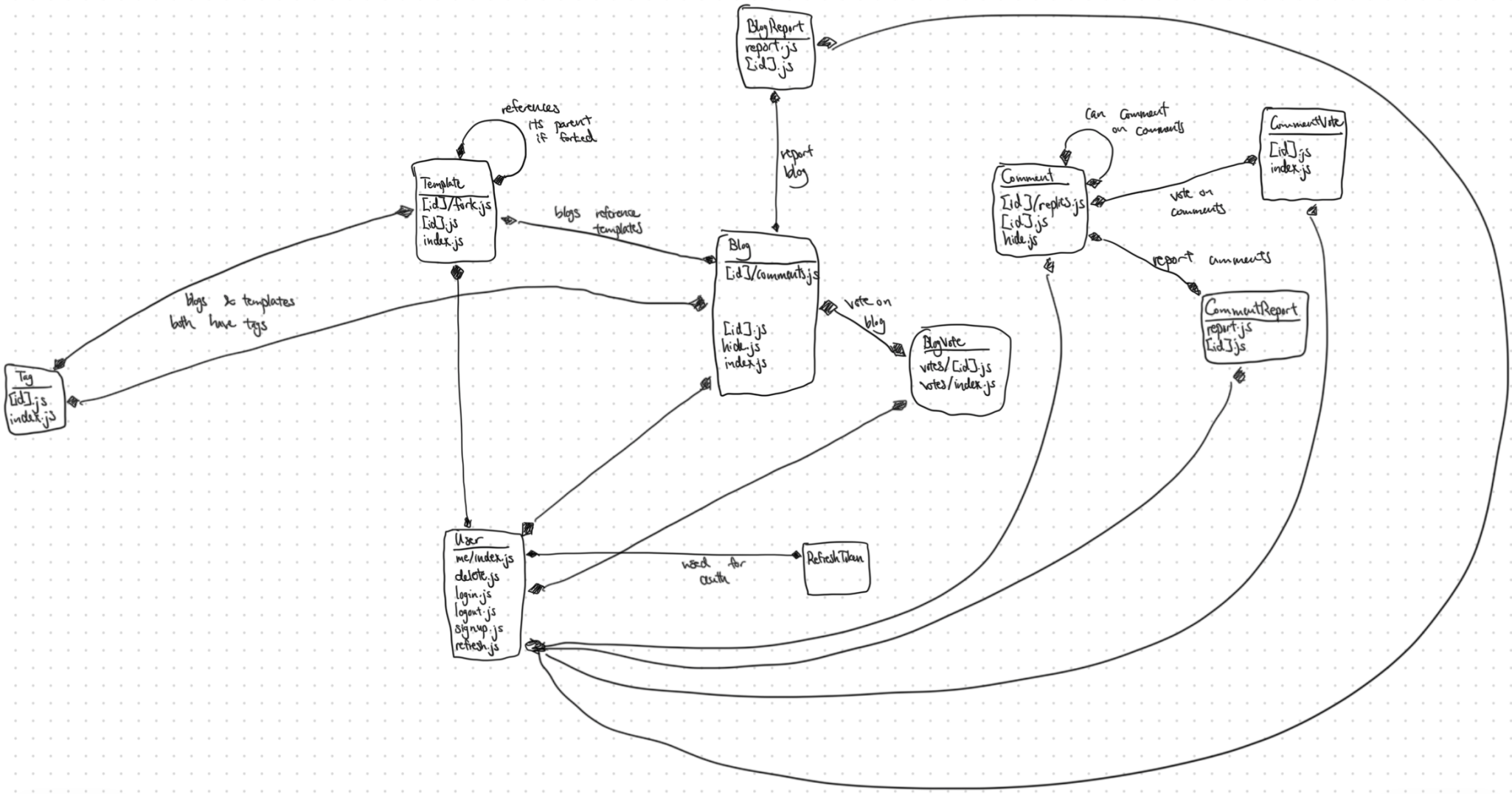
BlogReport
report.js
[id].js
references its parent if forked
Template
[id]/fork.js
[id].js
index.js
report blog
blogs reference templates
Blog
[id]/comments.js
[id].js
hide.js
index.js
vote on blog
BlogVote
votes/[id].js
votes/index.js
blogs & templates both have tags
Tag
[id].js
index.js
Can Comment on comments
Comment
[id]/replies.js
[id].js
hide.js
CommentVote
[id].js
index.js
vote on comments
report comments
CommentReport
report.js
[id].js
User
me/index.js
delete.js
login.js
logout.js
signup.js
refresh.js
need for auth
RefreshToken

```prisma
model User {
  id Int @id @default(autoincrement())
  username String @unique
  firstName   String
  lastName    String
  email       String @unique
  password String // Hashed

  // Optional
  avatar      String?
  phoneNumber String?
  role String @default("USER")

  // Empty upon creation
  createdAt DateTime @default(now())
  templates Template[]
  refreshTokens RefreshToken[]
  blogs Blog[]
  comments Comment[]
  commentVotes CommentVote[]
  blogVotes BlogVote[]
  blogReports BlogReport[]
  commentReports CommentReport[]
}

model RefreshToken {
  id Int @id @default(autoincrement())
  token String @unique
  userId Int
  user User @relation(fields: [userId], references: [id], onDelete: Cascade, onUpdate: Cascade)
  expiresAt DateTime
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

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

