[Draw your reader in with an engaging abstract. It is typically a short summary of the document. When you’re ready to add your content, just click here and start typing.]

Web Advanced API Documentation

Georgi Stoychev, 536097

Gerralt Gottemaker

API description template

Please see the template tables (for each verb) that you can use to create the API specification that fits the given API (posted on Blackboard). To see an example for each Verb, see the result for the Server-side homework week 1.

Instead of this document and templates, you can use other tools to create the API specification as well. An example of such a tool is the Swagger Editor (https://editor.swagger.io/).

Table of Contents

1. Class diagram

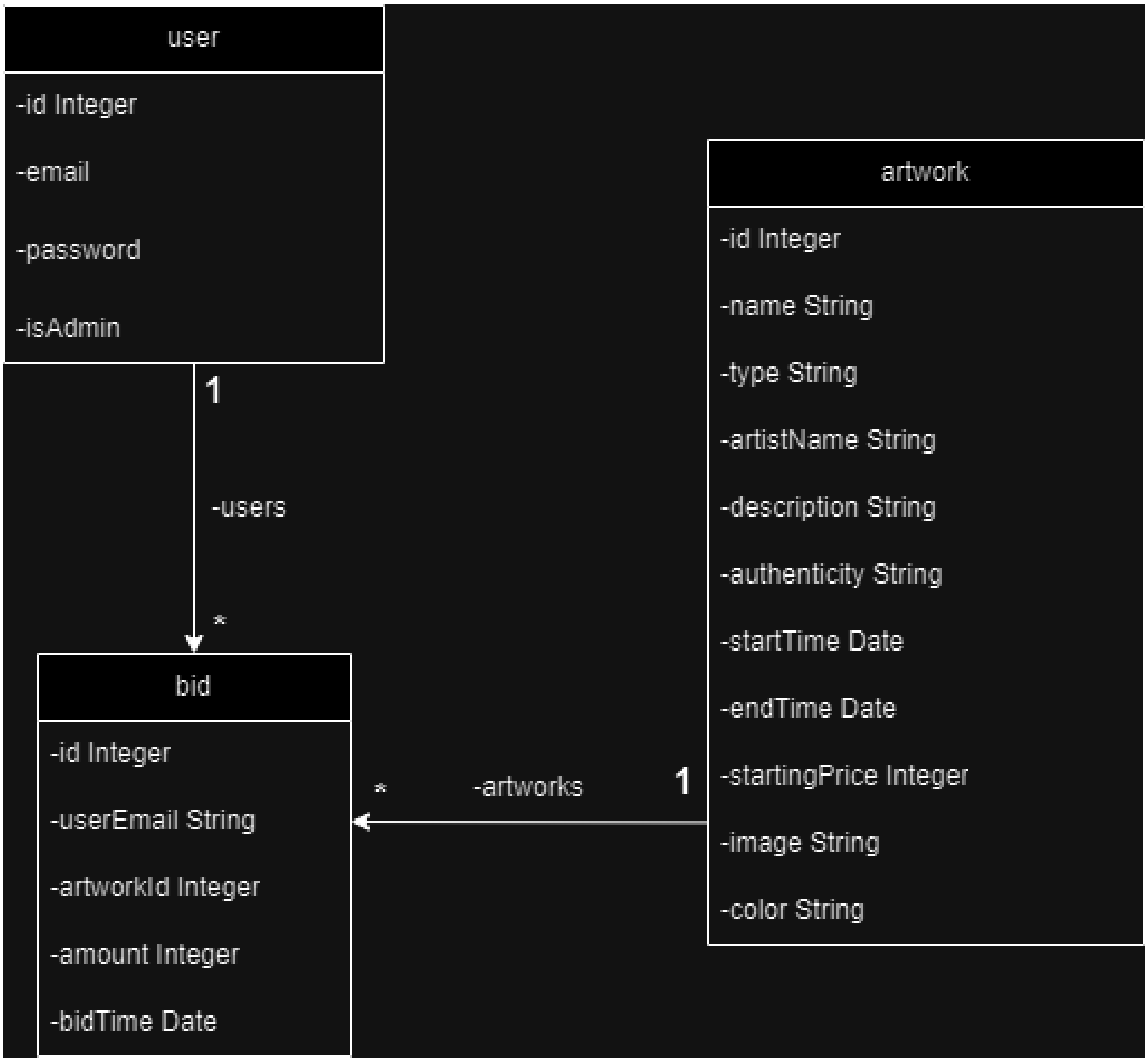

Figure 1 Class Diagram for My Auction Website- ArtPickle™

The above class diagram showcases the layout of my auction website with three entities: User, Artwork, and Bid.

