

A HANDS-ON TUTORIAL

Exporting a Lab Report

Assemble one diagnostic report and its lab results into a FHIR document Bundle

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

Panagiotis Liakakos

June 2026

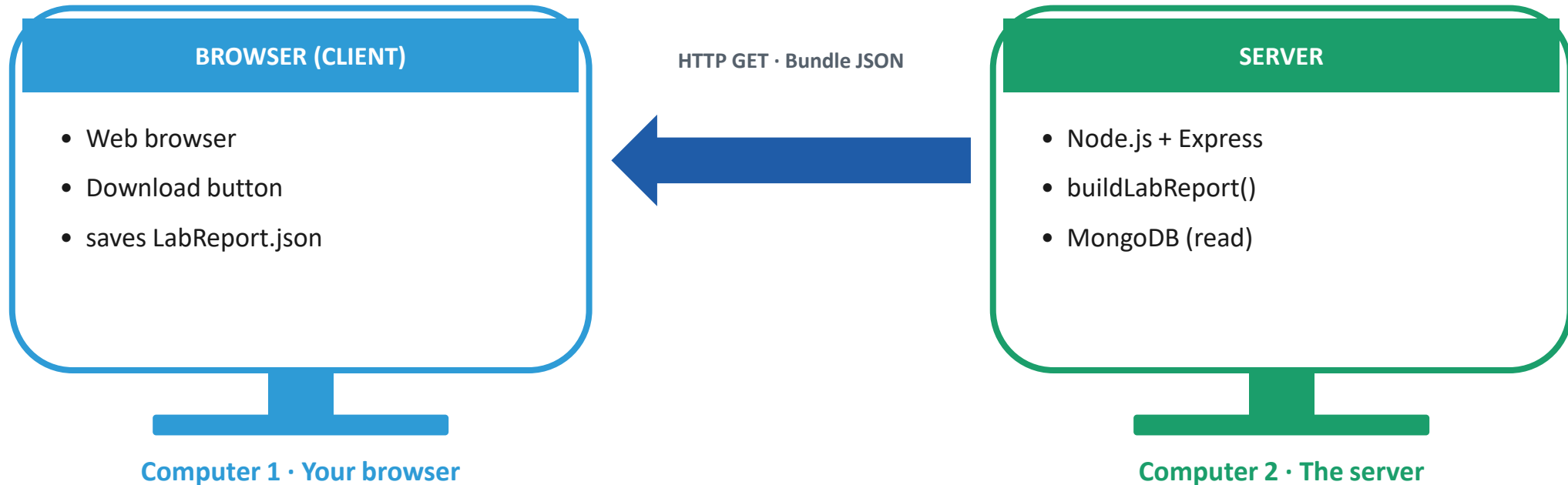
What you'll learn

By the end, you will be able to:

- Gather the chosen DiagnosticReport and its laboratory Observations from MongoDB.
- Pull in the Practitioners the report and its results reference.
- Reshape the report and wrap its results in a LOINC-coded Composition.
- Assemble a FHIR document Bundle and download it as a file.

Goal: turn one diagnostic report into a downloadable FHIR document Bundle.

How the data travels: server → browser



A patient's data is spread across many MongoDB collections. The server gathers the chosen report and its lab results, wraps them in a Composition, and returns one document Bundle that the browser saves as a file.

What you need to build a Lab Report



You only need three tools

- Mongoose .find() / .lean() — read the report, its Observations, and Practitioners.
- Plain JavaScript — reshape the report, build the Composition and 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 builder is ~190 lines in labReportService.js (buildLabReport), plus a one-line route and a small frontend download handler.

