

A HANDS-ON TUTORIAL

Anonymizing FHIR Data

Strip identifiers from uploaded FHIR files and download the de-identified result

This tutorial covers the file anonymizer — it de-identifies whole uploaded files using an external tool, separate from the import and export flows.

Panagiotis Liakakos

June 2026

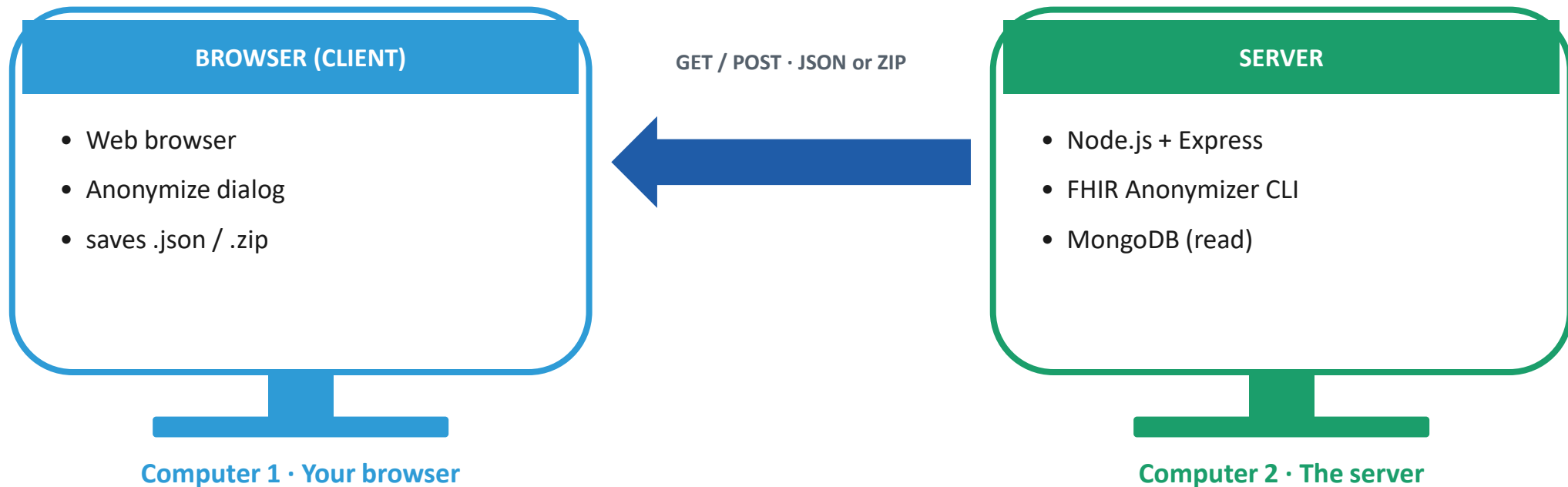
What you'll learn

By the end, you will be able to:

- Load an uploaded FHIR file's stored JSON from MongoDB.
- Stage it in a private workspace for an external tool.
- Run a config-driven FHIR anonymizer to strip identifiers.
- Download the de-identified result — one file, or several as a zip.

Goal: turn uploaded FHIR files into de-identified downloads.

How the data travels: server → browser



Uploaded files live in MongoDB. The server writes a copy to a temp workspace, runs Microsoft's FHIR anonymizer CLI over it, reads back the de-identified JSON, and returns it — as one file or a zip — for the browser to save.

What you need to anonymize FHIR data



You only need three tools

- Mongoose .findById() — load the uploaded file's stored JSON.
- Microsoft's FHIR Tools for Anonymization (an external CLI) + a customized config.json — the actual de-identification.
- Node's child_process + fs to run it, and the browser Blob (a zip via archiver for several) to save.

Prerequisites: Node 18+, MongoDB with uploaded files, and the anonymizer CLI installed on the server (github.com/microsoft/Tools-for-Health-Data-Anonymization).

The orchestration is ~50 lines in anonymizerService.js, plus two routes and a small frontend dialog.

