

Lillian

A beautiful, cross-platform file manager.



Recap of 1st iteration

Initial vision, requirements and plans for 2nd iteration

Vision

- Clean, intuitive user interface
 - Drag and drop files between multiple windows
 - Keyboard shortcuts
- Support for multiple local and remote file systems
 - Files stored on disks accessible via system interface
 - SSH file systems
 - Google Drive, One Drive and other cloud storage systems
- Stability and speed
- Client for multiple platforms

A look back at 1st iteration

- Goals
 - Create a proof of concept that can serve as foundation for further iterations
 - Design and implement a user interface that is intuitive and can be used for multiple platforms
- Requirements
 - Displays contents of a folder visible via native OS interface
 - Users can navigate between folders
 - Works out-of-the-box on Ubuntu and Windows
 - Always quick and responsive

All requirements were met.

Initial plans for 2nd iteration

- Initial plans included:
 - Implement file opening, moving and copying
 - Allow for drag-and-drop operations between application windows
 - Integrate application with at least one remote drive
- Initial plans were scaled down, to accommodate:
 - Implementation of a fast (on-the-fly streaming) transfer from / to a remote drive with specific API
 - Fixing bugs from 1st iteration



2nd iteration

Plans, challenges, solutions and final result

Final plans for 2nd iteration

- Functionality
 - Ability to navigate through files in a remote location (specific API and responsiveness as a requirement)
 - Add aliases for most important locations (local and remote)
- User Interface
 - Display loading and error information on status bar
 - Add settings window for managing locations and aliases
 - Minor changes in shapes and sizes of UI components
- Bug fixes

Adding external location

- Challenge: loading large folders
 - Request cannot be blocking
 - Users should see files as soon as possible
- Solution
 - Read HTTP response body chunk-by-chunk
 - Parse raw response text to JSON objects on the fly
 - Stream parsed JSON objects from main thread to UI thread in batches of 10 files / folders
- Useful tools
 - OBOE – library for parsing JSON on the fly

Parsing JSON stream with OBOE

```
const arr = []; // array for caching folders (parsed to valid JSON objects)
oboe({
  method: 'POST',
  url: locData.url + `?l=${locData.login}&p=${locData.pass}&q=${locData.path}`,
  agent: false,
  json: locData
}).node('!.*', (data) => {
  arr.push(data);

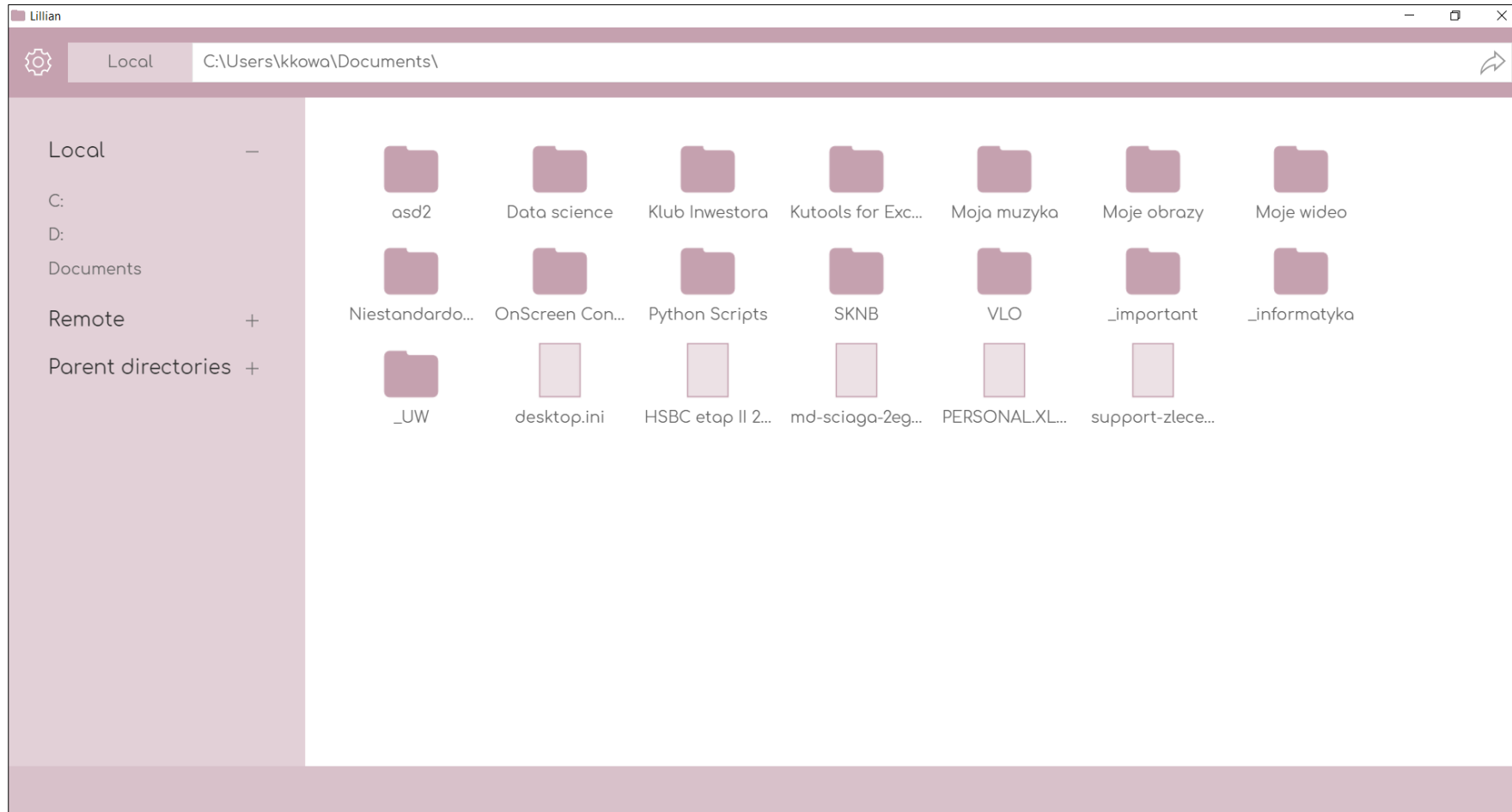
  if (arr.length === 10) { // send batches of 10 files/folders to UI thread
    const parsedObjects = parseRemoteJsonChunk(arr, rMsg);

    if (parsedObjects === null) {
      sendError(event);
    } else {
      addExtraAndSend(current_session_id, event, parseRemoteJsonChunk(arr, rMsg));
    }
  }
}).fail((error) => {
  sendError(event);
}).done(() => {
  if (arr.length !== 0) {
    addExtraAndSend(current_session_id, event, parseRemoteJsonChunk(arr, rMsg));
  }
});
```

