

A HANDS-ON TUTORIAL

Exporting FHIR Data

Rebuild a patient's records into one downloadable FHIR Bundle

This tutorial focuses on the Complete FHIR Bundle export; IPS and Lab Report exports use separate builders.

Panagiotis Liakakos

June 2026

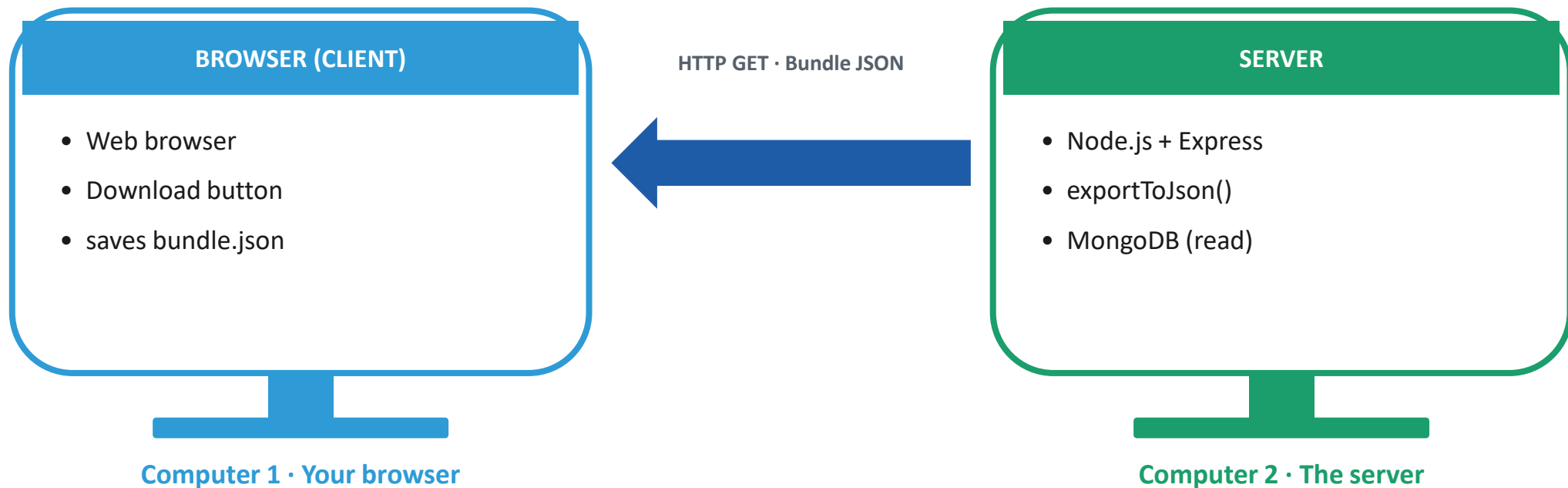
What you'll learn

By the end, you will be able to:

- Gather the patient's raw-FHIR-backed resources from MongoDB.
- Pull in the shared resources they reference.
- Rewrite the references so the result is fully self-contained.
- Assemble a FHIR collection Bundle and download it as a file.

Goal: turn a patient's stored data into one downloadable FHIR Bundle.

How the data travels: server → browser



A patient's data is spread across many MongoDB collections. The server gathers it, restores the links between resources, and returns one FHIR Bundle that the browser saves as a file.

What you need to export FHIR data



You only need three tools

- Mongoose `.find()` / `.lean()` — read each collection's stored resource.
- Plain JavaScript — walk references and build the Bundle. No FHIR library needed.
- Browser Blob + a download link — save the JSON as a file.

Prerequisites: Node 18+ and a running MongoDB (mongod) holding ingested data — then the three tools above work.
The core export is ~70 lines in `fhirService.js` (`exportToJson`), plus a one-line route and a small frontend download handler.

