ATTENZIONE — CONTENUTI EXTRA

- Il contenuto di queste slide **NON è parte del programma di Tecnologie Web per l'anno accademico 2023/2024.**
- Questi argomenti **non** appariranno nelle prove scritte, **né** saranno chiesti durante la discussione del progetto.
- Per quanto riguarda il progetto, si sottolinea che:
  - La traccia del progetto **vieta esplicitamente l'utilizzo di CMS.**
  - È ammesso, invece, l'utilizzo di **GraphQL**.

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

WEB TECHNOLOGIES — BONUS LECTURE

(HEADLESS) CONTENT MANAGEMENT SYSTEMS AND GRAPHQL

Luigi Libero Lucio Starace, PhD

luigiliberolucio.starace@unina.it

https://luistar.github.io

https://www.docenti.unina.it/luigiliberolucio.starace

MANAGING CONTENT: CHALLENGES

The key goal of many web applications is to display content to users

- Content may **change** very frequently
- Content may **grow exponentially** (e.g.: ~7M pages in Wikipedia)

This poses some challenges:

- Hand-coding entire web pages for content updates requires technical expertise and results in slower workflows
  - Those in charge of contents are typically not the web devs
- Managing different versions of the published content is not trivial
- Ensuring a structured workflow is adopted is challenging

CONTENT MANAGEMENT SYSTEMS

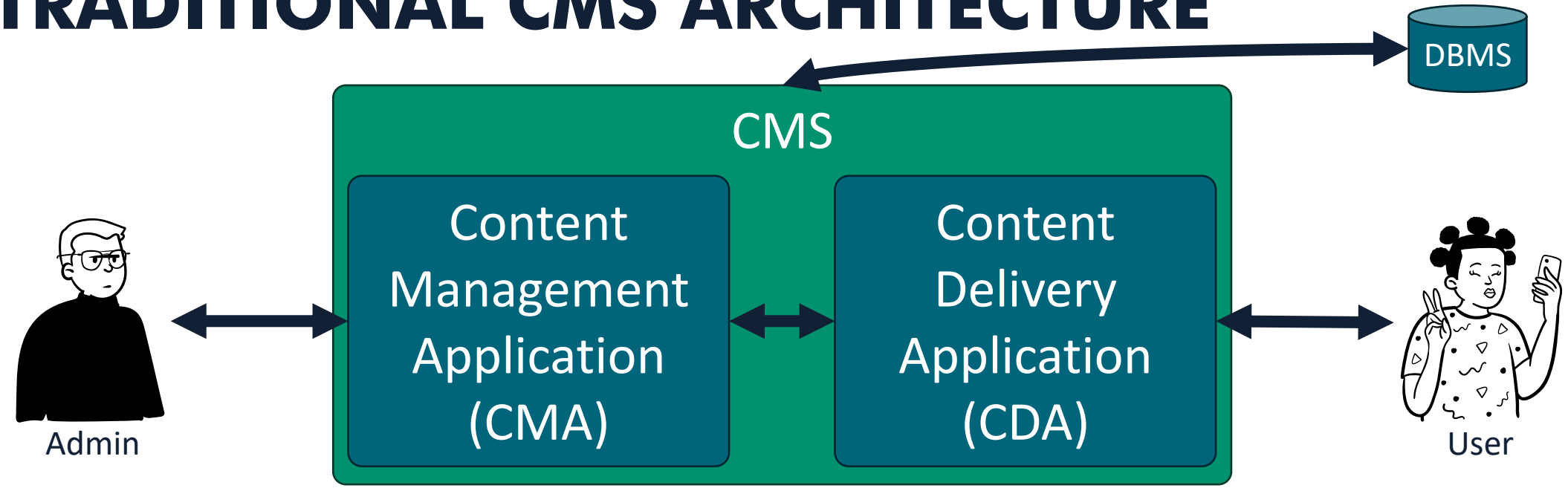
- A Content Management System (CMS) is a web application designed to allow collaborative **editing**, **organizing** and **publishing** of content
- Without the need to write a single line of code, using the User Interface (UI) provided by the CMS itself
  - Empowers non-technical users to manage and update content without constant reliance on web devs.
- CMSs are often used to run blogs, news, and e-commerce websites

CMS: FEATURES

Most CMS include the following features

- **Web publishing** (e.g.: content is available to users via web pages)
- **Formatting content** (typically with a WYSIWYG graphical editor)
- **Version control** (keep track of multiple versions of the same content)
- **Indexing**, **search** and **retrieval** of content
- User **authentication/authorization** with a group-based permission system
- Built-in **media manager** for multimedia content
- **Extensibility** and **customizability** (via themes and plugins)

TRADITIONAL CMS ARCHITECTURE



Traditional CMS are composed by two key parts:

- **Content Management Application**: allows non-technical users to manage content (e.g.: through its web pages)
- **Content Delivery Application**: Makes the updated content available to end-users (e.g.: as web pages)

TRADITIONAL CMSs

Traditional CMS have a **monolithic** architecture

- CMA and CDA are closely integrated in a monolith
- This makes traditional CMSs easy to distribute and operate (they are a single web application with two components)

These advantages come with some limitations:

- Might **not** be **flexible enough** for some needs
- Might **limit distribution channels** (traditional CMSs generally prioritize web outputs. What if I need to deploy a REST API?)

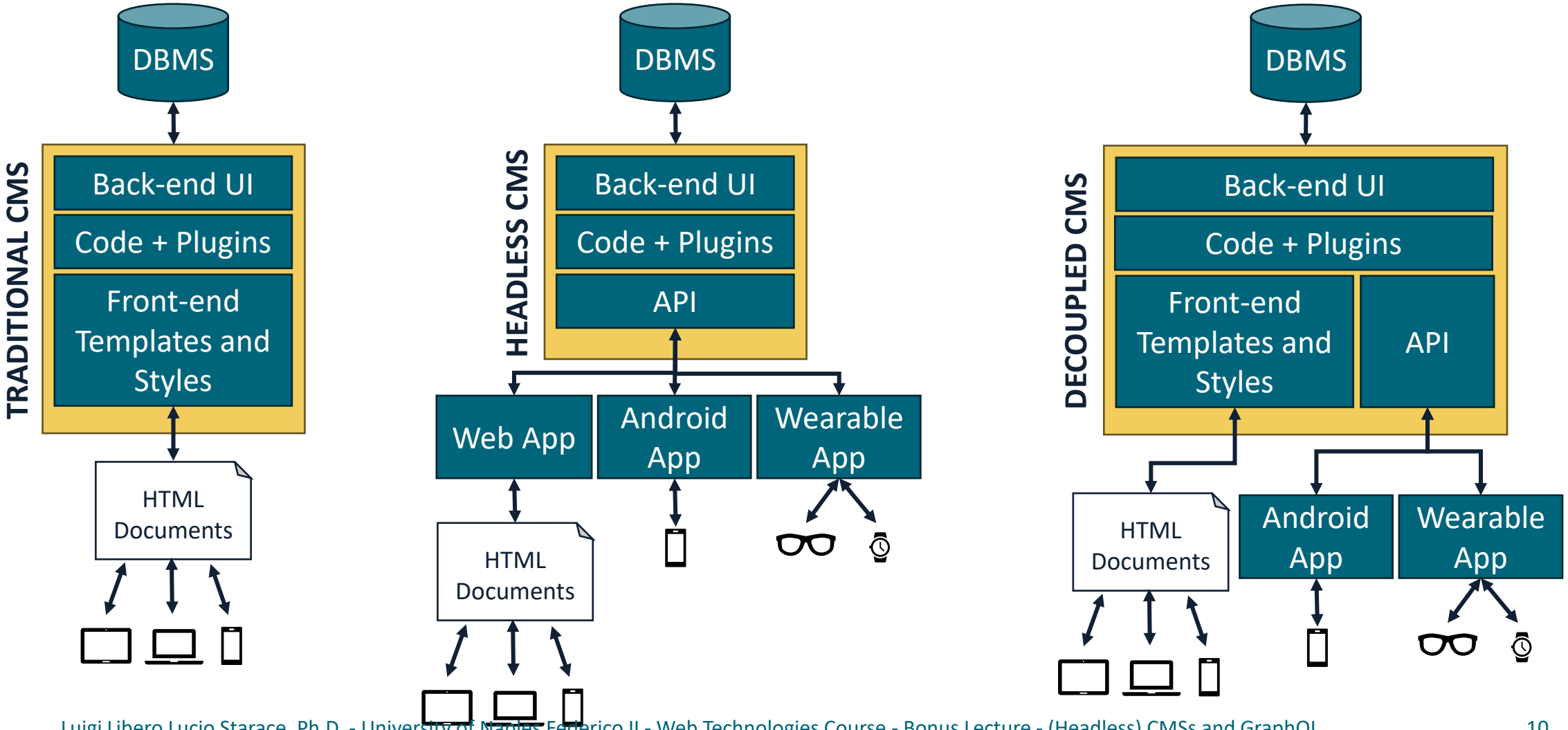
HEADLESS CMSs

- **Headless** CMSs are a recent trend
- Based on the idea to separate
  - the CMA (the *body*, which takes care of content organization) and
  - the CDA (the *head*, which takes care of the presentation)
- An Headless CMS is a CMS without its *head*. It does not manage the presentation of content, and only exposes an API
- This way, content can be deployed on any channel we need

DECOUPLED CMSs

- Decoupled CMSs are traditional CMSs in which the CMA and the CDA are decoupled.
- Users can use the included CDA, or implement different ways to distribute the content using the APIs exposed by the CMA
- Most Traditional CMS are nowadays decoupled (e.g.: WordPress)

CMSs: OVERVIEW



CMSs: SECURITY RISKS

- CMS are great targets for attacks
  - The same CMS is used on a large number of websites
  - A vulnerability in a CMS can be exploited on a large number of websites
- It is **imperative** to keep CMSs updated to patch vulnerabilities as soon as possible
- E.g.: CVE-2022-21661
  - WordPress between versions 3.7.37 and 5.8.3 was vulnerable to **SQL Injections**!

USING A (TRADITIONAL) CMS

TRADITIONAL CMS MARKET SHARE

- A number of CMS exists (full list here). Well-known ones include:
  - WordPress (the most popular CMS)
  - Joomla (powers the website of the Computer Science courses at UniNA)
  - WooCommerce (e-commerce websites solution)
  - MediaWiki (powers Wikipedia, Fandom, WikiHow, …)
- WordPress is the most widely used (used by **~43% of all websites**!)
- Note that some of these (e.g.: WordPress) feature a **decoupled architecture**.

USING A (TRADITIONAL) CMS

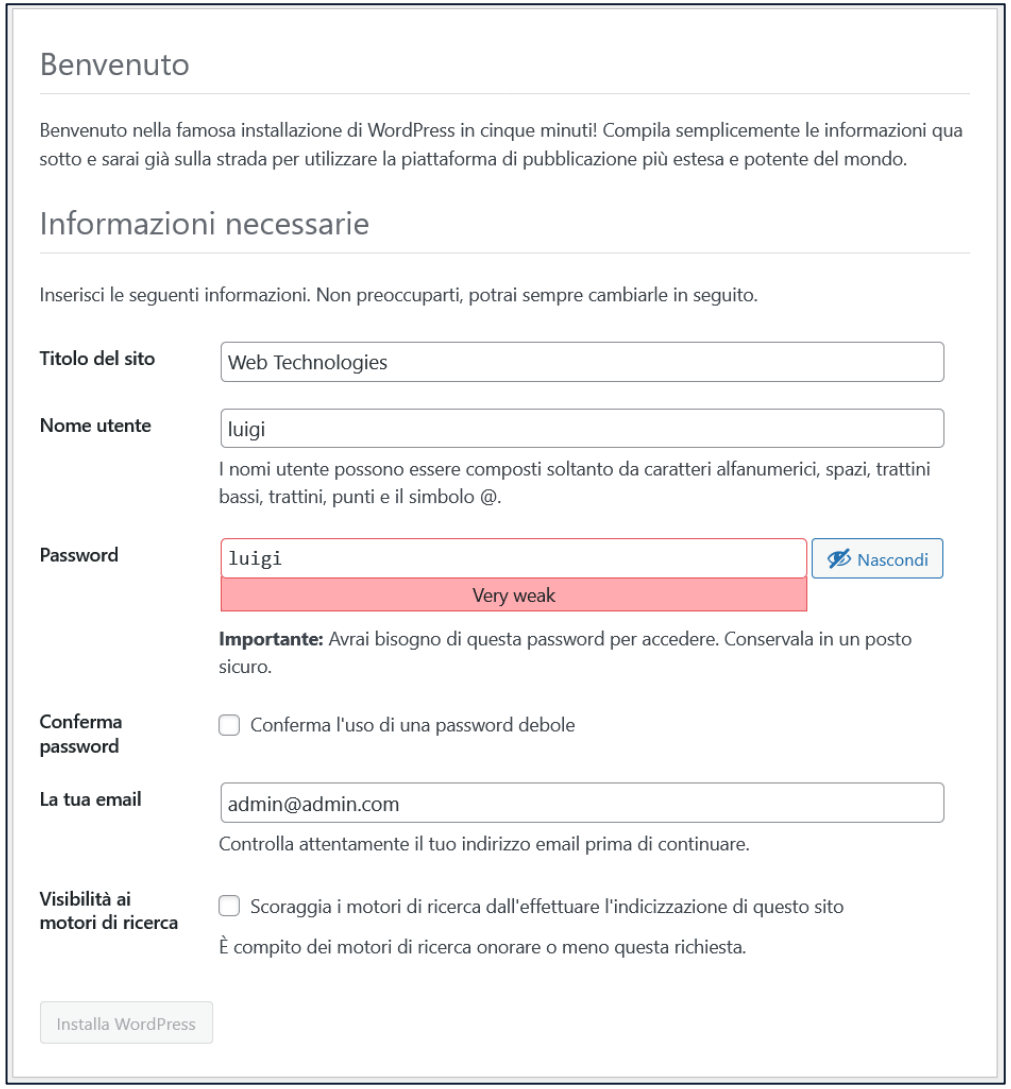
- A traditional CMS is typically a web app as any other
- In the initial install phase, it requires the user to specify the data necessary to connect to a relational database (e.g.: url, username, password)
- Upon connection, it creates the necessary tables
- Generally, during the install phase, the CMS asks the user to specify the credentials (username, password) for the administration account
- After install, the admin can create and organize content, add new users, define the templates and customize the appearence of the website, etc… Unauthorized users can access the content.

