

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

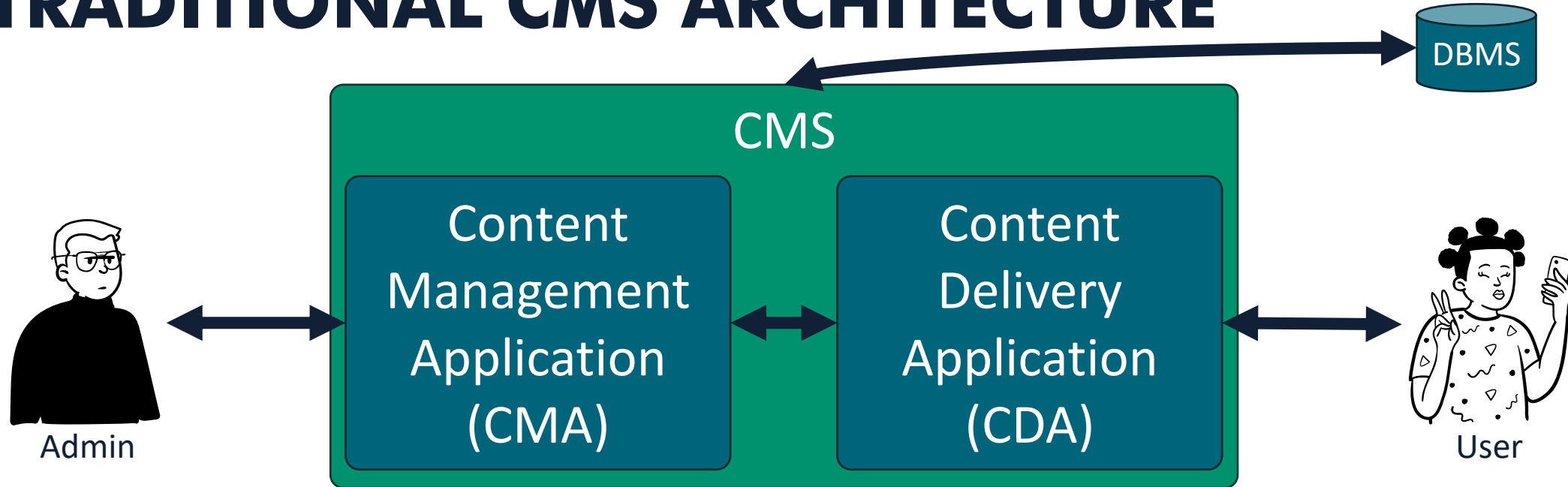
- A Content Management System (CMS) is a web application designed to allow collaborative **editing**, **organizing** and **publishing** of content
- Without the need to write a single line of code, using the User Interface (UI) provided by the CMS itself
 - Empowers non-technical users to manage and 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?)

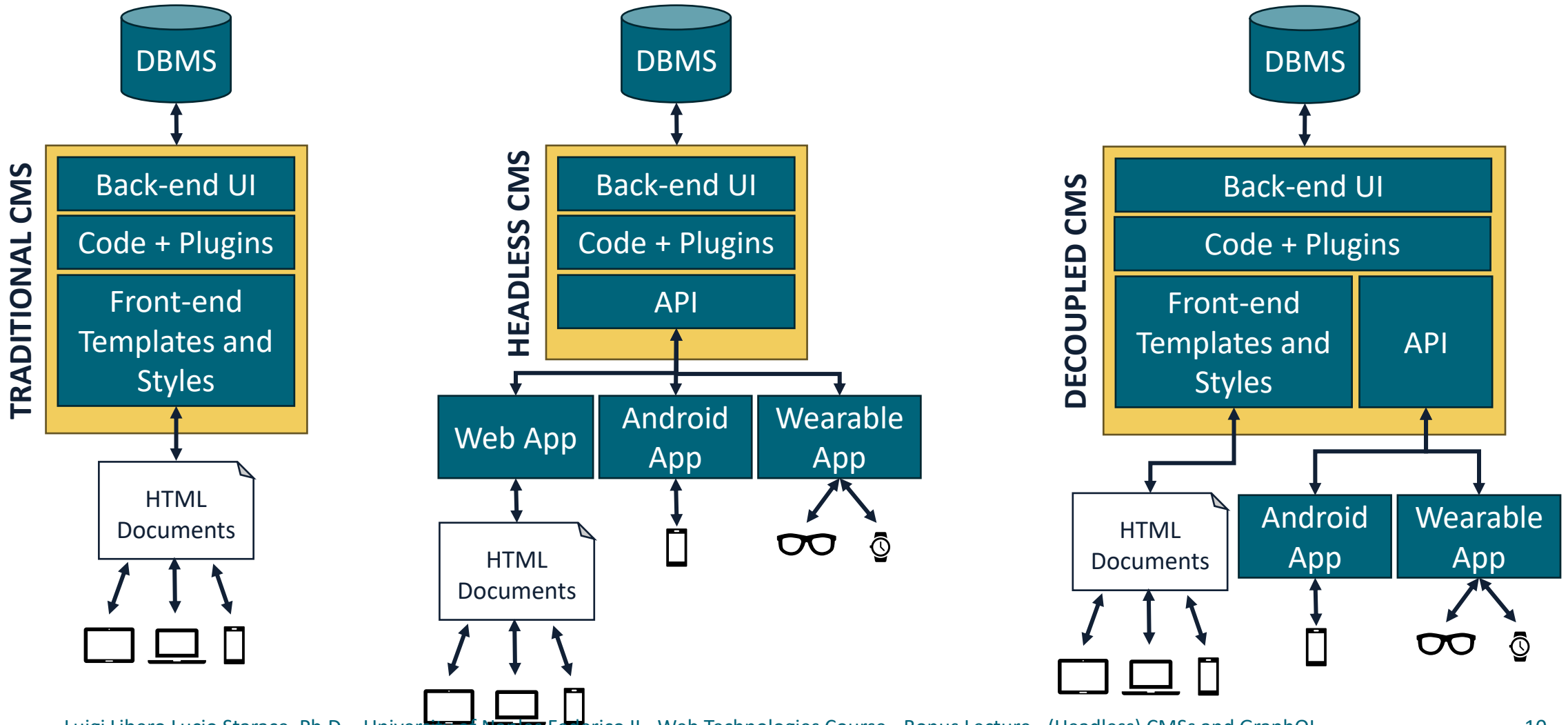
HEADLESS CMSs

- **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 appearance 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

Benvenuto

Benvenuto nella famosa installazione di WordPress in cinque minuti! Compila semplicemente le informazioni qua sotto e sarai già sulla strada per utilizzare la piattaforma di pubblicazione più estesa e potente del mondo.

Informazioni necessarie

Inserisci le seguenti informazioni. Non preoccuparti, potrai sempre cambiarle in seguito.

Titolo del sito

Web Technologies

Nome utente


luigi

I nomi utente possono essere composti soltanto da caratteri alfanumerici, spazi, trattini bassi, trattini, punti e il simbolo @.

Password

luigi

Very weak

 Nascondi

Importante: Avrai bisogno di questa password per accedere. Conservala in un posto sicuro.

Conferma password

☐ Conferma l'uso di una password debole

La tua email

admin@admin.com

Controlla attentamente il tuo indirizzo email prima di continuare.

