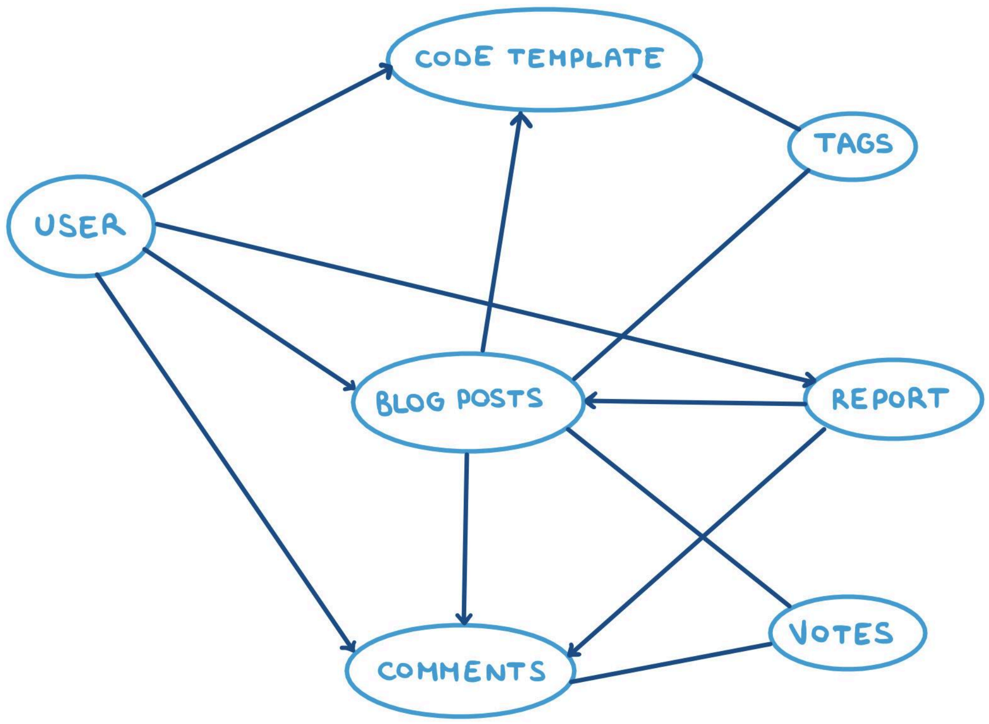
Scriptorium

Anna, Mohammed, JP

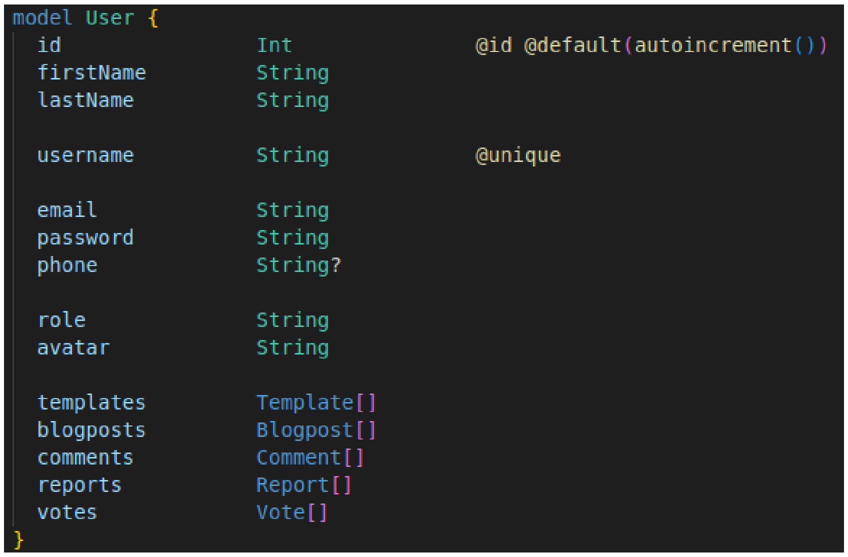
Model Design



We connected the models that must interact with each other. Users are able to create templates, blog posts, and comments. Templates have owners, they have tags, as well as blog posts that reference them. Blog posts also have owners, they reference code templates, and tags. They can be reported by users, and they have reply threads. Comments and blog posts have upvotes and downvotes.