USING A (TRADITIONAL) CMS

- Let’s install WordPress (using Docker) and let’s check it out
- Live demo time!

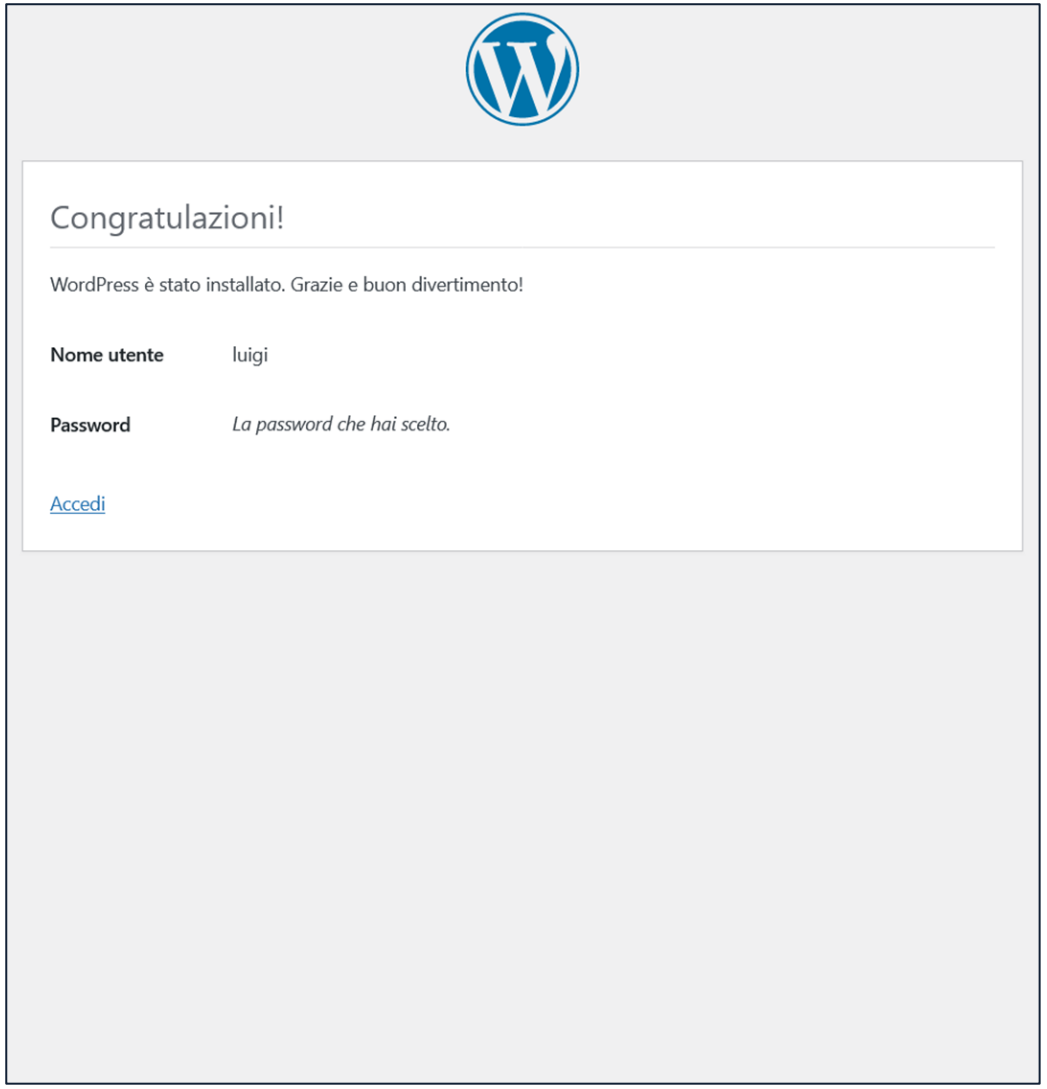
INSTALLING WORDPRESS

- Data for the connection to the database is passed via environment variables (see **compose.yml** file)
- Here is the web page where we specify the title of the website and we create an admin user

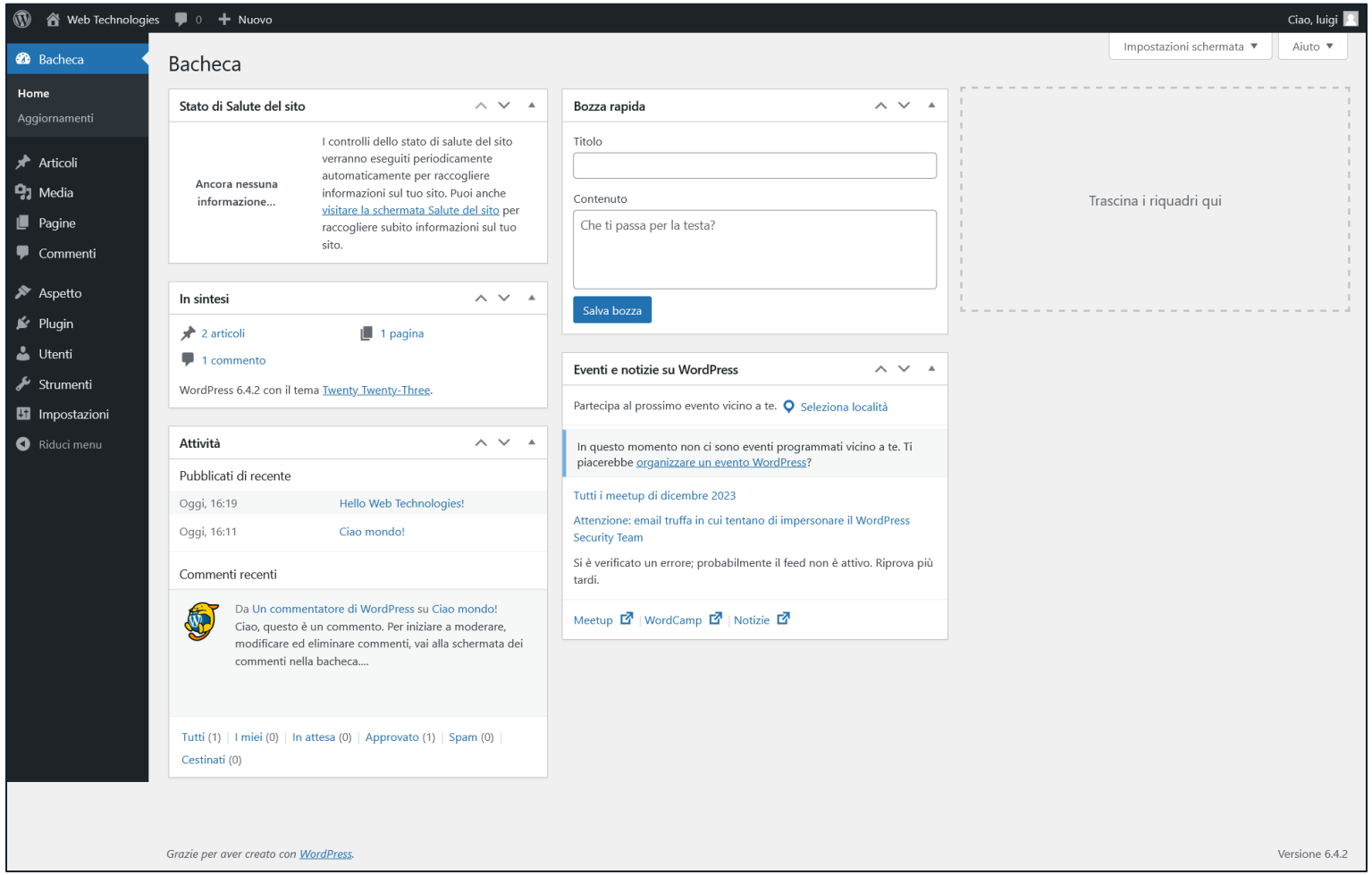


INSTALLING WORDPRESS

- Done. WordPress has been installed.
- We can now login using the admin account we created.
- An example homepage has been created for us. We can access it at http://localhost



WORDPRESS: ADMIN DASHBOARD



HEADLESS CMSs

HEADLESS CMS

Popular Headless CMS include:

- Strapi (based on Node.js)
- Wagtails (based on the Django framework for Python)
- Ghost (also based on Node.js)
- CrafterCMS (based on Java)

STRAPI: GETTING STARTED

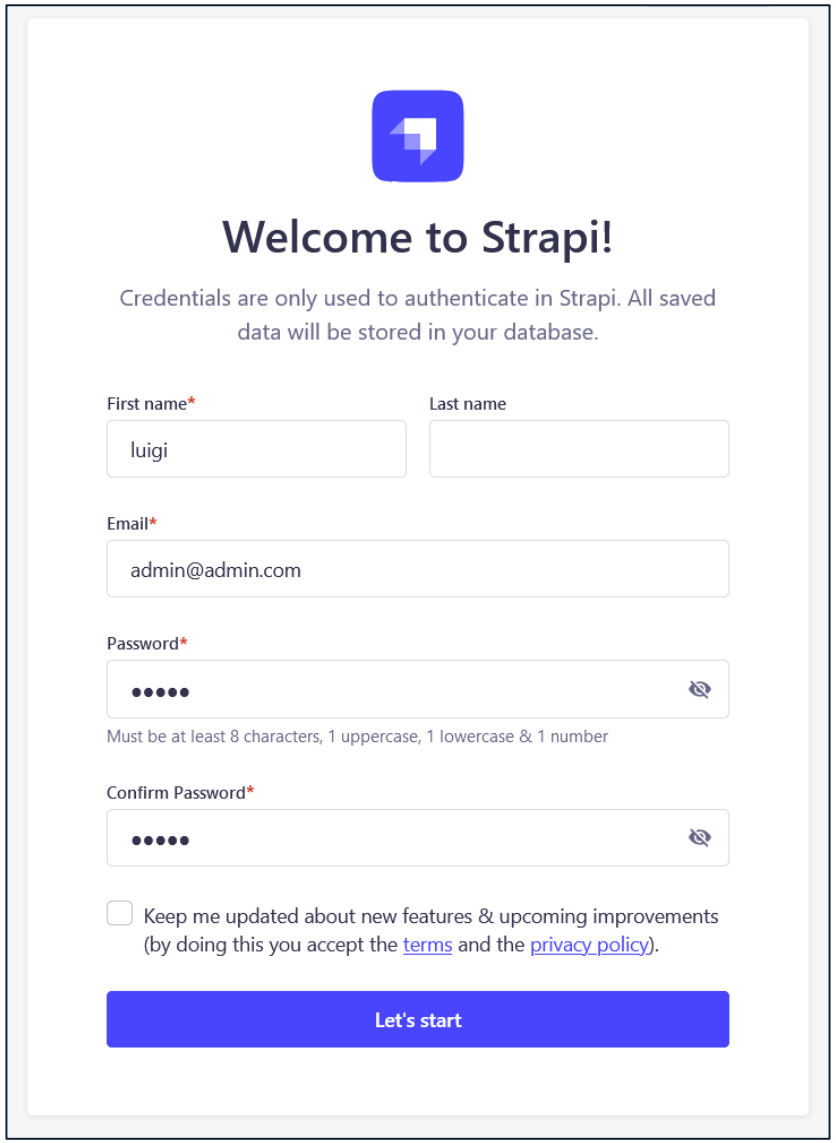
```
@luigi → D/O/T/W/2/e/1/strapi $ npx create-strapi-app@latest project-name

@luigi → D/O/T/W/2/e/1/strapi $ cd project-name

@luigi → D/O/T/W/2/e/1/strapi/project-name $ npm run develop
```

- **npx create-strapi-app** is the fastest way to get started
- A wizard will
  - Ask for some basic information about our project
  - Create an npm project
  - Install dependencies
- **npm run develop** can be executed to run the Strapi app in dev mode