User: Contains credentials and a boolean indicating administrative rights (isAdmin).
Artwork: Details items up for auction, including pricing and auction timing.
Bid: Records bids on artworks, including the bid amount and time and on which artwork the bid was placed, by which user.
Each of the above entities has a unique identifier (id).
Relationships:

Each User can place multiple Bids, but a Bid is associated with only one User (one-to-many relationship).

Each Artwork can have multiple Bids from various users, but a bid belongs to one user reflecting the one-to-many relationship.
The use of userEmail in Bid instead of a userId is a design choice of mine which as I write this I realize is not the best choice in terms of security concerns. The design shows that Artwork and User entities are independent of each other, connected only through Bid entities. This structure allows tracking of which users are bidding on which artworks without direct association between users and artworks.

2. GET requests

Add your requests here. Copy-paste the template for each different request.

Template GET table:

GET	**http://localhost:3000/artworks**		
Gets and displays all artworks.			
Parameters:	**Name**	**Type**	**Description**
** required*		path	Displays all the artworks
	type	query	Filters artworks by the type and displays the artwork/s associated with that desired type.
	price	query	Filters the artwork on the price.
	color	query	Filters artworks by the color type and displays the artwork/s of the specified color
Responses:	**Code**	**Description / example if successful**	
	200	List of artworks. Can be empty. Sends JSON.	

GET	**http://localhost:3000/artworks/{id}**		
Gets one artwork, based on given id			
Parameters:	**Name**	**Type**	**Description**
** required*	id*	path	id of artwork to find
Responses:	**Code**	**Description / example if successful**	
	200	Displays details of the artwork with the given id. Sends JSON.	
	404	Thrown when no artwork can be found. Message : 'The artwork was not found'	

GET	http://localhost:3000/artworks/won/userEmail		
Gets the won auctions/artworks by a user			
Parameters:	**Name**	**Type**	**Description**
** required*	email	path	Displays all the auctions won by the user with the specified user email
Responses:	**Code**	**Description / example if successful**	
	200	Returns all the won auctions by the user, or an empty array if the user did not have the higgest bid (if the user did not end up winning any auctions). Sends JSON.	

GET	http://localhost:3000/users		
Gets and displays all registered users.			
Parameters:	**Name**	**Type**	**Description**
** required*		path	Displays all the users
Responses:	**Code**	**Description / example if successful**	
	200	List of all users. Can be empty. Sends JSON.	

GET	http://localhost:3000/users/{id}		
Fetches details of a specific user by their id			
Parameters:	**Name**	**Type**	**Description**
** required*	id*	path	id of the user to fetch
Responses:	**Code**	**Description / example if successful**	
	200	Displays details of the user with the given id. Sends JSON.	
	404	Thrown when no user can be found. Message : `User with ID ${id} not found!`	

GET	http://localhost:3000/artworks/{Id}/bids		
Fetch all bids for a specific artwork			
Parameters:	**Name**	**Type**	**Description**
** required*	Id*	path	Id of artwork to fetch bids for
Responses:	**Code**	**Description / example if successful**	
	200	List of bids for the given artwork, can be empty if no bids were placed. Sends JSON.	
	404	Artwork not found. Message : 'Artwork not found'	

GET	**http://localhost:3000/tokens**		
Gets user data after verifying provided JWT token.			
Parameters:	**Name**	**Type**	**Description**
** required*			Authorization header with the 'Bearer ' prefix and jwt token
Responses:	**Code**	**Description / example if successful**	
	200	Returns user data without the password. Sends JSON.	
	400	'Authorization header is missing or incorrect.'	
	401	'Invalid token.'	
	404	'User not found.'	