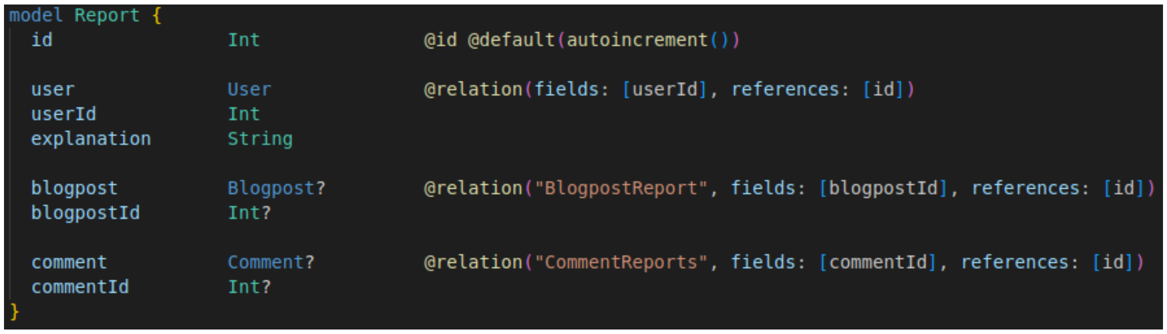
Users

- Users have to have personal information (first name, last name, email, phone number)
- They must have authentication (username, password)
- Role to determine access (user vs admin)
- Users can make code templates, make blog posts, and make comments
- They can also make reports


```
model User {
  id                  Int                 @id @default(autoincrement())
  firstName           String
  lastName            String

  username            String              @unique

  email               String
  password            String
  phone               String?

  role                String
  avatar              String

  templates           Template[]
  blogposts           Blogpost[]
  comments            Comment[]
  reports             Report[]
  votes               Vote[]
}
```

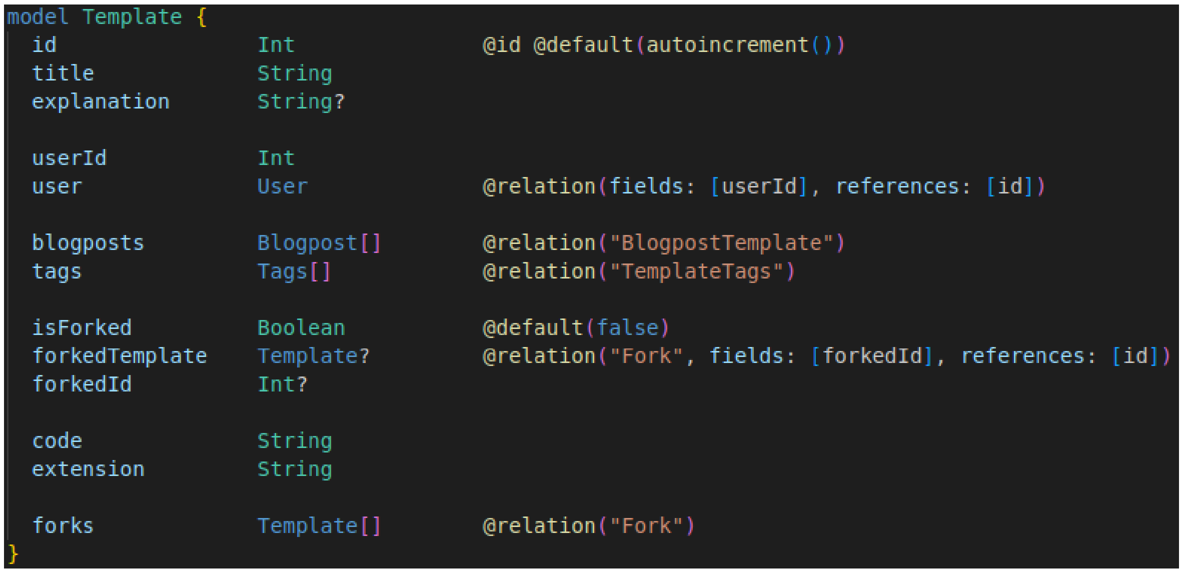
Report:

