Software Requirements Specification (SRS)

for project codenamed *natsu*
according to IEEE 830

Dec 2023 – Jan 2024

Table of Contents

Version History

Below is a curated list of changes made to this document in chronological order. Dates are in *ISO 8601* [ext.] format, that is, *YYYY-MM-DD*. Version *1.0* marks the release version of this document.

Date	Ver.	Author(s)	Change(s)
2023-12-07	0.1	*TophUwO*	**ADD** content structure for *Introduction* section; **ADD** text for *Purpose...*, *...Conventions*, and *...References*
2023-12-07	0.1.1	*TophUwO*	**ADD** initial content for *...Rationale* sub-section
2023-12-07	0.1.2	*TophUwO*	**ADD** contents for product feature overview
2023-12-07	0.1.3	*TophUwO*	**ADD** contents for section *General Description*
2023-12-08	0.2	*TophUwO*	**ADD** rest of document structure; **ADD** contents for multiple sections
2023-12-08	0.3	*TophUwO*	**ADD** content section for *Core System Features* (i.e., the requirements overview); **ADD** content for all sub-sections in section *General Description*; **ADD** table describing software quality requirements; **ADD** content for section *External Interface Requirements*
2023-12-09	0.5	*TophUwO*	**ADD** contents for detailed requirements for *Creation of Subjects...* sub-section inside the *Specific Requirements* section
2023-12-12	0.5.2	*TophUwO*	**ADD** specific requirements for *ReviewControl* widget and additional (optional) requirements for the application
2023-12-12	0.5.5	*TophUwO*	**FINISH** specific requirements for *Review of Items* sub-section; **ADD** diagram illustrating *stroke fanning*
2024-01-18	0.5.6	*TophUwO*	**ADD** package diagram v1
2024-01-25	0.6	*TophUwO*	**RESTRUCTURE** document entirely; adhere more closely to *IEEE 830* now
2024-01-26	0.6.1	*TophUwO*	**ADD** original user story
2024-01-28	0.6.2	*TophUwO*	**ADD** software quality attribute table; **ADD** initial references table
2024-01-28	0.6.3	*TophUwO*	**ADD** contents for *Product Perspective*, *Product Functions*, and *Business Rules* sub-sections

2024-01-28	0.6.4	*TophUwO*	**ADD** document structure for section *System Features*; **ADD** content for sub-section *Product Perspective*
2024-01-28	0.6.5	*TophUwO*	**ADD** contents for *User Classes and Characteristics* sub-section
2024-01-28	0.6.6	*TophUwO*	**ADD** TLA and OOA class diagram
2024-01-28	0.6.7	*TophUwO*	**CHANGE** highlighting of (external) links
2024-01-28	0.7	*TophUwO*	**ADD** contents for the remaining sub-sections of section *General Description*; **ADD** section *Other Requirements* plus its initial sub-section *Internationalization*; **ADD** initial optional requirements (functional and nonfunctional)
2024-01-28	0.7.1	*TophUwO*	**ADD** section *Appendix B: Supplementary Resources*
2024-01-30	0.7.2	*TophUwO*	**ADD** content for various sub-sections of section *External Interface Requirements*; **ADD** more optional requirements; **ADD** example illustration of *Genkouyoushi*
2024-01-30	0.7.5	*TophUwO*	**ADD** contents for remaining sub-sections of section *External Interface Requirements*
2024-02-03	0.8	*TophUwO*	**ADD** initial contents for section *System Features*; **ADD** initial GUI mock-ups alongside descriptions
2024-02-03	0.8.1	*TophUwO*	**ADD** remaining diagrams and table placeholders in section *System Features*
2024-02-03	0.8.2	*TophUwO*	**ADD** remaining GUI mock-ups; **ADD** requirements specifications for *customization manager* package

1. Introduction

1.1. Purpose

This document describes the software requirements specification (SRS) for a standalone desktop and notebook application dedicated to enhancing the user's *Hanzi/Kanji/Hanja* learning experience and efficiency. The application is codenamed *natsu* (jp. for *Summer*) and will be referred to as such throughout the remainder of this document. This document was created in accordance with the *IEEE 830* standard, is part of the internal developer documentation, and serves as a benchmark of what the application is intended to do and be used for.

1.2. Document Conventions

In this document, technical and other domain-specific terms relevant to the implementation are in *italics* while table heads are rendered in **boldface**. See section *Appendix A: Glossary* for an explanation of important technical terms used in the context of the product's domain. Other nomenclature (also in *italics*) may be further explained by either footnotes or in section *1.5. References*. Underlined terms in italics are also hyperlinks to other sections within this document but may point to external resources in exceptional cases. In those cases, these links are suffixed with an [ext.] annotation. This document manages its own version history (see section *Version History*). Each commit holds data such as Date, Version, Contributor(s), and Changes. Changes usually begin with a verb in infinitive form, all-caps, in boldface, describing the change's category, e.g. **ADD**, **REMOVE**, **FIX**, etc. Section *Appendix B: Supplementary Resources* contains additional media (such as images, diagrams, lists, etc.) which may be useful in the application's development. Section *Appendix C: Original User Story* contains the original mock-up product description this document was derived from during analysis and requirements engineering. Figures such as GUI mock-ups and OOA models can be found in *Appendix D: GUI Prototypes* and *Appendix E: Analysis Models*, respectively. *Appendix F: TBD* is reserved for bullet points describing details that are yet to be determined. Version *1.0* should not feature any content under that section. Romanized Japanese words in this document and all other internal documentation documents are transcribed using the *Hepburn* [ext.] transcription system.

1.3. Intended Audience and Reading Suggestions

Since this is a personal project, the only stakeholder is the project manager, developer and maintainer, *TophUwO* [ext.]. This document is also meant for future contributors of the software described within this document. Developers will be most interested in section *4. System Features*, while project managers are best served focusing on sections *1. Introduction* and *2. General Description*. Generally, the document is best read top-down since most information in the advanced chapters assumes knowledge of facts presented in earlier sections. For information regarding the appendixes, see sub-section *1.2. Document Conventions*.

1.4. Product Scope

Languages are a very prevalent subject in today's globalized society. For the current and following generations of people, mastery of different languages appears mandatory in a variety of industries that thrive in an international environment. Particularly Eastern-Asian languages are popular in the Western world due to interest, requirements, or simply popular culture. What many of the most popular languages like Mandarin, Japanese, and to an extent (South-)Korean have in common is the use of logographic characters of ancient Chinese origin. While these characters are referred to as *Kanji* (漢字) in Japanese, regional names may differ. From here on, in this SRS and all other technical documents describing this software, these characters will be referred to as *Kanji*. This SRS describes an application that is supposed to help overcome some of the frustrations commonly encountered when learning characters of Chinese origin. While there are high-quality solutions aimed at learning characters by manual input, no solution checks all the boxes. This is the gap *natsu* will fill. I believe that this application will be useful to those who want to learn Japanese handwriting in a controlled and customizable environment.

1.5. References

Below is a non-exhaustive list of useful resources that are likely to be frequently consulted throughout the conceptualization and implementation of the software described by this document. The resources presented beneath may serve as a starting point but do in no way represent a complete and necessarily sufficient resource for the entirety of the subject. Additional research is expressly endorsed. *DLA* refers to the *date of last access*. Dates are in *ISO 8601* [ext.] format, that is, *YYYY-MM-DD*.

Title	DLA	Link
Spaced Repetition	2024-01-28	*Jump! (www.en.wikipedia.org)*
IEEE Recommended Practice for Software Requirements Specifications	2024-01-28	*Jump! (www.math.uaa.alaska.edu)*
Qt6 Documentation	2024-01-28	*Jump! (www.doc.qt.io)*
KanjiVG project	2024-01-28	*Jump! (www.kanjivg.tagaini.net)*
DOXYGEN	2024-01-28	*Jump! (www.doxygen.nl)*
ISO/IEC 25010:2011	2024-01-28	*Jump! (www.iso25000.com)*
Hyougai Kanji	2024-01-28	*Jump! (www.en.wikipedia.org)*
Stroke Fanning	2024-01-28	*Jump! (www.dragice.fr)*
ISO 8601 - date and time format	2024-01-28	*Jump! (www.iso.org)*
SQLite	2024-01-28	*Jump! (www.sqlite.org)*
Anki	2024-01-29	*Jump! (www.apps.ankiweb.net)*
Unicode	2024-01-29	*Jump! (www.home.unicode.org)*
Hepburn Romanization	2024-01-29	*Jump! (www.en.wikipedia.org)*

2. General Description

2.1. Product Perspective

The software described by this document is a standalone application, supposed to be used for enhancing one's Chinese and Japanese language studies. Below is a view of all major components of natsu's software ecosystem. The *web-portal* component is an optional component; it's absence is not hindering the main application in its core functionality.

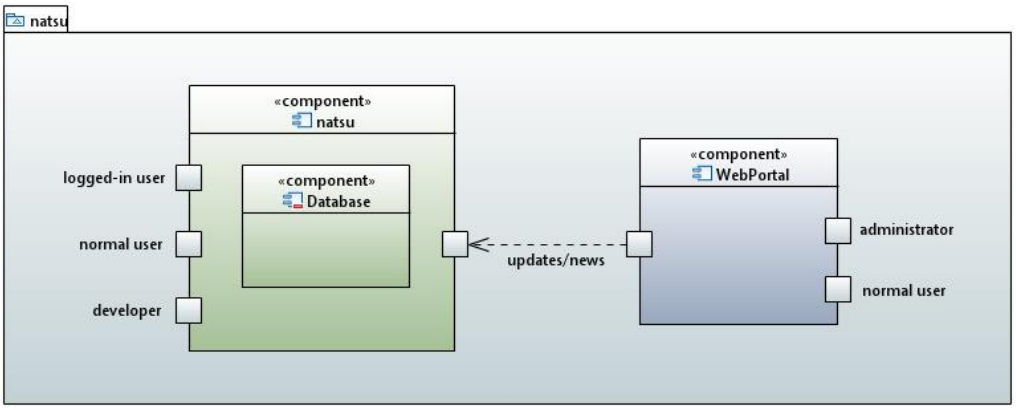

Figure 1: System Context Diagram of natsu and its topmost level components

This document describes the functionality and properties of the *offline client*—named "natsu" in the above diagram; the web-portal component will be described by its own SRS document.

