

Forms, REST og CRUD

REST - Update og delete

Dagens Formål

- Viden om hvordan vi opdaterer og sletter eksisterende data (objekter) i databasen via REST API'et.
- Derudover hvordan vi skaber overblik og “ro” i de seneste dages “kaos” af begreber, koncepter og kodeblokke.

Agenda

The screenshot shows a code editor window with a menu bar at the top. The menu items include View, Selection, Find, Debugger, Packages, Window, Help, and More. Below the menu, there is a toolbar with icons for selection, find, debugger, packages, window, help, and more. The main area of the editor displays a file named 'main.js' with the following content:

```
main.js
1 "use strict";
2
3 fetch("http://headlesscms.cederdorff.com/wp-json/wp/v2/posts?_embed")
4   .then(function(response) {
5     return response.json();
6   })
7   .then(function(json) {
8     appendPosts(json);
9   });
10
11 function appendPosts(posts) {
12   for (let post of posts) {
13     console.log(post);
14     document.querySelector("#grid-posts").innerHTML += `
15       <article>
16         <h3>${post.title.rendered}</h3>
17         <p>Email: <a href="mailto:${post.acf.email}">${post.acf.email}</a></p>
18         <p>Phone: ${post.acf.phone}</p>
19       </article>
20     `;
21   }
22 }
```

1. Demo af Post App med GET, POST, PUT og DELETE

- Demo af Create, Read, Update og Delete
- Networktab og requests
- Gennemgå struktur og kode med slides
- Importer ny datastruktur til Firebase:
15. Import a New Data Structure

