Description of Input Validation

Backend:

joi package

We use **joi** package to validate request passing to our API.

When user call an API, it will call our controller code to call database operations.

For example, when user call

```
// PUT /api/user/:id
router.put("/:id", verifyToken, updateUserById);
```

It will call verifyToken and **updateUserById**, **verifyToken** is used to verity user token.

We will do validation of request input when entering **updateUserById**, here is the code:

```
const updateUserById = async (req, res) => {
  // Check if the request is valid
  const { error } = updateUserByIdValidation({
    id: req.params.id,
    ...req.body,
  });
  if (error) {
    return res
      .status(400)
      .send({ success: false, msg: error.details[0].message });
  }
  ...
```

Here we will call **updateUserByIdValidation** on our request input, if it fail to pass the validation, the backend will return error code 400.

And in **updateUserByIdValidation**, we use **joi** to do the validation,

```
// updateUser validation
const updateUserByIdValidation = (data) => {
  const schema = joi.object({
    id: joi
      .string()
      .regex(/^[0-9a-fA-F]{24}$/)
      .required(),
    password: joi.string(),
```

```
    avatarUrl: joi.string().min(0),
    isLocked: joi.boolean(),
    attempts: joi.number().min(0).max(3),
    lockTime: joi.date(),
    deactivated: joi.boolean(),
    hasNewMessage: joi.boolean(),
    hasNewNotification: joi.boolean(),
  });
  return schema.validate(data);
};
```

We will do the exactly procedure in every API request.

And for your convenience, here is our **user** model:

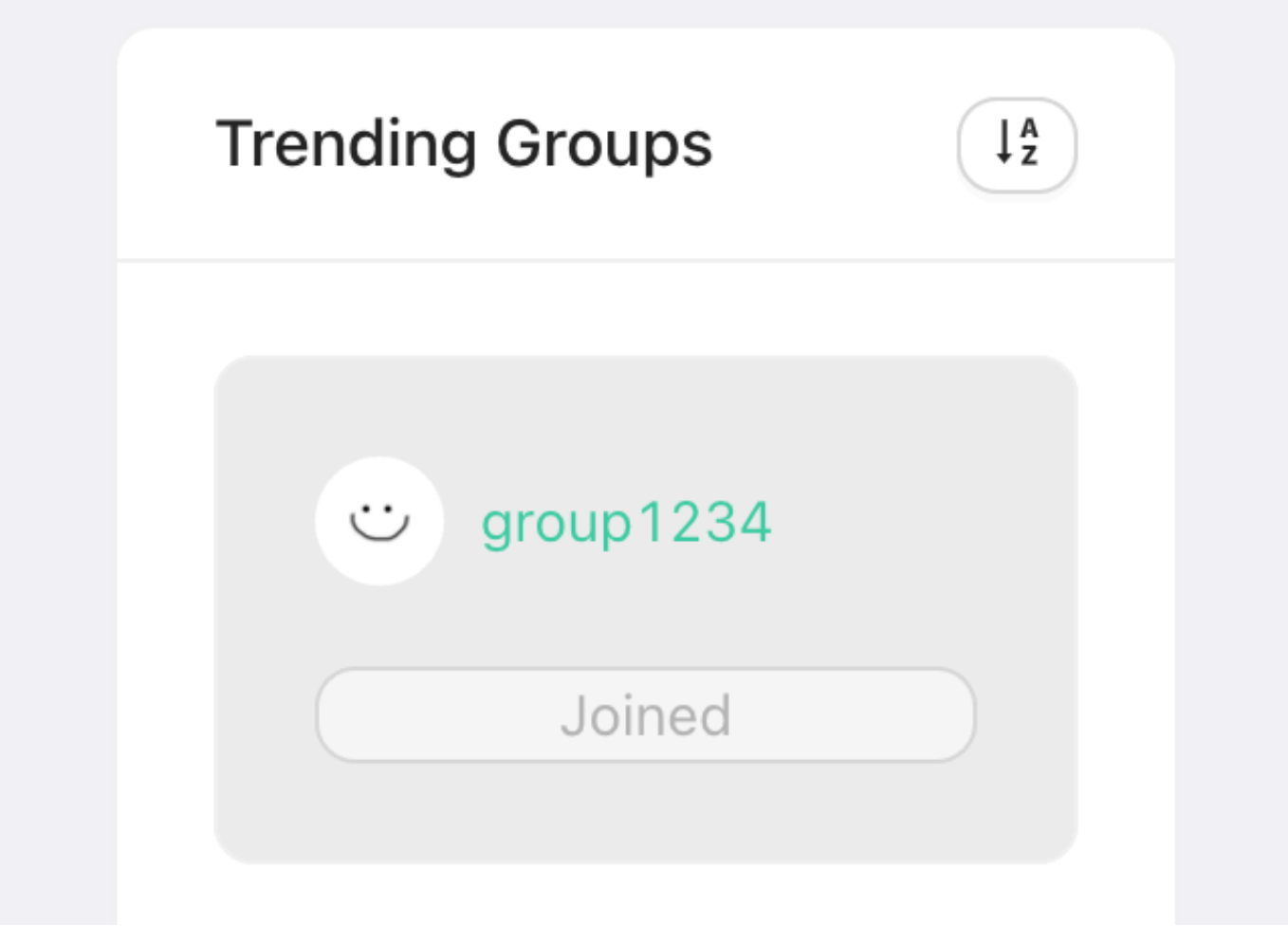
```
const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    min: 5,
    max: 20,
  },
  email: {
    type: String,
    required: true,
    min: 6,
    max: 255,
  },
  password: {
    type: String,
    required: true,
    min: 6,
    max: 1024,
  },
  date: {
    type: Date,
    default: Date.now,
  },
  avatarUrl: {
    type: String,
  },
  attempts: {
    type: Number,
    min: 0,
    max: 3,
    default: 0,
```

```
  },
  lockTime: {
    type: Date,
  },
  isLocked: {
    type: Boolean,
    default: false,
  },
  deactivated: {
    type: Boolean,
    default: false,
  },
  hasNewMessage: {
    type: Boolean,
    default: false,
  },
  hasNewNotification: {
    type: Boolean,
    default: false,
  },
});
```