Visibilità ai motori di ricerca

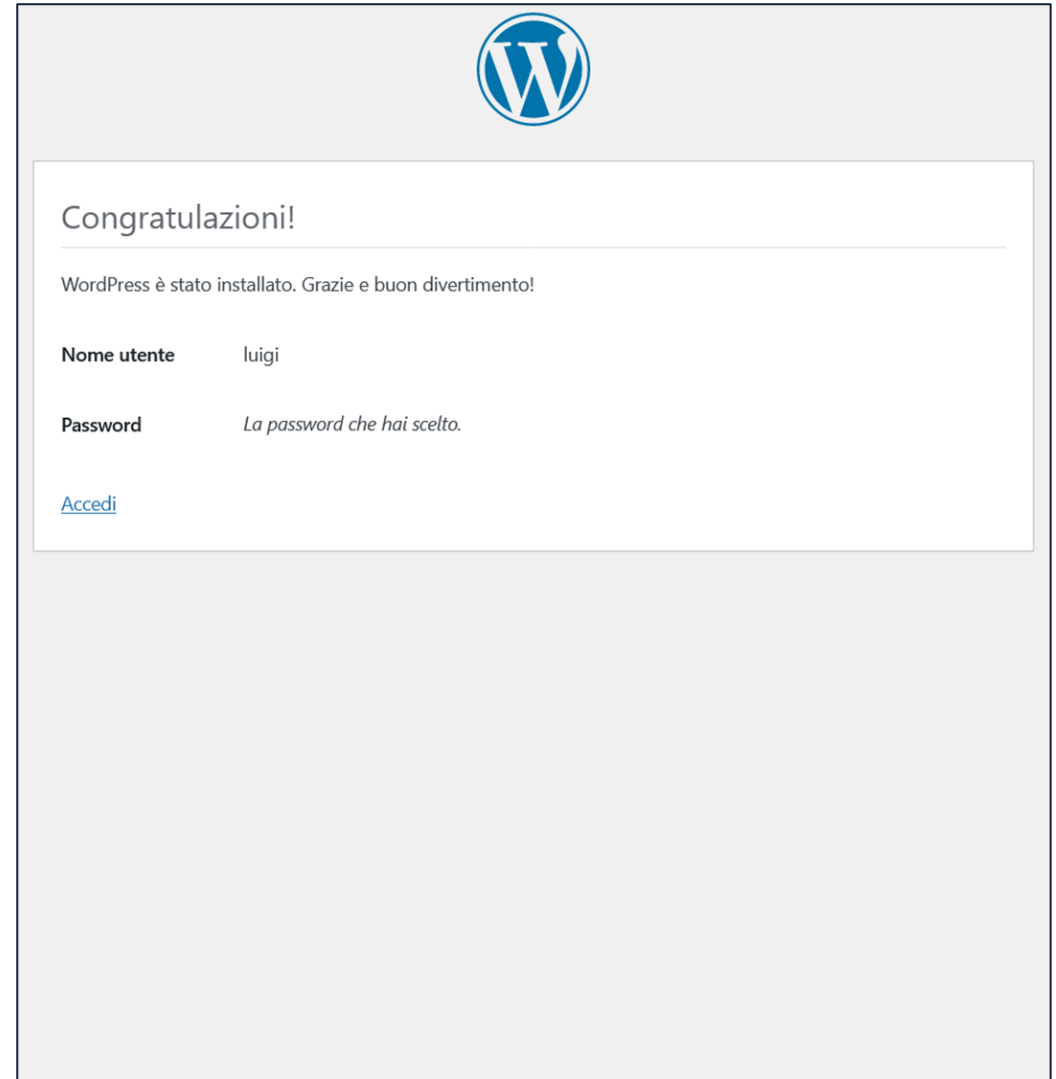
☐ Scoraggia i motori di ricerca dall'effettuare l'indicizzazione di questo sito

È compito dei motori di ricerca onorare o meno questa richiesta.

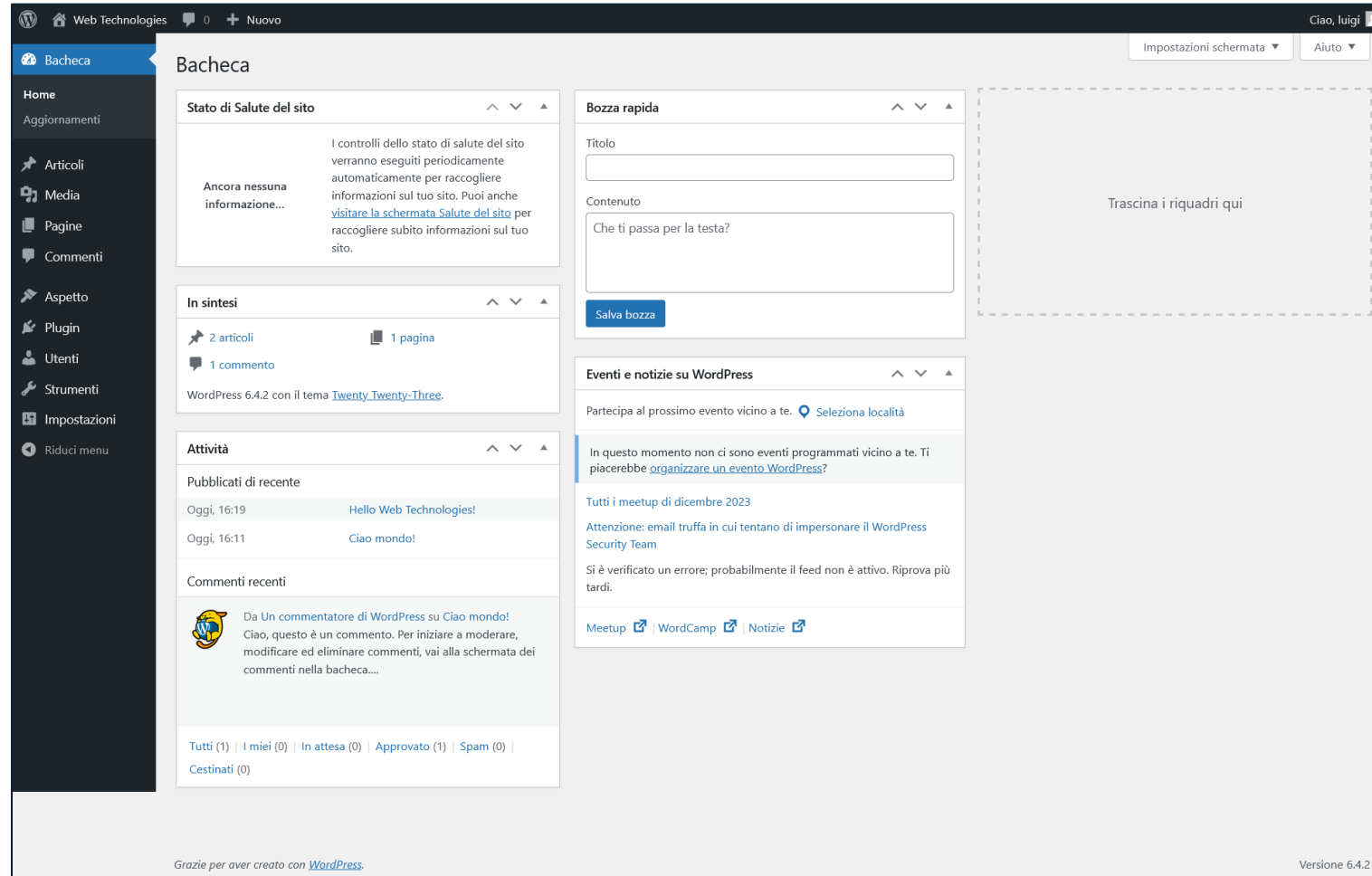
Installa WordPress

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



Welcome to Strapi!

Credentials are only used to authenticate in Strapi. All saved data will be stored in your database.

