Flask REST API

1. Zainstaluj odpowiednie moduły z poziomu File->Settings:

flask

connexion

flask-sqlalchemy

flask-marshmallow

flask-Bootstrap

marshmallow-sqlalchemy

marshmallow

swagger-ui-bundle

1. W folderze Day13 załóż nowy folder o nazwie server, utwórz w nim plik o nazwie app.py
2. Uzupełnij kod pliku app.py:

**import** os  
**import** connexion  
**from** flask\_sqlalchemy **import** SQLAlchemy  
**from** flask\_marshmallow **import** Marshmallow  
**from** flask\_bootstrap **import** Bootstrap  
  
main\_directory = os.path.abspath(os.path.dirname(\_\_file\_\_))  
  
app\_connexion = connexion.App(\_\_name\_\_, specification\_dir=main\_directory)  
  
app = app\_connexion.app  
Bootstrap(app)  
  
*# database path*sqlite\_path = **"sqlite:///"** + os.path.join(main\_directory, **"music\_store.db"**)  
  
*# Configure the SqlAlchemy part of the app instance*app.config[**"SQLALCHEMY\_ECHO"**] = **True**app.config[**"SQLALCHEMY\_DATABASE\_URI"**] = sqlite\_path  
app.config[**"SQLALCHEMY\_TRACK\_MODIFICATIONS"**] = **False***# sql alchemy orm to communicate with sqlite db*db = SQLAlchemy(app)  
  
*# marshmallow init*ma = Marshmallow(app)

Plik app.py będzie służył jako plik z główną konfiguracją aplikacji, jak aplikacja serwera – flask/connection oraz dostep do bazy danych

1. Utwórz plik index.py w folderze server – plik ten będzie służyl jako glówny skrypt uruchamiający naszą aplikację. Poniżej kod, który należy umieścić w pliku:

**from** flask **import** render\_template  
**from** Day13.server.app **import** app\_connexion  
**from** Day13.server.models **import** Album, AlbumSchema  
  
*# Get the application instance*connex\_app = app\_connexion  
  
*# create a URL route in our application for "/"*@connex\_app.route(**"/"**)  
**def** home():  
 *"""  
 This function just responds to the browser URL  
 localhost:8080/* **:return***: the rendered template  
 """* **return "<h1>Hello, iSA Python World!</h1>"  
  
  
if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 connex\_app.run(port=8080, debug=**True**)

1. Uruchom plik index.py, otwórz przeglądarkę i otwórz adres <http://localhost:8080/>
2. Zmień kod w pliku index.py, aby w przeglądarce wyświetlił się napis: „That is REST API example” – zapisz plik index.py po zmianach, przejrzyj logi konsoli i odśwież stronę przeglądarki – czy tekst się zmienił?
3. Zatrzymaj uruchomiony skrypt. W folderze server dodaj plik swagger.yml, zawartość pliku:

**swagger**: **"2.0"  
info**:  
 **description**: This is the swagger file with our REST API spec  
 **version**: **"1.0.0"  
 title**: Swagger Rest Documentation  
**consumes**:  
 - application/json  
**produces**:  
 - application/json  
  
**basePath**: /api  
  
*# Paths supported by the server application***paths**:  
 **/album**:  
 **get**:  
 **operationId**: album.read\_all  
 **tags**:  
 - Albums  
 **summary**: Gets all the albums, sorted by artist  
 **description**: Gets all the albums, sorted by artist  
 **responses**:  
 **200**:  
 **description**: Successfully returned albyms  
 **schema**:  
 **type**: array  
 **items**:  
 **properties**:  
 **album\_id**:  
 **type**: string  
 **description**: Id of the albym  
 **artist**:  
 **type**: string  
 **description**: Title of the album  
 **title**:  
 **type**: string  
 **description**: Title of the album  
 **image\_url**:  
 **type**: string  
 **description**: Image URL of the album  
 **last\_edited\_at**:  
 **type**: string  
 **description**: Creation/Update timestamp of the album

1. W folderze server stwórz plik album.py:

**import** json  
**from** flask **import** jsonify  
  
albums = [  
 {  
 **"album\_id"**: 1,  
 **"artist"**: **"Michael Jackson"**,  
 **"title"**: **"Thriller"**,  
 **"image\_url"**: **"https://upload.wikimedia.org/wikipedia/en/thumb/5/55/Michael\_Jackson\_-\_Thriller.png/220px-Michael\_Jackson\_-\_Thriller.png"** },  
 {  
 **"album\_id"**: 2,  
 **"artist"**: **"Metallica"**,  
 **"title"**: **"Kill'em all"**,  
 **"image\_url"**: **"https://upload.wikimedia.org/wikipedia/en/thumb/5/55/Michael\_Jackson\_-\_Thriller.png/220px-Michael\_Jackson\_-\_Thriller.png"** }  
]  
  