What you need to anonymize FHIR data



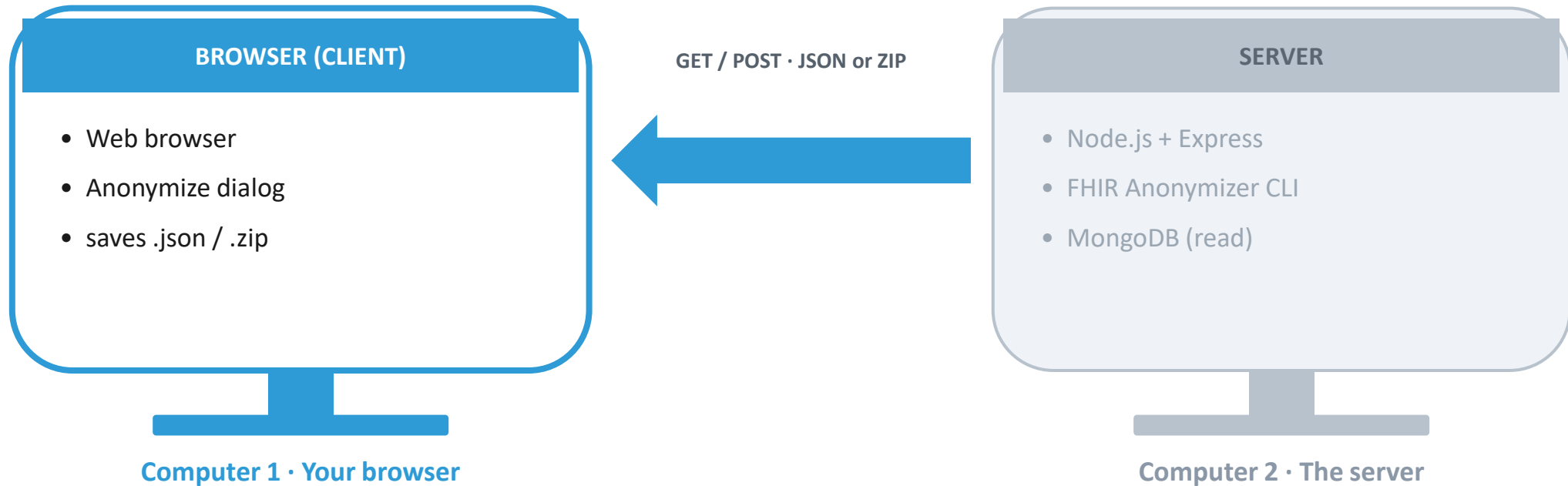
Five things not to forget

- Anonymize the stored rawJson — the uploaded FHIR JSON, as parsed and stored.
- The de-id rules live in config.json — a customized copy of the tool's sample config, not our code.
- Give each job its own temp input/output dirs so concurrent jobs don't collide.
- Always clean up the temp dirs afterwards.
- One file downloads as JSON; several stream together as a zip.

Prerequisites: Node 18+, MongoDB with uploaded files, and the anonymizer CLI installed on the server (github.com/microsoft/Tools-for-Health-Data-Anonymization).

The orchestration is ~50 lines in anonymizerService.js, plus two routes and a small frontend dialog.

First: the browser side (Computer 1)



The first step runs on Computer 1 — the user's web browser. It picks the files to anonymize, requests the server, and saves the de-identified result.

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

IN PLAIN ENGLISH

- The dialog lists the patient's uploaded files; the user ticks the ones to anonymize.

frontend · AnonymizeDialog.jsx

```
// frontend – anonymize the selected file(s), save the result
async function handleDownload(selected) {
  let res, filename;
  if (selected.size === 1) { // one file -> JSON
    const id = [...selected][0];
    res = await fetch(`${API}/patients/anonymize/${id}`);
    filename = "file-anonymized.json";
  } else { // several -> a zip
    res = await fetch(`${API}/patients/anonymize`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ fileIds: [...selected] }),
    });
    filename = "anonymized.zip";
  }

  const blob = await res.blob(); // JSON or zip
  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

- The dialog lists the patient's uploaded files; the user ticks the ones to anonymize.
- **One file requests the single-file endpoint; several POST a list of file ids.**

frontend · AnonymizeDialog.jsx