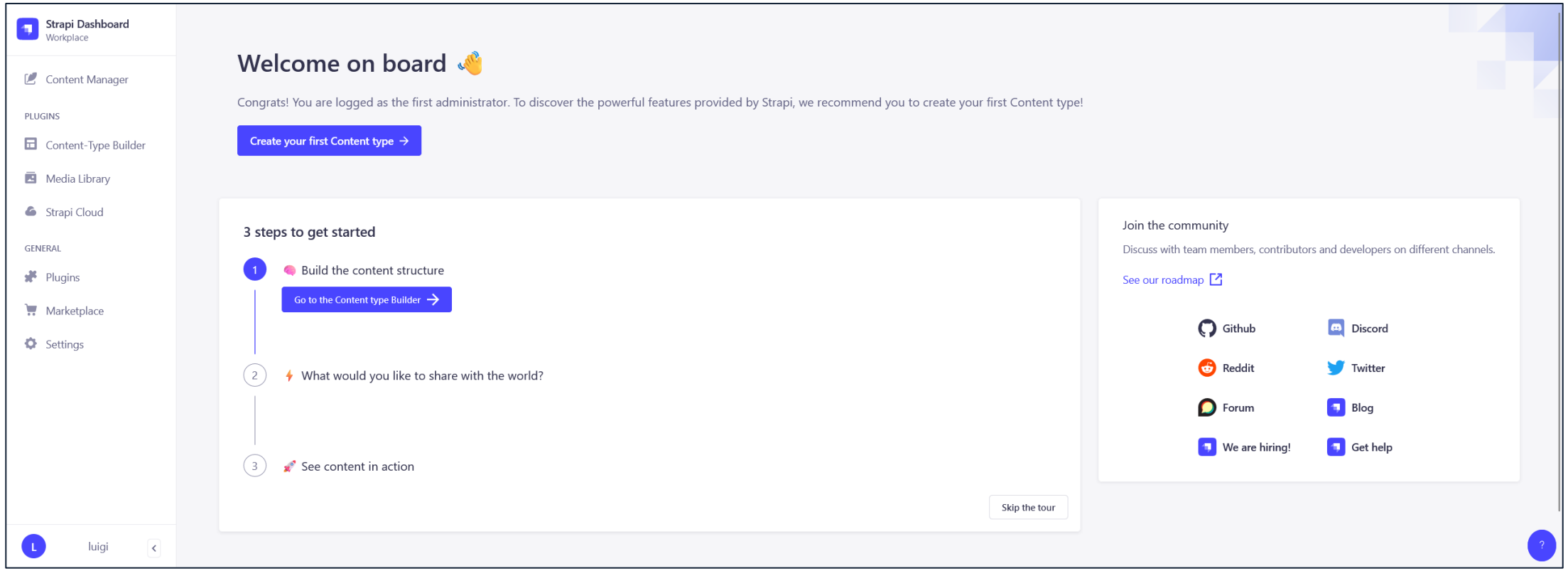
STRAPI: GETTING STARTED

```
@luigi ➜ D/O/T/W/2/e/1/strapi/project-name $ npm run develop

> todo-api@0.1.0 develop
> strapi develop

✓ Building build context (577ms)
✓ Creating admin (16224ms)
✓ Loading Strapi (1505ms)
✓ Generating types (1413ms)

Welcome back!

To manage your project 🚀, go to the administration panel at:
http://localhost:1337/admin

To access the server ⚡, go to:
http://localhost:1337
```

