

Reflectus alternatyve

Pokalbio transkriptas (vartotojas ir asistentas)

Sugeneruota: 2026-01-04 07:34 (vietinis laikas)

Turinys

Šiame PDF pateikiamas visas šio pokalbio tekstas, išskaitant kodų blokus.

1. Vartotojas

reflect. sukurti darbų planą

2. Asistentas

Reflect (Reflectus alternatyva) – darbų planas nuo 0 iki paleidimo

0) Sprendimai, kuriuos reikia apsibrėžti (kad planas “susikabintų”)

1) Tik **web** ar ir **mobile** (pradžiai siūlau web + mobile-friendly).

2) Vartotojai: **Mokinys / Mokytojas / Admin (mokykla)**. Ar bus **Tėvas**?

3) Klasių modelis: mokytojas kuria klasę ir kviečia, ar importas iš mokyklos?

4) Refleksijos tipai: **laisva**, **klausimynas**, **įrodymai (nuotraukos/ failai)**, **įsivertinimas balais**.

5) Matomumas: ką mokytojas mato visada, o kas yra privatu (pvz., “tik mokytojui”).

6) Ar reikės **pažymiu** / **dienyno** funkcijų (geriau nepradėti nuo to).

7) Auth: paprastas el. paštas/slaptažodis ar **Google/Microsoft** (mokykloms dažnai reikia Microsoft).

8) GDPR: ar naudotojai **nepilnamečiai** (beveik visada) → reikės aiškių sutikimų/taisyklių.

1) Produktų specifikacija (PRD) ir MVP ribos

Tikslias MVP: mokinys reflektuoja, mokytojas mato progresą ir gali duoti grįztamajį ryšį.

MVP funkcijos

- **Paskyros ir rolės**: Mokinys / Mokytojas / Admin.

- **Klasės**: mokytojas sukuria klasę, sugeneruoja kvietimo kodą/linką.

- **Refleksijos**:

- šablonai (klausimai) + laisvas tekstas

- “nuotaika / savijauta” (paprasta skalė)

- “ką išmokau / kas sunku / ko reikia pagalbos”

- **Mokytojo peržiūra**:

- mokinio timeline

- klasės suvestinė (kas pateikė, kas ne)

- filtrai pagal datą / šabloną / mokinį

- **Komentarai/feedback**: mokytojas parašo komentarą prie refleksijos.

- **Prisegti failai** (nebūtina pirmą dieną, bet dažniausiai reikės): nuotrauka/pdf (ribojimai).

- **Audit trail (minimum)**: kas ką peržiūrėjo / keitė (mokykloms svarbu).

Ne-MVP (vėliau): chat'as, pažymiai, integracijos su Teams/Google Classroom, tėvų paskyros, pažangios ataskaitos.

2) UX/UI darbai (kad dev būtų greitas)

Sukuriame 10-12 ekranų “wireframe” lygmeniu (Figma arba paprastai).

Ekranai:

- Login / Register

- Pasirinkti rolę (ar admin priskiria)

- Mokinys: Dashboard (šiandienos refleksija, istorija)

- Refleksijos kūrimas (pagal šabloną)

- Refleksijos peržiūra + komentarai

- Mokytojas: Klasės suvestinė

- Mokytojas: Mokinio profilis (progresas)

- Klasės valdymas (mokiniai, kvietimo kodas)
- Admin: mokyklos vartotojų valdymas (minimaliai)

3) Techninė architektūra (paprasta, tvarkinga, saugi)

Rekomenduojama struktūra (monorepo)

- `backend/` (API)
- `frontend/` (web)
- `infra/` (Docker, deploy skriptai, CI)

Backend (siūlymas)

- Node.js + Express (ar NestJS jei norit griežtesnės struktūros)
- DB: **PostgreSQL** (labai tinkta klasėms/ryšiams/ataskaitoms)
- ORM: Prisma
- Auth: JWT + refresh token (httpOnly cookie) arba session
- Failai: S3 compatible (Cloudflare R2 / AWS S3 / MinIO)

Frontend

- React (Vite)
- UI: paprasta komponentų biblioteka (pvz. shadcn)
- State: minimaliai (React Query)

4) Duomenų modelis (MVP)

Minimalios lentelės:

- `users` (id, email, role, status)
- `schools` (id, name)
- `classes` (id, school_id, teacher_id, name, join_code)
- `class_members` (class_id, student_id)
- `reflection_templates` (id, class_id/null, title, questions_json)
- `reflections` (id, student_id, class_id, template_id, answers_json, mood, created_at)
- `reflection_comments` (id, reflection_id, teacher_id, text, created_at)
- `files` (id, owner_id, reflection_id, url, type, size)
- `audit_logs` (actor_id, action, entity, entity_id, created_at, meta_json)

5) API planas (MVP endpoint'ai)

