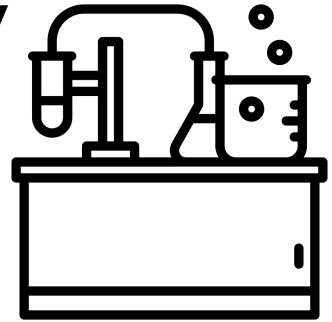
Generating Running Prototypes

```
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.toLowerCas
59      )
60    );
61    res.json(filtered.map(task => ({ ...task, _links: genera
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 no
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;
```

127.0.0.1:3000/tasks/


```
[
  - {
      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: [
```

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

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

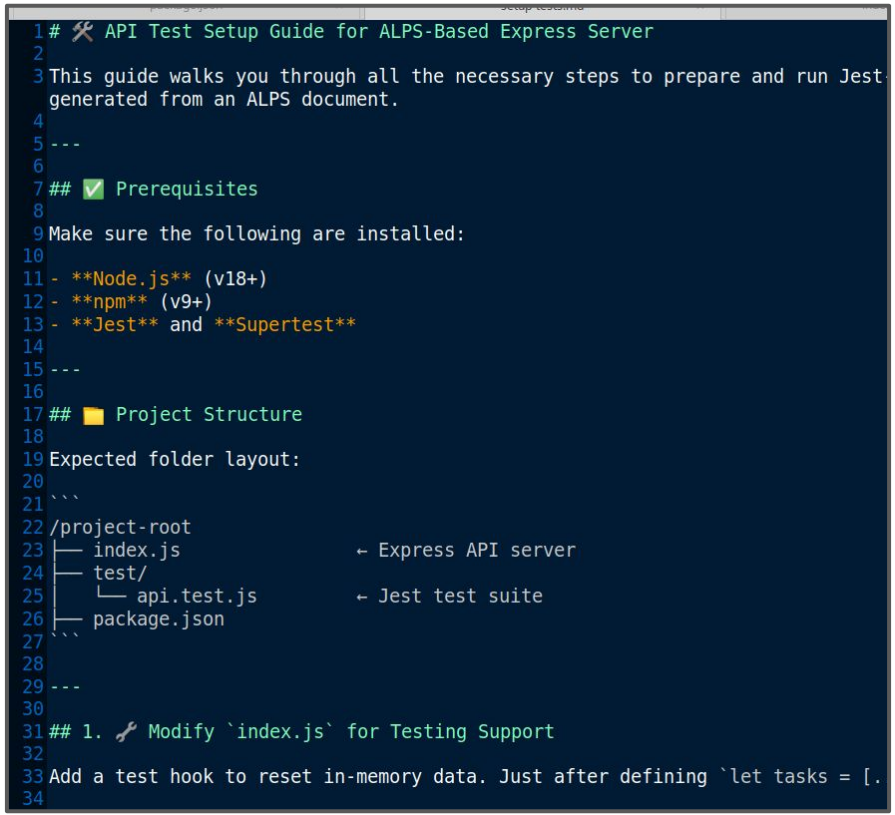
Generating API Tests

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



```
# 🛠️ API Test Setup Guide for ALPS-Based Express Server

This guide walks you through all the necessary steps to prepare and run Jest
generated from an ALPS document.

---

## ✅ Prerequisites

Make sure the following are installed:

- **Node.js** (v18+)
- **npm** (v9+)
- **Jest** and **Supertest**

---

## 📁 Project Structure

Expected folder layout:

```
/project-root
├── index.js                ← Express API server
├── test/
│   └── api.test.js         ← Jest test suite
├── package.json
```

---

1. 🔧 Modify `index.js` for Testing Support

Add a test hook to reset in-memory data. Just after defining `let tasks = [.
```