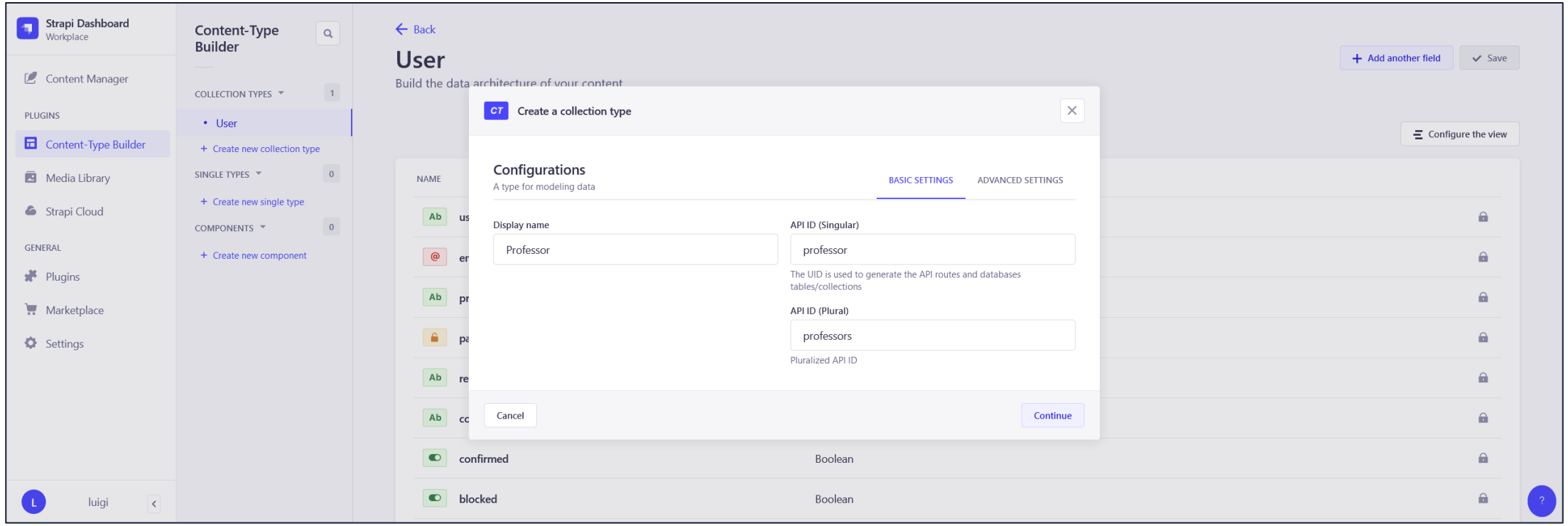
STRAPI: CREATING THE ADMIN USER



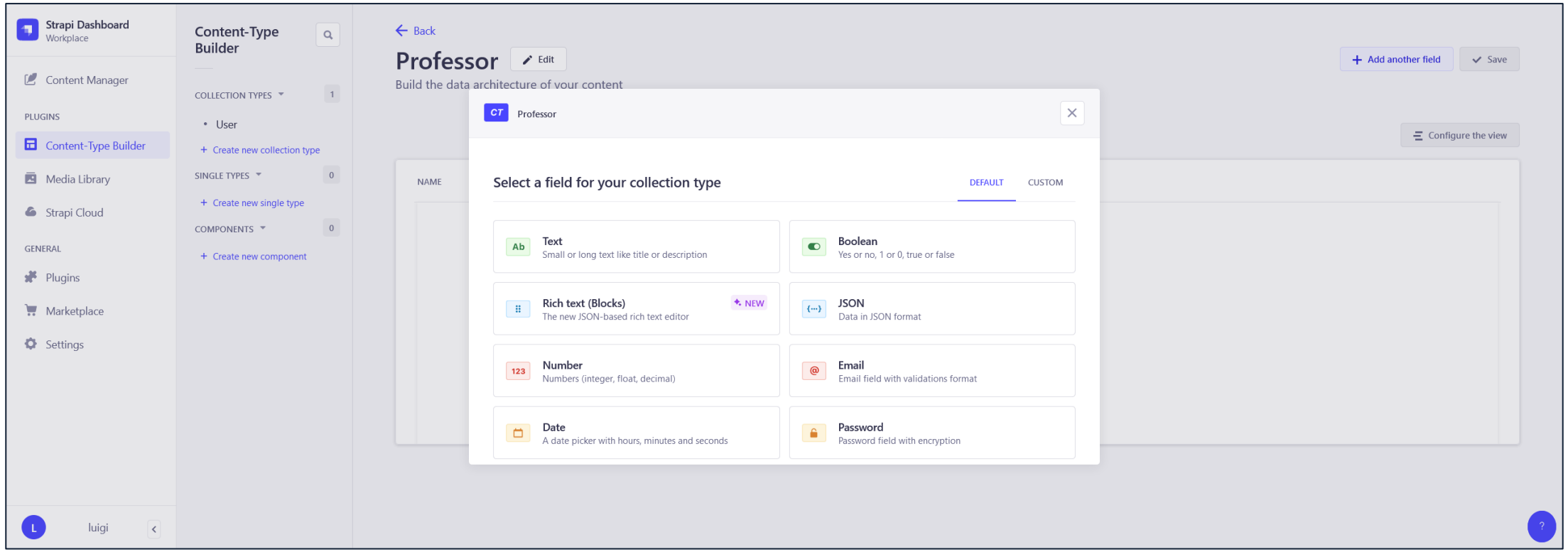
STRAPI: ADMIN DASHBOARD



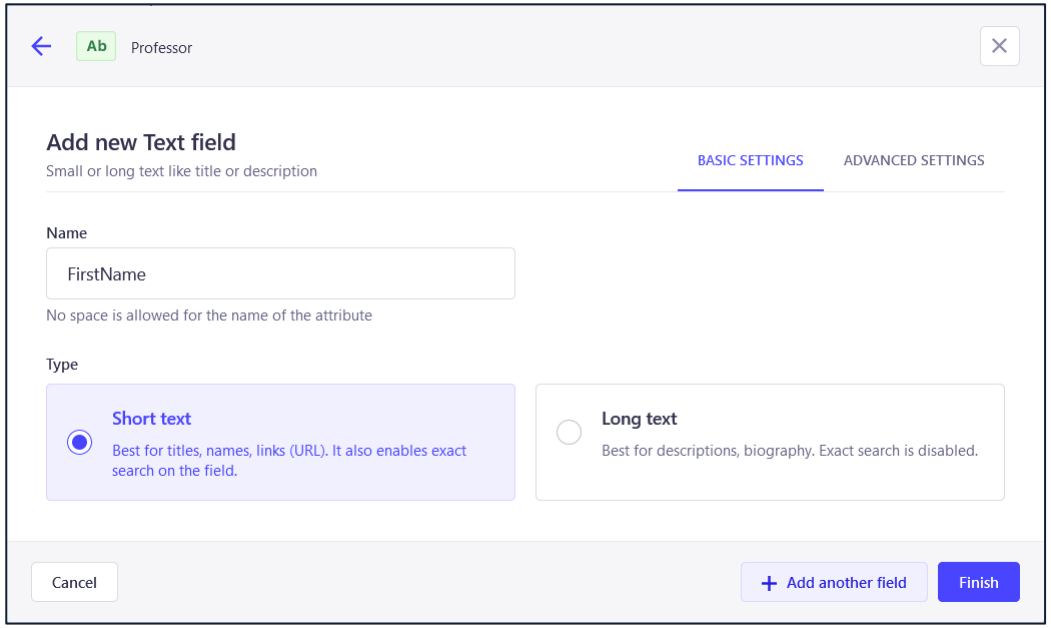
STRAPI: CREATING RESOURCES

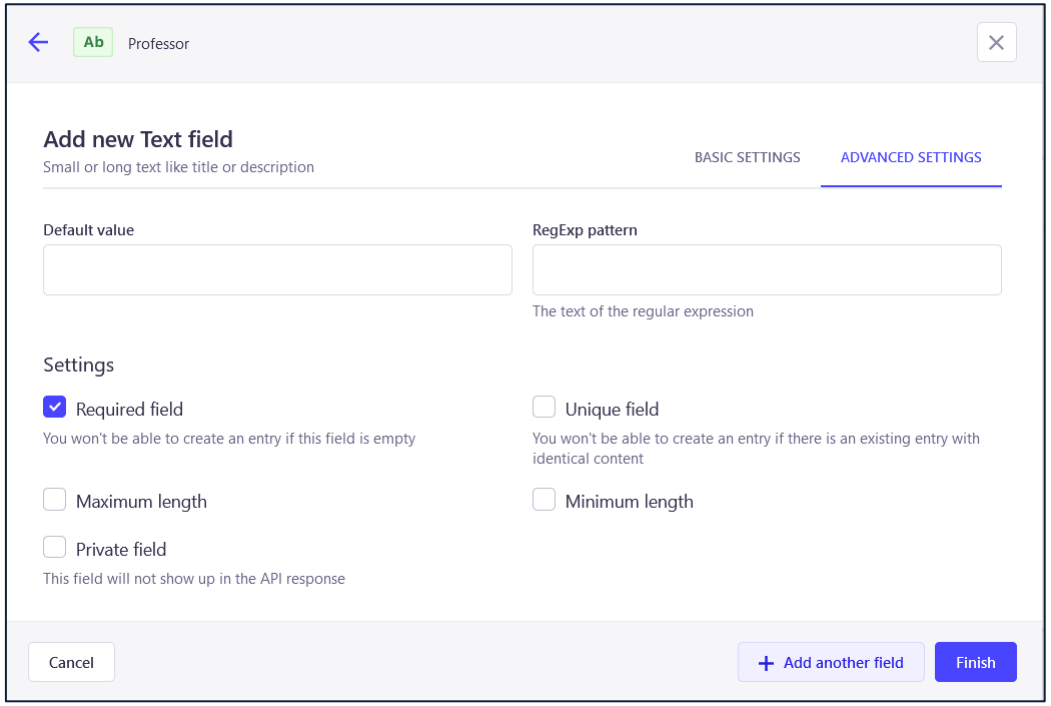


STRAPI: DEFINING RESOURCES

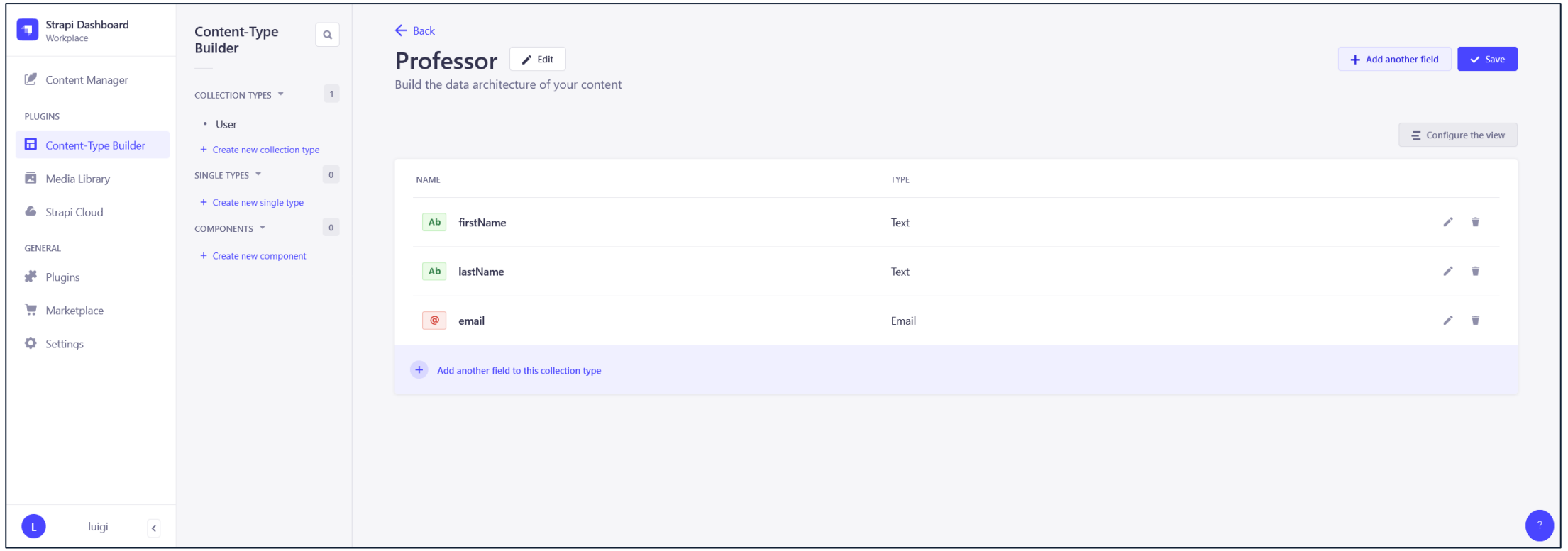


STRAPI: DEFINING RESOURCES

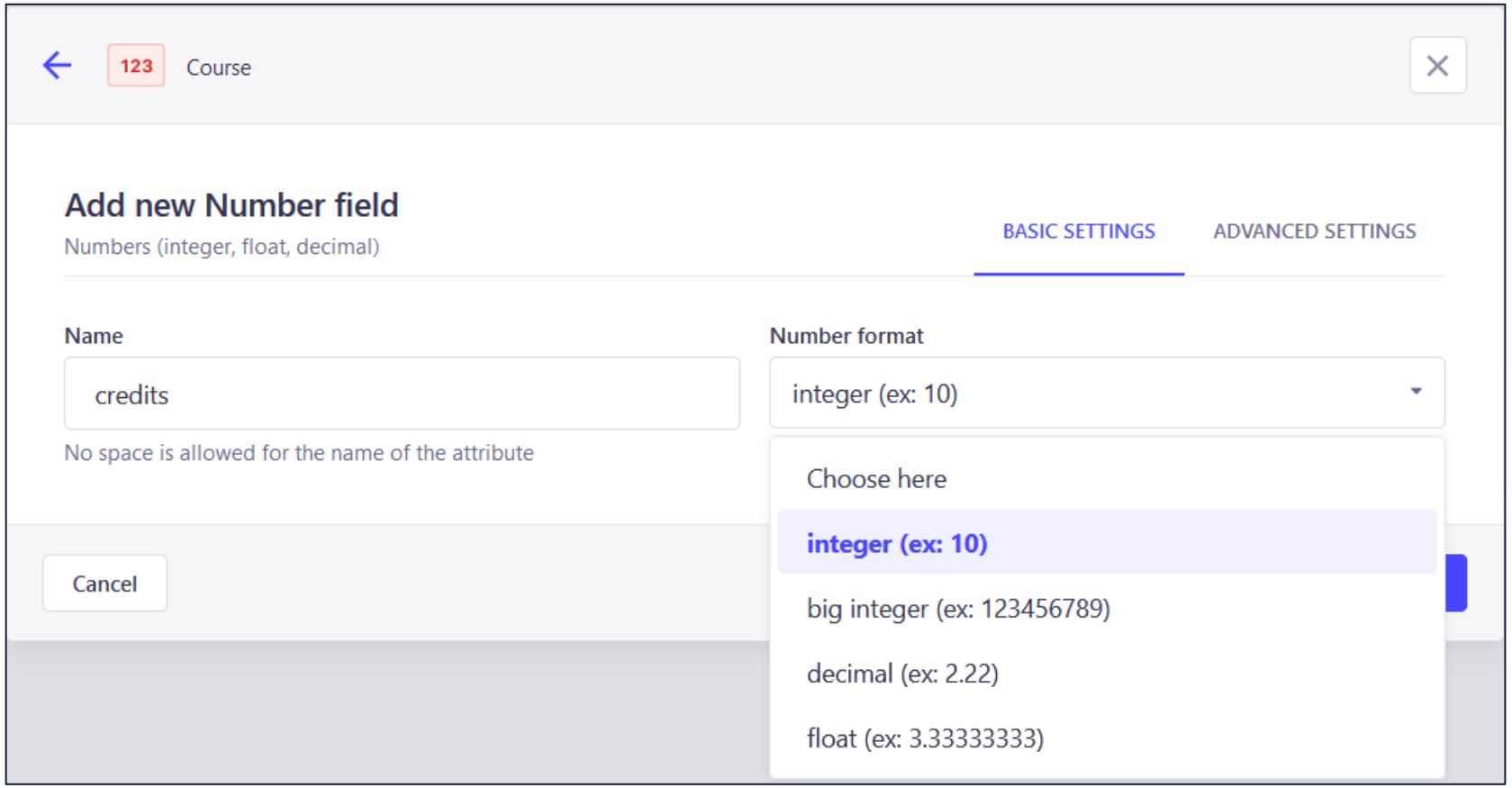




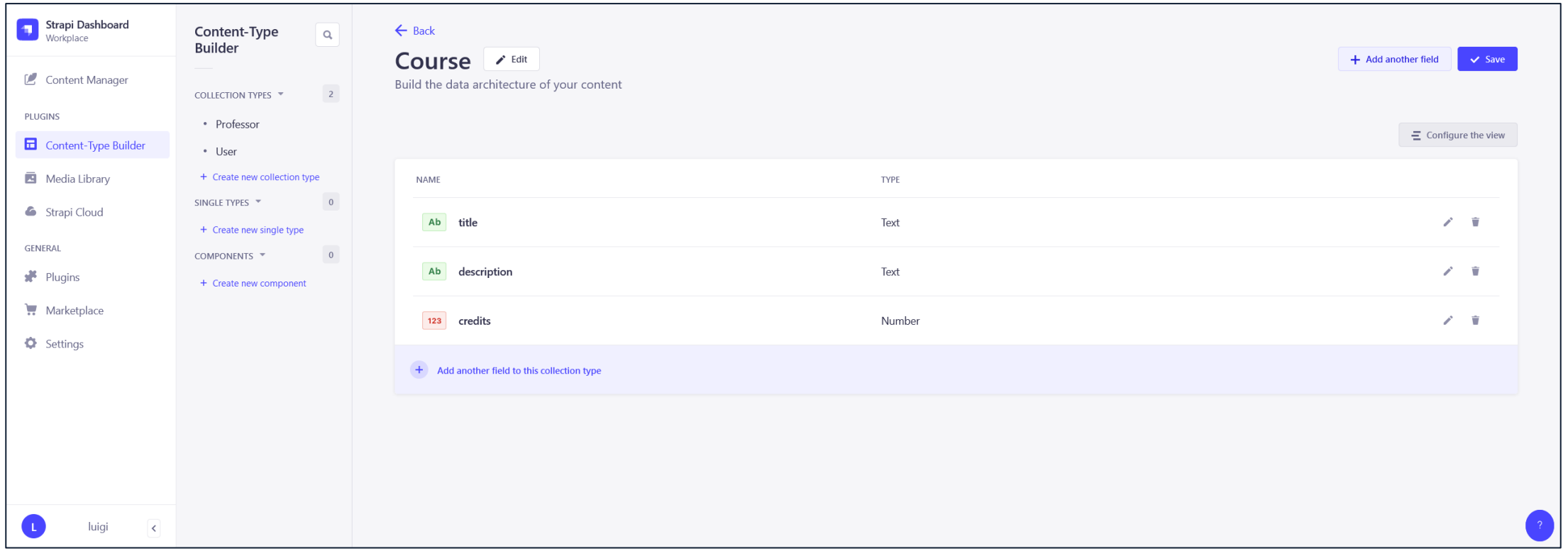
STRAPI: DEFINING RESOURCES



STRAPI: DEFINING RESOURCES

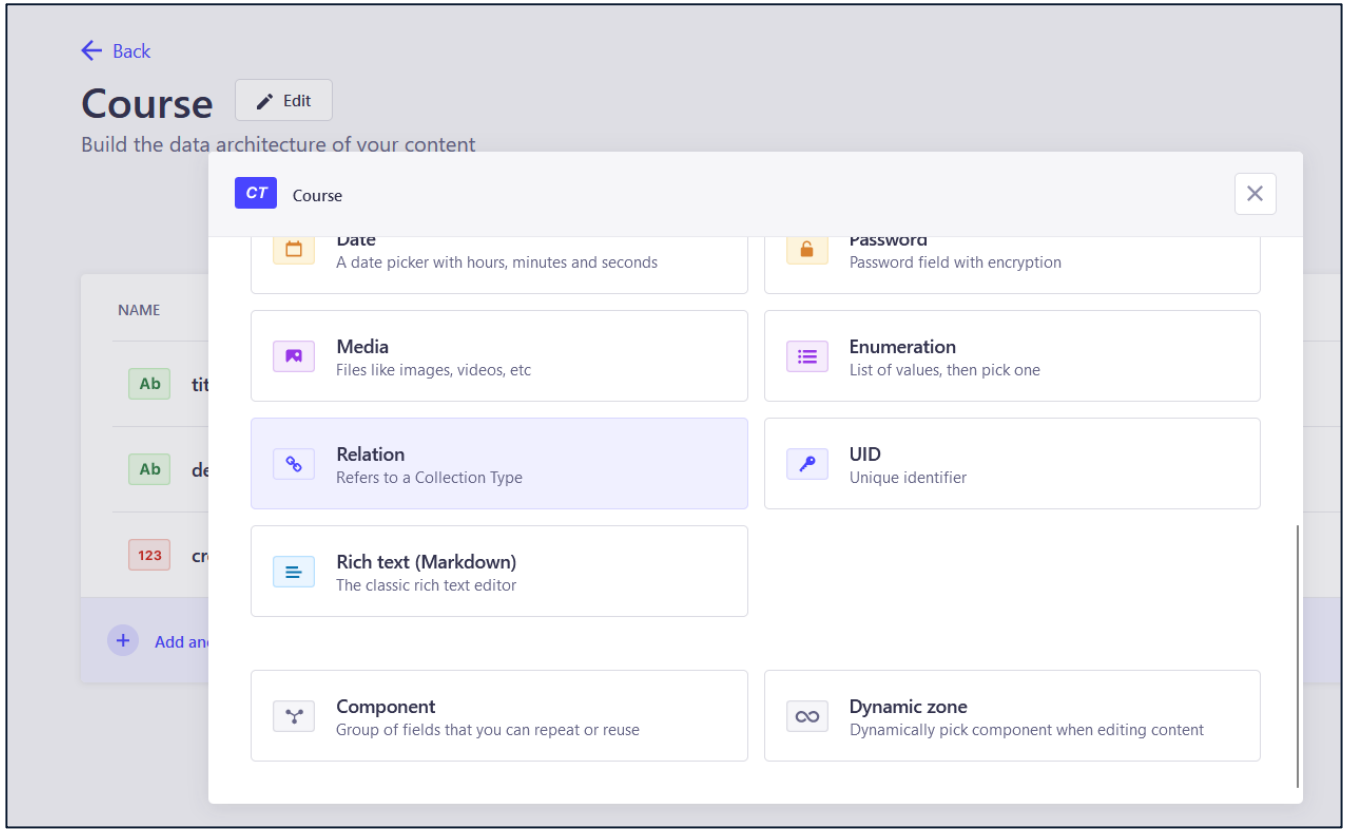


STRAPI: DEFINING RESOURCES

