

A HANDS-ON TUTORIAL

Exporting a Patient Summary (IPS)

Assemble a patient's key records into one standard IPS document Bundle

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

Panagiotis Liakakos

June 2026

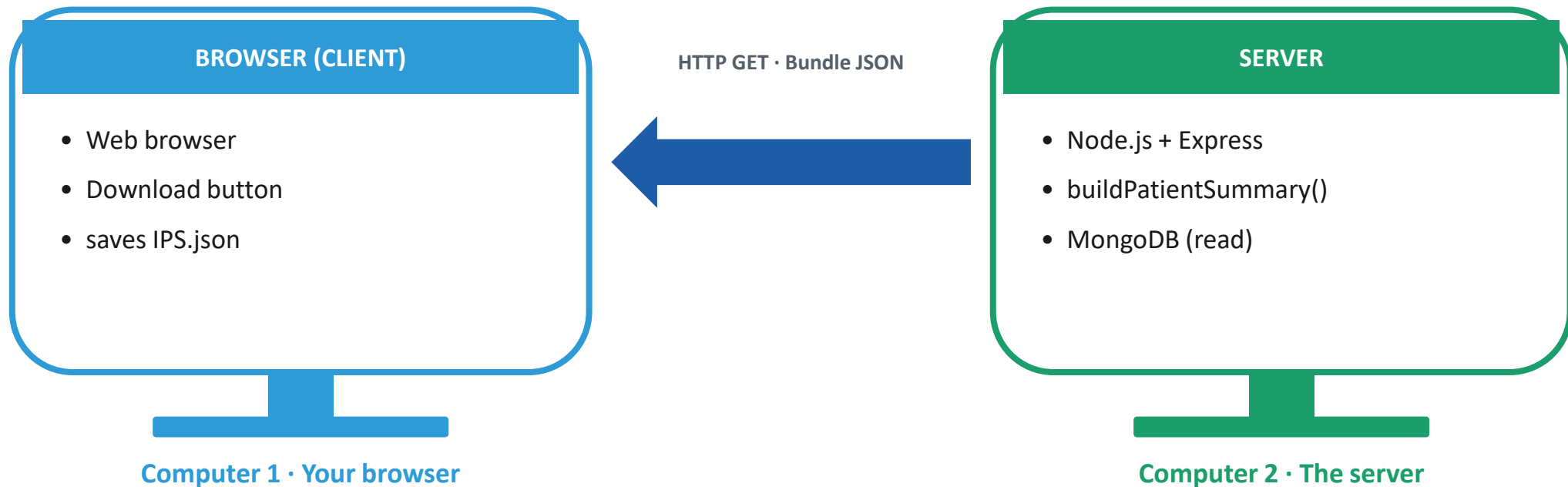
What you'll learn

By the end, you will be able to:

- Gather the patient and the mandatory IPS resources from MongoDB.
- Add any optional sections the user ticks in the dialog.
- Group resources into LOINC-coded sections under a Composition.
- Assemble a FHIR document Bundle and download it as a file.

Goal: turn a patient's records into one standard IPS document Bundle.

How the data travels: server → browser



A patient's data is spread across many MongoDB collections. The server gathers the IPS resources, arranges them into sections under a Composition, and returns one document Bundle that the browser saves as a file.

What you need to build an IPS



You only need three tools

- Mongoose `.find()` / `.lean()` — read each collection's stored resource.
- Plain JavaScript — assemble sections, a 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 ~200 lines in `ipsService.js` (`buildPatientSummary`), plus a one-line route and a small frontend download handler.