First name*

luigi

Last name

Email*

admin@admin.com

Password*

•••••

Must be at least 8 characters, 1 uppercase, 1 lowercase & 1 number

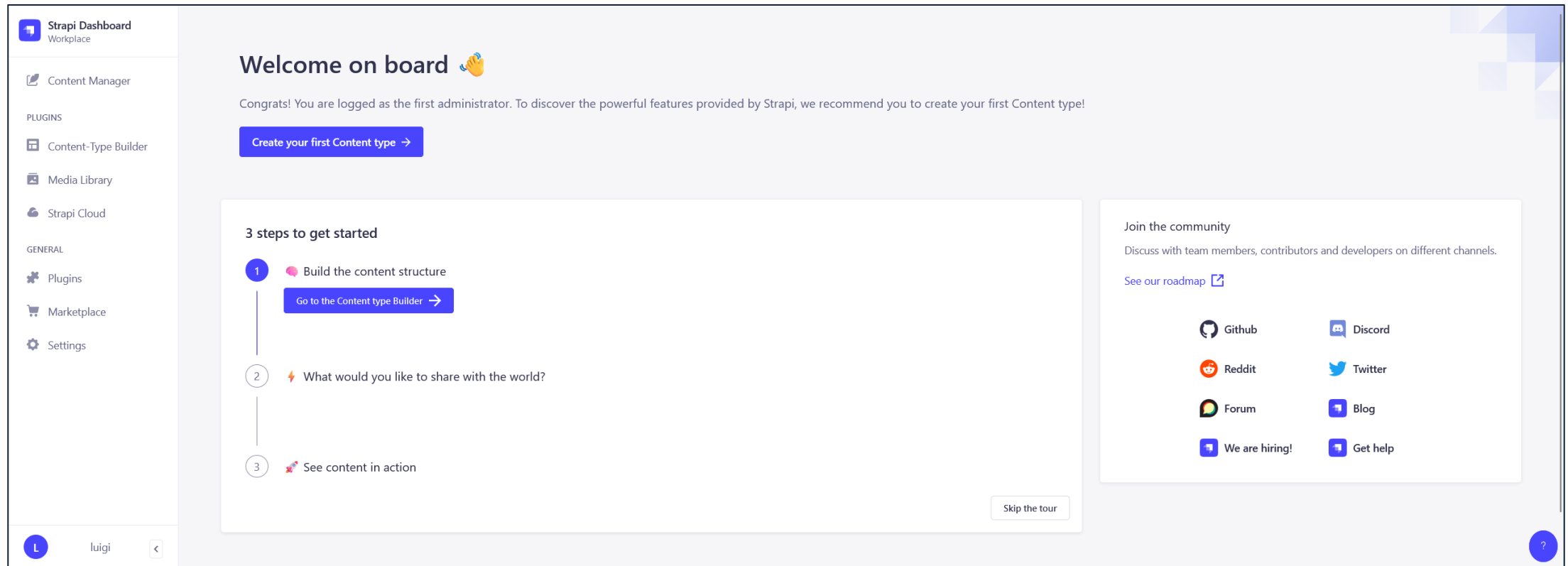
Confirm Password*

•••••

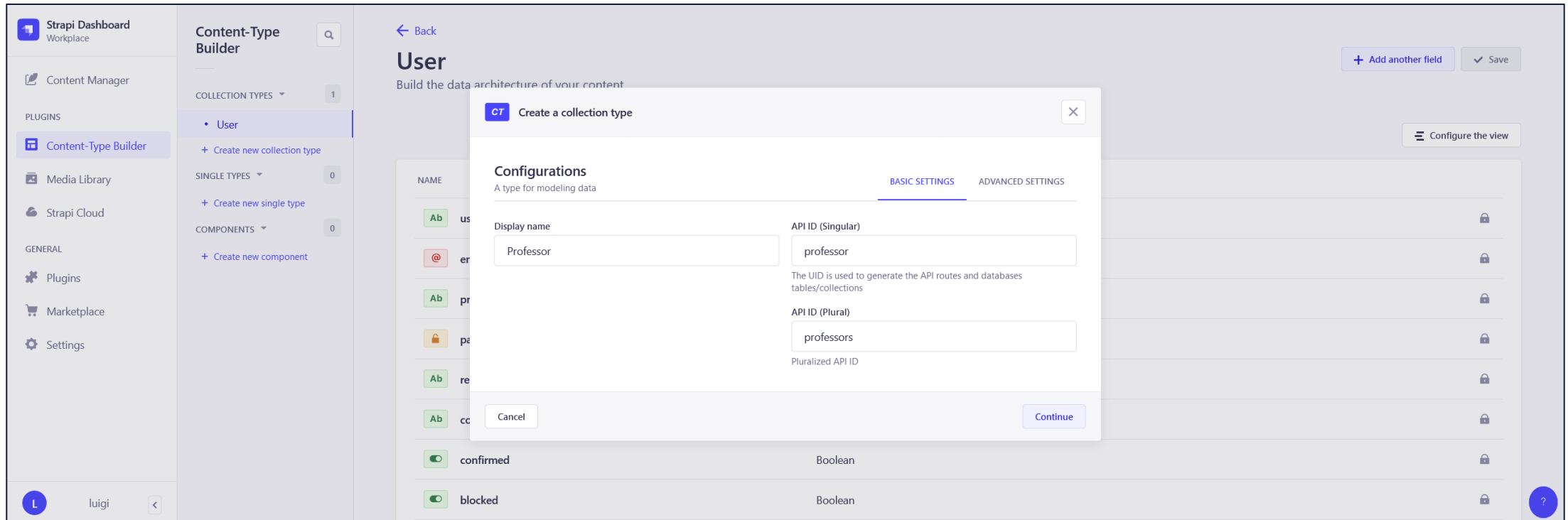
☐ Keep me updated about new features & upcoming improvements (by doing this you accept the [terms](#) and the [privacy policy](#)).

Let's start

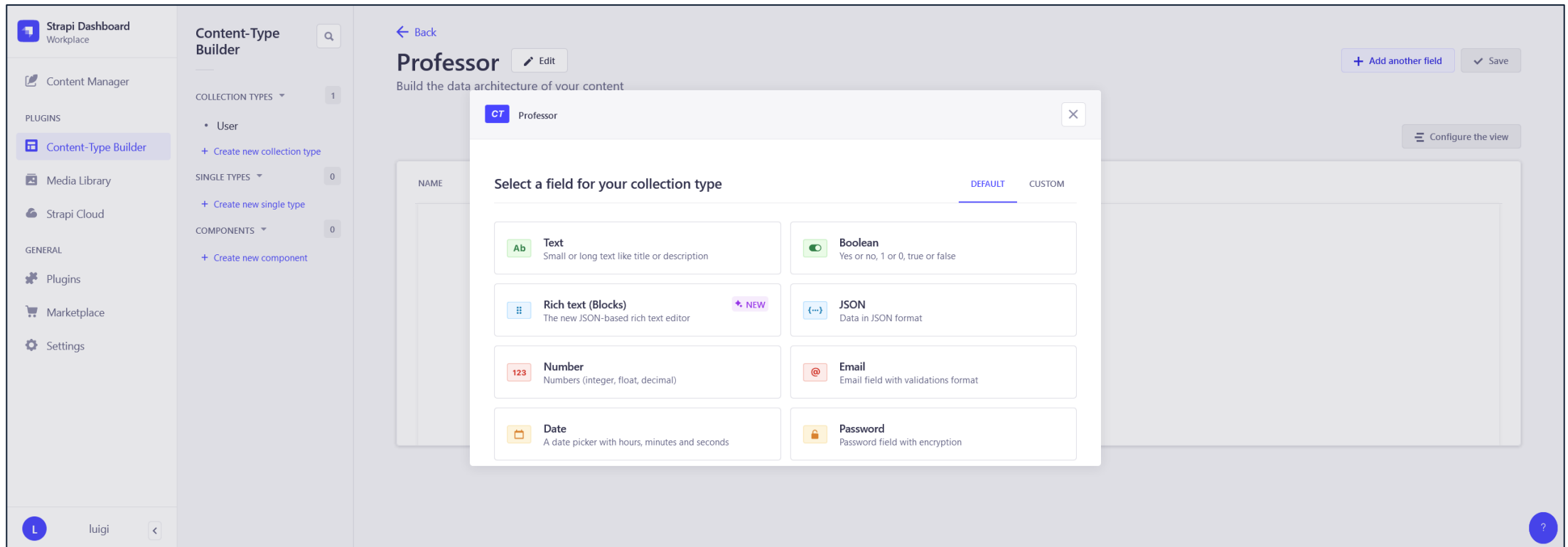
STRAPI: ADMIN DASHBOARD



STRAPI: CREATING RESOURCES



STRAPI: DEFINING RESOURCES



STRAPI: DEFINING RESOURCES

[←](#) **Ab** Professor [×](#)

Add new Text field

Small or long text like title or description

[BASIC SETTINGS](#) [ADVANCED SETTINGS](#)

Name

No space is allowed for the name of the attribute

Type

☒ **Short text**
Best for titles, names, links (URL). It also enables exact search on the field.