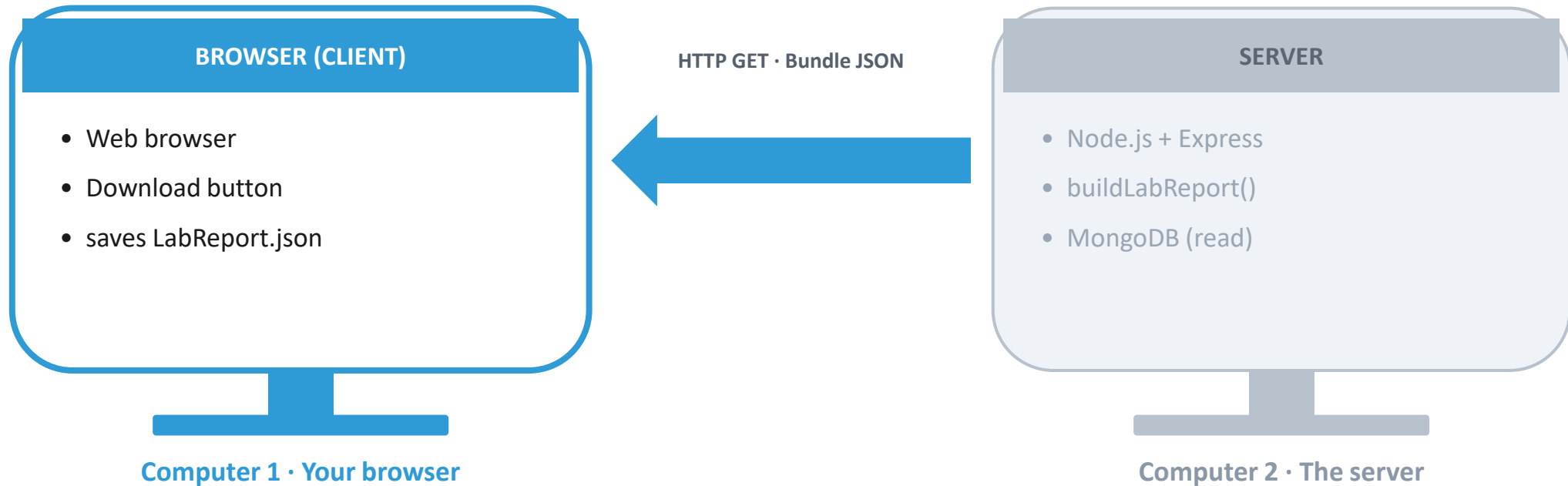
What you need to build a Lab Report

Five things not to forget

- Build from the stored rawFhir — it is the untouched original resource.
- Include only the report's laboratory Observations (category = laboratory).
- Wrap everything in a Bundle of type “document”, led by a Composition.
- Give every entry a fullUrl — use urn:uuid:<id>.
- Reshape the report: point it at the patient, keep only bundled results, link it to the Composition.

Prerequisites: Node 18+ and a running MongoDB (mongod) holding ingested data — then the three tools above work.
The core builder is ~190 lines in labReportService.js (buildLabReport), 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 report and saves the returned Bundle as a JSON file.

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

IN PLAIN ENGLISH

- The dialog lists the patient's DiagnosticReports; the user picks one to export.

frontend · DownloadDialog.jsx

```
// frontend – request the lab report, save it as a file
async function handleDownload(patientId, drId) {
  const endpoint =
    `${API}/patients/${patientId}/diagnostics/lab?drId=${drId}`;

  // ask the server to build the report
  const res = await fetch(endpoint);
  if (!res.ok) throw new Error((await res.json()).error);

  // 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 = "LabReport.json";
  a.click();
  URL.revokeObjectURL(url);
}
```

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

IN PLAIN ENGLISH

- The dialog lists the patient's DiagnosticReports; the user picks one to export.
- **Clicking Download requests the lab endpoint for that report, passing its id as a drId parameter.**

frontend · DownloadDialog.jsx

```
// frontend – request the lab report, save it as a file
async function handleDownload(patientId, drId) {
  const endpoint =
    `${API}/patients/${patientId}/diagnostics/lab?drId=${drId}`;

  // ask the server to build the report
  const res = await fetch(endpoint);
  if (!res.ok) throw new Error((await res.json()).error);

  // 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 = "LabReport.json";
  a.click();
  URL.revokeObjectURL(url);
}
```

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

IN PLAIN ENGLISH

- The dialog lists the patient's DiagnosticReports; the user picks one to export.
- Clicking Download requests the lab endpoint for that report, passing its id as a drId parameter.
- **The JSON response is read as a Blob — an in-memory file.**

frontend · DownloadDialog.jsx

```
// frontend — request the lab report, save it as a file
async function handleDownload(patientId, drId) {
  const endpoint =
    `${API}/patients/${patientId}/diagnostics/lab?drId=${drId}`;

  // ask the server to build the report
  const res = await fetch(endpoint);
  if (!res.ok) throw new Error((await res.json()).error);

  // 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 = "LabReport.json";
  a.click();
  URL.revokeObjectURL(url);
}
```