2. REST, CRUD og HTTP-metoder: PUT og DELETE

- Hvad er særligt for PUT og DELETE?
- REST og URL-struktur

3. Update og delete med Firebase Database REST API

- Anvendelse af Fetch, HTTP-metoden og request body med JSON

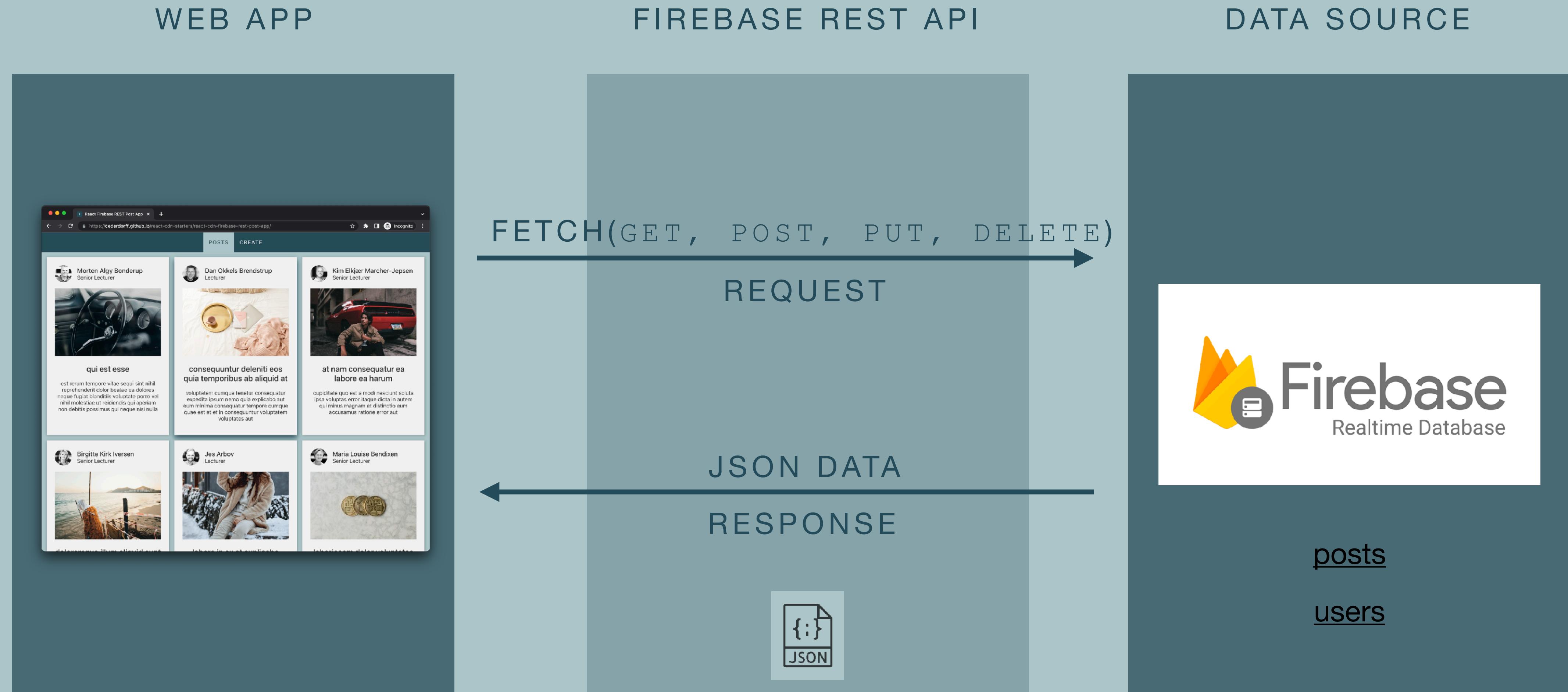
1. SEMESTERS STRUKTUR

Semesteret består af 16 undervisningsuger (inkl. en masse helligdage)



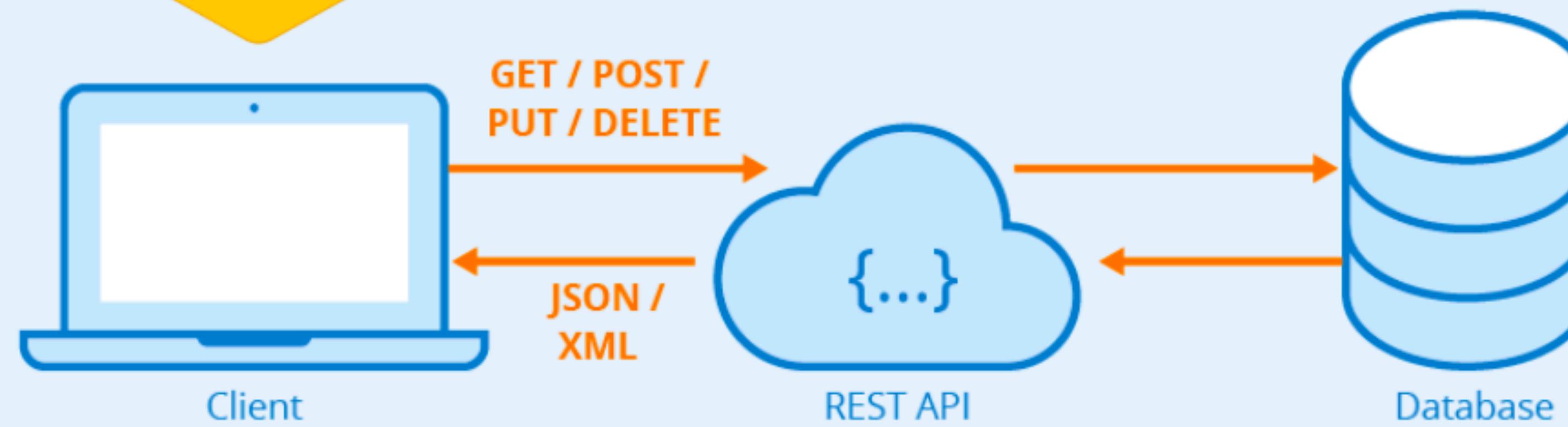
Eksamensperiode (12.-14. juni)

Fetch, HTTP Request & Response





Firebase



Fruter Post App 127.0.0.1:5500/index.html

Post App with Firebase REST API

CREATE NEW POST

Posts



Qui est esse

Est rerum tempore vitae sequi sint nihil reprehenderit dolor beatae ea dolores neque fugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis qui aperiam non debitis possimus qui neque nisi nulla

[DELETE](#) [UPDATE](#)



Consequuntur deleniti eos quia temporibus ab aliquid at

Voluptatem cumque tenetur consequatur expedita ipsum nemo quia explicabo aut eum minima consequatur tempore cumque quae est et et in consequuntur voluptatem voluptates aut

[DELETE](#) [UPDATE](#)



At nam consequatur ea labore ea harum

Cupiditate quo est a modi nesciunt soluta ipsa voluptas error itaque dicta in autem qui minus magnam et distinctio eum accusamus ratione error aut

[DELETE](#) [UPDATE](#)



Doloremque illum aliquid sunt Updated 🔥

Doloremque ex facilis sit sint culpa soluta assumenda eligendi non ut eius sequi ducimus vel quasi veritatis est dolores

