Hi %Username%

You'll need to create a simple project using NestJS, TypeScript, GraphQL and ReactJS.

Note: The task will contain required and optional features. You can skip optional features, but their implementation will benefit you from others.

The project is very basic and simple. It's a simple blog with 2 types of users Admins and Authors. Posts can be commented on.

Use next types:

```
enum RoleEnum{
  Admin,
  Author
}

type User {
  id: String!
  name: String!
  email: String!
  role: RoleEnum!
  posts: [Post!]!
  comments: [Comment!]!
}

type Post {
  id: String!
  author: User!
  text: String!
  comments: [Comment!]!
}

type Comment {
  id: String!
  post: Post!
  author: User!
  text: String!
}
```

Tasks:

Prepare project structure, PostgreSQL using docker-compose, NestJS configuration module
Implement ReactJS frontend(You can you any UI Frameworks, for example Bootstrap)
(optional) Use styled-components for styling
(optional) Write unit tests using Jest
Authentication Page, which will be using JWT strategy on BE and local user storage in file (optional) Store users in DB
Authorization using Guards
Setup Roles for actions (Admin, Author) Admins can do everything, authors can create/delete their own posts and comments.
Create/update/delete post BE Mutation + FE Page

```
mutation{
  post(text:string, id?: string): Post
  deletePost(id:string): Boolean
}
```

Note:
author should be taken from authorization token
delete post should delete all comments for this post as well.

Create/update/delete comment BE Mutation + FE Page

```
mutation{
  comment(postId: string, text: string, id?: string)
  deleteComment(id:string) Boolean
}
```

Note: Author id should be taken from authorization token.

Posts BE Query + FE Page

```
query{
  posts()[Post!]{
    id: String!
    author: User!
    text: String!
    comments: [Comment!]!
  }
}
```

(optional) pagination for comments in posts
(optional) filters for comments in posts
Author with his/her posts/comments BE Query + FE Page

```
query{
  author(id: string): Author{
    id: String!
    email: String!
    posts: [Post!]!
    comments: [Comment!]!
  }
}
```

Notes:

You should use code first approach (it means, that your schema.graphql file will be automatically generated)

All results should be provided via link to GitHub repository. The solution should have a README.md with full installation/run instructions. Please provide meaningful commits following https://www.conventionalcommits.org/en/v1.0.0/ convention.