2.2. Product Functions

At its core, the application should be a standalone software product that manages flashcards by means of an SRS (*spaced repetition system*). These flashcards should contain information on the Kanji character, such as *radicals*, *readings*, *stroke information*, *common vocabulary*, etc. These flashcards can be created and modified by the user. Flashcards are part of a deck that can be imported/exported from either the proprietary *natsu deck format* (NDF), or other formats such as *CSV*- or *SQL*-based databases. Progress on each item must be saved persistently in a database. Optionally, items can be grouped together in a *level order* which allows for certain subjects to be studied to retention until other items are processed. The application will be shipped with many pre-installed Kanji characters, sourced from the excellent *KanjiVG* project. There should be a way to add new characters to the repository, either by pre-made data or manual creation by means of drawing. In

the review, the user is now required to write the subject whose meaning/hint is shown. The application relies on *self-evaluation* to determine progress. Furthermore, the application should be focused on customization via application and icon styles. It should be possible to set a global *accent color* that will be used for *icons*, *focus indicators* and miscellaneous items present in the GUI. Furthermore, *plug-ins* can extend the core application's functionality in regards of additional GUI elements, additional resources, and data processing facilities. The application is to be available at all major desktop and mobile desktop platforms, as well as select mobile platforms, boasting reasonable hardware properties that make the operation and usage of the application worthwhile.

2.3. User Classes and Characteristics

The application features a multi-user infrastructure, with multiple levels of authentication: *normal* (*unregistered/guest*) user, *registered user*, and *developer*. An unregistered user has access to all components of the application that require no user information, such as the *character-* and *deck editor*. Registered users have the additional ability to review decks and sync their progress across devices using the *online portal*. Developers are registered users with the ability to view and use *developer tools* built into natsu to make development and debugging of both the application and plug-ins easier without the requirement of dedicated external tools.

2.4. Operating Environment

The application is to be run on all major desktop and mobile desktop platforms, as well as mobile platforms based on Android™ and iOS® (that is, iPadOS®). On touch devices, the minimum supported display size is 7". The minimum supported operating system is Windows® 7 or macOS® 10.11 'El Capitan'. Regarding *Linux*-based platforms, the application should be tested on all major desktop environments (i.e., *GNOME*, *KDE*, *Xfce*, *MATE*, and *Cinnamon*). Support for more platforms with smaller display sizes is an optional feature.

2.5. Design and Implementation Constraints

The application is to be implemented using the *Qt® Widgets* cross-platform application development framework (version 6.6.1 or newer). The programming language used for the core application is C++17 or newer. The language of the codebase, inline documentation as well as internal (technical) documents is *English (United States)*. Third-party frameworks

other than Qt® can be used if they are not licensed under a strong copyleft license like *GPL*. Inside the Qt® ecosystem, all APIs that are not solely available under a strong copyleft license can be used freely. Persistent storage of application settings and review data is to be implemented using a relational database system like *SQLite* or MySQL®.

2.6. User Documentation

Aside from the SRS provided by this document, there will be a *DOXYGEN*-based online developer documentation available. Furthermore, there must be some form of user documentation, outlining the application's features. The user documentation must be accessible from the application (e.g., via a hyperlink or an embedded manual). The format of the user documentation is HTML. A printable PDF edition is not required but may be shipped. There should be a way to download the online documentation for offline use.

2.7. Assumptions and Dependencies

The application's source code is to be released under an open-source license. This requires all used external components to be released and used under non-copyleft licenses. If the license terms for one of the main technologies used (e.g., Qt®) change, the release of the application described by this document may be negatively affected.

3. External Interface Requirements

3.1. User Interfaces

The application's user interface should be consistent across platforms. That is, it should be both usable from desktop- and mobile platforms. Additionally, the user interface should scale with the DPI of the output device. For more detailed suggestions of user interface designs, see section *Appendix D: GUI Prototypes*.

The general GUI layout consists of a *content area* and a main *toolbar* as seen in the image below.

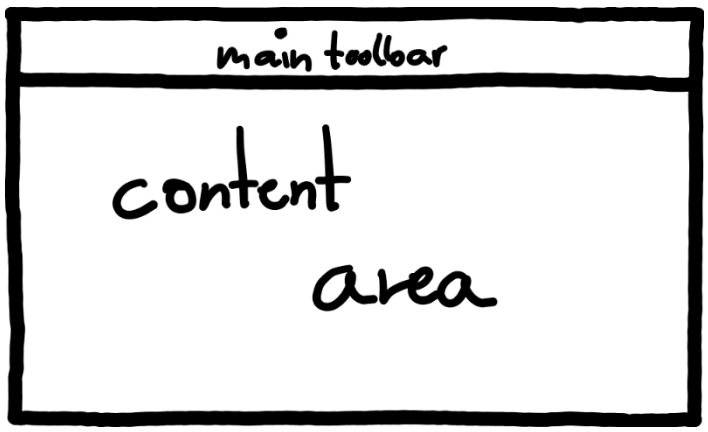

Figure 2: natsu's main window layout

The layout is optimized for *landscape* orientation; optimization for portrait-oriented devices is an optional feature. The main toolbar must be non-movable and optically separated from the content area. The content area contains a dock area used for widgets that can be moved around freely. The window layout of the dock area may be saved across sessions and imported/exported via application settings import/export. Since the application contains widgets designed for handwritten input, there should be standard window layouts optimized for left- and right-handed people, respectively. The user can create new window layouts in the settings. Even when a window layout is selected and active, dock widgets can still be moved around freely. Their layout, however, is not saved as soon as the application leaves that specific view.

3.2. Hardware Interfaces

The application natively supports mouse and touch input. Touch input may be supplied by touchscreens. Support for graphics tablets (like Wacom™) is optional. If input is done

through an active pen, support for pen features like erasing or tools the pen natively provides via buttons is optional. The software does not interact directly with input or output hardware. Rendering of the application is, by default, done through Qt®-provided abstract interfaces. Rendering using less abstract technologies like Vulkan®, etc. are optional. Hardware-accelerated rendering using OpenGL®, however, is a required feature which is disabled by default but may be enabled by the user through the settings dialog. Support for printing is done through abstract interfaces provided by Qt® itself or the host platform. If printing is not supported on any target platform, printing becomes an optional feature.

3.3. Software Interfaces

Software interfaces include dialogs for opening and saving files as well as dialogs for choosing fonts and colors. Where possible, OS-specific dialogs should be used. If this is not possible and the functionality provided by *Qt® Dialogs* is enough, these dialogs should be used. Informational or warning message boxes should feature a checkbox that controls whether the message box is shown to the user the next time it is about to be shown. In these cases, this setting may be changed separately in the settings dialog for every dialog that falls into that category. Error messages boxes cannot be turned off. If the application terminates abnormally, an error message is shown that also allows the user to send a crash report to the application maintainer. This setting may also be toggled.

3.4. Communications Interfaces

Communication with the web portal will generally be done through HTTPS and is thus generally encrypted. Local communication is carried out through opt-in desktop notifications. If external files are to be opened, the application may open default applications registered for that purpose (e.g., PDF viewer, web browser, E-Mail application, ...). To send private messages to users, e.g. for resetting the user's account password, the application may send E-Mails to the given address. If no E-Mail was provided, no E-Mails are sent.

4. System Features

Below is a collection of all major functionalities contained in natsu's main client. Each component features a use-case diagram (UCD), describing common operations accessible by users and other components. Each visible use-case is then explained in a table listing the use-case's title, pre- and post-conditions, as well as detailed functional requirements.

4.1. Plug-in Manager

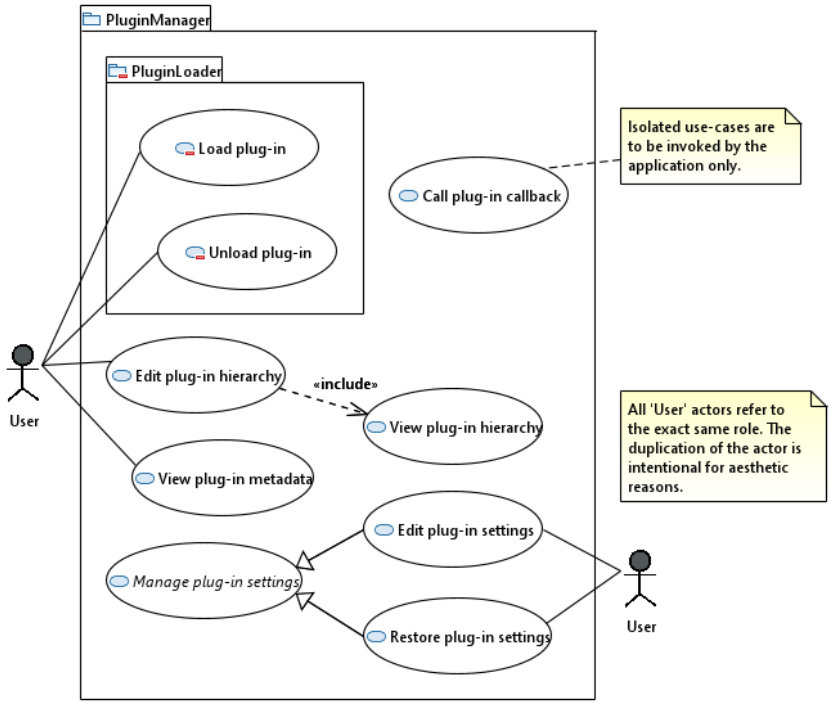

Figure 3: use-case diagram for the plug-in manager component

Title	Conditions	Requirements
View plug-in hierarchy	**Pre:** - plug-in dialog opened - plug-in hierarchy tab selected **Post:** none	- show table-like view, displaying plug-in *name*, *author*, *version*, *description*, *installation date*, and *state* - rows and text are selectable - titles/descriptions too long to display are either wrapped or shortened with '…' - table uses alternating row colors - alternatively, instead of a table, show rows of *cards* for plug-in entries

Edit plug-in hierarchy	**Pre:** - plug-in dialog opened - plug-in hierarchy tab selected **Post:** - plug-in order changed - saved persistently	- allow drag and drop between plug-in entries, changing their precedence
View plug-in metadata	**Pre:** - views plug-in hierarchy - clicks on a button belonging to a plug-in entry **Post:** none	- click on a 'details' button next to a plug-in entry - show plug-in metadata in a separate dialog in a properties table - allow closing the dialog to get back to the plug-in hierarchy view
Edit plug-in settings	**Pre:** - views plug-in hierarchy - clicks on a button belonging to a plug-in entry **Post:** - plug-in settings changed - saved persistently	- click on a 'settings' button next to the 'details' button - allow applying, closing, and resetting the plug-in settings dialog - allow closing the dialog to get back to the plug-in hierarchy view
Restore plug-in settings	**Pre:** - plug-in settings dialog opened **Post:** - plug-in settings restored to defaults - saved persistently	- click on a 'Restore plug-in settings' button in the plug-in settings dialog - still allow editing settings after restoring - allow closing the dialog to get back to the plug-in hierarchy view
Call plug-in callback	**Pre:** - plug-in manager initialized **Post:** - side-effects of plug-in callback	- call plug-in hooks in case of a specific event (e.g., *on_review_start*) top-down in the hierarchy - catch all synchronous exceptions and report back, don't terminate - do not call hooks of disabled plug-ins - report errors in form of log messages

