Verb	Paramètres	Corp de la demande	Type de réponse	Fonction

POST	/api/auth/signup	{
		firstname : string,
		lastname : string,
		email : string,
		password: string
		}
			{
			userId : string,
			firstname : string,
			lastname : string,
			isAdmin :boolean,
			isDeleted: boolean,
			photourl: string,
			token: string
			}
				Ajout d'utilisateur. Dans la table users une ligne est ajoutée avec des valeurs pour les champs passés dans la requête et pour les autres :
				isAdmin :false,
				isDeleted: false,
				photourl: null.

POST	/api/auth/login	{
		email: string,
		password : string
		}
			Cookie('jwt', jwtvalue,
			{ httpOnly: true,
			maxAge : 24 * 60 * 60 * 1000})
			{
			userId: string,
			firstname : string,
			lastname : string,
			isAdmin :boolean,
			isDeleted: boolean,
			photourl: string,
			token : string
			}
				Login d'utilisateur

GET	/api/auth/user	cookie	{
			userId : string,
			firstname : string,
			lastname : string,
			isAdmin :boolean,
			isDeleted: boolean,
			photourl: string,
			}
				Authentification d'un utilisateur

POST	/api/auth/logout		Cookie ('jwt', '',
			{maxAge: 0})
			{
			message: 'success'
			}
				Déconnection d'un utilisateur

GET	/api/users/		Tableau des objets :
			{
			userId: number,
			firstname: String,
			lastname: String,
			email: String,
			photourl: String,
			isAdmin: boolean,
			isDeleted: boolean
			}
				Recevoir tous les utilisateurs comme tableau des objets

GET	/api/users/:id		{ userId: number,	Renvoie l'user avec l'id fourni

			firstname: String, lastname: String, email: String, photourl: String, isAdmin: boolean, isDeleted: boolean }	

PUT	/api/users/:id	Body
		{
		firstname: String,
		lastname: String
		email: String
		}
		File
			res.status(200):
			{
			message: L'utilisateur modifié !
			}
			res.status(400)
			{ error : "L'utilisateur n'est pas trouvé !"} si l'utilisateur avec cet id n'est pas dans la base users
			res.status(400)
			{error} en cas d'une erreur lors d'update
				Met à jour l'utilisateur avec l'identifiant fourni. Si une image est téléchargée, capture et met à jour l'image URL de l'utilisateur + change les. Si aucun fichier n'est fourni, seuls les champs sont update.

DELETE	/api/users/:id		res.status(200)
			{ message: "L'utilisateur supprimé !" }
			res.status(400)
			{ error: "Une erreur est survenu lors de suppression de l'utilisateur !" }
				Suppression d'un utilisateur
				La ligne dans la table **users** n'est pas supprimé, elle est remplacé par :
				{
				firstname: "L'utilisateur est désinscrit",
				lastname: '',
				email: '',
				password: '',
				photourl: null,
				isdeleted: true
				}

DELETE	/api/users/:id/admin		res.status(200) { message: "L'utilisateur supprimé !" } res.status(400) { error: "Une erreur est survenu lors de suppression de l'utilisateur !" }	Suppression d'un utilisateur par admin La ligne dans **users** est supprimée avec les posts et comments en cascade.
GET	/api/posts/		{ postId: Number, content:String, postIsDeleted:, createdAt: DataTime, updatedAt: DataTime, userId: Number, lastname: String, firstname: String,	Renvoie un tableau des objets

			userPhotourl: String, userIsDeleted: Boolean, photourl:Boolean, likeCount: Number, dislikeCount: Number }	

GET	/api/posts/:id		{
			postId: Number,
			content:String,
			postIsDeleted:,
			createdAt: DataTime,
			updatedAt: DataTime,
			userId: Number,
			lastname: String,
			firstname: String,
			userPhotourl: String,
			userIsDeleted: Boolean,
			photourl:Boolean,
			likeCount: Number,
			dislikeCount: Number
			}
				Renvoie un post correspondant à un id

GET	/api/posts/:id/user		{
			postId: Number,
			content:String,
			postIsDeleted:,
			createdAt: DataTime,
			updatedAt: DataTime,
			userId: Number,
			lastname: String,
			firstname: String,
			userPhotourl: String,
			userIsDeleted: Boolean,
			photourl:Boolean,
			likeCount: Number,
			dislikeCount: Number
			}
				Renvoie tableau des objets post correspondant à un userId