```
// frontend – anonymize the selected file(s), save the result
async function handleDownload(selected) {
  let res, filename;
  if (selected.size === 1) {          // one file -> JSON
    const id = [...selected][0];
    res = await fetch(`${API}/patients/anonymize/${id}`);
    filename = "file-anonymized.json";
  } else {                            // several -> a zip
    res = await fetch(`${API}/patients/anonymize`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ fileIds: [...selected] }),
    });
    filename = "anonymized.zip";
  }

  const blob = await res.blob();      // JSON or zip
  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

- The dialog lists the patient's uploaded files; the user ticks the ones to anonymize.
- One file requests the single-file endpoint; several POST a list of file ids.
- **The response is read as a Blob — a JSON file, or a zip for several.**

frontend · AnonymizeDialog.jsx

```
// frontend — anonymize the selected file(s), save the result
async function handleDownload(selected) {
  let res, filename;
  if (selected.size === 1) { // one file -> JSON
    const id = [...selected][0];
    res = await fetch(`${API}/patients/anonymize/${id}`);
    filename = "file-anonymized.json";
  } else { // several -> a zip
    res = await fetch(`${API}/patients/anonymize`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ fileIds: [...selected] }),
    });
    filename = "anonymized.zip";
  }

  const blob = await res.blob(); // JSON or zip
  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

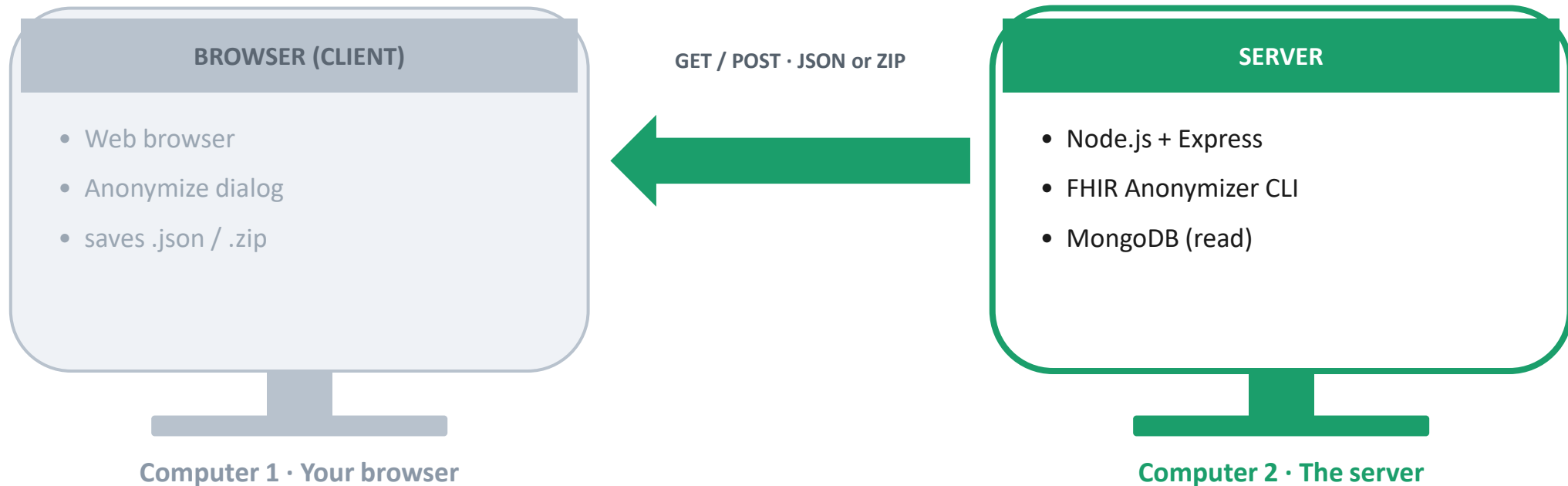
- The dialog lists the patient's uploaded files; the user ticks the ones to anonymize.
- One file requests the single-file endpoint; several POST a list of file ids.
- The response is read as a Blob — a JSON file, or a zip for several.
- **A temporary link triggers the browser's Save dialog, then is cleaned up.**

frontend · AnonymizeDialog.jsx

```
// frontend — anonymize the selected file(s), save the result
async function handleDownload(selected) {
  let res, filename;
  if (selected.size === 1) { // one file -> JSON
    const id = [...selected][0];
    res = await fetch(`${API}/patients/anonymize/${id}`);
    filename = "file-anonymized.json";
  } else { // several -> a zip
    res = await fetch(`${API}/patients/anonymize`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ fileIds: [...selected] }),
    });
    filename = "anonymized.zip";
  }

  const blob = await res.blob(); // JSON or zip
  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 loads the file, runs the external anonymizer over it, and returns the de-identified result.