4.1.1. Plug-in Loader

Title	Conditions	Requirements
Load plug-in	**Pre:** - plug-in manager initialized **Post:** - plug-in loaded or unmodified in case of an error	- allow loading a plug-in from a plug-in file (ZIP archive) - unpack ZIP to a user-specific directory - dynamically load plug-in executable - call initialization hooks - do not load plug-in twice if already loaded
Unload plug-in	**Pre:** - at least one plug-in loaded **Post:** - plug-in unloaded	- remove dynamic link to library - remove local files for plug-in

4.2. Customization Manager

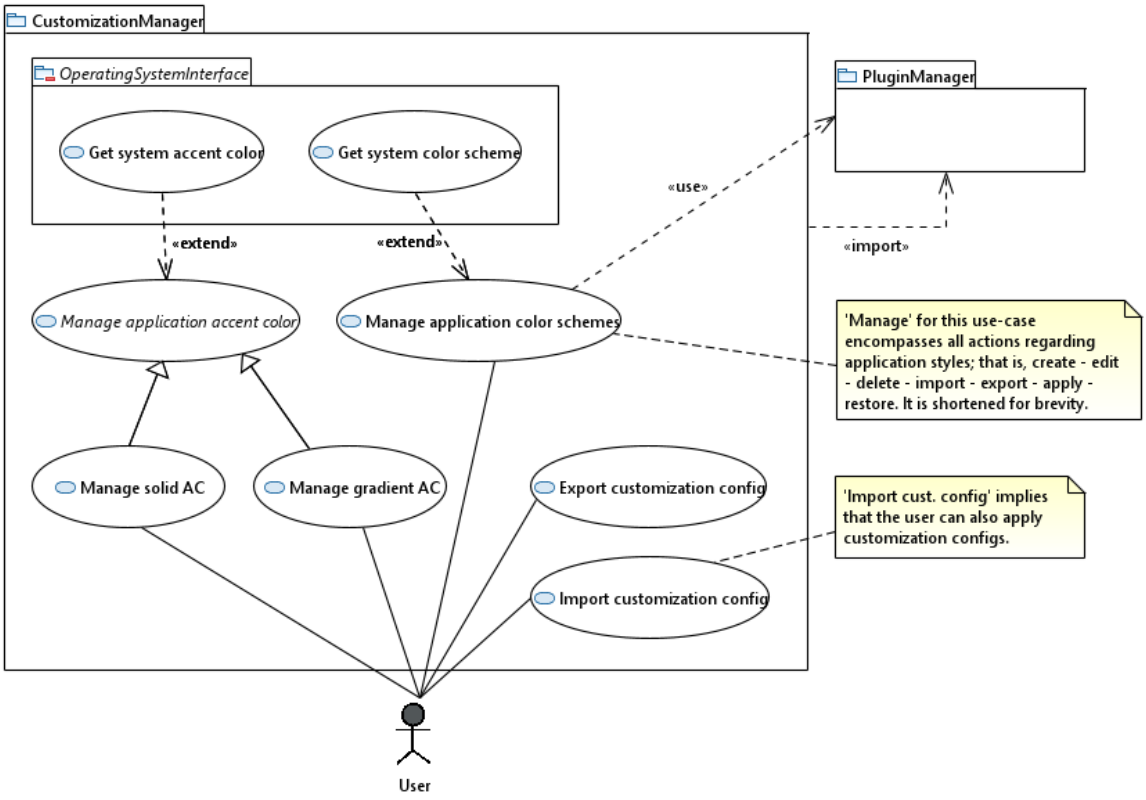

Figure 4: use-case diagram for the customization manager component

Title	Conditions	Requirements
Get system accent color	**Pre:** - customization manager initialized **Post:** none	- retrieve accent color using OS-specific interfaces - if not obtainable, return default accent color
Get system color scheme	**Pre:** - customization manager initialized **Post:** none	- retrieve color scheme using OS-specific interfaces - if not obtainable, return default theme
Manage solid AC	**Pre:** - customization dialog opened **Post:** none	- allow solid accent colors - set colors using color picker (Qt or OS) - allow preview of any icon in a small window
Manage gradient AC	**Pre:** - customization dialog opened **Post:** none	- allow gradient accent colors - linear gradient and radial gradient - set gradient using custom dialog - set colors using color picker (Qt or OS) - allow preview of icons in a small window
Manage application color schemes	**Pre:** - customization dialog opened **Post:** - color schemes applied or modified - saved persistently	- allow loading new application color schemes from files - allow creating new color schemes with a special dialog and allow modifying existing color schemes - apply new color schemes with instant effect
Import customization config	**Pre:** - customization dialog opened **Post:** - config imported and applied - saved persistently	- import from file path - apply customization config, overwriting the current - do not change in case of an error
Export customization config	**Pre:** - customization dialog opened **Post:** - config exported	- export to file

4.3. Character Repository

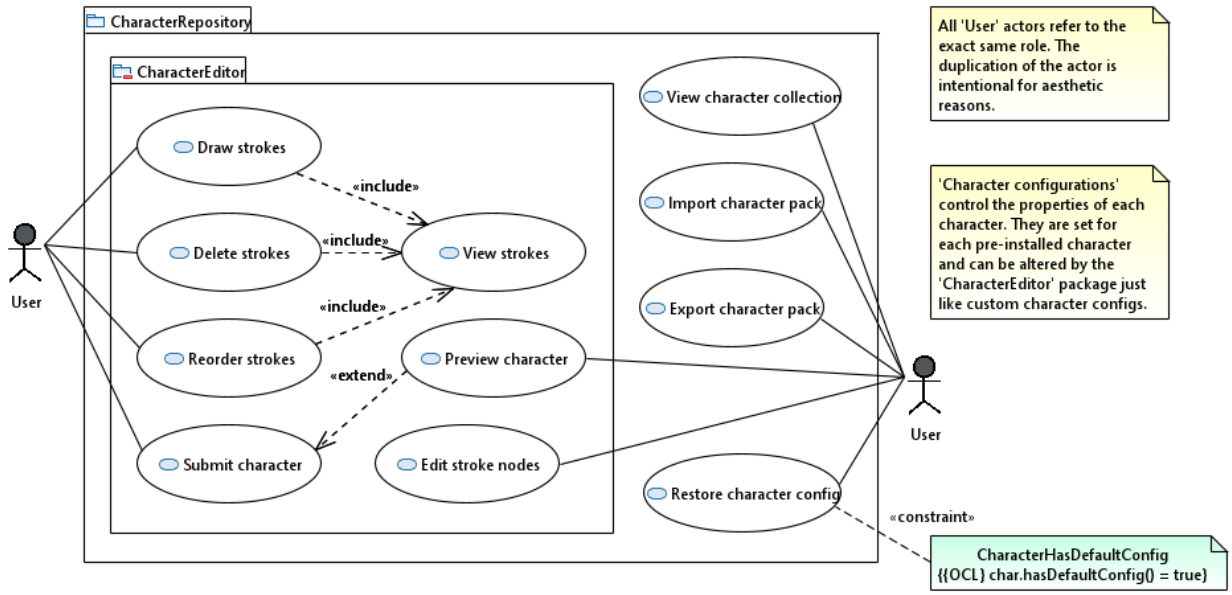

Figure 1: use-case diagram for the character repository

Title	Conditions	Requirements
View character collection		
Import character pack		
Export character pack		
Restore character config		

4.3.1. Character Editor

Title	Conditions	Requirements
View strokes		
Preview character		
Edit stroke nodes		
Draw strokes		
Delete strokes		
Reorder strokes		