- Reports are made by a specific user, given an explanation for reporting.
- They can report a blogpost, and a comment


```
model Report {
  id                Int               @id @default(autoincrement())

  user              User              @relation(fields: [userId], references: [id])
  userId            Int
  explanation       String

  blogpost          Blogpost?         @relation("BlogpostReport", fields: [blogpostId], references: [id])
  blogpostId        Int?

  comment           Comment?          @relation("CommentReports", fields: [commentId], references: [id])
  commentId         Int?
}
```

Template:

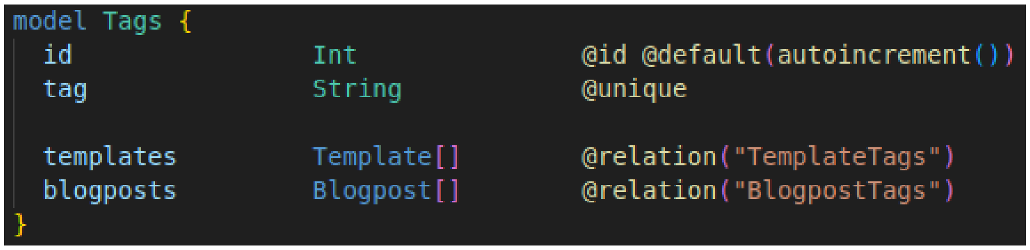
- Templates have titles, explanation, and list of tags
- They are also created by one specific user
- They contain code
- They can be forked by other users, and have reference to the original template



```
model Template {
  id              Int             @id @default(autoincrement())
  title           String
  explanation     String?

  userId          Int
  user            User            @relation(fields: [userId], references: [id])

  blogposts       Blogpost[]      @relation("BlogpostTemplate")
  tags            Tags[]          @relation("TemplateTags")

  isForked        Boolean         @default(false)
  forkedTemplate  Template?       @relation("Fork", fields: [forkedId], references: [id])
  forkedId        Int?

  code            String
  extension       String

  forks           Template[]      @relation("Fork")
}
```

Tags:

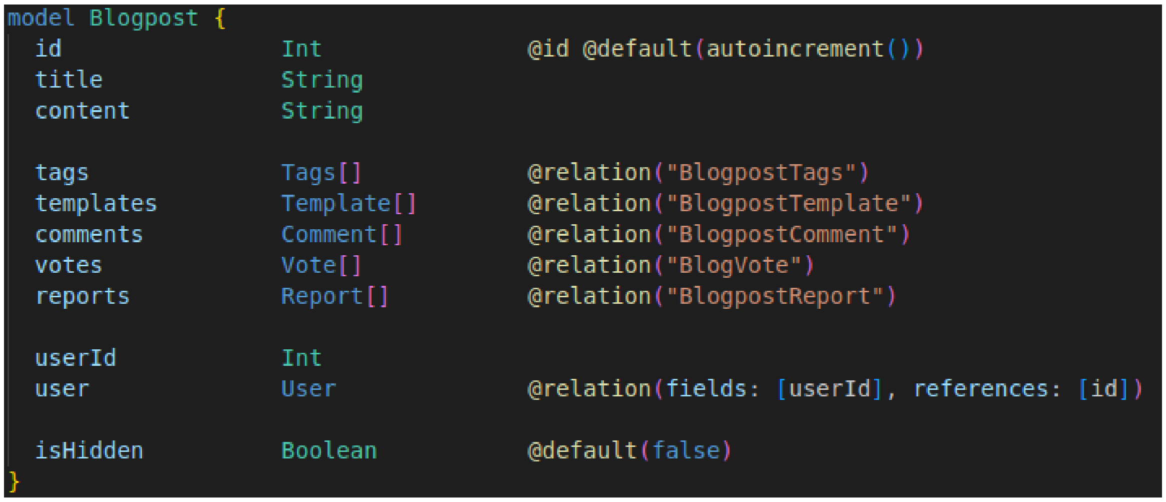
- Tags aren't owned by anyone
- They just have a string indicating the tag
- They are referenced on templates and blog posts



```
model Tags {
  id              Int             @id @default(autoincrement())
  tag             String          @unique

  templates       Template[]      @relation("TemplateTags")
  blogposts       Blogpost[]      @relation("BlogpostTags")
}
```

