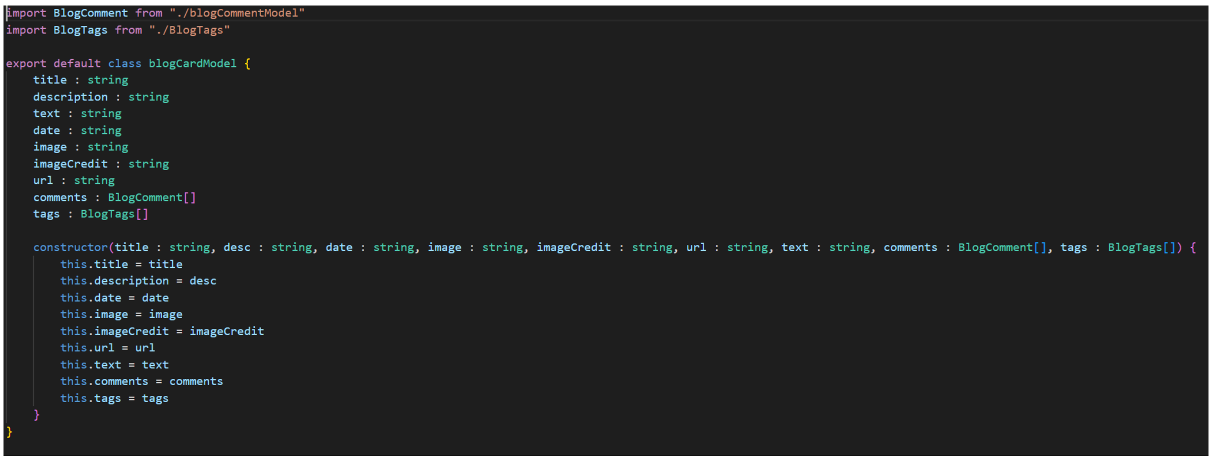
How are posts and comments stored?

Everything is saved as instance variables in the class blogCardDB.tsx, in the class' array "cards". This is not the cleanest solution. A better option would be saving them in JSON format or in a real database. However, for this use case, this implementation is good enough. It is used because it the easiest to implement

How do I create blog posts?

The structure of blog posts is defined by the class blogCardModel in (blogCardModel)/blogCardModel.tsx. The class is very simple and only contains instance variables and a constructor, see Figure 1.



```
import BlogComment from "./blogCommentModel"
import BlogTags from "./BlogTags"

export default class blogCardModel {
    title : string
    description : string
    text : string
    date : string
    image : string
    imageCredit : string
    url : string
    comments : BlogComment[]
    tags : BlogTags[]

    constructor(title : string, desc : string, date : string, image : string, imageCredit : string, url : string, text : string, comments : BlogComment[], tags : BlogTags[]) {
        this.title = title
        this.description = desc
        this.date = date
        this.image = image
        this.imageCredit = imageCredit
        this.url = url
        this.text = text
        this.comments = comments
        this.tags = tags
    }
}
```

Figure 1: the class blogCardModel which defines the structure of blog posts

Table 1 contains explanations of all the fields in the above class and explanations on their purpose.

Variable name	Usage
title	The title of the post. Used both on the main page of the blog and on the page of the blog post.
description	The description of the post. Used only on the main page to summarize what the blog post is about.
text	The main text of the post. Used only on the page of the post. The text allows for HTML elements to be written directly in it. For more details see furhter down. To make a new line, use "\r\n". This means for a new paragraph, use "\r\n\r\n".
date	The date that the post was made. Does not have to follow any particular structure, but using "XXth of MONTH XXXX" is recommended. Used both on the main page and page of post.
image	URL to image associated with the post. Can either be a locally saved image or an image from the web. To link to an online image copy and

	paste the image address. To use a local image, save the image in the "blog/public" directory and refer to it here as "/[FILE NAME INCLUDING FORMAT]". The same image will be used on the main page and the page of the post. Images of any size and aspect ratio can be used but landscape 16:9 pictures tend to look the best.
imageCredit	Credit for image creator. Used if image is for example CC-Attribution. Will be displayed on both main page and page of the blog. Text will show after hard-coded text "Image credit: " (imageCredit = "Foo Bar" => Full text will be "Image credit: Foo Bar). Pass empty string "" to have no credit text.
url	The URL of the post. For example, url = "ab123" will make the URL of the post [blog.com]/blogentry/ab123. The url variable can be anything, but use of five alphanumeric symbols is recommended to create cohesion.
comments	An array with the comments on the post. An empty array ([]) means the post has no comments. For more information on comments, see "How do I create comments?".
tags	An array containing the tags of the post. Must be of enum type BlogTags defined in BlogTags.ts. The tags will be visible under the title on both the main page and the page of the post. Tags are used to filter posts on the main page. An empty array ([]) means the post has not been tagged. For more information on tags, see "How do tags work?".