What you need to build an IPS

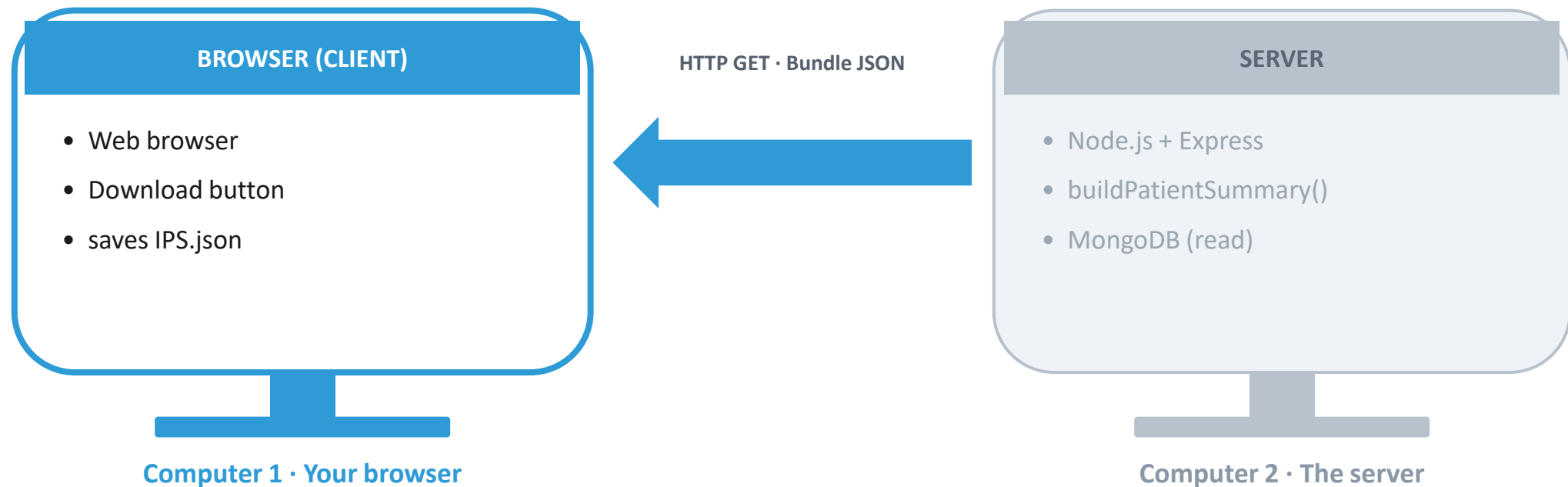


Five things not to forget

- Build from the stored rawFhir — it is the untouched original resource.
- Three sections are mandatory: Medications, Allergies, Problems.
- Group everything in a Bundle of type “document”, led by a Composition.
- Give every entry a fullUrl — use urn:uuid:<id>.
- Make the Patient universal: add the IPS profile, strip US-Core extensions.

Prerequisites: Node 18+ and a running MongoDB (mongod) holding ingested data — then the three tools above work.
The core builder is ~200 lines in ipsService.js (buildPatientSummary), 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 summary and saves the returned Bundle as a JSON file.

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

IN PLAIN ENGLISH

- The backend always builds the mandatory sections; in the dialog, present mandatory types are pre-ticked and optional ones can be added.

frontend · DownloadDialog.jsx

```
// frontend — request the IPS, save it as a file
async function handleDownload(patientId, selected) {
  const inc = Object.keys(selected)
    .filter((k) => selected[k]).join(",");
  const endpoint =
    `${API}/patients/${patientId}/summary?include=${inc}`;

  // ask the server to build the summary
  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 = "IPS.json";
  a.click();
  URL.revokeObjectURL(url);
}
```

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

IN PLAIN ENGLISH

- The backend always builds the mandatory sections; in the dialog, present mandatory types are pre-ticked and optional ones can be added.
- **Clicking Download requests the summary endpoint for that patient, passing the chosen optional resources as an include parameter.**

frontend · DownloadDialog.jsx

```
// frontend – request the IPS, save it as a file
async function handleDownload(patientId, selected) {
  const inc = Object.keys(selected)
    .filter((k) => selected[k]).join(",");
  const endpoint =
    `${API}/patients/${patientId}/summary?include=${inc}`;

  // ask the server to build the summary
  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 = "IPS.json";
  a.click();
  URL.revokeObjectURL(url);
}
```

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

IN PLAIN ENGLISH

- The backend always builds the mandatory sections; in the dialog, present mandatory types are pre-ticked and optional ones can be added.
- Clicking Download requests the summary endpoint for that patient, passing the chosen optional resources as an include parameter.
- **The JSON response is read as a Blob — an in-memory file.**

frontend · DownloadDialog.jsx

```
// frontend — request the IPS, save it as a file
async function handleDownload(patientId, selected) {
  const inc = Object.keys(selected)
    .filter((k) => selected[k]).join(",");
  const endpoint =
    `${API}/patients/${patientId}/summary?include=${inc}`;

  // ask the server to build the summary
  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 = "IPS.json";
  a.click();
  URL.revokeObjectURL(url);
}
```