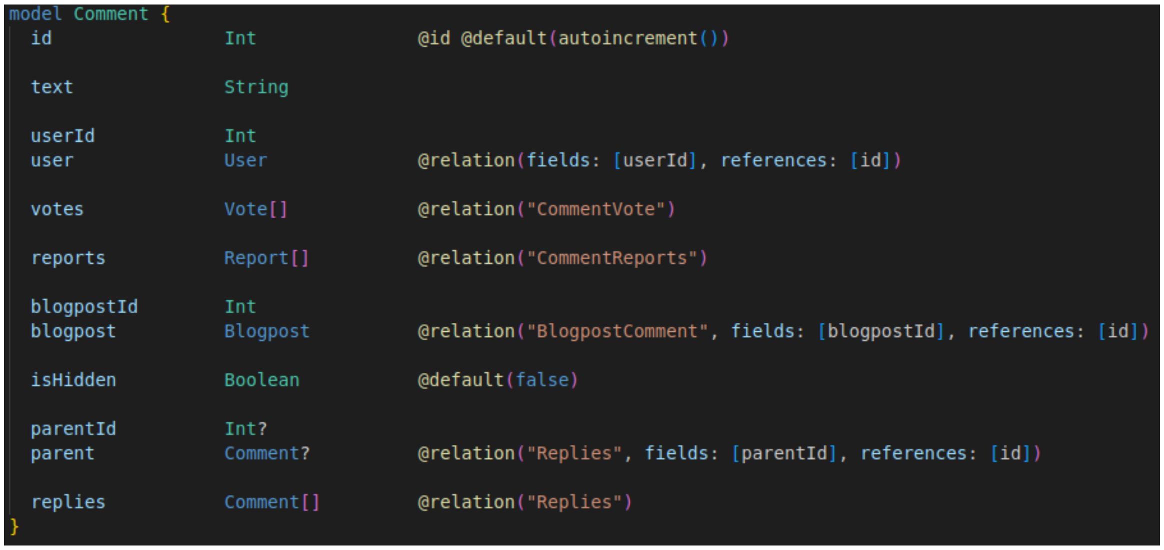
Blogposts:

- Blogposts have a title and content
- They have a list of tags, and list of templates that they reference
- They have comments
- Blogposts are created by a single user
- They can be upvoted and downvoted
- They can be reported, once they are reported, they are hidden to users except admins



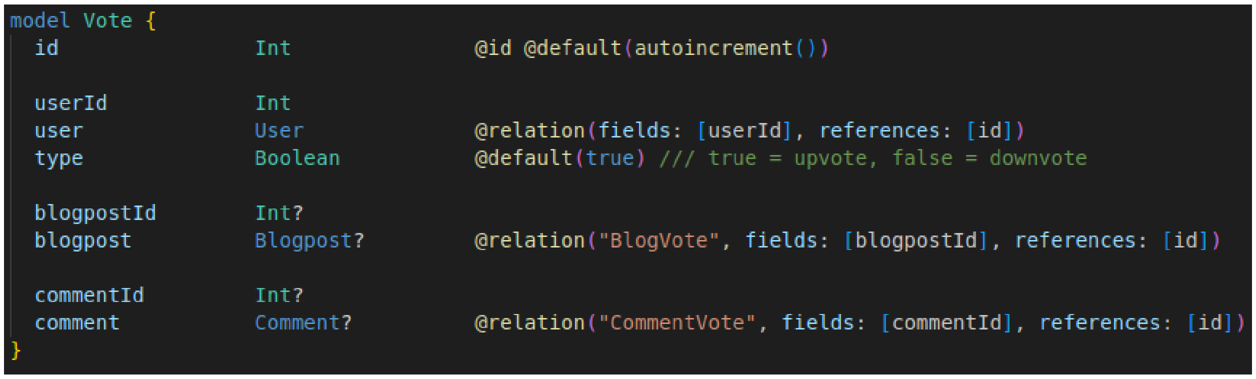
```
model Blogpost {
  id                  Int                 @id @default(autoincrement())
  title               String
  content             String

  tags                Tags[]              @relation("BlogpostTags")
  templates           Template[]          @relation("BlogpostTemplate")
  comments            Comment[]           @relation("BlogpostComment")
  votes               Vote[]              @relation("BlogVote")
  reports             Report[]            @relation("BlogpostReport")

  userId              Int
  user                User                @relation(fields: [userId], references: [id])

  isHidden            Boolean             @default(false)
}
```

