Blue letters on a black background

Description automatically generated

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ SISTEMŲ KATEDRA

Aleksandras Bogdanovas

**1 namų darbas**

Programų inžinerijos studijų programa, valstybinis kodas 6121BX023

Programų sistemų studijų kryptis

Atliko: Aleksandras Bogdanovas Prifs-21/4

Tikrino: ...

Vilnius, 2025

Turynis

[1. Ivadas 3](#_Toc190761616)

[2. MongoDB ir projekto konfigūravimas 4](#_Toc190761617)

[3. Kolekcijos ir atributai 5](#_Toc190761618)

[4. API realizacija 7](#_Toc190761619)

[5. API interaktyvus atvaizdavimas 11](#_Toc190761620)

[6. Duomenų eksportavimas 15](#_Toc190761621)

[7. Diagramų kūrimas 17](#_Toc190761622)

[8. Isvados 21](#_Toc190761623)

1. Ivadas

Šiame darbe pristatytas NoSQL duomenų modelio realizacijos procesas naudojant dokumentinę MongoDB duomenų bazę. Darbo metu buvo atlikti keli esminiai etapai:

* **MongoDB galimybių analizė**
* **Lokalus MongoDB diegimas**
* **Duomenų bazės kūrimas per API**
* **CRUD operacijų vykdymas**
* **Migracija į MongoDB Atlas debesį**

Darbo metu buvo siekiama ne tik atlikti praktinius uždavinius, bet ir giliau suprasti NoSQL duomenų modelių bei MongoDB privalumus bei taikymo galimybes šiuolaikiniuose duomenų saugojimo sprendimuose. Pateikti etapai ir iliustruojami paveikslėliais, siekiant aiškiai parodyti kiekvieno žingsnio eigą ir pasiektus rezultatus.

1. MongoDB ir projekto konfigūravimas

459-465 mongoDB ir jam reikalingų paketų instaliavimas.

466 – serverio paleidimas lokaliame kompiuteryje

476-477 – node.js serverio kūrimas su hono karkasu.

**A computer screen with white text

Description automatically generated**

firstHomework db kūrimas

**A screenshot of a computer

Description automatically generated**

Prisijungimas prie lokalaus ir atlaso serverio.

**A screenshot of a computer

Description automatically generated**

1. Kolekcijos ir atributai

User kolekcija ir jos atributai

A screen shot of a computer program

Description automatically generated

Project kolekcija ir jos atributai

A screen shot of a computer program

Description automatically generated

Task kolekcija ir jos atributai

A screen shot of a computer code

Description automatically generated

Comment kolekcija ir jos atributai

A screen shot of a computer code

Description automatically generated

1. API realizacija

Toliau bus parodytas User visi CRUD metodai, kitų kolekcijų CRUD realizacija yra panaši:

1. import { getDB } from "../config/mongodb";
2. import { userSchema } from "../beans/User";
3. import { createRoute, OpenAPIHono, z } from "@hono/zod-openapi";
4. import { ObjectId } from "mongodb";
5. const getUsers = createRoute({
6. method: "get",
7. path: "/",
8. responses: {
9. 200: {
10. content: {
11. "application/json": {
12. schema: userSchema.array(),
13. },
14. },
15. description: "Returns users",
16. },
17. },
18. });
19. const createUser = createRoute({
20. method: "post",
21. path: "/",
22. request: {
23. body: {
24. content: {
25. "application/json": {
26. schema: userSchema.omit({ \_id: true }),
27. },
28. },
29. },
30. },
31. responses: {
32. 200: {
33. content: {
34. "application/json": {
35. schema: z.object({
36. insertedId: z.string(),
37. acknowledged: z.boolean(),
38. }),
39. },
40. },
41. description: "Creates new user and returns it",
42. },
43. },
44. });
45. const deleteUser = createRoute({
46. method: "delete",
47. path: "",
48. request: {
49. body: {
50. content: {
51. "application/json": {
52. schema: z.object({
53. \_id: z.string(),
54. }),
55. },
56. },
57. },
58. },
59. responses: {
60. 200: {
61. content: {
62. "application/json": {
63. schema: z.object({
64. deletedCount: z.number(),
65. acknowledged: z.boolean(),
66. }),
67. },
68. },
69. description: "Deletes user",
70. },
71. },
72. });
73. const updateUser = createRoute({
74. method: "put",
75. path: "",
76. request: {
77. body: {
78. content: {
79. "application/json": {
80. schema: z.object({
81. \_id: z.string(),
82. name: z.string().nullable(),
83. email: z.string().email().nullable(),
84. role: z.enum(["frontend", "backend", "fullstack"]),
85. }),
86. },
87. },
88. },
89. },
90. responses: {
91. 200: {
92. content: {
93. "application/json": {
94. schema: z.object({
95. modifiedCount: z.number(),
96. acknowledged: z.boolean(),
97. }),
98. },
99. },
100. description: "Updates user",
101. },
102. },
103. });
104. export const createUserModule = () => {
105. const app = **new** OpenAPIHono();
106. return app
107. .openapi(getUsers, async (*c*) => {
108. const db = getDB();
109. const users = await db.collection("users").find().toArray();
110. const validatedUsers = userSchema.array().safeParse(users);
111. if (!validatedUsers.success) {
112. throw **new** *Error*("Failed to validate users");
113. }
114. return *c*.json(validatedUsers.data);
115. })
116. .openapi(createUser, async (*c*) => {
117. const db = getDB();
118. const body = *c*.req.valid("json");
119. const result = await db.collection("users").insertOne(body);
120. return *c*.json({
121. insertedId: result.insertedId,
122. acknowledged: result.acknowledged,
123. });
124. })
125. .openapi(deleteUser, async (*c*) => {
126. const db = getDB();
127. const body = *c*.req.valid("json");
128. const result = await db
129. .collection("users")
130. .deleteOne({ \_id: **new** ObjectId(body.\_id) });
131. return *c*.json({
132. acknowledged: result.acknowledged,
133. deletedCount: result.deletedCount,
134. });
135. })
136. .openapi(updateUser, async (*c*) => {
137. const db = getDB();
138. const body = *c*.req.valid("json");
139. const updateSchema = z
140. .object({
141. \_id: z.string(),
142. name: z.string().nullable(),
143. email: z.string().email().nullable(),
144. role: z.string().nullable(),
145. })
146. .partial();
147. const validatedUpdateData = updateSchema.parse(body);
148. if ("\_id" in validatedUpdateData) {
149. delete validatedUpdateData.\_id;
150. }
151. const filteredUpdateData: *Partial*<typeof validatedUpdateData> = {};
152. for (const key in validatedUpdateData) {
153. const typedKey = key as keyof typeof validatedUpdateData;
154. const value = validatedUpdateData[typedKey];
155. if (value !== "" && value !== null) {
156. filteredUpdateData[typedKey] = value;
157. }
158. }
159. const result = await db
160. .collection("users")
161. .updateOne(
162. { \_id: **new** ObjectId(body.\_id) },
163. { $set: filteredUpdateData }
164. );
165. return *c*.json({
166. acknowledged: result.acknowledged,
167. modifiedCount: result.modifiedCount,
168. });
169. });
170. };
171. API interaktyvus atvaizdavimas