Table 1: explanations of variables in blogCardModel

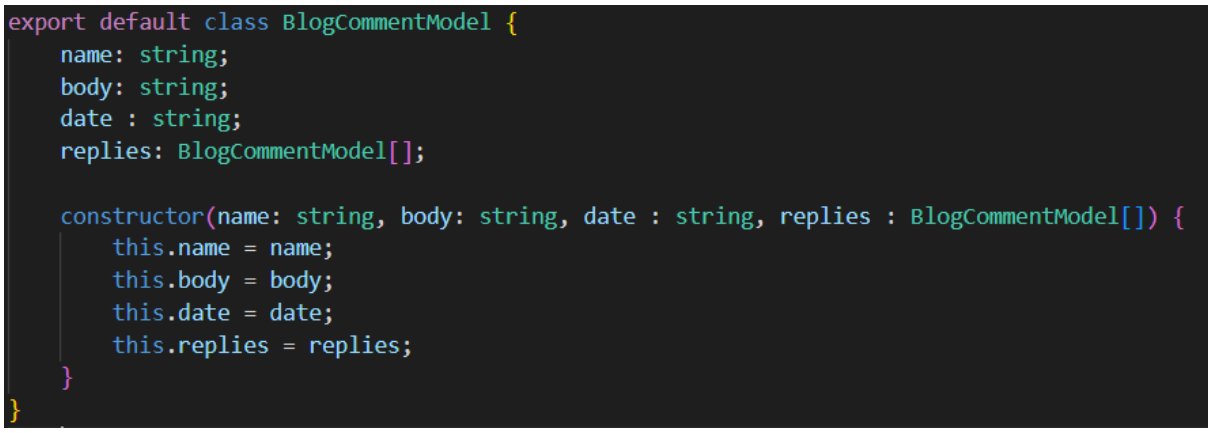
All normal blog posts are in the list "cards" in blogCardDB.tsx. To create a post you need to enter it into the code in that array. This is done the same way as creating any JavaScript object; using the "new" keyword and entering the desired arguments of the post in the constructor. All posts in the "cards" array will be visible on the main page of the blog in the order they appear in the code.

To create a post that does not appear on the main page, create a folder in /blogentry. Name the folder according to what URL you want the post to have. For example, a folder called "xy789" will result in the post being on the URL [blog.com]/blogentry/xy789. In the folder, create a file called *exactly* page.tsx. In file, create the React component that will represent the blog post. To use the same structure as other posts you can copy the code from blogentry/[entries]/page.tsx and replace all the occurrences of {blogEntry.XXXXX} with the desired value.

The text variable allows HTML elements in it. This is mainly meant to be used for creating links, but can be used in other ways as well. For example, if the text variable is "Click <a href='[URL]'> here </a>", then the HTML anchor tags will be interpreted and the text on the blog post will only show "Click here", with the word "here" being a link to [URL]. Note that the URL needs to be absolute, i.e. have the full https:// part.

How do I create comments?

Comments are defined by the BlogCommentModel class in (blogCardModel)/blogCommentModel.tsx. The class is very simple and only contains instance variables and a constructor, see Figure 2.



```
export default class BlogCommentModel {
    name: string;
    body: string;
    date : string;
    replies: BlogCommentModel[];

    constructor(name: string, body: string, date : string, replies : BlogCommentModel[]) {
        this.name = name;
        this.body = body;
        this.date = date;
        this.replies = replies;
    }
}
```

Figure 2: the BlogCommentModel class which defines comments.

Table 2 contains explanations for all the variables and their purpose:

Name of variable	Usage
name	The name of the user that made the comment.
body	The body of the comment
date	The date the comment was made. Does not have to follow any particular structure, but using “XXth of MONTH XXXX” is recommended.
replies	An array with all the replies to this comment.