What you need to export FHIR data

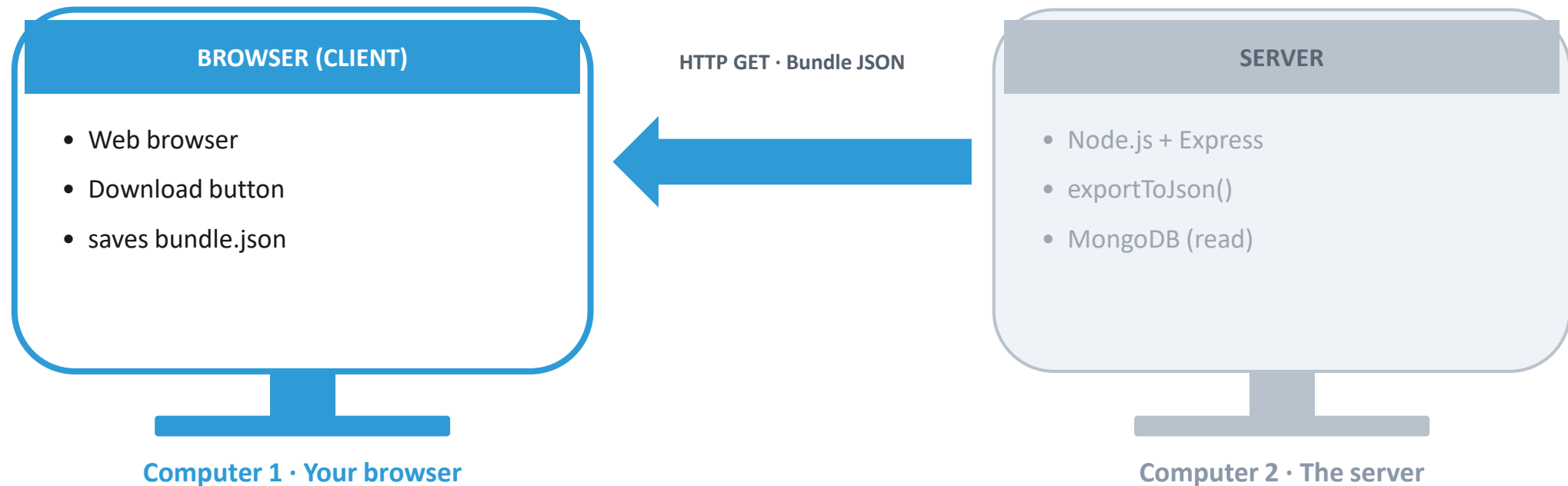


Five things not to forget

- Build from the stored rawFhir — it is the untouched original resource.
- Group everything in a Bundle of type "collection".
- Give every entry a fullUrl — use urn:uuid:<id>.
- Rewrite references to those fullUrls so the bundle resolves on its own.
- Pull in referenced resources (Organizations, Practitioners) too.

Prerequisites: Node 18+ and a running MongoDB (mongod) holding ingested data — then the three tools above work.
The core export is ~70 lines in fhirService.js (exportToJson), plus a one-line route and a small frontend download handler.

First: the browser side (Computer 1)



The first step runs on Computer 1 — the user's web browser. It requests the export and saves the returned Bundle as a JSON file.

Step 1 — From a click to a saved file (browser)

IN PLAIN ENGLISH

- A Download button calls one handler with the chosen patient.

frontend · DownloadDialog.jsx

```
// frontend — request the bundle, save it as a file
async function handleDownload(patientId) {
  const endpoint = `${API}/fhir/export/${patientId}`;
  const filename = `patient-${patientId}-fhir-bundle.json`;

  // ask the server to build the Bundle
  const res = await fetch(endpoint, {
    credentials: "include", // send session cookie
  });
  if (!res.ok) throw new Error("export failed");

  // turn the JSON response into a saved file
  const blob = await res.blob();
  const url = URL.createObjectURL(blob);
  const a = document.createElement("a");
  a.href = url;
  a.download = filename;
  a.click();
  URL.revokeObjectURL(url);
}
```

Step 1 — From a click to a saved file (browser)

IN PLAIN ENGLISH

- A Download button calls one handler with the chosen patient.
- **It requests the export endpoint for that patient, sending the session cookie — which the export route does not verify (no auth middleware).**

frontend · DownloadDialog.jsx

```
// frontend — request the bundle, save it as a file
async function handleDownload(patientId) {
  const endpoint = `${API}/fhir/export/${patientId}`;
  const filename = `patient-${patientId}-fhir-bundle.json`;

  // ask the server to build the Bundle
  const res = await fetch(endpoint, {
    credentials: "include", // send session cookie
  });
  if (!res.ok) throw new Error("export failed");

  // turn the JSON response into a saved file
  const blob = await res.blob();
  const url = URL.createObjectURL(blob);
  const a = document.createElement("a");
  a.href = url;
  a.download = filename;
  a.click();
  URL.revokeObjectURL(url);
}
```