**def** read\_all():  
 *"""  
 This function responds to a request for /api/album  
 with the lists of available albums* **:return***: json string of list of albums  
 """  
 # Create the list of albums from our data* **return** jsonify(albums)

1. W kodzie pliku index.py za linią 6 dodaj konfiguracje servera z wczesniej stworzonego pliku YAML:

*# Read the swagger.yml file to configure the endpoints*connex\_app.add\_api(**"swagger.yml"**)

1. Uruchom plik index.py, a następnie przejdź w przeglądarce do adresu

<http://localhost:8080/api/ui>

Rozwiń dokumentację do sekcji Albums i wypróbuj wywołanie metody GET /album

1. Uzupełnij konfigurację pliku swagger.yml o pozostałe metody do sekcji Albums:

Plik powinien wyglądać następująco:

**swagger**: **"2.0"  
info**:  
 **description**: This is the swagger file with our REST API spec  
 **version**: **"1.0.0"  
 title**: Swagger Rest Documentation  
**consumes**:  
 - application/json  
**produces**:  
 - application/json  
  
**basePath**: /api  
  
*# Paths supported by the server application***paths**:  
 **/album**:  
 **get**:  
 **operationId**: album.read\_all  
 **tags**:  
 - Albums  
 **summary**: Gets all the albums, sorted by artist  
 **description**: Gets all the albums, sorted by artist  
 **responses**:  
 **200**:  
 **description**: Successfully returned albyms  
 **schema**:  
 **type**: array  
 **items**:  
 **properties**:  
 **album\_id**:  
 **type**: string  
 **description**: Id of the albym  
 **artist**:  
 **type**: string  
 **description**: Title of the album  
 **title**:  
 **type**: string  
 **description**: Title of the album  
 **image\_url**:  
 **type**: string  
 **description**: Image URL of the album  
 **last\_edited\_at**:  
 **type**: string  
 **description**: Creation/Update timestamp of the album  
  
 **post**:  
 **operationId**: album.create  
 **tags**:  
 - Albums  
 **summary**: Create an album  
 **description**: Create an new album  
 **parameters**:  
 - **name**: album  
 **in**: body  
 **description**: Album to create  
 **required**: True  
 **schema**:  
 **type**: object  
 **properties**:  
 **artist**:  
 **type**: string  
 **description**: Artist of album to create  
 **title**:  
 **type**: string  
 **description**: Title of album to create  
 **image\_url**:  
 **type**: string  
 **description**: Image URL of album to create  
 **responses**:  
 **201**:  
 **description**: Successfully created album  
 **schema**:  
 **properties**:  
 **album\_id**:  
 **type**: string  
 **description**: Id of the album  
 **artist**:  
 **type**: string  
 **description**: Artist of album to create  
 **title**:  
 **type**: string  
 **description**: Title of album to create  
 **image\_url**:  
 **type**: string  
 **description**: Image URL of album to create  
 **timestamp**:  
 **type**: string  
 **description**: Creation/Update timestamp of the album record  
  
 **/album/{album\_id}**:  
 **get**:  
 **operationId**: album.read\_one  
 **tags**:  
 - Albums  
 **summary**: Read one album data  
 **description**: Get one album  
 **parameters**:  
 - **name**: album\_id  
 **in**: path  
 **description**: Id of the album  
 **type**: integer  
 **required**: True  
 **responses**:  
 **200**:  
 **description**: Successfully read album  
 **schema**:  
 **type**: object  
 **properties**:  
 **album\_id**:  
 **type**: string  
 **description**: Id of the album  
 **artist**:  
 **type**: string  
 **description**: Artist of album  
 **title**:  
 **type**: string  
 **description**: Title of album  
 **image\_url**:  
 **type**: string  
 **description**: Image URL of album  
 **timestamp**:  
 **type**: string  
 **description**: Creation/Update timestamp of the album record  
  