Frontend:

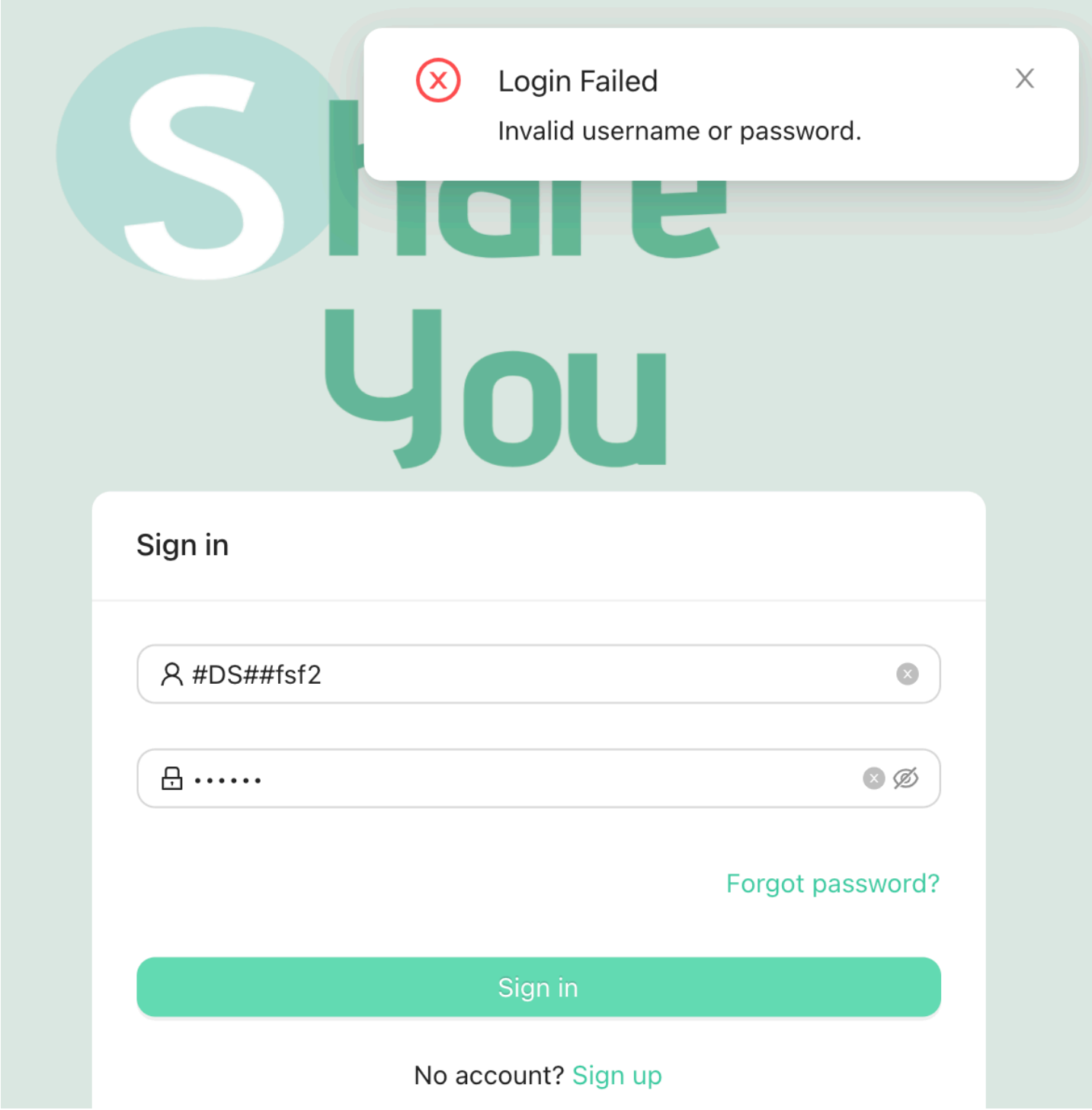
Disable Button:

In frontend, we will also restrict invalid user behavior by disable some button. For example, we want to prohibit user from requesting to join a group several times, we will check whether this request exists in our database and disable the join botton. And if the user has joined the group, the button should also be disabled.



