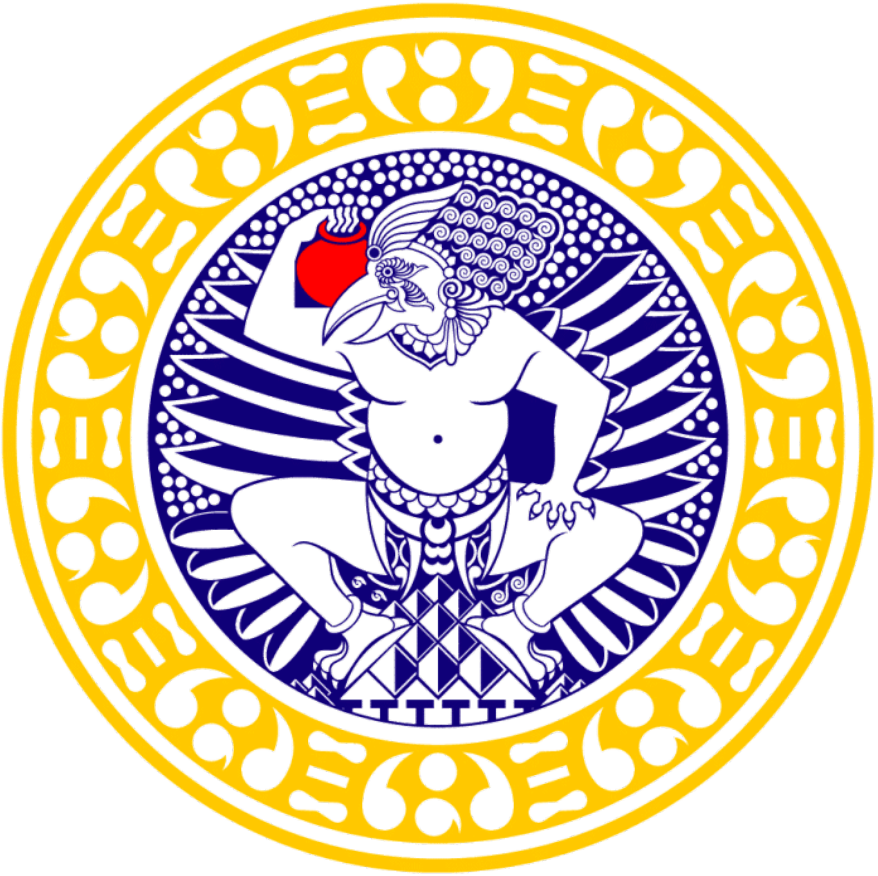
LAPORAN UTS PEMROGRAMAN BACKEND LANJUT PRAKTIKUM



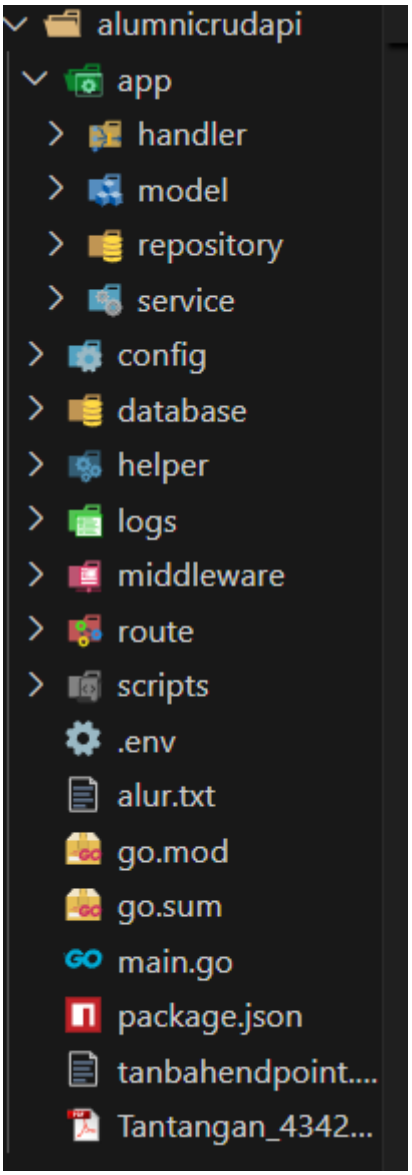
Disusun Oleh:

Pedja Rafsanjani (434231099)