Step 2 — Load the file and stage it (server)

IN PLAIN ENGLISH

- Look up the uploaded file by its id; if it's missing, stop with an error.

backend/services/anonymizerService.js

```
// backend — load the file and stage it for the tool
async function anonymizePatientFile(fileId) {
  const fileDoc = await patientFileService.getFileById(fileId);
  if (!fileDoc) throw new Error("File not found");

  // a private input/output workspace per job
  const jobId = Date.now().toString(36);
  const inDir = path.join(INPUT_ROOT, jobId);
  const outDir = path.join(OUTPUT_ROOT, jobId);
  fs.mkdirSync(inDir, { recursive: true });
  fs.mkdirSync(outDir, { recursive: true });

  // write the uploaded JSON where the tool will read it
  const inFile = path.join(inDir, "src.json");
  fs.writeFileSync(inFile, JSON.stringify(fileDoc.rawJson));
  // ...
}
```

Step 2 — Load the file and stage it (server)

IN PLAIN ENGLISH

- Look up the uploaded file by its id; if it's missing, stop with an error.
- **Name a private per-job input and output workspace (the CLI works on directories).**

backend/services/anonymizerService.js

```
// backend — load the file and stage it for the tool
async function anonymizePatientFile(fileId) {
  const fileDoc = await patientFileService.getFileById(fileId);
  if (!fileDoc) throw new Error("File not found");

  // a private input/output workspace per job
  const jobId = Date.now().toString(36);
  const inDir = path.join(INPUT_ROOT, jobId);
  const outDir = path.join(OUTPUT_ROOT, jobId);
  fs.mkdirSync(inDir, { recursive: true });
  fs.mkdirSync(outDir, { recursive: true });

  // write the uploaded JSON where the tool will read it
  const inFile = path.join(inDir, "src.json");
  fs.writeFileSync(inFile, JSON.stringify(fileDoc.rawJson));
  // ...
}
```

Step 2 — Load the file and stage it (server)

IN PLAIN ENGLISH

- Look up the uploaded file by its id; if it's missing, stop with an error.
- Name a private per-job input and output workspace (the CLI works on directories).
- **Create those directories.**

backend/services/anonymizerService.js

```
// backend — load the file and stage it for the tool
async function anonymizePatientFile(fileId) {
  const fileDoc = await patientFileService.getFileById(fileId);
  if (!fileDoc) throw new Error("File not found");

  // a private input/output workspace per job
  const jobId = Date.now().toString(36);
  const inDir = path.join(INPUT_ROOT, jobId);
  const outDir = path.join(OUTPUT_ROOT, jobId);
  fs.mkdirSync(inDir, { recursive: true });
  fs.mkdirSync(outDir, { recursive: true });

  // write the uploaded JSON where the tool will read it
  const inFile = path.join(inDir, "src.json");
  fs.writeFileSync(inFile, JSON.stringify(fileDoc.rawJson));
  // ...
}
```

Step 2 — Load the file and stage it (server)

IN PLAIN ENGLISH

- Look up the uploaded file by its id; if it's missing, stop with an error.
- Name a private per-job input and output workspace (the CLI works on directories).
- Create those directories.
- **Write the file's stored JSON into the input folder for the tool to read.**

backend/services/anonymizerService.js

```
// backend — load the file and stage it for the tool
async function anonymizePatientFile(fileId) {
  const fileDoc = await patientFileService.getFileById(fileId);
  if (!fileDoc) throw new Error("File not found");

  // a private input/output workspace per job
  const jobId = Date.now().toString(36);
  const inDir = path.join(INPUT_ROOT, jobId);
  const outDir = path.join(OUTPUT_ROOT, jobId);
  fs.mkdirSync(inDir, { recursive: true });
  fs.mkdirSync(outDir, { recursive: true });

  // write the uploaded JSON where the tool will read it
  const inFile = path.join(inDir, "src.json");
  fs.writeFileSync(inFile, JSON.stringify(fileDoc.rawJson));
  // ...
}
```

