# Extra – A simple expense tracker

# Technische Dokumentation

von Oguzhan Karaca, Marzieh Rayatzadeh und Timo Röhle

# Inhaltsverzeichnis

## Ordnerstruktur

Dieser Abschnitt beschreibt die Ordnerstruktur der Extra-App

```
assets
```

### Beschreibung

Der **assets** Ordner ist für Dateien gedacht, die nicht direkt zum Programm gehören aber vom Programm benutzt werden.

### Dateien

**schema_definitions.sql**
Script um die Tabellen der Datenbank anzulegen.

**default_categories.sql**
Script zum Einfügen der Standardkategorien in die Datenbank.

**default_payment_methods.sql**
Script zum Einfügen der Standardbezahlmethoden in die Datenbank.

**demo_data.sql**
Ein Datensatz fiktiver Ausgaben zu Demonstrationszwecken.

## config

### Beschreibung

Der config Ordner ist für Konfigurationsdateien vorgesehen.

### Dateien

**config.json**
Wird im Augenblick dazu genutzt das zuletzt benutzte „Profil“ zu speichern.
In der Zukunft können weitere Konfigurationseinstellungen in dieser Datei gespeichert werden.

## modules

Hier sind Module unseres Programmes abgelegt.

### Dateien

**db.py**
Enthält die DBController Klasse und ist für die Kommunikation mit den Datenbanken zuständig

**models.py**
Enthält die Datenmodelle Expense, Category und Payment_method

**plotter.py**
Enthält die Extra_plotter Klasse welche Matplotlib benutzt um Diagramme aus Datensätzen zu erzeugen.

**api.py**
Enthält die Extra_api Klasse und ist die Schnittstelle zwischen Frontend und Backend.

## databases

Der Ordner databases enthält die Datenbanken. Jedes neu angelegte Profil erhält eine eigene SQLite Datenbank die im Ordner databases gespeichert wird.

### Dateien

**extra.db**
Datenbank mit vor angelegten Kategorien und Bezahlmethoden um dem Benutzer einen komfortablen Einstieg zu ermöglichen.

**demo.db**
Eine Datenbank mit etwa 250 fiktiven Einträgen zu Demonstrationszwecken.

## docs

Ordner für die Technische- und Benutzerdokumentation und alles was dazu gehört (Diagramme, Bilder etc.)

### Unterordner

**docs/draft**
**docs/documentation**

## docs/draft

Ordner für Dokumente der Planungsphase.

### Dateien

**Platzhalter**

## docs/documentation

Ordner für die Dokumentation

### Dateien

**Platzhalter**

## ui

Ordner für das User-Interface

### Dateien

**extra_pyside.py**
Enthält die GUI von Extra. Erstellt mit Pyside6

## exports

Dient als Zielordner für die Datenexporte der Extra-App

### Dateien

## Datenbank

Dieser Abschnitt beschreibt das eingesetzte DBMS und den Aufbau der Datenbank.

### DBMS

SQLite dient als DBMS.

### Diagramm

## Klassen

Dieser Abschnitt beschreibt die Klassen samt ihrer Attribute, Methoden und Abhängigkeiten.

### Category

#### Modul

modules/models.py

#### Beschreibung

Modell für unsere Ausgabenkategorien

#### Abhängigkeiten

json

#### Argumente

**id [type: int, default: None]**
**name [type: string, default: None]**
**parent_category [type: int, default: None]**

#### Attribute

**id: int**
id der Kategorie
**name: string**
name der Kategorie
**parent_category: int**
id der zugehörigen Hauptkategorie

## Methoden

### get_id(self) -> int:
**Argumente:** Keine
**Beschreibung:** Gibt die id der Kategorie zurück.
**Returns:** id: int

### get_parent_category(self) -> int:
**Argumente:** Keine
**Beschreibung:** Gibt id der Übergeordneten Kategorie zurück
**Returns:** id: int

### get_name(self) -> str:
**Argumente:** Keine
**Beschreibung:** Gibt den Inhalt des Namensattributes zurück.
**Returns:** name: str

### set_id(self, id: int) -> bool:
**Argumente:** id: int
**Beschreibung:** Setzt das id Attribut auf den übergebenen Wert.
**Returns:** bool

### json_import(self, json_str: str) -> None:
**Argumente:** json_str: str
**Beschreibung:** Liest einen JSON-String ein und setzt die Übergebenen Attribute
**Returns:** -

### json_export(self) -> str:
**Argumente:** Keine
**Beschreibung:** Exportiert alle Attribute als JSON-String
**Returns:** JSON encoded dict {id: int, name: str, parent_category: int}

### def __str__(self) -> str:
**Argumente:** Keine
**Beschreibung:** Gibt einen formatierten String zurück, der das erstellte Objekt beschreibt.
**Returns:** str

# Payment_Method

## Modul

modules/models.py

## Beschreibung

Modell für unsere Bezahlmethoden.