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

IN PLAIN ENGLISH

- The backend always builds the mandatory sections; in the dialog, present mandatory types are pre-ticked and optional ones can be added.
- Clicking Download requests the summary endpoint for that patient, passing the chosen optional resources as an include parameter.
- The JSON response is read as a Blob — an in-memory file.
- **A temporary link triggers the browser's Save dialog (IPS.json), then is cleaned up.**

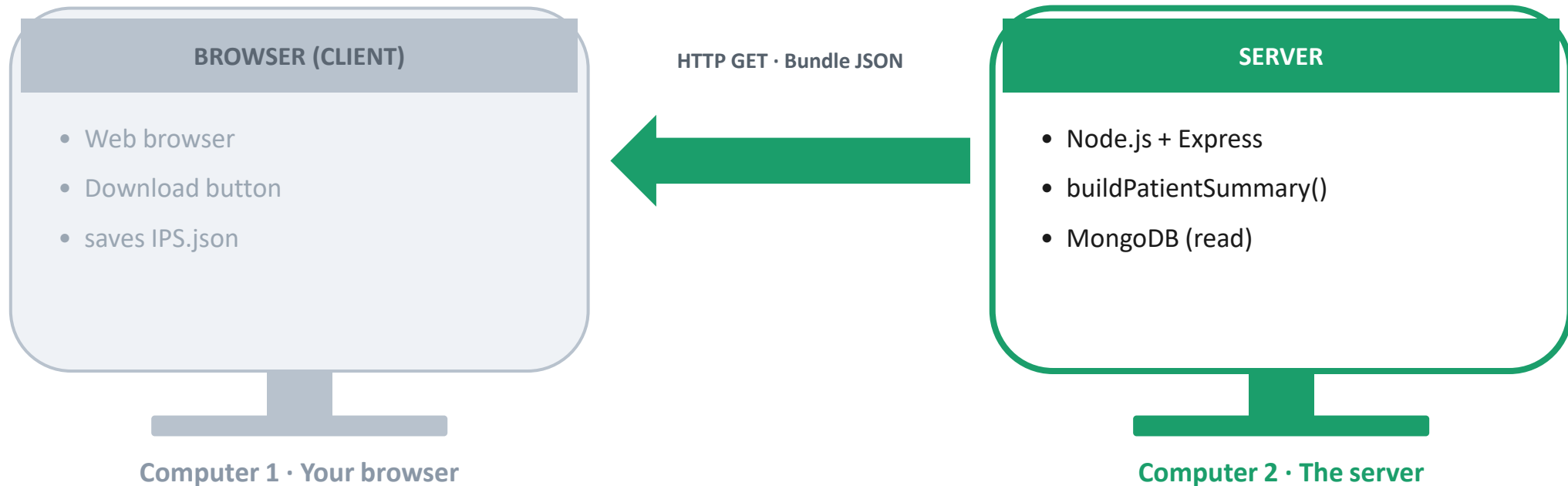
frontend · DownloadDialog.jsx

```
// frontend — request the IPS, save it as a file
async function handleDownload(patientId, selected) {
  const inc = Object.keys(selected)
    .filter((k) => selected[k]).join(",");
  const endpoint =
    `${API}/patients/${patientId}/summary?include=${inc}`;

  // ask the server to build the summary
  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 = "IPS.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 patient's resources from MongoDB and assembles them into one IPS document Bundle.

Step 2 — Gather the IPS resources (server)

IN PLAIN ENGLISH

- Look up the patient; if there is none (or no rawFhir), stop with an error.

backend/services/ipsService.js

```
// backend — fetch the patient + IPS resources
async function buildPatientSummary(patientId, include = []) {
  const patientDoc = await Patient.findOne({ patientId }).lean();
  if (!patientDoc?.rawFhir) throw new Error("Patient not found");

  // three mandatory IPS sections
  const [meds, allergies, problems] = await Promise.all([
    MedicationRequest.find({ patientId }).lean(),
    AllergyIntolerance.find({ patientId }).lean(),
    Condition.find({ patientId }).lean(),
  ]);