FAKULTAS VOKASI
UNIVERSITAS AIRLANGGA

2025

STRUKTUR



Handler/pekerjaan_handler.go

```
func (h *PekerjaanHandler) ListTrash(c *fiber.Ctx) error {
    username := c.Locals("username").(string)
    role := c.Locals("role").(string)
    userID := c.Locals("user_id").(int)

    page, _ := strconv.Atoi(c.Query("page", "1"))
    limit, _ := strconv.Atoi(c.Query("limit", "10"))
    search := c.Query("search", "")

    log.Printf("User %s (%s) accessing GET /pekerjaan/trash
page=%d limit=%d", username, role, page, limit)
```

```
    list, err := h.pekerjaanService.ListTrash(search, page,
limit, userID, role)
    if err != nil {
        return helper.ErrorResponse(c,
fiber.StatusInternalServerError, "Failed to get trash data")
    }

    return c.JSON(fiber.Map{
        "success": true,
        "message": "Trash data retrieved successfully",
        "data":    list,
    })
}

func (h *PekerjaanHandler) RestorePekerjaan(c *fiber.Ctx) error {
    username := c.Locals("username").(string)
    role := c.Locals("role").(string)
    userID := c.Locals("user_id").(int)
    id, err := strconv.Atoi(c.Params("id"))
    if err != nil {
        return helper.ErrorResponse(c, fiber.StatusBadRequest,
"Invalid ID parameter")
    }

    log.Printf("User %s (%s) restoring pekerjaan ID %d",
username, role, id)

    if err := h.pekerjaanService.RestorePekerjaan(id, userID,
role); err != nil {
        switch err.Error() {
        case "pekerjaan not found":
            return helper.ErrorResponse(c, fiber.StatusNotFound,
"Pekerjaan not found in trash")
        case "access denied: you can only restore your own
pekerjaan":
            return helper.ErrorResponse(c, fiber.StatusForbidden,
"Access denied: you can only restore your own pekerjaan")
        default:
            return helper.ErrorResponse(c,
fiber.StatusInternalServerError, "Failed to restore pekerjaan")
        }
    }
```

```
    return helper.SuccessResponse(c, "Pekerjaan restored
successfully", nil)
}

func (h *PekerjaanHandler) HardDeletePekerjaan(c *fiber.Ctx)
error {
    username := c.Locals("username").(string)
    role := c.Locals("role").(string)
    userID := c.Locals("user_id").(int)
    id, err := strconv.Atoi(c.Params("id"))
    if err != nil {
        return helper.ErrorResponse(c, fiber.StatusBadRequest,
"Invalid ID parameter")
    }

    log.Printf("User %s (%s) hard-deleting pekerjaan ID %d",
username, role, id)

    if err := h.pekerjaanService.HardDeletePekerjaan(id, userID,
role); err != nil {
        if err == sql.ErrNoRows {
            // not found or not in trash/unauthorized
            return helper.ErrorResponse(c, fiber.StatusNotFound,
"Pekerjaan not found in trash or access denied")
        }
        return helper.ErrorResponse(c,
fiber.StatusInternalServerError, "Failed to hard delete
pekerjaan")
    }

    return helper.SuccessResponse(c, "Pekerjaan permanently
deleted", nil)
}
```

Repository/pekerjaan_repository.go

```go
func (r *pekerjaanRepository) ListTrashAdmin(search string, limit, offset int) ([]model.PekerjaanAlumni, error) {
    q := `
        SELECT id, alumni_id, nama_perusahaan, posisi_jabatan, bidang_industri, lokasi_kerja,
               gaji_range, tanggal_mulai_kerja, tanggal_selesai_kerja, status_pekerjaan,
               deskripsi_pekerjaan, is_deleted, deleted_at, deleted_by, created_at, updated_at
        FROM pekerjaan_alumni
        WHERE is_deleted = TRUE
          AND ($1 = '' OR nama_perusahaan ILIKE '%'||$1||'%' OR posisi_jabatan ILIKE '%'||$1||'%' OR bidang_industri ILIKE '%'||$1||'%' OR lokasi_kerja ILIKE '%'||$1||'%')
        ORDER BY deleted_at DESC NULLS LAST
        LIMIT $2 OFFSET $3
    `
    rows, err := database.DB.Query(q, search, limit, offset)
    if err != nil {
        return nil, err
    }
    defer rows.Close()

    var out []model.PekerjaanAlumni
    for rows.Next() {
        var p model.PekerjaanAlumni
        if err := rows.Scan(
            &p.ID, &p.AlumniID, &p.NamaPerusahaan, &p.PosisiJabatan, &p.BidangIndustri, &p.LokasiKerja,
            &p.GajiRange, &p.TanggalMulaiKerja, &p.TanggalSelesaiKerja, &p.StatusPekerjaan,
            &p.DeskripsiPekerjaan, &p.IsDeleted, &p.DeletedAt, &p.DeletedBy, &p.CreatedAt, &p.UpdatedAt,
        ); err != nil {
            return nil, err
        }
        out = append(out, p)
    }
    return out, nil
}

func (r *pekerjaanRepository) ListTrashUser(userID int, search string, limit, offset int) ([]model.PekerjaanAlumni, error) {
    q := `
        SELECT pa.id, pa.alumni_id, pa.nama_perusahaan, pa.posisi_jabatan, pa.bidang_industri, pa.lokasi_kerja,
               pa.gaji_range, pa.tanggal_mulai_kerja, pa.tanggal_selesai_kerja, pa.status_pekerjaan,
               pa.deskripsi_pekerjaan, pa.is_deleted, pa.deleted_at, pa.deleted_by, pa.created_at, pa.updated_at
        FROM pekerjaan_alumni pa
        JOIN alumni a ON a.id = pa.alumni_id
        WHERE pa.is_deleted = TRUE
          AND a.user_id = $1
          AND ($2 = '' OR pa.nama_perusahaan ILIKE '%'||$2||'%' OR pa.posisi_jabatan ILIKE '%'||$2||'%' OR pa.bidang_industri ILIKE '%'||$2||'%' OR pa.lokasi_kerja ILIKE '%'||$2||'%')
        ORDER BY pa.deleted_at DESC NULLS LAST
        LIMIT $3 OFFSET $4
    `
    rows, err := database.DB.Query(q, userID, search, limit, offset)
    if err != nil {
        return nil, err
    }
    defer rows.Close()

    var out []model.PekerjaanAlumni
    for rows.Next() {
        var p model.PekerjaanAlumni
        if err := rows.Scan(
            &p.ID, &p.AlumniID, &p.NamaPerusahaan, &p.PosisiJabatan, &p.BidangIndustri, &p.LokasiKerja,
            &p.GajiRange, &p.TanggalMulaiKerja, &p.TanggalSelesaiKerja, &p.StatusPekerjaan,
            &p.DeskripsiPekerjaan, &p.IsDeleted, &p.DeletedAt, &p.DeletedBy, &p.CreatedAt, &p.UpdatedAt,
        ); err != nil {
            return nil, err
        }
        out = append(out, p)
    }
    return out, nil
}

func (r *pekerjaanRepository) Restore(id int) error {
    q :=
        UPDATE pekerjaan_alumni
        SET is_deleted = FALSE, deleted_at = NULL, deleted_by = NULL, updated_at = NOW()
        WHERE id = $1 AND is_deleted = TRUE
    `
    res, err := database.DB.Exec(q, id)
    if err != nil {
        return err
    }
    n, _ := res.RowsAffected()
    if n == 0 {
        return sql.ErrNoRows
    }
    return nil
}

func (r *pekerjaanRepository) HardDeleteAdmin(id int) error {
    q := `DELETE FROM pekerjaan_alumni WHERE id = $1 AND is_deleted = TRUE`
    res, err := database.DB.Exec(q, id)
    if err != nil {
        return err
    }
    n, _ := res.RowsAffected()
    if n == 0 {
        return sql.ErrNoRows
    }
    return nil
}

func (r *pekerjaanRepository) HardDeleteUser(id int, userID int) error {
    q := `
        DELETE FROM pekerjaan_alumni pa
        USING alumni a
        WHERE pa.id = $1
          AND pa.is_deleted = TRUE
          AND a.id = pa.alumni_id
          AND a.user_id = $2
    `
    res, err := database.DB.Exec(q, id, userID)
    if err != nil {
        return err
    }
    n, _ := res.RowsAffected()
    if n == 0 {
        return sql.ErrNoRows
    }
    return nil
}