Step 1 — From a click to a saved file (browser)

IN PLAIN ENGLISH

- A Download button calls one handler with the chosen patient.
- It requests the export endpoint for that patient, sending the session cookie — which the export route does not verify (no auth middleware).
- **The JSON response is read as a Blob — an in-memory file.**

frontend · DownloadDialog.jsx

```
// frontend — request the bundle, save it as a file
async function handleDownload(patientId) {
  const endpoint = `${API}/fhir/export/${patientId}`;
  const filename = `patient-${patientId}-fhir-bundle.json`;

  // ask the server to build the Bundle
  const res = await fetch(endpoint, {
    credentials: "include", // send session cookie
  });
  if (!res.ok) throw new Error("export failed");

  // turn the JSON response into a saved file
  const blob = await res.blob();
  const url = URL.createObjectURL(blob);
  const a = document.createElement("a");
  a.href = url;
  a.download = filename;
  a.click();
  URL.revokeObjectURL(url);
}
```

Step 1 — From a click to a saved file (browser)

IN PLAIN ENGLISH

- A Download button calls one handler with the chosen patient.
- It requests the export endpoint for that patient, sending the session cookie — which the export route does not verify (no auth middleware).
- The JSON response is read as a Blob — an in-memory file.
- **A temporary link triggers the browser's Save dialog, then is cleaned up.**

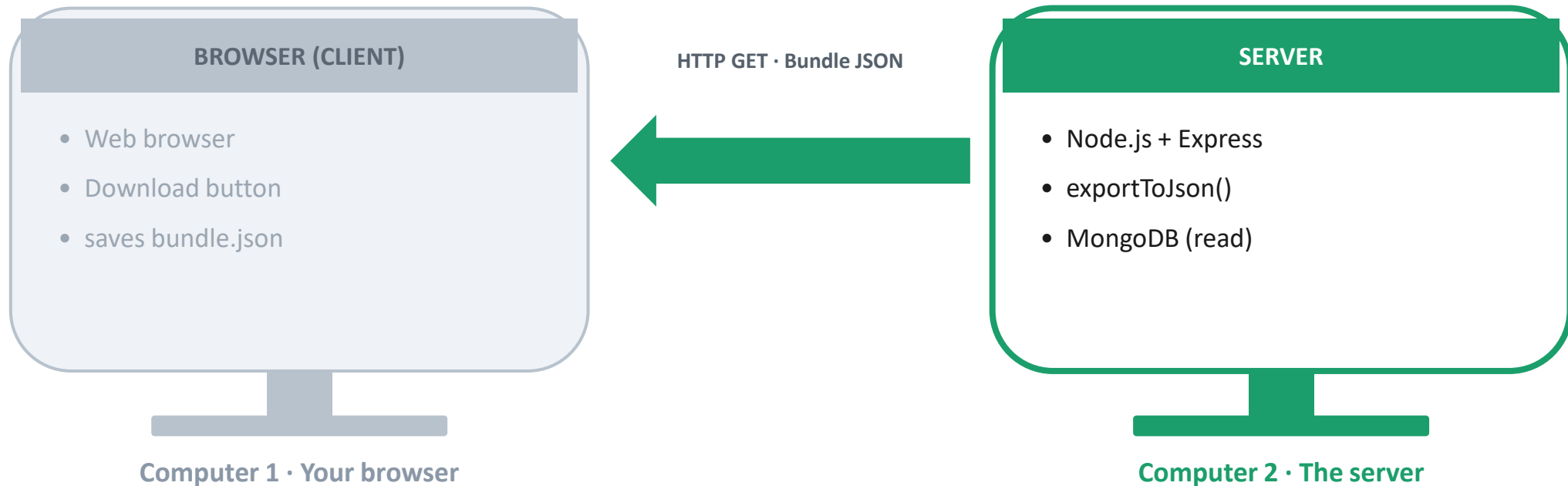
frontend · DownloadDialog.jsx

```
// frontend — request the bundle, save it as a file
async function handleDownload(patientId) {
  const endpoint = `${API}/fhir/export/${patientId}`;
  const filename = `patient-${patientId}-fhir-bundle.json`;

  // ask the server to build the Bundle
  const res = await fetch(endpoint, {
    credentials: "include", // send session cookie
  });
  if (!res.ok) throw new Error("export failed");

  // turn the JSON response into a saved file
  const blob = await res.blob();
  const url = URL.createObjectURL(blob);
  const a = document.createElement("a");
  a.href = url;
  a.download = filename;
  a.click();
  URL.revokeObjectURL(url);
}
```

Now: the server side (Computer 2)



The next steps run on Computer 2 — the Node / Express server, which reads the patient's resources from MongoDB and assembles them into one Bundle.