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

IN PLAIN ENGLISH

- The dialog lists the patient's DiagnosticReports; the user picks one to export.
- Clicking Download requests the lab endpoint for that report, passing its id as a drId parameter.
- The JSON response is read as a Blob — an in-memory file.
- **A temporary link triggers the browser's Save dialog (LabReport.json), then is cleaned up.**

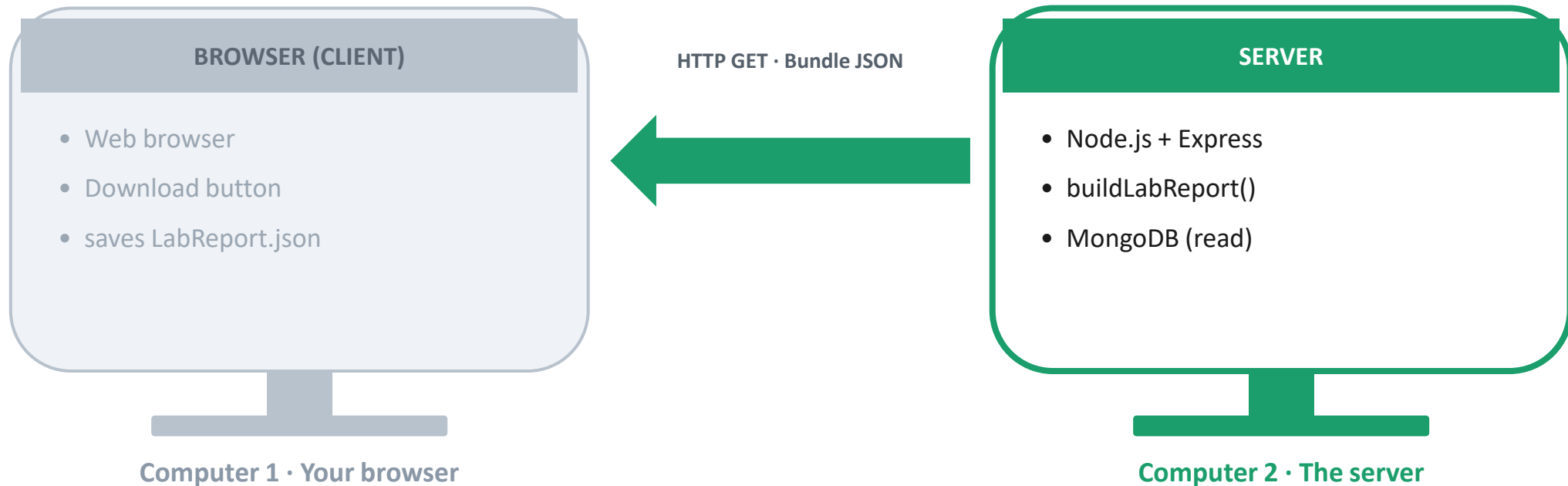
frontend · DownloadDialog.jsx

```
// frontend — request the lab report, save it as a file
async function handleDownload(patientId, drId) {
  const endpoint =
    `${API}/patients/${patientId}/diagnostics/lab?drId=${drId}`;

  // ask the server to build the report
  const res = await fetch(endpoint);
  if (!res.ok) throw new Error((await res.json()).error);

  // 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 = "LabReport.json";
  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 report and its results from MongoDB and assembles them into one document Bundle.

Step 2 — Gather the report and its results (server)

IN PLAIN ENGLISH

- Look up the patient and the chosen **DiagnosticReport**; if either is missing, stop with an error.

backend/services/labReportService.js

```
// backend – gather the report, its lab results, and staff
async function buildLabReport(patientId, drId) {
  const patient = await Patient.findOne({ patientId }).lean();
  if (!patient) throw new Error("Patient not found");
  const dr = await DiagnosticReport.findOne({
    patientId, "rawFhir.id": drId }).lean();
  if (!dr) throw new Error("DiagnosticReport not found");

  // observation ids the report points to
  const obsIds = (dr.rawFhir?.result || []).map(refToUuid);

