Important rules and remarks:

- You are allowed to use any source of information; **communication with third parties is forbidden**
- You must be able to explain your code and you must be able to perform modifications if asked; otherwise the exam is not passed
- If there is anything unclear, **ask** us
- You must use the technologies from the lab/lecture
- There are NO PARTIAL points, only the ones specified in the problem statement
- **The requirements MUST be solved in the order they appear in the problem statement; the points for any feature are given only if all previous features are completely correct**
- **You must accept the exam assignment (**available on the group/channel**) and work on the generated github repository**
- **Each feature (e.g. F1, F2 etc) will be developed on a different feature branch whose name contains the feature name; e.g: 'feature/F1', 'feature/F2' etc**
- **After solving each feature, the feature branch will be merged into the main/master/develop branch;** push your changes to github as often as possible
- **After solving each feature, mark that feature as solved (with an 'x') in the attendance file** (the attendance file will be shared on the group/channel)
- **After solving each feature, notify the teachers for evaluation; *you must present each feature before proceeding to the next one (***)***

BlogApp

The following domain entities and relations are given:

- AppUser (name: String, birthday: any)
- Address (city: String, street: String)
- Follower (name: String, birthday: any)
- Post (title: String, content: String)
- Comment (text: String)
- An AppUser has an Address
- A Follower has an Address
- Bidirectional one to many relationships between: AppUser and Follower; AppUser and Post; Post and Comment
- All relationships are lazily loaded

At the repository level, AppUserRepository extends
BlogRepository<T extends BaseEntity<ID>, ID extends Serializable> extends JpaRepository<T, ID>
There will be no other interfaces or classes at the repository level.
The AppUserRepository interface contains ONLY the following methods:

- AppUser findWithFollowers(Long id);
- AppUser findWithPostsAndFollowers(Long id);

The AppUserService interface contains ONLY the following methods:

- AppUser find(Long id); // returns the user without relations (e.g. followers, posts)
- AppUser findWithFollowers(Long id); // the posts relation is not fetched
- AppUser findWithPostsAndFollowers(Long id); // all relations are fetched

By calling any service method, a single select statement will be issued. All methods, as well as hibernate sql statements, will be logged.

A REST API will expose certain endpoints as specified below in the requirements of an Angular frontend. The project will be modular (core, web).

Hello user screen

At the address http://localhost:4200/blog/user-hello/{id}, the following text will be shown using any html elements: “Hello <user-name>”, where <user-name> is the name of the user with the id equal to {id} **[F1 5p].** The backend will be accessed using a POST request **[F2 15p].**

User with followers screen

At the address http://localhost:4200/blog/user-followers/{id}, the following elements will be shown:

- The text “User <user-name> with followers:”, where <user-name> is the name of the user with id equal to {id}
- An ordered list of the followers (of user with id equal to {id}); only the follower name will be shown

From the backend, using a single GET request, only the data for one user (the user with id equal to {id}) will be returned**,** with the followers relation only (i.e. without posts) **[F3 20p]**, and for each follower only the id and name are returned **[F4 20p].**

User with posts and followers screen

At the address http://localhost:4200/blog/user-posts-followers/{id}, the following elements will be shown:

- The text “User <user-name> with followers:”, where <user-name> is the name of the user with id equal to {id}
- An ordered list of the followers (of user with id equal to {id}); only the follower name will be shown
- An ordered list of the posts (of user with id equal to {id}); only the post title will be shown
- Within each post, an ordered list of the comments corresponding to that post; only the comment text will be shown

From the backend, using a single GET request, only the data for one user (the user with id equal to {id}) will be returned**,** with its followers and posts **[F5 20p]**, for each follower only the id, name and city are returned **[F6 30p],** for each post only the id, title and comments **[F7 30p]**, for each comment the id and text **[F8 30p]**.

User details screen

At the address http://localhost:4200/blog/user-details/{id}, **at most** the following elements will be shown, based on the **query-example** (details bellow) defined in the Angular service that access the backed for this screen:

- The text “User <user-name> details:”, where <user-name> is the name of the user with id equal to {id}
- An ordered list of the followers (of user with id equal to {id}); only the follower name will be shown
- An ordered list of the posts (of user with id equal to {id}); only the post title will be shown
- Within each post, an ordered list of the comments corresponding to that post; only the comment text will be shown

The Angular service corresponding to this component will always access a single backend endpoint using a single POST request, and the backend will return a result matching a **query-example** defined in the Angular service.

If query-example = {appUser: {id: id}} then only the text “User details:” will be shown and from the backend only the id of that user is returned (e.g: {id: ‘-1’}).

If query-example = {appUser: {id: id, name: 'a'}} then only the text “User <user-name> details:”, where <user-name> is the name of the user with id equal to {id} is shown. From the backend, only the user id and the corresponding user name are returned (e.g: {"id":-1,"name":"u1"}).

From one case to another, only the query-example is changing (i.e. no other code changes); this applies to all of the following features

[F9 30p].

If query-example = {appUser: {id: id, name: 'a', followers: [{name: 'f1', birthday: 'bd1'}]}} then the user data with followers (but without posts) is shown, and for each follower only the name and birthday fields are returned from the backend **[F10 40p]**.

If query-example = {appUser: {id: id, name: 'a', followers: [{name: 'f1', birthday: 'bd1'}], posts: [{title: 't1'}]}} then the user data with followers and posts is displayed (without comments to posts) and for each post only the title is returned from the backend **[F11 40p]**.

If query-example = {appUser: {id: id, name: 'a', followers: [{name: 'f1', birthday: 'bd1'}], posts: [{title: 't1', content: 'con1', comments: [{text: 't1'}]}]}} then for each post the content will also be returned from the backend, as well as the comments which will be displayed on the screen **[F12 40p]**.

The provided solution will work for any field combination (from the data model) **[F13 40p]**.

The provided solution will fetch data from the db according to what is specified in the query-example. For example, if the followers and posts are not specified in the query example then these relations will not be fetched from the db **[F14 40p]**.

Remarks:

- In each database table, there will be at least two records
- Each container of objects (i.e. list/set) will have at least two elements
- **(***)** details at the exam

- In the query-examples, any *values* (except the value of appUser id) has no relevance; so, for example, the value **‘a’** from the query example {appUser: {id: id, name: **'a'**}} is irrelevant, i.e., just an example.