 **put**:  
 **operationId**: album.update  
 **tags**:  
 - Albums  
 **summary**: Update an album  
 **description**: Update an album  
 **parameters**:  
 - **name**: album\_id  
 **in**: path  
 **description**: Id of the album to update  
 **type**: integer  
 **required**: True  
 - **name**: album  
 **in**: body  
 **schema**:  
 **type**: object  
 **properties**:  
 **artist**:  
 **type**: string  
 **description**: Artist of album  
 **title**:  
 **type**: string  
 **description**: Title of album  
 **image\_url**:  
 **type**: string  
 **description**: Image URL of album  
 **responses**:  
 **200**:  
 **description**: Successfully updated album  
 **schema**:  
 **properties**:  
 **album\_id**:  
 **type**: string  
 **description**: Id of the album  
 **artist**:  
 **type**: string  
 **description**: Artist of album  
 **title**:  
 **type**: string  
 **description**: Title of album  
 **image\_url**:  
 **type**: string  
 **description**: Image URL of album  
 **timestamp**:  
 **type**: string  
 **description**: Creation/Update timestamp of the album record  
  
 **delete**:  
 **operationId**: album.delete  
 **tags**:  
 - Albums  
 **summary**: Delete an album from the db  
 **description**: Delete an album  
 **parameters**:  
 - **name**: album\_id  
 **in**: path  
 **type**: integer  
 **description**: Id of the album to delete  
 **required**: true  
 **responses**:  
 **200**:  
 **description**: Successfully deleted an album!

1. Po uzupełnieniu sekcji konfiguracji, należy dodać właściwy kod do obsługi albumów – najpierw jednak skorzystamy z biblioteki SQLAlchemy i stworzymy model naszego albumu, przystosowany do przechowywania w bazie danych. W folderze server stwórz plik models.py:

**from** Day13.server.app **import** db, ma  
**import** datetime  
  
**class** Album(db.Model):  
 \_\_tablename\_\_ = **"album"** album\_id = db.Column(db.Integer, primary\_key=**True**)  
 artist = db.Column(db.String(64))  
 title = db.Column(db.String(128))  
 image\_url = db.Column(db.String(255))  
 last\_edited\_at = db.Column(  
 db.DateTime, default=datetime.datetime.utcnow, onupdate=datetime.datetime.utcnow  
 )  
  
**class** AlbumSchema(ma.Schema):  
 **class** Meta:  
 *# model = Album,* fields = (**"album\_id"**, **"artist"**, **"title"**, **"image\_url"**, **"last\_edited\_at"**)  
 sqla\_session = db.session

1. Następnie zmień kod pliku album.py, tak aby korzystać tym razem z danych dostępnych w bazie danych:

**from** Day13.server.app **import** db  
**from** Day13.server.models **import** Album, AlbumSchema  
**from** flask **import** make\_response, abort  
  
**def** read\_all():  
 *"""  
 This function responds to a request for /api/album  
 with the lists of available albums* **:return***: json string of list of albums  
 """  
 # Create the list of people from our data* albums = Album.query.order\_by(Album.artist).all()  
  
 *# Serialize the data for the response* album\_schema = AlbumSchema(many=**True**)  
 dump = album\_schema.dump(albums)  
 data = dump.data  
 **return** data  
  
  
**def** read\_one(album\_id):  
 *"""  
 This function responds to a request for /api/album/{album\_id}  
 with one matching album from albums* **:param** *album\_id: Id of album to find* **:return***: album with passed id (if founded)  
 """  
 # Get the album from db* album = Album.query.filter(Album.album\_id == album\_id).one\_or\_none()  
  
 **if** album **is not None**:  
  
 *# Serialize the data for the response* album\_schema = AlbumSchema()  
 data = album\_schema.dump(album).data  
 **return** data  
  
 *# Otherwise, nope, didn't find that album* **else**:  
 abort(  
 404,  
 **"Album not found for Id: {album\_id}"**.format(album\_id=album\_id),  
 )  
  
  
**def** create(album):  
 *"""  
 This function creates a new album  
 based on the passed data* **:param** *album: album to create in people structure* **:return***: 201 on success, 406 on album exists  
 """* artist = album.get(**"artist"**)  
 title = album.get(**"title"**)  
 image\_url = album.get(**"image\_url"**)  
  
 existing\_album = (  
 Album.query.filter(Album.artist == artist)  
 .filter(Album.title == title)  
 .one\_or\_none()  
 )  
  
 **if** existing\_album **is None**:  
  
 *# Create a album instance using the schema* schema = AlbumSchema()  
 new\_album = Album(artist=album[**"artist"**], title=album[**"title"**], image\_url=album[**"image\_url"**])  
  
