GITGOOD

Help gitGood, gitBetter

SERVICES CONTRACT

DEMO 3

WEATHER TO WEAR

EPI-USE®

KYLE LIEBENBERG DIYA BUDHIA ALISHA PERUMAL	IBRAHIM SAID TAYLOR SERGEL

Table of Contents

Authentication Service

OVERVIEW

The Authentication Service handles **user registration, login, deletion, and authenticated profile retrieval.** It follows a REST over HTTP architecture, with JSON as the data exchange format and JWT for authentication.

COMMUNICATION PROTOCOL

- Protocol: HTTP/1.1
- Architecture: REST
- Content Type: application/json; charset=utf-8
- Authentication: Bearer JWT in Authorization header
- Internal DB Access: Prisma ORM to a relational database (likely PostgreSQL)

DATA FORMAT

- Request & Response format: JSON
- Property naming convention: camelCase
- Timestamps: ISO-8601 UTC format
- IDs: String UUID format

SECURITY

- Token Type: JWT (HS256)
- Token Expiry: 1 hour
- Token Payload:

json

```
{
  "id": "string",
  "email": "string",
  "iat": 1730000000,
  "exp": 1730003600
}
```

PASSWORD POLICY

Must match regex:
^(?=.*[a-z])(?=.*[A-Z])(?=.*\W).{8,}$

Which indicates passwords must have:

≥ 8 characters
one lowercase
one uppercase
one special character

ERROR HANDLING

Error Response Format

json

```
{
  "error": "Description of the error"
}
```

Status Codes Used:

- **400** - *Bad Request*
  - Missing required fields, password policy fail, user already exists, or user not found (delete)
- **401** - *Unauthorized*
  - Missing token or login failure
- **403** - *Forbidden*
  - Invalid token

ENDPOINTS

POST api/auth/signup

Registers a new user.
Middleware: signupPasswordValidation

Request Body:

```
{
  "name": "Jane Doe",
  "email": "jane@example.com",
  "password": "Str0ng!Pass"
}
```

201 - Created Response:

```
{
  "message": "User registered
successfully",
  "token": "<jwt>",
  "user": {
    "id": "uuid",
    "name": "Jane Doe",
    "email": "jane@example.com"
  }
}
```

Possible Errors:

- **400** - Missing required fields
- **400** - User already exists
- **400** - Password must have the required security requirements

POST api/auth/login

Authenticates a user.

Request Body:

```
{
  "email": "jane@example.com",
  "password": "Str0ng!Pass"
}
```

201 - Created Response:

```
{
  "message": "Login successful",
  "token": "<jwt>",
  "user": {
    "id": "uuid",
    "name": "Jane Doe",
    "email": "jane@example.com"
  }
}
```

Possible Errors:

- **400** - Missing email or password
- **401** - User not found
- **401** - Invalid Credentials

DELETE api/auth/users/:id

Deletes a user by ID (Protected).

Headers:

```
Authorization: Bearer <jwt>
```

200 OK RESPONSE

```
{
  "message": "User deleted
successfully",
  "user": {
    "id": "uuid",
    "name": "Jane Doe",
    "email": "jane@example.com"
  }
}
```

Possible Errors:

- **401** - Missing token
- **403** - Invalid token
- **400** - User not found

GET api/auth/profile

Returns the authenticated user's profile (Protected).

Headers:

```
Authorization: Bearer <jwt>
```

200 OK RESPONSE

```
{
  "message": "You are authenticated!",
  "user": {
    "id": "uuid",
    "email": "jane@example.com"
  }
}
```

Possible Errors:

- **401** - Missing token
- **403** - Invalid token
- **400** - User not found

TESTABILITY

- **Unit tests**
  - For registerUser, loginUser, and removeUser with Prisma mocked.
- **Integration tests**
  - Using Supertest for all endpoints.
- **JWT tests**
  - Verify expiry and signature validation in middleware.
- **Contract tests**
  - Generated from OpenAPI spec to ensure compatibility between front-end and back-end.

Closet Service

OVERVIEW

The Closet Service manages a user's wardrobe items. It supports single and batch uploads with background removal and color extraction, retrieval (all/by category), update, deletion, and favourite toggling. All endpoints are REST over HTTP/1.1, JWT-protected, and exchange JSON (except uploads, which use multipart/form-data).

DOMAIN TYPES (PRISMA ENUMS)

```
enum Category {
  SHIRT
  HOODIE
  PANTS
  SHORTS
  SHOES
  TSHIRT
  LONGSLEEVE
  SWEATER
  JACKET
  JEANS
  BEANIE
  HAT
  SCARF
  GLOVES
  RAINCOAT
  UMBRELLA
}
```

```
enum LayerCategory {
  base_top
  base_bottom
  mid_top
  mid_bottom
  outerwear
  footwear
  headwear
  accessory
}
```

```
enum Material {
  Cotton
  Wool
  Polyester
  Leather
  Nylon
  Fleece
}
```

```
enum Style {
  Formal
  Casual
  Athletic
  Party
  Business
  Outdoor
}
```

NOTE

Error **400** if invalid ENUM value is used

SECURITY

- JWT (HS256)
- validated by authenticateToken middleware.
- **401** - missing token
- **403** - invalid token

ERROR HANDLING

Error Response Format

json

```
{ "message": string }
```

Status Codes Used:

- **400** - *Bad Request*
  - Invalid/missing fields, invalid enums, batch shape errors, missing file(s)
- **401** - *Unauthorized*
  - No/invalid user context
- **403** - *Forbidden*
  - Invalid token
- **404** - Not Found

TIMEOUTS & LIMITS

- Client request timeout: 30s (uploads), 15s (read/update/delete)
- Background removal service timeout: 20s
- Color extraction service timeout: 10s
- Max upload size: single image ≤ 10 MB; batch total ≤ 50 MB
- Allowed file types: image/jpeg, image/png
- Retries:
  - Safe to retry GET endpoints.
  - POST /upload and /upload/batch: only retry if the first attempt definitively failed (network), or use an idempotency key (recommended).
  - DELETE is idempotent.

