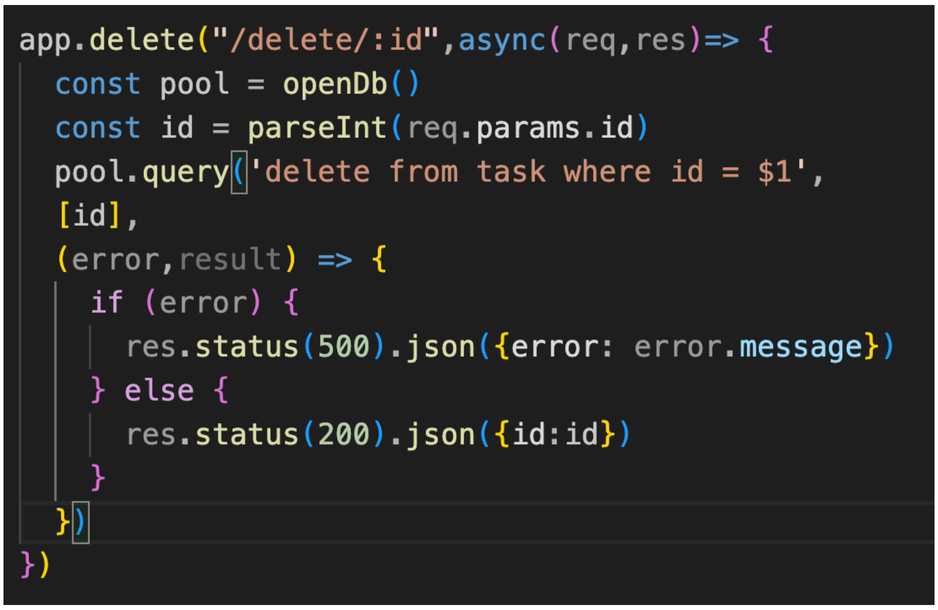
Exercise. Todo (part 5) – Deleting task

Delete feature will be added to the website. You need to both modify back-end and front-end to include this feature to your web app.

Key learnings for this part are:

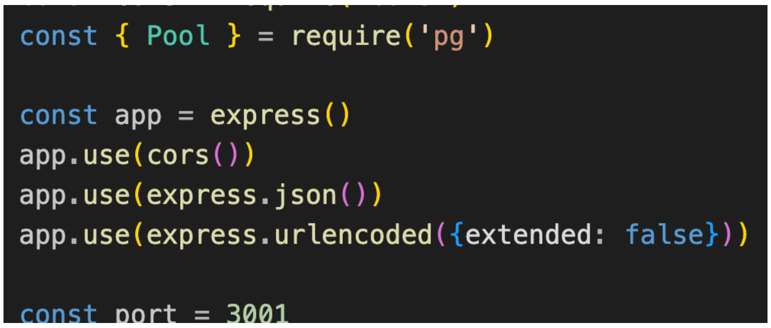
- Creating and testing endpoint for handling HTTP delete requests
- Dividing (further) UI logic into functions
- Using Bootstrap Icons
- Using data- attribute for storing data related to element in HTML document

Start by implementing deletion functionality to the backend. Create delete method, that receives id as query parameter (e.g., http://localhost:3001/delete/1).



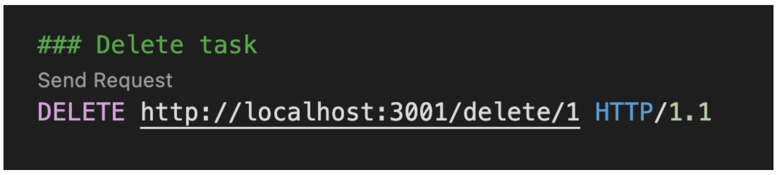
```
app.delete("/delete/:id",async(req,res)=> {
  const pool = openDb()
  const id = parseInt(req.params.id)
  pool.query('delete from task where id = $1',
  [id],
  (error,result) => {
    if (error) {
      res.status(500).json({error: error.message})
    } else {
      res.status(200).json({id:id})
    }
  })
})
```

Since id for the deleted task is passed as a part of url, you need to define, that express can read parameters from url address.