```
  title String
  @@unique([authorId, title])
  explanation String
  code        String
  language    String // must enforce types C, CPP, JAVA, PYTHON, JAVASCRIPT
  author      User     @relation(fields: [authorId], references: [id], onDelete: Cascade, onUpdate: Cascade)
  authorId    Int
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt
  blogs       Blog[]

  tags        Tag[]

  hidden Boolean @default(false)

  // Fork
  forkedCopies Template[] @relation("ForkedTemplates")
  forkedFromId Int?
  forkedFrom Template? @relation("ForkedTemplates", fields: [forkedFromId], references: [id], onDelete: Cascade,
onUpdate: Cascade)
}

model Tag {
  id          Int @id @default(autoincrement())
  name        String  @unique

  templates    Template[]

  blogs       Blog[]
}

model Blog {
  id Int @id @default(autoincrement())
  title String
  description String
  content String
  tags Tag[]
```

```
  authorId Int
  author User @relation(fields: [authorId], references: [id], onDelete: Cascade, onUpdate: Cascade)
  templates Template[]
  comments Comment[]
  blogReports BlogReport[]
  hidden Boolean @default(false)

  // Upvotes
  upvotes BlogVote[]
}

model Comment {
  id Int @id @default(autoincrement())
  content String
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt

  blog Blog @relation(fields: [blogId], references: [id], onDelete: Cascade, onUpdate: Cascade)
  blogId Int

  attachment String?

  // Author
  author User @relation(fields: [authorId], references: [id], onDelete: Cascade, onUpdate: Cascade)
  authorId Int

  parent Comment? @relation("CommentReplies", fields: [parentId], references: [id], onDelete: Cascade, onUpdate:
Cascade)
  parentId Int?

  // Replies
  replies Comment[] @relation("CommentReplies")
  votes CommentVote[]

  // Reports
  reports CommentReport[]
```

```
  // Rating
  rating Float // Calculated by upvotes - downvotes. Should be automated in vote registration.

  // Hidden
  hidden Boolean @default(false)
}

model BlogVote {
  upvoteId Int @id @default(autoincrement())
  upVote Boolean // if false, then downVote
  blogId Int
  blogIdConstraint Blog @relation(fields: [blogId], references: [id], onDelete: Cascade, onUpdate: Cascade)

  // Author
  by Int
  byUser User @relation(fields: [by], references: [id], onDelete: Cascade, onUpdate: Cascade)
}

model CommentVote {
  upvoteId Int @id @default(autoincrement())
  upVote Boolean // if false, then downVote
  commentId Int
  commentIdConstraint Comment @relation(fields: [commentId], references: [id], onDelete: Cascade, onUpdate:
Cascade)