Comment:
- Comments have texts and are made by a specific user
- They can be upvoted and downvoted
- They belong to a specific blog post
- They can be hidden once they are reported
- We have a hierarchical structure to the replies, and you can reply to a specific comment, or create a comment on the base blogpost



```
model Comment {
  id                  Int                 @id @default(autoincrement())

  text                String

  userId              Int
  user                User                @relation(fields: [userId], references: [id])

  votes               Vote[]              @relation("CommentVote")

  reports             Report[]            @relation("CommentReports")

  blogpostId          Int
  blogpost            Blogpost            @relation("BlogpostComment", fields: [blogpostId], references: [id])

  isHidden            Boolean             @default(false)

  parentId            Int?
  parent              Comment?            @relation("Replies", fields: [parentId], references: [id])

  replies             Comment[]           @relation("Replies")
}
```

Votes:
- Votes are made by a specific user
- They can be upvotes or downvotes
- They can either belong to a blogpost or a comment



```
model Vote {
  id            Int         @id @default(autoincrement())

  userId        Int
  user          User        @relation(fields: [userId], references: [id])
  type          Boolean     @default(true) /// true = upvote, false = downvote

  blogpostId    Int?
  blogpost      Blogpost?   @relation("BlogVote", fields: [blogpostId], references: [id])

  commentId     Int?
  comment       Comment?    @relation("CommentVote", fields: [commentId], references: [id])
}
```

API Endpoints:

Authentication API:

Method	API	Explanation
POST	/api/auth/signup	create account
POST	/api/auth/login	logging into created account
POST	/api/auth/logout	sends success message, though logging out achieved when token expires (1 hr)
PUT	/api/auth/updateprofile	updating details in user profile

Authentication API
/api/auth/signup - POST

- Create a user account. Admin accounts are not allowed to be created through a public api.
- Request body:
  - firstName
  - lastName
  - username
  - password
  - email
  - avatar
  - phoneNumber
  - role
- Example Request:

```
{
   "firstName": "JP",
   "lastName": "Medina",
   "username": "jpm3",
   "password": "mjp",
   "email": "jpm@gmail.com",
   "avatar": "asdf",
   "role": "user"
}
```

- Example Response:

```
{
    "username": "jpm3",
    "email": "jpm@gmail.com",
    "firstName": "JP",
    "lastName": "Medina",
    "role": "user"
}
```

/api/auth/login - POST

- Login in a user through credentials - username and password. A token is generated which allows a user to be logged in for 1 hour.
- Request Body:
    - username
    - password
- Example Request:

```
{
    "username": "jpm3",
    "password": "mjp"
}
```

- Example Response:

```
{
    "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjIsInVzZXJuYW1lIjoianBtMyIsI
nJvbGUiOiJ1c2VyIiwiaWF0IjoxNzMwNjc0OTM2LCJleHAiOjE3MzA2Nzg1MzZ9.2GnWd0ZKvzS4V-R
Mk3iYt4DNCfa8xTUb5ycfBeIE5zU"
}
```

/api/auth/logout - POST

- Logout a user. Endpoint does not actually do anything. When token expires, then logout will be achieved
- Request Body: None

/api/auth/updateprofile - PUT

- Update a users profile information
- Request body:
    - username
    - firstName
    - lastName
    - email

        - avatar
        - phoneNumber
- Example Request:

```
{
   "username": "jpp"
}
```