```

Service/Pekerjaan_service.go

```go
func (s *pekerjaanService) ListTrash(search string, page, limit
int, userID int, role string) ([]model.PekerjaanAlumni, error) {
    if page < 1 {
        page = 1
    }
    if limit < 1 || limit > 100 {
        limit = 10
    }
    offset := (page - 1) * limit

    if role == "admin" {
        return s.pekerjaanRepo.ListTrashAdmin(search, limit,
offset)
    }
    return s.pekerjaanRepo.ListTrashUser(userID, search, limit,
offset)
}

func (s *pekerjaanService) RestorePekerjaan(id int, userID int,
role string) error {
    // check ownership for non-admin
    if role != "admin" {
        ownerID, err := s.pekerjaanRepo.GetOwnerUserID(id)
        if err != nil {
            if err == sql.ErrNoRows {
                return errors.New("pekerjaan not found")
            }
            return err
        }
        if ownerID != userID {
            return errors.New("access denied: you can only
restore your own pekerjaan")
        }
    }
    return s.pekerjaanRepo.Restore(id)
}

func (s *pekerjaanService) HardDeletePekerjaan(id int, userID
int, role string) error {
    if role == "admin" {
        return s.pekerjaanRepo.HardDeleteAdmin(id)
    }
```

```
    // user only own
    return s.pekerjaanRepo.HardDeleteUser(id, userID)
}
```