 *# Add the album to the database* db.session.add(new\_album)  
 db.session.commit()  
  
 *# Serialize and return the newly created album in the response* data = schema.dump(new\_album).data  
  
 **return** data, 201  
  
 *# Album exists already* **else**:  
 abort(  
 409,  
 **f"Album {**artist**}: {**title**} exists already"** )  
  
  
**def** update(album\_id, album):  
 *"""  
 This function updates an existing album  
 Throws an error if an album  
 already exists in the database.* **:param** *album\_id: Id of the album to update* **:param** *album: new data of album* **:return***: updated album data  
 """* album\_to\_update = Album.query.filter(  
 Album.album\_id == album\_id  
 ).one\_or\_none()  
  
 *# Check if we're trying to create duplicates in db* artist = album.get(**"artist"**)  
 title = album.get(**"title"**)  
 image\_url = album.get(**"image\_url"**)  
  
 existing\_album = (  
 Album.query.filter(Album.artist == artist)  
 .filter(Album.title == title)  
 .one\_or\_none()  
 )  
  
 **if** album\_to\_update **is None**:  
 abort(  
 404,  
 **"Album not found for Id: {album\_id}"**.format(album\_id=album\_id),  
 )  
  
 *# Would our update create a duplicate?* **elif** (  
 existing\_album **is not None and** existing\_album.album\_id != album\_id  
 ):  
 abort(  
 409,  
 **f"Album {**artist**}: {**title**} exists already"** )  
  
 *# Otherwise go ahead and update!* **else**:  
  
 *# turn the passed data into a db object* schema = AlbumSchema()  
 update\_album\_data = Album(artist=artist, title=title, image\_url=image\_url)  
  
 *# Set the id to the album we want to update* update\_album\_data.album\_id = album\_to\_update.album\_id  
  
 *# merge the new object into the old and commit it to the db* db.session.merge(update\_album\_data)  
 db.session.commit()  
  
 *# return updated album* data = schema.dump(update\_album\_data).data  
  
 **return** data, 200  
  
  
**def** delete(album\_id):  
 *"""  
 This function deletes an album* **:param** *album\_id: Id of the album to delete* **:return***: 200 on successful delete, 404 if not found  
 """  
 # Get the album from db* album = Album.query.filter(Album.album\_id == album\_id).one\_or\_none()  
  
 **if** album **is not None**:  
 db.session.delete(album)  
 db.session.commit()  
 **return** make\_response(  
 **f"Album with id {**album\_id**} deleted"**, 200  
 )  
  
 *# Didn't find that album* **else**:  
 abort(  
 404,  
 **f"Album with id {**album\_id**} not found"** )

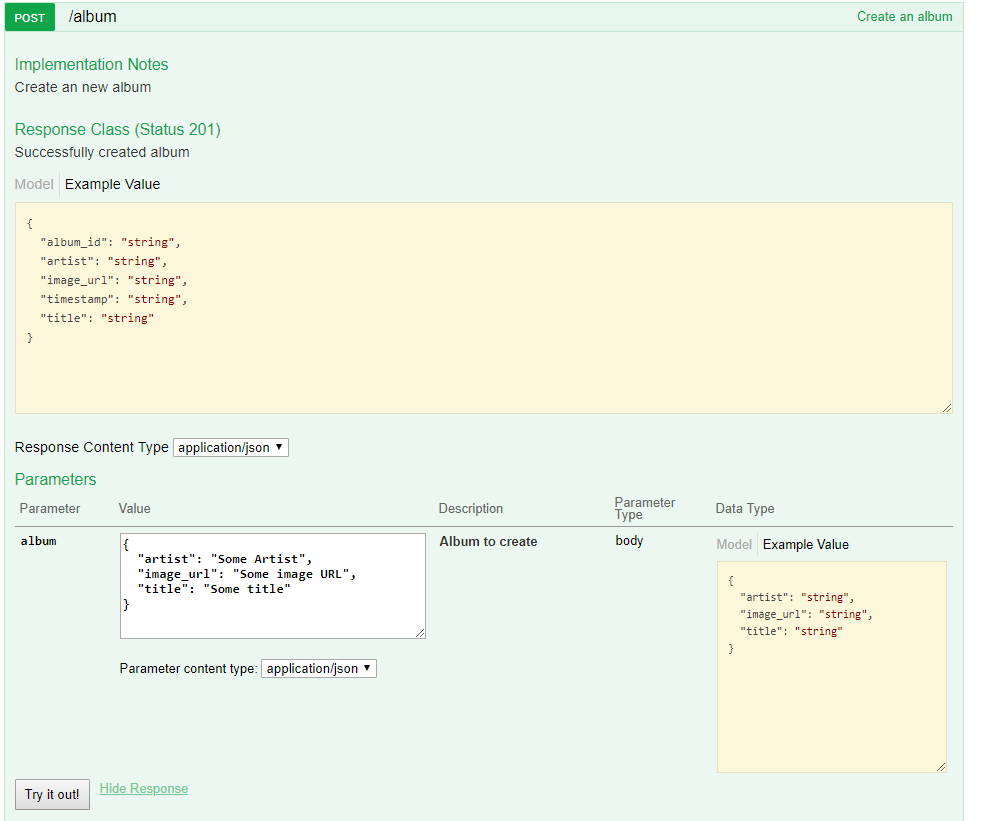
1. Aby aplikacja działała prawidłowo, należy jeszcze zainijować naszą bazę danych SQLite – stwórz plik init\_database.py:/uruchom plik

