P2 Documentation

Project Apps and Structure

My project only has one app called main. (There's also the default app). My project uses Django class based views. They are all found within main/views.py. Important files to understand my project are main/views.py, main/models.py, and main/serializers.py,

Model Design

Notes: when a model inherits from another model the inherited attributes are not listed.
Unless otherwise specified everything is a CharField. The Charfields are of different lengths but it would be too verbose to make a note of the length of everything. Bolded fields are foreign keys

Model name	Fields
User	Username, email, password, first_name, last_name, phone_number, pic(Imagefield)
Restaurant	**Owner(user)**, name, desc, address, postalcode, phonenumber, time
RestaurantPic	**Restaurant**, pic(Imagefield), priority(determines order of pictures - priority 0=logo)
Userevent	**Restaurant**, name, time(datetime), etype (stores what type of event)
Menuitem inherits from Userevent	Desc, price(numeric)
Blogpost inherits from Userevent	text
Ownerevent	**Restaurant**, **user**, time(datetime), etype (stores what type of event)
Comment inherits from Ownerevent	text
RestaurantLike	
Follows	

End points

Endpoints that require authentication have their names in bold
Fields that are required are also bolded

In order to not be repetitive some behavior is the same for all endpoints.
If a user who is unauthorized tries to access an end point that requires authentication the api will return a HTTP 401
If a user does not enter a required field they will receive a HTTP 400 Response

Endpoint Name - Method	Fields	Responses	Description
users/ - get		HTTP 200 + a paginated list of all the users	Gets a paginated list of all the users
users/ - post	**username, first_name, last_name, email, phonenumber,** pic(image)	HTTP 201 + the new users information	Creates a new user
edit_user/ - patch	username, first_name, last_name, email, phonenumber	HTTP 200	Edits a users information
token/refresh/ - post	**username, password**	HTTP 200 + tokens on login HTTP 401 on wrong password/username	Simplejwt tokenpairview
token/refresh/	**refresh**	HTTP 200 + token HTTP 401 on bad token	Simplejwt tokenrefreshview
restaurant/ - get	Name, address, desc	HTTP 200 + a paginated list of all the users	Gets a paginated list of all the restaurants. If name address or desc is provided filters by provided attributes
restaurant/ - post	**Name, desc, address, postalcode, phonenumber**	HTTP 200 + restaurant info if successfully created restaurant If user already has	Creates a new restaurant

		restaurant HTTP 409	
restaurant/ - patch	Name, desc, address, postalcode, phonenumber	HTTP 200 if successfully changed restaurant info If user does not have restaurant HTTP 409	Edits restaurant info
restaurant/ - delete		HTTP 200 if successfully deleted restaurant If user does not have restaurant HTTP 409	Deletes restaurant
restaurant_pics/ -get	restaurant_id	HTTP 200 is successfully got restaurant pictures HTTP 412 if restaurant with id restaurant_id does not exist	Gets paginated list of restaurant pictures
restaurant_pics/ - post	**Priority, pic(image)**	HTTP 201 + Pic info if successfully added picture to restaurant HTTP 412 attempted to add pics with the same priority	Add picture to restaurant
restaurant_pics/ - delete	**pic_id**	HTTP 200 successfully deleted HTTP 409 User doesn't have restaurant HTTP 412 picture with pic_id does not exist or doesn’t belong to user	Delete picture
menus/ - get	**restaurant_id**	HTTP 200 + Paginated List of menu items Or if restaurant does not exist HTTP 412	Get a restaurants menu items
menus/ - post	**Name, desc, price**	HTTP 201 + menuitem info on successful insert HTTP 412 if user does not have restaurant	Adds item to menu

menus/ - delete	**item_id**	HTTP 200 on successful delete HTTP 409 on user does not have restaurant HTTP 412 on menu item does not exist	Deletes item from menu
blogs/ - get	blog_id, restaurant_id	HTTP 200 returns paginated list of blog post Called with restaurant_id of a restaurant that does not exist	Gets paginated list of blog posts
blogs/ - post	**Text, name**	HTTP 200 + Post info on successful blog creation HTTP 412 if user does not have restaurant	Creates new blog post
blogs/ - delete	**blog_id**	HTTP 200 on successful deletion HTTP 409 if restaurant or blog post does not exist	Deletes Blog Post
comments/ -get	restaurant_id, user_id	HTTP 200 + comment info on successful comment get HTTP 412 if called with restaurant_id and/or user_id that dont exist	Get comments. If restaurant_id or user_id filters comments by user and/or restaurant
comments/ -post	**restaurant_id, text**	HTTP 200 + comment info on successful comment Post HTTP 412 if restaurant does not exist	Posts comments
restaurant_like/ -get	restaurant_id, user_id	HTTP 200 + like info on successful comment get HTTP 412 if called	Gets paginated list of likes, filters by restaurant or user_id if provided

		with restaurant_id and/or user_id that dont exist	
restaurant_like/ - post	**restaurant_id**	HTTP 200 on successful Like HTTP 412 on restaurant doesn't exist	Likes restaurant
restaurant_like/ - delete	**restaurant_id**	HTTP 200 on successful delete HTTP 412 on restaurant or like does not exist	Unlikes restaurant
follows/ -get	restaurant_id, user_id	HTTP 200 + follow info on successful comment get HTTP 412 if called with restaurant_id and/or user_id that don't exist	Gets paginated list of follows, filters by restaurant or user_id if provided
follows/ - post	**restaurant_id**	HTTP 200 on successful Like HTTP 412 on restaurant doesn't exist	Follows restaurant
follows/ - delete	**restaurant_id**	HTTP 200 on successful delete HTTP 412 on restaurant or follow does not exist	Unfollows restaurant
The Following Ends points are currently not implemented			
userNotifications/ - get		HTTP 200 with Pagination notifications	Get notifications for user (ie new item or blog post at followed restaurant)
ownerNotifications/ -get		HTTP 200 with Pagination notifications	Get notifications for restaurant owner (ie new follow, comment or like on restaurant)