Bracket - Ein minimalistischer Sourcecode-Editor in Python

Projektbeschreibung

Bracket zielt darauf ab, ein moderner Code-Editor zu sein. Er gliedert sich in den typischen Workflow eines Entwicklers ein. Die Bearbeitung des Sourcecodes erfolgt innerhalb von Projektverzeichnissen. Diese können per Kommandozeile geöffnet werden mit:

```
bracket <VERZEICHNIS>
```

Der Editor erleichtert das Arbeiten mit größeren Projekten durch einen File-Tree. Er unterstützt zudem performantes Syntax-Highlighting mit dem De-Facto-Standard `treesitter`. Mehrere Dateien können gleichzeitig geöffnet werden und sind durch eine Tab-Ansicht erreichbar. Dateien können mit einer Menüleiste oder Tastenkürzel gespeichert, geöffnet und geschlossen werden.

Technische Grundlagen

GTK

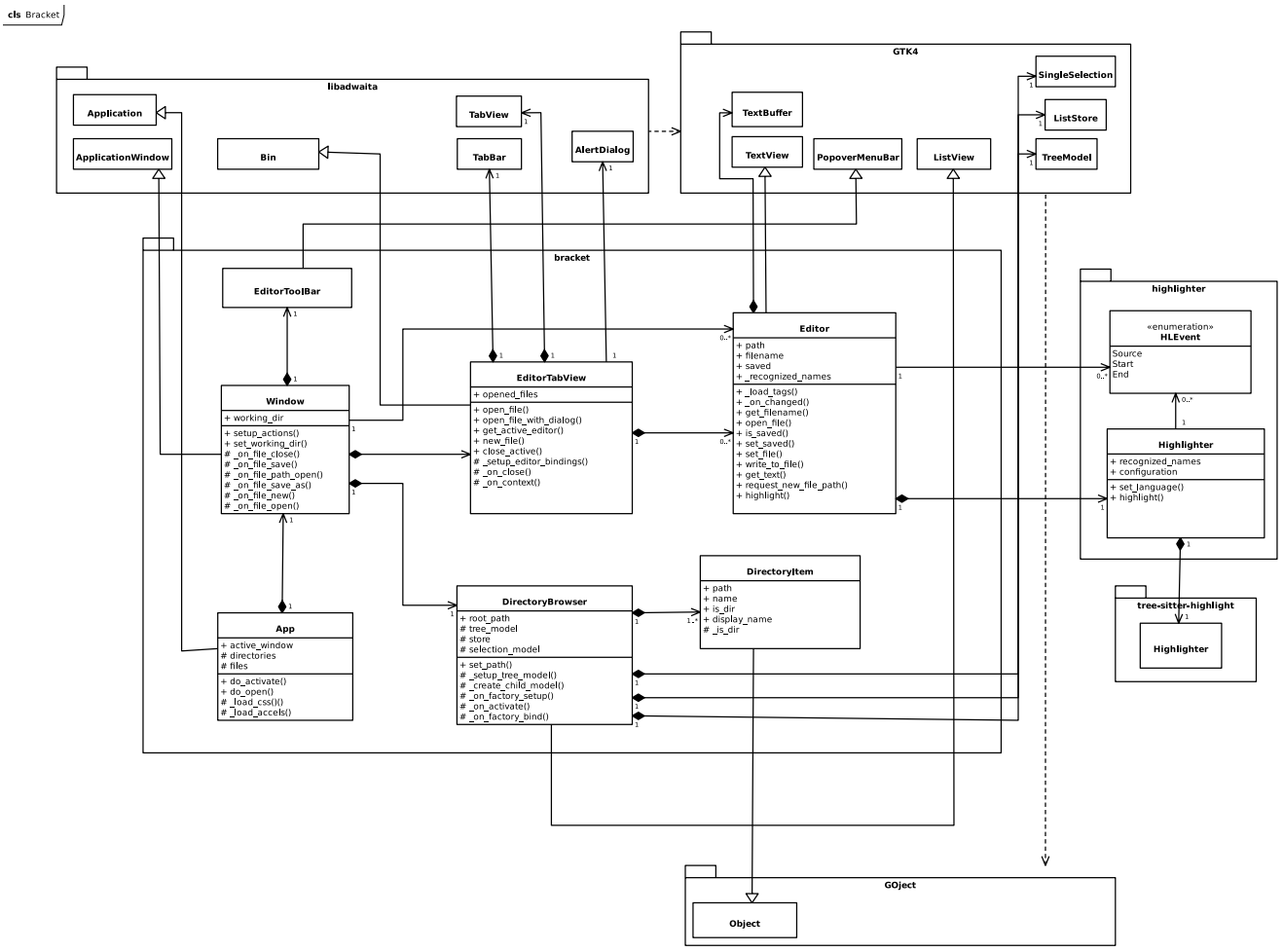
Die App basiert auf dem recht modernen GTK4-Grafiktoolkit und der Adwaita-Bibliothek für ein zeitgemäßes Erscheinungsbild. GTK4 bietet eine robuste Grundlage für Desktop-Anwendungen im Linux-Bereich mit nativer Performance. Adwaita ist dabei eine Sammlung moderner UI-Komponenten, die für eine bessere Benutzererfahrung sorgen und die Implementierung wesentlich erleichtern. GTK ist in C geschrieben. Objektorientierung und Vererbung laufen deshalb über das eigene GObject-System. Zur Vereinfachung von UI-Deklaration, der Anbindung von Event-Callbacks (z.B. Funktion die aufgerufen wird beim klicken eines Buttons) wird das Blueprint-Templating-System verwendet, entsprechende Dateien finden sich in seperaten `.blp`-Dateien