3. POST requests

Template POST table:

POST	**http://localhost:3000/artworks**		
Adds a new artwork.			
Parameters:	**Name**	**Type**	**Description**
** required*	artwork *	body	The artwork to add. There is a validation on the server side that it needs to pass in order to be successfully posted. Example: example of json body { "name": "The Starry Night", "type": "painting", "artistName": "Vincent Van Gogh", "description": "The Starry Night is an oil-on-canvas painting by the Dutch Post-Impressionist painter Vincent van Gogh.", "authenticity": "yes", "startTime": Date.now(), "endTime": Date.now() + AUCTION_DURATION_3_MINUTES, "startingPrice": 2090, "image": "https://lh3.googleusercontent.com", "color": "blue", "bids": [ ]

			}
Responses:	**Code**	**Description / example if successful**	
	201	Artwork created. Sends JSON of created artwork.	
	400	Invalid input data	

POST	**http://localhost:3000/ users**		
Adds/registers a new user			
Parameters:	**Name**	**Type**	**Description**

** required*		body	The user data to be added. Must be JSON with email and password.
			
			Example: example of json body
			{
			
			"email": "test1@example.com",
			"password": "Testing1!
			
			}

Responses:	**Code**	**Description / example if successful**	
	201	User created. Returns JSON with user details and token	
	400	'Email and password are required' or 'Email already exists'.	
	500	Internal server error with the error message.	

POST	**http://localhost:3000/artworks/{Id}/bids**		
Adds a bid to the specified artwork.			
Parameters:	**Name**	**Type**	**Description**
** required*	id*	path	The ID of the artwork to which the bid is being placed
	userEmail	body	The email of the user placing the bid.
	amount	body	The amount of the bid. Must be a number greater than the last bid, must be positive number, cannot be String, only int value.
Responses:	**Code**	**Description / example if successful**	
	201	Bid added to artwork. Returns JSON of the new bid.	
	400	'Bid must be higher than the last bid.' or 'Auction has ended.'	
	404	'Artwork not found'	

POST	http://localhost:3000/tokens		
Creates and returns a new JWT token. (Logs in a user with authorization)			
Parameters:	**Name**	**Type**	**Description**
** required*	email*	body	The email address of the user.
	password*	body	The password of the user.
Responses:	**Code**	**Description / example if successful**	
	201	Token created. Returns JSON with the new token.	
	400	'Email and password are required.' or 'No user found with that email.' or 'Incorrect password.'	

4. PUT requests

Template PUT table:

PUT	http://localhost:3000/artworks/{id}		
Edits the details of an existing artwork.			
Parameters:	**Name**	**Type**	**Description**
** required*	id *	path	The id of the artwork to be modified
	image	body	Updated image URL for the artwork.
	name	body	Updated artwork name
	type	body	Updated artwork type
	artistName	body	Updated name of the artist
	description	body	Updated artwork description
	authenticity	body	Updated authenticity status of the artwork
	startTime	body	Updated artwork/auction start time
	endTime	body	Updated artwork/auction end time
	startingPrice	body	Updated starting price for the artwork. Must be a positive number.
	color	body	Updated color choice
Responses:	**Code**	**Description / example if successful**	
	201	Artwork details updated. Returns JSON of the artwork.	
	400	'Price must be a positive number.'	
	404	'Artwork not found'	

5. DELETE requests

Template PUT table:

DELETE	http://localhost:3000/artworks/{id}		
Deletes the specified artwork, based on the given id			
Parameters:	**Name**	**Type**	**Description**
** required*	id *	path	The id of the artwork to be deleted.
Responses:	**Code**	**Description / example if successful**	
	204	Artwork successfully deleted, no content returned.	
	404	'Artwork with id [id] not found'	

DELETE	http://localhost:3000/artworks/{id}/bids/{bidId}		
Deletes a bid from the specified artwork.			
Parameters:	**Name**	**Type**	**Description**
** required*	id	path	The id of the artwork from which the bid is deleted.
	bidId	path	The id of the bid to be deleted.
Responses:	**Code**	**Description / example if successful**	
	204	Bid successfully deleted, no content returned.	
	404	'Artwork with id [id] not found' or 'Bid with id [bidId] not found'	