  // Author
  by Int
  byUser User @relation(fields: [by], references: [id], onDelete: Cascade, onUpdate: Cascade)
}

model BlogReport {
  blogReportId Int @id @default(autoincrement())

  blogId Int
  blogIdConstraint Blog @relation(fields: [blogId], references: [id], onDelete: Cascade, onUpdate: Cascade)
  description String
  screenshots String?
```

```
  resolved Boolean @default(false)

  // Author
  byUser Int
  byUserConstraint User @relation(fields: [byUser], references: [id], onDelete: Cascade, onUpdate: Cascade)
}

model CommentReport {
  commentReportId Int @id @default(autoincrement())

  commentId Int
  commentIdConstraint Comment @relation(fields: [commentId], references: [id], onDelete: Cascade, onUpdate:
Cascade)
  description String
  screenshots String?

  resolved Boolean @default(false)

  // Author
  byUser Int
  byUserConstraint User @relation(fields: [byUser], references: [id], onDelete: Cascade, onUpdate: Cascade)
}
```

documentation can be found online at https://
documenter.getpostman.com/view/39442053/2sAY4xBhLb

CSC309 Backend

users

basic-user-operation

POST signup as a user

http://localhost:3000/api/user/signup

Allows users to sign up. Users will automatically have the role USER, as there is only one ADMIN user in the system, which is created during setup.

User is then automatically logged in by being given an access token.

Body raw (json)

json

```
{
    "username": "alice",
    "password": "Abcdefgh!1",
    "firstName": "alice",
    "lastName": "au",
    "email": "alice@test.com"
}
```

POST login as a user

http://localhost:3000/api/user/login

Logs a user in. Can log in using a `username` or an `email`. Responds with access tokens

Logs a user in. Can log in using a `username` or an `email`. Responds with access tokens.

Body raw (json)

```
json

{
    "username": "alice",
    "password": "Abcdefgh!1"
}
```

POST login as an admin

http://localhost:3000/api/user/login

Same behaviour as above.

Used for signing in admin user.

Body raw (json)

```
json

{
    "username": "admin",
    "password": "admin"
}
```

POST logout as a user

http://localhost:3000/api/user/logout

Logs out user by deleting refresh token. The token ensures that the user's session is invalidated, enhancing security by preventing unauthorized access post-logout.

HEADERS

Authorization	{{accessToken}}

Body raw (json)

```
json

{
    "refreshToken": "{{refreshToken}}"
}
```

POST refresh authentication token

http://localhost:3000/api/user/refresh

Used to exchange a valid refresh token for a new access token, ensuring continued authentication for the user without requiring them to log in again.

HEADERS

Authorization	{{accessToken}}

Body raw (json)

```
json

{
    "refreshToken": "{{refreshToken}}"
}
```

---

me

GET get own user info

http://localhost:3000/api/user/me

Retrieves details about the currently authenticated user.

HEADERS

HEADERS

Authorization	{{accessToken}}

PUT update own user info

http://localhost:3000/api/user/me

Updates profile for the currently authenticated user.

HEADERS

Authorization	{{accessToken}}

Body raw (json)

json

```
{
    "firstName": "eren",
    "lastName": "jaeger",
    "email": "aot@iseyama.com",
    "password": "Abcdefgh!1",
    "newPassword": "LeviJeans123!!!!",
    "avatar": "image.jpg",
    "phoneNumber": "62341231"
}
```

DELETE delete account as a user

http://localhost:3000/api/user/delete

Deletes the currently authenticated user's account.

HEADERS

Authorization	{{accessToken}}

Body raw (json)

```
json

{
    "password": "Abcdefgh!1"
}
```

templates

POST create a template as a user

http://localhost:3000/api/templates

Posts template with `id` with the following fields:
`title`: the title of the template

`explanation`: a description of the template's contents

`code`: the actual code in the template

`language`: programming language of the code

HEADERS

Authorization	{{accessToken}}

Body raw (json)