- Example Response:

```
{
   "message": "Profile updated",
   "user": {
       "id": 2,
       "firstName": "JP",
       "lastName": "Medina",
       "username": "jpp",
       "email": "jpm@gmail.com",
       "password":
"$2b$10$6ASo84vB3ZYRavqePn3ig.tALrryfbx9QCx5uDvSFS0lPVNCU9dTi",
       "phone": null,
       "role": "user",
       "avatar": "asdf"
   }
}
```

Template API:

Method	API	Explanation
GET	/api/template	Fetches all templates. (public)
GET	/api/template/[id]	View template with id. (public)
POST	/api/template/fork	Forks an existing template identified by id, creating a new template for the user who forked it. (public)
GET	/api/template/user	Fetches all templates for a specific user. (auth)
POST	/api/template/create	Creates a new template for a specific user. (auth)
PUT	/api/template/update	Updates a specific template by id for a specific user. (auth)
DELETE	/api/template/delete	Deletes a specific template by id for a specific user. (auth)

Template API

/api/template - GET

- Fetches all the templates (with pagination). Filters by title, tags, and code content.
- Request body:
    - title: string

    - tags: list of tag ids
    - codeContent: string
- Example Request - /api/template?title=Hello
- Example Response

```
{
   "templates": [
       {
           "id": 1,
           "title": "Python Hello World",
           "explanation": null,
           "userId": 2,
           "isForked": false,
           "forkedId": null,
           "code": "print('Hello from Python!')",
           "extension": "python"
       },
       {
           "id": 3,
           "title": "Javascript Hello World",
           "explanation": null,
           "userId": 3,
           "isForked": false,
           "forkedId": null,
           "code": "console.log(\"Hello from JavaScript!\");",
           "extension": "javascript"
       },
   ],
   "currentPage": 1,
   "totalPages": 1
}
```

/api/template/[id] - GET

- View template with specific id.
- Request body: None
- Example Request - /api/template/3
- Example Response

```
{
   "id": 3,
   "title": "Javascript Hello World",
   "explanation": null,
   "userId": 3,
   "isForked": false,
   "forkedId": null,
    "code": "console.log(\"Hello from JavaScript!\");",
    "extension": "javascript"
}
```

/api/template/fork - POST

- Forks an existing template by id. Creating a new template for the user who forked it. Must be an authenticated user to fork.
- Request body:
  - templId: template id to fork
- Example Request:
  -

/api/template/user - GET

- Fetches all templates for users currently signed in. Paginated result
- Request body: None
- Example Request: /api/template/user
- Example Response:

```
{
    "templates": [
        {
            "id": 1,
            "title": "Python Hello World",
            "explanation": null,
            "userId": 2,
            "isForked": false,
            "forkedId": null,
            "code": "print('Hello from Python!')",
            "extension": "python"
        },
        {
            "id": 3,
            "title": "Javascript Hello World",
            "explanation": null,
            "userId": 3,
            "isForked": false,
            "forkedId": null,
            "code": "console.log(\"Hello from JavaScript!\");",
            "extension": "javascript"
        },
    ],
    "currentPage": 1,
    "totalPages": 1
}
```

/api/template/create - POST

- Creates a new template for a specific user. The user comes from authentication.
- Request body:
  - title: string
  - explanation: string
  - code: string
  - extension: string - language of code template
  - tagsId: list - list of tag ids
- Example Request: /api/template/create

```
{
   "title": "Python Hello World",
   "code": "print('Hello from Python!')",
   "extension": "python",
   "tagsId": [2]
}
```

- Example Response:

```
{
   "title": "Python Hello World",
   "code": "print('Hello from Python!')",
   "extension": "python",
   "tagsId": [2]
}
```

/api/template/update - PUT

- Updates a specific template by id. Editing a template requires to be the author of the template.
- Request body:
  - tempId: template id to edit
  - title: string
  - explanation: string
  - code: string
  - extension: string
  - tagsAdded: list of tags to add
  - tagsRemoved: list of tags to remove
- Example Request:

```
{
   "tempId": 2,
   "explanation": "Error in code execution due to a unused variable in C."
}
```