Route.go

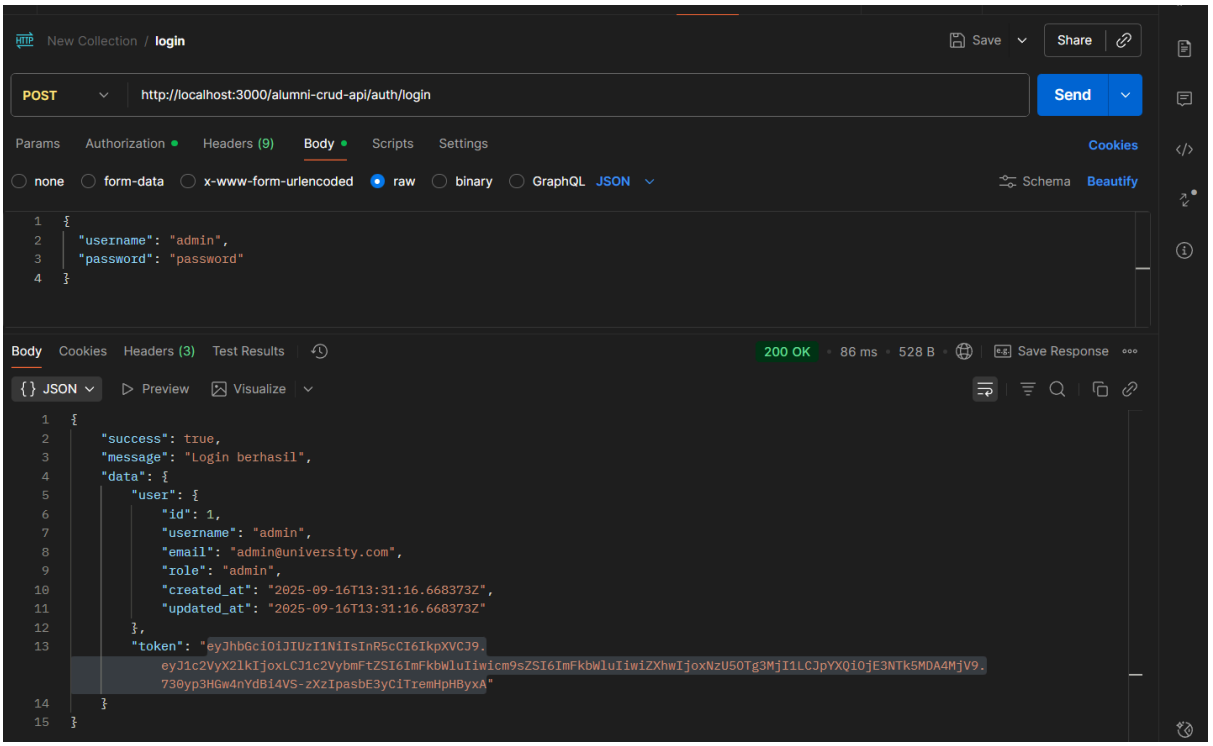
```
package route

import (
    "alumni-crud-api/app/handler"
    "alumni-crud-api/middleware"

    "github.com/gofiber/fiber/v2"
)

func SetupRoutes(fiberApp *fiber.App, alumniHandler
*handler.AlumniHandler, pekerjaanHandler
*handler.PekerjaanHandler, authHandler *handler.AuthHandler) {
    fiberApp.Get("/", func(c *fiber.Ctx) error {
        return c.JSON(fiber.Map{
            "success": true,
            "message": "Alumni CRUD API Server",
            "version": "1.0.0",
            "endpoints": fiber.Map{
                "auth": fiber.Map{
                    "POST /alumni-crud-api/auth/login":  "Login
to get access token",
                    "GET /alumni-crud-api/auth/profile": "Get
user profile (requires auth)",
                },
                "alumni": fiber.Map{
                    "GET /alumni-crud-api/alumni":         "Get
all alumni (requires auth)",
                    "GET /alumni-crud-api/alumni/:id":     "Get
alumni by ID (requires auth)",
                    "POST /alumni-crud-api/alumni":        "Create
new alumni (admin only)",
                    "PUT /alumni-crud-api/alumni/:id":     "Update
alumni (admin only)",
                    "DELETE /alumni-crud-api/alumni/:id": "Delete
alumni (admin only)",
                },
                "pekerjaan": fiber.Map{
```

```
                    "GET /alumni-crud-api/pekerjaan":
"Get all pekerjaan (requires auth)",
                    "GET /alumni-crud-api/pekerjaan/:id":
"Get pekerjaan by ID (requires auth)",
                    "GET
/alumni-crud-api/pekerjaan/alumni/:alumni_id":  "Get pekerjaan by
alumni ID (admin only)",
                    "POST /alumni-crud-api/pekerjaan":
"Create new pekerjaan (admin only)",
                    "PUT /alumni-crud-api/pekerjaan/:id":
"Update pekerjaan (admin only)",
                    "DELETE /alumni-crud-api/pekerjaan/:id":
"Delete pekerjaan (admin only)",
                    "PATCH
/alumni-crud-api/pekerjaan/:id/soft-delete":  "Soft delete
pekerjaan (admin or owner)",
                    "GET /alumni-crud-api/pekerjaan/trash":
"List trash (admin: all, user: own)",
                    "PATCH