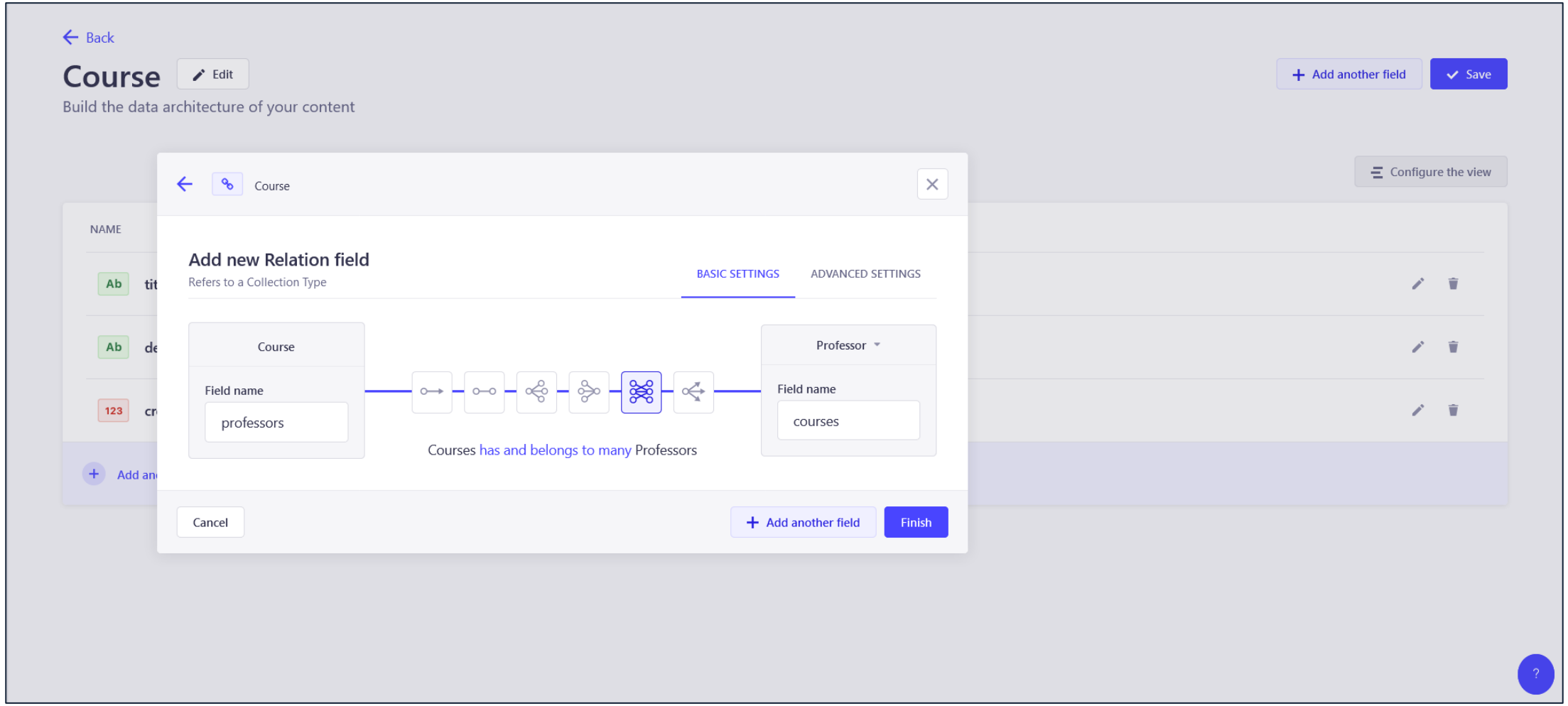


STRAPI: RELATIONS BETWEEN RESOURCES

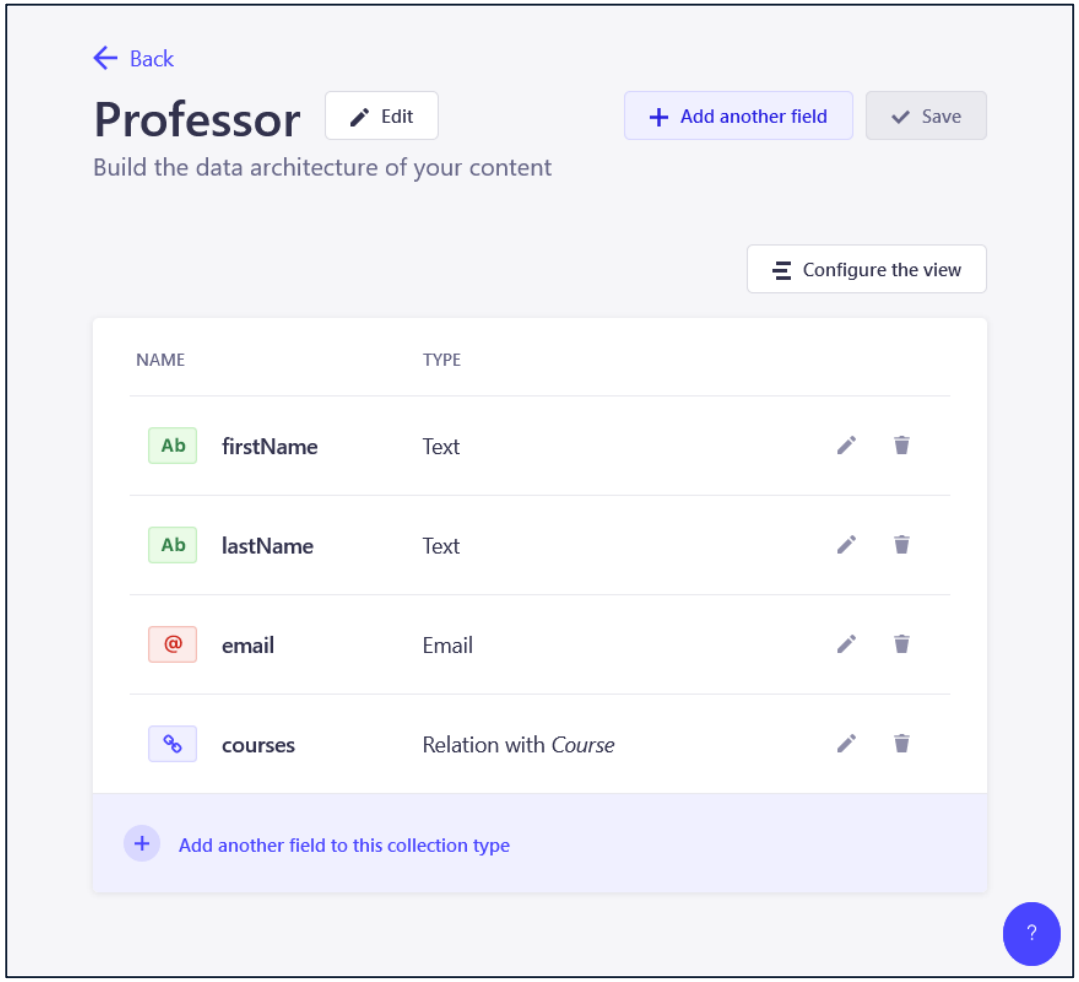
- Strapi allows us to define relations between resources using the visual editor

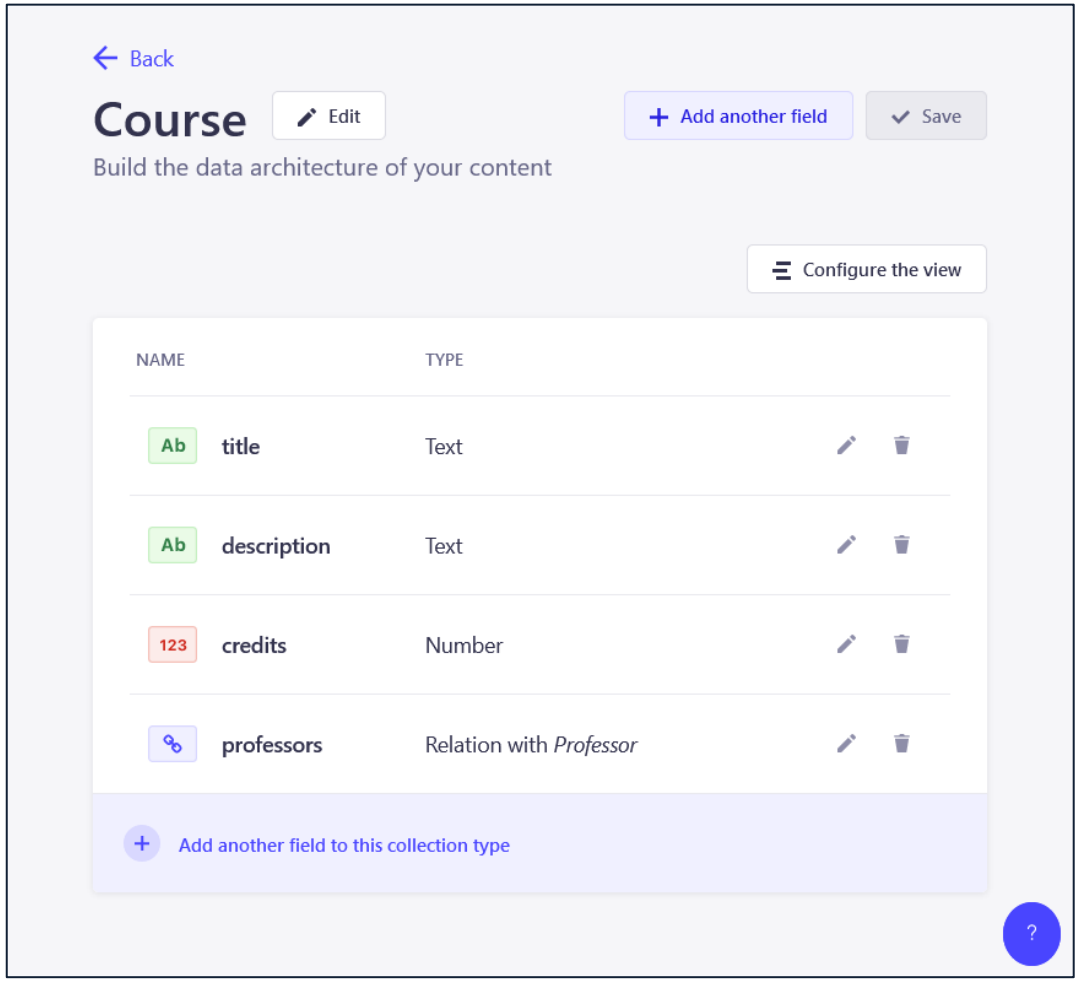


STRAPI: RELATIONS BETWEEN RESOURCES



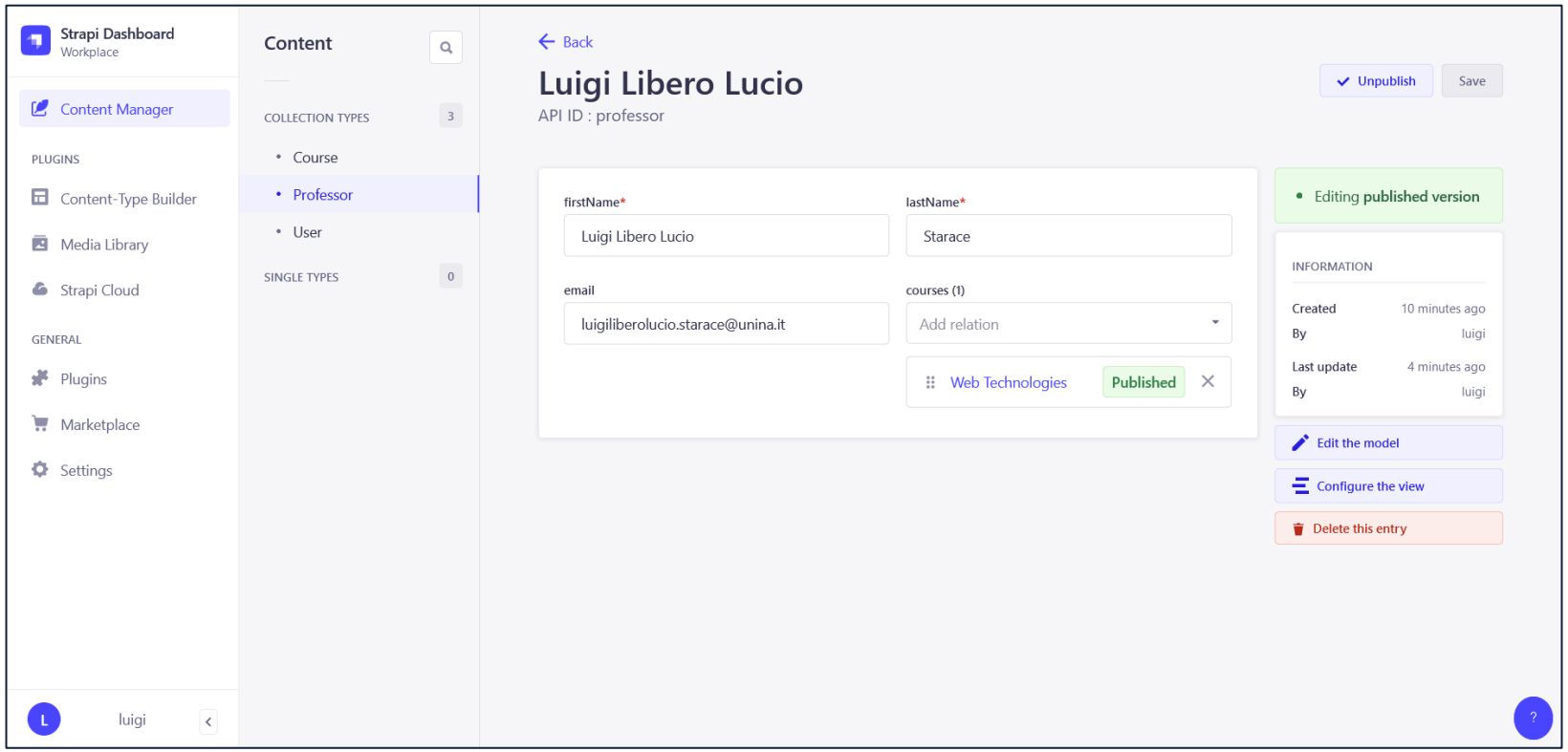
STRAPI: RELATIONS BETWEEN RESOURCES





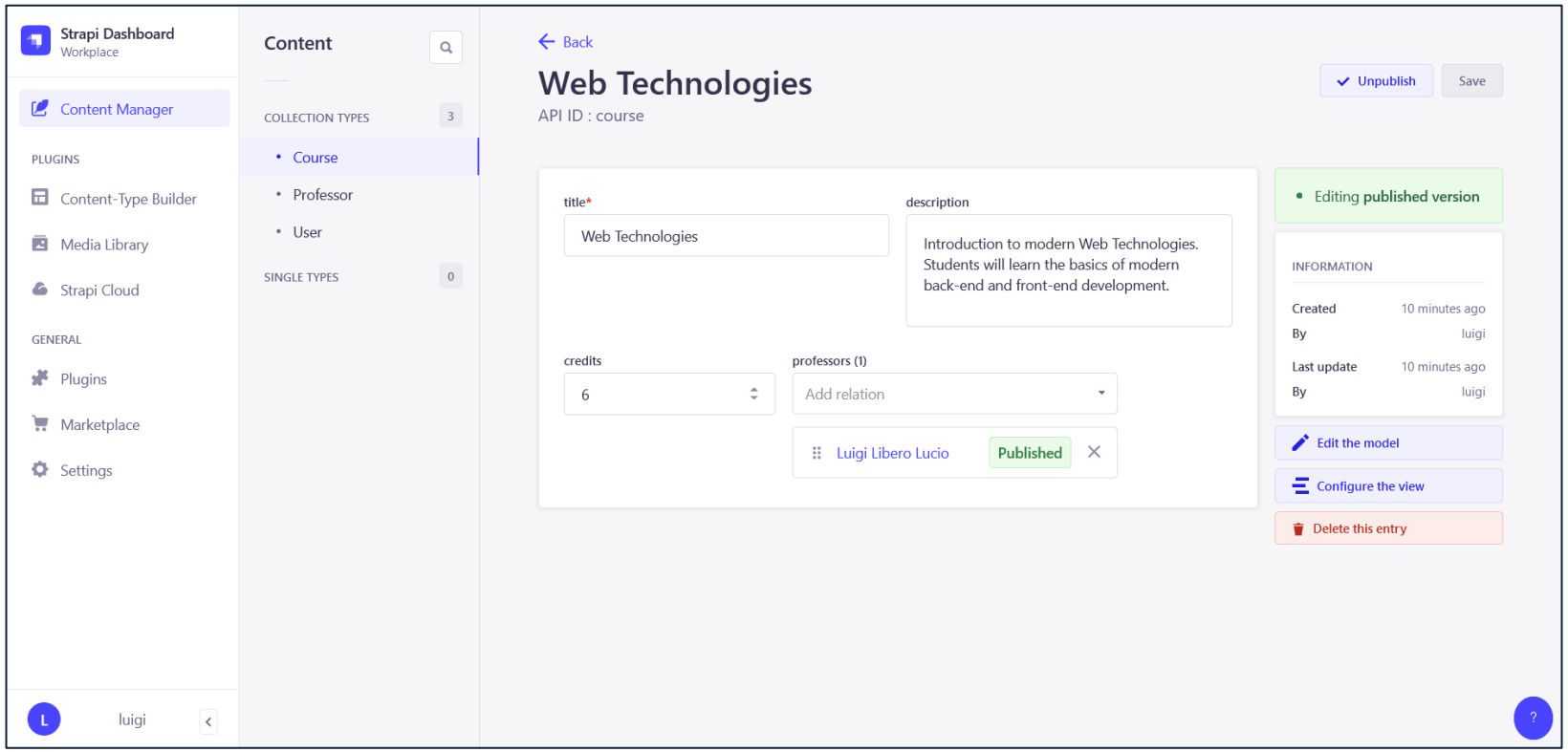
STRAPI: CREATING CONTENT

- So far, we defined the «shape» of the resources we want to manage
- Strapi allows us to create the actual content, using its GUI

