Paper AD02

Overcoming Challenges in Collaborative Spreadsheet Editing with Shiny, SpreadJS and JSON-Patch

Kamil Foltyński, Cytel Inc., Cointrin, Switzerland
Sebastià Barceló Bauzà, Cytel Inc., Cointrin, Switzerland

ABSTRACT

As collaborative workflows become increasingly relevant in clinical programming, enabling real-time spreadsheet editing within Shiny applications presents unique technical challenges. Tools like SpreadJS provide a strong foundation but integrating them into R-based environments at scale can result in performance and concurrency issues - especially when handling large Excel files.

In developing a collaborative Shiny tool, we encountered two key obstacles: slow performance due to large JSON payloads and difficulty managing simultaneous user edits without data conflicts. Our initial solution - converting full Excel sheets to JSON and tracking changes via Git - introduced latency and synchronization risks.

To overcome this, we implemented the fast-json-patch library, which tracks only granular changes rather than full document states. This significantly reduced payload size, improved responsiveness, and enabled robust conflict resolution.

This paper outlines the technical approach, implementation strategies, and practical lessons learned. Attendees will gain insights into applying similar techniques for collaborative tools in pharmaceutical programming environments.

INTRODUCTION

Spreadsheet tools are central in clinical programming, but generic cloud editors fall short for regulated use cases that demand deep auditability, robust access control, and server-side metadata integration. ePPT was designed to meet these needs while enabling real-time collaboration. The following sections summarize the technical background, challenges, solutions, and implementation outcomes.

BACKGROUND & REQUIREMENTS

Why not just use a general cloud spreadsheet? Our requirements were specific to regulated programming workflows:

- End-to-end audit trail of every cell-level change linked to user identity and time.
- Ability to enrich spreadsheets with server-side programming metadata (e.g., program author, last modifier, last execution timestamp).
- Integrated dashboards: programming status, log/lst health, and time-based issue tracking.
- Cross-checks between planning sheets and issue logs (e.g., prevent “ready” when open issues remain).

TECHNICAL BACKGROUND

SHINY

Shiny provides a reactive web framework for R that enables rich, interactive applications. Its reactive graph simplifies UI updates but requires careful design for low-latency, multi-user collaboration.

SPREADJS

SpreadJS is a client-side spreadsheet component with Excel-compatible behavior, formatting, and formulas. It exposes granular cell-level events and a JSON model for workbook state - crucial for tracking edits and synchronizing clients.

HTMLWIDGETS

htmlwidgets R package is the bridge that integrates JavaScript libraries with R and Shiny. In ePPT, SpreadJS is wrapped as an htmlwidget (spreadJS), which bundles the JS and CSS assets, initializes the grid in the browser, and wires two-way communication with Shiny. User actions in SpreadJS are emitted to the server via `Shiny.setInputValue(...)`, while server-side renders use `renderSpreadJS(...)` and server-side renders use `spreadJS(...)` to push JSON payloads back to the client. This mechanism is how we capture cell-level edits and transport them as JSON Patch without building a custom web server.

JSON-PATCH (RFC 6902)

JSON-Patch represents changes as a list of operations (add, remove, replace, move, copy, test) applied to JSON documents via JSON Pointer paths. Using fast-json-patch on the client, we capture minimal deltas instead of full workbook payloads.

WHY THIS STACK

Shiny anchors R-centric workflows, SpreadJS delivers spreadsheet fidelity in the browser, and JSON-Patch provides efficient, conflict-aware synchronization - together enabling compliant collaboration without round-tripping full files.

CHALLENGES ENCOUNTERED

- Excel-grade UX in a Shiny app despite limits of R UI toolkits
- Integrating a browser spreadsheet with Shiny/R
- Implementing a robust audit trail of cell-level changes
- Avoiding megabyte-scale payloads and slow reloads
- Concurrency and conflict resolution on shared sheets

CHALLENGES AND SOLUTIONS

1. EXCEL-GRADE UX IN A SHINY APP

Problem: R UI toolkits don't feel like Excel.

Solution: Use spreadJS - a JavaScript spreadsheet library designed for enterprise web applications, enabling developers to build Excel-like interfaces directly in the browser (formatting, formulas, events).