  // optional sections the user ticked (?include=...)
  const wanted = include.filter((t) => OPTIONAL_RESOURCES[t]);
  const optional = await Promise.all(wanted.map(
    (t) => OPTIONAL_RESOURCES[t].model.find({ patientId }).lean(),
  ));

  // practitioners referenced by the medications
  const ids = meds.map((m) => refToUuid(m.rawFhir?.requester));
  const pracs = await Practitioner
    .find({ "rawFhir.id": { $in: ids.filter(Boolean) } }).lean();
}
```

Step 2 — Gather the IPS resources (server)

IN PLAIN ENGLISH

- Look up the patient; if there is none (or no rawFhir), stop with an error.
- **Fetch the three mandatory collections — Medications, Allergies, Problems — in parallel.**

backend/services/ipsService.js

```
// backend — fetch the patient + IPS resources
async function buildPatientSummary(patientId, include = []) {
  const patientDoc = await Patient.findOne({ patientId }).lean();
  if (!patientDoc?.rawFhir) throw new Error("Patient not found");

  // three mandatory IPS sections
  const [meds, allergies, problems] = await Promise.all([
    MedicationRequest.find({ patientId }).lean(),
    AllergyIntolerance.find({ patientId }).lean(),
    Condition.find({ patientId }).lean(),
  ]);

  // optional sections the user ticked (?include=...)
  const wanted = include.filter((t) => OPTIONAL_RESOURCES[t]);
  const optional = await Promise.all(wanted.map(
    (t) => OPTIONAL_RESOURCES[t].model.find({ patientId }).lean(),
  ));

  // practitioners referenced by the medications
  const ids = meds.map((m) => refToUuid(m.rawFhir?.requester));
  const pracs = await Practitioner
    .find({ "rawFhir.id": { $in: ids.filter(Boolean) } }).lean();
}
```

Step 2 — Gather the IPS resources (server)

IN PLAIN ENGLISH

- Look up the patient; if there is none (or no rawFhir), stop with an error.
- Fetch the three mandatory collections — Medications, Allergies, Problems — in parallel.
- **Fetch any optional collections the user ticked, ignoring unknown types.**

backend/services/ipsService.js

```
// backend — fetch the patient + IPS resources
async function buildPatientSummary(patientId, include = []) {
  const patientDoc = await Patient.findOne({ patientId }).lean();
  if (!patientDoc?.rawFhir) throw new Error("Patient not found");

  // three mandatory IPS sections
  const [meds, allergies, problems] = await Promise.all([
    MedicationRequest.find({ patientId }).lean(),
    AllergyIntolerance.find({ patientId }).lean(),
    Condition.find({ patientId }).lean(),
  ]);

  // optional sections the user ticked (?include=...)
  const wanted = include.filter((t) => OPTIONAL_RESOURCES[t]);
  const optional = await Promise.all(wanted.map(
    (t) => OPTIONAL_RESOURCES[t].model.find({ patientId }).lean(),
  ));

  // practitioners referenced by the medications
  const ids = meds.map((m) => refToUuid(m.rawFhir?.requester));
  const pracs = await Practitioner
    .find({ "rawFhir.id": { $in: ids.filter(Boolean) } }).lean();
}
```

Step 2 — Gather the IPS resources (server)

IN PLAIN ENGLISH

- Look up the patient; if there is none (or no rawFhir), stop with an error.
- Fetch the three mandatory collections — Medications, Allergies, Problems — in parallel.
- Fetch any optional collections the user ticked, ignoring unknown types.
- **Gather the Practitioners referenced by the medication requests.**

backend/services/ipsService.js

```
// backend — fetch the patient + IPS resources
async function buildPatientSummary(patientId, include = []) {
  const patientDoc = await Patient.findOne({ patientId }).lean();
  if (!patientDoc?.rawFhir) throw new Error("Patient not found");

  // three mandatory IPS sections
  const [meds, allergies, problems] = await Promise.all([
    MedicationRequest.find({ patientId }).lean(),
    AllergyIntolerance.find({ patientId }).lean(),
    Condition.find({ patientId }).lean(),
  ]);

  // optional sections the user ticked (?include=...)
  const wanted = include.filter((t) => OPTIONAL_RESOURCES[t]);
  const optional = await Promise.all(wanted.map(
    (t) => OPTIONAL_RESOURCES[t].model.find({ patientId }).lean(),
  ));