- Example Response

```
{
   "id": 2,
   "title": "C Error",
   "explanation": "Error in code execution due to a unused variable in C.",
   "userId": 2,
   "isForked": false,
   "forkedId": null,
   "code": "#include <stdio.h>\n\nint unused_function() {\n    return
42;\n}\n\nint main() {\n    int x;\n    printf(\"Hello, world!\\n\");\n
return 0;\n}",
   "extension": "c"
}
```

/api/template/delete - DELETE

- Deletes a specific template by id for a user. It requires you to be the author of the template in order to delete it.
- Request body:
  - tempId: template id to delete
- Example Request: /api/template/delete

```
{
   "tempId": 2
}
```

- Example Response:

```
{
   "message": "Template deleted."
}
```

Tags API:

Method	API	Explanation
GET	/api/tags	Fetches all tags matching a search query.
GET	/api/tags/[id]	Get tag matching id.
GET	/api/tags/search	Search for tags matching a query.
POST	/api/tags/create	Create a new tag.
DELETE	/api/tags/delete	Deletes a tag by id if it is not in use.

Tags API:
/api/tags - GET

- Fetches all the tags matching a search query
- Request body: None
- Example Request: /api/tags/search?query=hello
- Example Response:
```
[
    {
        "id": 3,
        "tag": "hello-world"
    }
]
```

/api/tags/[id] - GET
- Get a specific tag by id
- Request body: None
- Example Request: /api/tags/3
- Example Response:
```
[
    {
        "id": 3,
        "tag": "hello-world"
    }
]
```

/api/tags/create - POST
- Create a new tag
- Request body:
    - tag: tag text
- Example Request:
```
{
    "tag": "java"
}
```
- Example Response
```
{
    "id": 1,
    "tag": "java"
}
```

/api/tags/delete
- Deletes a tag by id if its not in use. Prevents hard delete in case certain blog posts and templates reference it.
- Request Body:
    - id: id of tag to delete
- Example Request:

```
{
   "id": 1
}
```

- Example Response:

```
{
   "message": "Tag successfully deleted."
}
```

Execute API:

/api/code/execute - POST

- Executes a code template. Takes in stdin as standard input for the file
- Request body
  - tempId: id of template to execute
  - stdin: standard input to supply to file
- Example Request:

```
{
   "tempId": 1
}
```

- Example Response:

```
{
   "stderr": "Command failed: gcc -Werror -Wall -Wextra temp_c.c -o
temp_c\ntemp_c.c: In function ‘main’:\ntemp_c.c:8:9: error: unused variable ‘x’
[-Werror=unused-variable]\n    8 |     int x;\n      |         ^\ncc1: all
warnings being treated as errors\n"
}
```

Blogpost API:

Method	API	Explanation
POST	/api/blogpost/create	Create a blogpost
DELETE	/api/blogpost/delete	Delete a blogpost
GET	/api/blogpost/index	Search for blogposts
PUT	/api/blogpost/update	Update a blogpost
PUT	/api/blogpost/vote	Upvote or downvote a blogpost
GET	/api/blogpost/[id]	Get a blogpost by id

Blogpost API

/api/blogpost/create - POST

- Create a new blog post
- Request body:

    - title: title of the blog post
    - content: explaining the blog post
    - tagsId: tags that the blog post references
    - templatesId: templates that the blog post references
- Example Request:
```
{
   "title": "Hello world scripts",
   "content": "A list of templates with hello world scripts",
   "tagsId": [1, 2],
   "templatesId": [2, 4, 7]
}
```
- Example Response:
```
{
   "id": 1,
   "title": "Hello world scripts",
   "content": "A list of templates with hello world scripts",
   "userId": 2,
   "isHidden": false
}
```

/api/blogpost/update - PUT

- Update a blog post’s title or content or tags or templates
- Request body:
    - blogpostId
    - title: new title
    - content: new content
    - tagsAdded: tags to be added
    - tagsRemoved: tags to be removed
    - templatesAdded: templates to be added
    - templatesRemoved: templates to be removed
- Example Request:
```
{
   "blogpostId": 1,
   "title": "scripts",
   "content": "nothing"
}
```
- Example Response:
```
{
   "id": 1,
   "title": "SCRIPTS",
   "content": "nothing",
   "userId": 2,
   "isHidden": false
}
```