- Auth: `/auth/register`, `/auth/login`, `/auth/logout`, `/auth/refresh`

- Vartotojas: `/me`

- Klasės:

- `POST /classes` (mokytojas)
- `POST /classes/join` (mokinys su kodu)

- `GET /classes/:id`

- Šablonai: `GET/POST /templates`

- Refleksijos:

- `POST /reflections`
- `GET /reflections?classId=&studentId=&from=&to=`
- `GET /reflections/:id`

- Komentarai: `POST /reflections/:id/comments`

- Failai: `POST /files/presign` (įkėlimas tiesiai į S3)

- ## 6) Saugumas ir teisiniai minimumai (nepraleisti)
- **RBAC**: mokytojas mato tik savo klasses; mokinys mato tik savo.
 - **Rate limiting** ir brute-force apsauga login'e.
 - **httpOnly cookies** refresh tokenams.
 - **Audit logs** bent peržiūroms/keitimams.
 - **Šifravimas**: TLS visur; DB atsarginės kopijos; secret'ai tik per env/secret manager.
 - **GDPR**:
 - aiškus privatumo aprašas (kas saugoma, kiek laiko)
 - duomenų trynimo/eksporto procesas
 - jei nepilnamečiai → mokyklos/teisėto pagrindo dalis (sutartis su mokykla; minimalūs duomenys).

7) Įgyvendinimo darbų eilė (tvarka, kad nesulūžtu)

A. Bazė (projekto "skeletas")

- 1) Repo sukurimas + monorepo struktūra
- 2) Docker dev aplinka: Postgres, backend, frontend
- 3) CI (GitHub Actions): lint + test + build
- 4) Konfig: ` `.env.example` , logging, error handler

B. Auth + rolės

- 1) Register/login + email patvirtinimas (galima atidėti, bet geriau turėti)
- 2) Roles/permissions middleware
- 3) `/me` endpoint + frontend session flow

C. Klasės

- 1) Kurti klasę (mokytojas)
- 2) Join su kodu (mokinys)
- 3) Klasės narių sąrašas + mokytojo suvestinė

D. Refleksijos

- 1) Template kūrimas (mokytojas)
- 2) Refleksijos pateikimas (mokinys)
- 3) Peržiūra + filtrai (mokytojas)
- 4) Komentarai

E. Failai (jei reikia)

- 1) Presigned upload į S3/R2
- 2) Failų tipų/size limitai
- 3) Susiejimas su refleksija

F. Kokybė ir paleidimas

- 1) Testai (API integraciniai + keli e2e)
- 2) Monitoring (logs, uptime)
- 3) Deploy: staging → production
- 4) Backup/restore patikra

8) Infrastruktūra (paprasta ir patikima)

Staging + Production atskirai.

- Host'as: EU regionas (pvz. Hetzner/DO/AWS EU)
- DB: managed Postgres (arba savo Postgres su backup automatika)
- Storage: S3 compatible (R2/S3)
- Reverse proxy: Nginx
- TLS: Let's Encrypt

- Monitoring: Uptime + basic metrics

9) "Definition of Done" (kad žinotum, jog MVP tikrai paruoštas)

- Mokinys gali prisijungti, prisijungti prie klasės, užpildyti refleksiją.
- Mokytojas mato klasės suvestinę, atsiverčia mokinį, pamato refleksijas, palieka komentarą.
- Teisės veikia (kitas mokytojas nemato svetimų klasių).
- Yra backup'ai, logai, pagrindinės apsaugos (rate limit, secure cookies).
- Staging ir production deploy veikia vienu komandu/CI.

Kad galėčiau "užfiksuoti" galutinį darbų planą į konkrečius sprintus ir užduotis (su check-box'ais), parašyk trumpai:

- 1) ar startuojam **tik web**?
- 2) ar norit **Google/Microsoft login** iškart, ar vėliau?
- 3) ar būtini **failai/nuotraukos** pirmam MVP?
- 4) vienas mokytojas → kelios klasės (taip/ne)?
- 5) ar tai bus **vienai mokyklai** (private) ar **platforma daug mokyklų**?

3. Vartotojas

2

bus tik login. bus sistemoje surašyti prisijungimai. viska duos mokykla.
tai roles bus paskirtos pagal prisijungimo pastus.

patikslinimas

4. Asistentas

Patikslinimas (pagal tavo sąlygą "bus tik login, viską duos mokykla")

Kaip veiks sistema

- **Nėra registracijos.** Vartotojas negali susikurti paskyros pats.
- **Visos paskyros iš anksto suvestos sistemoje** (mokykla pateikia sąrašą).
- **Prisijungimas = tik login (email + slaptažodis)**.
- **Rolė nustatoma pagal tą patį email**, kuris jau yra įrašytas sistemoje:
- jei email įrašytas kaip `role=student` → prisijungus vartotojas yra mokinys
- jei `role=teacher` → mokytojas
- (rekomenduoju turėti dar `role=school_admin`, nes kažkas turi suvesti/keisti vartotojus)

Svarbu: rolė **nenustatoma "iš email teksto" automatiškai** (pvz. pagal domeną), o
pagal įrašą DB. Tai yra saugiau ir paprasčiau valdyti.