  // only laboratory-category observations
  const obs = await Observation.find({
    patientId, "rawFhir.id": { $in: obsIds },
    "rawFhir.category.coding.code": "laboratory" }).lean();

  // practitioners referenced by the report + observations
  const pracRefs = [ ...(dr.rawFhir?.performer || []),
    ...obs.flatMap((o) => o.rawFhir?.performer || []) ];
  const pracIds = pracRefs.map(refToUuid).filter(Boolean);
  const pracs = await Practitioner.find(
    { "rawFhir.id": { $in: pracIds } }).lean();
}
```

Step 2 — Gather the report and its results (server)

IN PLAIN ENGLISH

- Look up the patient and the chosen DiagnosticReport; if either is missing, stop with an error.
- **Read the Observation ids the report points to in its result list.**

backend/services/labReportService.js

```
// backend – gather the report, its lab results, and staff
async function buildLabReport(patientId, drId) {
  const patient = await Patient.findOne({ patientId }).lean();
  if (!patient) throw new Error("Patient not found");
  const dr = await DiagnosticReport.findOne({
    patientId, "rawFhir.id": drId }).lean();
  if (!dr) throw new Error("DiagnosticReport not found");

  // observation ids the report points to
  const obsIds = (dr.rawFhir?.result || []).map(refToUuid);

  // only laboratory-category observations
  const obs = await Observation.find({
    patientId, "rawFhir.id": { $in: obsIds },
    "rawFhir.category.coding.code": "laboratory" }).lean();

  // practitioners referenced by the report + observations
  const pracRefs = [ ...(dr.rawFhir?.performer || []),
    ...obs.flatMap((o) => o.rawFhir?.performer || []) ];
  const pracIds = pracRefs.map(refToUuid).filter(Boolean);
  const pracs = await Practitioner.find(
    { "rawFhir.id": { $in: pracIds } }).lean();
}
```

Step 2 — Gather the report and its results (server)

IN PLAIN ENGLISH

- Look up the patient and the chosen DiagnosticReport; if either is missing, stop with an error.
- Read the Observation ids the report points to in its result list.
- **Fetch only those Observations whose category is laboratory.**

backend/services/labReportService.js

```
// backend – gather the report, its lab results, and staff
async function buildLabReport(patientId, drId) {
  const patient = await Patient.findOne({ patientId }).lean();
  if (!patient) throw new Error("Patient not found");
  const dr = await DiagnosticReport.findOne({
    patientId, "rawFhir.id": drId }).lean();
  if (!dr) throw new Error("DiagnosticReport not found");

  // observation ids the report points to
  const obsIds = (dr.rawFhir?.result || []).map(refToUuid);

  // only laboratory-category observations
  const obs = await Observation.find({
    patientId, "rawFhir.id": { $in: obsIds },
    "rawFhir.category.coding.code": "laboratory" }).lean();

  // practitioners referenced by the report + observations
  const pracRefs = [ ...(dr.rawFhir?.performer || []),
    ...obs.flatMap((o) => o.rawFhir?.performer || []) ];
  const pracIds = pracRefs.map(refToUuid).filter(Boolean);
  const pracs = await Practitioner.find(
    { "rawFhir.id": { $in: pracIds } }).lean();
}
```

Step 2 — Gather the report and its results (server)

IN PLAIN ENGLISH

- Look up the patient and the chosen DiagnosticReport; if either is missing, stop with an error.
- Read the Observation ids the report points to in its result list.
- Fetch only those Observations whose category is laboratory.
- **Gather the Practitioners referenced by the report and its Observations.**

backend/services/labReportService.js

```
// backend – gather the report, its lab results, and staff
async function buildLabReport(patientId, drId) {
  const patient = await Patient.findOne({ patientId }).lean();
  if (!patient) throw new Error("Patient not found");
  const dr = await DiagnosticReport.findOne({
    patientId, "rawFhir.id": drId }).lean();
  if (!dr) throw new Error("DiagnosticReport not found");

  // observation ids the report points to
  const obsIds = (dr.rawFhir?.result || []).map(refToUuid);

  // only laboratory-category observations
  const obs = await Observation.find({
    patientId, "rawFhir.id": { $in: obsIds },
    "rawFhir.category.coding.code": "laboratory" }).lean();

