Ordem a fazer:

Modelos e dados de teste

- Garantir que os models novos estão criados:
  - `PopUpNotifications`, `InterestTypes`, `UserPreferences`, `InterestPopupConfig`, `UserPopupState`.
- Criar 1–2 documentos de teste em cada coleção (via script ou Mongo Compass) para validar que as coleções existem e os campos batem certo.

2. Service core: obter popups para o aluno

- Criar `services/popupNotifications.service.js` (ou nome parecido).
- Implementar função `getPopupsForUser(userId, roles)` que:
  - Lê `UserPreferences` para saber os interesses do aluno.
  - Lê `InterestPopupConfig` para ver se há campanha de interesses ativa para esses roles.
  - Lê `PopUpNotifications` filtrando por: ativo, datas válidas e categorias ∈ interesses.
  - Lê `UserPopupState` e remove avisos cujo `dismiss_count >= max_dismiss_count`.
  - Devolve quais os pop-ups a mostrar (incluindo, se aplicável, o pop-up especial de seleção de interesses).

3. Endpoints para o frontend consumir

- Criar controller/route para aluno:
  - `GET /popup-notifications/me` → chama `getPopupsForUser(userId, roles)` e devolve a lista.
  - `POST /user-popup-state/dismiss` → recebe `popup_id` e atualiza `dismiss_count` e `last_dismiss_at` em `UserPopupState`.
  - `GET /user-preferences/me` e `PUT /user-preferences/me` para ler/guardar interesses do aluno.

4. Endpoints de backoffice (admin/web/ON)

- Criar rotas/controllers para gestão no backoffice:
  - `PopUpNotifications`: `GET`, `POST`, `PUT` (e opção de desativar com `ativo = false`).
  - `InterestTypes`: `GET` (ativos), `POST`, `PUT /:codigo`.
  - `InterestPopupConfig`: `GET /active`, `POST`, `PUT /:id`.
- Estes endpoints usam services simples (create/list/update) em ficheiros `services/*`.

5. Integração com frontend/app

- Ligar o frontend (app do aluno) ao endpoint `GET /popup-notifications/me` para mostrar os avisos.
- No clique “Ok entendido” ou “Não mostrar novamente”, chamar `POST /user-popup-state/dismiss` (eventualmente com flag para “não mostrar novamente” tratar logo como `dismiss_count = max_dismiss_count`).
- No ecrã de definições, usar `GET/PUT /user-preferences/me` para gerir interesses.

6. Backoffice UI e vídeo para a professora

- Criar páginas simples na ON/web para:
  - Gerir tipos de interesse.
  - Gerir campanhas (`InterestPopupConfig`).
  - Criar/editar avisos (`PopUpNotifications`, incluindo `max_dismiss_count`, datas, prioridade).
- Gravar vídeo a mostrar:
  - Admin configura interesses/campanha/avisos no backoffice.
  - Aluno entra na app, escolhe interesses, vê avisos limitados em número e tempo (não aparecem infinitamente).

Models:

PopUpNotifications

```
js

import Mongoose from "mongoose";

export const PopUpNotification = Mongoose.model("PopUpNotifications",
Mongoose.Schema({
    titulo: {
        type: String,
        required: true
    },
    descricao: {
        type: String,
        required: true
    },
    categoria: {
        type: String,          // liga com InterestTypes.codigo
        required: true
```

```
    },
    prioridade: {
        type: Number,           // ex: 1, 2, 3...
        required: true
    },
    texto_botao: {
        type: String,
        required: true
    },
    max_dismiss_count: {
        type: Number,
        required: true,
        default: 3
    },
    data_inicio: {
        type: Date,
        required: true
    },
    duracao_dias: {
        type: Number,
        required: true
    },
    data_fim: {
        type: Date              // opcional, pode ser calculado
    },
    ativo: {
        type: Boolean,
        required: true,
        default: true
    }
}, {
    timestamps: true
}));
```

InterestTypes

```
js

export const InterestType = Mongoose.model("InterestTypes", Mongoose.Schema({
    codigo: {
        type: String,
        required: true,
        unique: true
    },
    nome: {
        type: String,
        required: true
    },
    ativo: {
        type: Boolean,
```

```
        required: true,
        default: true
    }
}, {
    timestamps: true
}));
```

UserPreferences

```
js
```

```
export const UserPreference = Mongoose.model("UserPreferences", Mongoose.Schema({
    id_utilizador: {
        type: String,
        required: true,
        unique: true
    },
    interesses: {
        type: [String],     // array de InterestTypes.codigo
        required: true,
        default: []
    },
    last_interest_popup_id: {
        type: String        // liga com InterestPopupConfig._id (string/ObjectId)
    }
}, {
    timestamps: true
}));
```