Ko tai reikalauja iš sistemos (kad veiktų realiai mokykloje)

1) Vartotojų suvedimo būdas (mokyklai)

Minimaliai turi būti vienas iš šių variantų:

A) Admin suveda ranka per admin panelę

- tinkai mažai mokyklai / pilotui

****B) CSV importas (rekomenduoju MVP)****

Mokykla duoda CSV, pvz. stulpeliai:

- `email, role, first_name, last_name, class_id(or class_name)`
- (jei reikia) `temp_password` arba sistema sugeneruoja

****C) Integracija su mokyklos sistemomis** (vėliau, ne MVP)**

2) Slaptažodžių politika (nes mokykla “duos prisijungimus”)

Kad nereiktų siuntinėti slaptažodžių nesaugiai, geriausias MVP sprendimas:

- mokykla duoda ****laikiną slaptažodį****
- ****pirmo prisijungimo metu priverstinai pakeisti slaptažodį****

Alternatyva (jei norit dar saugiau): ****vienkartinis prisijungimo linkas**** (magic link), bet tada reikia email siuntimo infrastruktūros.

Kas keičiasi tavo ankstesniame plane (konkrečiai)

API (backend)

Išmeti:

- `/auth/register`
- klasės “join code” logiką (`/classes/join`) jei mokiniai priskiriami iš anksto

Palieki / pridedi:

- `POST /auth/login`
- `POST /auth/logout`
- `GET /me`
- `POST /auth/change-password` (privalomas pirmam login)
- Admin (mokyklai):
 - `POST /admin/users/import-csv`
 - `POST /admin/users` (rankinis sukūrimas)
 - `PATCH /admin/users/:id` (role, class, status)
 - `POST /admin/users/:id/reset-password` (sugeneruoja laikiną)

DB modelis (minimum)

`users` lentelėje papildomai:

- `role` (student/teacher/school_admin)
- `status` (active-disabled)
- `must_change_password` (true/false)

Narystė klasėse:

- jei mokiniai priskiriami iš anksto → `class_members` valdo admin/mokytojas (pagal jūsų taisyklę)

UI (frontend)

Išmeti “registraciją” ir “prisijungimą prie klasės su kodu”.

Palieki:

- Login ekrana
- Pirmo prisijungimo “pasikeisk slaptažodį”
- Student/Mokytojas dashboard pagal rolę
- School admin: vartotojų importas + valdymas

Minimalus darbų planas (MVP) pagal šitą patikslinimą

1) ****Auth be registracijos****

- login + session (JWT/refresh cookie)

- rolės iš DB
- priverstinis slaptažodžio keitimas pirmą kartą

2) **School admin panelė**

- CSV importas
- vartotojų aktyvavimas/išjungimas
- reset password

3) **Klasės ir priskyrimai**

- admin priskiria mokinius klasėms
- mokytojas mato tik savo klases

4) **Refleksijos ir mokytojo peržiūra**

- šablonai
- mokinio refleksija
- mokytojo suvestinė + komentarai

5) **Audit + saugumas**

- rate limit ant login
- lockout po N bandymų
- audit log bent: login, password reset, role/class change

Vienas klausimas, kad nesuklystume (kritinis)

Mokyklos prisijungimai bus:

- **A)** mokyklos išduoti **bendri** (pvz. mokiniai neturi savo email, o gauna "user123" tipo)
ar
- **B)** kiekvienas turi **savo realų email**?

Jei A — login bus "username", ne email, ir CSV struktūra keičiasi. Jei B — einam su email kaip aukščiau.

5. Vartotojas

mokykla suves į duomenų baze viska

6. Asistentas

Gerai. Tada darom modelį **be registracijos ir be "join code"**, o mokykla **provisionina (suveda/atnaujina) visus duomenis DB**. Svarbiausia: mokykla neturi jungtis prie DB "iš išorės" bet kaip (nes tai didžiausia saugumo skylė). Todėl apibrėžiam 1 iš 3 saugią variantą, ir pagal tai darom darbų planą.

1) Kaip realiai "mokykla suves į DB" (saugūs variantai)

- ### Variantą A (rekomenduojamas): Mokyklos admin per vidinę Admin UI
- Mokykla suveda vartotojus/klases per web admin panelę.
 - DB lieka uždaryta nuo interneto (prie jos jungiasi tik backend).
 - Pliusai: saugiausia, paprasčiausia eksplotuoti, auditas.

Variantas B: Importas per serverio skriptą (CSV → DB)

- Mokykla pateikia CSV failą, o jūs (ar jū admin) paleidžia importo komandą serveryje.
- DB vis tiek neliečiama iš išorės.
- Pliusai: greita MVP, nereikia gražios admin UI.