/api/blogpost/vote - PUT

- Upvote or downvote an entire blog post.
- Request body:
  - blogpostId
  - type: upvote or downvote
- Example Request:

```
{
   "blogpostId": 1,
   "type": "upvote"
}
```

- Example Response

```
{
   "id": 1,
   "content": "nothing",
   "userId": 2,
   "votes": [
       {
           "id": 3,
           "userId": 3,
           "type": true,
           "blogpostId": 1,
           "commentId": null
       },
   ]
}
```

/api/blogpost/delete - DELETE

- Delete an entire blog post.
- Request body:
  - blogpostId
- Example Request:

```
{
   "blogpostId": 2
}
```

- Example Response

```
{
   "message": "Blogpost deleted."
}
```

Comments API:

Method	API	Explanation
POST	/api/comments/create	Create a comment under a blog post.
GET	/api/comments/[id]	Fetch the comment by id.
DELETE	/api/comments/delete	Delete a comment.
POST	/api/comments/reply	Reply to a comment.
PUT	/api/comments/edit	Edit a comment.
PUT	/api/comments/report	Report an existing comment.
PUT	/api/comments/vote	Upvote or downvote a comment.

Comments API
/api/comments/create - POST

- Create a comment under a blog post
- Request body:
  - blogpostId: the blog post id
  - text: the text of the comment
- Example Request:

```
{
   "blogpostId": 1,
   "text": "Great blog post !!"
}
```

- Example Response

```
{
   "id": 1,
   "text": "Great blog post !!",
   "userId": 3,
   "blogpostId": 1,
   "isHidden": false,
   "parentId": null
}
```

/api/comments/[id] - GET

- Get a comment by id
- Request body: None
- Example Request: /api/comments/4
- Example Response:

```
{
   "id": 4,
   "text": "Thank you!",
   "userId": 2,
   "blogpostId": 1,
   "isHidden": false,
   "parentId": 1
}
```

/api/comments/delete - DELETE

- Delete a comment. Either soft deletes if it has replies, or hard deletes if it has no replies
- Request body:
    - commentId: comment to delete
- Example Request

```
{
    "commentId": 3
}
```

- Example Response

```
{
    "message": "Comment deleted."
}
```

/api/comments/reply - POST

- Reply to a comment in a thread.
- Request Body:
    - commentId: comment to reply to
    - text: of the comment
- Example Request

```
{
    "commentId": 1,
    "text": "Thank you!"
}
```

- Example Response

```
{
    "id": 4,
    "text": "Thank you!",
    "userId": 2,
    "blogpostId": 1,
    "isHidden": false,
    "parentId": 1
}
```

/api/comments/edit - PUT

- Edit an existing comment. Must be author's comment to edit
- Request Body:
    - commentId: comment to reply to

    - text: of the comment
- Example Request:

```
{
   "commentId": 1,
   "text": "It is amazing."
}
```

- Example Response:

```
{
   "id": 1,
   "text": "It is amazing.",
   "userId": 3,
   "blogpostId": 1,
   "isHidden": false,
   "parentId": null
}
```

/api/comments/report - PUT

- Report an existing comment.
- Request Body:
    - commentId: comment to report
    - explanation: explanation of report
- Example Request:

```
{
   "commentId": 3,
   "explanation": "The deleted to be deleted is rude."
}
```

- Example Response:

```
{
   "id": 3,
   "text": "Will be deleted.",
   "userId": 2,
   "blogpostId": 1,
   "isHidden": true,
   "parentId": null
}
```

/api/comments/vote - PUT

- Upvote or downvote a comment
- Request body:
    - commentId: comment to vote on
    - type: upvote or downvote
- Example Request:

```
{
   "commentId": 1,
   "type": "upvote"
}
```

- Example Response:

```
{
   "id": 1,
   "text": "It is amazing.",
   "userId": 3,
   "blogpostId": 1,
   "parentId": null,
   "votes": [
       {
           "id": 1,
           "userId": 2,
           "type": true,
           "blogpostId": 1,
           "commentId": 1
       }
   ]
}
```