☐ **Long text**
Best for descriptions, biography. Exact search is disabled.

[←](#) **Ab** Professor [×](#)

Add new Text field

Small or long text like title or description

[BASIC SETTINGS](#) [ADVANCED SETTINGS](#)

Default value

RegExp pattern

The text of the regular expression

Settings

☒ **Required field**
You won't be able to create an entry if this field is empty

☐ **Unique field**
You won't be able to create an entry if there is an existing entry with identical content

☐ **Maximum length**

☐ **Private field**
This field will not show up in the API response

☐ **Minimum length**

STRAPI: DEFINING RESOURCES

The screenshot displays the Strapi Content-Type Builder interface. On the left, a sidebar contains navigation links: Strapi Dashboard Workplace, Content Manager, Plugins (with Content-Type Builder selected), Media Library, Strapi Cloud, and a General section with Plugins, Marketplace, and Settings. The main area is titled 'Content-Type Builder' and shows a search bar, a 'COLLECTION TYPES' dropdown with 1 item ('User'), a 'SINGLE TYPES' dropdown with 0 items, and a 'COMPONENTS' dropdown with 0 items. The 'Professor' collection type is selected, with an 'Edit' button. Below the title, it says 'Build the data architecture of your content'. A table lists the fields for the 'Professor' collection type:

NAME	TYPE	
firstName	Text	
lastName	Text	
email	Email	

At the bottom of the table, there is a button '+ Add another field to this collection type'. In the top right corner, there are buttons '+ Add another field' and 'Save'. In the bottom right corner, there is a 'Configure the view' button. The bottom of the interface shows the user 'luigi' and a help icon.

STRAPI: DEFINING RESOURCES

← 123 Course

Add new Number field

Numbers (integer, float, decimal)

BASIC SETTINGS ADVANCED SETTINGS

Name

credits

No space is allowed for the name of the attribute

Number format

integer (ex: 10)

Choose here

- integer (ex: 10)**
- big integer (ex: 123456789)
- decimal (ex: 2.22)
- float (ex: 3.33333333)

Cancel

STRAPI: DEFINING RESOURCES

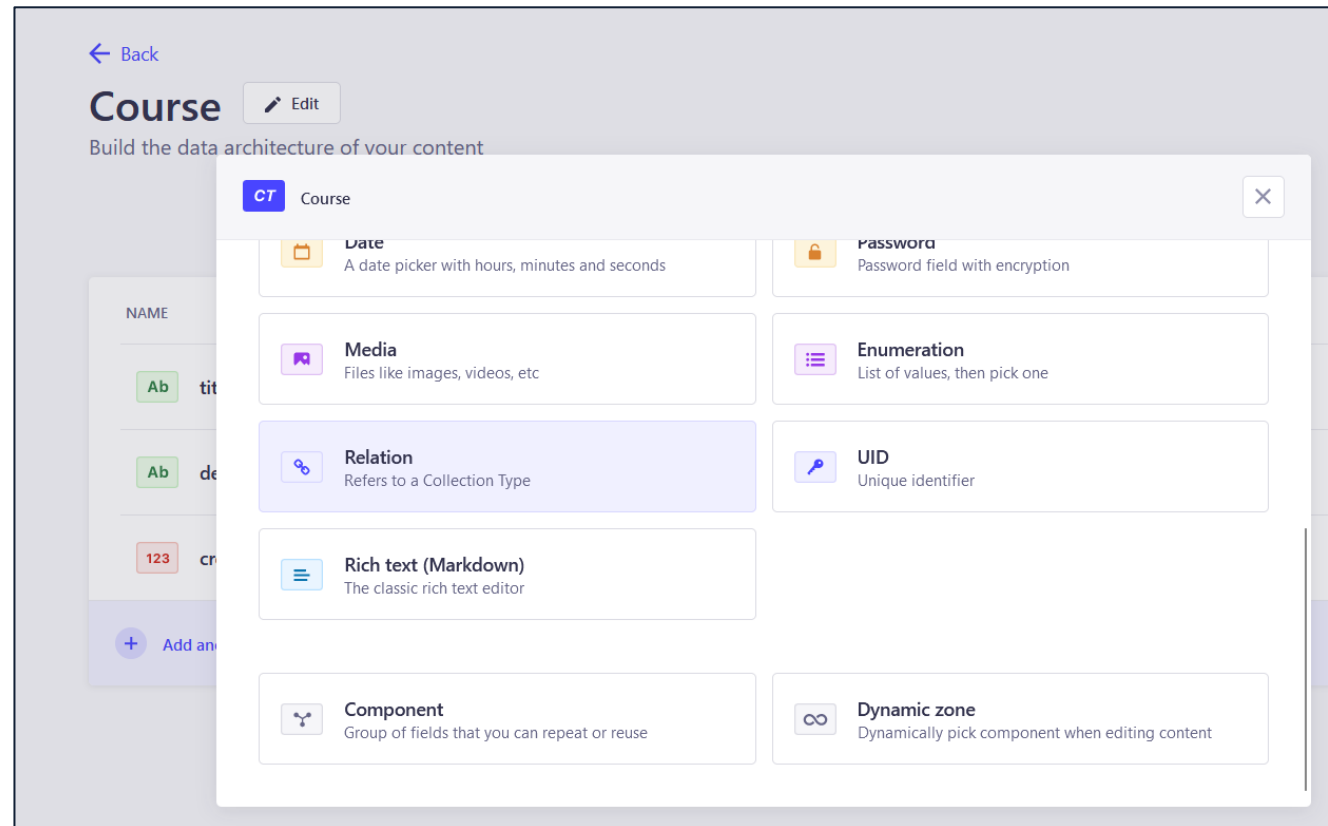
The screenshot displays the Strapi Content-Type Builder interface. On the left, a sidebar contains navigation links: Strapi Dashboard Workplace, Content Manager, Plugins (with Content-Type Builder selected), Media Library, Strapi Cloud, and a General section with Plugins, Marketplace, and Settings. The main area is titled 'Content-Type Builder' and shows a search bar, a list of collection types (Professor, User) with a count of 2, and single types and components counts of 0. The 'Course' collection type is selected, showing a 'Back' link, an 'Edit' button, and a description: 'Build the data architecture of your content'. A table lists the fields for the 'Course' collection type:

NAME	TYPE	
Ab title	Text	
Ab description	Text	
123 credits	Number	

Below the table is a button to 'Add another field to this collection type'. At the top right of the main area are buttons for '+ Add another field' and 'Save'. A 'Configure the view' button is also present. The bottom of the interface shows a user profile 'luigi' and a help icon.

STRAPI: RELATIONS BETWEEN RESOURCES

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



STRAPI: RELATIONS BETWEEN RESOURCES

The screenshot displays the Strapi admin interface for managing a 'Course' resource. A modal window titled 'Add new Relation field' is open, showing the configuration for a new relation between 'Course' and 'Professor'.