**import** os  
**from** Day13.server.app **import** db  
**from** Day13.server.models **import** Album  
  
*# Data to initialize database with*Albums = [  
 {**"artist"**: **"Michael Jackson"**, **"title"**: **"Thriller"**, **"image\_url"**: **"https://upload.wikimedia.org/wikipedia/en/thumb/5/55/Michael\_Jackson\_-\_Thriller.png/220px-Michael\_Jackson\_-\_Thriller.png"**},  
 {**"artist"**: **"Linkin Park"**, **"title"**: **"Meteora"**, **"image\_url"**: **"https://image.ceneostatic.pl/data/products/48665992/i-linkin-park-meteora-album-cover-sticker.jpg"**},  
]  
  
*# Delete database file if it exists currently***if** os.path.exists(**"music\_store.db"**):  
 os.remove(**"music\_store.db"**)  
  
*# Create the database*db.create\_all()  
  
*# populate the database***for** album **in** Albums:  
 a = Album(artist=album.get(**"artist"**), title=album.get(**"title"**), image\_url=album.get(**"image\_url"**))  
 db.session.add(a)  
  
db.session.commit()

1. Uruchom plik index.py, przejdź do strony [http://localhost:8080/api/ui/#/](http://localhost:8080/api/ui/" \l "/) i przetestuj działanie API – spróbuj dodać nowe albumy, zaktualizować istniejące:

Przykladowy JSON – W PUNCKIE 20 ZAMIENIAMY „3 albumy” na jeden kod (patrz oryginalny kod do index.html

{"artist": "Metallica", "title": "New Metallica", "image\_url": "https://www.eventim.de/magazin/fileadmin/\_processed\_/a/8/csm\_metallica-tour-tickets-2019-1\_292aad9ab2.jpg"}



1. Stwórz folder templates w folderze server i umieść w nowo utworzonym folderze templates plik base.html:

{# ``base.html`` is the template all our other templates derive from. While  
 Flask-Bootstrap ships with its own base, it is good form to create a custom  
 one for our app, as it allows customizing some aspects.  
  
 Deriving from bootstap/base.html gives us a basic page scaffoling.  
  
 You can find additional information about template inheritance at  
  
 http://jinja.pocoo.org/docs/templates/#template-inheritance  
#}  
{%- extends "bootstrap/base.html" %}  
  
{# We also set a default title, usually because we might forget to set one.  
 In our sample app, we will most likely just opt not to change it #}  
{% block title %}Music Store{% endblock %}  
  
{# While we are at it, we also enable fixes for legacy browsers. First we  
 import the necessary macros: #}  
{% import "bootstrap/fixes.html" as fixes %}  
  
{# Then, inside the head block, we apply these. To not replace the header,  
 ``super()`` is used: #}  
{% block head %}  
{{super()}}  
  
{#- Docs: http://pythonhosted.org/Flask-Bootstrap/macros.html#fixes  
 The sample application already contains the required static files. #}  
{{fixes.ie8()}}  
{%- endblock %}  
  
{# Adding our own CSS files is also done here. Check the documentation at  
 http://pythonhosted.org/Flask-Bootstrap/basic-usage.html#available-blocks  
 for an overview. #}  
{% block styles -%}  
 {{super()}} {# do not forget to call super or Bootstrap's own stylesheets  
 will disappear! #}  
 <**link rel="stylesheet" type="text/css"  
 href="{{url\_for('static', filename='main.css')}}"**>  
{% endblock %}