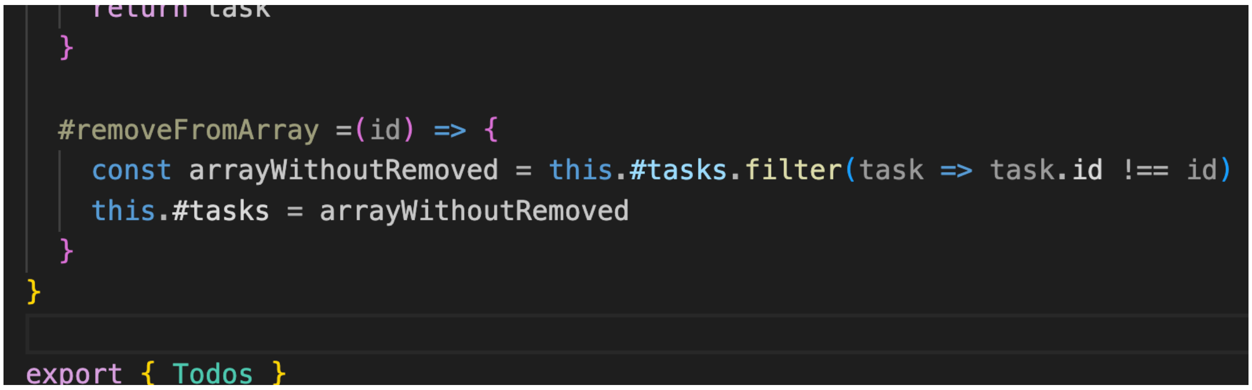
```
const { Pool } = require('pg')

const app = express()
app.use(cors())
app.use(express.json())
app.use(express.urlencoded({extended: false}))

const port = 3001
```

Test created endpoint with Rest Client by creating following script. HTTP method is delete and id (in this example 1) for the deleted record (task) on the database is part of url (HTTP query parameter).



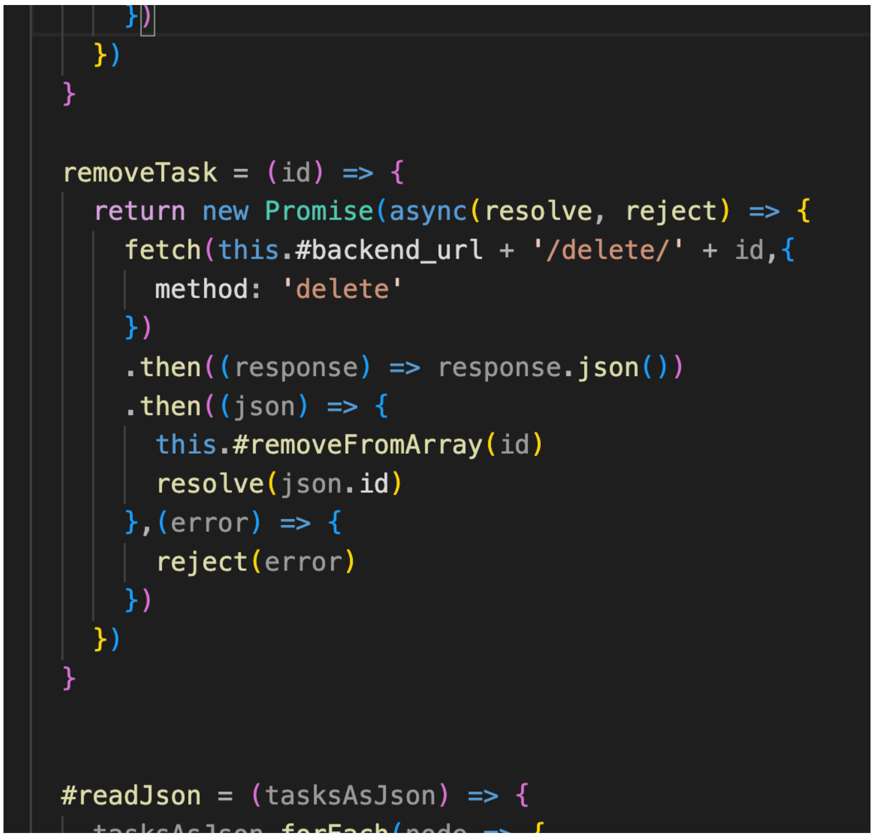
```
### Delete task
Send Request
DELETE http://localhost:3001/delete/1 HTTP/1.1
```

On client modify Todos class by adding new private method that removes task from array based on id. There are multiple ways how to remove item from array in JavaScript and here filter is used. First all tasks that are not removed are filtered and then assigned into member variable (array tasks).



