#1

Database Tables: I would design the authorization module using three database tables:

. **Users table:** This table would store information about each user, such as user_id and user_name.

Users Table:

Column Name	Data Type	Description
user_id	int	Unique ID for each user
user_name	varchar	Name of the user

. **Roles table:** This table would store information about each role, such as role_id and role_name.

Roles Table:

Column Name	Data Type	Description
role_id	int	Unique ID for each role
role_name	varchar	Name of the role

. **Privileges table:** This table would store information about each privilege, such as privilege_id and privilege_name.

Privileges Table:

Column Name	Data Type	Description
privilege_id	int	Unique ID for each privilege
privilege_name	varchar	Name of the privilege

. I would also create a mapping table: **User_Role table:** This table would map each user to one or more roles.

User_Role Table:

Column Name	Data Type	Description
user_id	int	Foreign key to Users table
role_id	int	Foreign key to Roles table

. Role_Privilege table: This table would map each role to one or more privileges.

Role_Privilege Table:

Column Name	Data Type	Description
role_id	int	Foreign key to Roles table
privilege_id	int	Foreign key to Privileges table

#2

I would create four classes to read and write into the database tables:

User: This class would represent a user and would have properties such as user_id, user_name, and roles.

Role: This class would represent a role and would have properties such as role_id, role_name, and privileges.

Privilege: This class would represent a privilege and would have properties such as privilege_id and privilege_name.

AuthorizationDAO: This class would contain methods to read and write to the database tables.

#3

To create an API that returns all privileges for a user when the user id is passed to the API, I would create a method in the AuthorizationDAO class that takes a user id as input and returns a list of privileges for that user. This method would join the User_Role, Role_Privilege, and Privileges tables to get the privileges associated with the user.

Edge Cases:

If a user has no roles, the API would return an empty list. If a user has roles but no privileges, the API would return an empty list. If the user id passed to the API is invalid, the API would return an error message.

Scalability: The design of the database tables allows for scalability as it can handle a large number of users, roles, and privileges. The AuthorizationDAO class can be optimized for performance to handle a large number of requests.

Alternatives: An alternative design for the database tables would be to have a single table for both roles and privileges, with a column indicating whether the entry is a role or a privilege. This would simplify the database structure but would make querying for user privileges more complex.

Another alternative would be to use an external authorization service such as Keycloak or Auth0, which provide more advanced features such as multi-factor authentication and social logins. However, this would add complexity to the application and may not be necessary for smaller projects.

Overall, the design choices for the authorization module depend on the specific requirements of the application and the resources available for implementation and maintenance.