EXTERNAL MICROSERVICE CONTRACTS

- **Background Removal** (BG_REMOVAL_URL)
  - Method
    - POST
  - Content-Type
    - multipart/form-data
  - Body: field file
    - binary image
  - Response
    - 200 with binary (PNG with background removed)
  - Errors
    - 4xx/5xx; on error, Closet Service returns 500 with message Failed to remove background from image.
- **Color Extraction** (COLOR_EXTRACT_URL)
  - Method
    - POST
  - Content-Type
    - multipart/form-data
  - Body
    - field file: binary image (PNG after background removal)
  - Response 200 (JSON)

```
{ "colors": ["#112233", "#445566", "#778899"] }
```

ENDPOINTS

POST closet/upload

Single image uplaod

Auth: required
Content-Type: multipart/form-data
Form fields:

- image (file) - **required**
- category (string, Category) - **required**
- layerCategory (string, LayerCategory) - **required**
- Optional metadata:
  - colorHex (string, eg. #rrggbb)
  - warmthFactor (number)
  - waterproof ("true" | "false")
  - style (string, Style)
  - material (string, Material)

Request Body:

```
{
  "id": "uuid",
  "category": "TOPS",
  "imageUrl": "/uploads/file_no_bg.png",
  "createdAt": "2025-08-12T10:15:30.000Z",
  "colorHex": "#1a2b3c",
  "warmthFactor": 2,
  "waterproof": true,
  "style": "CASUAL",
  "material": "COTTON"
}
```

201 - Created Response:

```
{
  "id": "uuid",
  "category": "TOPS",
  "imageUrl": "/uploads/file_no_bg.png",
  "createdAt": "2025-08-12T10:15:30.000Z",
  "colorHex": "#1a2b3c",
  "warmthFactor": 2,
  "waterproof": true,
  "style": "CASUAL",
  "material": "COTTON"
}
```

Possible Errors:

- **400** - No file
  - { "message": "No file provided" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **400** - Invalid Category
  - { "message": "Invalid category: <value>" }
- **500** - Internal Server Error
  - { "message": "Failed to remove background from image" }

POST closet/upload/batch

Batch uplaod

Auth: required
Content-Type: multipart/form-data
Contract: You send files plus an items JSON array. Each item refers to a file by its fieldname.
Form fields - Used for every file:

- image (file) - **required**
- category (string, Category) - **required**
- layerCategory (string, LayerCategory) - **required**
- Optional metadata:
  - colorHex (string, eg. #rrggbb)
  - warmthFactor (number)
  - waterproof ("true" | "false")
  - style (string, Style)
  - material (string, Material)

Request Body:

```
{
  "id": "uuid",
  "category": "TOPS",
  "imageUrl": "/uploads/file_no_bg.png",
  "createdAt": "2025-08-12T10:15:30.000Z",
  "colorHex": "#1a2b3c",
  "warmthFactor": 2,
  "waterproof": true,
  "style": "CASUAL",
  "material": "COTTON"
}
```

201 - Created Response:

```
{
  "id": "uuid",
  "category": "TOPS",
  "imageUrl": "/uploads/file_no_bg.png",
  "createdAt": "2025-08-12T10:15:30.000Z",
  "colorHex": "#1a2b3c",
  "warmthFactor": 2,
  "waterproof": true,
  "style": "CASUAL",
  "material": "COTTON"
}
```

Possible Errors:

- **400** - No file
  - { "message": "No file provided" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **400** - Invalid Category
  - { "message": "Invalid category: <value>" }
- **500** - Internal Server Error
  - { "message": "Failed to remove background from image" }

GET closet/category/:category

List by category

Auth: required

200 OK RESPONSE


```
{
  "id": "uuid",
  "category": "TOPS",
  "imageUrl": "/uploads/file_no_bg.png",
  "createdAt": "2025-08-12T10:15:30.000Z",
  "colorHex": "#1a2b3c",
  "warmthFactor": 2,
  "waterproof": true,
  "style": "CASUAL",
  "material": "COTTON"
}
```


Possible Errors:

- **400** - Invalid Category
- **401** - Unauthorized
  - { "message": "Unauthorized" }