Submit character		

4.4. Review Manager

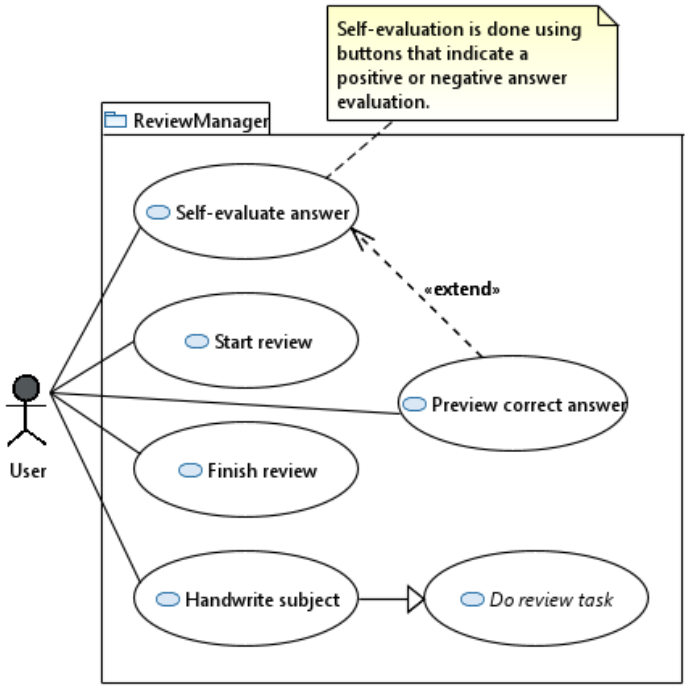

Figure 2: use-case diagram for the review manager

Title	Conditions	Requirements

4.5. Deck Manager

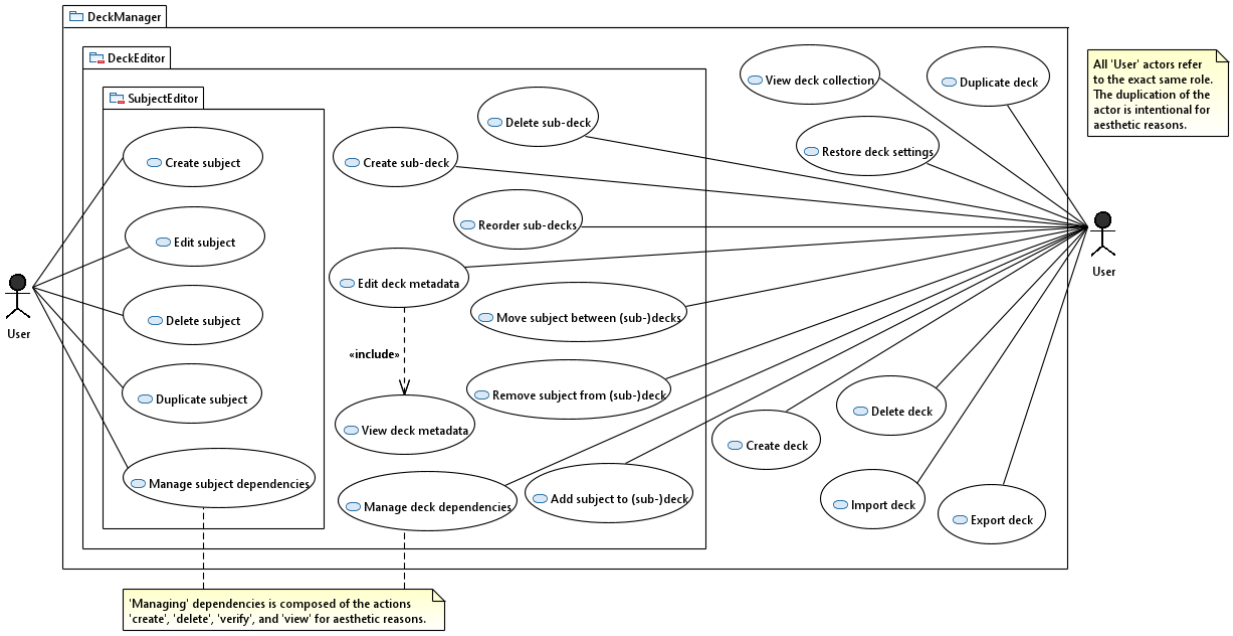

Figure 3: use-case diagram for the DeckManager component

Title	Conditions	Requirements

4.5.1. Deck Editor

Title	Conditions	Requirements

4.5.2. Subject Editor

Title	Conditions	Requirements

4.6. User Manager

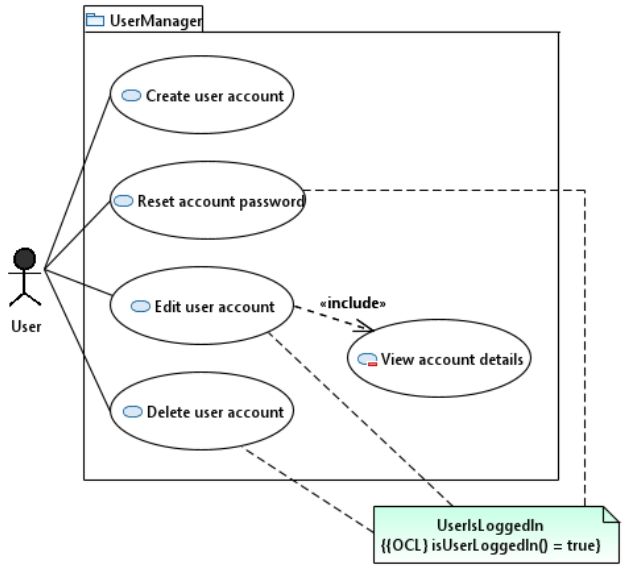

Figure 4: use-case diagram for the user manager component

Title	Conditions	Requirements

4.7. Database Manager

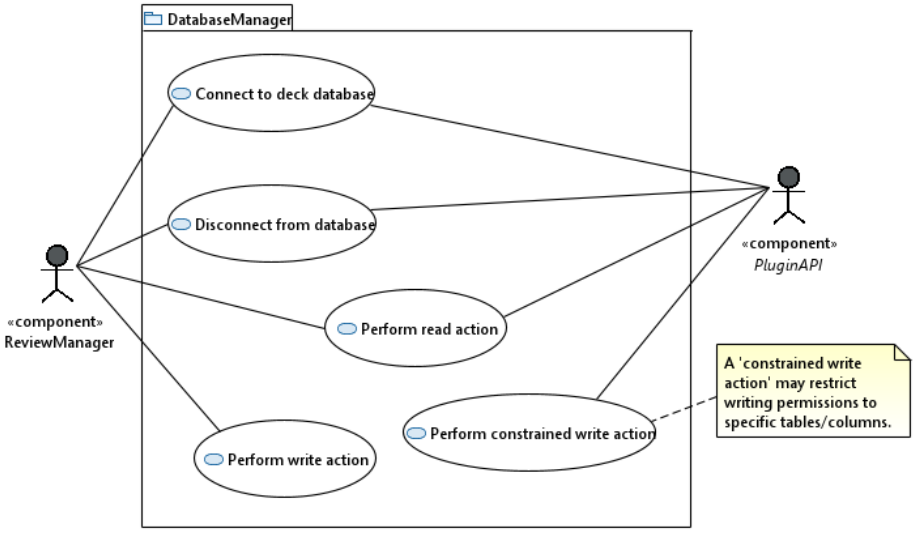

Figure 5: use-case diagram for the database manager

Title	Conditions	Requirements

4.8. Application Settings

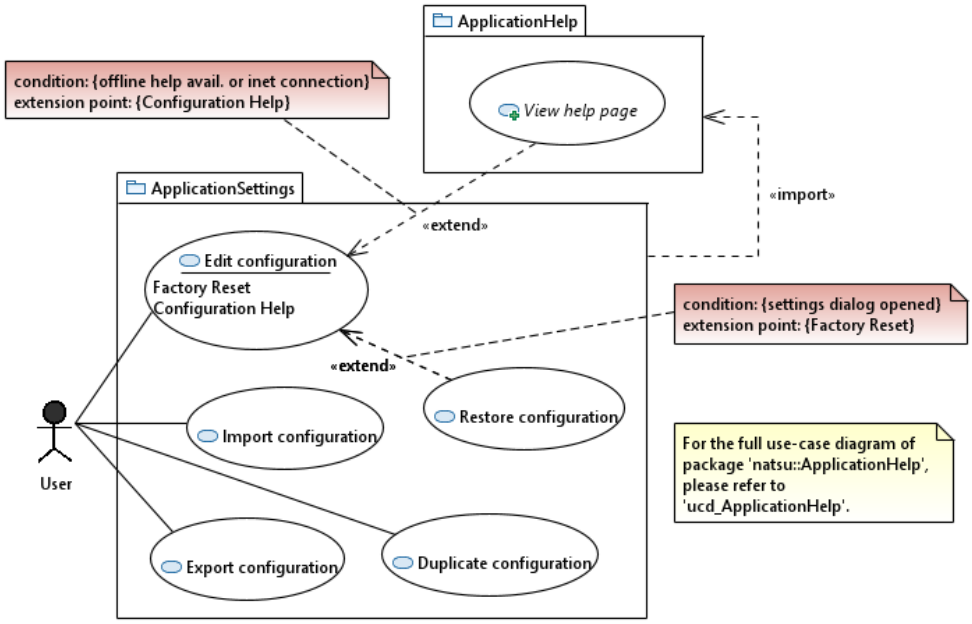

Figure 6: use-case diagram for the application settings component

Title	Conditions	Requirements

4.9. Application Help

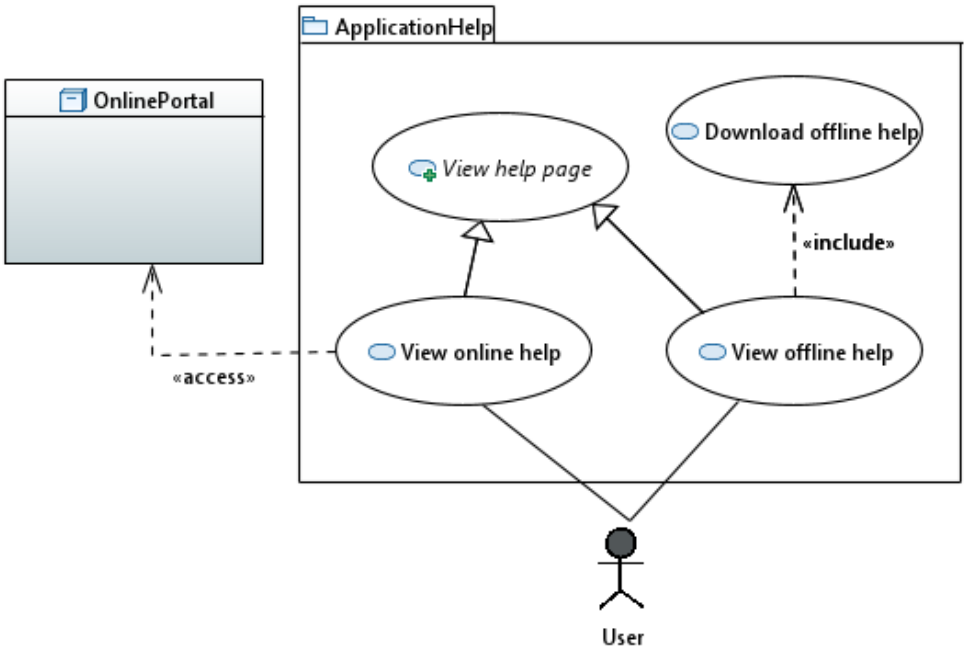

Figure 7: use-case diagram for the application help component

Title	Conditions	Requirements

4.10. Developer Tools

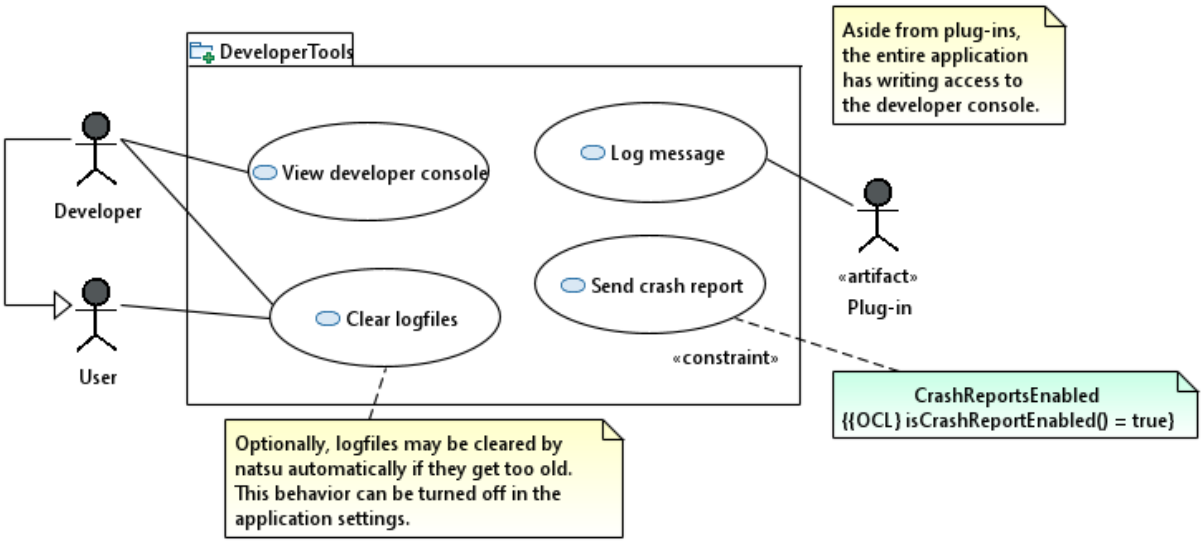

Figure 8: use-case diagram for developer tools

Title	Conditions	Requirements

5. Other Nonfunctional Requirements

5.1. Performance Requirements

There should be no noticeable lag when interacting with the application's user interface. If an action is expected to take longer than 0.5 seconds, a progress bar indicating progress should be shown. Very expensive operations should be offloaded to separate threads to not block the main thread. When desired, GPU-accelerated rendering should be used. When using GPU-accelerated rendering, Qt®'s built-in *QOpenGLWidget* should be used where supported. Rendering using other technologies (i.e. *Direct2D*®, *Vulkan*®, etc.) are completely optional features.

5.2. Safety Requirements

There are no specific safety requirements in place. If the graphical user interface (GUI) is to be animated, the animations should not feature rapidly blinking lights which may cause irritation for some groups of people. Animations should be fluid and may be turned off in the settings. Where fluid animations are not possible, they should be omitted entirely.

5.3. Security Requirements

Registered user accounts must have a non-empty password phrase associated with it. These passwords must satisfy the following requirements:

- at least six characters in length; max. length is 30 characters
- at least one letter, one number, and one symbol
- not a prefix, infix, or suffix of the user's username
- can use the entirety of the *Unicode*® character set

Passwords that do not satisfy these requirements must be rejected. There should be a way to display the specific requirements to the user upon request. These passwords must be saved and securely hashed. Use of a *cryptographic hash function* such as *SHA-256* is strongly recommended. If user data is transferred from the application to the online portal or vice-versa, the communication must be encrypted. Before the application exits, any logged-in user should be logged out. If the application shuts down abnormally and is restarted, a user that was previously logged in should be logged out automatically. There should be no way to remember access data. Crash reports, if enabled, should be sent anonymously (that is, with no personal information about the user such as username, password, IP,

MAC, etc.). Hardware properties such as processor, memory configuration, etc. and software settings may be sent alongside crash reports. When a crash occurs for the first time, the user must be asked to send them. Whatever the choice, the user must be informed that the current selection may be altered at any time in the application settings.

5.4. Software Quality Attributes

Below is a list of all *ISO/IEC 25010:2011* software quality requirements alongside an integer value, the level of importance, in the interval [1, 5] where *5* denotes very high importance and *1* denotes low importance relative to level *5*.

Requirement	Description	Priority
functional suitability	This characteristic represents the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions.	**4**
performance efficiency	This characteristic represents the performance relative to the number of resources used under stated conditions.	**3**
compatibility	Degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions while sharing the same hardware or software environment.	**1**
usability	Degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.	**5**
reliability	Degree to which a system, product or component performs specified functions under specified conditions for a specified period.	**2**
security	Degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization.	**2**

maintainability	This characteristic represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it or adapt it to changes in environment, and in requirements.	**3**
portability	Degree of effectiveness and efficiency with which a system, product or component can be transferred from one hardware, software or other operational or usage environment to another.	**4**

5.5. Business Rules