Designing communication

- Challenge: design maintainable IPC for new features
 - IPC message schema has to be understood by developers working on both UI (renderer) and main threads
 - Communication schema should not constrain new features
- Solution
 - All major data processing and transformations performed on main thread (except wrapping data with HTML)
 - REST-inspired message naming: addDisc, updateDisc, deleteDisc
 - Keeping all messages and their content in standardized format (JSON objects, paths in POSIX strings, etc.)
 - Detailed and updated readme with message schemas: never committing a change in IPC without changing docs

Refined user interface



Demo



Summary

State of development, lessons learned and outlook

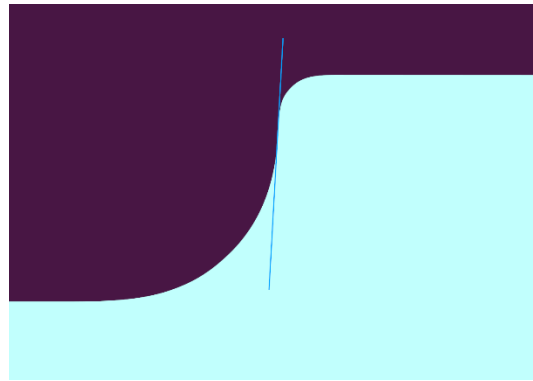
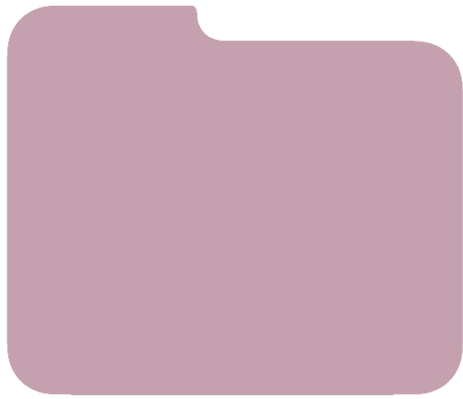
Summary & outlook

- Completed all final plans for 2nd iteration
- Fixed all known bugs from 1st iteration
- Future development opportunities:
 - Implement file opening, moving and copying
 - Allow for drag-and-drop operations between application windows
 - Add OSX support
 - Integrate application with more remote drives

Efficient native app development

- Using electron.js allowed us to develop a native windows and linux app very efficiently
- Stats until now (2nd iteration demo):
 - 90 commits by 4 developers
 - 1454 lines of code (1078 excluding tests and templates)
- Almost platform-independent:
 - Only ~20 lines of platform-dependent code
 - In theory deployable to OSX without additional overhead

Light colors and smooth curves



Using proven methods for natural, non-invasive design

Our Team

Tomasz Miśków

UI design, general research
Development:
Front-end: HTML and CSS,
Handling UI and IPC events

Krzysztof Kowalczyk

Presentation, testing
and code review
Development:
Remote API communication

Krzysztof Olejnik

Development:
File system support,
IPC design and documentation,
Remote API communication

Maciej Twardowski

Development;
File system support,
User data storage,
Handling IPC events

Thanks!

Content sources:

iPhone X image:

<https://www.apple.com/pl/iphone-x/>

iPhone X notch image and article:

<https://medium.com/tall-west/no-cutting-corners-on-the-iphone-x-97a9413b94e>

Source code and related documentation, including license information (will be available publicly 10 days after the presentation):

<https://github.com/kowaalczyk/lillian-file-manager>