Step 2 — Gather the patient's resources (server)

IN PLAIN ENGLISH

- Look up the patient; if there is none, stop with an error.

backend/services/fhirService.js

```
// backend — gather everything we hold for this patient
async function exportToJson(patientId) {
  const patientDoc = await Patient.findOne({ patientId }).lean();
  if (!patientDoc) throw new Error("Patient not found");

  // start the list with the patient, then every collection
  const resources = [];
  if (patientDoc.rawFhir) resources.push(patientDoc.rawFhir);

  for (const Model of PATIENT_SCOPED_MODELS) {
    const docs = await Model.find({ patientId }).lean();
    for (const d of docs)
      if (d.rawFhir) resources.push(d.rawFhir);
  }
  // ...resources[] now holds every raw-FHIR-backed resource...
}
```

Step 2 — Gather the patient's resources (server)

IN PLAIN ENGLISH

- Look up the patient; if there is none, stop with an error.
- **Start the list with the patient's own resource.**

backend/services/fhirService.js

```
// backend — gather everything we hold for this patient
async function exportToJson(patientId) {
  const patientDoc = await Patient.findOne({ patientId }).lean();
  if (!patientDoc) throw new Error("Patient not found");

  // start the list with the patient, then every collection
  const resources = [];
  if (patientDoc.rawFhir) resources.push(patientDoc.rawFhir);

  for (const Model of PATIENT_SCOPED_MODELS) {
    const docs = await Model.find({ patientId }).lean();
    for (const d of docs)
      if (d.rawFhir) resources.push(d.rawFhir);
  }
  // ...resources[] now holds every raw-FHIR-backed resource...
}
```

Step 2 — Gather the patient's resources (server)

IN PLAIN ENGLISH

- Look up the patient; if there is none, stop with an error.
- Start the list with the patient's own resource.
- **Walk every patient-scoped collection that stores the original resource.**

backend/services/fhirService.js

```
// backend — gather everything we hold for this patient
async function exportToJson(patientId) {
  const patientDoc = await Patient.findOne({ patientId }).lean();
  if (!patientDoc) throw new Error("Patient not found");

  // start the list with the patient, then every collection
  const resources = [];
  if (patientDoc.rawFhir) resources.push(patientDoc.rawFhir);

  for (const Model of PATIENT_SCOPED_MODELS) {
    const docs = await Model.find({ patientId }).lean();
    for (const d of docs)
      if (d.rawFhir) resources.push(d.rawFhir);
  }
  // ...resources[] now holds every raw-FHIR-backed resource...
}
```

Step 2 — Gather the patient's resources (server)

IN PLAIN ENGLISH

- Look up the patient; if there is none, stop with an error.
- Start the list with the patient's own resource.
- Walk every patient-scoped collection that stores the original resource.
- **Collect each stored resource into one working list.**

backend/services/fhirService.js

```
// backend — gather everything we hold for this patient
async function exportToJson(patientId) {
  const patientDoc = await Patient.findOne({ patientId }).lean();
  if (!patientDoc) throw new Error("Patient not found");

  // start the list with the patient, then every collection
  const resources = [];
  if (patientDoc.rawFhir) resources.push(patientDoc.rawFhir);

  for (const Model of PATIENT_SCOPED_MODELS) {
    const docs = await Model.find({ patientId }).lean();
    for (const d of docs)
      if (d.rawFhir) resources.push(d.rawFhir);
  }
  // ...resources[] now holds every raw-FHIR-backed resource...
}
```

Step 3 — Pull in what they reference (server)

IN PLAIN ENGLISH

- Resources point at shared Organizations and Practitioners.