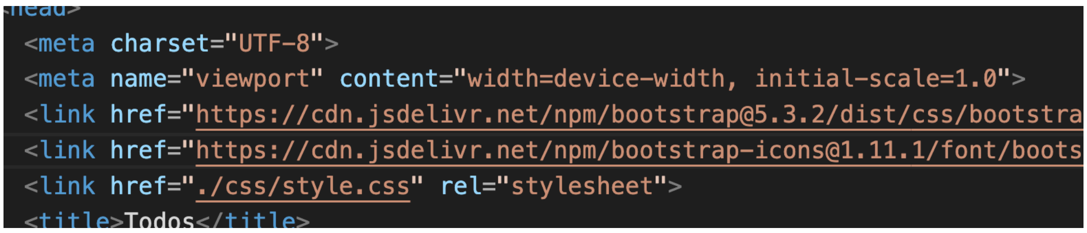
```
      return task
    }

    #removeFromArray =(id) => {
      const arrayWithoutRemoved = this.#tasks.filter(task => task.id !== id)
      this.#tasks = arrayWithoutRemoved
    }
}

export { Todos }
```

Add method for removing task (Todos class). Id will be passed as part of url and method is delete. Resolve will return id of deleted task (which is read from response but basically parameter id has the same value).



```
      })
    })
  }

  removeTask = (id) => {
    return new Promise(async(resolve, reject) => {
      fetch(this.#backend_url + '/delete/' + id,{
        method: 'delete'
      })
      .then((response) => response.json())
      .then((json) => {
        this.#removeFromArray(id)
        resolve(json.id)
      },(error) => {
        reject(error)
      })
    })
  }


  #readJson = (tasksAsJson) => {
```

On index.html add CDN for Bootstrap icons. Get CDN, for example, here and place new link between existing ones.

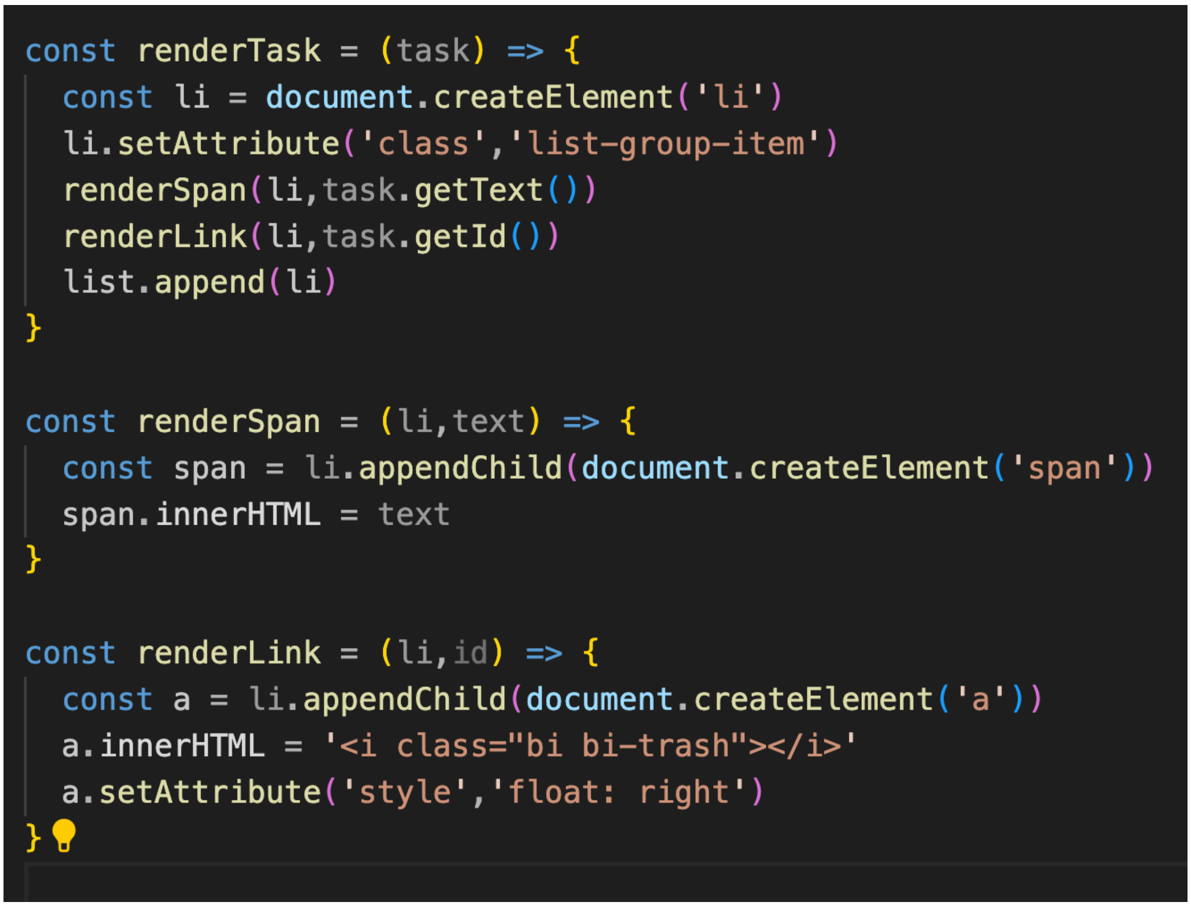


```
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstra
  <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.1/font/boots
  <link href="./css/style.css" rel="stylesheet">
```