/alumni-crud-api/pekerjaan/:id/restore":      "Restore from trash
(admin or owner)",
                    "DELETE
/alumni-crud-api/pekerjaan/:id/hard-delete": "Hard delete from
trash (admin or owner)",
                },
            },
        })
    })

    fiberApp.Get("/health", func(c *fiber.Ctx) error {
        return c.JSON(fiber.Map{
            "success": true,
            "status":  "healthy",
            "message": "Server is running properly",
        })
    })

    api := fiberApp.Group("/alumni-crud-api")

    auth := api.Group("/auth")
    auth.Post("/login", authHandler.Login)

    protected := api.Group("", middleware.AuthRequired())
```

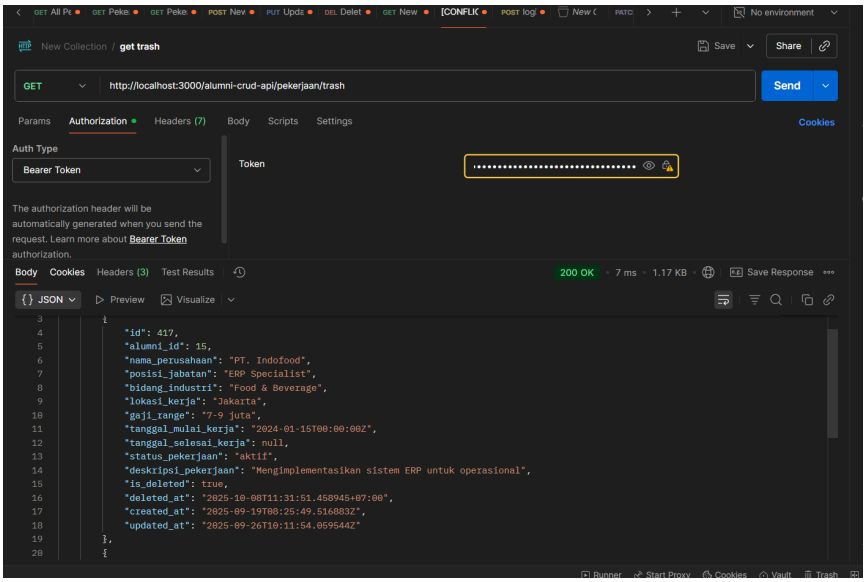
```
    protected.Get("/auth/profile", authHandler.GetProfile)

    alumni := protected.Group("/alumni")
    alumni.Get("/", alumniHandler.GetAllAlumni)
// Admin and User can access
    alumni.Get("/:id", alumniHandler.GetAlumniByID)
// Admin and User can access
    alumni.Post("/", middleware.AdminOnly(),
alumniHandler.CreateAlumni)      // Admin only
    alumni.Put("/:id", middleware.AdminOnly(),
alumniHandler.UpdateAlumni)    // Admin only
    alumni.Delete("/:id", middleware.AdminOnly(),
alumniHandler.DeleteAlumni) // Admin only

    pekerjaan := protected.Group("/pekerjaan")
    pekerjaan.Get("/", pekerjaanHandler.GetAllPekerjaan)
    // Register static routes before dynamic parameter routes so
they don't get treated as :id
    // Allow authenticated users to access their own trash;
service filters admin vs user
    pekerjaan.Get("/trash", pekerjaanHandler.ListTrash)
    pekerjaan.Get("/alumni/:alumni_id", middleware.AdminOnly(),
pekerjaanHandler.GetPekerjaanByAlumniID)
    pekerjaan.Get("/:id", pekerjaanHandler.GetPekerjaanByID)
    pekerjaan.Post("/", middleware.AdminOnly(),
pekerjaanHandler.CreatePekerjaan)
    pekerjaan.Put("/:id", middleware.AdminOnly(),
pekerjaanHandler.UpdatePekerjaan)
    pekerjaan.Delete("/:id", middleware.AdminOnly(),
pekerjaanHandler.DeletePekerjaan)
    pekerjaan.Patch("/:id/soft-delete",
pekerjaanHandler.SoftDeletePekerjaan)
    pekerjaan.Patch("/:id/restore",
pekerjaanHandler.RestorePekerjaan)
    pekerjaan.Delete("/:id/hard-delete",
pekerjaanHandler.HardDeletePekerjaan)
}
```