backend/services/fhirService.js

```
// find the shared resources our resources point to
const referencedIds = new Set();
resources.forEach((r) => {
  walkReferences(r, (ref) => {
    const id = refToId(ref); // urn:uuid:x | Type/x -> x
    if (id) referencedIds.add(id);
    return ref;
  });
});

const ids = [...referencedIds];
const [orgs, pracs] = await Promise.all([
  Organization.find({ "rawFhir.id": { $in: ids } }).lean(),
  Practitioner.find({ "rawFhir.id": { $in: ids } }).lean(),
]);
for (const d of [...orgs, ...pracs])
  if (d.rawFhir) resources.push(d.rawFhir);
```

Step 3 — Pull in what they reference (server)

IN PLAIN ENGLISH

- Resources point at shared Organizations and Practitioners.
- **Walk every reference and collect the ids they mention.**

backend/services/fhirService.js

```
// find the shared resources our resources point to
const referencedIds = new Set();
resources.forEach((r) => {
  walkReferences(r, (ref) => {
    const id = refToId(ref); // urn:uuid:x | Type/x -> x
    if (id) referencedIds.add(id);
    return ref;
  });
});

const ids = [...referencedIds];
const [orgs, pracs] = await Promise.all([
  Organization.find({ "rawFhir.id": { $in: ids } }).lean(),
  Practitioner.find({ "rawFhir.id": { $in: ids } }).lean(),
]);
for (const d of [...orgs, ...pracs])
  if (d.rawFhir) resources.push(d.rawFhir);
```

Step 3 — Pull in what they reference (server)

IN PLAIN ENGLISH

- Resources point at shared Organizations and Practitioners.
- Walk every reference and collect the ids they mention.
- **Look those ids up in the shared collections.**

backend/services/fhirService.js

```
// find the shared resources our resources point to
const referencedIds = new Set();
resources.forEach((r) => {
  walkReferences(r, (ref) => {
    const id = refToId(ref); // urn:uuid:x | Type/x -> x
    if (id) referencedIds.add(id);
    return ref;
  });
});

const ids = [...referencedIds];
const [orgs, pracs] = await Promise.all([
  Organization.find({ "rawFhir.id": { $in: ids } }).lean(),
  Practitioner.find({ "rawFhir.id": { $in: ids } }).lean(),
]);
for (const d of [...orgs, ...pracs])
  if (d.rawFhir) resources.push(d.rawFhir);
```

Step 3 — Pull in what they reference (server)

IN PLAIN ENGLISH

- Resources point at shared Organizations and Practitioners.
- Walk every reference and collect the ids they mention.
- Look those ids up in the shared collections.
- **Add the resources found to the working list.**

backend/services/fhirService.js

```
// find the shared resources our resources point to
const referencedIds = new Set();
resources.forEach((r) => {
  walkReferences(r, (ref) => {
    const id = refToId(ref); // urn:uuid:x | Type/x -> x
    if (id) referencedIds.add(id);
    return ref;
  }));

const ids = [...referencedIds];
const [orgs, pracs] = await Promise.all([
  Organization.find({ "rawFhir.id": { $in: ids } }).lean(),
  Practitioner.find({ "rawFhir.id": { $in: ids } }).lean(),
]);
for (const d of [...orgs, ...pracs])
  if (d.rawFhir) resources.push(d.rawFhir);
```

Step 4 — Re-link into a self-contained set (server)

IN PLAIN ENGLISH

- Copy each resource so the originals are never modified.

backend/services/fhirService.js

```
// clone, de-duplicate, and rewrite references
const byId = new Map();
for (const r of resources) {
  const clone = structuredClone(r);
  if (!clone.id) clone.id = uuidv4(); // ensure an id
  if (!byId.has(clone.id)) byId.set(clone.id, clone);
}

const presentIds = new Set(byId.keys());
for (const r of byId.values()) {
  walkReferences(r, (ref) => {
    const id = refToId(ref);
    // point at the entry inside this bundle
    return id && presentIds.has(id)
      ? `urn:uuid:${id}`
      : ref; // external -> leave as is
  });
}
```

Step 4 — Re-link into a self-contained set (server)