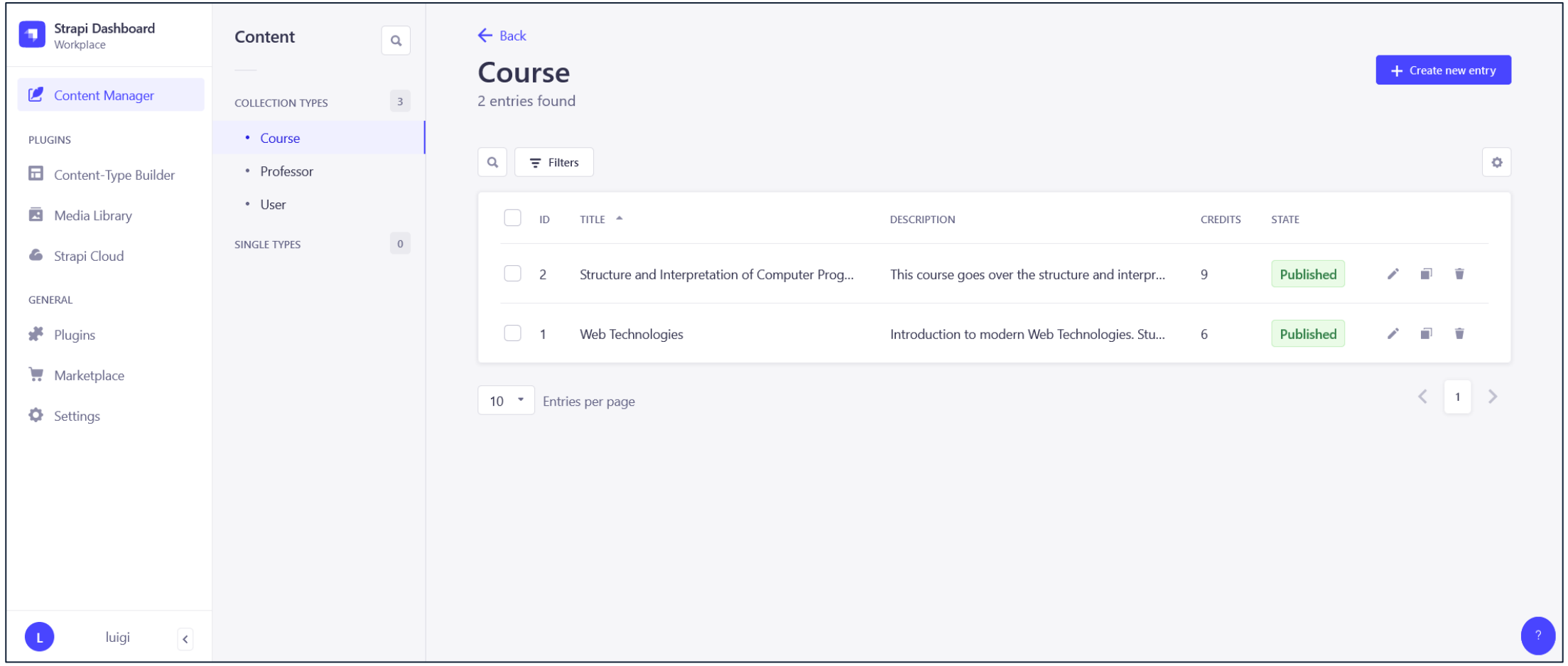


STRAPI: CREATING CONTENT

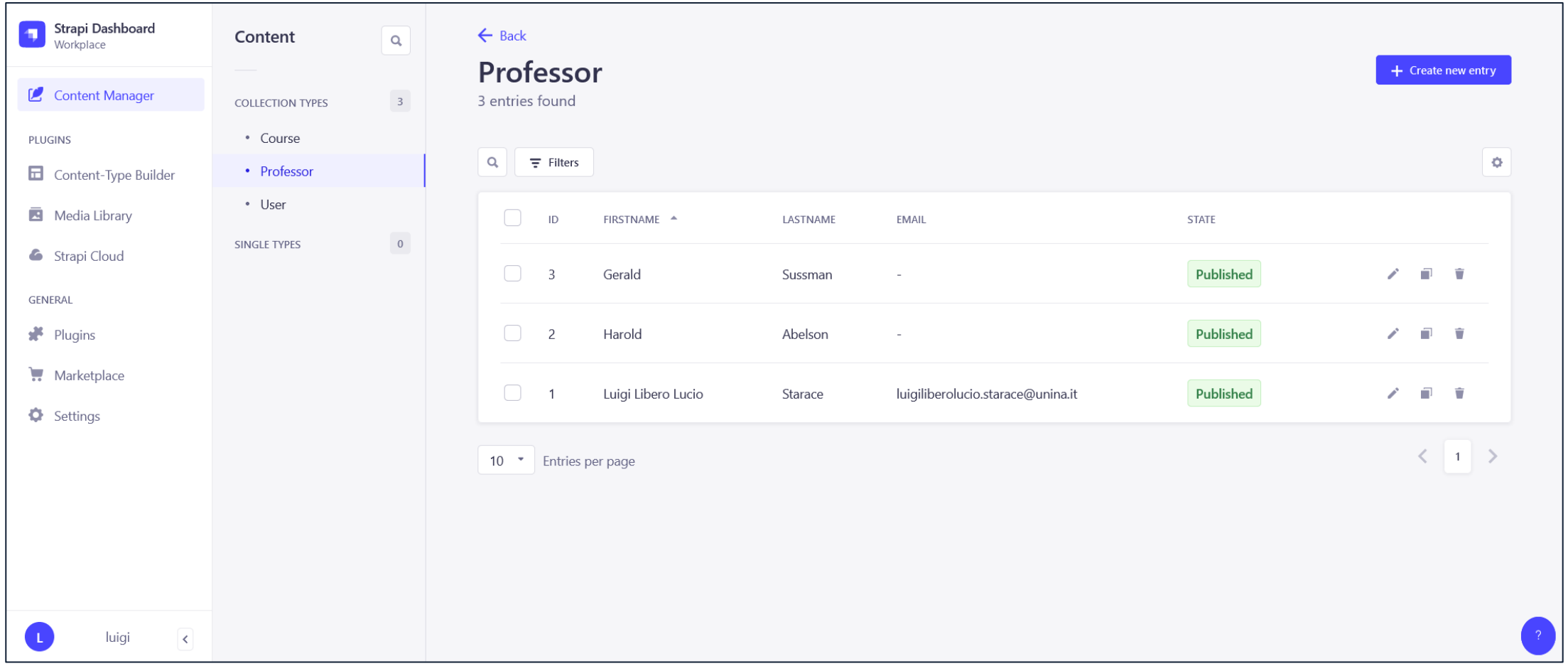
- So far, we defined the «shape» of the resources we want to manage
- Strapi allows us to create the actual content, using its GUI



STRAPI: MANAGING CONTENT



STRAPI: MANAGING CONTENT



STRAPI: USING THE REST API

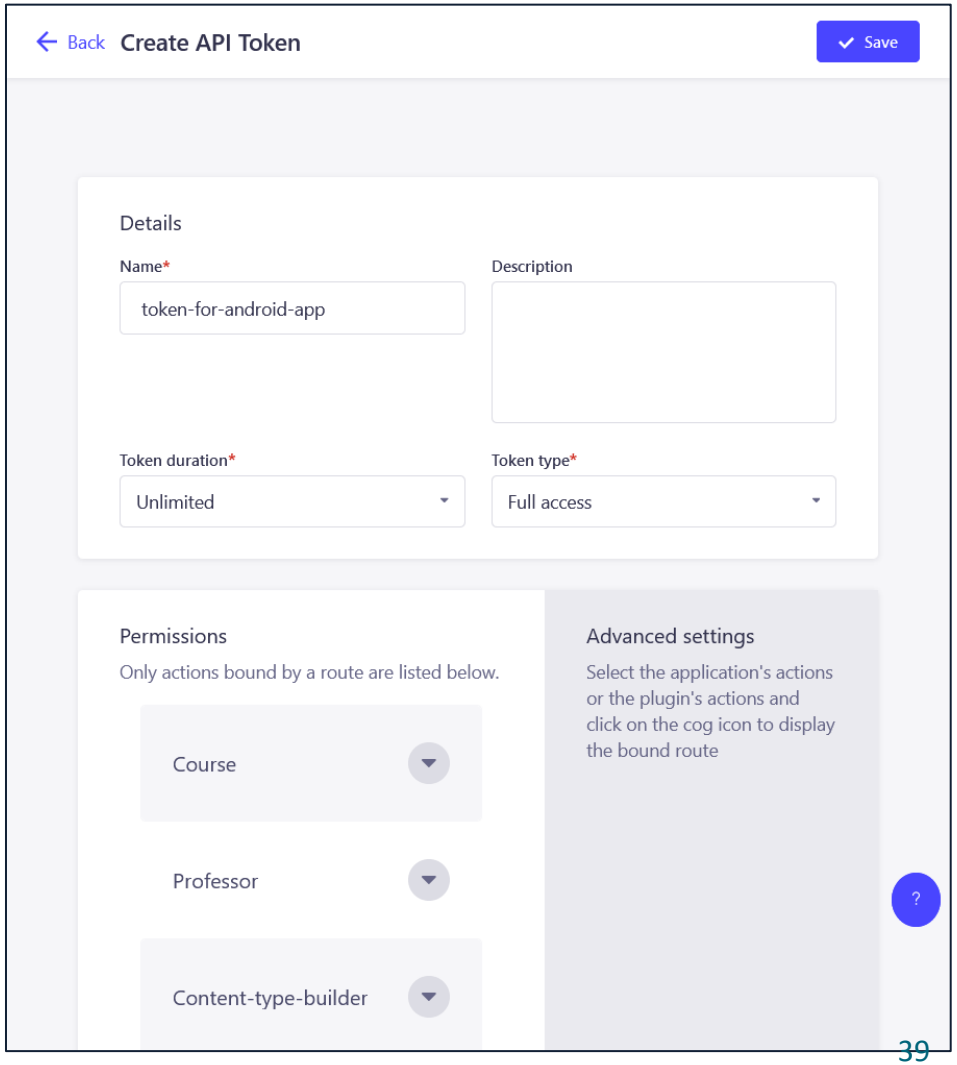
- For each resource, Strapi automatically generates REST endpoints

Method	URL	Description
GET	`/api/:pluralApiId`	Get list of entries
POST	`/api/:pluralApiId`	Create entry
GET	`/api/:pluralApiId/:resourceId`	Get a specific entry
PUT	`/api/:pluralApiId/:resourceId`	Update a specific entry
DELETE	`/api/:pluralApiId/:resourceId`	Delete a specific entry