Variantas C (nerekomenduoju): Tiesioginis DB pildymas

- Galima tik jei DB pasiekiamas **tik per VPN + whitelist + read/write ribojimai**, ir yra auditas.

- Kitaip: anksčiau ar vėliau bus incidentas (nutekėjimas / sugadinimas / neteisėtas priėjimas).

Jei tu sakai "mokykla suves į DB", praktiškai MVP'ė dažniausiai renkamės **B**, o vėliau pereinam į **A**.

2) Ką tai keičia produkto logikoje

- **Login tik su email + password.**
- **Rolė ir klasės priskyrimas ateina iš DB.**
- Nėra "register", nėra "join class".

Prisijungimo srautas:

- 1) user įveda email + slaptažodį
- 2) backend suranda `users` įrašą
- 3) patikrina hash'ą
- 4) grąžina session/JWT
- 5) frontend nukreipia pagal `role`

3) DB sutartis (ką mokykla PRIVALO užpildyti)

Minimalus duomenų rinkinys, be kurio MVP neveiks:

Lentelės

- `users`: `email` (unikalus), `role` (student/teacher/school_admin), `password_hash`, `status`, `must_change_password`
- `classes`: klasės pavadinimas, mokytojas
- `class_members`: mokinys → klasė (ryšys)
- (vėliau) `templates`, `reflections`, `comments`

Privalomos taisyklos

- `email` unikalus
- `role` tik iš leidžiamų reikšmių
- mokinys privalo turėti bent 1 klasę
- mokytojas mato tik savo klases (pagal `classes.teacher_id`)

4) Slaptažodžiai (kritiška)

Kad mokykla galėtų "suvesti prisijungimus" saugiai:

- DB NIEKADA nelaikom "plain" slaptažodžių.
- Laikom tik `password_hash` (argon2id arba bcrypt).
- Rekomendacija MVP:
 - mokykla suveda laikinius slaptažodžius
 - `must_change_password=true`
 - pirmo prisijungimo metu user privalo pasikeisti

5) Darbų planas pagal šitą modelį (MVP)

Sprint 1 — Pamatas

- 1) DB schema + migracijos

- 2) Backend auth: `POST /auth/login` , `POST /auth/logout` , `GET /me`
- 3) Role-based access (middleware): student/teacher/admin
- 4) Frontend: Login ekranas + nukreipimas pagal rolę

Sprint 2 — Mokyklos duomenų įkėlimas (Variantas B)

- 1) CSV formatas (sutartas šablonas)
- 2) Importo skriptas serveryje (CSV → upsert į DB)
- 3) Validacijos + klaidų raportas (kas nepraejo ir kodėl)
- 4) Priverstinis "change password" flow

Sprint 3 — Klasės ir refleksijos

- 1) Mokinio refleksijos kūrimas + istorija
- 2) Mokytojo klasės suvestinė + mokinio timeline
- 3) Mokytojo komentarai

Sprint 4 — Saugumas ir eksplotacijos

- 1) Rate limit login'ui + lockout po N bandymų
- 2) Audit log (ypač: login, password change, user status/role/class change)
- 3) Backup/restore patikra + monitoring

6) Svarbi pastaba dėl realios eksplotacijos

Jei mokykla "suvės į DB" tiesiogiai, tu prarandi:

- auditą (kas pakeitė ką)
- validacijas
- kontrolę (lengva sugadinti ryšius)

Todėl net jei pasirenkam Variantą B (skriptas), vis tiek gauni:

- vieną kontroliuojamą įėjimo tašką
- aiškias klaidas
- mažiau incidentų

Kad užfiksuočiau galutinį pasirinkimą be papildomų klausimų

Aš siūlau: **MVP = Variant B (CSV import skriptas)**, o po to **Variant A (admin panelė)**.

Jei sutinki, kitame žingsnyje duosiu:

- tikslų CSV šabloną (stulpeliai)
- importo taisykles (upsert logika)
- konkretų "serverio komandų" planą Ubuntu aplinkoje (kur paleisti, kur laikyti failą, kaip gauti log'ą).

7. Vartotojas

reflectus alternatyve . programėlės idejos apžvalga yra nuotraukos. oagal jas sukurkime front end mokiniui. ir backend kad kažkas šiektiek veiktu

8. Asistentas

Žemiau duodu **MVP mokinio frontend + minimalų backend**, kad realiai veiktų: **login → mokinio dashboard → sukurti šiandienos refleksiją → matyti istoriją → matyti detalę**.

Jei nori 1:1 pagal tavo nuotraukas (layout/tekstai), įkelk tas nuotraukas į šitą chat'ą – bet pradžiai darom veikiančią bazę.