Client snippet (SpreadJS → JSON-Patch and emit):

```
// Compare baselines and snapshot, enrich with user/timestamp, debounce events
 const jsonPatches = jsonpatch.compare(baseData, snapData).map((op, i) => ({
   ...op, user: currentUser, timestamp: eventTimestamp, sequence: baseSeq + i
 }));
 Shiny.setInputValue(elementId + "_jsonpatches", {
   SpreadEvent: "exportJsonPatches",
   jsonStr: JSON.stringify(jsonPatches),
   args: args, user: currentUser, timestamp: eventTimestamp
 }, { priority: "event" });
Event coalescing:
const handleSpreadEventDebounced = _.debounce(handleSpreadEvent, 50, { trailing: true, maxWait: 1500 });
 eventsList.forEach(evt => spread.bind(evt, handleSpreadEventDebounced));
```

2. INTEGRATING SPREADJS WITH SHINY

Problem: Connect the browser grid to R server code bidirectionally.

Solution: Wrap SpreadJS via htmlwidgets; emit events and JSON to Shiny with Shiny.setInputValue(...); render via spreadJS()/renderSpreadJS().

Client emit example:

```
Shiny.setInputValue(elementId + "_data", {
  SpreadEvent: "exportSpreadData",
  jsonStr: jsonString,
  args, user: currentUser,
  timestamp: eventTimestamp },
{ priority: "event" });
```

3. IMPLEMENTING AUDIT TRAIL

Problem: Implement audit trail - a chronological, time-stamped record that documents the sequence of user changes to spreadJS workbook

Solution: The complete spreadJS workbook can be converted to JSON format with method **`spread.toJSON()`** in JavaScript layer and sent to Shiny server with **`Shiny.setInputValue()`**. Then changes as JSON Lines appended to NDJSON and can be tracked with old, proven **`git commit.`**

Example NDJSON:

```
{"op":"add","path":"/sheets/Doc/data/dataTable/3","value":{"1":{"value":"change 1"}},"user":"kamil"}
{"op":"add","path":"/sheets/Doc/data/dataTable/4","value":{"1":{"value":"change 2"}},"user":"kamil"}
{"op":"replace","path":"/sheets/Doc/data/dataTable/4/1/value","value":"change 3","user":"sebastia"}
{"op":"replace","path":"/sheets/Doc/data/dataTable/4/1/value","value":"change 4","user":"kamil"}
```

4. AVOIDING MEGABYTE-SCALE PAYLOADS

Problem: Full spreadJS workbook converted to JSON (from 5 to >30MB) too heavy for multi-user updates.

Solution: Implement JSON-Patch (RFC 6902) standard that allows updating a JSON document by sending the changes rather than the whole document. Send minimal JSON-Patch deltas using `fast-json-patch` JS library that can compare 2 JSONs:

```
var documentA = {user: {firstName: "Albert", lastName: "Einstein"}};
var documentB = {user: {firstName: "Albert", lastName: "Collins"}};
var diff = jsonpatch.compare(documentA, documentB);
//diff == [{op: "replace", path: "/user/lastName", value: "Collins"}]
```

Client (SpreadJS → emit JSON Patch with metadata):

```
// Compute minimal JSON patches vs. baseline and enrich with user/timestamp/sequence
 const snapData  = spread.toJSON(serializationOptionConfigForJsonpatch(), null, 2);
 const bl        = spread.__epptBaselines || {};
 const baseData  = bl.data;
 let jsonPatches = jsonpatch.compare(baseData, snapData);
 const eventTimestamp  = getHighPrecisionTimestamp();
 const currentUser     = getCurrentUser();
 const baseSequenceInt = Math.floor(performance.now() * 1000);
 jsonPatches = jsonPatches.map((patch, index) => ({
   ...patch,
   user: currentUser,
   timestamp: eventTimestamp,
   sequence: baseSequenceInt + index
 }));

// Emit to Shiny as a single event payload
 Shiny.setInputValue(elementId + "_jsonpatches", {
   SpreadEvent: "exportJsonPatches",
   jsonStr: JSON.stringify(jsonPatches),
   args: args,
   user: currentUser,
   timestamp: eventTimestamp
 }, { priority: "event" });
```