Generating API Tests

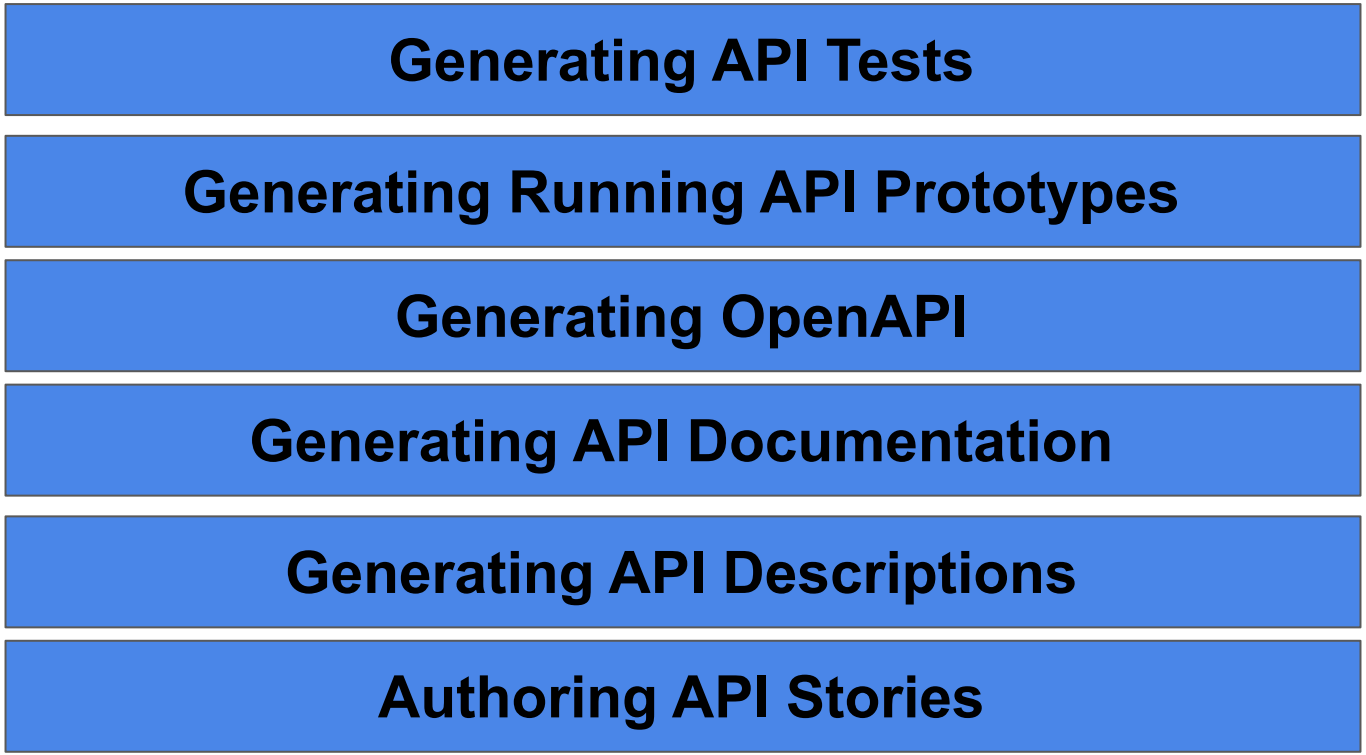
Run test suite

```
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
Ran 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 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

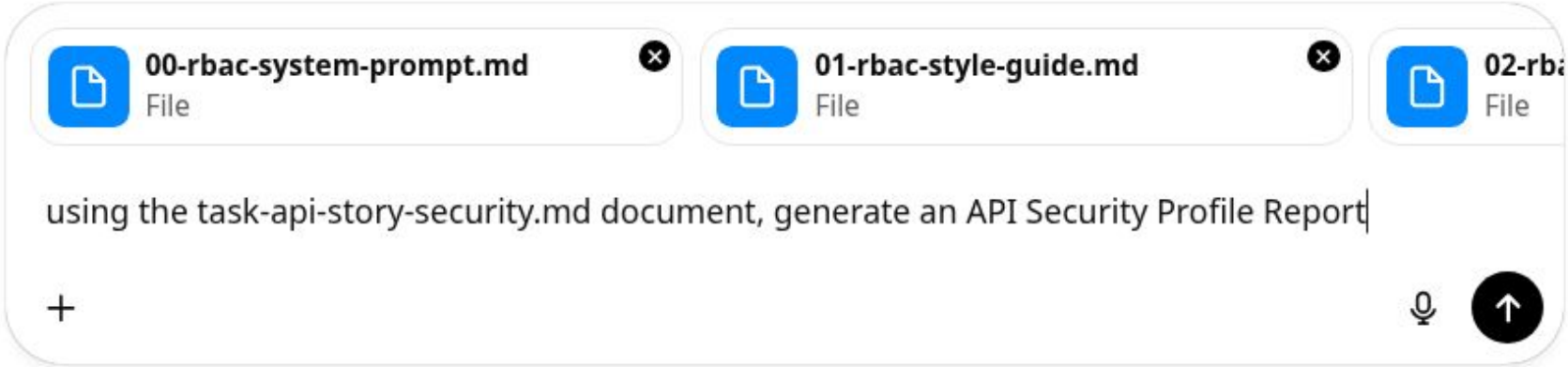
Generating Security Profiles

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



Generating Security Profiles

Run security context process

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

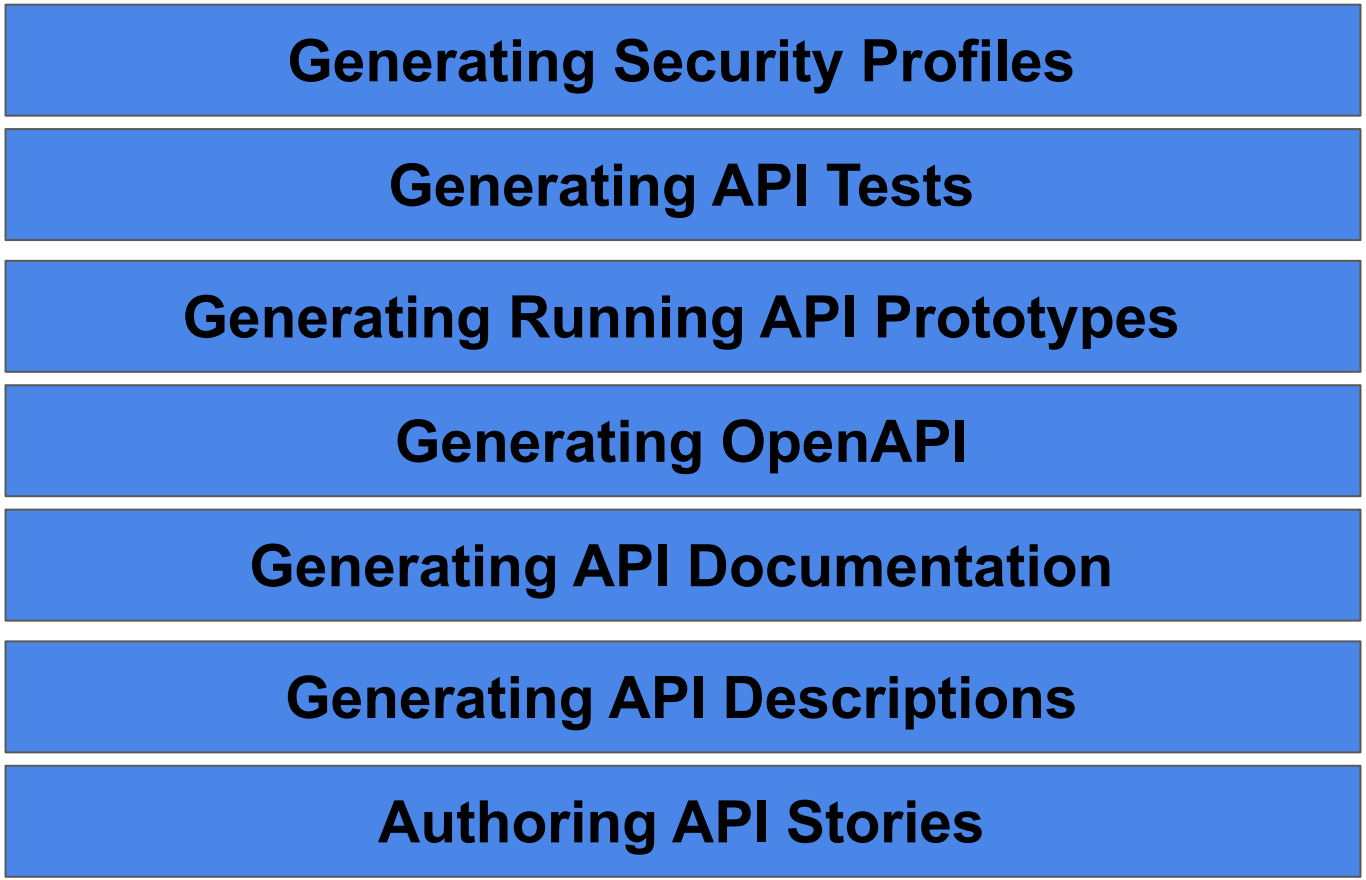