1) Projekto struktūra (monorepo)

Sukuriam naują katalogą (Ubuntu terminale):

Code: bash

```
mkdir -p ~/projektai/reflectus-alt && cd ~/projektai/reflectus-alt  
mkdir backend
```

Struktūra:

Code

```
reflectus-alt/  
  backend/  
  frontend/  
  docker-compose.yml
```

2) DB (Mongo) – greičiausias MVP

Variant A: per Docker (rekomenduojamas)

Sukurk `docker-compose.yml` šakniniame kataloge `~/projektai/reflectus-alt`:

Code: yaml

```
services:  
  mongo:  
    image: mongo:7  
    ports:  
      - "27017:27017"  
    volumes:  
      - mongo_data:/data/db  
volumes:  
  mongo_data:
```

Paleisk:

Code: bash

```
cd ~/projektai/reflectus-alt  
docker compose up -d
```

3) Backend (Express + JWT per httpOnly cookie)

3.1. Backend install

Ubuntu terminale:

Code: bash

```
cd ~/projektai/reflectus-alt/backend  
npm init -y  
npm i express cors cookie-parser dotenv mongoose bcrypt jsonwebtoken  
npm i -D nodemon  
mkdir -p src/{config,models,middleware,routes,utils}
```

`backend/package.json` papildyk scripts:

Code: json

```
"scripts": {  
  "dev": "nodemon src/server.js",  
  "seed": "node src/utils/seed.js"  
}
```

3.2. Backend .env

Sukurk `backend/.env.example`:

Code: env

```
PORT=5000
MONGO_URI=mongodb://127.0.0.1:27017/reflectus_alt
JWT_SECRET=change_me_long_random
CLIENT_ORIGIN=http://localhost:5173
```

Tada:

Code: bash

```
cp .env.example .env
```

3.3. Backend kodas (jdék failus tiksliai)

`backend/src/config/db.js`

Code: js

```
const mongoose = require("mongoose");

async function connectDB(uri) {
  mongoose.set("strictQuery", true);
  await mongoose.connect(uri);
  console.log("MongoDB connected");
}

module.exports = { connectDB };
```

`backend/src/models/User.js`

Code: js

```
const mongoose = require("mongoose");

const UserSchema = new mongoose.Schema(
{
  email: { type: String, required: true, unique: true, lowercase: true, trim: true },
  role: { type: String, enum: ["student", "teacher", "school_admin"], required: true },
  passwordHash: { type: String, required: true },
  status: { type: String, enum: ["active", "disabled"], default: "active" },
  mustChangePassword: { type: Boolean, default: true },
  firstName: { type: String, default: "" },
  lastName: { type: String, default: "" }
},
{ timestamps: true }
);

module.exports = mongoose.model("User", UserSchema);
```

`backend/src/models/Reflection.js`

Code: js

```

const mongoose = require("mongoose");

const ReflectionSchema = new mongoose.Schema(
{
  studentId: { type: mongoose.Schema.Types.ObjectId, ref: "User", required: true, index: true },
  dateKey: { type: String, required: true, index: true }, // YYYY-MM-DD
  mood: { type: Number, min: 1, max: 5, required: true },
  answers: {
    learned: { type: String, required: true },
    hard: { type: String, required: true },
    help: { type: String, required: true }
  },
  teacherComment: { type: String, default: "" }
},
{ timestamps: true }
);

ReflectionSchema.index({ studentId: 1, dateKey: 1 }, { unique: true });

module.exports = mongoose.model("Reflection", ReflectionSchema);

```

`backend/src/middleware/auth.js`

Code: js

```

const jwt = require("jsonwebtoken");

function requireAuth(req, res, next) {
  const token = req.cookies?.access_token;
  if (!token) return res.status(401).json({ error: "UNAUTHORIZED" });

  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET);
    req.user = payload; // { id, role, email }
    return next();
  } catch {
    return res.status(401).json({ error: "UNAUTHORIZED" });
  }
}

function requireRole(...roles) {
  return (req, res, next) => {
    if (!req.user?.role || !roles.includes(req.user.role)) {
      return res.status(403).json({ error: "FORBIDDEN" });
    }
    next();
  };
}

module.exports = { requireAuth, requireRole };