Certain operations inside the application can only be carried out by a registered and logged in user. The user account, however, only needs to be local and does not necessarily have to be linked to an account present in any online database. Any registered user account can promote itself to a *developer account*, gaining local access to debugging features for plug-ins and the application itself. The promotion is not synched globally and only applies to the user being logged into the specific instance of natsu the user promoted themselves in.

6. Other Requirements

6.1. Internationalization

The application's default language is *English (United States)*. Additionally, the application must provide a way to change the GUI's language to *German (Germany)*. Translations into languages other than English or German are optional features. There should be a way to install new languages. Language packs should be distributed as Qt [*.qm] files generated from Qt®'s own in-ecosystem internationalization tool, *Qt® Linguist*. Language settings should be saved per-user. There should be a way to get the current operating system's language and set the same language in the application, if existing. If no language pack for the current OS language can be found or installed, use the set default language (must be an installed language pack). Application help, if shown, is displayed in the same language as the current application's language.

7. Optional Requirements

This section contains a curated list of optional requirements, sorted based on their characteristics. Optional requirements are requirements that are not needed to fulfil the specification represented by this document but may be implemented and shipped in the form of platform updates. Since the application features a plug-in infrastructure, shipping by means of plug-ins is also an option. If they extend the functionality of the application's main components directly, they will be listed as *Functional Requirements* while features that define or alter miscellaneous properties of the application are listed as *Nonfunctional Requirements*.

The tables below are structured as follows: Each feature is associated with one or more packages defined in the *TLA of natsu's main client*. Each feature is then explained with a brief text or a few bullet-points. External requirements, other dependencies, and additional TBD points are also listed.

7.1. Functional Requirements

Package	Description	Externals/TBD
CharacterRepository	- OCR for characters of Chinese origin (possibly AI model?) - used in *CharacterEditor* and potentially also the *ReviewManager* for automatic evaluation of user input	- training- and test data (regulations?) - Python embed?
ReviewManager, CharacterRepository	- graphics tablet support (e.g. Wacom™, etc.)	
DeckManager	- *Anki* deck import	- regulations?
ReviewManager, CharacterRepository	- *canvas* image export	
CharacterRepository	- physical practice sheet generator for Kanji; like *Genkouyoushi* - print support; print to paper and PDF - render to SVG or various image formats - example: click *here*	
natsu (root pkg)	- full support for Python scripting - extend UI with Python	- Python embed?

CharacterRepository	- generation of character property sheet; customizable	- license regulations regarding KanjiVG
	-	-
	-	-
	-	-
	-	-

7.2. Nonfunctional Requirements

Package	Description	Externals
ApplicationSettings	- support for additional languages	
ReviewManager	- support for desktop notifications as soon as new reviews are available	
ApplicationHelp	- support for additional application help languages	
ApplicationSettings	- group settings into two categories called something like *normal* and *expert* where *normal* only shows commonly used settings items while *expert* shows all available items	- sorting criteria
ApplicationSettings	- show brief help string when hovering over the label for a specific settings item	
PluginManager	- install plug-ins from online repository	

Appendix A: Glossary

Below is a curated list of context-specific nomenclature used throughout this document, alongside brief explanations.

Term	**Meaning**
radical	graphical parts of a Kanji used for indexing
reading	a way of pronouncing a character; context-sensitive
stroke	smallest distinguishable grapheme in a Kanji/radical
Qt	cross-platform application framework
GPL	popular license for open-source development
(Kanji) variant	different variant of given Kanji such as *Hyougai* variants
review item	same as *flashcards*
Anki	open-source flashcard learning software based on spaced repetition
Unicode	technical standard specifying a character set that is supposed to hold all written characters ever produced by humanity
Genkouyoushi	paper used in Japanese handwritten documents featuring gridlines and other guides; click *here* for an example
OS	operating system

Appendix B: Supplementary Resources

This section lists supplementary resources used by sections of this document. They do not represent content that is to be added to the application, but examples used to illustrate features, requirements, or properties. Attribution as per original file license terms is done through the resource's caption, along with a (potentially external) link to the original file's license.

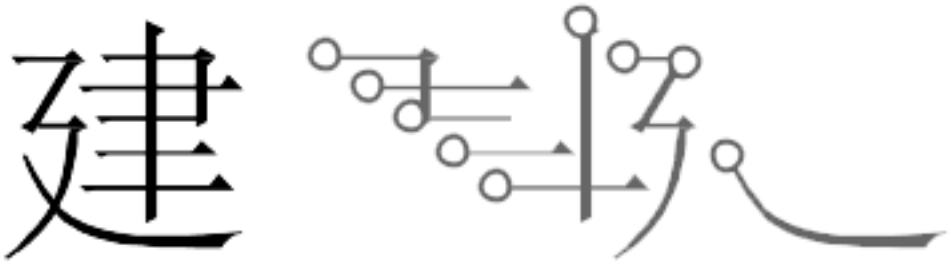

Figure 9: illustration of 'Stroke Fanning', via dragice.fr [ext.]