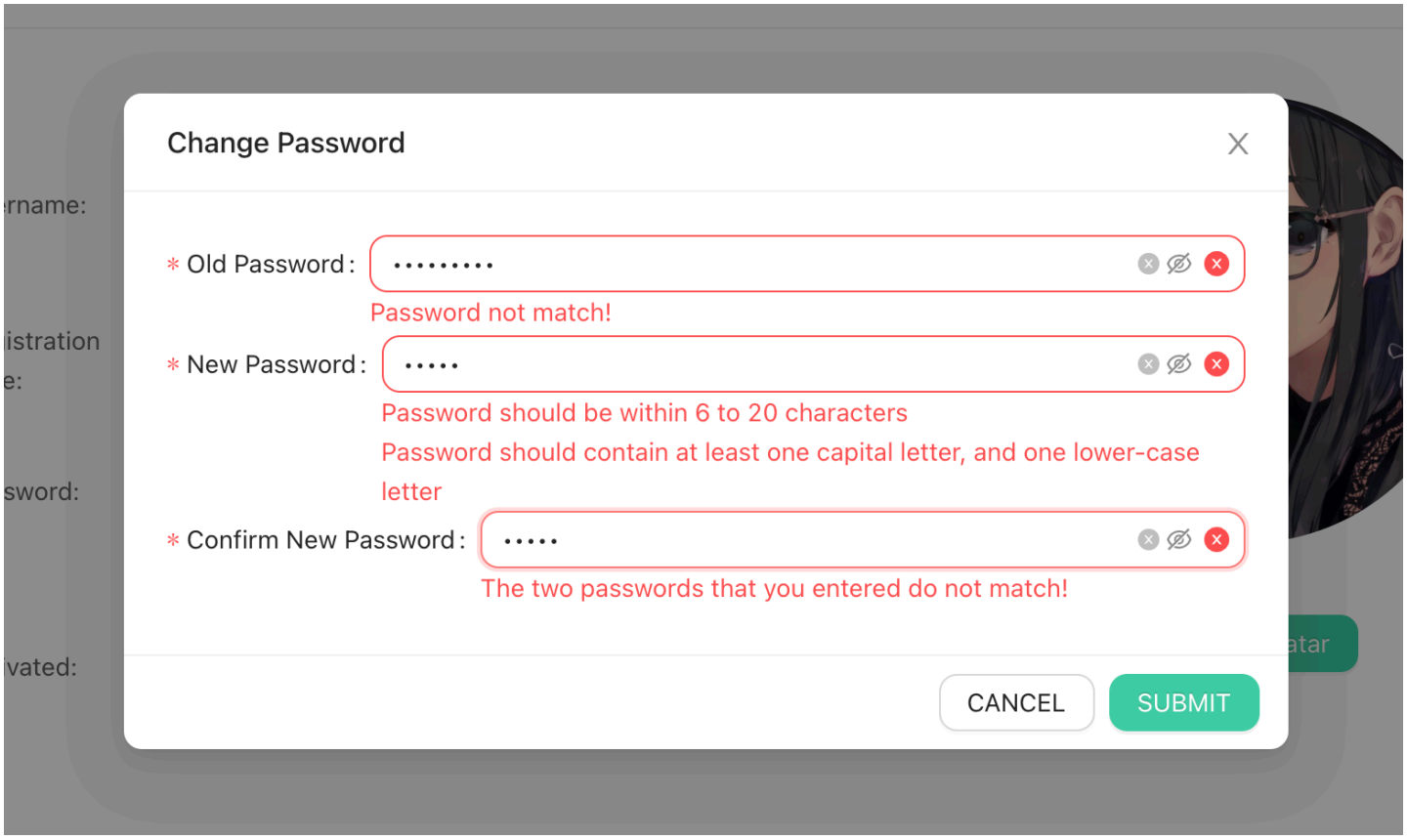
Register/Login:

When user register/login, we will use regex to judge whether the input is valid character string. Also, upon login, if user type some username not appeared in database, the user will also be noticed.



Change Password

Upon changing password, the frontend will judge whether the orginal password is input correctly and whether the new password differs from the orginal one and whether the verified new password same with the new password.


Change Password
* Old Password :
Password not match!
* New Password :
Password should be within 6 to 20 characters
Password should contain at least one capital letter, and one lower-case letter
* Confirm New Password :
The two passwords that you entered do not match!
CANCEL
SUBMIT