Step 3 — Run the anonymizer CLI (server)

IN PLAIN ENGLISH

- Invoke Microsoft's FHIR anonymizer CLI with the input, output, and config paths.

backend/services/anonymizerService.js

```
// run the external anonymizer, then read the result
await new Promise((resolve, reject) =>
  execFile(BIN, ["-i", inDir, "-o", outDir, "-c", CFG],
    (err) => (err ? reject(err) : resolve())));

// the tool writes the de-identified file into outDir
const outFile = path.join(outDir, "src.json");
const json = JSON.parse(fs.readFileSync(outFile, "utf8"));

// clean up the workspace
fs.rmSync(inDir, { recursive: true, force: true });
fs.rmSync(outDir, { recursive: true, force: true });

// name it "<original>-anon.json" and return
const filename = (fileDoc.filename || "patient-file")
  .replace(/\.json$/i, "-anon.json");
return { filename, json };
```

Step 3 — Run the anonymizer CLI (server)

IN PLAIN ENGLISH

- Invoke Microsoft's FHIR anonymizer CLI with the input, output, and config paths.
- **Read the de-identified JSON back from the output folder.**

backend/services/anonymizerService.js

```
// run the external anonymizer, then read the result
await new Promise((resolve, reject) =>
  execFile(BIN, ["-i", inDir, "-o", outDir, "-c", CFG],
    (err) => (err ? reject(err) : resolve())));

// the tool writes the de-identified file into outDir
const outFile = path.join(outDir, "src.json");
const json = JSON.parse(fs.readFileSync(outFile, "utf8"));

// clean up the workspace
fs.rmSync(inDir, { recursive: true, force: true });
fs.rmSync(outDir, { recursive: true, force: true });

// name it "<original>-anon.json" and return
const filename = (fileDoc.filename || "patient-file")
  .replace(/\.json$/i, "-anon.json");
return { filename, json };
```

Step 3 — Run the anonymizer CLI (server)

IN PLAIN ENGLISH

- Invoke Microsoft's FHIR anonymizer CLI with the input, output, and config paths.
- Read the de-identified JSON back from the output folder.
- **Delete the per-job workspace, whatever happened.**

backend/services/anonymizerService.js

```
// run the external anonymizer, then read the result
await new Promise((resolve, reject) =>
  execFile(BIN, ["-i", inDir, "-o", outDir, "-c", CFG],
    (err) => (err ? reject(err) : resolve())));

// the tool writes the de-identified file into outDir
const outFile = path.join(outDir, "src.json");
const json = JSON.parse(fs.readFileSync(outFile, "utf8"));

// clean up the workspace
fs.rmSync(inDir, { recursive: true, force: true });
fs.rmSync(outDir, { recursive: true, force: true });

// name it "<original>-anon.json" and return
const filename = (fileDoc.filename || "patient-file")
  .replace(/\.json$/i, "-anon.json");
return { filename, json };
```

Step 3 — Run the anonymizer CLI (server)

IN PLAIN ENGLISH

- Invoke Microsoft's FHIR anonymizer CLI with the input, output, and config paths.
- Read the de-identified JSON back from the output folder.
- Delete the per-job workspace, whatever happened.
- **Derive a "-anon" filename and return the result.**

backend/services/anonymizerService.js

```
// run the external anonymizer, then read the result
await new Promise((resolve, reject) =>
  execFile(BIN, ["-i", inDir, "-o", outDir, "-c", CFG],
    (err) => (err ? reject(err) : resolve())));

// the tool writes the de-identified file into outDir
const outFile = path.join(outDir, "src.json");
const json = JSON.parse(fs.readFileSync(outFile, "utf8"));

// clean up the workspace
fs.rmSync(inDir, { recursive: true, force: true });
fs.rmSync(outDir, { recursive: true, force: true });

// name it "<original>-anon.json" and return
const filename = (fileDoc.filename || "patient-file")
  .replace(/\.json$/i, "-anon.json");
return { filename, json };
```