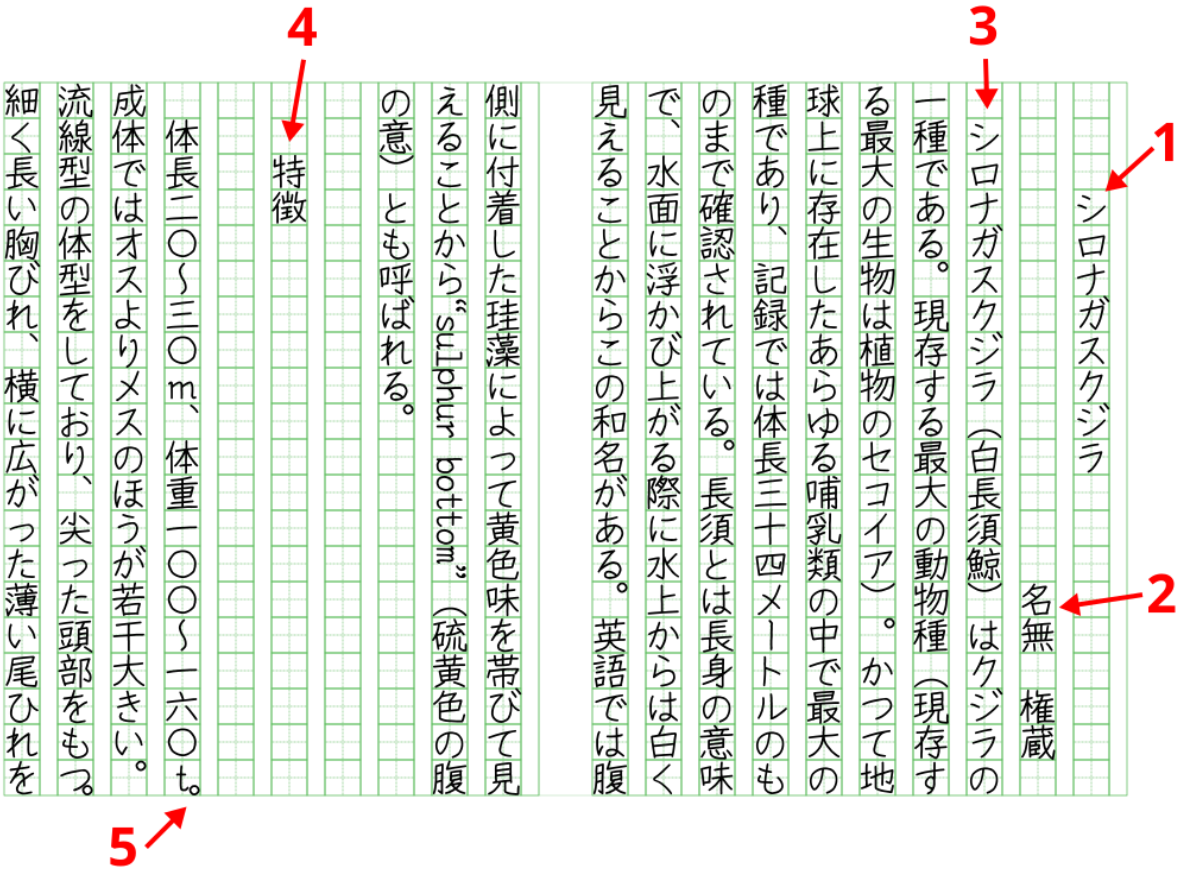

Figure 10: illustration of Genkouyoushi (ignore numbering); via commons.wikimedia.org [ext.]

Appendix C: Original User Story

Languages are a very prevalent subject in today's globalized society. For the current and following generations of people, mastery of different languages is mandatory in a variety of industries that thrive in an international environment. Particularly Eastern-Asian languages are popular in the Western world due to interest, career requirements, or simply popular culture. Research has shown that the learning effect of characters and words in foreign languages when handwriting, using ink- or digital pen, is superior to typing on a keyboard[1]. Existing solutions to utilize handwriting characters, however, often lack one serious aspect: features, extensibility, customizability, or sheer platform support. For this reason, an all-new hand-writing learning application based on spaced recognition (SR), focused on characters of Chinese origin, is to be implemented. In the following, the implementation will be codenamed *natsu* (jp. for *Summer*).

The central component of the application will be the *review* component, allowing the user to practice *review items* which were currently installed in the application using *decks*. There should be multiple ways to study the deck, mainly *source* → *target* (i.e. translation/meaning is shown, representation in target language is requested), *target* → *source* (representation is shown, meaning is requested). Since the software is based on spaced recognition, the application must calculate an appropriate interval after each successful recall. After several consecutive successful recalls, the item will be *buried*/*burned*, meaning it will not appear in reviews any longer but can still be reviewed when requested. For that reason, there should be a way to specifically review burned items. When reviewing items, the user is required to handwrite the answer on a *canvas*, capturing the user's input via mouse/tablet pen/touch input. Once the user is finished and confident in their answer, the user selects a button that shows the answer in a different widget, next to the user's answer. The user can interact with this widget to view the stroke order of the subject (if applicable) in a variety of ways (from static, animation, to stroke fanning and more).

Decks should be created inside the application, using a *deck editor* component. There is no limit of how many items can be inside a deck, but the application should be able to handle decks of at least 100.000 items. Decks can be imported and exported by the deck editor. Exported decks can then be distributed as a single disk file, and subsequently be imported into another *natsu* instance. Backwards compatibility between deck and application

[1] *https://www.frontiersin.org/articles/10.3389/fnhum.2021.679191/full*

versions should be preserved as much as possible. To mitigate the problem, the application should allow migrating the deck to any application version, up or down the version tree.

To display characters alongside information such as stroke order and other data, the application must curate a *character repository*. This component manages all glyphs the user can use to construct review items. If a review item contains a character that is not currently available inside the repository, a warning must be shown but the subject should still be accepted. To extend the character repository, a *character editor* component is introduced. The character editor allows the user to handwrite a new character on a canvas, like the canvas used in reviews. The input is then transformed into paths that can then be drawn and transformed by the character view widget. Characters are drawn stroke by stroke, allowing the user to undo/redo strokes. The character preview widget should allow to preview the character at various sizes, in various stroke order display methods. The functionality provided by this widget should be like the functionality exposed by the *character view* widget used in reviews. Furthermore, there should be a way to reorder strokes using a graphical widget resembling a *story board*. In theory, there can be multiple characters in the repository representing the same Unicode codepoint. However, when a review item is created, the user must disambiguate and choose a specific variant.

A good learning environment requires good customization options, functionally as well as visually. For this reason, the application should implement ways to alter the appearance of itself, as well as ways to extend the application's functionality using plug-ins. These plug-ins can be either native (i.e. using shared objects) and/or scripts. The plug-ins can add additional widgets to most views in the application, add application themes, additional icons, etc. Visual customization should be done through a *customization manager* component. The two main ways of customization are *accent colors* and *themes*. The accent color is used for icons and widget effects, while themes are used to alter the application's base color palette. There should be pre-installed themes and accent colors. Accent colors can be either solid colors or gradients. Where possible, there should be a way to set an automatic accent color and theme according to system preferences. Additionally, there should be support for internationalization; language packs can be installed from downloadable resources. Some language packs come shipped with the retail application.

The application should have support for drawing tablets and touchscreens. Touchscreens can be desktop touch screens as well as mobile devices. The application should support both Android and iOS (iPadOS) tablets.

Appendix D: GUI Prototypes

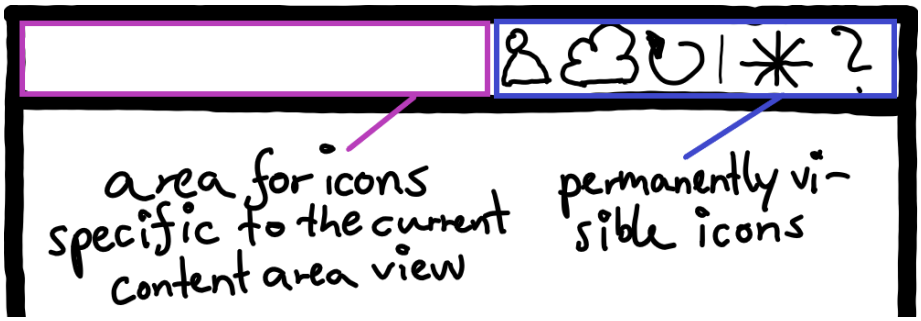

Figure 11: main toolbar area description

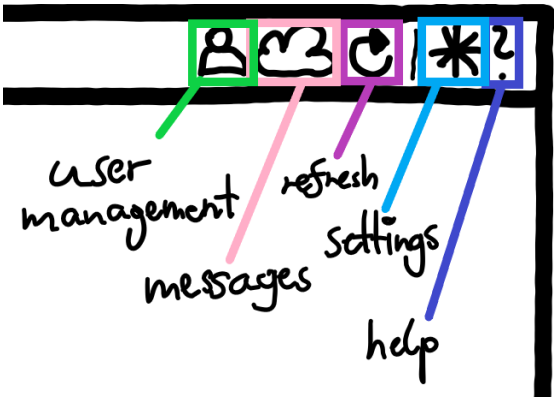

Figure 13: main toolbar persistent actions

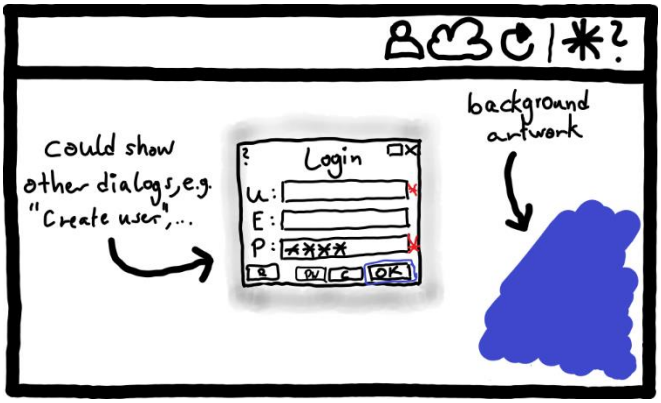

Figure 14: natsu welcome screen

In *developer mode* (setting accessible via settings), the persistent icons are extended by an icon representing a command line window (positioned between the *settings* and *help* icons), giving access to developer features.

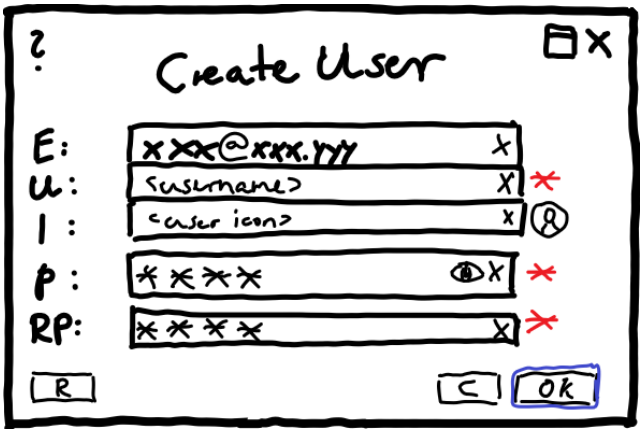

Figure 12: GUI mock-up of the 'Create User' dialog

The welcome screen depicted in *Figure 5* may show other dialog windows depending on the situation. If, for example, no user exists, the *Create User* dialog is shown instead. The dialogs can be cancelled to gain access to the main toolbar and other widgets.