```

`backend/src/routes/auth.js`

Code: js

```

const express = require("express");
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const User = require("../models/User");
const { requireAuth } = require("../middleware/auth");

const router = express.Router();

router.post("/login", async (req, res) => {
  const { email, password } = req.body || {};
  if (!email || !password) return res.status(400).json({ error: "MISSING_FIELDS" });

  const user = await User.findOne({ email: String(email).toLowerCase().trim() });
  if (!user || user.status !== "active") return res.status(401).json({ error: "INVALID_CREDENTIALS" });

  const ok = await bcrypt.compare(password, user.passwordHash);
  if (!ok) return res.status(401).json({ error: "INVALID_CREDENTIALS" });

  const token = jwt.sign(
    { id: user._id.toString(), email: user.email, role: user.role },
    process.env.JWT_SECRET,
    { expiresIn: "2h" }
  );

  res.cookie("access_token", token, {
    httpOnly: true,
    sameSite: "lax",
    secure: false
  });

  res.json({
    ok: true,
    user: {
      id: user._id,
      email: user.email,
      role: user.role,
      mustChangePassword: user.mustChangePassword,
      firstName: user.firstName,
      lastName: user.lastName
    }
  });
});

router.post("/logout", (req, res) => {
  res.clearCookie("access_token", { httpOnly: true, sameSite: "lax", secure: false });
  res.json({ ok: true });
});

router.get("/me", requireAuth, async (req, res) => {
  const user = await User.findById(req.user.id).select("email role mustChangePassword firstName lastName");
  if (!user || user.status !== "active") return res.status(401).json({ error: "UNAUTHORIZED" });
  res.json({ ok: true, user });
});

router.post("/change-password", requireAuth, async (req, res) => {
  const { newPassword } = req.body || {};
  if (!newPassword || String(newPassword).length < 8) return res.status(400).json({ error: "WEAK_PASSWORD" });

  const passwordHash = await bcrypt.hash(String(newPassword), 12);
  await User.findByIdAndUpdate(req.user.id, { passwordHash, mustChangePassword: false });

  res.json({ ok: true });
});

module.exports = router;

```

`backend/src/routes/reflections.js`

Code: js

```
const express = require("express");
const Reflection = require("../models/Reflection");
const { requireAuth, requireRole } = require("../middleware/auth");

const router = express.Router();

function todayKey() {
  const d = new Date();
  const yyyy = d.getFullYear();
  const mm = String(d.getMonth() + 1).padStart(2, "0");
  const dd = String(d.getDate()).padStart(2, "0");
  return `${yyyy}-${mm}-${dd}`;
}

router.get("/", requireAuth, requireRole("student"), async (req, res) => {
  const items = await Reflection.find({ studentId: req.user.id }).sort({ createdAt: -1 }).limit(60);
  res.json({ ok: true, items });
});

router.get("/today", requireAuth, requireRole("student"), async (req, res) => {
  const key = todayKey();
  const item = await Reflection.findOne({ studentId: req.user.id, dateKey: key });
  res.json({ ok: true, dateKey: key, item: item || null });
});

router.post("/", requireAuth, requireRole("student"), async (req, res) => {
  const { mood, answers } = req.body || {};
  if (!mood || !answers?.learned || !answers?.hard || !answers?.help) {
    return res.status(400).json({ error: "MISSING_FIELDS" });
  }

  const doc = await Reflection.create({
    studentId: req.user.id,
    dateKey: todayKey(),
    mood: Number(mood),
    answers: {
      learned: String(answers.learned),
      hard: String(answers.hard),
      help: String(answers.help)
    }
  });

  res.json({ ok: true, item: doc });
});

router.get("/:id", requireAuth, requireRole("student"), async (req, res) => {
  const item = await Reflection.findOne({ _id: req.params.id, studentId: req.user.id });
  if (!item) return res.status(404).json({ error: "NOT_FOUND" });
  res.json({ ok: true, item });
});

module.exports = router;
```

`backend/src/utils/seed.js` (dev test vartotojui)

Code: js

```
require("dotenv").config();
const bcrypt = require("bcrypt");
const { connectDB } = require("../config/db");
const User = require("../models/User");

(async () => {
    await connectDB(process.env.MONGO_URI);

    const email = "student@test.lt";
    const existing = await User.findOne({ email });
    if (!existing) {
        const passwordHash = await bcrypt.hash("Test12345!", 12);
        await User.create({
            email,
            role: "student",
            passwordHash,
            status: "active",
            mustChangePassword: false,
            firstName: "Test",
            lastName: "Student"
        });
        console.log("👤 Seeded:", email, "password: Test12345!");
    } else {
        console.log("👤 Already exists:", email);
    }

    process.exit(0);
})();
```

`backend/src/server.js`

Code: js

```

require("dotenv").config();
const express = require("express");
const cors = require("cors");
const cookieParser = require("cookie-parser");
const { connectDB } = require("./config/db");

const authRoutes = require("./routes/auth");
const reflectionRoutes = require("./routes/reflections");

const app = express();

app.use(cors({
  origin: process.env.CLIENT_ORIGIN,
  credentials: true
}));
app.use(express.json({ limit: "1mb" }));
app.use(cookieParser());

app.get("/health", (req, res) => res.json({ ok: true }));

app.use("/api/auth", authRoutes);
app.use("/api/reflections", reflectionRoutes);

// Central error handler
app.use((err, req, res, next) => {
  console.error(` ERROR:`, err);
  res.status(500).json({ error: "SERVER_ERROR" });
});

(async () => {
  await connectDB(process.env.MONGO_URI);
  const port = process.env.PORT || 5000;
  app.listen(port, () => console.log(` API on http://localhost:${port}`));
})();