1. Do folderu templates dodaj jeszcze plik index.html:

{# This simple template derives from ``base.html``. See ``base.html`` for  
 more information about template inheritance. #}  
{%- extends "base.html" %}  
  
{# Loads some of the macros included with Flask-Bootstrap. We are using the  
 utils module here to automatically render Flask's flashed messages in a  
 bootstrap friendly manner #}  
{% import "bootstrap/utils.html" as utils %}  
  
  
{# Inside the ``content`` is where you should place most of your own stuff.  
 This will keep scripts at the page end and a navbar you add on later  
 intact. #}  
{% block content %}  
 <**div class="container"**>  
 {%- with messages = get\_flashed\_messages(with\_categories=True) %}  
 {%- if messages %}  
 <**div class="row"**>  
 <**div class="col-md-12"**>  
 {{utils.flashed\_messages(messages)}}  
 </**div**>  
 </**div**>  
 {%- endif %}  
 {%- endwith %}  
 <**div class="jumbotron"**>  
 <**h1**>Welcome to iSA Music Store</**h1**>  
 <**p**>This example application demonstrates some of the  
 features of <**a href="http://pythonhosted.org/Flask-Bootstrap"**>  
 Flask-Bootstrap</**a**>.</**p**>  
 <**p**>  
 <**a class="btn btn-lg btn-default" href="http://pythonhosted.org/Flask-Bootstrap"  
 role="button"** >Show docs</**a**>  
 </**p**>  
 </**div**>  
 <**div class="row"**>  
  
 <**div class ="col-md-4"**>  
 <**div class="thumbnail"**>  
 <**img src="https://i.iplsc.com/-/000712WJNHHX8EBF-C122.jpg"**>  
  
 <**div class="caption"**>  
 <**h3**>Thriller</**h3**>  
 <**p**>  
 This is album of Michael Jackson - Thriller  
 </**p**>  
 </**div**>  
 </**div**>  
 </**div**>  
  
 <**div class ="col-md-4"**>  
 <**div class="thumbnail"**>  
 <**img src="https://image.ceneostatic.pl/data/products/48665992/i-linkin-park-meteora-album-cover-sticker.jpg"**>  
  
 <**div class="caption"**>  
 <**h3**>Meteora</**h3**>  
 <**p**>  
 This is album of Linkin Park Band  
 </**p**>  
 </**div**>  
 </**div**>  
 </**div**>  
  
 <**div class ="col-md-4"**>  
 <**div class="thumbnail"**>  
 <**img src="https://i.iplsc.com/-/000712WJNHHX8EBF-C122.jpg"**>  
  
 <**div class="caption"**>  
 <**h3**>Thriller</**h3**>  
 <**p**>  
 This is album of Michael Jackson - Thriller  
 </**p**>  
 </**div**>  
 </**div**>  
 </**div**>  
  
 <**ul**>  
 {% for album in albums %}  
 <**li**>{{ album.artist }} {{ album.title }}</**li**>  
 {% endfor %}  
 </**ul**>  
  
 </**div**>  
  
  
  
 </**div**>  
{%- endblock %}

1. Zaktualizuj plik index.py:

**from** flask **import** render\_template  
**from** Day13.server.app **import** app\_connexion  
**from** Day13.server.models **import** Album, AlbumSchema  
  
*# Get the application instance*connex\_app = app\_connexion  
  
*# Read the swagger.yml file to configure the endpoints*connex\_app.add\_api(**"swagger.yml"**)  
  
*# create a URL route in our application for "/"*@connex\_app.route(**"/"**)  
**def** home():  
 *"""  
 This function just responds to the browser URL  
 localhost:5000/* **:return***: the rendered template "index.html"  
 """* albums = Album.query.order\_by(Album.artist).all()  
  
 *# Serialize the data for the response  
 # album\_schema = AlbumSchema(many=True)  
 # dump = album\_schema.dump(albums)  
 # data = dump.data* **return** render\_template(**"index.html"**, albums=albums)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 connex\_app.run(port=8080, debug=**True**)

1. Uruchom plik index.py I przejdz do strony <http://localhost:8080/>
2. Zmien kod w pliku index.htm, aby albumy nie były wyświetlane jako prosta lista, ale jako „karty” – uzyj istniejącego kodu z elementem

<**div class ="col-md-4"**>  
 <**div class="thumbnail"**>