GET /all

List all items for a user

Auth: required

200 OK RESPONSE


```
[
  {
    "id": "uuid1",
    "category": "TOPS",
    "imageUrl": "/uploads/file_no_bg.png",
    "createdAt": "2025-08-12T10:15:30.000Z",
    "colorHex": "#1a2b3c",
    "dominantColors": ["#1a2b3c", "#aabbcc", "#334455"],
    "warmthFactor": 2,
    "waterproof": true,
    "style": "CASUAL",
    "material": "COTTON",
    "favourite": false
  },
    {
    "id": "uuid2",
    "category": "TOPS",
    "imageUrl": "/uploads/file_no_bg.png",
    "createdAt": "2025-08-12T10:15:30.000Z",
    "colorHex": "#1a2b3c",
    "dominantColors": ["#1a2b3c", "#aabbcc", "#334455"],
    "warmthFactor": 2,
    "waterproof": true,
    "style": "CASUAL",
    "material": "COTTON",
    "favourite": false
  }
]
```


Possible Errors:

- **400** - Invalid Category
- **401** - Unauthorized
  - { "message": "Unauthorized" }

DELETE closet/:id
Delete item

Auth: required
204 No Content
Possible Errors:

- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **404** - Item not found

PATCH closet/:id
Update the item metadata

Auth: required

Request Body:

```
{
  "category": "TOPS",
  "layerCategory": "BASE",
  "colorHex": "#abcdef",
  "warmthFactor": 3,
  "waterproof": true,
  "style": "CASUAL",
  "material": "COTTON"
}
```

201 - Created Response (with the new update):

```
{
  "id": "uuid",
  "category": "TOPS",
  "imageUrl": "/uploads/file_no_bg.png",
  "createdAt": "2025-08-12T10:15:30.000Z",
  "colorHex": "#1a2b3c",
  "warmthFactor": 2,
  "waterproof": true,
  "style": "CASUAL",
  "material": "COTTON"
}
```

Possible Errors:

- **400** - Invalid field/enums
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **400** - Invalid Category
  - { "message": "Invalid category: <value>" }
- **404** - Item not found

PATCH closet/:id/favourite

Toggle favourite

Auth: required

200 OK

```
{ "id": "uuid", "favourite": true }
```

Possible Errors:

- **400** - Invalid field/enums
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **400** - Invalid Category
  - { "message": "Invalid category: <value>" }
- **404** - Item not found

VALIDATION RULES

- *category, layerCategory, style, material* must be valid enum values.
- *warmthFactor* must be numeric.
- *waterproof* accepts boolean or string "true"|"false"
- For batch uploads, every item must have a matching file via filename (fieldname), else **400**.

TESTABILITY

- ***Unit***
  - *mock Prisma + microservice calls (axios).*
- ***Integration***
  - *Supertest for each route:*
- ***Auth guard***
  - *missing/invalid token → 401/403.*
- ***Upload single/batch***
  - *happy path 201; no file / bad items → 400.*
- ***Get by category/all***
  - *200 with arrays; unauthorized 401.*
- ***Delete/update/toggle***
  - *200/204 happy path; non-existent 404.*

Events Service

OVERVIEW

The Events Service lets authenticated users create, read, update, and delete personal events. On create/update it fetches short-range weather and stores a summary alongside the event. Structure, tone, and conventions match your prior service contracts.

COMMUNICATION PROTOCOL

- Protocol: HTTP/1.1
- Architecture: REST
- Content Type: application/json; charset=utf-8
- Authentication: Bearer JWT in Authorization header
- Internal DB Access: Prisma ORM to a relational database (likely PostgreSQL)

DATA FORMAT

- Requests/Responses: JSON, camelCase properties
- Dates: ISO-8601 (UTC) strings for dateFrom, dateTo
- IDs: String UUIDs
- Weather field: persisted/returned as a JSON-stringified value in this version (see examples). Consider parsing to an object in a future version.

SECURITY

- JWT (HS256) enforced by shared auth middleware.
- Missing token → 401, invalid token → 403. (Aligned with other services.)

VALIDATION RULES

- Create: name, location, dateFrom, dateTo, style are required.
- Date window: dateFrom must not be in the past (today 00:00) and must be within next 3 days.
- Update: body must include id; at least one of name, location, dateFrom, dateTo, style. If location or dateFrom changes, the same date window checks apply and weather must be fetchable.

ERROR HANDLING

Error Response Format

json

```
{ "message": string }
```

Status Codes Used:

- **400** - *Bad Request*
  - invalid/missing fields, invalid enums, ownership violations (e.g., adding an item not owned by user).
- **401** - *Unauthorized*
  - missing/invalid user context.
- **403** - *Forbidden*
  - invalid JWT (middleware).
- **404** - *Not Found*
  - outfit/outfit-item not found (or not owned).
- **429** - *Too Many Requests*
  - if rate limiting enabled.
- **5xx**
  - server errors.

TIMEOUTS & LIMITS

- **Client timeout:**
  - 15s for reads/updates/deletes; 20s for create (weather calls).
- **Weather call timeout:**
  - ~8–10s per call.
- **Rate limits:**
  - 60 writes / 15 min / user; 300 reads / 15 min / user.

EXTERNAL DEPENDENCY CONTRACT — WEATHER

- **Function used:**
  - getWeatherByDay(location, yyyy-mm-dd)
- **Create:**
  - per-day failures are tolerated (summary null).
- **Update:**
  - failure to fetch for the new start date → 400 "Weather forecast unavailable…".

