Virtualai

Techninė ataskaita

Projektą galime skirstyti į dvi dalis:

- Vidinis programavimas (back-end) :

  Naudojama kalba - C#. Pasinaudojant ASP.NET Core karkasu buvo įgyvendinta aplikacijų programavimo sąsaja, o Entity Framework ORM, buvo panaudotas duomenų prieigai MySQL reliacinėje duomenų bazėje.

- Išorinis programavimas (front-end):

  Naudojama kalba - Typescript. Vieno puslapio aplikacijos (SPA) vartotojo sąsajai sukurti naudota JavaScript biblioteka React ir dizaino sistema Material-UI.

Projekto architektūra susideda iš sluoksnių:

- Vidinis programavimas (back-end):
  - DAL (su api) - (ang. data access layer) sluoksnis atsakingas už biznio duomenų prieigą.
  - BLL (su api) - (ang. business logic layer) visa biznio logika yra rašoma čia.
- Išorinis programavimas (front-end):
  - UI (Presentation, View) - sluoksnis atsakingas atvaizduoti datą puslapyje ir reaguoti į vartotojo interakcijas.
  - Domain Layer - saugoma sinchronizuota data “state” objekte.
  - Data Access Layer - “promise” tipo atsakymais iš serverio pagrįstas ir nuo kitų dalių atskirtas bendravimas su serveriu.

Kokybiniai reikalavimai

Concurrency

```
156         if (isUserValid)
157         {
158             var user = users.Find(user => user.Email == authRequest.Email);
159
160             var claimsPrincipal = CreateClaims(user);
161             await Request.HttpContext.SignInAsync("Cookies", claimsPrincipal);
162
163             return NoContent();
164         }
165         else
```

UserController.cs (160)

The user session is created using cookies authorization on “api/Users/login” endpoint. After login users are able to do tasks on the website using multiple browsers tabs.

Security

```
25      public async Task<UserSubject> GetUserSubjectsById(int id)
26      {
27          var userSubject = await _context.UserSubjects.FindAsync(id);
28          return userSubject;
29      }
```

Entity Framework apsaugo nuo “sql injection” atakų. Todėl mūsų visi metodai tiesiogiai susiję su duomenų kūrimu/keitimu yra apsaugoti. (UserSubjectController.cs (25-29))

Data Access

```
44      // POST: api/Subjects
45      [HttpPost]
        0 references
46      public async Task<IActionResult> PostSubject(Subject subject)
47      {
48          _context.Subjects.Add(subject);
49          await _context.SaveChangesAsync();
50
51          if (subject.ParentId == null)
52          {
53              subject.ParentId = subject.Id;
54          }
55          _context.Entry(subject).State = EntityState.Modified;
56          await _context.SaveChangesAsync();
57
58          return NoContent();
59      }
60
```

SubjectsController.cs (45-59)

Data consistency; Optimistic locking Nėra

Memory management

API kontroleris yra sukonstruojamas kiekvienos užklausos įgyvendinimo metu ir sunaikinamas atlikus užklausą(request scoped), todėl nėra skiriama atmintis šiuo metu nenaudojamiems komponentams.

Reactive programming; Asynchronous/non-blocking communication

```
37      public async Task<ActionResult<User>> GetUser(int id)
38      {
39          var user = await _context.Users.FindAsync(id);
40          user.Team = await _context.Teams.FindAsync(user.TeamId);
41
42          if (user == null)
43          {
44              return NotFound();
45          }
46
47          return user;
48      }
49
```

UserController.cs (37-48)

Cross-cutting functionality/Interceptors Nėra

Extensibility/Glass-box extensibility Nėra