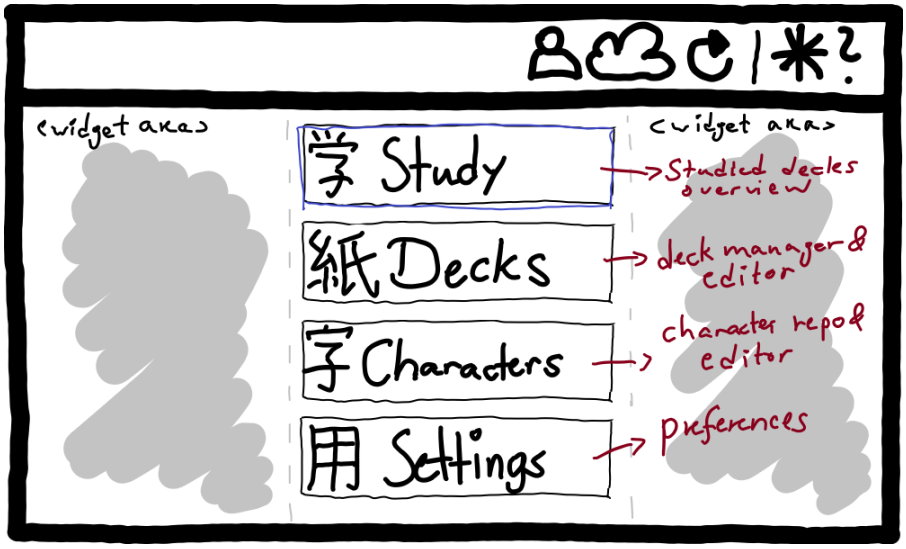

Figure 15: illustration of main menu; shown after log-in

In the main menu, the menu item *Study* opens the *deck study view*. The menu item *Decks* opens the *deck overview* which gives access to the *deck manager* and *deck editor*.

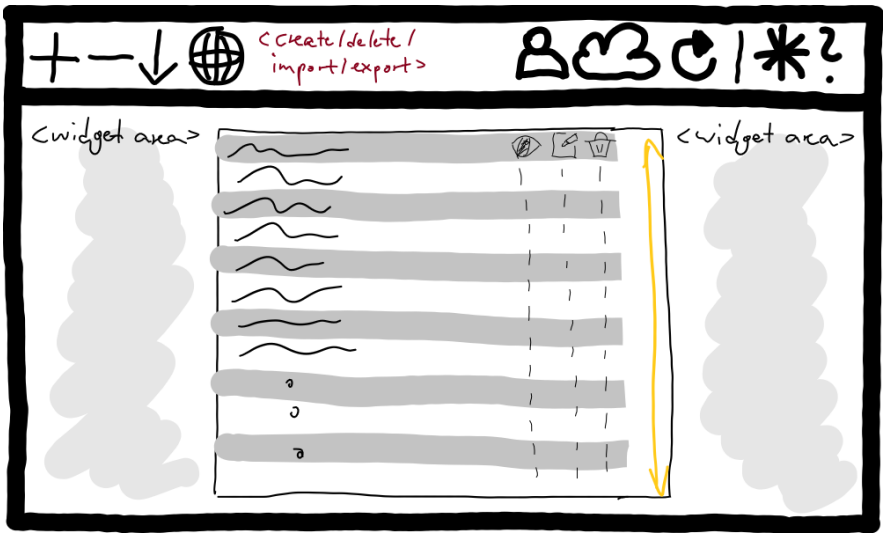

Figure 16: mock-up showing the deck overview

The deck overview features a table, aligned in the center area of the window, with *widget areas* left and right. Widget areas function as *extension points* for plug-ins to hook onto and

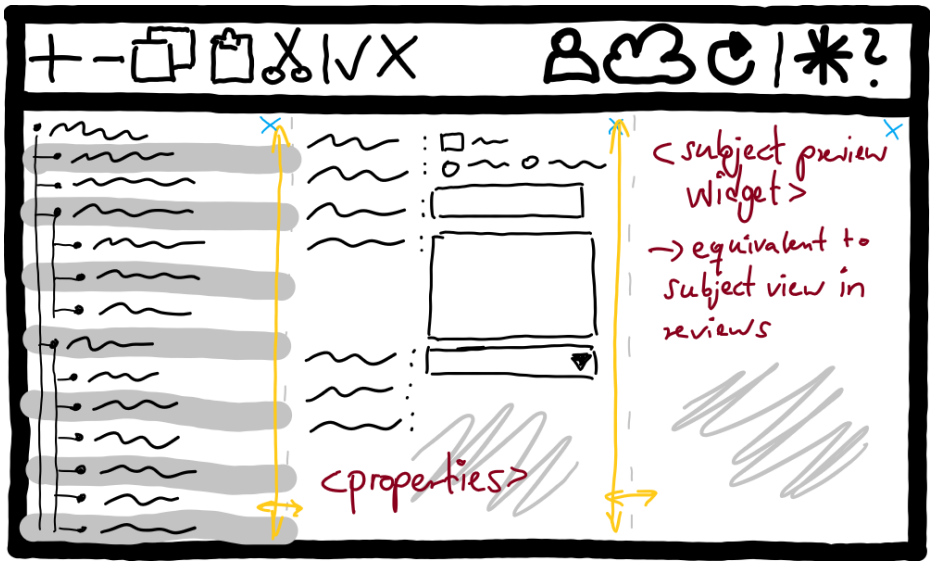

Figure 19: deck editor; showing all review items in the selected deck

display extension widgets. Each deck has three actions available in the table: *active*, *edit*, and *delete*. Changing the *active* state of a deck allows the user to quickly turn decks on or off in order to make their items appear or not appear in reviews.

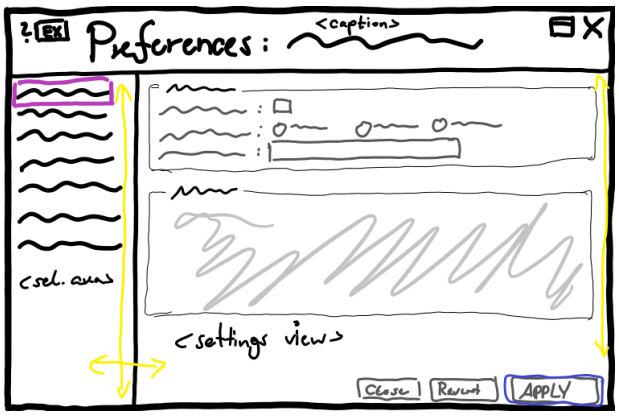

Figure 18: settings dialog variant 1; list view for navigation

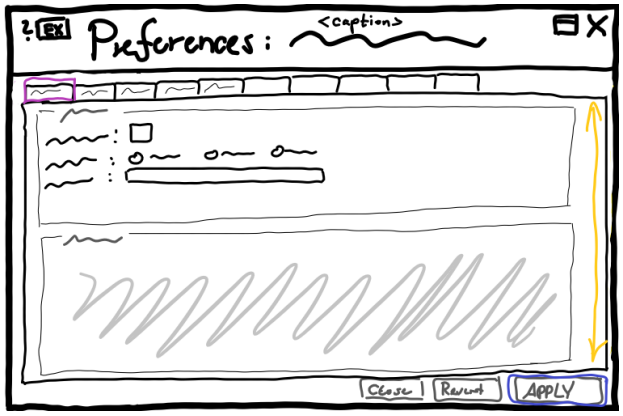

Figure 17: settings dialog variant 2; tabs for navigation

Above are two different takes on a settings dialog. Only one variant must be implemented. Even though both variants show that the settings are to be implemented as a top-level dialog, they might also be implemented as a *view* in the main window.