  // practitioners referenced by the report + observations
  const pracRefs = [ ...(dr.rawFhir?.performer || []),
    ...obs.flatMap((o) => o.rawFhir?.performer || []) ];
  const pracIds = pracRefs.map(refToUuid).filter(Boolean);
  const pracs = await Practitioner.find(
    { "rawFhir.id": { $in: pracIds } }).lean();
}
```

Step 3 — Prepare patient, org, and report (server)

IN PLAIN ENGLISH

- **Make the Patient universal: add the IPS profile, strip US-Core extensions.**

backend/services/labReportService.js

```
// make the Patient universal + a generator Organization
const patientRes = structuredClone(patient.rawFhir);
patientRes.meta.profile = ensureFirst(
  patientRes.meta.profile, IPS_PATIENT_PROFILE);
stripExt(patientRes, ["http://hl7.org/fhir/us/core/"]);

const org = {
  resourceType: "Organization",
  id: uuidv4(),
  name: "Lab-Report Generator Service",
};

// reshape the DiagnosticReport for the bundle
const drRes = structuredClone(dr.rawFhir);
drRes.subject = urn(patientRes.id, "Patient");
drRes.result = drRes.result.filter(
  (r) => obsIdsInBundle.has(refToUuid(r))); // keep bundled only
drRes.performer = drRes.performer
  .map(refToUuid).map((id) => urn(id, "Practitioner"));
drRes.extension = compositionLinks(compId); // 2 links -> Composition
```

Step 3 — Prepare patient, org, and report (server)

IN PLAIN ENGLISH

- Make the Patient universal: add the IPS profile, strip US-Core extensions.
- **Create a custodian Organization that generates the report.**

backend/services/labReportService.js

```
// make the Patient universal + a generator Organization
const patientRes = structuredClone(patient.rawFhir);
patientRes.meta.profile = ensureFirst(
  patientRes.meta.profile, IPS_PATIENT_PROFILE);
stripExt(patientRes, ["http://hl7.org/fhir/us/core/"]);

const org = {
  resourceType: "Organization",
  id: uuidv4(),
  name: "Lab-Report Generator Service",
};

// reshape the DiagnosticReport for the bundle
const drRes = structuredClone(dr.rawFhir);
drRes.subject = urn(patientRes.id, "Patient");
drRes.result = drRes.result.filter(
  (r) => obsIdsInBundle.has(refToUuid(r))); // keep bundled only
drRes.performer = drRes.performer
  .map(refToUuid).map((id) => urn(id, "Practitioner"));
drRes.extension = compositionLinks(compId); // 2 links -> Composition
```

Step 3 — Prepare patient, org, and report (server)

IN PLAIN ENGLISH

- Make the Patient universal: add the IPS profile, strip US-Core extensions.
- Create a custodian Organization that generates the report.
- **Reshape the DiagnosticReport: point it at the patient, keep only the bundled results.**

backend/services/labReportService.js

```
// make the Patient universal + a generator Organization
const patientRes = structuredClone(patient.rawFhir);
patientRes.meta.profile = ensureFirst(
  patientRes.meta.profile, IPS_PATIENT_PROFILE);
stripExt(patientRes, ["http://hl7.org/fhir/us/core/"]);

const org = {
  resourceType: "Organization",
  id: uuidv4(),
  name: "Lab-Report Generator Service",
};

// reshape the DiagnosticReport for the bundle
const drRes = structuredClone(dr.rawFhir);
drRes.subject = urn(patientRes.id, "Patient");
drRes.result = drRes.result.filter(
  (r) => obsIdsInBundle.has(refToUuid(r))); // keep bundled only
drRes.performer = drRes.performer
  .map(refToUuid).map((id) => urn(id, "Practitioner"));
drRes.extension = compositionLinks(compId); // 2 links -> Composition
```

Step 3 — Prepare patient, org, and report (server)

IN PLAIN ENGLISH

- Make the Patient universal: add the IPS profile, strip US-Core extensions.
- Create a custodian Organization that generates the report.
- Reshape the DiagnosticReport: point it at the patient, keep only the bundled results.
- **Rewrite its performer references to urn:uuid and link it to the Composition.**

backend/services/labReportService.js

