

TREESTORAGE: HIERARCHICAL STORAGE WITH VERSIONING AND INHERITED ACCESS CONTROL IN A UNIVERSITY INFORMATION SYSTEM

Petr Straka¹, Oldrich Trenz¹

¹Department of Informatics, Faculty of Business and Economics, Mendel University in Brno, Zemědělská 1, 613 00 Brno, Czech Republic

DOI: https://doi.org/10.11118/ecos2026_036

ABSTRACT

Large information systems accumulate knowledge that is often transferred informally rather than stored in the system itself. This article describes the design and working prototype of Treestorage, a reusable hierarchical storage component for INET, the economic and personnel information system of Masaryk University. Treestorage is intended to organise agenda-related HTML notes, references to binary documents and parameterised SQL reports in a tree structure, with read and edit rights inherited from parent nodes and version history stored for edited nodes. Before implementing the component, available alternatives at the university were assessed, especially Google Drive, OneDrive and git-based services such as GitHub/GitLab; these options were not selected because they would either keep data and authentication outside INET or fail key requirements such as fine-grained folder-level read access. The implemented data model uses an adjacency list, reads nodes as a flat list and rebuilds the tree in the front end, while inherited rights are resolved through a PostgreSQL materialised view based on a recursive query. The result is a functional prototype that can be tested internally and further integrated after related INET 2.0 components are completed.

Keywords: information systems, hierarchical storage, access control, versioning, materialized view, adjacency list

JEL Code: C88

1 INTRODUCTION

INET, the information system of Masaryk University, has grown since 1998 and was opened to users in 2000 (Měcháček and Kohoutková, 2000). Today it runs hundreds of applications across economics, personnel and study agendas. With that scale comes a problem that has little to do with code. The knowledge of how a particular specialised application is meant to be used, together with the rarely-run SQL reports that answer recurring administrative questions, is held by individual



This article is licensed under CC BY-SA 4.0 Attribution-ShareAlike 4.0 International license. <https://creativecommons.org/licenses/by-sa/4.0/deed.en>

people. When one of them leaves or changes role, the knowledge often goes too. The system offers no shared, access-controlled place to keep notes, document links and reusable reports next to the agenda they belong to.

That is the gap Treestorage was built to close. The thesis set out to design and implement a hierarchical storage component that any INET application can embed, with read and edit rights set on a parent node cascading to everything below it and an earlier state of any node recoverable from its history. One constraint shaped everything else: the component had to live inside INET rather than beside it, so that this supporting knowledge stays under the same authentication and the same data umbrella as the agenda it serves.

1.1 Why not an existing product

The first question is whether a ready-made tool would do. The thesis examined storage options available in the university environment, especially document-management services such as Google Drive and OneDrive and git-based services such as GitHub/GitLab. For the most relevant API-based options, small proof-of-concept clients were prepared to test basic operations such as listing, creating, moving and deleting nodes and setting permissions. Google Drive and OneDrive provide storage and inheriting permissions, but integration would depend on external service accounts, OAuth/Microsoft Entra mechanisms and data stored outside INET. Confluence and Outline were considered only as contextual knowledge-base tools and were not treated as direct replacements for report-aware hierarchical storage.

The git family, represented mainly by GitHub/GitLab, looked promising because versioning is its native strength, yet it fits this task only partially. Git-based services cannot naturally grant read permission on a single folder; access is typically handled at the repository level. They also do not track empty folders, which complicates the preparation of an empty tree structure, and binary documents are not handled as efficiently as textual files. These limitations, together with the need to keep the component inside INET and under its existing authentication and agenda context, were the main reasons for building a custom component rather than adopting an external product directly.

1.2 Storing a tree in a relational database

A hierarchy is a rooted tree, and a relational database can keep one in several ways. The thesis weighed four classical patterns (adjacency list, path enumeration, the closure table and the nested set model) against the tree idioms of a graph database such as Neo4j with its Cypher language and of a document database such as MongoDB (Samoilenko, 2025; Neo4j n.d.; MongoDB n.d.). The MongoDB single-document tree caps at a 16 MB document, which rules it out for an open-ended store.