ENDPOINTS

GET events/getEvents

Get all Events

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

200 - OK Response:


```
[
  {
    "id": "b1a2-…",
    "name": "Matric Dance",
    "location": "Hatfield, Pretoria",
    "weather": "[{\"date\":\"2025-08-18\",\"summary\":{\"avgTemp\":20}}]",
    "dateFrom": "2025-08-18T17:00:00.000Z",
    "dateTo": "2025-08-18T23:00:00.000Z",
    "style": "FORMAL"
  }
]
```


GET events/getEvent

Get single Event by Id

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Body:

```
{ "id": "b1a2-…" }
```

200 - OK Response:


```
[
  {
    "id": "b1a2-…",
    "name": "Matric Dance",
    "location": "Hatfield, Pretoria",
    "weather": "[{\"date\":\"2025-08-18\",\"summary\":{\"avgTemp\":20}}]",
    "dateFrom": "2025-08-18T17:00:00.000Z",
    "dateTo": "2025-08-18T23:00:00.000Z",
    "style": "FORMAL"
  }
]
```


Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **400** - Invalid Category
  - { "message": "Invalid category: <value>" }
- **500** - Internal Server Error

POST events/createEvent

Create an Event

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Body:

```
{
  "name": "Soccer Tournament",
  "location": "Arcadia, Pretoria",
  "dateFrom": "2025-08-18T08:00:00.000Z",
  "dateTo": "2025-08-19T18:00:00.000Z",
  "style": "SPORT"
}
```

201 - Created Response:

```
{
  "id": "e4c5-…",
  "name": "Soccer Tournament",
  "location": "Arcadia, Pretoria",
  "weather": "[{\"date\":\"2025-08-18\",\"summary\":{\"avgTemp\":21}}, {\"date\":\"2025-08-19\",\"summary\":null}]",
  "dateFrom": "2025-08-18T08:00:00.000Z",
  "dateTo": "2025-08-19T18:00:00.000Z",
  "style": "SPORT"
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **400** - Invalid Category
  - { "message": "Invalid category: <value>" }
- **500** - Internal Server Error

PUT events/updateEvent

Edit an Event

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Body:

```
{
  "id": "e4c5-…",
  "name": "Updated Name",
  "location": "Menlyn, Pretoria",
  "dateFrom": "2025-08-19T09:00:00.000Z",
  "dateTo": "2025-08-19T20:00:00.000Z",
  "style": "CASUAL"
}
```

200 - OK Response:

```
{
  "id": "e4c5-…",
  "name": "Updated Name",
  "location": "Menlyn, Pretoria",
  "weather": "{\"avgTemp\":22,\"minTemp\":12,\"willRain\":false}",
  "dateFrom": "2025-08-19T09:00:00.000Z",
  "dateTo": "2025-08-19T20:00:00.000Z",
  "style": "CASUAL"
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **400** - Invalid Category
  - { "message": "Invalid category: <value>" }
- **500** - Internal Server Error

DELETE events/deleteEvent

Delete an Event

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Body:

```
{ "id": "b1a2-…" }
```

TESTABILITY

- ***Unit***
  - *mock Prisma + microservice calls (axios).*
- ***Integration***
  - *Supertest for each route:*
- ***Auth guard***
  - *missing/invalid token → 401/403.*
- ***Upload single/batch***
  - *happy path 201; no file / bad items → 400.*
- ***Get by category/all***
  - *200 with arrays; unauthorized 401.*
- ***Delete/update/toggle***
  - *200/204 happy path; non-existent 404.*

Outfits Service

OVERVIEW

The Outfits Service lets authenticated users create, read, update, delete, and favourite outfits composed of closet items. It also exposes endpoints to manage items within an outfit and to recommend outfits using weather-aware, color-harmony and K-NN personalization logic. Architecture is REST over HTTP/1.1 with JSON payloads and JWT authentication. This section mirrors the structure and conventions used in your Authentication/Closet docs

COMMUNICATION PROTOCOL

- **Protocol**
  - HTTP/1.1
- **Architecture**
  - REST
- **Content Type**
  - application/json; charset=utf-8
- **Authentication**
  - Bearer JWT in Authorization header (all endpoints require auth)

DATA FORMAT

- Requests/Responses: JSON
- Property naming: camelCase
- Dates: ISO-8601 UTC (e.g., 2025-08-14T10:30:00Z)
- IDs: String UUIDs (as used by Prisma)
- These mirror the earlier services for consistency.

SECURITY

- JWT (HS256) validated by shared authenticateToken middleware.
- Missing token → 401; invalid token → 403.

DOMAIN TYPES

```
enum LayerCategory {
  base_top
  base_bottom
  mid_top
  mid_bottom
  outerwear
  footwear
  headwear
  accessory
}
```

```
enum OverallStyle {
  Formal
  Casual
  Athletic
  Party
  Business
  Outdoor
}
```

```
model Outfit {
  id             String       @id @default(uuid())
  userId         String
  user           User         @relation(fields: [userId], references: [id])
  createdAt      DateTime     @default(now())
  weatherSummary String?
  warmthRating   Int
  waterproof     Boolean
  userRating     Int?
  overallStyle   OverallStyle
  outfitItems    OutfitItem[]
  favourite      Boolean      @default(false)
  eventId        String?
  event          Event?       @relation(fields: [eventId], references:
}id])
```

ERROR HANDLING

Error Response Format

json

```
{ "message": string }
```

Status Codes Used:

- **400** - *Bad Request*
  - invalid/missing fields, invalid enums, ownership violations (e.g., adding an item not owned by user).
- **401** - *Unauthorized*
  - missing/invalid user context.
- **403** - *Forbidden*
  - invalid JWT (middleware).
- **404** - *Not Found*
  - outfit/outfit-item not found (or not owned).
- **429** - *Too Many Requests*
  - if rate limiting enabled.
- **5xx**
  - server errors.