tree-sitter

Das Syntax-Highlighting wird durch ein in Rust geschriebenes Python-Modul realisiert, das die bewährte `tree-sitter`-Bibliothek

verwendet. Diese Kombination ermöglicht performante und präzise Syntaxanalyse in Echtzeit.

Funktionsweise



Anwendungsstart

Entry-Point der Anwendung ist die Datei `bracket.py.in`. Diese wird später vom Build-System im System installiert und per Kommandozeile aufgerufen. Sie lädt von der App benötigte Ressourcen wie z.B. UI-Definitionsdateien und CSS. Der Pfad zum separaten `highlighter`-Modul wird ebenfalls zu Pythons Pfaden für Abhängigkeiten hinzugefügt.

Anschließend wird eine `App`-Instanz erstellt, gestartet und eventuelle Kommandozeilenargumente übergeben.

Anwendungsarchitektur

Die `App`-Klasse erbt von der `Adw.Application` und ist somit in der Lage, mit dem GTK-Grafiktoolkit und dem Adwaita-Design-System zu interagieren. Sie fungiert als zentraler Koordinator der Anwendung.
Sie verwaltet

- **Ressourcen** - also das Laden von CSS-Stylesheets und UI-Definitionen aus den Resourcen
- **Tastenkürzel** - das Konfigurieren der Tastenbelegung auf basis einer `.json`-Datei
- **Fenster-Verwaltung**: Erstellung und Verwaltung von Editor-Fenstern
- **Kommandozeilenargumente**: Kommandozeilenargumente für zu öffnende Dateien und Verzeichnisse werden verarbeitet

Fensterstruktur

Jedes `Window` repräsentiert ein Anwendungsfenster und enthält die Hauptkomponenten:

EditorTabView

Die Klasse erbt von Adw.Bin (Container-Klasse - wie ein `<div>`). Sie verwaltet mehrere geöffnete Dateien in einer Tab-Ansicht mit den Widgets `Adw.TabBar` (Die Leiste mit den Dateien) und `Adw.TabView` (enthält Editoren). Sie ist auch Zustädnig für das Öffnen, Schließen und Wechseln zwischen Dateien und zeigt Dialoge für ungespeicherte Änderungen beim Schließen von Tabs via `Adw.AlertDialog`

DirectoryBrowser

Die Klasse implementiert einem FileTree mit GTKs TreeListModel. Sie ermöglicht Navigation durch das Projektverzeichnis, unterstützt das Öffnen von Dateien und zeigt Verzeichnisse, die etweitert und geschlossen werden können.

Das Erstellen der einzelnen Einträge erfolgt mit GTKs Factory-Model, ist also recht verbos. Vereinfacht werden die einzelnen Eintrage der Ordnerstruktur rekursiv bearbeitet und für jedes ein repräsentatives Widget erstellt.

Editor

Die Editor-Klasse fungiert als Kernkomponente für die Textbearbeitung, und basiert stark auf den vielfältigen Funktionen von `Gtk.TextView` und `Gtk.TextBuffer`. Diese regeln unter anderem die Einfärbung von Text, bearbeiten, Undo/Redo und das Rendern von Text.
Die Klasse selbert verwaltet den Dateizustand (gespeichert/ungespeichert), integriert das Rust-basierte Syntax-

Highlighting-Modul und behandelt Dateierstellung, -öffnung und -speicherung

Syntax-Highlighting-System

Das Highlighting-System besteht aus zwei Teilen:

Python-Seite

in den Datein `editor.py` und `themes.py`
Hier werden zunächst Farbthemen aus JSON-Dateien geladen und anschließend `GTK.TextTags` für verschiedene Syntaxelemente und deren Styling kreiert. Bei Änderungen am Text werden Highlighting-Events (Enthalten Styling-Informationen über bestimmte Text-Abschnitte) auf den Text an.