  // practitioners referenced by the medications
  const ids = meds.map((m) => refToUuid(m.rawFhir?.requester));
  const pracs = await Practitioner
    .find({ "rawFhir.id": { $in: ids.filter(Boolean) } }).lean();
}
```

Step 3 — Build the clinical sections (server)

IN PLAIN ENGLISH

- Each clinical area becomes a section, tagged with a LOINC code.

backend/services/ipsService.js

```
// turn each group of docs into a LOINC-coded section
function addSection(title, loinc, display, docs) {
  const section = {
    title,
    code: { coding: [{ system: "http://loinc.org",
      code: loinc, display }] },
  };
  if (docs.length) {
    section.entry = [];
    docs.forEach((d) => {
      if (!d.rawFhir) return; // e.g. Encounter has none
      const res = structuredClone(d.rawFhir);
      res.id = res.id || String(d._id);
      const full = `urn:uuid:${res.id}`;
      section.entry.push({ reference: full });
      bundleEntries.push({ fullUrl: full, resource: res });
    });
  } else {
    section.emptyReason = UNAVAILABLE; // "no data" marker
  }
  sections.push(section);
}
```

Step 3 — Build the clinical sections (server)

IN PLAIN ENGLISH

- Each clinical area becomes a section, tagged with a LOINC code.
- **If the section has data, list each resource by a urn:uuid reference.**

backend/services/ipsService.js

```
// turn each group of docs into a LOINC-coded section
function addSection(title, loinc, display, docs) {
  const section = {
    title,
    code: { coding: [{ system: "http://loinc.org",
      code: loinc, display }] },
  };
  if (docs.length) {
    section.entry = [];
    docs.forEach((d) => {
      if (!d.rawFhir) return; // e.g. Encounter has none
      const res = structuredClone(d.rawFhir);
      res.id = res.id || String(d._id);
      const full = `urn:uuid:${res.id}`;
      section.entry.push({ reference: full });
      bundleEntries.push({ fullUrl: full, resource: res });
    });
  } else {
    section.emptyReason = UNAVAILABLE; // "no data" marker
  }
  sections.push(section);
}
```

Step 3 — Build the clinical sections (server)

IN PLAIN ENGLISH

- Each clinical area becomes a section, tagged with a LOINC code.
- If the section has data, list each resource by a urn:uuid reference.
- **Push those resources into the bundle as entries.**

backend/services/ipsService.js

```
// turn each group of docs into a LOINC-coded section
function addSection(title, loinc, display, docs) {
  const section = {
    title,
    code: { coding: [{ system: "http://loinc.org",
      code: loinc, display }] },
  };
  if (docs.length) {
    section.entry = [];
    docs.forEach((d) => {
      if (!d.rawFhir) return; // e.g. Encounter has none
      const res = structuredClone(d.rawFhir);
      res.id = res.id || String(d._id);
      const full = `urn:uuid:${res.id}`;
      section.entry.push({ reference: full });
      bundleEntries.push({ fullUrl: full, resource: res });
    });
  } else {
    section.emptyReason = UNAVAILABLE; // "no data" marker
  }
  sections.push(section);
}
```

Step 3 — Build the clinical sections (server)

IN PLAIN ENGLISH

- Each clinical area becomes a section, tagged with a LOINC code.
- If the section has data, list each resource by a urn:uuid reference.
- Push those resources into the bundle as entries.
- **If empty, record an “Unavailable” reason instead of an entry.**

backend/services/ipsService.js

```
// turn each group of docs into a LOINC-coded section
function addSection(title, loinc, display, docs) {
  const section = {
    title,
    code: { coding: [{ system: "http://loinc.org",
                                                              code: loinc, display }] },
  };
  if (docs.length) {
    section.entry = [];
    docs.forEach((d) => {
      if (!d.rawFhir) return; // e.g. Encounter has none
      const res = structuredClone(d.rawFhir);
      res.id = res.id || String(d._id);
      const full = `urn:uuid:${res.id}`;
      section.entry.push({ reference: full });
      bundleEntries.push({ fullUrl: full, resource: res });
    });
  } else {
    section.emptyReason = UNAVAILABLE; // "no data" marker
  }
  sections.push(section);
}
```

Step 4 — Assemble the Composition (server)

IN PLAIN ENGLISH

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

backend/services/ipsService.js

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

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

// the document header that ties the sections together
const comp = {
  resourceType: "Composition",
  status: "final",
  type: { coding: [{ system: "http://loinc.org", code: "60591-5",
                    display: "Patient summary document" }] },
  subject: { reference: `urn:uuid:${patientRes.id}` },
  author:   [{ reference: `urn:uuid:${org.id}` }],
  custodian: { reference: `urn:uuid:${org.id}` },
  section:  sections,
};
```