Course Resource Configuration:

- Field name: professors

Professor Resource Configuration:

- Field name: courses

Relation Type: The diagram shows a many-to-many relationship between 'Course' and 'Professor'. The text below the diagram states: 'Courses has and belongs to many Professors'.

Buttons:

- Cancel
- + Add another field
- Finish

Background Interface:

- Back button
- Edit button
- + Add another field
- Save button
- Configure the view button

STRAPI: RELATIONS BETWEEN RESOURCES

[< Back](#)

Professor

Edit

+ Add another field

✓ Save

Build the data architecture of your content

☰ Configure the view

NAME	TYPE
<div>Ab</div> firstName	Text
<div>Ab</div> lastName	Text
<div>@</div> email	Email
<div>🔗</div> courses	Relation with <i>Course</i>

+ Add another field to this collection type

?

[< Back](#)

Course

Edit

+ Add another field

✓ Save

Build the data architecture of your content

☰ Configure the view

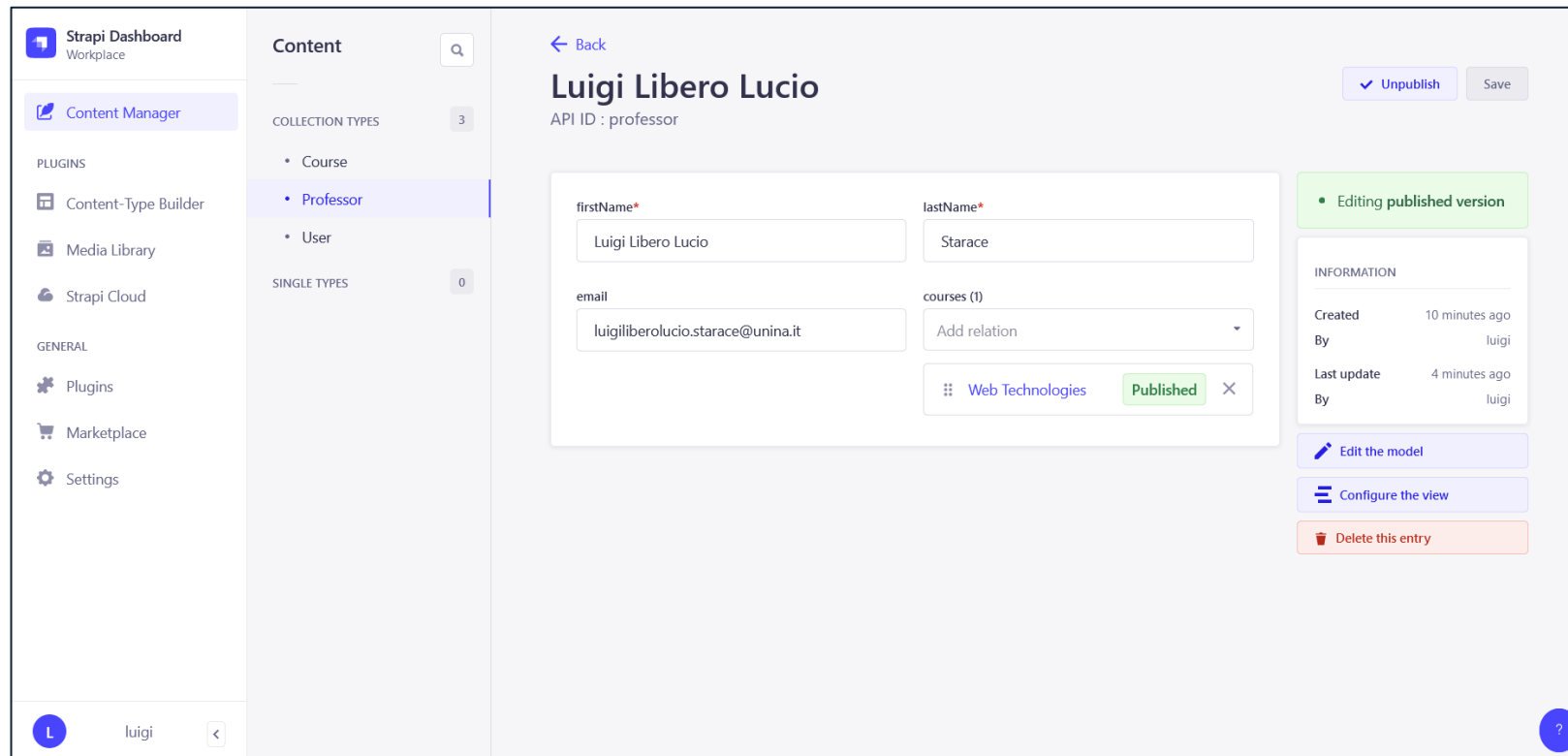
NAME	TYPE
<div>Ab</div> title	Text
<div>Ab</div> description	Text
<div>123</div> credits	Number
<div>🔗</div> professors	Relation with <i>Professor</i>

+ Add another field to this collection type

?