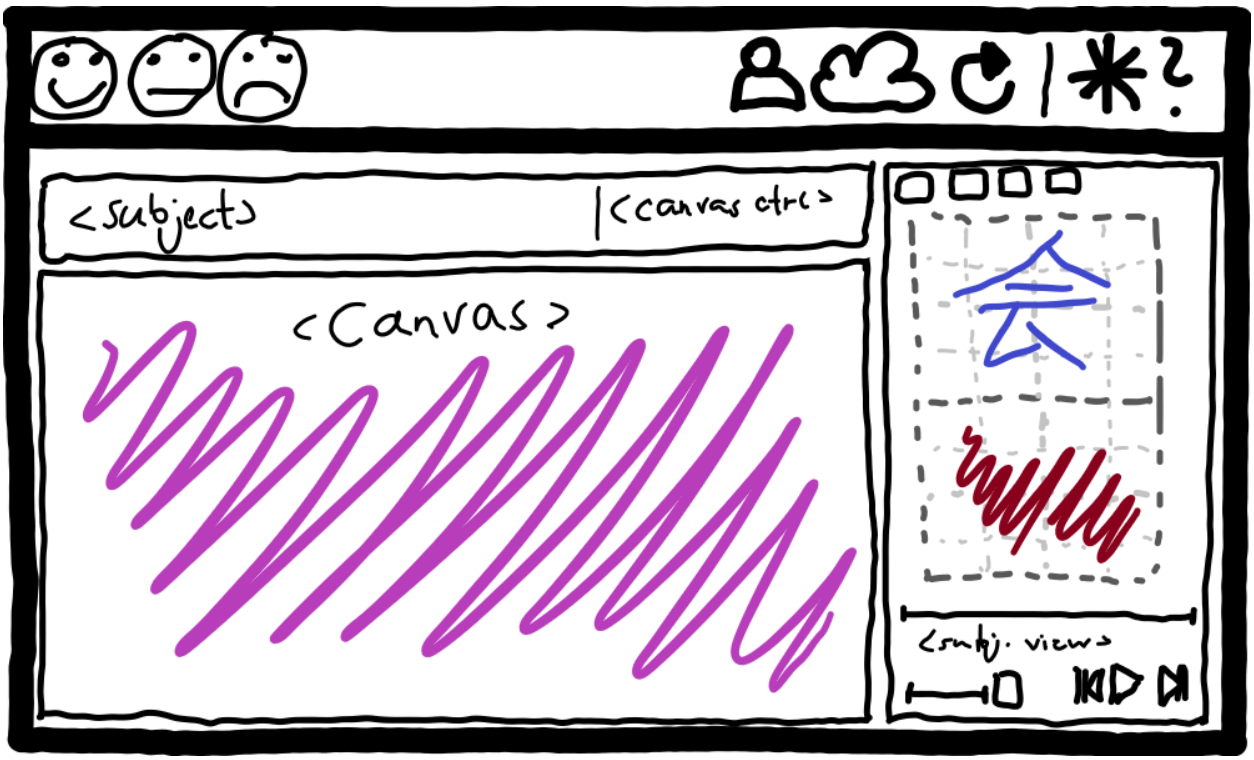

Figure 21: mock-up for review view

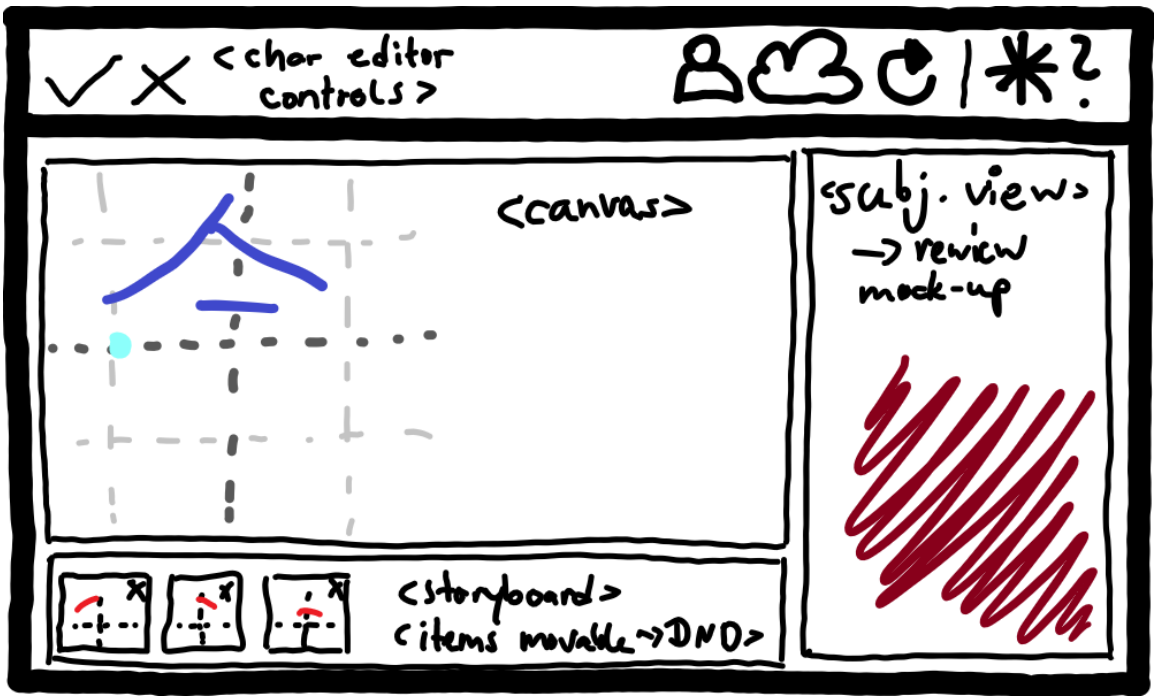

Figure 20: mock-up for the character editor view

Appendix E: Analysis Models

Below are the main models derived from the original user story and the contents of this document.

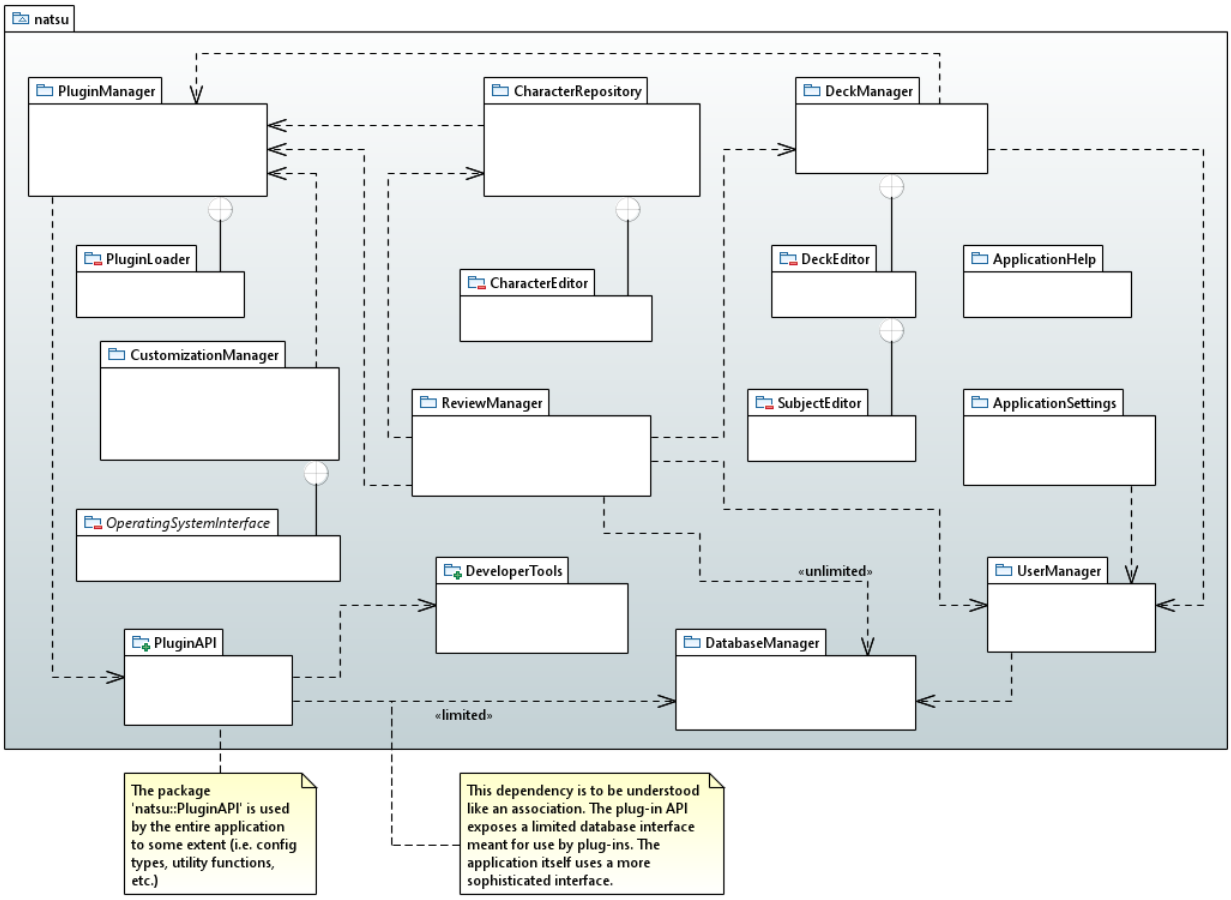

Figure 22: top-level architecture (TLA) diagram for natsu's offline client

The above TLA diagram represents how components (represented as packages) inside the offline client interact with each other. Not all components in the TLA diagram are explained thoroughly in this document. For description and documentation of the plug-in architecture, please refer to natsu's *developer documentation*.

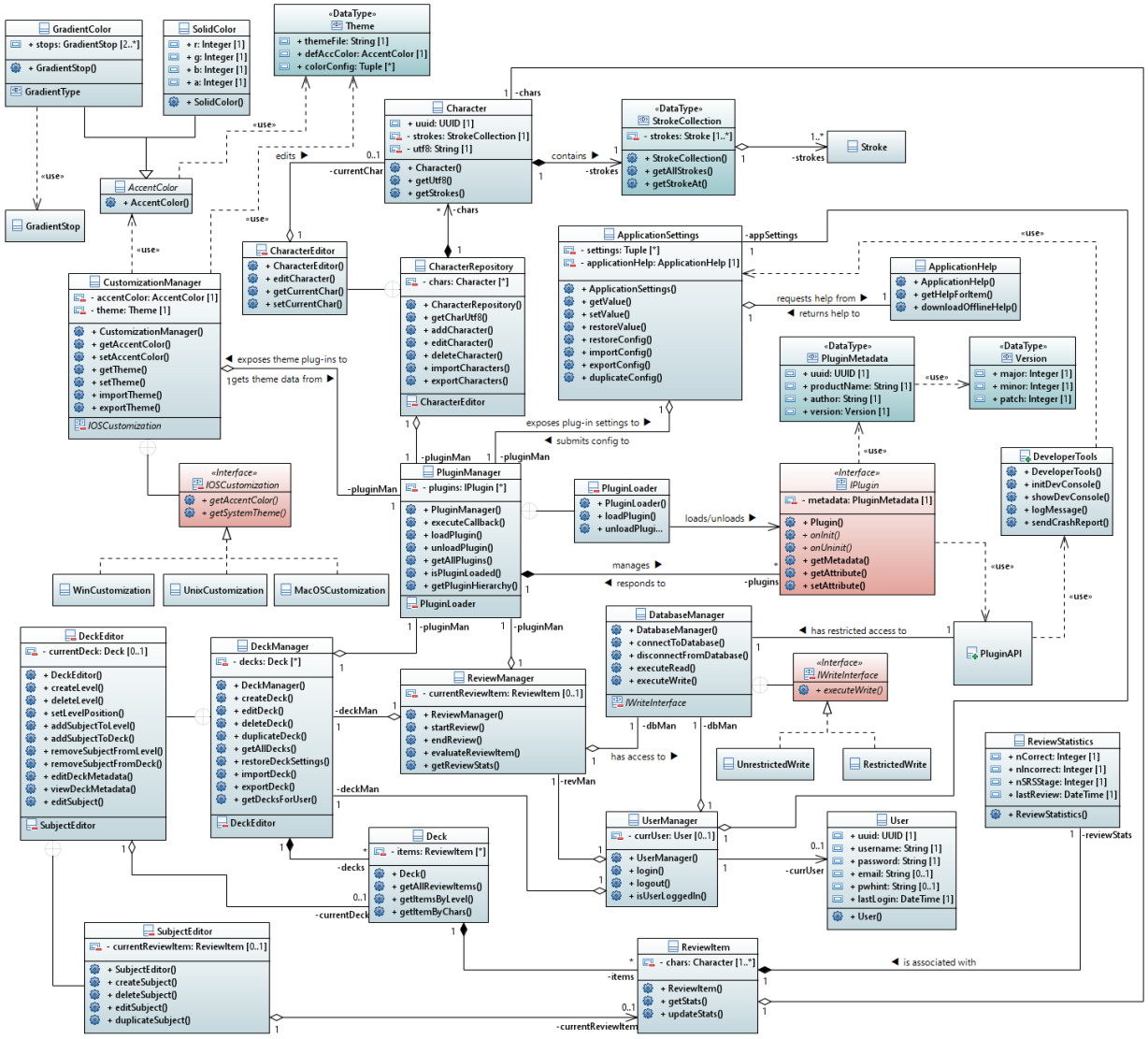

Figure 23: analysis class diagram of natsu's offline client

Appendix F: TBD

- mobile device requirements (display size, OS version, ...)
- NDF
- plug-in hooks