Step 4 — Assemble the Composition (server)

IN PLAIN ENGLISH

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

backend/services/ipsService.js

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

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

// the document header that ties the sections together
const comp = {
  resourceType: "Composition",
  status: "final",
  type: { coding: [{ system: "http://loinc.org", code: "60591-5",
                    display: "Patient summary document" }] },
  subject: { reference: `urn:uuid:${patientRes.id}` },
  author:   [{ reference: `urn:uuid:${org.id}` }],
  custodian: { reference: `urn:uuid:${org.id}` },
  section:  sections,
};
```

Step 4 — Assemble the Composition (server)

IN PLAIN ENGLISH

- Make the Patient universal: add the IPS profile, strip US-Core extensions.
- Create a custodian Organization that authors the summary.
- **Build a Composition: type “Patient summary”, subject, author, custodian.**

backend/services/ipsService.js

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

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

// the document header that ties the sections together
const comp = {
  resourceType: "Composition",
  status: "final",
  type: { coding: [{ system: "http://loinc.org", code: "60591-5",
                    display: "Patient summary document" }] },
  subject: { reference: `urn:uuid:${patientRes.id}` },
  author:   [{ reference: `urn:uuid:${org.id}` }],
  custodian: { reference: `urn:uuid:${org.id}` },
  section:  sections,
};
```

Step 4 — Assemble the Composition (server)

IN PLAIN ENGLISH

- Make the Patient universal: add the IPS profile, strip US-Core extensions.
- Create a custodian Organization that authors the summary.
- Build a Composition: type “Patient summary”, subject, author, custodian.
- **Attach the sections built in Step 3 to the Composition.**

backend/services/ipsService.js

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

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

// the document header that ties the sections together
const comp = {
  resourceType: "Composition",
  status: "final",
  type: { coding: [{ system: "http://loinc.org", code: "60591-5",
                    display: "Patient summary document" }] },
  subject: { reference: `urn:uuid:${patientRes.id}` },
  author:   [{ reference: `urn:uuid:${org.id}` }],
  custodian: { reference: `urn:uuid:${org.id}` },
  section:  sections,
};
```

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

IN PLAIN ENGLISH

- Return a Bundle object of type “document”.

backend/services/ipsService.js

```
// assemble the IPS document Bundle
const bundleId = uuidv4();
return {
  resourceType: "Bundle",
  id: bundleId,
  type: "document", // IPS is a document Bundle
  timestamp: new Date().toISOString(),
  meta: { profile: [IPS_BUNDLE_PROFILE] },
  identifier: {
    system: "urn:ietf:rhc: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 },
    ...bundleEntries, // section resources + practitioners
  ],
};
```

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

IN PLAIN ENGLISH

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

backend/services/ipsService.js

```
// assemble the IPS document Bundle
const bundleId = uuidv4();
return {
  resourceType: "Bundle",
  id: bundleId,
  type: "document", // IPS is a document Bundle
  timestamp: new Date().toISOString(),
  meta: { profile: [IPS_BUNDLE_PROFILE] },
  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 },
    ...bundleEntries, // section resources + practitioners
  ],
};
```

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

IN PLAIN ENGLISH

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

backend/services/ipsService.js

```
// assemble the IPS document Bundle
const bundleId = uuidv4();
return {
  resourceType: "Bundle",
  id: bundleId,
  type: "document",           // IPS is a document Bundle
  timestamp: new Date().toISOString(),
  meta: { profile: [IPS_BUNDLE_PROFILE] },
  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 },
    ...bundleEntries,          // section resources + practitioners
  ],
};
```

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

IN PLAIN ENGLISH

- Return a Bundle object of type “document”.
- Stamp it with the IPS profile, a timestamp, and a unique identifier.
- List the Composition first, then Patient, Organization, and section resources.
- **Each entry pairs a urn:uuid fullUrl with its resource.**

backend/services/ipsService.js