An adjacency list was chosen: every node simply stores the identifier of its parent. It is the simplest model to write against and the cheapest to keep correct. It also sidesteps the usual objection, that reading a subtree needs recursion in SQL, by not doing the recursion in SQL at all. The whole space is read as one flat list and the tree is reassembled in the front end, a linear-time operation on the client. For the expected size of a Treestorage space this is more than enough, as the next sections show.

2 METHODOLOGY

The work followed an engineering path, not an experimental one: requirements analysis, a survey of existing solutions, database design, and a prototype implementation, closing with an outlook chapter. There is no quantitative benchmark, because the deliverable is a working component inside an existing framework rather than a measured model.

The component sits inside INET 2.0 and is implemented in Java 25, while data are stored in a PostgreSQL relational database. The user interface uses INET's internal web-UI framework, whose components are rendered to HTML through XSL transforms. The dynamic tree is handled on the client side in JavaScript/React and supports node movement through the native HTML drag-and-drop API. Persistence is implemented through JPA entities, which reduces the need to write SQL manually for standard data operations.

Treestorage tables carry the prefix **wdoc_**, the cross-agenda report tables **a_**. A **wdoc_space** row splits storage into independent spaces, one tree per major INET agenda such as Economics, each tied to a domain. The central **wdoc_node** table holds the space, the parent (the nearest ancestor), the node type, a bilingual name in Czech and English, four generic content columns (**str_val_1**, **str_val_2**, **int_val_1**, **int_val_2**) and audit columns. Every change copies the data columns into a new row of **wdoc_node_log**, and that is how version history is kept. Access rights live in **wdoc_node_acl**, one row per node and person with two booleans, **can_read** and **can_edit**. Reports are defined in **a_query** with their parameters in **a_query_param**. Figure 1 shows the full model.