```
// make the Patient universal + a generator Organization
const patientRes = structuredClone(patient.rawFhir);
patientRes.meta.profile = ensureFirst(
  patientRes.meta.profile, IPS_PATIENT_PROFILE);
stripExt(patientRes, ["http://hl7.org/fhir/us/core/"]);

const org = {
  resourceType: "Organization",
  id: uuidv4(),
  name: "Lab-Report Generator Service",
};

// reshape the DiagnosticReport for the bundle
const drRes = structuredClone(dr.rawFhir);
drRes.subject = urn(patientRes.id, "Patient");
drRes.result = drRes.result.filter(
  (r) => obsIdsInBundle.has(refToUuid(r))); // keep bundled only
drRes.performer = drRes.performer
  .map(refToUuid).map((id) => urn(id, "Practitioner"));
drRes.extension = compositionLinks(compId); // 2 links -> Composition
```

Step 4 — Build the report's Composition (server)

IN PLAIN ENGLISH

- Build a Composition typed as a Laboratory report (LOINC 11502-2).

backend/services/labReportService.js

```
// the Composition – the lab report's document header
const comp = {
  resourceType: "Composition",
  status: "final",
  type: { coding: [ loinc("11502-2", "Laboratory report") ] },
  subject: urn(patientRes.id, "Patient"),
  author: [urn(org.id, "Organization")],
  custodian: urn(org.id, "Organization"),
  section: [{
    title: "Laboratory Results",
    code: { coding: [ loinc("11502-2", "Laboratory report") ] },
    focus: urn(drRes.id, "DiagnosticReport"),
    entry: obs.map((o) => urn(o.rawFhir.id, "Observation")),
  }],
};
```

Step 4 — Build the report's Composition (server)

IN PLAIN ENGLISH

- Build a Composition typed as a Laboratory report (LOINC 11502-2).
- **Record the subject — the patient — and the author and custodian organisation.**

backend/services/labReportService.js

```
// the Composition – the lab report's document header
const comp = {
  resourceType: "Composition",
  status: "final",
  type: { coding: [ loinc("11502-2", "Laboratory report") ] },
  subject: urn(patientRes.id, "Patient"),
  author: [urn(org.id, "Organization")],
  custodian: urn(org.id, "Organization"),
  section: [{
    title: "Laboratory Results",
    code: { coding: [ loinc("11502-2", "Laboratory report") ] },
    focus: urn(drRes.id, "DiagnosticReport"),
    entry: obs.map((o) => urn(o.rawFhir.id, "Observation")),
  }],
};
```

Step 4 — Build the report's Composition (server)

IN PLAIN ENGLISH

- Build a Composition typed as a Laboratory report (LOINC 11502-2).
- Record the subject — the patient — and the author and custodian organisation.
- **Add one “Laboratory Results” section focused on the DiagnosticReport.**

backend/services/labReportService.js

```
// the Composition – the lab report's document header
const comp = {
  resourceType: "Composition",
  status: "final",
  type: { coding: [ loinc("11502-2", "Laboratory report") ] },
  subject: urn(patientRes.id, "Patient"),
  author: [urn(org.id, "Organization")],
  custodian: urn(org.id, "Organization"),
  section: [{
    title: "Laboratory Results",
    code: { coding: [ loinc("11502-2", "Laboratory report") ] },
    focus: urn(drRes.id, "DiagnosticReport"),
    entry: obs.map((o) => urn(o.rawFhir.id, "Observation")),
  }],
};
```

Step 4 — Build the report's Composition (server)

IN PLAIN ENGLISH

- Build a Composition typed as a Laboratory report (LOINC 11502-2).
- Record the subject — the patient — and the author and custodian organisation.
- Add one “Laboratory Results” section focused on the DiagnosticReport.
- **List the laboratory Observations as the section's entries.**

backend/services/labReportService.js

```
// the Composition – the lab report's document header
const comp = {
  resourceType: "Composition",
  status: "final",
  type: { coding: [ loinc("11502-2", "Laboratory report") ] },
  subject: urn(patientRes.id, "Patient"),
  author: [urn(org.id, "Organization")],
  custodian: urn(org.id, "Organization"),
  section: [{
    title: "Laboratory Results",
    code: { coding: [ loinc("11502-2", "Laboratory report") ] },
    focus: urn(drRes.id, "DiagnosticReport"),
    entry: obs.map((o) => urn(o.rawFhir.id, "Observation")),
  }],
};
```

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

IN PLAIN ENGLISH

- Return a Bundle object of type “document”.

backend/services/labReportService.js