Row on a list will contain text and icon. Therefore, code (index.js on client side) is changed so, that li element contains span for text, link and icon (i element is used for Bootstrap icons). To make code more structured, clear, and readable, separate functions for creating span and link are created.

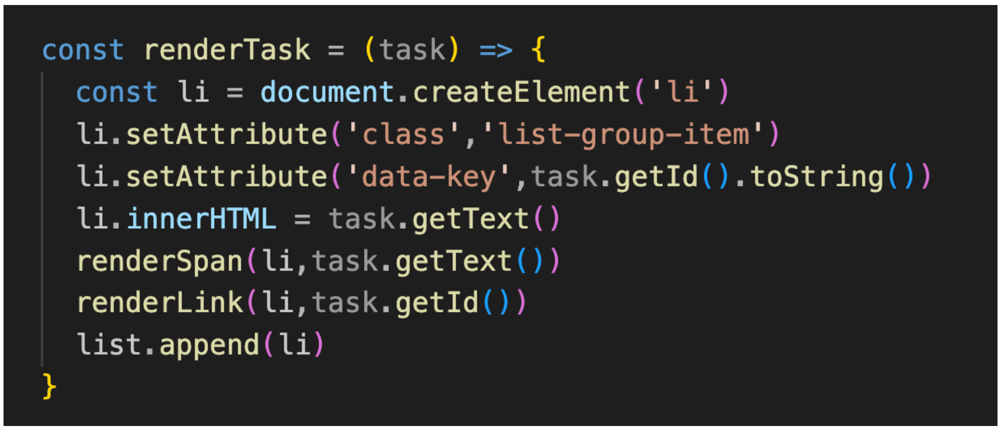
RenderSpan functions receives list item element and text as parameters. Span is appended as child for list item containing text (task description).

Link function receives list item and task id as parameters. Link contains Bootstrap icon (trash). Link and icon are displayed on the right (at the end of the list row so it is floated to right using CSS). At this point nothing happens yet if icon is pressed.