```

3.4. Paleidimas

Code: bash

```

cd ~/projektai/reflectus-alt/backend
npm run seed
npm run dev

```

4) Frontend (React Vite) – mokinio UI

4.1. Sukurti Vite projekta

Ubuntu terminale:

Code: bash

```

cd ~/projektai/reflectus-alt
npm create vite@latest frontend -- --template react
cd frontend
npm i
npm i axios react-router-dom

```

Sukurk `frontend/.env.example`:

Code: env

```
VITE_API_URL=http://localhost:5000
```

Code: bash

```
cp .env.example .env
```

4.2. Frontend kodas

`frontend/src/api/client.js`

Code: js

```
import axios from "axios";

export const api = axios.create({
  baseURL: import.meta.env.VITE_API_URL,
  withCredentials: true
});
```

`frontend/src/pages/Login.jsx`

Code: jsx

```
import { useState } from "react";
import { api } from "../api/client";

export default function Login({ onLogin }) {
  const [email, setEmail] = useState("student@test.lt");
  const [password, setPassword] = useState("Test12345!");
  const [err, setErr] = useState("");

  async function submit(e) {
    e.preventDefault();
    setErr("");
    try {
      const { data } = await api.post("/api/auth/login", { email, password });
      onLogin(data.user);
    } catch (e) {
      setErr(e?.response?.data?.error || "LOGIN_FAILED");
    }
  }

  return (
    <div style={{ maxWidth: 420, margin: "60px auto", padding: 16 }}>
      <h2>Prisijungimas</h2>
      <form onSubmit={submit} style={{ display: "grid", gap: 10 }}>
        <input value={email} onChange={e=>setEmail(e.target.value)} placeholder="El. paštas" />
        <input value={password} onChange={e=>setPassword(e.target.value)} placeholder="Slaptažodis" />
        <button type="submit">Prisijungti</button>
        {err ? <div style={{ color: "crimson" }}>{err}</div> : null}
      </form>
    </div>
  );
}
```

`frontend/src/pages/Dashboard.jsx`

Code: jsx

```

import { useEffect, useState } from "react";
import { api } from "../api/client";
import { Link } from "react-router-dom";

export default function Dashboard({ user, onLogout }) {
  const [today, setToday] = useState(null);
  const [items, setItems] = useState([]);

  async function load() {
    const t = await api.get("/api/reflections/today");
    setToday(t.data);
    const r = await api.get("/api/reflections");
    setItems(r.data.items || []);
  }

  useEffect(() => { load(); }, []);

  return (
    <div style={{ maxWidth: 900, margin: "30px auto", padding: 16 }}>
      <div style={{ display: "flex", justifyContent: "space-between", alignItems: "center" }}>
        <div>
          <h2>Mokinio zona</h2>
          <div>{user.email}</div>
        </div>
        <button onClick={onLogout}>Atsijungti</button>
      </div>

      <div style={{ marginTop: 20, padding: 12, border: "1px solid #ddd", borderRadius: 8 }}>
        <h3>Šiandien ({today?.dateKey})</h3>
        {today?.item ? (
          <div>
            <div>[] Refleksija pateikta</div>
            <div to={`/r/${today.item._id}`}>Peržiūrėti</Link>
          </div>
        ) : (
          <div>
            <div>[] Refleksijos dar nėra</div>
            <Link to="/new">Sukurti refleksiją</Link>
          </div>
        )}
      </div>

      <div style={{ marginTop: 20 }}>
        <h3>Istorija</h3>
        <div style={{ display: "grid", gap: 8 }}>
          {items.map(x => (
            <Link key={x._id} to={`/r/${x._id}`}>
              <div><b>x.dateKey</b> • Mood: {x.mood}/5</div>
              <div style={{ opacity: 0.8 }}>{x.answers?.learned?.slice(0, 80)}...</div>
            </Link>
          ))}
        {items.length === 0 ? <div>Kol kas nėra įrašų.</div> : null}
      </div>
    </div>
  );
}

