Data-exchange technologies

One exchange uses several layers

A mobile app does not reach into another system's database. It communicates through an agreed interface.

- **API:** a defined interface through which software requests data or actions.
- **REST:** an architectural approach that treats data and capabilities as resources addressed by identifiers such as URLs.
- **HTTP:** an application-layer protocol that carries web requests and responses.
- **JSON or XML:** formats that structure data inside a message.

> **Example:** An app uses an API designed with REST conventions, sends an HTTPS request and receives JSON. These terms describe different parts of one exchange.

Request and response

```
GET /api/events/42 HTTP/1.1
Host: example.org
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json
```

{"event_id": 42, "title": "Repair café", "places": 6}

The client identifies a resource and acceptable representation. The server processes the request and returns a status, metadata headers and optional body.

Trace the exchange

Recognise the method, resource identifier, requested format, response status and body type.

Check your answer

Method: GET; resource: `/api/events/42`; requested format: JSON; status: `200 OK`; body: one JSON event object.

HTTP methods communicate intent

Method	Typical purpose	Example
GET	Retrieve a representation without changing the resource	`GET /events/42`
POST	Submit data, often to create a resource	`POST /bookings`
PUT	Replace a resource representation at a known identifier	`PUT /bookings/81`
PATCH	Apply a partial update	`PATCH /bookings/81`
DELETE	Request removal of a resource	`DELETE /bookings/81`

The server defines the actual API contract. A method name alone does not grant permission or guarantee a particular implementation.

Choose a method

Determine a suitable method for retrieving all venues, creating a booking and changing only its status.

Check your answer

`GET /venues`, `POST /bookings` and `PATCH /bookings/{id}` are suitable, assuming the API contract uses these conventions.

Status codes describe outcomes

- **2xx:** request succeeded, e.g. 200 OK or 201 Created.
- **4xx:** the request cannot be completed as sent, e.g. 400 Bad Request, 401 Unauthorised, 403 Forbidden or 404 Not Found.
- **5xx:** the server failed while handling an apparently valid request, e.g. 500 Internal Server Error or 503 Service Unavailable.

> **Authentication and authorisation differ:** 401 commonly indicates missing or invalid authentication; 403 indicates the server understood the identity but refuses the action.

Handle more than success

A robust client checks the response before using the body.

```
SET response = SendRequest(request)
```

IF response.status = 200 THEN SET data = ParseJSON(response.body) DisplayEvents(data) ELSE IF response.status = 401 THEN OUTPUT "Sign in again" ELSE IF response.status = 503 THEN OUTPUT "Service unavailable. Try again later." ELSE OUTPUT "Request could not be completed" ENDIF

Parsing a missing or error body as if it contains valid event data can create a second error.

JSON and XML represent structured data

JSON

```
{
  "sensor_id": 7,
  "reading": 18.4
```

```
}
```

XML

```
<sensor>
  <sensor_id>7</sensor_id>
  <reading>18.4</reading>
</sensor>
```

- JSON uses objects, arrays, keys and values and maps naturally to common application data structures.
- XML uses nested elements and attributes, supports namespaces and can be validated against a schema.
- Either can represent the example. Suitability depends on the systems, contract, tooling, validation and interoperability requirements.

Neither format encrypts data. Secure transport and access controls are separate concerns.

Compare formats

A partner's established system accepts only XML validated against its published schema. **Recommend** JSON or XML and **Explain** the decisive evidence.

Check your answer

Use XML because compatibility with the receiving system and its published validation schema is required. Choosing JSON for shorter syntax would not satisfy the exchange contract.

REST resources and representations

A resource is a conceptual item such as an event or booking. A URL identifies it; a representation such as JSON describes its current data.

```
GET /events                 # collection
GET /events/42              # one event
GET /events?suburb=Milton   # filtered collection
POST /events/42/bookings    # create a related booking
```

Good resource paths normally use nouns. The HTTP method communicates the action.

REST exchanges are usually stateless: each request includes the information needed to process it. The server may store resources, but it should not rely on hidden conversational state from an earlier request.

APIs need a contract

Documentation should specify:

- endpoints and supported methods
- authentication and permissions
- required headers, parameters and body fields
- data types, formats and allowed values
- response bodies and status codes
- rate limits, pagination and versioning where relevant.

A client and server can both produce valid JSON yet fail to interoperate if one sends `eventDate` and the other requires `event_date`.

Analyse a failed exchange

The contract requires `POST /reports` with `{"location_id": INTEGER, "description": STRING}`. A client sends:

```
POST /report
{"location": "Oval", "description": 42}
```

Analyse three incompatibilities.

Check your answer

- The endpoint is singular rather than `/reports`.
- The required `location_id` key is replaced by `location`.

- `description` contains a number rather than a string.

Secure and reliable exchange

- Use encrypted, authenticated transport such as HTTPS for sensitive exchanges.
- Authenticate the caller and check authorisation for every protected operation.
- Validate all external data before processing.
- Return useful but non-sensitive error information.
- Use timeouts, controlled retries and duplicate protection.
- Log relevant outcomes without unnecessarily recording secrets or personal data.

Retrying a create request blindly can produce duplicates. An idempotency key or server-side duplicate check can let a repeated request be recognised safely.

Independent practice: repair reports API

A mobile app sends repair reports to a council API. The contract uses JSON and includes:

```
POST /reports
Authorization: Bearer [token]
Content-Type: application/json
```

{ "location_id": 18, "category": "water_leak", "description": "Tap will not turn off" }

Possible responses are 201 with the created report, 400 with field errors, 401 when the token is invalid and 503 during an outage.

1. **Explain** the role of the API, HTTP, JSON and authentication token.
2. **Determine** how the app should respond to each status.
3. **Evaluate** the exchange if it uses HTTP rather than HTTPS on public Wi-Fi.
4. **Recommend** one strategy that reduces duplicate reports after a timeout.

View a sample solution

1. The API defines the permitted exchange, HTTP carries the request and response, JSON structures the report, and the token presents authentication evidence used when checking access.
2. 201: show confirmation and returned ID; 400: show specific field errors without clearing valid input; 401: require authentication again; 503: preserve the report and offer a controlled retry.
3. HTTP provides weak confidentiality on the untrusted network because the token and report may be intercepted. HTTPS should authenticate the server and encrypt data in transit.
4. Send a unique idempotency key with retries so the server can return the original result rather than create another record.

Revision #5
Created 2026-10-06 00:40:30 UTC by Mr Napper
Updated 2026-10-06 01:32:23 UTC by Mr Napper