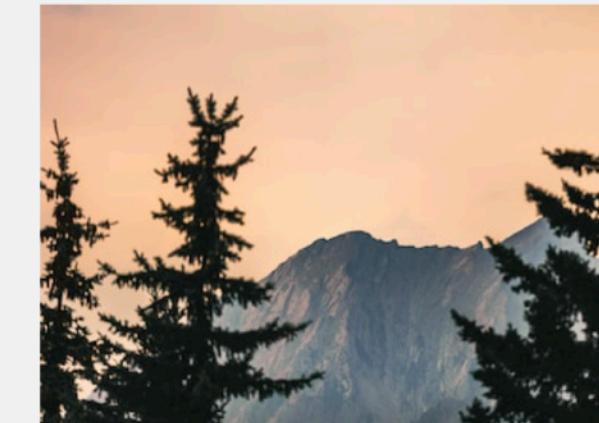
[DELETE](#) [UPDATE](#)



Labore in ex et explicabo corporis aut quas

Ex quod dolorem ea eum iure qui provident amet quia qui facere excepturi et repudiandae asperiores molestias provident minus incidunt vero fugit rerum sint sunt excepturi provident

[DELETE](#) [UPDATE](#)



```
app.js — post-app
JS app.js ×
JS app.js > ⚡ deletePost
1 "use strict";
2
3 const endpoint = "https://post-rest-api-default.firebaseio.com";
4
5 window.addEventListener("load", initApp);
6
7 function initApp() {
8     updatePostsGrid(); // update the grid of posts: get and show all posts
9     updateUsersGrid(); // update the grid of users: get and show all users
10
11    // event listener for create new post button
12    document.querySelector("#btn-create-post").addEventListener("click", createPostClicked);
13 }
14
15 // ===== events ===== //
16
17 > function createPostClicked() { ...
25 }
26
27 // ===== posts ===== //
28 > async function updatePostsGrid() { ...
32 }
33
34 // Get all posts - HTTP Method: GET
35 > async function getPosts() { ...
40 }
41
42 > function showPosts(listOfPosts) { ...
48 }
49
50 > function showPost(postObject) { ...
82 }
83
84 // Create a new post - HTTP Method: POST
85 > async function createPost(title, body, image) { ...
95 }
96
97 // Update an existing post - HTTP Method: PUT
98 > async function deletePost(id) [...
104 ]
```

1. initApp kalder updatePostsGrid

2. updatePostsGrid kalder showPosts. showPosts henter alle posts via et GET Request. showPosts laver objekt med objekter om til array med objekter og returnerer array tilbage til updatePostsGrid.

3. updatePostsGrid kalder showPosts. showPosts looper igennem alle posts og for hver post kaldes showPost.

4. showPost foretager DOM-Manipulation på baggrund af ét postobjekt, der gives som argument (fra loop i showPosts).

```
app.js — post-app
JS app.js > showPost
42  function showPosts(listOfPosts) {
43      document.querySelector("#posts").innerHTML = ""; // reset the content of section#posts
44
45      for (const post of listOfPosts) {
46          showPost(post); // for every post object in listOfPosts, call showPost
47      }
48  }
49
50 function showPost(postObject) {
51     const html = `/*html*/
52         <article class="grid-item">
53             
54             <h3>${postObject.title}</h3>
55             <p>${postObject.body}</p>
56             <div class="btns">
57                 <button class="btn-delete">Delete</button>
58                 <button class="btn-update">Update</button>
59             </div>
60         </article>
61     `; // html variable to hold generated html in backtick
62     document.querySelector("#posts").insertAdjacentHTML("beforeend", html); // append html to the DOM - section#posts
63
64     // add event listeners to .btn-delete and .btn-update
65     document.querySelector("#posts article:last-child .btn-delete").addEventListener("click", deleteClicked);
66     document.querySelector("#posts article:last-child .btn-update").addEventListener("click", updateClicked);
67
68     // called when delete button is clicked
69     function deleteClicked() {
70         deletePost(postObject.id); // calls deletePost with the id of the post as argument (parameter)
71     }
72
73     // called when update button is clicked
74     function updateClicked() {
75         const title = `${postObject.title} Updated 🔥`;
76         const body = "Doloremque ex facilis sit sint culpa soluta assumenda eligendi non ut eius sequi ducimus vel quas
77         const image =
78             "https://images.unsplash.com/photo-1465779171454-aa85ccf23be6?ixlib=rb-4.0.3&ixid=MnwxMjA3fDB8MHxzaW5mbwY2h8M
79         // call update post with "hard coded" values - tbd: values from a form
80         updatePost(postObject.id, title, body, image);
81     }
82 }
```

1. showPosts looper igennem alle posts og for hver post kaldes showPost med et post som argument (parameter).
2. deleteClicked event tilføjes Delete-knappen.
3. deletePost kaldes med postobjektets id.
4. updateClicked event tilføjes Update-knappen.
5. updatePost kaldes med de informationer der skal til for at opdatere et post's properties: id, title, body og image.

