▯ SCHOLARBENCH — Plans & Policies System

Backend + Frontend Architecture Documentation (For Developers)

▯ 1. OVERVIEW

Scholarbench will introduce three plans:

- Free
- Plus
- Premium

Each plan defines what features a user can access and limitations (example: daily question limit).

We implement a **Policy-Based Access Control (PBAC)** system:

- Every plan has policies.
- Some roles have additional overrides.
- Specific users can have personalized overrides.

The backend resolves final policies, and the frontend uses those policies to enable/disable UI features.

▯ 2. BACKEND IMPLEMENTATION

2.1 Backend Tech

- Node.js / Express / Sequelize (or Spring Boot if needed)
- MySQL

▯ 2.2 Database Schema

Table: plans

Column	Type	Description
id	int	primary key
name	varchar	Free / Plus / Premium
price	decimal	Monthly price
billingPeriod	enum	MONTHLY / YEARLY

Table: resources

Column	Type	Description
id	int	primary key
key	varchar	Unique key (UPLOAD_PDF, AI_SUMMARY)
description	text	Human-readable

Table: plan_policies

Column	Type	Description
id	int	
planId	int	
resourceId	int	
value	varchar	(boolean / number / json)

Table: role_policies

Column	Type	Description
id	int	
roleId	int	
resourceId	int	
value	varchar	

Table: user_policies

Column	Type	Description
id	int	
userId	int	
resourceId	int	
value	varchar	

users table (add two columns)

Column	Type
roleId	int
planId	int

▯ 2.3 Policy Resolution Logic (VERY IMPORTANT)

Final policy = the **highest priority** match:

1. User-level override → highest
2. Role-level override
3. Plan default policy

Backend Function: getPolicyValue

```
async function getPolicyValue(userId, resourceKey) {
   const user = await User.findByPk(userId, { include: [Role, Plan] });

   // 1. User override
   const userPol = await UserPolicy.findPolicy(user.id, resourceKey);
   if (userPol) return userPol.value;

   // 2. Role override
   const rolePol = await RolePolicy.findPolicy(user.roleId, resourceKey);
   if (rolePol) return rolePol.value;

   // 3. Plan default
   const planPol = await PlanPolicy.findPolicy(user.planId, resourceKey);
   if (planPol) return planPol.value;

   return null;
}
```

▯ 2.4 Backend Login Response

When a user logs in, backend returns:

```
{
  "user": {
    "id": 1,
    "name": "John",
    "plan": "Free",
    "role": "Student"
  },
  "policies": {
    "UPLOAD_PDF": false,
    "AI_SUMMARY": true,
    "QUESTION_LIMIT_DAILY": 10,
    "STORAGE_LIMIT_MB": 500
  }
}
```

Frontend uses this object for all restrictions.

▯ 2.5 Protecting Backend Routes

Example: Protect PDF Upload API

```
router.post("/pdf/upload", async (req, res) => {
    if (!(await getPolicyValue(req.user.id, "UPLOAD_PDF"))) {
        return res.status(403).json({
            message: "Please upgrade your plan to upload PDFs"
        });
    }

    // ... actual logic
});
```

▯ 3. FRONTEND IMPLEMENTATION

3.1 Frontend Tech

- React / Next.js
- Zustand or React Context for global state

▯ 3.2 Frontend Login Flow

1. User logs in → Frontend calls `/auth/me`
2. Set global state:

```
setUser(response.data.user);
setPolicies(response.data.policies);
```

▯ 3.3 usePolicy() Hook

```
export function usePolicy() {
  const { policies } = useAuthContext();

  function can(resourceKey: string) {
    return policies[resourceKey] === true;
  }

  function limit(resourceKey: string) {
    return policies[resourceKey] ?? null;
  }

  return { can, limit };
}
```

▯ 3.4 Policy Guard Component

```
export default function PolicyGuard({ resource, children, fallback }) {
  const { can } = usePolicy();

  if (!can(resource)) {
    return fallback ?? <UpgradePage />;
  }

  return <>{children}</>;
}
```

Usage:

```
<PolicyGuard resource="AI_SUMMARY">
  <AISummaryPage />
</PolicyGuard>
```

▯ 3.5 Restricting UI Elements

```
const { can } = usePolicy();

<Button disabled={!can("UPLOAD_PDF")}>
  Upload PDF
</Button>

{!can("UPLOAD_PDF") && <UpgradeBanner feature="PDF Upload" />}
```

▯ 3.6 Handling Numeric Limits

```
const questionsToday = user.questionUsageToday;
const limit = limit("QUESTION_LIMIT_DAILY");

if (questionsToday >= limit) {
  showUpgradeModal();
}
```

Backend also re-checks limits.

▯ 4. OPTIONAL BUT RECOMMENDED

4.1 Feature Flags

For beta testing: Backend sends:

```
"featureFlags": {
  "BETA_FLASHCARDS": true
}
```

Same guard mechanism applies.

4.2 Upgrade Page

Show features of each plan based on policy differences.

4.3 Admin Dashboard for Setting Policies

Junior developer does not need this now, but architecture supports it easily.

▯ 5. WHAT DEV MUST BUILD

Backend

- Create database tables
- Write policy resolution function
- Add middleware to protect routes
- Return user + final policies on login

- Implement plan, role, user policy seeders

Frontend

- Store policies globally
- Implement usePolicy hook
- Implement PolicyGuard component
- Protect pages + UI components
- Show upgrade UI on failure