TIMEOUTS AND LIMITS

- Client request timeout: 15s for reads/updates/deletes.
- Create & recommend: up to 20s (more computation/DB aggregation).
- Rate limits (recommended): 60 writes / 15 min / user; 300 reads / 15 min / user.
- These limits mirror the approach used for Closet timeouts/limits.

ENDPOINTS

POST /
Create an outfit

Auth: required
201 - Created Response:

```
{
  "outfitItems": [
    { "closetItemId": "uuid", "layerCategory": "base_top", "sortOrder": 1 }
  ],
  "warmthRating": 8,
  "waterproof": false,
  "overallStyle": "Casual",
  "weatherSummary": "{\"avgTemp\":20,\"minTemp\":12,...}",
  "userRating": 4
}
```

Possible Errors:

- **400** - Invalid field/enums

Validations:

- outfitItems must be a non-empty array.
- overallStyle and each layerCategory must be valid enums.
- All closetItemIds must belong to the user

GET /
Get all Outfits

Auth: required
200 OK - array of outfits with outfitItems (and each item's closetItem).

Possible Errors:

- **401**

GET /:id
Get Outfit by ID

Auth: required
200 OK - full outfit with nested items & closetItem details.

Possible Errors:

- **401**
- **404** - Not found
  - Outfit isn't owned or not found

PUT /:id

Update outfit (metadata and/or items)

Auth: required
200 OK - full outfit with nested items & closetItem details.

```
{
  "userRating": 5,
  "overallStyle": "Formal",
  "outfitItems": [
    { "closetItemId": "uuid", "layerCategory": "mid_top", "sortOrder": 1 }
  ]
}
```

Possible Errors:

- **401**
- **404** - Not found
  - Outfit isn't owned or not found
- **400** - Invalid field/enums

Validations:

- If outfitItems provided, the service replaces all current items with the new list.
- Ownership verified for the outfit and each referenced closet item.

DELETE outfit/:id

Delete Outfit

Auth: required
200 OK

```
{ "success": true }
```

Possible Errors:

- **401**
- **404** - Not found
  - Outfit isn't owned or not found

GET /:id/items

Get items for outfit

Auth: required
200 OK - array of outfit items (each includes closetItem).

Possible Errors:

- **401**
- **404** - Not found
  - Outfit isn't owned or not found

Social Service

OVERVIEW

The Social Service enables authenticated users to create and manage posts, comments, likes, and follows, with optional linkage to closet items and embedded weather snapshots. Architecture is REST over HTTP/1.1, responses are JSON; inputs are documented as JSON bodies

COMMUNICATION PROTOCOL

- Protocol: HTTP/1.1
- Architecture: REST
- Content Type: application/json; charset=utf-8
- Authentication: Bearer JWT in Authorization header
- Internal DB Access: Prisma ORM to a relational database (likely PostgreSQL)

DATA FORMAT

- Requests/Responses: JSON, camelCase properties
- Dates: ISO-8601 (UTC) strings for dateFrom, dateTo
- IDs: String UUIDs
- Weather field: persisted/returned as a JSON-stringified value in this version (see examples). Consider parsing to an object in a future version.

SECURITY

- JWT (HS256) enforced by shared auth middleware.
- Missing token → 401, invalid token → 403. (Aligned with other services.)

VALIDATION RULES

- Auth required for all mutations; most reads also require auth (as implemented).
- Create post: either upload an image (multipart/form-data) or omit; body supports caption, location, weather (object or stringified JSON), and closetItemId.
- Add/update comment: content required (non-empty).
- Likes: user cannot like twice; unlike requires an existing like.
- Follow: cannot follow self; duplicate follows rejected; unfollow requires existing relationship.
- Pagination: limit (default 20), offset (default 0).
- Includes: string array, e.g., ["user","comments","likes"] or ["user"] for likes, when supported by service.

ERROR HANDLING

Error Response Format

json

```
{ "message": string }
```

Status Codes Used:

- **400** - *Bad Request*
  - invalid/missing fields, invalid enums, ownership violations (e.g., adding an item not owned by user).
- **401** - *Unauthorized*
  - missing/invalid user context.
- **403** - *Forbidden*
  - invalid JWT (middleware).
- **404** - *Not Found*
  - outfit/outfit-item not found (or not owned).
- **429** - *Too Many Requests*
  - if rate limiting enabled.
- **5xx**
  - server errors.