```
json

{
    "title": "csc309 week 2",
    "explanation": "javascript intro",
    "code": "console.log(\"hi\");",
    "language": "javascript"
}
```

GET search templates as a visitor/user

http://localhost:3000/api/templates?page=1&pageSize=5

Retrieves a paginated list of templates based on specified filters and criteria:

`page`: Page number of results.

`pageSize`: Number of items per page.

`search`: Keyword for filtering.

`tags`: Comma-separated tags.

`authorId`: ID of the author.

`language`: Programming language,

HEADERS

Authorization	{{accessToken}}

PARAMS

page	1
pageSize	5
search	csc309
authorId	1
language	javascript

GET get single template as user/visitor

http://localhost:3000/api/templates/1

Retrieve data for a single template.

POST fork a template as user

http://localhost:3000/api/templates/3/fork

Creates a copy of another template.

HEADERS

Authorization	{{accessToken}}

PARAMS

Body raw (json)

```json
{
    "title": "forked template 2"
}
```

PUT update a template as user

http://localhost:3000/api/templates/3

Updates template with `id` for the following optional fields:
`title`: the title of the template

`explanation`: a description of the template's contents

`code`: the actual code in the template

`tags`: the tags associated with the template

`language`: programming language of the code

HEADERS

Authorization	{{accessToken}}

Body raw (json)

json

```
{
    "title": "csc309 week 2",
    "explanation": "javascript intro",
    "code": "console.log(\"hi\");",
    "tags": "simple,educational",
    "language": "javascript"
}
```

DELETE delete template as user

http://localhost:3000/api/templates/2

Deletes template if the authenticated user is the author.

HEADERS

Authorization	{{accessToken}}

---

reports

---

commentReports

GET get a commentReport as user/admin

http://localhost:3000/api/reports/commentReports/3

Gets comment report with `id`. Only visible to author of the comment and admin.

PUT update a commentReport as admin

http://localhost:3000/api/reports/commentReports/3

Updates comment report with `id` with the following optional fields. Fails if user is does not have role `ADMIN`

Updates comment report with `id` with the following optional fields. Fails if user is does not have role `ADMIN`.

`description`

`screenshots`

`resolved`

Body raw (json)

```
json

{
    "description": "bad comment",
    "screenshots": "image.png",
    "resolved": true
}
```

DELETE delete a commentReport as admin.

http://localhost:3000/api/reports/commentReports/3

Deletes a comment report with `id`. Will fail if user does not have role `ADMIN`.

---

blogReports

GET get a blogReport

http://localhost:3000/api/reports/blogReports/1?

Gets blog report with `id`.

HEADERS

Authorization	{{accessToken}}

PARAMS

PUT update a blogReport

http://localhost:3000/api/reports/blogReports/1

Updates blog report with `id` with the following optional fields:
`description`

`screenshots`

`resolved`

HEADERS

Authorization	{{accessToken}}

Body raw (json)

json

```
{
    "description": "bad comment",
    "screenshots": "image.png",
    "resolved": true
}
```

DELETE delete a blogReport

http://localhost:3000/api/reports/blogReports/2

Deletes a comment report with `id`. Will fail if user is not the author of the report.

HEADERS

Authorization	{{accessToken}}

blogs

blogs

votes

POST register a blog vote

http://localhost:3000/api/blogs/votes/

Adds a vote to blog with `blogId`. Upvotes if `upVote` is `true`, downvotes if `upVote` is `false`.

HEADERS

Authorization	{{accessToken}}

Body raw (json)

json

```
{
    "upVote": false,
    "blogId": 1
}
```

report

POST report a blog

http://localhost:3000/api/blogs/report

Reports blog with `blogId` with the following fields:
`description`

`screenshots` (optional)

HEADERS

Authorization	{{accessToken}}

Body raw (json)

json

```
{
    "blogId": "1",
    "description": "bad comment",
    "screenshots": "image.png"
}
```

GET search blogReports

http://localhost:3000/api/blogs/report?page=1&pageSize=5&search=bad

Retrieves a paginated list of blogReports based on specified filters and criteria:

`page`: Page number of results.

`pageSize`: Number of items per page.

`search`: Keyword for filtering.

HEADERS

Authorization	{{accessToken}}

PARAMS

page	1
pageSize	5
search	bad

POST create a blog

http://localhost:3000/api/blogs

Register a blog with the following fields:

`title`

`description`

`content`

`tags` (optional)

HEADERS

Authorization	{{accessToken}}

Body raw (json)