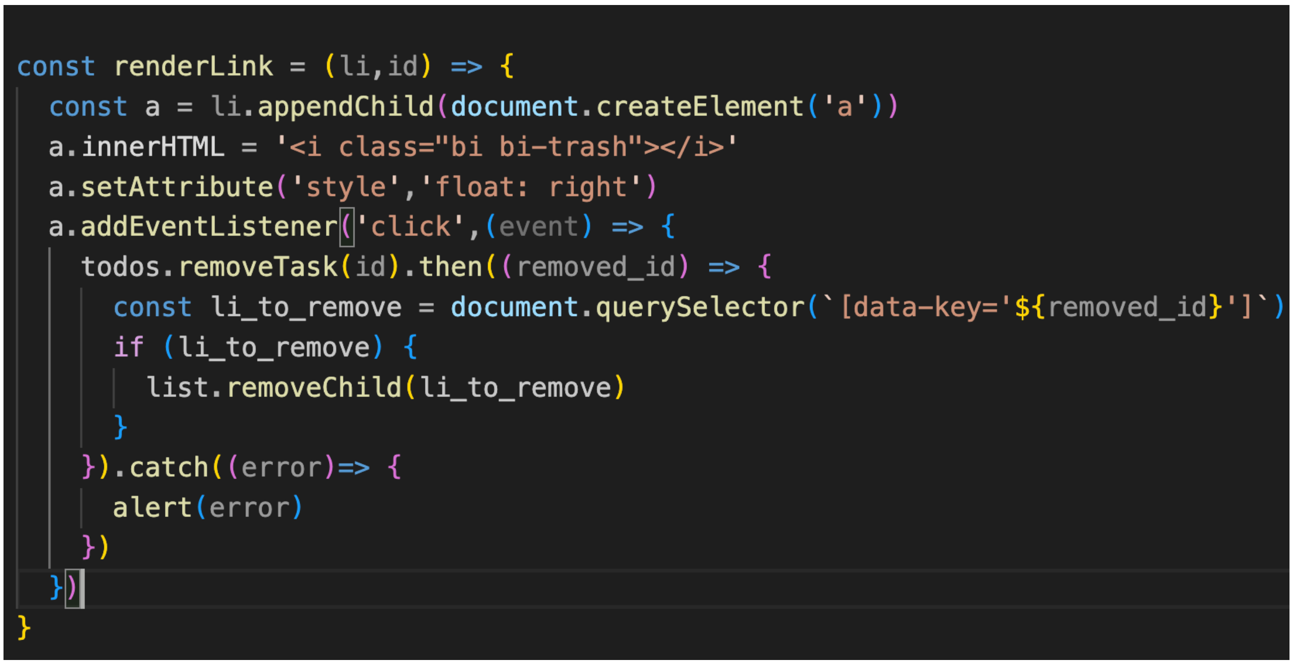
```
const renderTask = (task) => {
  const li = document.createElement('li')
  li.setAttribute('class','list-group-item')
  renderSpan(li,task.getText())
  renderLink(li,task.getId())
  list.append(li)
}

const renderSpan = (li,text) => {
  const span = li.appendChild(document.createElement('span'))
  span.innerHTML = text
}

const renderLink = (li,id) => {
  const a = li.appendChild(document.createElement('a'))
  a.innerHTML = '<i class="bi bi-trash"></i>'
  a.setAttribute('style','float: right')
}
```

Test out that web page is displaying icons on the list. Modify renderTask function so, that each list item (row) contains data-key attribute, which has task id stored into it. This will be used for locating list item that needs to be removed from UI when task is deleted.



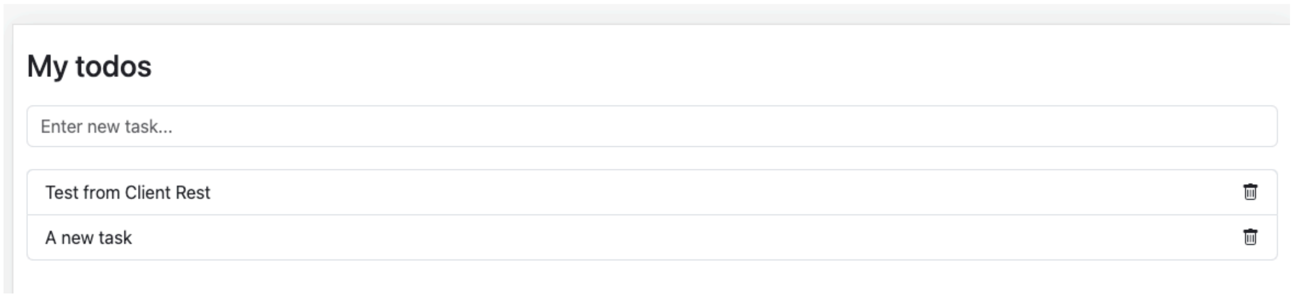
```
const renderTask = (task) => {
  const li = document.createElement('li')
  li.setAttribute('class','list-group-item')
  li.setAttribute('data-key',task.getId().toString())
  li.innerHTML = task.getText()
  renderSpan(li,task.getText())
  renderLink(li,task.getId())
  list.append(li)
}
```

Add click listener to link. When link (icon) is clicked, removeTask method is called from todos object. Data is deleted through HTTP call to the backend and UI is updated. List element that is clicked contains previously implemented data-key attribute with task id, which can be used to locate list item that needs to be removed from UI.



```
const renderLink = (li,id) => {
  const a = li.appendChild(document.createElement('a'))
  a.innerHTML = '<i class="bi bi-trash"></i>'
  a.setAttribute('style','float: right')
  a.addEventListener('click',(event) => {
    todos.removeTask(id).then((removed_id) => {
      const li_to_remove = document.querySelector(`[data-key='${removed_id}']`)
      if (li_to_remove) {
        list.removeChild(li_to_remove)
      }
    }).catch((error)=> {
      alert(error)
    })
  })
}
```

Test out that you can delete rows. Try to refresh browser and verify, that data is removed (and will not reappear to the page after refresh).


My todos

Enter new task...

Test from Client Rest

A new task