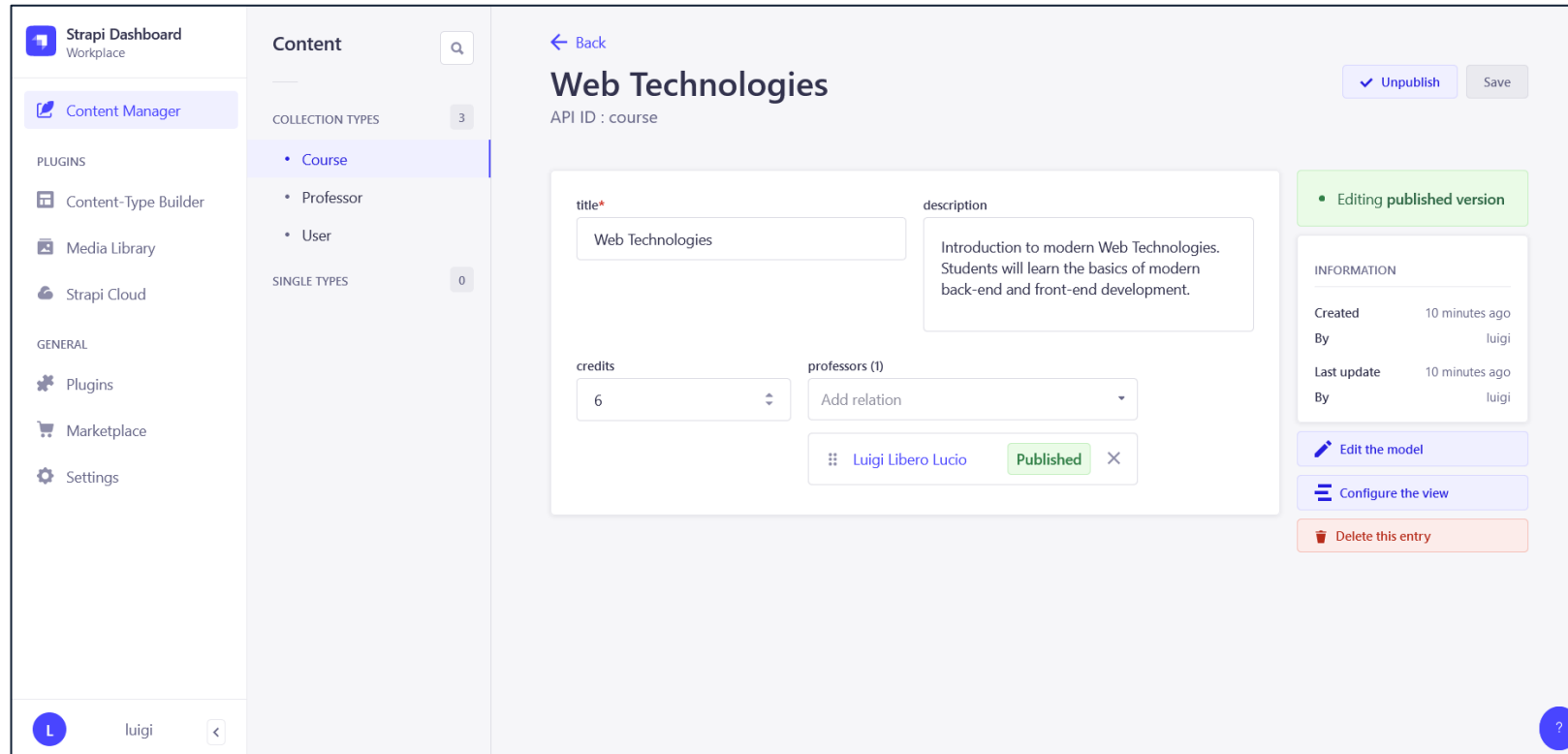
STRAPI: CREATING CONTENT

- 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

The screenshot displays the Strapi Dashboard interface. On the left, a sidebar contains navigation links: 'Strapi Dashboard Workplace', 'Content Manager' (highlighted), 'PLUGINS' (with sub-links for 'Content-Type Builder', 'Media Library', and 'Strapi Cloud'), and 'GENERAL' (with sub-links for 'Plugins', 'Marketplace', and 'Settings'). The main area is titled 'Content' and shows 'COLLECTION TYPES' with 'Course' selected (3 entries) and 'SINGLE TYPES' with 'User' (0 entries). The 'Course' collection type is expanded, showing a list of 2 entries. The list has columns for ID, TITLE, DESCRIPTION, CREDITS, and STATE. Both entries are marked as 'Published' and have edit, duplicate, and delete icons. The first entry has ID 2 and the second has ID 1. At the bottom of the list, there is a pagination control showing '10' entries per page and a page indicator '1'.

Strapi Dashboard Workplace

Content Manager

PLUGINS

- Content-Type Builder
- Media Library
- Strapi Cloud

GENERAL

- Plugins
- Marketplace
- Settings

Content

COLLECTION TYPES 3

- Course**
- Professor
- User

SINGLE TYPES 0

Course

2 entries found

Search Filters

ID	TITLE	DESCRIPTION	CREDITS	STATE
2	Structure and Interpretation of Computer Prog...	This course goes over the structure and interpr...	9	Published
1	Web Technologies	Introduction to modern Web Technologies. Stu...	6	Published

10 Entries per page

1

STRAPI: MANAGING CONTENT

The screenshot displays the Strapi Dashboard interface. On the left, a sidebar contains navigation links: 'Strapi Dashboard Workplace', 'Content Manager' (highlighted), 'Plugins', 'Content-Type Builder', 'Media Library', 'Strapi Cloud', 'GENERAL', 'Plugins', 'Marketplace', and 'Settings'. The main area is titled 'Content' and shows 'COLLECTION TYPES' with 'Professor' selected (3 entries) and 'SINGLE TYPES' with 'User' (0 entries). The 'Professor' collection type is expanded, showing a list of 3 entries. Each entry has a checkbox, ID, FIRSTNAME, LASTNAME, EMAIL, and STATE. The STATE column shows 'Published' for all entries. At the bottom, there is a '10 Entries per page' dropdown and a pagination control showing '1'.

Strapi Dashboard Workplace

Content Manager

PLUGINS

- Content-Type Builder
- Media Library
- Strapi Cloud

GENERAL

- Plugins
- Marketplace
- Settings

Content

COLLECTION TYPES 3

- Course
- Professor**
- User

SINGLE TYPES 0

Professor

3 entries found

Search Filters

<input type="checkbox"/>	ID	FIRSTNAME ^	LASTNAME	EMAIL	STATE	
<input type="checkbox"/>	3	Gerald	Sussman	-	Published	
<input type="checkbox"/>	2	Harold	Abelson	-	Published	
<input type="checkbox"/>	1	Luigi Libero Lucio	Starace	luigiliberolucio.starace@unina.it	Published	

10 Entries per page

< 1 >

luigi

?

STRAPI: USING THE REST API

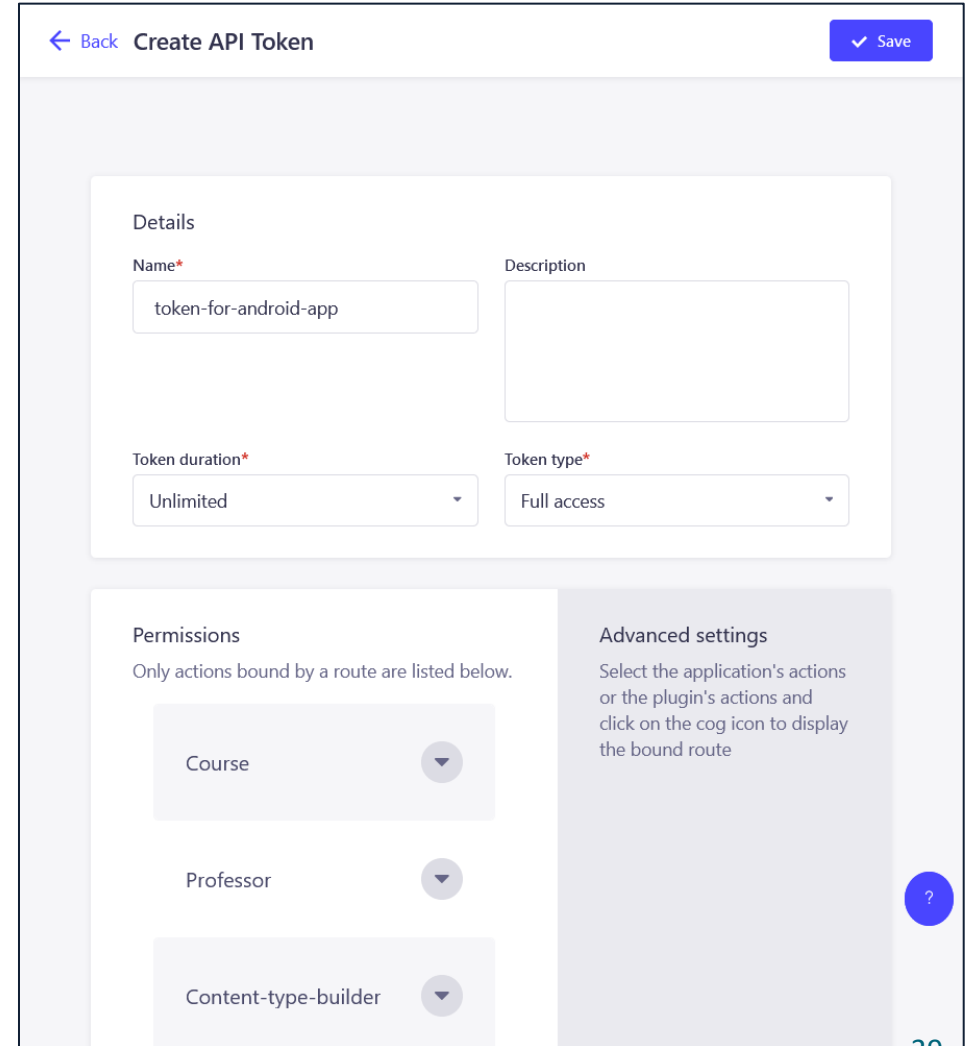
- For each resource, Strapi automatically generates REST endpoints

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)



The screenshot shows the 'Create API Token' form in the Strapi UI. At the top, there is a navigation bar with a 'Back' link and a 'Save' button. The form is divided into three main sections: 'Details', 'Permissions', and 'Advanced settings'. The 'Details' section contains fields for 'Name' (with the value 'token-for-android-app'), 'Description', 'Token duration' (set to 'Unlimited'), and 'Token type' (set to 'Full access'). The 'Permissions' section lists resources with dropdown menus: 'Course', 'Professor', and 'Content-type-builder'. The 'Advanced settings' section provides instructions on how to select application or plugin actions and how to bind them to a route. A help icon is visible in the bottom right corner of the form area.