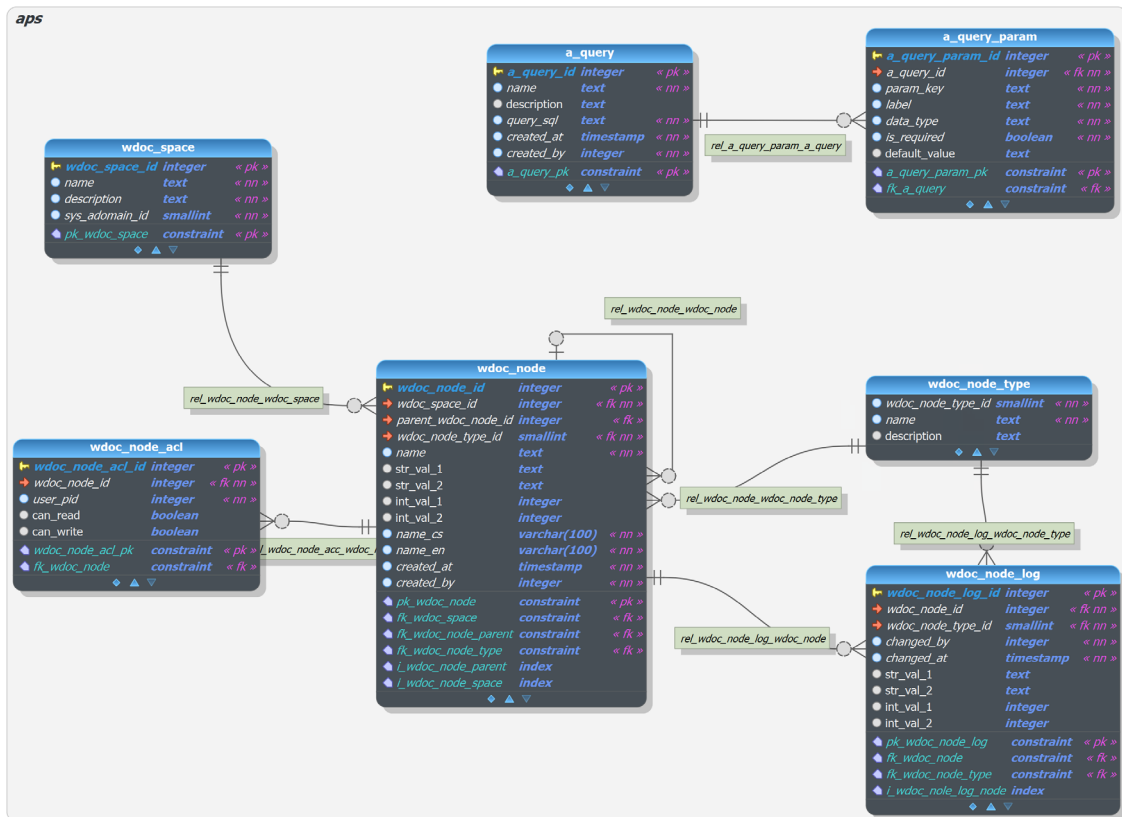


Fig. 1: Entity-relationship diagram of the Treestorage data model, showing the core node table with its versioning log, node types, access-control list and dynamic-query tables (adopted from Straka 2026)

A node is one of four types in this first version: DIR (a folder), HTML (text or rich-text content intended to be edited through the INET WYSIWYG component), DOCUMENT (a reference to binary data expected to be handled through DMSX) and QUERY (a reference to a dynamic SQL report). The four generic content columns allow a single table to carry different node types and make later extension possible without changing the basic structure. The use-case model defines three roles. A reader views permitted nodes, browses history and runs reports. An editor edits and adds nodes and manages rights on relevant parts of the tree. A superuser creates level-zero nodes and maintains the top-level structure.

Inherited rights are the part of the design worth dwelling on, because they are resolved without paying for recursion on every request. Rights cascade from parent to child, but recomputing that for each page load would be wasteful. Instead the effective rights are precomputed in a PostgreSQL materialised view, `wdoc_node_acl_effective`, built from a recursive query: it starts at the root nodes paired with every person who holds at least one explicit right anywhere, then walks down the tree, inheriting the parent's rights unless an explicit ACL row overrides them. A unique index on the person-and-node pair keeps lookups fast, and the view is refreshed only when rights actually change (a database trigger on the ACL table is noted as a likely later addition). Reports get one more safeguard: when a report is saved, an SQL checker parses the body and blocks the save unless every declared variable is also defined in the parameter table, so a report cannot be stored in a state that would fail at run time.