Naudojant @hono/zod-openapi buvo padarytas API ir jos interaktyvus atvaizdavimas

A screenshot of a computer

Description automatically generated

Naujo User kūrimas

A screenshot of a computer

Description automatically generated

User sarašas

A screenshot of a computer

Description automatically generated

User redagavimas

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

User trinimas

A screenshot of a computer

Description automatically generated

Kitų kolekcijų valdymas vyksta panašiai.

1. Duomenų eksportavimas

Duomenų paruošimas eksportui A screen shot of a computer

Description automatically generated

Duomenų eksportavimas į atlas serverį

A screen shot of a computer

Description automatically generated

Atlas po duomenų importavimo

A screenshot of a computer

Description automatically generated

1. Diagramų kūrimas

A screenshot of a graph

Description automatically generated

Šioje diagramoje rodoma, kiek vartotojų priskirta kiekvienai rolei. Duomenys imami iš kolekcijos USER ir grupuojami pagal lauką ROLE.

A screenshot of a graph

Description automatically generated

Šioje diagramoje rodomas užduočių skaičius pagal jų statusą. Duomenys imami iš kolekcijos tasks, o užduotys grupuojamos pagal lauką status. Diagramoje pateikiama, kiek užduočių turi kiekvienas statusas (pvz., "todo", "in-progress", "done") naudojant stulpelinę diagramą.

A screenshot of a graph

Description automatically generated

Šioje diagramoje pateikiamas užduočių skaičius pagal jų prioritetus. Duomenys imami iš kolekcijos tasks ir užduotys grupuojamos pagal lauką priority. Diagramoje matomas kiekvieno prioriteto (žemas, vidutinis, aukštas) užduočių skaičius, naudojant histogramos arba apskritiminės diagramos tipą.

A screenshot of a computer

Description automatically generated

Šioje diagramoje pavaizduota, kiek komentarų parašė kiekvienas autorius. Duomenys imami iš kolekcijos comments ir grupuojami pagal lauką authorId. Kiekvienas stulpelis atspindi konkretaus autoriaus parašytų komentarų skaičių.

1. Isvados

Projekte demonstruotas pilnas MongoDB dokumentinės duomenų bazės panaudojimo ciklas. Sukurtos kolekcijos (users, projects, tasks, comments) ir įgyvendintos CRUD operacijos per API leido efektyviai valdyti ir manipuliuoti duomenimis. Be to, duomenų migracija į MongoDB Atlas debesį suteikė galimybę naudotis skalabilumo ir centralizuoto stebėjimo privalumais. Sukurtos diagramos – vartotojų pasiskirstymas pagal roles, užduočių pasiskirstymas pagal statusą ir prioritetus, bei komentarų pasiskirstymas pagal autorius – aiškiai iliustruoja sistemos struktūrą ir suteikia vertingų įžvalgų duomenų analizėje. Projektas sėkmingai parodo, kaip šiuolaikinės technologijos gali būti taikomos informacinių sistemų kūrime ir palaikyme.