```
// assemble the IPS document Bundle
const bundleId = uuidv4();
return {
  resourceType: "Bundle",
  id: bundleId,
  type: "document",           // IPS is a document Bundle
  timestamp: new Date().toISOString(),
  meta: { profile: [IPS_BUNDLE_PROFILE] },
  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 },
    ...bundleEntries,          // section resources + practitioners
  ],
};
```

The missing link — expose the endpoint

IN PLAIN ENGLISH

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

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

// ---- expose the summary: route + controller ----
// backend/routes/patientRoutes.js
router.get("/:patientId/summary", downloadPatientSummary);

// backend/controllers/ipsController.js
async function downloadPatientSummary(req, res, next) {
  try {
    const { patientId } = req.params;
    const { include } = req.query; // "Observation,Procedure"
    const bundle = await buildPatientSummary(patientId, include);
    res.set("Content-Type", "application/fhir+json");
    res.json(bundle); // browser downloads this
  } catch (err) {
    next(err);
  }
}
// GET /api/patients/<id>/summary -> the IPS Bundle
```

The missing link — expose the endpoint

IN PLAIN ENGLISH

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

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

// ---- expose the summary: route + controller ----
//      backend/routes/patientRoutes.js
router.get("/:patientId/summary", downloadPatientSummary);

//      backend/controllers/ipsController.js
async function downloadPatientSummary(req, res, next) {
  try {
    const { patientId } = req.params;
    const { include } = req.query; // "Observation,Procedure"
    const bundle = await buildPatientSummary(patientId, include);
    res.set("Content-Type", "application/fhir+json");
    res.json(bundle);           // browser downloads this
  } catch (err) {
    next(err);
  }
}
// GET /api/patients/<id>/summary -> the IPS Bundle
```

The missing link — expose the endpoint

IN PLAIN ENGLISH

- One route maps GET `/api/patients/:patientId/summary` to the controller.
- The controller reads the `patientId` from the URL.
- **It reads the optional `?include list`, then calls the `build service`.**

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

// ---- expose the summary: route + controller ----
//      backend/routes/patientRoutes.js
router.get("/:patientId/summary", downloadPatientSummary);

//      backend/controllers/ipsController.js
async function downloadPatientSummary(req, res, next) {
  try {
    const { patientId } = req.params;
    const { include } = req.query; // "Observation,Procedure"
    const bundle = await buildPatientSummary(patientId, include);
    res.set("Content-Type", "application/fhir+json");
    res.json(bundle); // browser downloads this
  } catch (err) {
    next(err);
  }
}
// GET /api/patients/<id>/summary -> the IPS Bundle
```

The missing link — expose the endpoint

IN PLAIN ENGLISH

- One route maps GET `/api/patients/:patientId/summary` to the controller.
- The controller reads the `patientId` from the URL.
- It reads the optional `?include list`, then calls the build service.
- **It sends the finished Bundle back as `application/fhir+json`.**

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

// ---- expose the summary: route + controller ----
//      backend/routes/patientRoutes.js
router.get("/:patientId/summary", downloadPatientSummary);

//      backend/controllers/ipsController.js
async function downloadPatientSummary(req, res, next) {
  try {
    const { patientId } = req.params;
    const { include } = req.query; // "Observation,Procedure"
    const bundle = await buildPatientSummary(patientId, include);
    res.set("Content-Type", "application/fhir+json");
    res.json(bundle); // browser downloads this
  } catch (err) {
    next(err);
  }
}
// GET /api/patients/<id>/summary -> the IPS Bundle
```

What you learned

The whole pipeline, in six moves

- A Download button requests the summary endpoint and saves the response as a file.
- The server fetches the patient and the mandatory IPS collections.
- It adds any optional sections the user ticked, plus the Practitioners named on the medication requests.
- It groups resources into LOINC-coded sections under a Composition.
- It wraps everything in a FHIR Bundle of type “document”.
- A small route and controller expose it at GET /api/patients/:patientId/summary.

You can now turn a patient's records into a standard IPS document Bundle.