```
json

{
    "title": "cool blog about stuff",
    "description": "description about stuff",
    "content": "yap yap yap",
    "tags": "cool, funny, awesome",
    "templateIds": "1"
}
```

GET list blogs with filtering

http://localhost:3000/api/blogs?page=1&pageSize=5&search=funny&template=3

Retrieves a paginated list of blogs based on specified filters and criteria:

`page`: Page number of results.

`pageSize`: Number of items per page.

`search`: Keyword for filtering.

`tags`: Comma-separated tags.

HEADERS

Authorization	{{accessToken}}

PARAMS

page	1
pageSize	5
search	funny
tags	funny
template	3

GET get a blog

http://localhost:3000/api/blogs/1

Retrieves information about blog with `id`.

HEADERS

Authorization	{{accessToken}}

PUT update a blog

http://localhost:3000/api/blogs/1

Updates blog with `id` with the following optional fields:
`title content`

HEADERS

Authorization	{{accessToken}}

Body raw (json)

json

```
{
```

```
    "title": "blog about lame stuff",
    "content": "more yap"
}
```

DELETE delete a blog

http://localhost:3000/api/blogs/1

Deletes blog with blog `id`. Fails if user is not the author of the blog.

HEADERS

Authorization	{{accessToken}}

PUT hide a blog as admin

http://localhost:3000/api/blogs/hide

Hides blog with `id`. Will fail if user does not have role `ADMIN`.

Body raw (json)

json

```
{
    "id": "2"
}
```

---

comments

---

top-level

POST create top-level comment

http://localhost:3000/api/blogs/1/comments

HEADERS

Authorization	{{accessToken}}

Body raw (json)

```
json

{
    "content": "this post sucks"
}
```

GET get all top-level comments on a post

http://localhost:3000/api/blogs/1/comments?page=1&pageSize=1

PARAMS

page	1
pageSize	1

---

replies

POST create a reply

http://localhost:3000/api/comments/1/replies

HEADERS

Authorization	{{accessToken}}

Body raw (json)

Body raw (json)

```
json

{
    "content": "no it doesn't"
}
```

GET retrieve paginated comments as a user/visitor

http://localhost:3000/api/comments/1/replies

votes

POST register a comment vote

http://localhost:3000/api/comments/votes/

Adds a vote to blog with `blogId`. Upvotes if `upVote` is `true`, downvotes if `upVote` is `false`.

HEADERS

Authorization	{{accessToken}}

Body raw (json)

```
json

{
    "upVote": false,
    "commentId": 1
}
```

GET get a comment

http://localhost:3000/api/comments/3

Retrieves information on comment with `id`.

PUT update a comment

http://localhost:3000/api/comments/3

Updates comment with `id` with the following optional fields:
`content`

`attachment`

Body raw (json)

json

```
{
    "content": "cool code",
    "attachment": "image.png"
}
```

DELETE delete a comment

http://localhost:3000/api/comments/3

Deletes comment with `id`. Will fail if user is not the author of the comment.

PUT hide a comment as admin

http://localhost:3000/api/comments/hide

Hides comment with `id`. Fails if user does not have role `ADMIN`.

HEADERS

Authorization	{{accessToken}}

Body raw (json)

```
json

{
    "id": "3"
}
```

POST report a comment

http://localhost:3000/api/comments/report.js

Posts a report for a comment with `commentId` with the following fields:

`description`

`screenshots` (optional)

Body raw (json)

```
json

{
    "commentId": "3",
    "description": "bad comment",
    "screenshots": "image.png"
}
```

GET New Request

POST execute code

http://localhost:3000/api/execute

Executes `code` in `language` with `stdin` as arguments.

Body raw (json)

json

```
{
    "language": "python",
    "code": "print(input())",
    "stdin": "Hello World"
}
```