← Back Create API Token Save

Details

Name* token-for-android-app

Description

Token duration* Unlimited

Token type* Full access

Permissions

Only actions bound by a route are listed below.

Course

Professor

Content-type-builder

Advanced settings

Select the application's actions or the plugin's actions and click on the cog icon to display the bound route

?

STRAPI: AUTHORIZING USERS

The screenshot displays the Strapi Dashboard interface. On the left, a sidebar contains navigation links: 'Strapi Dashboard Workplace', 'Content Manager', 'PLUGINS' (Content-Type Builder, Media Library, Strapi Cloud), 'GENERAL' (Plugins, Marketplace), and 'Settings' (highlighted). The main content area is titled 'Settings' and includes a 'Back' link. The 'API Tokens' section is active, showing a list of tokens. The first token, 'token-for-android-app', is expanded, revealing its details: a long alphanumeric token string, a key icon, and a warning to copy the token. Below this, the 'Details' section includes fields for 'Name*' (filled with 'token-for-android-app'), 'Description', 'Token duration*' (set to 'Unlimited'), and 'Token type*' (set to 'Full access'). The 'Permissions' section at the bottom indicates that only actions bound by a route are listed. The user 'luigi' is logged in, as shown in the bottom left corner.

Strapi Dashboard
Workplace

Content Manager

PLUGINS

- Content-Type Builder
- Media Library
- Strapi Cloud

GENERAL

- Plugins
- Marketplace
- Settings**

Settings

GLOBAL SETTINGS

- Overview
- API Tokens**
- Internationalization
- Media Library
- Review Workflows
- Single Sign-On
- Transfer Tokens
- Webhooks

ADMINISTRATION PANEL

- Roles
- Users
- Audit Logs

EMAIL PLUGIN

- Configuration

USERS & PERMISSIONS PLUGIN

- Roles

token-for-android-app Regenerate Save

97060c91103a1a264d3d4a18f60ae25c6adad456a53b2afa18ae26bbd80686b61fa2e3d99e0cf667463053bca6488c2dba0b92057a15aa9307c29682da67958f669886dbffdbfa4a3f71f9bb6b2c94e0530e5e2a5dd24b44c300f0d3b44806f64ec6d0a3ad3f9e3382b980c212972d3de34e4176e7ebaec8eb615ecda9fe12

Make sure to copy this token, you won't be able to see it again!

Details

Name*

Description

Token duration*

Token type*

Expiration date: Unlimited

Permissions
Only actions bound by a route are listed

Advanced settings
Select the application's

luigi

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:

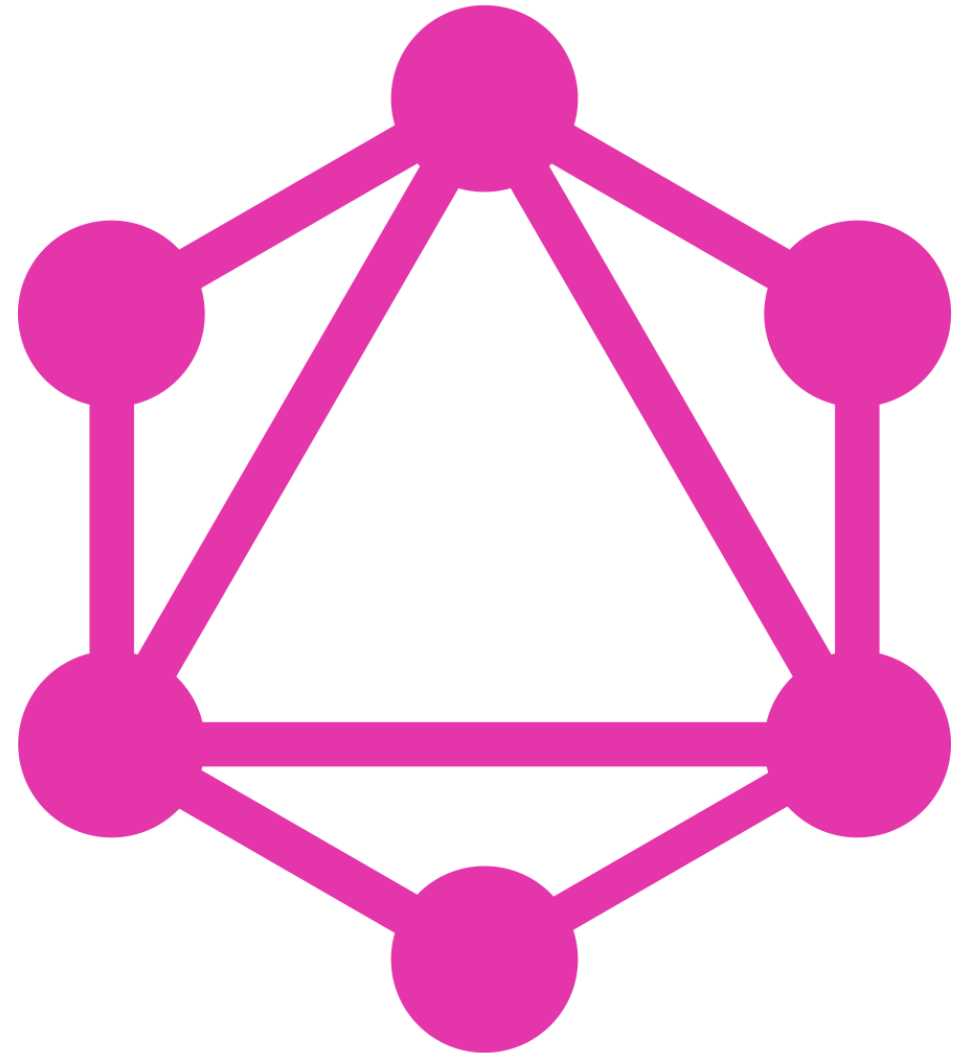
- **Over-fetching**
 - Sometimes we do not need all the fields of a resource
- **Under-fetching**
 - Sometimes we need linked resources, and it might be necessary to perform additional requests to fetch them
- **API changes and evolution**
 - APIs evolve over time. Thus, they need to be versioned, maintain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