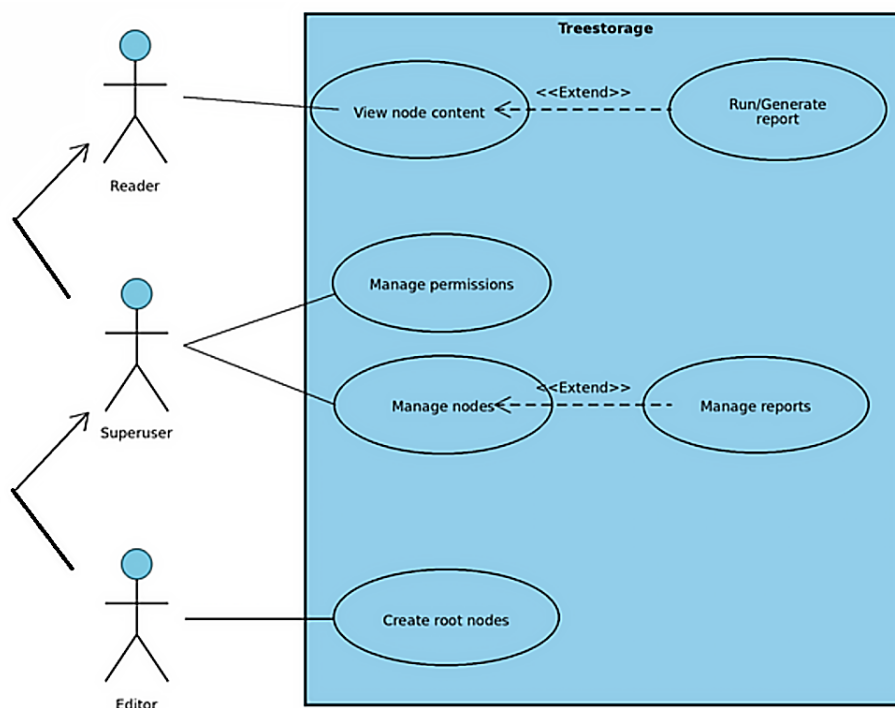
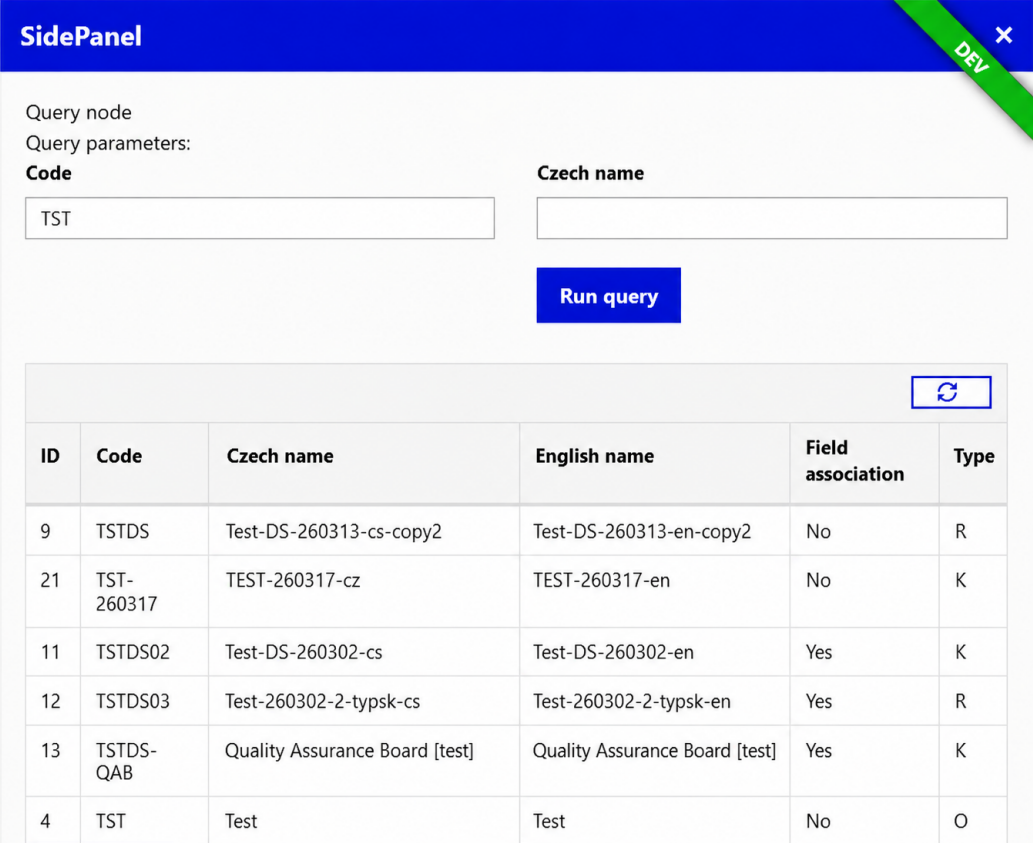


Fig. 2: UML use case diagram of the Treestorage system with the Reader, Superuser and Editor roles and their operations over hierarchical nodes (adopted from Straka 2026)

3 RESULTS

The outcome is a functional prototype, not a set of measurements. Treestorage works as a reusable INET component: it embeds a hierarchical tree into any INET application and, for each node, shows a rich-text note, a document link, or a dynamic SQL report tied to that agenda. Version history is in place (earlier states sit in `wdoc_node_log` and can be reopened), and the inheriting read/edit access control runs through the materialised view rather than per-request recursion. The first version ships four node types, three user roles, three report-parameter input types (text, numeric and a person-id select), and two new report tables (Straka, 2026). Figure 3 shows the dynamic-report component with a generated result table.