1. MEMBUAT ENDPOINT TRASH

a. Admin

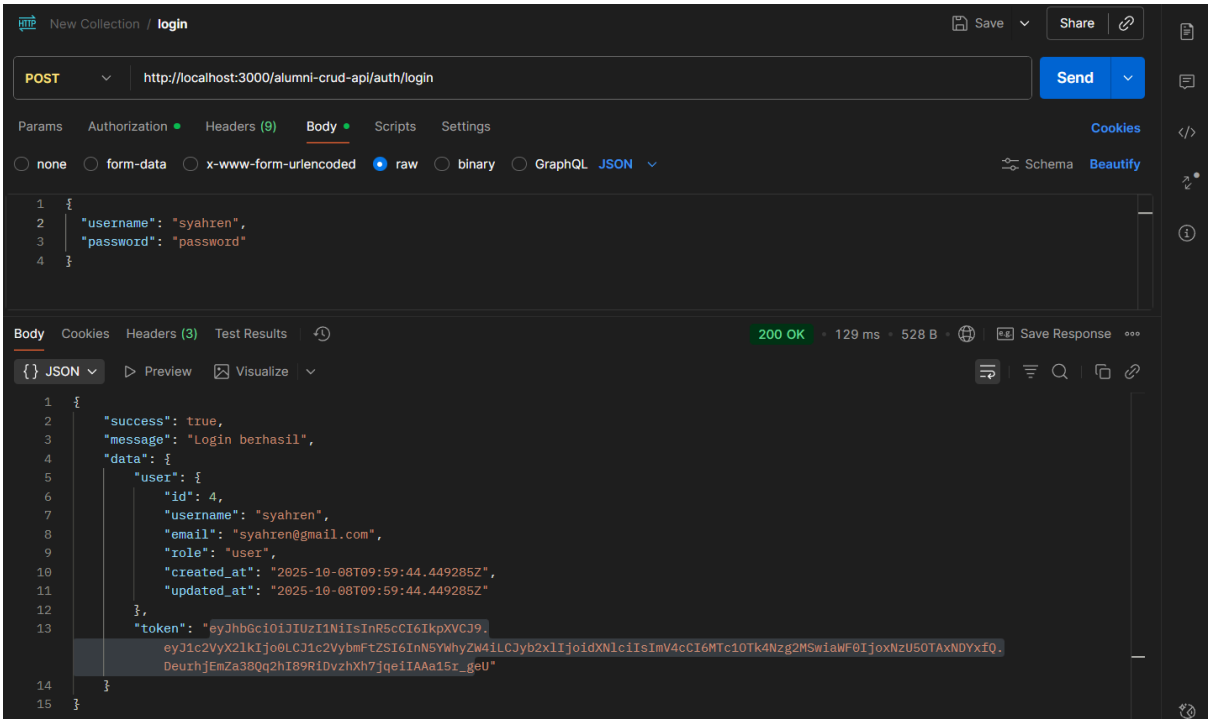


Admin login terlebih dahulu dan mendapat token

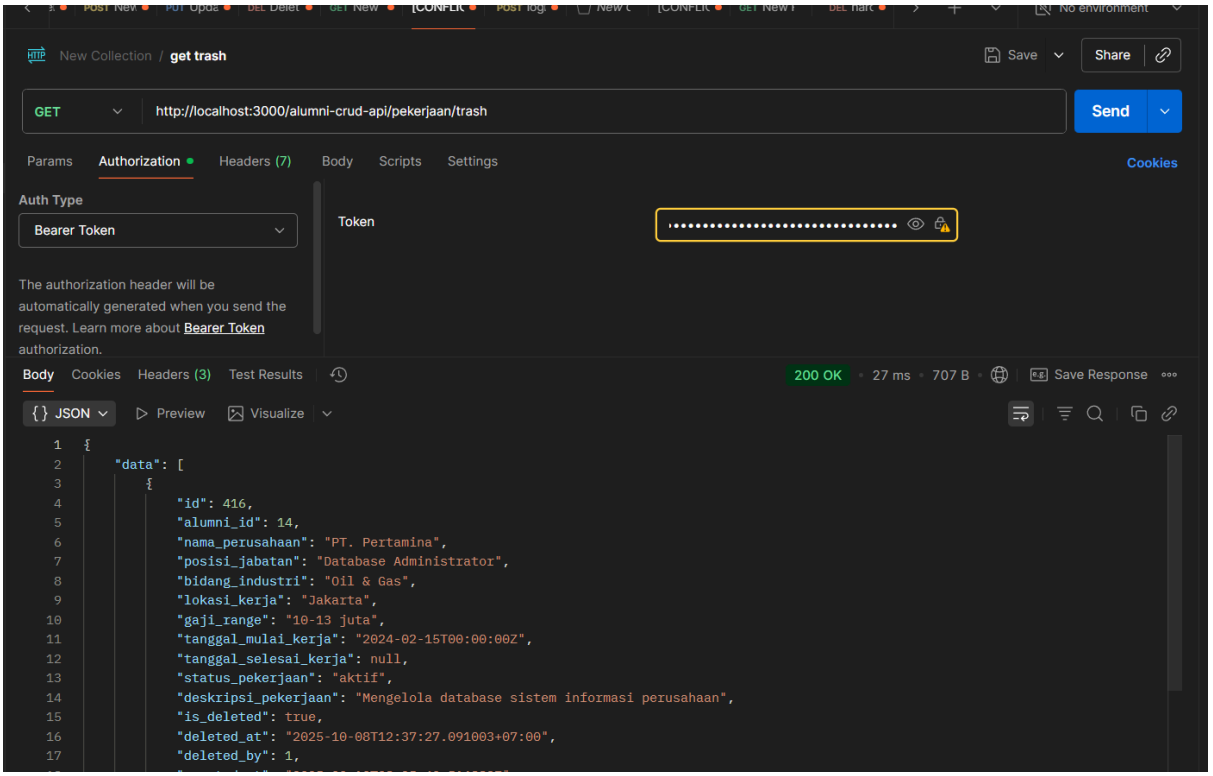


Admin dapat menampilkan semua trash yang ada, trash sendiri adalah data pekerjaan yang sudah di soft delete

b. User



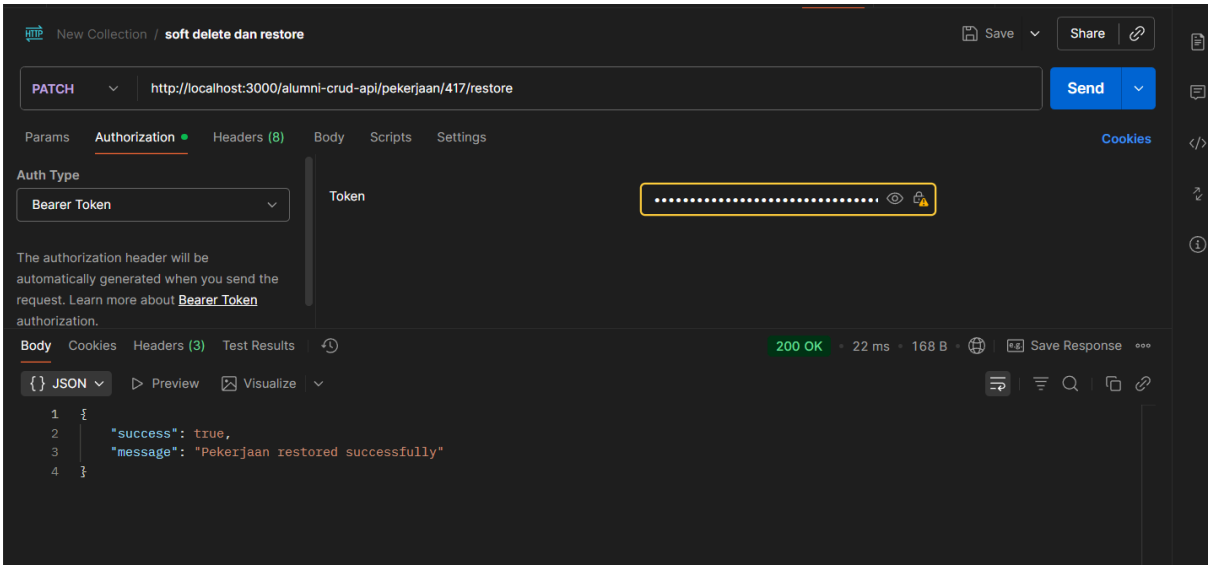
Login sebagai user Syahren dan mendapatkan token