Step 4 — Serve one file or a zip (server)

IN PLAIN ENGLISH

- The single-file route anonymizes one file and sends it as a JSON attachment.

backend/controllers/anonymizerController.js

```
// controller – one file as JSON, or many as a zip
async function downloadAnonymizedFile(req, res) {
  const { fileId } = req.params;
  const { filename, json } = await anonymizePatientFile(fileId);
  res.setHeader("Content-Disposition",
    `attachment; filename="${filename}"`);
  res.type("application/json").send(JSON.stringify(json, null, 2));
}

async function anonymizeMultipleFiles(req, res) {
  const { fileIds, format = "zip" } = req.body;
  const files = await anonymizePatientFiles(fileIds);
  if (format === "json") return res.json(files); // inline

  res.setHeader("Content-Disposition",
    `attachment; filename="anonymized-${Date.now()}.zip"`);
  res.type("application/zip");
  buildZipStream(files).pipe(res); // archiver
}
```

Step 4 — Serve one file or a zip (server)

IN PLAIN ENGLISH

- The single-file route anonymizes one file and sends it as a JSON attachment.
- **The bulk route anonymizes a list of files in parallel.**

backend/controllers/anonymizerController.js

```
// controller – one file as JSON, or many as a zip
async function downloadAnonymizedFile(req, res) {
  const { fileId } = req.params;
  const { filename, json } = await anonymizePatientFile(fileId);
  res.setHeader("Content-Disposition",
    `attachment; filename="${filename}"`);
  res.type("application/json").send(JSON.stringify(json, null, 2));
}
```

```
async function anonymizeMultipleFiles(req, res) {
  const { fileIds, format = "zip" } = req.body;
  const files = await anonymizePatientFiles(fileIds);
  if (format === "json") return res.json(files); // inline

  res.setHeader("Content-Disposition",
    `attachment; filename="anonymized-${Date.now()}.zip"`);
  res.type("application/zip");
  buildZipStream(files).pipe(res); // archiver
}
```

Step 4 — Serve one file or a zip (server)

IN PLAIN ENGLISH

- The single-file route anonymizes one file and sends it as a JSON attachment.
- The bulk route anonymizes a list of files in parallel.
- **With format “json”, it returns them inline.**

backend/controllers/anonymizerController.js

```
// controller – one file as JSON, or many as a zip
async function downloadAnonymizedFile(req, res) {
  const { fileId } = req.params;
  const { filename, json } = await anonymizePatientFile(fileId);
  res.setHeader("Content-Disposition",
    `attachment; filename="${filename}"`);
  res.type("application/json").send(JSON.stringify(json, null, 2));
}

async function anonymizeMultipleFiles(req, res) {
  const { fileIds, format = "zip" } = req.body;
  const files = await anonymizePatientFiles(fileIds);
  if (format === "json") return res.json(files); // inline

  res.setHeader("Content-Disposition",
    `attachment; filename="anonymized-${Date.now()}.zip"`);
  res.type("application/zip");
  buildZipStream(files).pipe(res); // archiver
}
```

Step 4 — Serve one file or a zip (server)

IN PLAIN ENGLISH

- The single-file route anonymizes one file and sends it as a JSON attachment.
- The bulk route anonymizes a list of files in parallel.
- With format “json”, it returns them inline.
- **Otherwise it streams them back together as a zip.**

backend/controllers/anonymizerController.js

```
// controller – one file as JSON, or many as a zip
async function downloadAnonymizedFile(req, res) {
  const { fileId } = req.params;
  const { filename, json } = await anonymizePatientFile(fileId);
  res.setHeader("Content-Disposition",
    `attachment; filename="${filename}"`);
  res.type("application/json").send(JSON.stringify(json, null, 2));
}

async function anonymizeMultipleFiles(req, res) {
  const { fileIds, format = "zip" } = req.body;
  const files = await anonymizePatientFiles(fileIds);
  if (format === "json") return res.json(files); // inline

  res.setHeader("Content-Disposition",
    `attachment; filename="anonymized-${Date.now()}.zip"`);
  res.type("application/zip");
  buildZipStream(files).pipe(res); // archiver
}
```

The missing link — expose the endpoints

IN PLAIN ENGLISH

- One GET route anonymizes a single file by its id.

