# Story 1: User

## Description:

As a user,  
I want to have a profile,  
So that I have a profile.

## Acceptance Criteria:

* A User Domain model
* Attributes:
  + Id: Number
  + Username: String -> Can’t be empty, can’t be null
  + Email: String -> Valid email, can’t be empty, can’t be null
  + Password: String -> Can’t be empty, Can’t be null, At least 8 characters
  + Creation\_date: LocalDate
  + Role: Role -> A separate domain enum object. Can be User or Admin.
  + Reputation: Level -> A separate domain enum object. Can be Beginner, Intermediate, Trusted. (We will add functionality for this later)

# Story 2: Thread

## Description:

As a user,  
I want to have threads,  
So I can ask questions.

## Acceptance Criteria:

* A Thread Domain class:
* Attributes:
  + Id: Number
  + Title: String -> Can’t be empty, can’t be null
  + Content: String -> Can’t be empty, can’t be null
  + Creation\_Date: LocalDate
  + Created\_By: User
  + Comments: List of Comment objects

# Story 3: Comment

## Description:

As a user,  
I want to have comments,  
So I can comment.

## Acceptance Criteria:

* A Comment Domain class:
* Attributes:
  + Id: Number
  + Content: String -> Can’t be empty, can’t be null
  + Creation\_Date: LocalDate
  + Created\_By: User
  + ThreadId: Number

# Story 4: Cosmos Database

## Description:

As an admin,  
I want to use cosmos database,  
So that I can save everything on azure.

## Acceptance Criteria:

* Replace the current Mongo code with Cosmos
* Create a UserRepository and these functions:
  + CreateUser
  + FindUserByUsername -> Returns User object
  + FindUserByEmail -> Returns User object
* Create a ThreadRepository and these functions:
  + CreateThread
  + FindThreadById -> Returns Thread object
  + FindTheadByUserUsername -> Returns Thread object
* Create a CommentRepository and these functions:
  + CreateComment
  + FindCommentById -> Returns Comment object
  + FindCommentByUserUsername -> Returns Comment object

# Story 5: Register User

## Description:

As a user,  
I want to register,  
So that I can access the app.

## Acceptance Criteria:

### Backend:

* There is a ‘/users/signup’ POST endpoint
* In the Request Body it takes:
  + A JSON with:
    - Username
    - Email
    - Password
* In the User Service:
  + The User object gets fetched by the username
  + If the user exists -> throw an exception
  + The User object gets fetched by the email
  + If the user exists -> throw an exception
  + Create a user object AND hash the password, use LocalDate.now() for the creation date, set the role to user by default.
  + Save the user in the repository
  + Authenticate the username, email address and role using JWT
  + Return the token, username and role

### Frontend:

* There’s a register page with a form:
* Form fields:
  + Username
  + Email
  + Password
  + These fields are also validated so that they’re not empty.
  + Throw the needed exceptions if needed.

# Story 6: Login User

## Description:

As a user,  
I want to login,  
So that I can access the app.

## Acceptance Criteria:

### Backend:

* There is a ‘/users/login’ POST endpoint
* In the Request Body it takes:
  + A JSON with:
    - Email
    - Password
* In the User Service:
  + The User object gets fetched by the email
  + If the user does not exist -> throw an exception
  + Compare the password with the hashed password. If they do not match -> throw an exception
  + Authenticate the username, email address and role using JWT
  + Return the token, username and role

### Frontend:

* There’s a login page with a form:
* Form fields:
  + Email
  + Password
  + These fields are also validated so that they’re not empty.
  + Throw the needed exceptions if needed.

# Story 7: Create Thread

## Description:

As a user,  
I want to create a thread,  
So that I can ask questions.

## Acceptance Criteria:

### Backend:

* There is a ‘/thread’ POST endpoint
* The user should be authorized to be able to create a thread.
* In the Request Body it takes:
  + A JSON with:
    - Title
    - Content
    - Username
* In the Thead Service:
  + The User object gets fetched by the username
  + If the user does not exist -> throw an exception
  + Create a thread object, creation date is the LocalDate.now()
  + Save it in the database
  + Return the created thread object

### Frontend:

* In the navigation bar there is a button ‘Create Thread’
* It’s a form with fields:
  + Title
  + Content
  + A post button
* The data gets send over to the backend

# Story 8: Create Comment

## Description:

As a user,  
I want to create a comment under a thread,  
So that I can ask questions.

## Acceptance Criteria:

### Backend:

* There is a ‘/comment/{threadId}’ POST endpoint
* The user should be authorized to be able to create a thread.
* In the Request Body it takes:
  + A JSON with:
    - Content
    - Username
* In the Comment Service:
  + The Thread object gets fetched by it’s ID
  + If the thread does not exist -> throw an exception
  + The User object gets fetched by the username
  + If the user does not exist -> throw an exception
  + Create a comment object, creation date is the LocalDate.now()
  + Save it in the database
  + Update the thread object in the database to include the comment
  + Return the comment object
* In the Thread Repository:
  + Create a function that adds a comment to the list of comments in a thread.

### Frontend:

* By clicking on a thread a thread opens up
* At the bottom of the thread there is a ‘comment section’.
* There is also an empty comment field for the user to write their comment in.
* By pressing the comment button OR by pressing enter you’re able to send the comment.

# Story 9: Test Data

## Description:

As an admin,  
I want to be able to test my application,  
So that I can validate my endpoints.

## Acceptance Criteria:

### Backend:

* Do your research :D We need a seed file.