A second concrete result is the survey that justified building the component at all. Table 1 should be read as an authorial synthesis of selected figures and constraints reported in chapter 5 of the thesis, not as a direct vendor table. The comparison is primarily functional: whether a service can be integrated into INET, whether it supports the required access-control model, and whether it can hold the intended types of content. Storage capacities and commercial prices are not fully comparable across the alternatives because they use different licensing models and institutional arrangements.



The screenshot shows a web interface titled "SidePanel" with a blue header and a green "DEV" badge. Below the header, there is a "Query node" section with "Query parameters:" and two input fields: "Code" (containing "TST") and "Czech name" (empty). A blue "Run query" button is positioned below these fields. To the right of the "Run query" button is a refresh icon. Below the form is a table with 6 columns: ID, Code, Czech name, English name, Field association, and Type. The table contains 7 rows of data.

ID	Code	Czech name	English name	Field association	Type
9	TSTDS	Test-DS-260313-cs-copy2	Test-DS-260313-en-copy2	No	R
21	TST-260317	TEST-260317-cz	TEST-260317-en	No	K
11	TSTDS02	Test-DS-260302-cs	Test-DS-260302-en	Yes	K
12	TSTDS03	Test-260302-2-typsk-cs	Test-260302-2-typsk-en	Yes	R
13	TSTDS-QAB	Quality Assurance Board [test]	Quality Assurance Board [test]	Yes	K
4	TST	Test	Test	No	O

Fig. 3: Web user interface of the dynamic-report component, listing queried nodes with their bilingual names, type and versioning metadata (adopted from Straka, 2026).

Tab. 1 Off-the-shelf storage options surveyed before building Treestorage (authorial synthesis of figures cited in chapter 5; “—” = not stated). Commercial list prices are per user and are not the rates the university actually pays

Criterion	Google Drive	OneDrive (MS 365)	GitHub / git
Education storage	100 TB shared / org (Fundamentals, free)	100 GB staff, 30 GB students	repo limit 10 GB
API / integration	Drive API v3	Microsoft Graph (+ MS Entra)	GitHub REST (git trees)
API rate limit	20 000 req / 100 s	3 000 req / 5 min (+ 100 GB down/h, 50 GB up/h)	metered
File-size limit	—	—	warning > 1 MB, cap 100 MB; LFS 10–250 GiB
List price (per user)	14 USD / 2 TB tier	12.5 USD (M365 Copilot Standard)	4 USD / user
Per-folder read control	yes (inheriting), Google accounts only	yes, also outside MS account	no (whole repo only)
Move a node	yes	yes	only by delete + recreate
Empty folder	yes	yes	no
Binary + versioning	yes	yes	inefficient (full file per version)
Indicative cost for all of MU	~3.5 mil. CZK/yr (33 000 students × 5 USD edu)	~5.2 mil. CZK/yr (2024 university licence)	hard to cost

The status is honest about its limits. The component is a functional prototype, testable internally at the university computing centre, and it waits on two INET 2.0 pieces, the WYSIWYG editor and the DMSX binary-upload component, before it can go to real users. It has not yet been measured under load; the assumptions behind that are spelled out in the discussion.

4 DISCUSSION

Why an adjacency list, when the textbook objection (no cheap subtree query in SQL) is so well known? In Treestorage, the objection is weakened by the intended use case. The component reads the nodes of a space as a flat list and rebuilds the tree in the browser, so the server does not repeatedly assemble subtrees through recursive SQL. The expected size of a space is relatively small, and

the design therefore favours simplicity, easy updates and maintainability. More elaborate alternatives such as a closure table or the nested set model would improve some query patterns, but at the cost of more complex write operations and maintenance.

Access control makes the opposite trade. Inheritance is genuinely recursive, so a materialised view backed by a recursive query precomputes it once and refreshes only when rights change, instead of recomputing it on every request. This is the right call when reads vastly outnumber rights changes, which is the normal pattern for stored knowledge: people read notes and run reports far more often than they regrant permissions. The cost is staleness between refreshes, handled for now by refreshing from inside the application. The moment rights start changing often, a database trigger on the ACL table becomes the cleaner answer, and the design already flags that path.