TIMEOUTS AND LIMITS

- Client request timeout: 15s
- Media upload (createPost): 30s (multipart)
- Rate limits (example): 60 writes / 15 min / user; 300 reads / 15 min / user
- On rate limit → 429 with Retry-After

ENDPOINTS

POST social/posts

Create a post

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Body:

```
{
  "caption": "At the park",
  "location": "Pretoria",
  "closetItemId": "string-uuid",
  "weather": { "avgTemp": 21, "minTemp": 12, "willRain": false }
  // if using multipart, include file field "image"
}
```

201 - Created Response:

```
{
  "message": "Post created successfully",
  "post": { /* Post */ }
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

GET social/posts

Get all posts

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Body:

```
{
  "limit": 20,
  "offset": 0,
  "include": ["user","likes","comments"]   // optional expansions supported by service
}
```

201 - Created Response:

```
{
  "message": "Posts retrieved successfully",
  "posts": [ /* Post[] */ ],
  "pagination": { "limit": 20, "offset": 0 }
}
```

GET social/posts/:id

Get post by ID

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Body:

```
{
  "include": ["user","likes","comments"]  // optional; controller currently uses ?include=user,likes,...
}
```

201 - Created Response:

```
{
  "message": "Post retrieved successfully",
  "post": { /* Post */ }
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

PUT social/posts/:id

Edit post by ID

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Body:

```
{
  "include": ["user","likes","comments"]  // optional; controller currently uses ?include=user,likes,...
}
```

201 - Created Response:

```
{
  "message": "Post retrieved successfully",
  "post": { /* Post */ }
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

DELETE posts/:id

Delete Post

Auth: required

200 OK

```
{ "success": true }
```

Possible Errors:

- **401**
- **404** - Not found
  - Outfit isn't owned or not found

POST social/posts/:postId/comments

Add comment

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Body:

```
{ "content": "Nice fit!" }
```

201 - Created Response:

```
{
  "message": "Comment added successfully",
  "comment": { /* Comment */ }
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

GET social/posts/:postId/comments

List all comments for post

Auth: not strictly required in controller; follow your policy.

Body:

```
{
  "limit": 20,
  "offset": 0,
  "include": ["user"]     // if supported by service
}
```

201 - Created Response:

```
{
  "message": "Comments retrieved successfully",
  "comments": [ /* Comment[] */ ],
  "pagination": { "limit": 20, "offset": 0 }
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

PUT social/comments/:id

Update comment

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Body:

```
{ "content": "Edited comment" }
```

201 - Created Response:

```
{
  "message": "Comment updated successfully",
  "comment": { /* Comment */ }
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

DELETE comment/:id

Delete comment

Auth: required

200 OK

```
{ "success": true }
```

Possible Errors:

- **401**
- **404** - Not found
  - Outfit isn't owned or not found

POST social/posts/:postId/likes
Like a post

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

200 OK Response

```
{
  "message": "Post liked successfully",
  "like": { /* Like */ }
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

DELETE social/posts/:postId/likes
Unlike a post

Auth: required

200 OK

```
{ "success": true }
```

GET social/posts/:postId/likes
List likes for a post

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Body:

```
{
  "limit": 20,
  "offset": 0,
  "include": ["user"]   // include liker user details
}
```

201 - Created Response:

```
{
  "message": "Likes retrieved successfully",
  "likes": [ /* Like[] */ ],
  "pagination": { "limit": 20, "offset": 0 }
}
```

POST social/users/:userId/follow

Follow a user

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

200 OK Response

```
{
  "message": "User followed successfully",
  "follow": { /* Follow */ }
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

DELETE social/users/:userId/follow

Unfollow a user

Auth: required

200 OK

```
{ "success": true }
```

Possible Errors:

- **401**
- **404** - Not found
  - Outfit isn't owned or not found

GET social/users/:userId/followers

List followers

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Body:

```
{ "limit": 20, "offset": 0 }
```

201 - Created Response:

```
{
  "message": "Followers retrieved successfully",
  "followers": [
    { "id": "string-uuid", "name": "Jane Doe", "avatarUrl": "/u/jane.png" }
  ],
  "pagination": { "limit": 20, "offset": 0 }
}
```

GET social/users/:userId/following

List following

Auth: optional (per your policy)

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Body:

```
{ "limit": 20, "offset": 0 }
```

201 - Created Response:

```
{
  "message": "Following users retrieved successfully",
  "following": [
    { "id": "string-uuid", "name": "John Smith", "avatarUrl": "/u/john.png" }
  ],
  "pagination": { "limit": 20, "offset": 0 }
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

POST social/users/search

Search users

Auth: required

Request Body:

```
{
  "q": "jan",
  "limit": 20,
  "offset": 0
}
```

200 OK Response

```
{
  "q": "jan",
  "limit": 20,
  "offset{
  "message": "Users retrieved successfully",
  "results": [
    { "id": "string-uuid", "name": "Jane Doe", "avatarUrl": "/u/jane.png" }
  ],
  "pagination": { "limit": 20, "offset": 0 }
}
": 0
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

TESTABILITY

- **Unit:**
  - service methods for ownership checks, duplicate like/follow handling, comment validation.
- **Integration (Supertest):**
  - auth guard; pagination; include expansions; JSON body acceptance for list endpoints (if you adopt that tweak); multipart path for createPost.
- **Contract tests:**
  - generate OpenAPI definitions from this contract and verify in CI.

Prefrences Service

OVERVIEW

The Preferences Service lets an authenticated user retrieve and update their personalization settings used by recommendations (e.g., default style, preferred colours, learning weight). It's REST over HTTP/1.1, JSON payloads, and JWT-protected.

COMMUNICATION PROTOCOL

- Protocol: HTTP/1.1
- Architecture: REST
- Content Type: application/json; charset=utf-8
- Authentication: Bearer JWT in Authorization header
- Internal DB Access: Prisma ORM to a relational database (likely PostgreSQL)

DATA FORMAT

- Requests/Responses: JSON, camelCase properties
- Dates: ISO-8601 (UTC) strings for dateFrom, dateTo
- IDs: String UUIDs
- Weather field: persisted/returned as a JSON-stringified value in this version (see examples). Consider parsing to an object in a future version.