Rust-Modul

in `highlighter/`
Das externe Model implementiert performantes Syntaxanalyse mit tree-sitter und exportiert Python-Bindings über PyO3. Es generiert Highlighting-Events (Start, Source, End) für die Python-Verarbeitung.

Kommunikation

Bracket verwendet GTK's Aktions-Framework für Menüs und Tastenkürzel.
Per Window-Actions (z.B. `win.file-new`) werden Dateioperationen (Neu, Öffnen, Speichern, Schließen) zum aktiven Fenster kommuniziert. Hier werden diese dan entsprechend bearbeitet.
Tastenkürzel für die Actions werden per `keymap.json` geladen und mit Regex validiert.

Dialoge

Für die bessere Interaktion mit den Benutzer benutzt Bracket Dialoge für:

- **Datei-Dialoge** - Öffnen und Speichern von Dateien mit dem nativen Filemanager
- **Bestätigungs-Dialoge** - Warnung vor dem Verlust ungespeicherter Änderungen, via `Adw.AlertDialog`
  Die Dialoge werden asynchron mit Callbacks verknüpft. Dadurch wird ein eventuelles einfrieren der UI verhindert.

Ablauf

Der ungefähre Programmablauf sieht wie folgt aus:

1. **Initialisierung**: App lädt Ressourcen und erstellt Hauptfenster
2. **Dateizugriff**: DirectoryBrowser oder Datei-Dialoge triggern Dateiereignisse wie z.B. das Öffnen
3. **Editor-Erstellung**: Neue Editor-Instanzen werden in TabView integriert
4. **Text-Verarbeitung**: Änderungen lösen Syntax-Highlighting über Rust-Modul aus
5. **Speicherung**: Dateioperationen werden über Aktions-System koordiniert

Reflexion

Das erstellen des Text-Editors war insgesamt sehr interessant da es mehrere Technologien vereint hat (mehrsprachige Projekte, Buildsystem, Grafikoberfläche). Eben dies hat die Entwicklung aber auch schwieriger gestaltet als erwartet. Insbesondere das Setup der Entwicklungsumgebung und der groben Projektstruktur war komplex.

Das verwendete Grafiktoolkit stellte sich als sehr potent und durchdacht heraus, was die Entwicklung wesentlich vereinfacht hat. Auch Pythons dynamische Natur hat die Entwicklung ebenfalls beschleunigt. Der Trade-Off für diese war der relativ komplexe Packaging-Prozess.

Der erstellte Code ist in sich sauber und funktionsfähig, für eine Fortsetzung der App wären jedoch Erweiterungen und Änderung an einigen Stellen sinnvoll:

- Support anderer Programmiersprachen als Python
- Erkennung der Programmiersprache am Dateiname
- Integration mit dem Language-Server-Protocol für Code-Completion und Fehlererkennung
- erweiterte Konfiguration über Dateien
- Suchen und Ersetzen
- ein verbessertes Design

Sollten diese Funktionen implementiert sein, stünde höchstwahrscheinlich ein größeres Refactoring an.

Quellen

- **GTK-Dokumentationen**
  - https://docs.gtk.org/gtk4/
  - https://docs.gtk.org/Pango/
  - https://docs.gtk.org/glib/
  - https://docs.gtk.org/gobject/
  - https://docs.gtk.org/gio/
  - https://gtk-rs.org/
- **GTK-Python-Bindings**
  - https://pygobject.gnome.org/
- **Adwaita-Dokumentation**
  - https://gnome.pages.gitlab.gnome.org/libadwaita/doc/1.4/index.html
- **Templating-System-Dokumentation**
  - https://gnome.pages.gitlab.gnome.org/blueprint-compiler/index.html
- **PYO3-Dokumentationen**
  - https://pyo3.rs/v0.25.0/
  - https://www.maturin.rs/
- **Tree-Sitter**
  - https://tree-sitter.github.io/tree-sitter/3-syntax-highlighting.html
  - https://github.com/tree-sitter/tree-sitter/tree/master/highlighter
- **StackOverflow**
  - https://stackoverflow.com/questions/52506854/how-do-you-create-a-project-directory-tree-view-and-tree-store-in-gtk
- **Rust**
  - https://doc.rust-lang.org/stable/rust-by-example/index.html
  - https://doc.rust-lang.org/stable/std/index.html