REST: OVER– AND UNDER–FETCHING

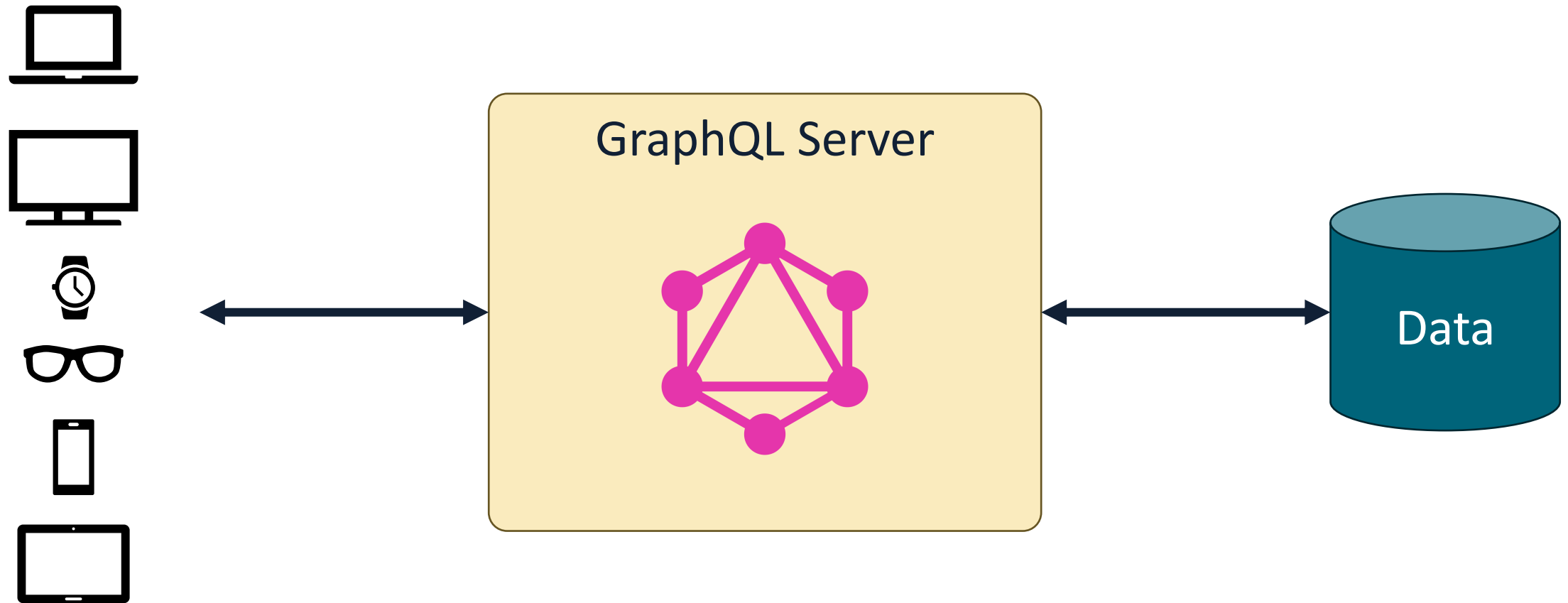
- Over-fetching and Under-fetching can both deteriorate performance and increase data transfers
- In an over-fetching scenario, we download data we do not need
- In an under-fetching scenario, we will need to perform (many) additional requests to get all the data we need
 - E.g.: first we get the current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

GRAPHQL: A QUERY LANGUAGE FOR APIs

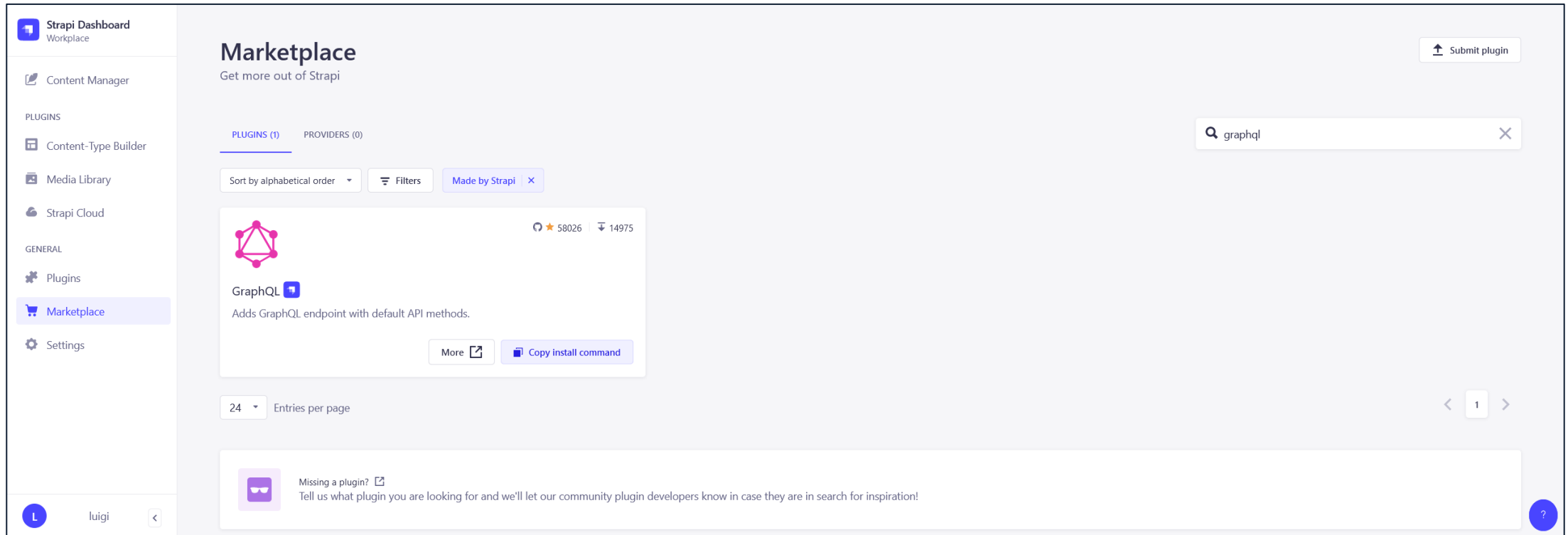
- Developed by Facebook
- Open source version published in 2015
- Specification available at <http://spec.graphql.org/>
- Designed to address the need for more **flexibility** and **efficiency**
- Instead of relying on pre-defined endpoints as in REST, allow users to **declaratively specify** which data 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 **CO₂**!) 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!)

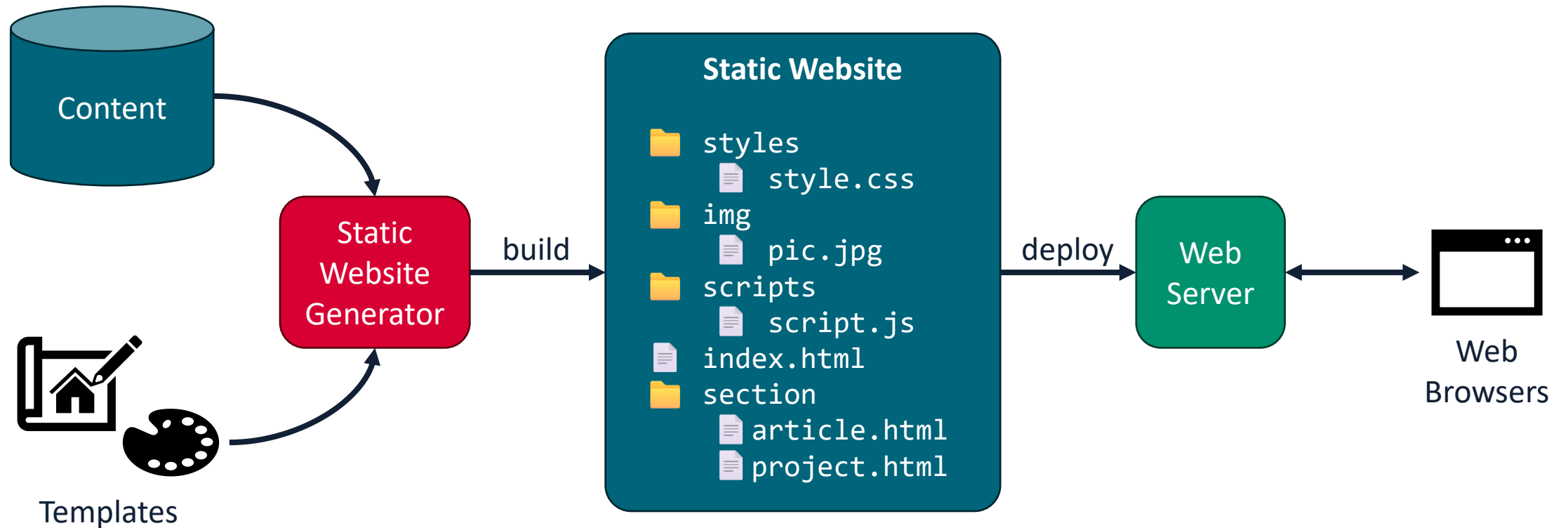
This approach is referred to as **Automatic Static Website Generation**

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](#)