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

// ---- expose the anonymizer: routes + controllers ----
// backend/routes/patientRoutes.js
router.get("/anonymize/:fileId", downloadAnonymizedFile);
router.post("/anonymize", anonymizeMultipleFiles);

// backend/controllers/anonymizerController.js
async function downloadAnonymizedFile(req, res) {
  const { filename, json } =
    await anonymizePatientFile(req.params.fileId);
  res.type("application/json").send(JSON.stringify(json));
}
async function anonymizeMultipleFiles(req, res) {
  const { fileIds, format = "zip" } = req.body;
  const files = await anonymizePatientFiles(fileIds);
  // format "json" -> res.json(files); else a .zip
}
```

The missing link — expose the endpoints

IN PLAIN ENGLISH

- One GET route anonymizes a single file by its id.
- **One POST route anonymizes a list of files (fileIds in the body).**

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

// ---- expose the anonymizer: routes + controllers ----
// backend/routes/patientRoutes.js
router.get("/anonymize/:fileId", downloadAnonymizedFile);
router.post("/anonymize", anonymizeMultipleFiles);

// backend/controllers/anonymizerController.js
async function downloadAnonymizedFile(req, res) {
  const { filename, json } =
    await anonymizePatientFile(req.params.fileId);
  res.type("application/json").send(JSON.stringify(json));
}
async function anonymizeMultipleFiles(req, res) {
  const { fileIds, format = "zip" } = req.body;
  const files = await anonymizePatientFiles(fileIds);
  // format "json" -> res.json(files); else a .zip
}
```

The missing link — expose the endpoints

IN PLAIN ENGLISH

- One GET route anonymizes a single file by its id.
- One POST route anonymizes a list of files (fileIds in the body).
- **Each route calls its controller function.**

backend/routes/patientRoutes.js + anonymizerController.js

```
// ---- expose the anonymizer: routes + controllers ----  
// backend/routes/patientRoutes.js  
router.get("/anonymize/:fileId", downloadAnonymizedFile);  
router.post("/anonymize", anonymizeMultipleFiles);  
  
// backend/controllers/anonymizerController.js  
async function downloadAnonymizedFile(req, res) {  
  const { filename, json } =  
    await anonymizePatientFile(req.params.fileId);  
  res.type("application/json").send(JSON.stringify(json));  
}  
  
async function anonymizeMultipleFiles(req, res) {  
  const { fileIds, format = "zip" } = req.body;  
  const files = await anonymizePatientFiles(fileIds);  
  // format "json" -> res.json(files); else a .zip  
}
```

The missing link — expose the endpoints

IN PLAIN ENGLISH

- One GET route anonymizes a single file by its id.
- One POST route anonymizes a list of files (fileIds in the body).
- Each route calls its controller function.
- **The controller returns a JSON file, or a zip of several.**

backend/routes/patientRoutes.js + anonymizerController.js

```
// ---- expose the anonymizer: routes + controllers ----  
// backend/routes/patientRoutes.js  
router.get("/anonymize/:fileId", downloadAnonymizedFile);  
router.post("/anonymize", anonymizeMultipleFiles);  
  
// backend/controllers/anonymizerController.js  
async function downloadAnonymizedFile(req, res) {  
  const { filename, json } =  
    await anonymizePatientFile(req.params.fileId);  
  res.type("application/json").send(JSON.stringify(json));  
}  
async function anonymizeMultipleFiles(req, res) {  
  const { fileIds, format = "zip" } = req.body;  
  const files = await anonymizePatientFiles(fileIds);  
  // format "json" -> res.json(files); else a .zip  
}
```

What you learned



The whole pipeline, in six moves

- The dialog lists the patient's uploaded files; the user picks the ones to anonymize.
- One file hits the single-file route; several are POSTed as a list of ids.
- The server loads each file's stored JSON and stages it in a per-job workspace.
- It runs the external, config-driven anonymizer, then reads the result and cleans up.
- It returns one de-identified JSON file, or a zip of several.
- Two small routes and controllers expose it under `/api/patients/anonymize`.

You can now turn uploaded FHIR files into de-identified downloads.