POST	/api/posts/	Body
		{
		userId : Number,
		content : String
		}
		File
			{
			postId: Number,
			content: String,
			postIsDeleted: Boolean,
			createdAt: DataTime,
			updatedAt : DataTime,
			userId: Number,
			photourl: String,
			}
				Crée un nouveau post dans la base posts Et une ligne dans la table **post_photos** avec id de post et url de photo si un file est présent dans la requête.
				Renvoie le post crée dans la table posts

PUT	/api/posts/:id			Modification de post //n'est pas réalisé dans cette version
DELETE	/api/posts/:id		res.status(200) : { message: "Post supprimé !" } res.status(400) :	Suppression d'un post si il existe dans la base posts

			{ error: "Une erreur est survenu lors de suppression de post !" }	

POST	/api/posts/:id/like	{
		userId : Number,
		postId : Number
		}
			{
			isCreated : boolean,
			isDeleted: boolean,
			postId: Number,
			userId: Number,
			likeCount: Number,
			dislikeCount: Number
			}
				Fonction like d'un post. Ajoute une ligne dans la table de **likes** avec id de post et id d'utilisateur
				Si like pour ce postId et userId existe – pas de modifications;
				Si like n'existe pas et dislike n'existe pas – like est crée
				Si like n'existe pas et un dislike existe pour ce postId et userId – dislike est supprimé et like est crée

POST	/api/posts/:id/dislike		{
			isCreated : boolean,
			isDeleted: boolean,
			postId: Number,
			userId: Number,
			likeCount: Number,
			dislikeCount: Number
			}
				Fonction like d'un post. Ajoute une ligne dans la table de **dislikes** avec id de post et id d'utilisateur
				Si dislike pour ce postId et userId existe – pas de modifications;
				Si dislike n'existe pas et like n'existe pas – dislike est crée
				Si dislike n'existe pas et un like existe pour ce postId et userId – like est supprimé et dislike est crée

GET	comments/ :id		{
			commentId: Number,
			content: String,
			commentIsDeleted: ,
			createdAt: DateTime,
			updatedAt:DateTime
			userId: Number,
			lastname: String,
			firstname: String,
			email: String,
			userPhotourl: ,
			userIsDeleted: Boolean,
			photourl: String,
			commentlikeCount: Number,
			commentdislikeCount: Number
			}
				renvoie un tableau des comments correspondant à un postId

POST	comments/	Body
		{
		userId: Number,
		postId:Number,
		content: String,
		}
		File
			{
			commentId: Number,
			content: String,
			commentIsDeleted: ,
			createdAt: DateTime,
			updatedAt:DateTime
			userId: Number,
			postId:Number,
			photourl: String
				Crée un nouveau comment dans la table **comments** et renvoie un comment crée et une ligne dans les **comment_photos** si il y a file dans la requête

			}	
DELETE	comments/ :id		res.status(200) : { message: "Comment supprimé !" } Si comment n'existe pas - res.status(400) : { error: "Comment n'existe pas !" }	Suppression d'un comment

POST	comments/like	{
		userId : Number,
		commentId : Number
		}
			{
			isCreated :Boolean,
			isDeleted: Boolean,
			commentId:Number,
			userId:Number,
			commentlikeCount: Number,
			commentdislikeCount: Number
			}
				Like d'un comment
				Ajoute une ligne dans la table de **comment_likes** avec id de comment et id d'utilisateur
				Si like pour ce commentId et userId existe – pas de modifications;
				Si like n'existe pas et dislike n'existe pas – like est crée
				Si like n'existe pas et un dislike existe pour ce postId et userId – dislike est supprimé et like est crée

POST	comments/dislike	{
		userId : Number,
		commentId : Number
		}
			{
			isCreated :Boolean,
			isDeleted: Boolean,
			commentId:Number,
			userId:Number,
			commentlikeCount: Number,
			commentdislikeCount: Number
			}
				Dislike d'un comment
				Ajoute une ligne dans la table de **comment_dislikes** avec id de comment et id d'utilisateur
				Si dislike pour ce commentId et userId existe – pas de modifications;
				Si dislike n'existe pas et like n'existe pas – dislike est crée
				Si dislike n'existe pas et un like existe pour ce commentId et userId – like est supprimé et dislike est crée