```
// assemble the lab report document Bundle
const bundleId = uuidv4();
return {
  resourceType: "Bundle",
  id: bundleId,
  type: "document",          // a clinical document
  timestamp: compDate,
  identifier: {
    system: "urn:ietf:rfc:3986",
    value: `urn:uuid:${bundleId}`,
  },
  entry: [
    { fullUrl: `urn:uuid:${comp.id}`,      resource: comp },
    { fullUrl: `urn:uuid:${patientRes.id}`, resource: patientRes },
    { fullUrl: `urn:uuid:${org.id}`,        resource: org },
    ...obsEntries,      // Observations
    ...pracEntries,     // Practitioners
    { fullUrl: `urn:uuid:${drRes.id}`,      resource: drRes },
  ],
};
```

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

IN PLAIN ENGLISH

- Return a Bundle object of type “document”.
- **Stamp it with a timestamp and a unique identifier.**

backend/services/labReportService.js

```
// assemble the lab report document Bundle
const bundleId = uuidv4();
return {
  resourceType: "Bundle",
  id: bundleId,
  type: "document", // a clinical document
  timestamp: compDate,
  identifier: {
    system: "urn:ietf:rfc:3986",
    value: `urn:uuid:${bundleId}`,
  },
  entry: [
    { fullUrl: `urn:uuid:${comp.id}`, resource: comp },
    { fullUrl: `urn:uuid:${patientRes.id}`, resource: patientRes },
    { fullUrl: `urn:uuid:${org.id}`, resource: org },
    ...obsEntries, // Observations
    ...pracEntries, // Practitioners
    { fullUrl: `urn:uuid:${drRes.id}`, resource: drRes },
  ],
};
```

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

IN PLAIN ENGLISH

- Return a Bundle object of type “document”.
- Stamp it with a timestamp and a unique identifier.
- **List the Composition first, then the Patient and Organization.**

backend/services/labReportService.js

```
// assemble the lab report document Bundle
const bundleId = uuidv4();
return {
  resourceType: "Bundle",
  id: bundleId,
  type: "document",           // a clinical document
  timestamp: compDate,
  identifier: {
    system: "urn:iETF:rFC:3986",
    value: `urn:uuid:${bundleId}`,
  },
  entry: [
    { fullUrl: `urn:uuid:${comp.id}`,      resource: comp },
    { fullUrl: `urn:uuid:${patientRes.id}`, resource: patientRes },
    { fullUrl: `urn:uuid:${org.id}`,        resource: org },
    ...obsEntries,           // Observations
    ...pracEntries,         // Practitioners
    { fullUrl: `urn:uuid:${drRes.id}`,      resource: drRes },
  ],
};
```

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

IN PLAIN ENGLISH

- Return a Bundle object of type “document”.
- Stamp it with a timestamp and a unique identifier.
- List the Composition first, then the Patient and Organization.
- **Then the Observations, Practitioners, and the reshaped DiagnosticReport.**

backend/services/labReportService.js

```
// assemble the lab report document Bundle
const bundleId = uuidv4();
return {
  resourceType: "Bundle",
  id: bundleId,
  type: "document",          // a clinical document
  timestamp: compDate,
  identifier: {
    system: "urn:ietf:rfc:3986",
    value: `urn:uuid:${bundleId}`,
  },
  entry: [
    { fullUrl: `urn:uuid:${comp.id}`, resource: comp },
    { fullUrl: `urn:uuid:${patientRes.id}`, resource: patientRes },
    { fullUrl: `urn:uuid:${org.id}`, resource: org },
    ...obsEntries,          // Observations
    ...pracEntries,         // Practitioners
    { fullUrl: `urn:uuid:${drRes.id}`, resource: drRes },
  ],
};
```

The missing link — expose the endpoint

IN PLAIN ENGLISH

- One route maps GET `/api/patients/:patientId/diagnostics/lab` to the controller.