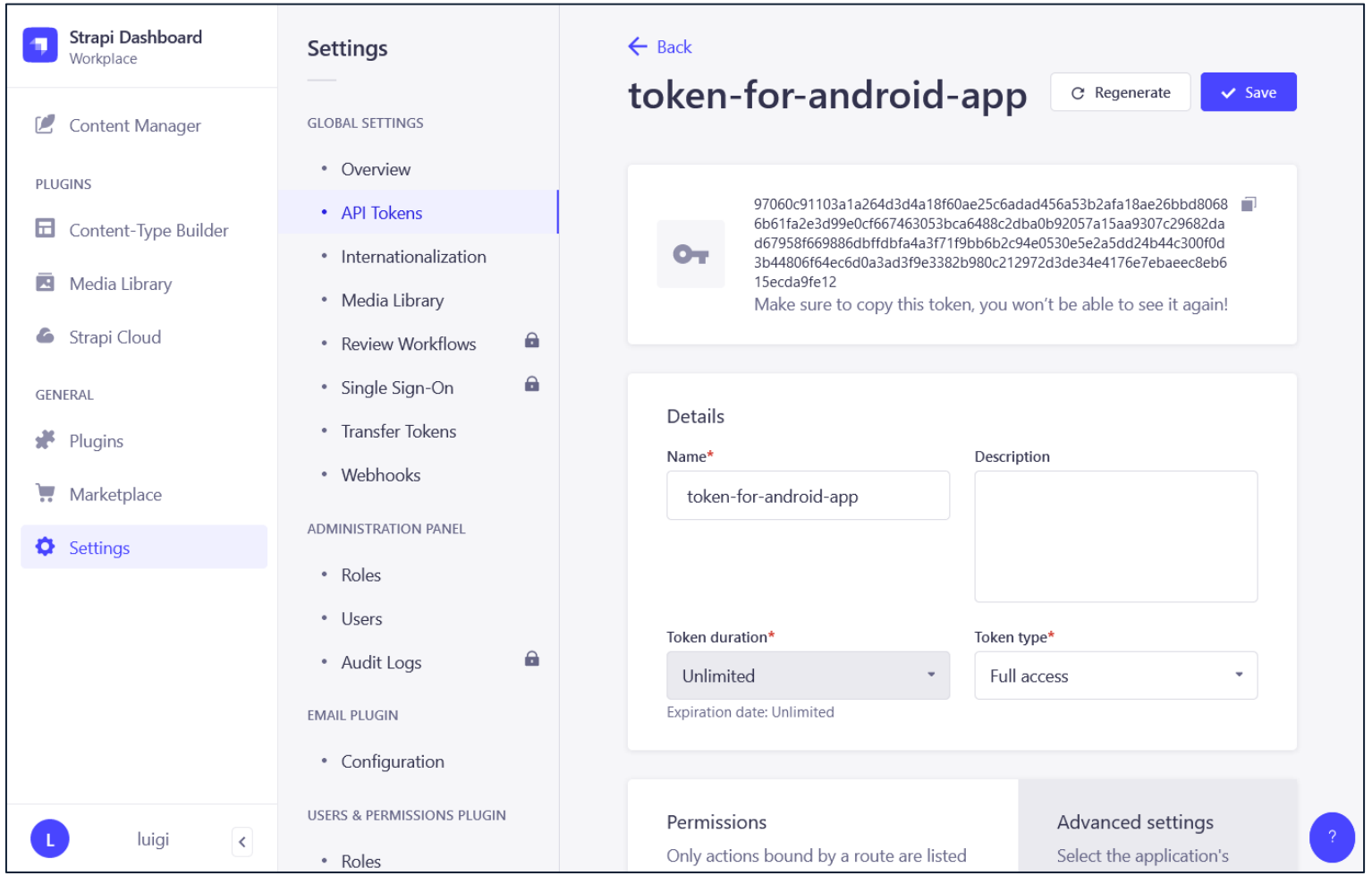
- E.g.: `GET /api/courses` or `DELETE /api/professors/2`

STRAPI: AUTHORIZING USERS

- The Strapi UI allows us to create authorization tokens
  - Settings > API Tokens > Create
- With each token, can specify a name, a duration, and read/write permissions for each resource
- Users will need to pass the token in the Authorization header when using the API (unless we specify that an endpoint can be accessed without authorization)



STRAPI: AUTHORIZING USERS



STRAPI: USING THE REST API

```
GET http://localhost:1337/api/professors
Authorization: Bearer 97060c911...e12
```


```
{
  "data": [
    {
      "id": 1,
      "attributes": {
        "firstName": "Luigi Libero Lucio",
        "lastName": "Starace",
        "email": "luigiliberolucio.starace@unina.it",
        "createdAt": "2023-12-29T07:19:19.746Z",
        "updatedAt": "2023-12-29T07:25:16.666Z",
        "publishedAt": "2023-12-29T07:20:35.307Z"
      }
    }, ...
```

STRAPI: USING THE REST API

```
GET http://localhost:1337/api/professors/2
Authorization: Bearer 97060c911...e12
```


```
{
  "data": {
    "id": 2,
    "attributes": {
      "firstName": "Harold",
      "lastName": "Abelson",
      "email": null,
      "createdAt": "2023-12-29T07:24:16.695Z",
      "updatedAt": "2023-12-29T07:24:17.915Z",
      "publishedAt": "2023-12-29T07:24:17.913Z"
    }
  },
  "meta": {}
}
```

STRAPI: USING THE REST API

```
POST http://localhost:1337/api/professors
Authorization: Bearer 97060c911...e12
Content-Type: application/json

{
  "data": {
    "firstName": "Anders",
    "lastName": "Hejlsberg",
    "courses": [3, 4]
  }
}
```

```
{
  "data": {
    "id": 7,
    "attributes": {
      "firstName": "Anders",
      "lastName": "Hejlsberg",
      "email": null,
      "createdAt": "2023-12-29T08:16:53.132Z",
      "updatedAt": "2023-12-29T08:16:53.132Z",
      "publishedAt": "2023-12-29T08:16:53.126Z"
    }
  },
  "meta": {}
}
```

STRAPI: REST API PARAMETERS

- Strapi endpoints support a number of Query String parameters to **filter**, **sort**, **paginate** results, and to **select fields** and **populate relations**

Parameter	Description
`populate`	Controls which linked resources should be included in the results
`fields`	Selects which fields to include in the result, and in which order
`filters`	Can be used to define custom queries and filter data
`sort`	Controls how the results are sorted
`pagination`	Can be used to control result pagination

STRAPI: POPULATING LINKED RESOURCES

```
GET http://localhost:1337/api/professors?populate=courses
Authorization: Bearer 97060c911...e12
```


```
{"data": [{
     "id": 1,
     "attributes": {
       "firstName": "Luigi Libero Lucio",
       "lastName": "Starace",
       "courses": {
         "data": [{
             "id": 1,
             "attributes": {
               "title": "Web Technologies",
               "credits": 6
             }
          }
```

STRAPI: CUSTOMIZING RETRIEVED FIELDS

```
GET http://localhost:1337/api/professors?fields[0]=lastName&fields[1]=firstName
Authorization: Bearer 97060c911...e12
```

```
{"data": [{
      "id": 1,
      "attributes": {
        "lastName": "Starace",
        "firstName": "Luigi Libero Lucio"
      }
    }, {
      "id": 2,
      "attributes": {
        "lastName": "Abelson",
        "firstName": "Harold"
      }
    }, ...
```

STRAPI: FILTERS

The filters parameter has the following syntax

`GET /api/:pluralApiId?filters[field][operator]=value`

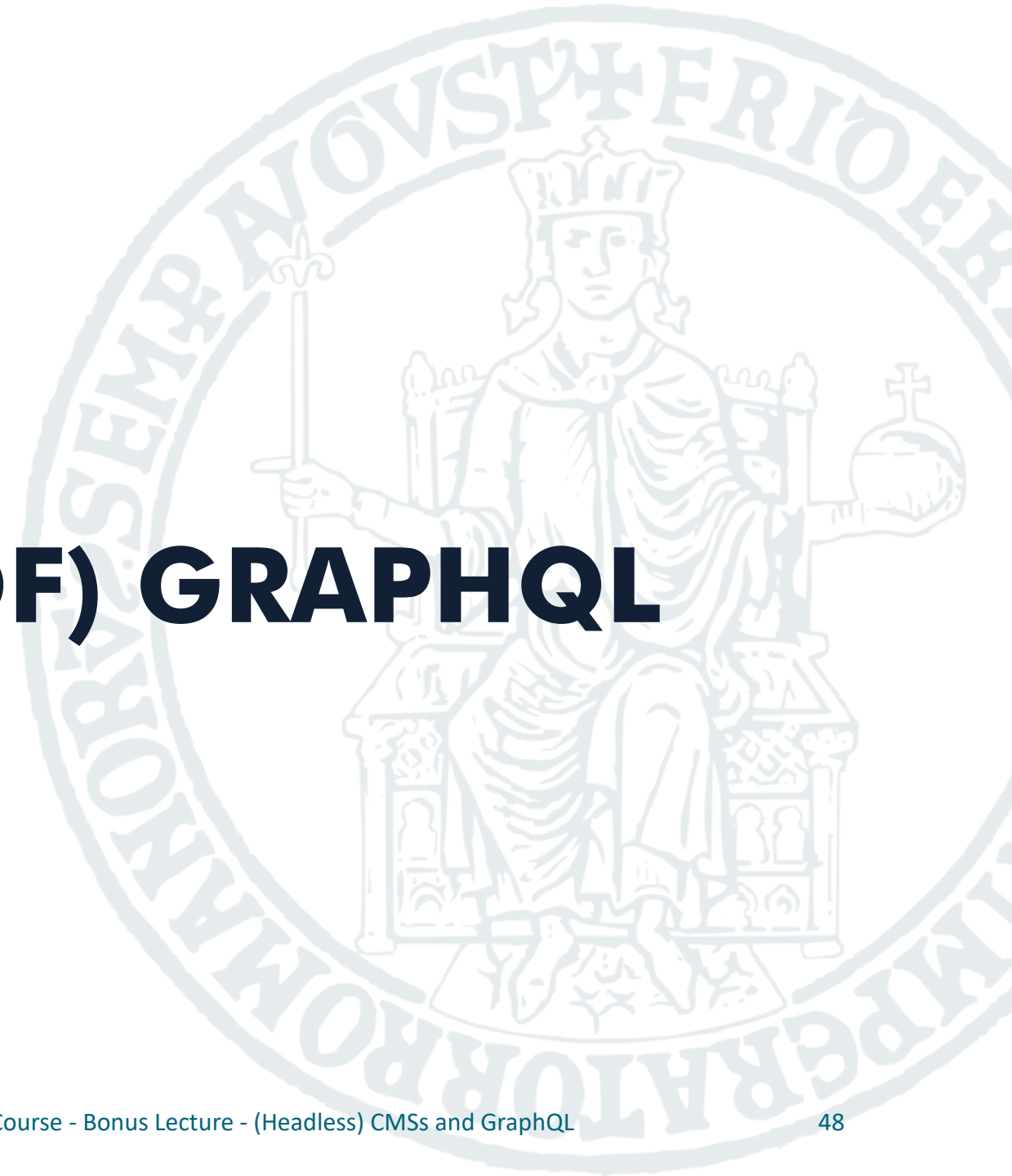
```
//get courses where credits = 6
GET /api/courses?filters[credits][$eq]=6
Authorization: Bearer 97060c911...e12
```

```
//get courses where credits >= 6 and title contains the string "Web"
GET /api/courses?filters[credits][$gte]=6&filters[title][$contains]=Web
Authorization: Bearer 97060c911...e12
```

```
//get professors whose firstName contains (case-insensitive) the string "Luigi"
GET /api/professors?filters[firstName][$containsi]=Luigi
Authorization: Bearer 97060c911...e12
```

Look at the docs for an exhaustive list of operators and more examples

(AN OVERVIEW OF) GRAPHQL



SOME ISSUES WITH REST

REST is a practical, widely used way to expose data from a server.

In some scenarios, however, the REST approach can be inefficient:

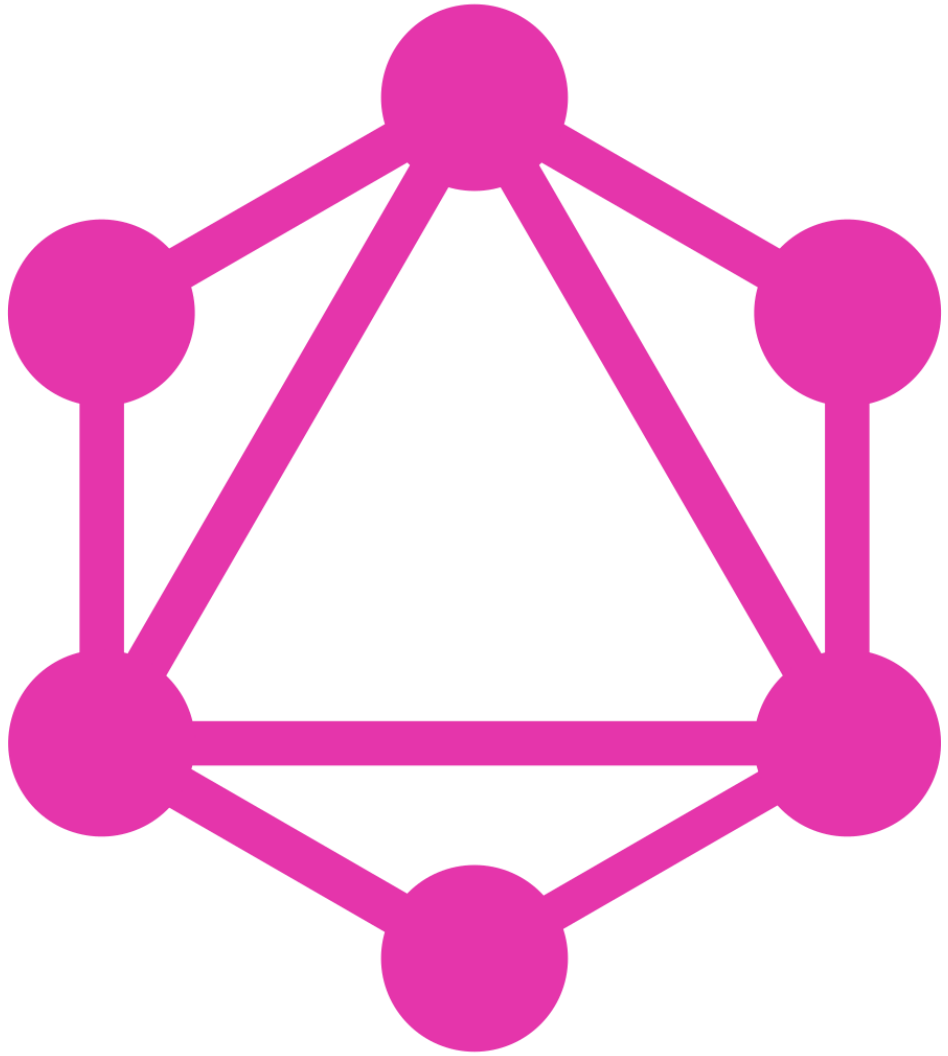
- **Over-fetching**
  - Sometimes we do not need all the fields of a resource
- **Under-fetching**
  - Sometimes we need linked resources, and it might be necessary to perform additional requests to fetch them
- **API changes and evolution**
  - APIs evolve over time. Thus, they need to be versioned, maintain a degree of retro-compatibility, deprecate some functionality over time, …