IN PLAIN ENGLISH

- Copy each resource so the originals are never modified.
- **Give any resource without an id a generated id for this export, and keep one copy per id.**

backend/services/fhirService.js

```
// clone, de-duplicate, and rewrite references
const byId = new Map();
for (const r of resources) {
  const clone = structuredClone(r);
  if (!clone.id) clone.id = uuidv4(); // ensure an id
  if (!byId.has(clone.id)) byId.set(clone.id, clone);
}

const presentIds = new Set(byId.keys());
for (const r of byId.values()) {
  walkReferences(r, (ref) => {
    const id = refToId(ref);
    // point at the entry inside this bundle
    return id && presentIds.has(id)
      ? `urn:uuid:${id}`
      : ref; // external -> leave as is
  });
}
```

Step 4 — Re-link into a self-contained set (server)

IN PLAIN ENGLISH

- Copy each resource so the originals are never modified.
- Give any resource without an id a generated id for this export, and keep one copy per id.
- **Record which ids are present in the export.**

backend/services/fhirService.js

```
// clone, de-duplicate, and rewrite references
const byId = new Map();
for (const r of resources) {
  const clone = structuredClone(r);
  if (!clone.id) clone.id = uuidv4(); // ensure an id
  if (!byId.has(clone.id)) byId.set(clone.id, clone);
}

const presentIds = new Set(byId.keys());
for (const r of byId.values()) {
  walkReferences(r, (ref) => {
    const id = refToId(ref);
    // point at the entry inside this bundle
    return id && presentIds.has(id)
      ? `urn:uuid:${id}`
      : ref; // external -> leave as is
  });
}
```

Step 4 — Re-link into a self-contained set (server)

IN PLAIN ENGLISH

- Copy each resource so the originals are never modified.
- Give any resource without an id a generated id for this export, and keep one copy per id.
- Record which ids are present in the export.
- **Rewrite each reference to a present resource as urn:uuid.**

backend/services/fhirService.js

```
// clone, de-duplicate, and rewrite references
const byId = new Map();
for (const r of resources) {
  const clone = structuredClone(r);
  if (!clone.id) clone.id = uuidv4(); // ensure an id
  if (!byId.has(clone.id)) byId.set(clone.id, clone);
}

const presentIds = new Set(byId.keys());
for (const r of byId.values()) {
  walkReferences(r, (ref) => {
    const id = refToId(ref);
    // point at the entry inside this bundle
    return id && presentIds.has(id)
      ? `urn:uuid:${id}`
      : ref; // external -> leave as is
  });
}
```

Step 5 — Wrap it all in a FHIR Bundle (server)

IN PLAIN ENGLISH

- Return a Bundle object of type "collection".

backend/services/fhirService.js

```
// assemble the self-contained FHIR Bundle
const bundleId = uuidv4();
return {
  resourceType: "Bundle",
  id: bundleId,
  type: "collection",
  timestamp: new Date().toISOString(),
  identifier: {
    system: "urn:iETF:rFC:3986",
    value: `urn:uuid:${bundleId}`,
  },
  entry: [...byId.values()].map((resource) => ({
    fullUrl: `urn:uuid:${resource.id}`,
    resource,
  })),
};
```

Step 5 — Wrap it all in a FHIR Bundle (server)

IN PLAIN ENGLISH

- Return a Bundle object of type "collection".
- **Stamp it with a timestamp and a unique identifier.**

backend/services/fhirService.js

```
// assemble the self-contained FHIR Bundle
const bundleId = uuidv4();
return {
  resourceType: "Bundle",
  id: bundleId,
  type: "collection",
  timestamp: new Date().toISOString(),
  identifier: {
    system: "urn:iETF:rFC:3986",
    value: `urn:uuid:${bundleId}`,
  },
  entry: [...byId.values()].map((resource) => ({
    fullUrl: `urn:uuid:${resource.id}`,
    resource,
  })),
};
```

Step 5 — Wrap it all in a FHIR Bundle (server)

IN PLAIN ENGLISH

- Return a Bundle object of type "collection".
- Stamp it with a timestamp and a unique identifier.
- **List every resource as an entry.**

backend/services/fhirService.js

```
// assemble the self-contained FHIR Bundle
const bundleId = uuidv4();
return {
  resourceType: "Bundle",
  id: bundleId,
  type: "collection",
  timestamp: new Date().toISOString(),
  identifier: {
    system: "urn:iETF:rFC:3986",
    value: `urn:uuid:${bundleId}`,
  },
  entry: [...byId.values()].map((resource) => ({
    fullUrl: `urn:uuid:${resource.id}`,
    resource,
  })),
};
```