Table 2: explanations of the variables in blogCommentModel

A comment can either be a comment directly on a post or a reply to another comment.

To add a comment in response to a post, add the comment to the post's “comments” array. The easiest way to do this is to create a new BlogCommentModel with "new" directly in the post's array "comments".

För att lägga till en kommentar som svar på en annan kommentar, lägg till kommentaren i den svarade kommentarens array “replies”. Detta kan göras på olika sätt, men görs kanske enklast genom att skapa en ny BlogCommentModel med “new” direkt i den svarade kommentarens array “replies”. Detta kan göras direkt i ett inläggs comments-array. Struktur kan bli lite förvirrande, det blir lättare om comments delas upp på flera rader.

To add a comment as a reply to another comment, add the comment to the replied comment's “replies” array. The easiest way to do this is to create a new BlogCommentModel with "new" directly in the replied comment's array "replies". This can be done directly in a post's comments array. The structure can get a little confusing, so it is recommended to add each reply to a separate line of code.

There is no limit for how many “layers” of replies a comment can have, meaning there can theoretically be an infinite chain of replies to replies. However, after a few replies the comments become hard to read, so it is recommended to not have more than four or five layers of replies.

How do tags work?

Each post can have a set of tags. It is not possible to use arbitrary tags. The tags that are available are those defined in enum BlogTags in BlogTags.ts. To create a new tag, add a new line to the enum.

To add a tag to a post, add the tag to the post's “tags” array. To do that, use BlogTags.[NAME OF TAG]. A post can have any number of tags, but there is no point in adding the same tag multiple times.

On the main page it is possible to filter on different tags. By clicking on a tag at the top, you filter on that tag, so only posts that have that tag show up. This works in conjunction with the regular search function. If multiple tags are clicked in, AND/OR search is used. This means that if, for example, the tags Mystery and Personal have been clicked on, all posts that have been tagged with Mystery AND/OR Personal will be displayed.

How do I edit posts and comments?

This can be done directly in the code for each post or comment. Go to blogCardDB.tsx and look up the post comment you want to edit, then simply change the value you wish to edit. Because most values are saved as simple strings it is very easy to edit most values to be any text you want.

Important to consider

A few important things to consider before launching the ARG.

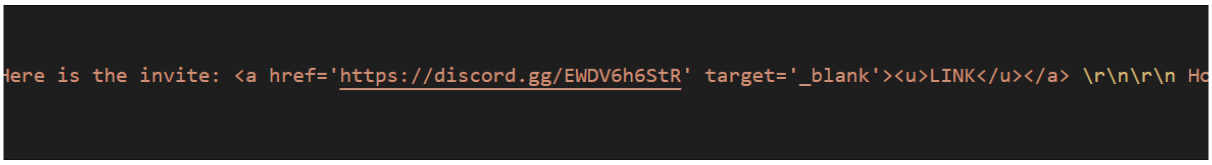
Dates

As of writing the dates on the website are configured as if the game is meant to be played during spring of 2024. If this is not the case when launching the game, make sure to edit the dates. The easiest way to do this in a way that is consistent with the story is to add whole years.

Discord invite

In the latest post on the blog, called “Please help me find Blue!”, there is an invite link to a Discord. As of writing, this invite leads to a dummy Discord server that should not be used for the actual game. Before launching the game, make sure to update the invite link in that post. To do this, follow these steps:

1. Create a new Discord server
   a. Configure the server in any way you see fit. It is recommended to make it a community server. Other than that, you can create any amount of channels or roles depending on what you want, but there is no need to have the server be anything else than the default.
2. Make sure the Puppet Master is in the Discord and is called “Red” or “Arvid” or “ArvidLovskog”, or some other name connecting to Arvid “Red” Lövskog who is the Puppet Master’s character. It is a good idea to create a completely new account for this purpose.
3. Create a permanent and unlimited Discord invite link.
   a. More details: https://support.discord.com/hc/en-us/articles/208866998-Invites-101
4. Copy the link and paste it where the current link is, meaning inside the quotes after href= in the “Please help me find Blue!” post. It is supposed to look like the picture below.



```
Here is the invite: <a href='https://discord.gg/EWDV6h6StR' target='_blank'><u>LINK</u></a> \r\n\r\n Ho
```