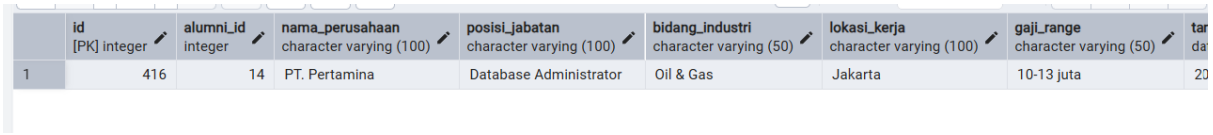
User bisa melihat Trashnya sendiri

2. MEMBUAT ENDPOINT UNTUK RESTORE

a. Admin



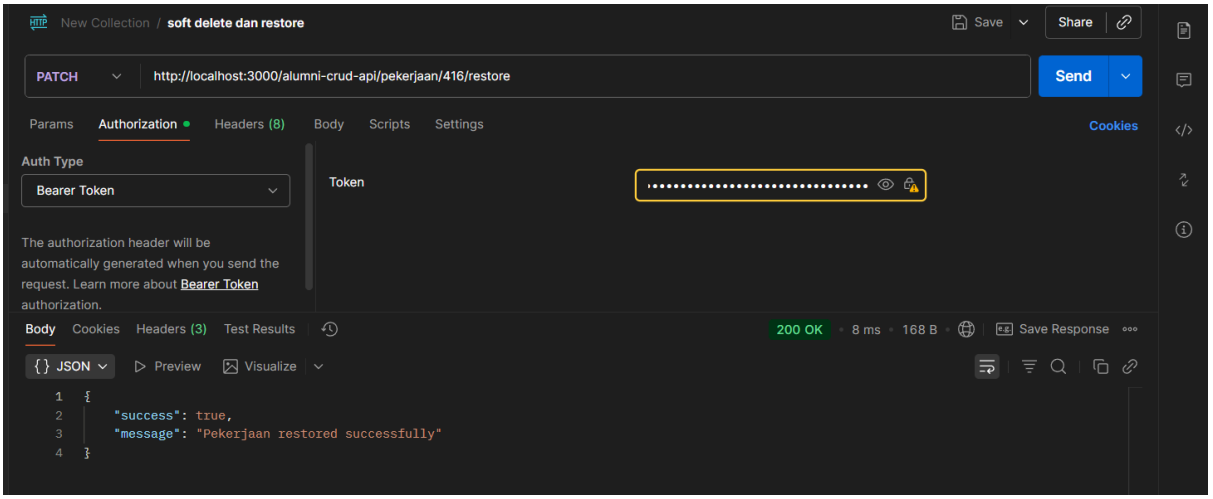
Admin dapat restore trash yang ada



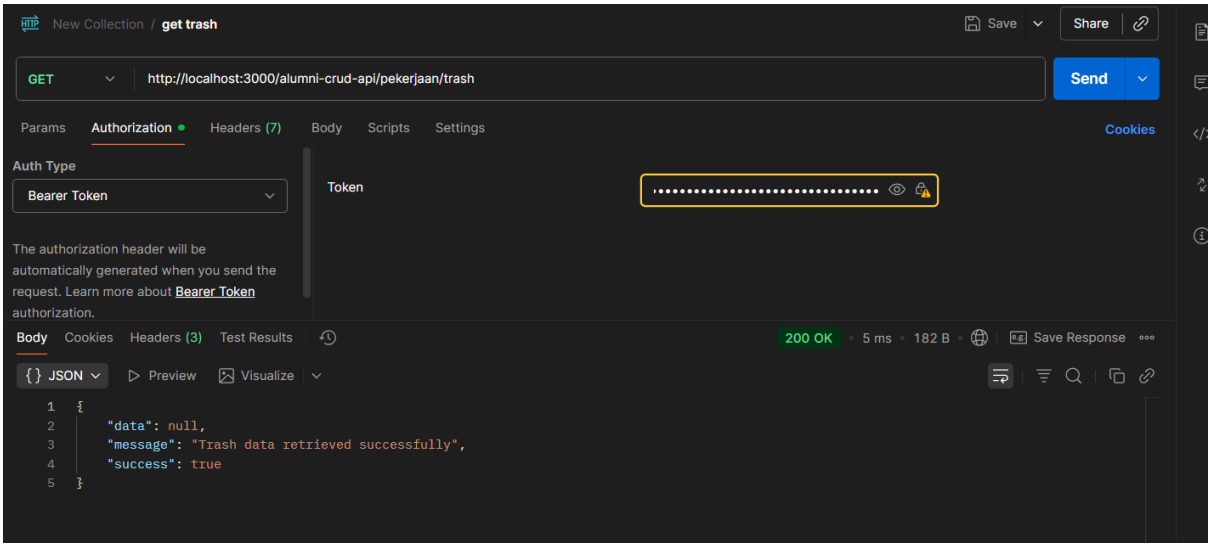
	id [PK] integer	alumni_id integer	nama_perusahaan character varying (100)	posisi_jabatan character varying (100)	bidang_industri character varying (50)	lokasi_kerja character varying (100)	gaji_range character varying (50)	tar da
1	416	14	PT. Pertamina	Database Administrator	Oil & Gas	Jakarta	10-13 juta	20

Setelah dilakukan restore, saat cek trash maka data yang sudah di restore hilang di trash, namun masih ada di data pekerjaan

b. User



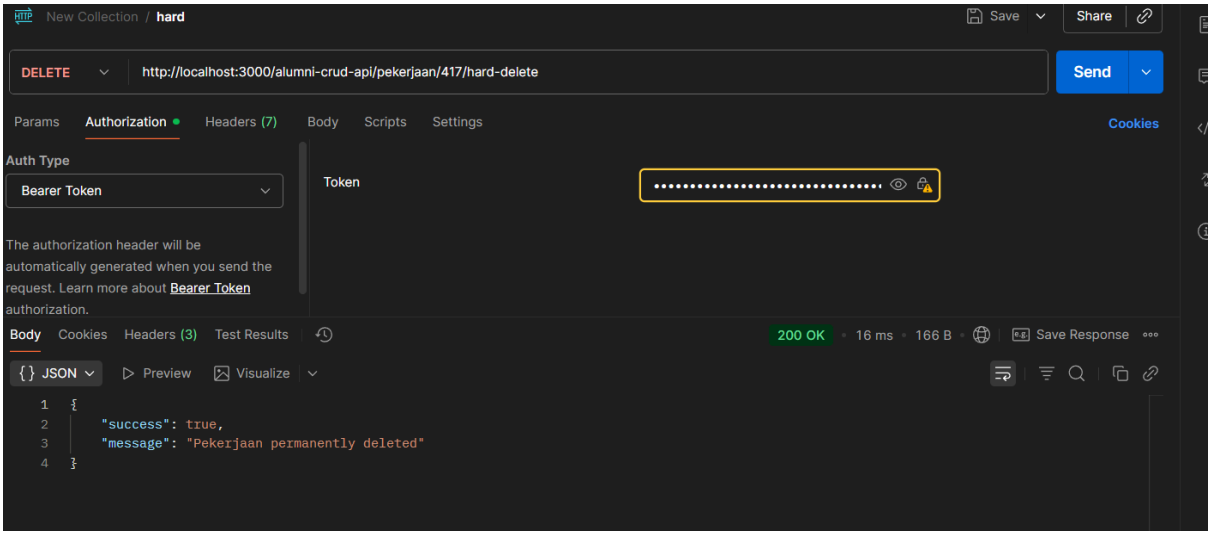
User dapat melakukan restore dirinya sendiri