```
backend/routes/patientRoutes.js + labReportController.js

// ---- expose the report: route + controller ----
// backend/routes/patientRoutes.js
router.get("/:patientId/diagnostics/lab", downloadLabReport);

// backend/controllers/labReportController.js
async function downloadLabReport(req, res, next) {
  try {
    const { patientId } = req.params;
    const { drId } = req.query; // report id chosen in the UI
    const bundle = await buildLabReport(patientId, drId);
    res.set("Content-Type", "application/fhir+json");
    res.json(bundle);          // browser downloads this
  } catch (err) {
    next(err);
  }
}
// GET /api/patients/<id>/diagnostics/lab?drId=<id>
```

The missing link — expose the endpoint

IN PLAIN ENGLISH

- One route maps GET `/api/patients/:patientId/diagnostics/lab` to the controller.
- **The controller reads the `patientId` from the URL.**

```
backend/routes/patientRoutes.js + labReportController.js

// ---- expose the report: route + controller ----
//      backend/routes/patientRoutes.js
router.get("/:patientId/diagnostics/lab", downloadLabReport);

//      backend/controllers/labReportController.js
async function downloadLabReport(req, res, next) {
  try {
    const { patientId } = req.params;
    const { drId } = req.query; // report id chosen in the UI
    const bundle = await buildLabReport(patientId, drId);
    res.set("Content-Type", "application/fhir+json");
    res.json(bundle);          // browser downloads this
  } catch (err) {
    next(err);
  }
}
// GET /api/patients/<id>/diagnostics/lab?drId=<id>
```

The missing link — expose the endpoint

Share
EHR Xchange FormatFORTH
FOUNDATION FOR RESEARCH AND TECHNOLOGY - HELLAS

IN PLAIN ENGLISH

- One route maps GET `/api/patients/:patientId/diagnostics/lab` to the controller.
- The controller reads the `patientId` from the URL.
- **It reads the chosen report id from the `drId` query, then calls the build service.**

```
backend/routes/patientRoutes.js + labReportController.js
```

```
// ---- expose the report: route + controller ----  
//      backend/routes/patientRoutes.js  
router.get("/:patientId/diagnostics/lab", downloadLabReport);
```

```
//      backend/controllers/labReportController.js  
async function downloadLabReport(req, res, next) {  
  try {  
    const { patientId } = req.params;  
    const { drId } = req.query; // report id chosen in the UI  
    const bundle = await buildLabReport(patientId, drId);  
    res.set("Content-Type", "application/fhir+json");  
    res.json(bundle);           // browser downloads this  
  } catch (err) {  
    next(err);  
  }  
}  
// GET /api/patients/<id>/diagnostics/lab?drId=<id>
```

The missing link — expose the endpoint

Share
EHR Xchange Format

FORTH

FOUNDATION FOR RESEARCH AND TECHNOLOGY - HELLAS

IN PLAIN ENGLISH

- One route maps GET `/api/patients/:patientId/diagnostics/lab` to the controller.
- The controller reads the `patientId` from the URL.
- It reads the chosen report id from the `drId` query, then calls the build service.
- **It sends the finished Bundle back as `application/fhir+json`.**

backend/routes/patientRoutes.js + labReportController.js

```
// ---- expose the report: route + controller ----
//      backend/routes/patientRoutes.js
router.get("/:patientId/diagnostics/lab", downloadLabReport);

//      backend/controllers/labReportController.js
async function downloadLabReport(req, res, next) {
  try {
    const { patientId } = req.params;
    const { drId } = req.query; // report id chosen in the UI
    const bundle = await buildLabReport(patientId, drId);
    res.set("Content-Type", "application/fhir+json");
    res.json(bundle); // browser downloads this
  } catch (err) {
    next(err);
  }
}
// GET /api/patients/<id>/diagnostics/lab?drId=<id>
```

What you learned

The whole pipeline, in six moves

- A Download button requests the lab endpoint for a chosen report and saves the response.
- The server looks up the patient and that DiagnosticReport.
- It collects the report's laboratory Observations and the Practitioners they reference.
- It reshapes the report and wraps its results in a LOINC-coded Composition.
- It wraps everything in a FHIR Bundle of type "document".
- A small route and controller expose it at GET /api/patients/:patientId/diagnostics/lab.

You can now turn a diagnostic report into a downloadable FHIR document Bundle.