```

`frontend/src/pages/NewReflection.jsx`

Code: jsx

```

import { useState } from "react";
import { api } from "../api/client";
import { useNavigate } from "react-router-dom";

export default function NewReflection() {
  const nav = useNavigate();
  const [mood, setMood] = useState(3);
  const [learned, setLearned] = useState("");
  const [hard, setHard] = useState("");
  const [help, setHelp] = useState("");
  const [err, setErr] = useState("");

  async function submit(e) {
    e.preventDefault();
    setErr("");
    try {
      const { data } = await api.post("/api/reflections", {
        mood,
        answers: { learned, hard, help }
      });
      nav(`/r/${data.item._id}`);
    } catch (e) {
      setErr(e?.response?.data?.error || "SAVE_FAILED");
    }
  }

  return (
    <div style={{ maxWidth: 900, margin: "30px auto", padding: 16 }}>
      <h2>Nauja refleksija</h2>
      <form onSubmit={submit} style={{ display: "grid", gap: 10 }}>
        <label>
          Savijauta (1-5):
          <input type="number" min="1" max="5" value={mood} onChange={e=>setMood(Number(e.target.value))}>
        </label>
        <textarea value={learned} onChange={e=>setLearned(e.target.value)} rows={4} placeholder="Ką buvo išvystytas?">
        <textarea value={hard} onChange={e=>setHard(e.target.value)} rows={4} placeholder="Kas buvo labiausiai?">
        <textarea value={help} onChange={e=>setHelp(e.target.value)} rows={4} placeholder="Ko man reikėjo padėti?">
        <button type="submit">Pateikti</button>
        {err ? <div style={{ color: "crimson" }}>{err}</div> : null}
      </form>
    </div>
  );
}

```

`frontend/src/pages/ReflectionDetails.jsx`

Code: jsx

```

import { useEffect, useState } from "react";
import { api } from "../api/client";
import { Link, useParams } from "react-router-dom";

export default function ReflectionDetails() {
  const { id } = useParams();
  const [item, setItem] = useState(null);

  useEffect(() => {
    api.get(`/api/reflections/${id}`).then(r => setItem(r.data.item));
  }, [id]);

  if (!item) return <div style={{ padding: 16 }}>Kraunama...</div>

  return (
    <div style={{ maxWidth: 900, margin: "30px auto", padding: 16 }}>
      <Link to="/"> Atgal</Link>
      <h2>{item.dateKey}</h2>
      <div>Mood: {item.mood}/5</div>

      <div style={{ marginTop: 16, display: "grid", gap: 10 }}>
        <div><b>Ką išmokau:</b><div>{item.answers.learned}</div></div>
        <div><b>Kas buvo sunku:</b><div>{item.answers.hard}</div></div>
        <div><b>Ko reikia:</b><div>{item.answers.help}</div></div>
        {item.teacherComment ? (
          <div style={{ marginTop: 10, padding: 10, border: "1px solid #ddd", borderRadius: 8 }}>
            <b>Mokytojo komentaras:</b>
            <div>{item.teacherComment}</div>
          </div>
        ) : null}
      </div>
    </div>
  );
}

```

`frontend/src/App.jsx`

Code: jsx

```

import { useEffect, useState } from "react";
import { BrowserRouter, Routes, Route, Navigate } from "react-router-dom";
import { api } from "./api/client";
import Login from "./pages/Login";
import Dashboard from "./pages/Dashboard";
import NewReflection from "./pages/NewReflection";
import ReflectionDetails from "./pages/ReflectionDetails";

export default function App() {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  async function loadMe() {
    try {
      const { data } = await api.get("/api/auth/me");
      setUser(data.user);
    } catch {
      setUser(null);
    } finally {
      setLoading(false);
    }
  }

  useEffect(() => { loadMe(); }, []);

  async function logout() {
    await api.post("/api/auth/logout");
    setUser(null);
  }

  if (loading) return <div style={{ padding: 16 }}>Kraunama...</div>

  return (
    <BrowserRouter>
      <Routes>
        <Route path="/login" element={user ? <Navigate to="/" /> : <Login onLogin={setUser} />} />
        <Route path="/" element={user ? <Dashboard user={user} onLogout={logout} /> : <Navigate to="/login" />} />
        <Route path="/new" element={user ? <NewReflection /> : <Navigate to="/login" />} />
        <Route path="/r/:id" element={user ? <ReflectionDetails /> : <Navigate to="/login" />} />
      </Routes>
    </BrowserRouter>
  );
}

```

4.3. Paleidimas

Frontend:

Code: bash

```

cd ~/projektai/reflectus-alt/frontend
npm run dev

```

Atidaryk naršyklėje:

- Frontend: `http://localhost:5173`
- Backend health: `http://localhost:5000/health`

Test prisijungimas:

- email: `student@test.lt`
- pass: `Test12345!`

5) Kas jau “šiek tiek veikia” po šitų žingsnių

- mokinys prisijungia

- mato ar šiandien jau pateikė refleksiją
- pateikia refleksiją
- mato istoriją ir detaļę

6) Kitas minimalus žingsnis (kad atitiktų Reflectus nuotraukas)

Įkelk tas **programėlės idėjos apžvalgos** nuotraukas čia (vienu ar keliais pranešimais).

Tada:

- 1) perpiešiu mokinio UI struktūrą 1:1 (kortelės, mygtukai, skyriai),
- 2) pritaikysiu tekstus ir laukus tiksliai pagal tavo ekranus,
- 3) pridėsiu "mokytojo komentarų" srautą / "užduotis" jei jos ten pavaizduotos.

9. Vartotojas

sukurk pdf faila is viso siyo pokalbio