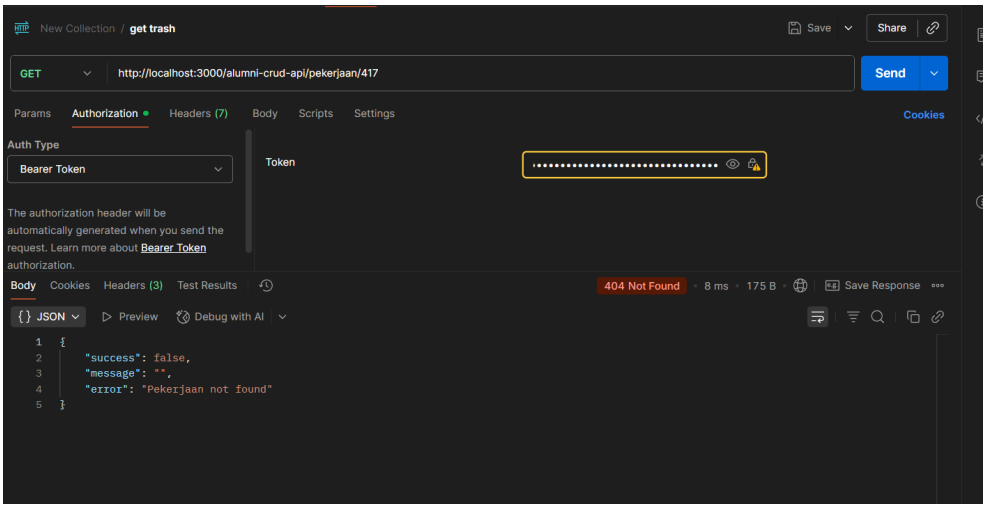
Saat dicek, data yang sudah di restore sudah hilang dari Trash

3. MEMBUAT ENDPOINT UNTUK HARD DELETE

a. Admin

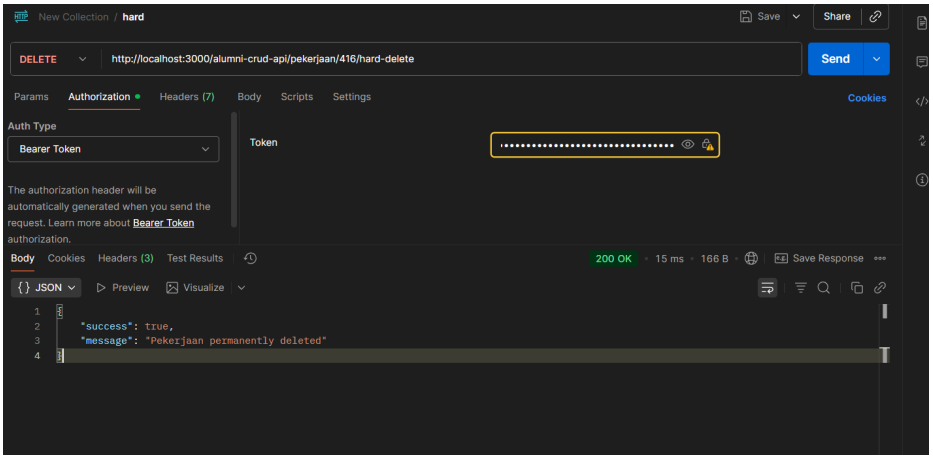


Admin dapat melakukan Hard Delete

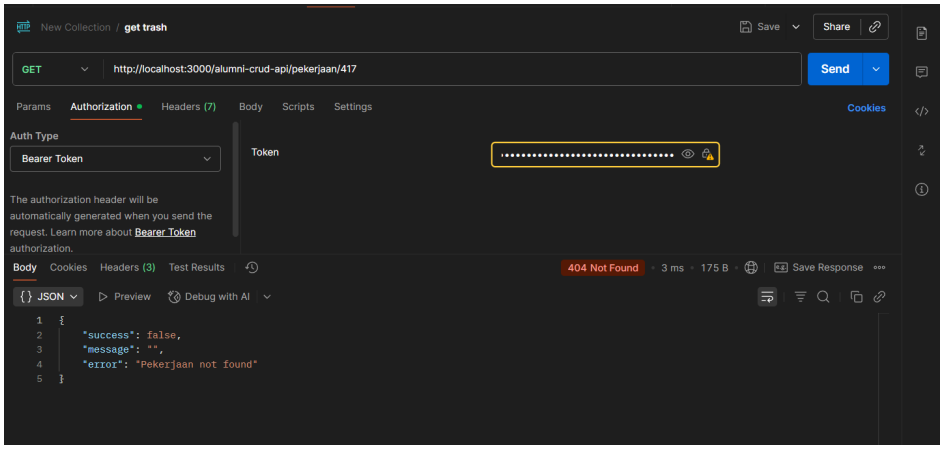


Pekerjaan sudah dihapus dan hilang dari database

b. User



User dapat melakukan Hard delete



Setelah melakukan hard delete lalu dicek, data pekerjaan sudah hilang permanen dari database