Action	Type	anon	user	admin
ShowHomePage	Safe	✔	✔	✔
GetTaskCollection	Safe		✔	✔

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

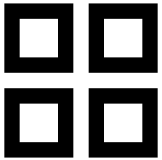
AI-Driven API Design Stack



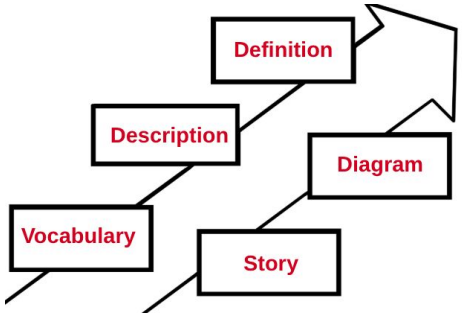
Course Review

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

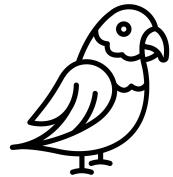


API Design Basics: Review



- 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

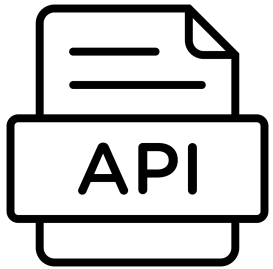
Authoring API Stories : Review

- 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

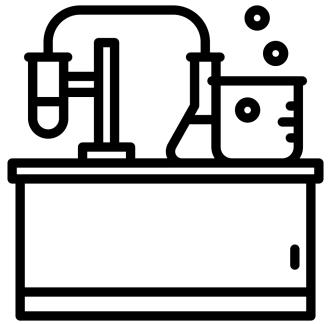


- What are Docs?
  - Design, implementation, operation
- Whole uses them?
  - Developers, customers, operators
- Basic Elements of API Docs
  - Purpose, properties, resources, actions

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

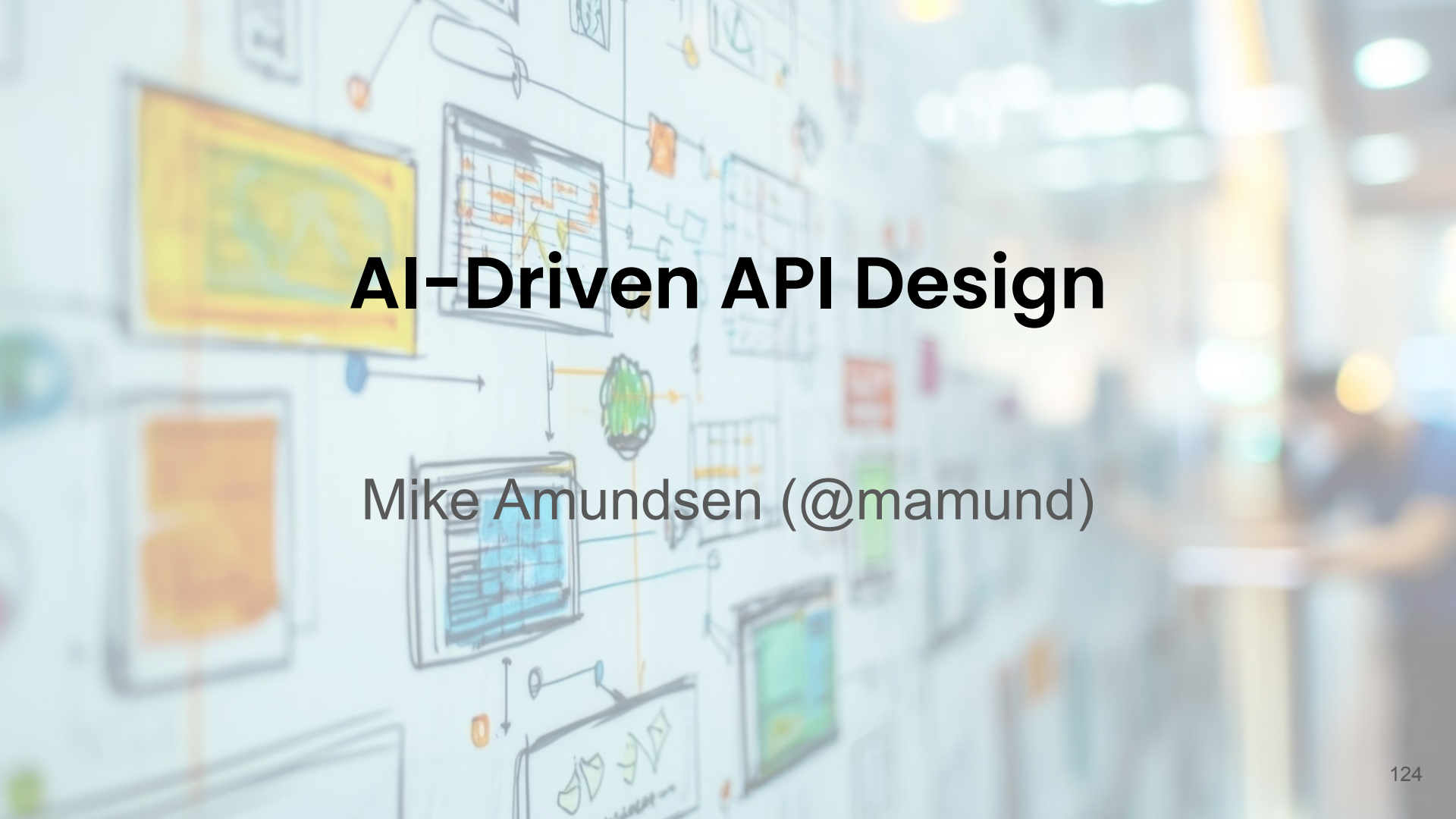
1.3K 5K 29K 15K

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



AI-Driven API Design

Mike Amundsen (@mamund)