SECURITY

- JWT (HS256) enforced by shared auth middleware.
- Missing token → 401, invalid token → 403. (Aligned with other services.)

VALIDATION RULES

- Auth required (401 if missing).
- GET returns 404 if the user has no saved preferences.
- PUT:
- style is required and must be a valid enum value.
- preferredColours is required, must be an array, length 1..5, each item a string (hex like #rrggbb).
- learningWeight is optional; if provided, should be numeric (e.g., 0.0..1.0).
- Server responds with 400 on invalid body

ERROR HANDLING

Error Response Format

json

```
{ "message": string }
```

Status Codes Used:

- **400** - *Bad Request*
  - invalid/missing fields, invalid enums, ownership violations (e.g., adding an item not owned by user).
- **401** - *Unauthorized*
  - missing/invalid user context.
- **403** - *Forbidden*
  - invalid JWT (middleware).
- **404** - *Not Found*
  - outfit/outfit-item not found (or not owned).
- **429** - *Too Many Requests*
  - if rate limiting enabled.
- **5xx**
  - server errors.

ENDPOINTS

GET social/preferences

get my preferences

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

200 OK Response

```
{
  "style": "Casual",
  "preferredColours": ["#112233",
"#aabbcc"],
  "learningWeight": 0.3
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

PUT social/preferences
Create or update my preferences

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Body:

```
{
  "style": "Casual",
  "preferredColours": ["#112233",
"#aabbcc"],
  "learningWeight": 0.3
}
```

201 - Created Response:

```
{
  "style": "Casual",
  "preferredColours": ["#112233",
"#aabbcc"],
  "learningWeight": 0.3
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

TIMEOUTS AND LIMITS

- Client request timeout: 10–15s
- Rate limits (example): 60 writes / 15 min / user; 300 reads / 15 min / user
- On breach → 429 Too Many Requests with Retry-After

TESTABILITY

- **Unit:**
  - GET: returns 404 when absent, 200 when present.
  - PUT: body validation (style required, colours array length, type checking), upsert behaviour.
- **Integration (Supertest):**
  - Auth guard (401), invalid bodies (400), happy paths (200).
- **Contract tests:**
  - generate OpenAPI and run in CI.

Users Service

OVERVIEW

The Users Service exposes endpoints for an authenticated user to:

- Fetch their profile (GET /me)
- Update their profile photo (S3-backed upload, POST /me/photo)
- Base URL (recommended): /api/v1/users
- Protocol: REST over HTTP/1.1
- Content types:
  - Requests: application/json (except photo upload uses multipart/form-data)
  - Responses: application/json; charset=utf-8
- Auth: Authorization: Bearer <JWT> required

COMMUNICATION PROTOCOL

- Protocol: HTTP/1.1
- Architecture: REST
- Content Type: application/json; charset=utf-8
- Authentication: Bearer JWT in Authorization header
- Internal DB Access: Prisma ORM to a relational database (likely PostgreSQL)

DATA FORMAT

- Requests/Responses: JSON, camelCase properties
- Dates: ISO-8601 (UTC) strings for dateFrom, dateTo
- IDs: String UUIDs
- Weather field: persisted/returned as a JSON-stringified value in this version (see examples). Consider parsing to an object in a future version.

VALIDATION RULES

- All endpoints require a valid authenticated user (req.user.id present) → otherwise 401 Unauthorized.
- Photo upload:
- Accepts a single file field named (e.g.) "file" as configured in your Multer middleware.
- Allowed MIME types map to extensions: image/png → .png, image/jpeg|image/jpg → .jpg, image/webp → .webp; otherwise .bin.
- File is uploaded to S3 via uploadBufferToS3 with key pattern: users/{userId}/profile/{timestamp}-{uuid}{ext}; the public URL is returned via cdnUrlFor(key).
- Missing file → 400.

ENDPOINTS

GET users/me
Get my profile

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

200 OK Response

```
{
  "user": {
    "id": "string-uuid",
    "name": "Jane Doe",
    "email": "jane@example.com",
    "profilePhotoUrl": "https://cdn.example.com/users/123/profile/1692301122333-uuid.jpg",
    "createdAt": "2025-08-18T09:00:00.000Z",
    "updatedAt": "2025-08-18T09:05:00.000Z"
  }
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

POST users/me/photo
Update my profile photo

Auth: required

Request Body:

Fields

- file: binary image file (required)

200 OK Response