Server (Shiny → append JSONL under a file lock and commit with retries):

```
observeEvent(input$spreadjs_jsonpatches, label = "Change from UI to changelog", priority = 90, {
  req(getOption("eppt.json_patch"))
  repo <- req(state$repo)
  changelog_path_val <- file.path(project_dir(), "changelog.jsonl")
```

```r
jp <- req(input$spreadjs_jsonpatches)
 if (nchar(jp$jsonStr) < 5) return()

# Asynchronous write guarded by a file lock (prevents concurrent writers)
 write_promise <- future({
  tryCatch({
    lock_file <- glue::glue("{changelog_path_val}.lock")
    lck <- filelock::lock(path = lock_file, timeout = 2L)
    on.exit(try(filelock::unlock(lck), silent = TRUE), add = TRUE)

   write_jsonpatches(jp$jsonStr, get_userid(), changelog_path_val)
    list(success = TRUE, path = changelog_path_val)
  }, error = function(e) list(success = FALSE, error = e$message))
 })
```

5. CONCURRENCY AND CONFLICT RESOLUTION

Problem: Concurrency and conflict resolution. Simultaneous changes by different users may result in overlapping edits or overwriting each other because diffs (patches) are appended to NDJSON and restored in sequential order.

Solution: Add a highly granular timestamp to each JSON Line in NDJSON and restore spreadJS workbook by reading diffs in chronological order.

Previous NDJSON looked like this:

```
{"op":"add","path":"/sheets/Doc/data/dataTable/3","value":{"1":{"value":"change 1"}},"user":"kamil"}
{"op":"add","path":"/sheets/Doc/data/dataTable/4","value":{"1":{"value":"change 2"}},"user":"kamil"}
{"op":"replace","path":"/sheets/Doc/data/dataTable/4/1/value","value":"change 3","user":"sebastia"}
{"op":"replace","path":"/sheets/Doc/data/dataTable/4/1/value","value":"change 4","user":"kamil"}
```

So after restoring SpreadJS JSON cell [4,1] will have value `change 4`, which could be wrong because user make change to spreadJS workbook in parallel and there is small - but always - delay between sending JSON patches from JS to Shiny backend, so we came with idea to timestamp each JSON patch and restore SpreadJS JSON by timestamps:

```
{"op":"add","path":"/sheets/Doc/data/dataTable/3","value":{"1":{"value":"change
1"}},"user":"kamil","timestamp":"2025-10-08T19:17:11.939700Z","sequence":19560701}
{"op":"add","path":"/sheets/Doc/data/dataTable/4","value":{"1":{"value":"change
2"}},"user":"kamil","timestamp":"2025-10-08T19:17:21.976400Z","sequence":29598501}
{"op":"replace","path":"/sheets/Doc/data/dataTable/4/1/value","value":"change
3","user":"sebastia","timestamp":"2025-10-08T19:17:58.319800Z","sequence":65940801}
{"op":"replace","path":"/sheets/Doc/data/dataTable/4/1/value","value":"change
4","user":"kamil","timestamp":"2025-10-08T19:17:50.204800Z","sequence":57825801}
```

So in this case cell [4,1] will have value `change 3` because was made later at `2025-10-08T19:17:58.319800Z` than `2025-10-08T19:17:50.204800Z`

CONCLUSION

Combining Shiny, SpreadJS, and JSON Patch enabled compliant, efficient collaborative spreadsheet editing tailored to clinical programming needs; sending minimal deltas with a rigorous audit trail and server-side validation improved responsiveness, made conflict resolution manageable, and kept audit, permissions, and metadata within our environment, balancing usability with regulatory requirements.

REFERENCES

- IETF RFC 6902: JavaScript Object Notation (JSON) Patch - https://www.rfc-editor.org/rfc/rfc6902
- fast-json-patch (JavaScript/TypeScript implementation of JSON Patch) - https://www.npmjs.com/package/fast-json-patch
- SpreadJS (GrapeCity) - browser-native spreadsheet component - https://www.grapecity.com/spreadjs
- Shiny for R (Posit) - https://shiny.posit.co/
- htmlwidgets for R - https://www.htmlwidgets.org/

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author(s) at:

Kamil Foltyński
Cytel Inc.
Route de Pré-Bois 20,
1216 Cointrin - Switzerland
Email: kamil.foltynski@cytel.com
Web: www.cytel.com

Sebastià Barceló Bauzà
Cytel Inc.
Route de Pré-Bois 20,
1216 Cointrin - Switzerland
Email: sebastia.barcelobauza@cytel.com
Web: www.cytel.com