REST: OVER– AND UNDER–FETCHING

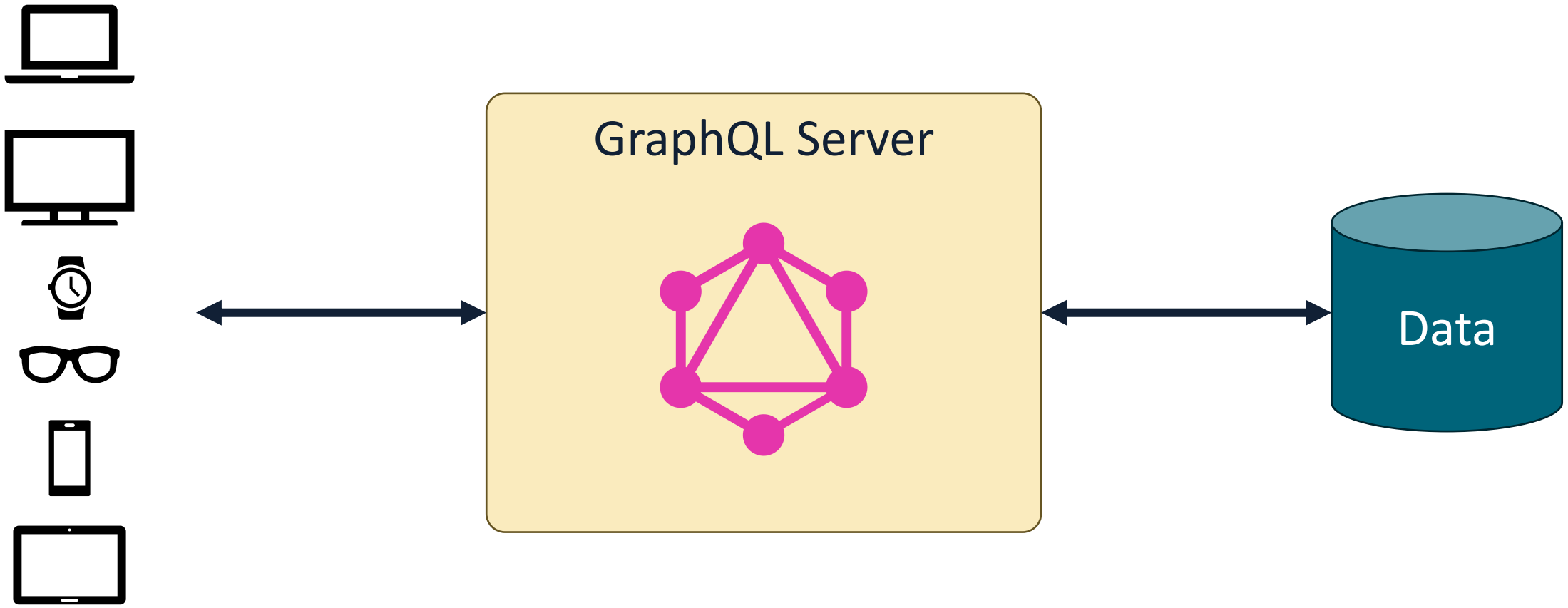
- Over-fetching and Under-fetching can both deteriorate performance and increase data transfers
- In an over-fetching scenario, we download data we do not need
- In an under-fetching scenario, we will need to perform (many) additional requests to get all the data we need
  - E.g.: first we get the current user, then we get its posts, then, for each post, we get its comments, etc…

GRAPHQL: A QUERY LANGUAGE FOR APIs

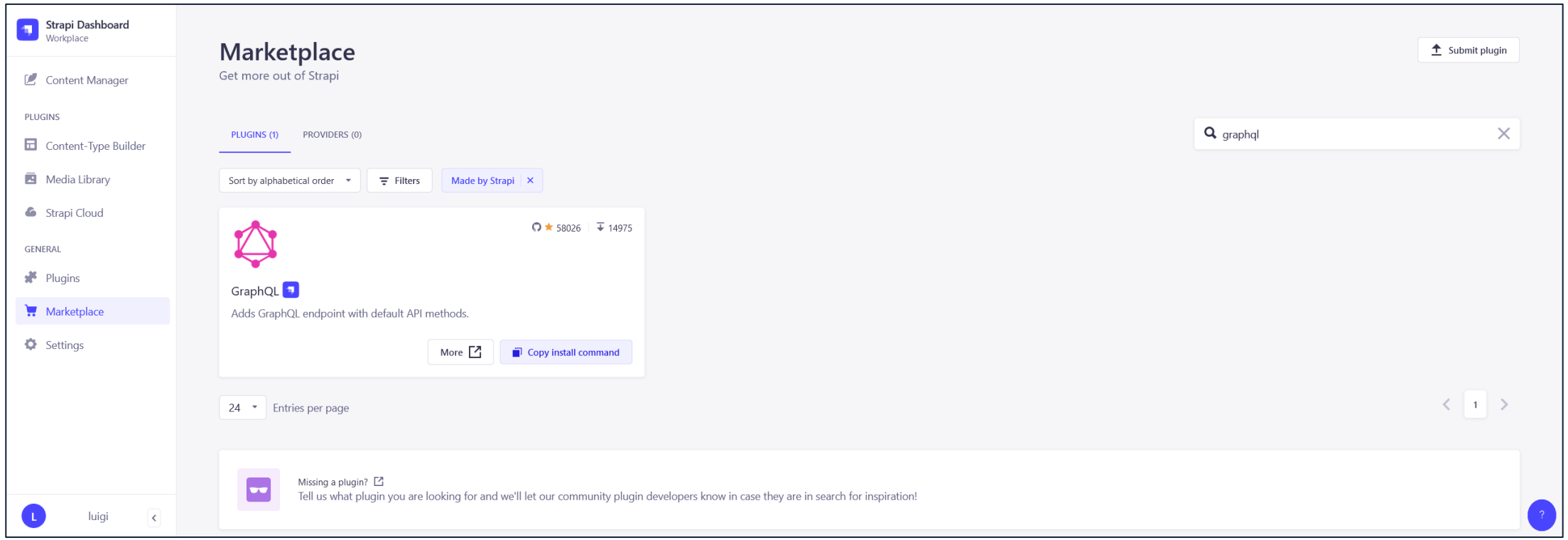
- Developed by Facebook
- Open source version published in 2015
- Specification available at http://spec.graphql.org/
- Designed to address the need for more **flexibility** and **efficiency**
- Instead of relying on pre-defined endpoints as in REST, allow users to **declaratively specify** which data they need using queries!



GRAPHQL: ARCHITECTURE



STRAPI WITH GRAPHQL: EXAMPLE




```
@luigi → D/O/T/W/2/e/1/strapi/project-name $ npm install @strapi/plugin-graphql
```

STRAPI WITH GRAPHQL: EXAMPLE

If you get this error when starting Strapi after installing the plugin:

```
Duplicate "graphql" modules cannot be used at the same time since different
versions may have different capabilities and behavior. The data from one version
used in the function from another could produce confusing and spurious results.
```

Add the following to the package.json file of the Strapi project:

```
"overrides": {
  "graphql": "^16"
},
```

STRAPI WITH GRAPHQL: EXAMPLE

- After restarting Strapi, a GraphQL Playground will be available at http://localhost:1337/graphql
- In the web technologies course we won't see the GraphQL specification in detail
- Let's just grasp the basics with a few examples

STRAPI WITH GRAPHQL: EXAMPLES

- Suppose we need to display only the last name of each professor
- We might write the following GraphQL query

```
query {
  professors {
    data {
      attributes {
        lastName
      }
    }
  }
}
```

```
{
  "data": {
    "professors": {
      "data": [{
          "attributes": {
            "lastName": "Starace"
          }
        }, {
          "attributes": {
            "lastName": "Abelson"
          }
        }]
    }
  }
}
```

STRAPI WITH GRAPHQL: EXAMPLES

- Suppose we need to display only the last name of each professor and the title of their courses

```
query {
  professors {
    data {
      attributes {
        lastName,
        courses {
          data {
            attributes {
              title
            }
          }
        }
      }
}}}
```

```
{
  "data": {
    "professors": {
      "data": {
          "attributes": {
            "lastName": "Starace",
            "courses": {
              "data": [{
                  "attributes": {
                    "title": "Web Technologies“
                  }
               }]
             }
           }
```

STRAPI WITH GRAPHQL: EXAMPLES

```
SELECT title, credits FROM courses
WHERE credits >= 6 AND (title CONTAINS «Web» OR title
CONTAINS «TypeScript»)
```

```
query {
  courses(filters: {
    credits: { gte: 6},
    or: [{title: {containsi: "Web"}}, {title: {containsi:"TypeScript"}}]
  }) {
    data {
      attributes {
        title, credits
      }
    }
  }
}
```

AUTOMATIC GENERATION OF STATIC WEBSITES

SOMETIMES CMS CAN BE INEFFICIENT

- Sometimes using a full CMS for a website might be **overkill**
- In some scenarios, data does not change that often
  - Think of a restaurant that updates the menu on its website ~ once a month
  - For an entire month, each page load will cause a query on the database, which will retrieve the same list of menu entries, and then the CMS will display the same content to users
  - A CMS might be wasting computation cycles (i.e., **money**, and **CO_2**!) to render the same pages over and over again
  - Caching mechanisms exist, but still there's some overhead and some computing required
- CMS need to be kept up to date, which introduces some overhead

AUTOMATIC STATIC WEBSITE GENERATION

In some scenarios, having a CMS (or a custom web app) render the web pages dynamically might be quite inefficient and wasteful.

A more efficient approach would be to:

1. **Render** the web pages only once, when some changes in the data occur
2. **Serve** the compiled (**static**) web pages to users (which is way more efficient than rendering these pages dynamically!)

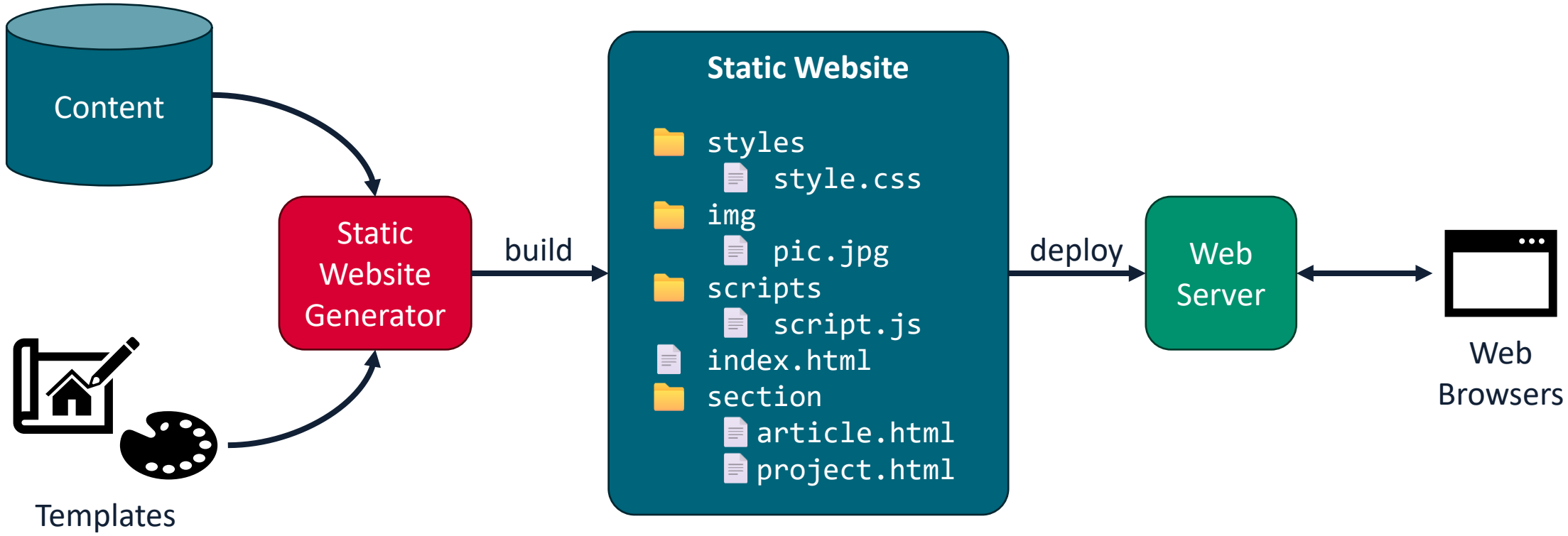
This approach is referred to as **Automatic Static Website Generation**

AUTOMATIC STATIC WEBSITE GENERATION

One approach to address these issues is to generate static websites automatically

- Hugo (written in Go, used for https://luistar.github.io)
- Gatsby (generate static websites using React)
- Jekyll (based on Ruby, used for https://fair-resilient-ai.github.io)
- Pelican (based on Python)
- Zola (based on Rust)
- Many more are listed here

AUTOMATIC STATIC WEBSITE GENERATION



REFERENCES

- **Definition of CMS**
  MDN web docs
  https://developer.mozilla.org/en-US/docs/Glossary/CMS
- **What is Headless CMS?**
  Amazon Web Services docs
  https://aws.amazon.com/what-is/headless-cms/
- **The Fullstack Tutorial for GraphQL**
  Free and open-source tutorial available at https://www.howtographql.com/
  Relevant parts: GraphQL Fundamentals (Introduction, GraphQL is the better REST, Core Concepts, Big Picture (Architecture)
- **What is a static site generator?**
  Article in the Cloudflare Learning Resources
  Available at https://www.cloudflare.com/learning/performance/static-site-generator/ and archived here