1. deletePost kaldes af deleteClicked.

The screenshot shows a code editor window with the file 'app.js' open. The code contains two main functions: 'deletePost' and 'updatePost'. The 'deletePost' function uses a DELETE request to remove a post from Firebase. The 'updatePost' function uses a PUT request to update a post in Firebase. Both functions include logging to the console and an update of the 'postsGrid' to reflect changes.

```
app.js — post-app
JS app.js ×
JS app.js > ...
96
97 // Update an existing post - HTTP Method: PUT
98 1 async function deletePost(id) {
99     const response = await fetch(`/${endpoint}/posts/${id}.json`, { method: "DELETE" });
100    if (response.ok) {
101        console.log("New post successfully deleted from Firebase 🚨");
102        updatePostsGrid(); // update the post grid to display all posts and the new post
103    }
104 }
105
106 // Delete an existing post - HTTP Method: DELETE
107 2 async function updatePost(id, title, body, image) {
108     const postToUpdate = { title, body, image }; // post update to update
109     const json = JSON.stringify(postToUpdate); // convert the JS object to JSON string
110     // PUT fetch request with JSON in the body. Calls the specific element in resource
111     const response = await fetch(`/${endpoint}/posts/${id}.json`, { method: "PUT", body: json });
112     // check if response is ok - if the response is successful
113
114     if (response.ok) {
115         console.log("Post successfully updated in Firebase 🚨");
116         updatePostsGrid(); // update the post grid to display all posts and the new post
117     }
118 }
```

Ln 41, Col 1 Spaces: 4 UTF-8 LF {} JavaScript ⚡ Port: 5500 ✓ Prettier

deletePost foretager et HTTP request med metode DELETE. Requestet foretages til det specifikke element, som skal slettes. Derfor id i url'en.

Hvis DELETE requestet er succesfuldt, kaldes updatePostsGrid, der sørger for at alle posts hentes på ny og vises i DOM'en.

2. updatePost kaldes af updateClicked.

updatePost laver et objekt med de værdier (props), der skal opdateres. Objektet konverteres til JSON og sendes med i body'en på et PUT Request.

Requestet foretages til det specifikke element, som skal opdateres. Derfor id i url'en.

Hvis PUT requestet er succesfuldt, kaldes updatePostsGrid, der sørger for at alle posts hentes på ny og vises i DOM'en.

1. createPostClicked event tilføjes Create-knappen, der ses i header.

createPostClicked genererer værdier, som burde komme fra en formular. Men for at vi kan træne, "hard-coder" og genererer vi nogle værdier.

2. createPost kaldes med de genererede værdier for title, body og image.

createPost laver et nyt postobjektet med title, body og image. Det konverteres til JSON. Dernæst foretages et HTTP POST request med JSON i requestets body.

Hvis POST requestet er succesfuldt, kaldes updatePostsGrid, der sørger for at alle posts hentes på ny og vises i DOM'en.

```
function initApp() {
    updatePostsGrid(); // update the grid of posts: get and show all posts
    updateUsersGrid(); // update the grid of users: get and show all users

    // event listener for create new post button
    document.querySelector("#btn-create-post").addEventListener("click", createPostClicked);
}

// ===== events ===== //
function createPostClicked() {
    const randomNumber = Math.floor(Math.random() * 100 + 1);
    const title = `My Post Title Number ${randomNumber}`;
    const body = "Quo deleniti praesentium dicta non quod aut est molestias molestias et offici
    const image =
        "https://plus.unsplash.com/premium_photo-1675330628475-b4e0e2a3c4a7?ixlib=rb-4.0.3&ixid
    // call createPost with "hard-coded" values - tbd: values from a form
    createPost(title, body, image);
}

// Create a new post - HTTP Method: POST
async function createPost(title, body, image) {
    const newPost = { title, body, image }; // create new post object
    const json = JSON.stringify(newPost); // convert the JS object to JSON string
    // POST fetch request with JSON in the body
    const response = await fetch(`$endpoint/posts.json`, { method: "POST", body: json });
    // check if response is ok - if the response is successful
    if (response.ok) {
        console.log("New post successfully added to Firebase 🔥");
        updatePostsGrid(); // update the post grid to display all posts and the new post
    }
}
```

The diagram illustrates the flow of the application logic. It shows two main steps: 1. The creation of a new post via the 'createPostClicked' function, which generates random data and calls the 'createPost' function. 2. The execution of the 'createPost' function, which performs an HTTP POST request to add a new post to Firebase.

Fetch, JSON, CRUD & REST

Slides:

<https://cederdorff.github.io/dat-js/slides/Fetch-JSON-CRUD-og-REST.pdf>