Step 5 — Wrap it all in a FHIR Bundle (server)

IN PLAIN ENGLISH

- Return a Bundle object of type "collection".
- Stamp it with a timestamp and a unique identifier.
- List every resource as an entry.
- **Each entry pairs a urn:uuid fullUrl with its resource.**

backend/services/fhirService.js

```
// assemble the self-contained FHIR Bundle
const bundleId = uuidv4();
return {
  resourceType: "Bundle",
  id: bundleId,
  type: "collection",
  timestamp: new Date().toISOString(),
  identifier: {
    system: "urn:iETF:rFC:3986",
    value: `urn:uuid:${bundleId}`,
  },
  entry: [...byId.values()].map((resource) => ({
    fullUrl: `urn:uuid:${resource.id}`,
    resource,
  })),
};
```

The missing link — expose the endpoint

IN PLAIN ENGLISH

- One route maps GET `/api/fhir/export/:patientId` to the controller.

```
backend/routes/fhirRoutes.js + fhirController.js

// ---- expose the export: route + controller ----
//      backend/routes/fhirRoutes.js
router.get("/export/:patientId", exportToJson);

//      backend/controllers/fhirController.js
async function exportToJson(req, res) {
  try {
    const { patientId } = req.params;
    const bundle = await fhirService.exportToJson(patientId);
    res.json(bundle);           // browser downloads this
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
}
// GET /api/fhir/export/<id> -> the Bundle
```

The missing link — expose the endpoint

IN PLAIN ENGLISH

- One route maps GET /api/fhir/export/:patientId to the controller.
- **The controller reads the patientId from the URL.**

```
backend/routes/fhirRoutes.js + fhirController.js

// ---- expose the export: route + controller ----
//      backend/routes/fhirRoutes.js
router.get("/export/:patientId", exportToJson);

//      backend/controllers/fhirController.js
async function exportToJson(req, res) {
  try {
    const { patientId } = req.params;
    const bundle = await fhirService.exportToJson(patientId);
    res.json(bundle);           // browser downloads this
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
}
// GET /api/fhir/export/<id> -> the Bundle
```

The missing link — expose the endpoint

IN PLAIN ENGLISH

- One route maps GET /api/fhir/export/:patientId to the controller.
- The controller reads the patientId from the URL.
- **It calls the service to build the Bundle.**

```
backend/routes/fhirRoutes.js + fhirController.js

// ---- expose the export: route + controller ----
//      backend/routes/fhirRoutes.js
router.get("/export/:patientId", exportToJson);

//      backend/controllers/fhirController.js
async function exportToJson(req, res) {
  try {
    const { patientId } = req.params;
    const bundle = await fhirService.exportToJson(patientId);
    res.json(bundle);           // browser downloads this
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
}
// GET /api/fhir/export/<id> -> the Bundle
```

The missing link — expose the endpoint

IN PLAIN ENGLISH

- One route maps GET /api/fhir/export/:patientId to the controller.
- The controller reads the patientId from the URL.
- It calls the service to build the Bundle.
- **It sends the Bundle back as JSON for the browser to download.**

backend/routes/fhirRoutes.js + fhirController.js

```
// ---- expose the export: route + controller ----  
//      backend/routes/fhirRoutes.js  
router.get("/export/:patientId", exportToJson);  
  
//      backend/controllers/fhirController.js  
async function exportToJson(req, res) {  
  try {  
    const { patientId } = req.params;  
    const bundle = await fhirService.exportToJson(patientId);  
    res.json(bundle); // browser downloads this  
  } catch (err) {  
    res.status(500).json({ error: err.message });  
  }  
}  
// GET /api/fhir/export/<id> -> the Bundle
```

What you learned

The whole pipeline, in six moves

- A Download button requests the export endpoint and saves the response as a file.
- The server gathers the patient's resources from every collection.
- It pulls in the Organizations and Practitioners they reference.
- It de-duplicates and rewrites references into urn:uuid form.
- It wraps everything in a FHIR Bundle of type "collection".
- A small route and controller expose it at GET /api/fhir/export/:patientId.

You can now turn a patient's stored data into a downloadable FHIR Bundle.