InterestPopupConfig

```
js
```

```
export const InterestPopupConfig = Mongoose.model("InterestPopupConfigs",
Mongoose.Schema({
    ativo: {
        type: Boolean,
        required: true,
        default: true
    },
    data_inicio: {
        type: Date,
        required: true
    },
    data_fim: {
        type: Date,
        required: true
    },
    roles: {
```

```
        type: [String],
        required: true,
        default: []         // ex: ["aluno"]
    }
}, {
    timestamps: true
}));
```

UserPopupState

```
js

export const UserPopupState = Mongoose.model("UserPopupStates", Mongoose.Schema({
    id_utilizador: {
        type: String,
        required: true
    },
    popup_id: {
        type: Mongoose.Schema.Types.ObjectId,
        ref: "PopUpNotifications",
        required: true
    },
    dismiss_count: {
        type: Number,
        required: true,
        default: 0
    },
    last_dismiss_at: {
        type: Date
    }
}, {
    timestamps: true
}));
```

Services:

`popupNotifications.service.js:`

(
Isto já implementa a lógica de:

- **verificar campanha de interesses ativa,**
- **filtrar avisos por data, interesses, e `max_dismiss_count` por utilizador**

)

import { PopUpNotification } from "../models/PopUpNotifications.js";

```
import { InterestType } from "../models/InterestTypes.js";
import { UserPreference } from "../models/UserPreferences.js";
import { InterestPopupConfig } from "../models/InterestPopupConfig.js";
import { UserPopupState } from "../models/UserPopupState.js";

export async function getPopupsForUser(userId, roles = []) {
  // 1) Buscar preferências do utilizador
  const userPrefs = await UserPreference.findOne({ id_utilizador: userId });

  const interesses = userPrefs?.interesses || [];

  // 2) Ver se existe campanha de popup de interesses ativa para estes roles
  const now = new Date();
  const activeInterestConfig = await InterestPopupConfig.findOne({
    ativo: true,
    data_inicio: { $lte: now },
    data_fim: { $gte: now },
    roles: { $in: roles }
  });

  // Decidir se é preciso mostrar o popup de interesses
  let shouldShowInterestPopup = false;
  if (activeInterestConfig) {
    // ainda não respondeu a esta campanha
    if (!userPrefs || String(userPrefs.last_interest_popup_id) !==
String(activeInterestConfig._id)) {
      shouldShowInterestPopup = true;
    }
  }

  // 3) Buscar avisos normais (PopUpNotifications) filtrados por datas e interesses
  const popupQuery = {
    ativo: true,
    data_inicio: { $lte: now },
    // se tiver data_fim usa, senão calcula com duracao_dias
    $or: [
      { data_fim: { $exists: true, $gte: now } },
      { data_fim: { $exists: false } }
    ]
  };

  if (interesses.length > 0) {
    popupQuery.categoria = { $in: interesses };
  }

  const allPopups = await PopUpNotification.find(popupQuery);

  // 4) Buscar estados de interação do utilizador
```

```
  const states = await UserPopupState.find({
    id_utilizador: userId,
    popup_id: { $in: allPopups.map(p => p._id) }
  });

  const stateMap = new Map();
  states.forEach(s => stateMap.set(String(s.popup_id), s));

  // 5) Filtrar avisos que já atingiram max_dismiss_count
  const popupsToShow = allPopups.filter(p => {
    const s = stateMap.get(String(p._id));
    if (!s) return true; // nunca visto
    return s.dismiss_count < p.max_dismiss_count;
  });

  return {
    shouldShowInterestPopup,
    interestPopupConfig: activeInterestConfig || null,
    popups: popupsToShow
  };
}
```

Routes:

adicionar no index em cada routes novo:

```
import popupNotificationsRoutes from "./popupNotifications.routes.js";
...
routes.use("/popupNotifications", popupNotificationsRoutes);
```

routes/popupNotifications.routes.js

```
import { Router } from "express";

import middlewares from "../middlewares/index.js";
import controllers from "../controllers/index.js";

const router = Router();

const auth = middlewares.auth;
const popupNotifications = controllers.popupNotifications;
```

```
// rota para a APP (frontoffice) obter os popups do próprio
utilizador
router.post(
  "/popupNotifications/me",
  auth.verifyToken,
  popupNotifications.getForMe
);

export default router;
```

Controllers:

adicionar no index em cada controller novo:

```
import popupNotifications from "./popupNotifications.controller.js";
...
popupNotifications,
```

controllers/popupNotifications.controller.js

```
import { getPopupsForUser } from
"../services/popupNotifications.service.js";

const getForMe = async (req, res) => {
  try {
    const { user } = req.user;          // vem do auth.verifyToken
    const userId = user.id_utilizador;  // campo usado na BD
    const roles = [user.role || "DEFAULT"];

    const result = await getPopupsForUser(userId, roles);

    return res.status(200).json({ data: result });
  } catch (err) {
    console.log(err);
    return res.status(500).json({ data: null });
  }
};

export default { getForMe };
```