```
{
  "message": "Updated",
  "user": {
    "id": "string-uuid",
    "name": "Jane Doe",
    "email": "jane@example.com",
    "profilePhotoUrl": "https://cdn.example.com/users/123/profile/1723978294000-5f1c6c8f-
9d4b-4d14-96b2-0e7c49b1a4a1.jpg",
    "createdAt": "2025-08-18T09:00:00.000Z",
    "updatedAt": "2025-08-18T09:10:12.000Z"
  }
}
```

STORAGE & CDN CONTRACT

- Key format: users/{userId}/profile/{timestamp}-{uuid}{ext}
- S3 Put: bucket from S3_BUCKET_NAME, content type from uploaded file MIME (application/octet-stream fallback).
- Public URL: derived via cdnUrlFor(key) and persisted to the user via usersService.setProfilePhoto(userId, publicUrl).

TESTABILITY

- **Unit (service & helpers):**
  - extFromMime mapping
  - fileBuffer behavior (memory vs disk storage)
  - usersService.setProfilePhoto side-effects
- **Integration (Supertest):**
  - GET /me happy path, 401, 404
  - POST /me/photo with multipart upload: 200/201, 400 (no file), 401 (no auth)
  - Verify response contains updated profilePhotoUrl

Weather Service

OVERVIEW

Provides current/near-term weather by location and a single-day forecast for a specific date. Internally calls getWeatherByLocation(location) and getWeatherByDay(location, yyyy-mm-dd).

COMMUNICATION PROTOCOL

- Base URL (recommended): /api/v1/weather
- Protocol: REST over HTTP/1.1
- Auth: none (unless you want to protect it)
- Content types:
- Requests: application/json
- Responses: application/json; charset=utf-8

DATA FORMAT

- Requests/Responses: JSON, camelCase properties
- Dates: ISO-8601 (UTC) strings for dateFrom, dateTo
- IDs: String UUIDs
- Weather field: persisted/returned as a JSON-stringified value in this version (see examples). Consider parsing to an object in a future version.

VALIDATION RULES

- All endpoints require a valid authenticated user (req.user.id present) → otherwise 401 Unauthorized.
- Photo upload:
- Accepts a single file field named (e.g.) "file" as configured in your Multer middleware.
- Allowed MIME types map to extensions: image/png → .png, image/jpeg|image/jpg → .jpg, image/webp → .webp; otherwise .bin.
- File is uploaded to S3 via uploadBufferToS3 with key pattern: users/{userId}/profile/{timestamp}-{uuid}{ext}; the public URL is returned via cdnUrlFor(key).
- Missing file → 400.

DATA MODELS

Weather (location Summary)

```
{
  "location": "Pretoria",
  "date": "2025-08-19",
  "summary": {
    "avgTemp": 19,
    "minTemp": 12,
    "maxTemp": 22,
    "willRain": false,
    "mainCondition": "Clouds"
  },
  "forecast": [
    { "hour": "00:00", "temp": 14, "willRain": false },
    { "hour": "03:00", "temp": 13, "willRain": false }
  ],
  "provider": "internal-or-3rd-party-id",
  "updatedAt": "2025-08-18T09:30:00Z"
}
```

Weather (Single Day)

```
{
  "location": "Pretoria",
  "current": {
    "temp": 21.3,
    "feelsLike": 22.0,
    "humidity": 0.55,
    "windKph": 12.0,
    "condition": "Clouds"
  },
  "forecast": [
    { "date": "2025-08-18", "min": 12, "max": 23, "willRain": false, "summary": "Partly
cloudy" },
    { "date": "2025-08-19", "min": 11, "max": 22, "willRain": false, "summary": "Clouds" }
  ],
  "provider": "internal-or-3rd-party-id",
  "updatedAt": "2025-08-18T09:30:00Z"
}
```

ERROR HANDLING

Error Response Format

json

```
{ "message": string }
```

Status Codes Used:

- **400** - *Bad Request*
  - invalid/missing fields, invalid enums, ownership violations (e.g., adding an item not owned by user).
- **401** - *Unauthorized*
  - missing/invalid user context.
- **403** - *Forbidden*
  - invalid JWT (middleware).
- **404** - *Not Found*
  - outfit/outfit-item not found (or not owned).
- **429** - *Too Many Requests*
  - if rate limiting enabled.
- **5xx**
  - server errors.

ENDPOINTS

POST /weather

Weather by location

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Request Body:

```
{ "location": "Pretoria" }
```

200 OK Response

```
{
  "location": "Pretoria",
  "current": { "temp": 21.3, "feelsLike": 22.0, "humidity": 0.55, "windKph": 12.0,
"condition": "Clouds" },
  "forecast": [
    { "date": "2025-08-18", "min": 12, "max": 23, "willRain": false, "summary": "Partly
cloudy" }
  ],
  "provider": "internal-or-3rd-party-id",
  "updatedAt": "2025-08-18T09:30:00Z"
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

POST /weather/day

Weather for specific day

Auth: required

Headers:

```
Authorization: Bearer <JWT>
Content-Type: application/json
```

Request Body:

```
{
  "location": "Pretoria",
  "date": "2025-08-19"    // ISO date, YYYY-MM-DD
}
```

200 OK Response

```
{
  "location": "Pretoria",
  "date": "2025-08-19",
  "summary": { "avgTemp": 19, "minTemp": 12, "maxTemp": 22, "willRain": false,
"mainCondition": "Clouds" },
  "forecast": [
    { "hour": "00:00", "temp": 14, "willRain": false },
    { "hour": "03:00", "temp": 13, "willRain": false }
  ],
  "provider": "internal-or-3rd-party-id",
  "updatedAt": "2025-08-18T09:30:00Z"
}
```

Possible Errors:

- **400** - No file
  - { "message": "Unauthorized" }
- **401** - Unauthorized
  - { "message": "Unauthorized" }
- **500** - Internal Server Error

TESTABILITY

- **Unit:**
  - validate body parsing, required fields, and propagation of service results; mapping of 404 when no day forecast; 500 on thrown errors.
- **Integration (Supertest):**
  - happy paths for both endpoints; 400/404/500 branches.
- **Contract tests:**
  - export OpenAPI (below) and verify with a mock server.