The survey result is worth dwelling on because it is tempting to assume that a hosted service is always the pragmatic choice. For an internal, access-controlled and report-aware store, this is not straightforward. Google Drive and OneDrive solve document storage but do not naturally solve the INET-specific problem of keeping notes, document references and reusable reports directly inside the agenda context. Git solves versioning well, but not fine-grained folder-level read rights or efficient binary-document handling. Treestorage's value is therefore not only storage capacity, but the integration of knowledge, documents and reports into the same controlled INET environment.

5 CONCLUSION

Treestorage delivers a working prototype of a component that a large information system lacked: a hierarchical place where supporting knowledge, document references and reusable reports can be kept under the same authentication and rights model as the agenda they explain. The design uses an adjacency-list tree, browser-side reconstruction, node-level versioning and inherited rights materialised in PostgreSQL. Existing university tools were useful reference points but not direct substitutes because they either externalise the data or do not provide the required integration and permission model. The next step is to complete the dependent INET 2.0 components, verify performance under realistic load and pilot the tool in one agenda before wider use.

Acknowledgements

This paper was supported by the project CZ.02.1.01/0.0/0.0/16_017/0002334 Research Infrastructure for Young Scientists, which is co-financed from the Operational Programme Research, Development and Education.

Anthropic Claude was used to summarize the text of the final thesis (Straka, 2026) and transform it into the form of this compact paper. The paper was then reviewed and fine-tuned by the author. The author is responsible for the correctness of the final text.

REFERENCES

- ATLASSIAN. n.d. Confluence — Atlassian. *Atlassian*. <https://www.atlassian.com/software/confluence>
- GITHUB. n.d. GitHub Acceptable Use Policies. *GitHub*. <https://docs.github.com/en/site-policy/acceptable-use-policies/github-acceptable-use-policies>
- GOOGLE. 2026. Google Drive API release notes. *Google for Developers*. <https://developers.google.com/workspace/drive/release-notes>
- MĚCHÁČEK, J., KOHOUTKOVÁ, J. 2000. Intranetový server Informačního systému MU [in Czech]. *Zpravodaj ÚVT MU*. XI(1), 4–7. <https://webserver.ics.muni.cz/zpravodaj/articles/194.html>
- MICROSOFT. 2025. Avoid getting throttled or blocked in SharePoint Online. *Microsoft Learn*. <https://learn.microsoft.com/en-us/sharepoint/dev/general-development/how-to-avoid-getting-throttled-or-blocked-in-sharepoint-online>
- MONGODB. n.d. Model Tree Structures. *MongoDB*. <https://www.mongodb.com/docs/manual/applications/data-models-tree-structures/>
- MOZILLA. n.d. JavaScript. *MDN Web Docs*. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- NEO4J. n.d. *Neo4j documentation*. <https://neo4j.com/docs/>
- ORACLE. n.d. What is Java technology and why do I need it? *java.com*. https://www.java.com/en/download/help/whatis_java.html
- OUTLINE. n.d. *Outline*. <https://www.getoutline.com>
- POSTGRESQL. n.d. PostgreSQL Contributor Profiles. *The PostgreSQL Global Development Group*. <https://www.postgresql.org/community/contributors/>
- RICHARDS, R. 2025. 2025 Google Workspace for Education Pricing and Licensing Updates. *Trafera*. <https://www.trafera.com/blog/2025-google-workspace-for-education-pricing-and-licensing-updates/>
- SAMOILENKO, S. 2025. Part 1: Nested sets. *Medium*. <https://medium.com/@serhiisamoilenko/part-1-nested-sets-239e70f59beb>
- SEYBOLT, B. 2025. Major Google for Education Licensing and Price Changes. *Cumulus Global*. <https://www.cumulusglobal.com/major-google-for-education-licensing-and-price-changes/>
- STRAKA, P. 2026. *Treestorage — hierarchical storage with versioning and access control* [in Czech]. Master's thesis, head: Trenz, O. Brno: Mendel University in Brno.

Contact information

Petr Straka: e-mail: xstraka6@node.mendelu.cz

Oldrich Trenz: e-mail: oldrich.trenz@mendelu.cz