## Abhängigkeiten

json

## Argumente

id [type: int, default: None]
name [type: string, default: None]

## Attribute

**id: int**
id der Bezahlmethode.
**name: string**
Bezeichnung der Bezahlmethode.

## Methoden

### get_id(self) -> int:
**Argumente:** Keine
**Beschreibung:** Gibt die id der Bezahlmethode zurück.
**Returns:** id: int

### get_name(self) -> str:
**Argumente:** Keine
**Beschreibung:** Gibt den Inhalt des Namensattributes zurück.
**Returns:** name: str

### set_id(self, id: int) -> bool:
**Argumente:** id: int
**Beschreibung:** Setzt das id Attribut auf den übergebenen Wert.
**Returns:** bool

### json_import(self, json_str: str) -> None:
**Argumente:** json_str: str
**Beschreibung:** Liest einen JSON-String ein und setzt die Übergebenen Attribute
**Returns:** -

### json_export(self) -> str:

**Argumente:** Keine
**Beschreibung:** Exportiert alle Attribute als JSON-String
**Returns:** JSON encoded dict {id: int, name: str}

**def __str__(self) -> str:**
**Argumente:** Keine
**Beschreibung:** Gibt einen formatierten String zurück, der das erstellte Objekt beschreibt.
**Returns:** str

## Expense

### Modul

modules/models.py

### Beschreibung

Modell für unsere Ausgaben

### Abhängigkeiten

json

### Argumente

**id [type: int, default: None]**
Die ID der Ausgabe
**date: [type: string, default: *empty string*]**
Das Datum der Ausgabe
**note [type: string, default: *empty string*]**
Optionale Notiz zu der Ausgabe
**category_id [type: int, default: None]**
ID der Ausgabenkategorie
**payment_method_id [type: int, default: None]**
ID der Bezahlmethode
**recurring: [type: str, default: None]**
Kann die Werte None, monthly or yearly haben um Ausgaben automatisch anzulegen die sich regelmäßig wiederholen. Zurzeit noch nicht implementiert

### Attribute

**- _id: int**
Die ID der Ausgabe
**- _date: string**

Das Datum der Ausgabe
**- _note: string**
Optionale Notiz zu der Ausgabe
**- _category: int**
ID der Ausgabenkategorie
**- _ payment_method: int**
ID der Bezahlmethode
**- _ recurring: string**
Kann die Werte None, monthly or yearly haben um Ausgaben automatisch anzulegen die sich regelmäßig wiederholen. Zurzeit noch nicht implementiert

---

## Methoden

### get_id(self) -> int:
**Argumente:** Keine
**Beschreibung:** Gibt die id der Bezahlmethode zurück.
**Returns:** id: int

### get_date(self) -> str:
**Argumente:** Keine
**Beschreibung:** Gibt den Inhalt des date Attributes zurück.
**Returns:** _date: str

### set_id(self, id: int) -> bool:
**Argumente:** id: int
**Beschreibung:** Setzt das _id Attribut auf den übergebenen Wert.
**Returns:** bool

### json_import(self, json_str: str) -> None:
**Argumente:** json_str: str
**Beschreibung:** Liest einen JSON-String ein und setzt die Übergebenen Attribute
**Returns:** -

### json_export(self) -> str:
**Argumente:** Keine
**Beschreibung:** Exportiert alle Attribute als JSON-String
**Returns:** JSON encoded dict {

```
id: int,
amount: float,
date: string,
category: {
    id: int,
    name: string,
    parent_category: int
},
payment_method: {
        id: int,
        name: string},
 note: string,
}
```

**dict_export(self) -> dict:**
**Argumente:** Keine
**Beschreibung:** Exportiert alle Attribute als Dict
**Returns:** export_dict = {
id: int,
amount: float,
date: string,
category: {
id: int,
name: string,
parent_category: int
},
payment_method: {
id: int,
name: string},
note: string,
}

**__str__(self) -> str:**
**Argumente:** Keine
**Beschreibung:** Gibt einen formatierten String zurück, der das erstellte Objekt beschreibt.
**Returns:** str

## DBController

### Modul

modules/db.py

### Beschreibung

Die Klasse DBController benutzt das sqlite3 Modul um mit den Datenbanken zu kommunizieren.

### Abhängigkeiten

sqlite3
json
modules/models.py
assets/default_payment_methods.sql
assets/default_categories.sql
assets/schema_definition.sql

## Argumente

**db_name: str**
Benötigt den Pfad zur derzeitigen benutzten Datenbank. Default: databases/extra.db

## Attribute

**db_name: str**
Enthält den Pfad der derzeitig genutzten Datenbank

## Methoden

### payment_method_export(self) -> list[tuple]:
**Argumente:** -
**Beschreibung:** Gibt eine Liste von Tupeln aller Einträge der payment_method Tabelle zurück.
**Returns:** list[tuple]

### category_export(self) -> list[tuple]:
**Argumente:** -
**Beschreibung:** Gibt eine Liste von Tupeln aller Einträge der category Tabelle zurück.
**Returns:** list[tuple]

### expense_export(self) -> list[tuple]:
**Argumente:** -
**Beschreibung:** Gibt eine Liste von Tupeln aller Einträge der expense Tabelle zurück.
**Returns:** list[tuple]

### db_export(self) -> dict{str, list[tuple]}:
**Argumente:** -
**Beschreibung:** Gibt ein Dict mit dem Datenbankpfad und Listen von Tupeln aller Einträge in der DB zurück.
**Returns:** export_dict = {
db_name: str,
expense_table: list[tuple],
category_table: list[tuple],
payment_method_table: list[tuple]
}

### category_expense(self, options: dict) -> list[tuple]:
**Argumente:** options: dict[str]
options = {year: str}
**Beschreibung:**
Gibt eine Liste von Tupeln zurück. Jeder Tuple besteht aus dem Kategorienamen und der Summe aller Ausgaben in der Kategorie.
**Returns:** list[tuple]

### profile_details(self) -> dict[str, float]:
**Argumente:** -
**Beschreibung:** Gibt ein Dict mit Informationen über die Datenbank zurück.

Diese Informationen beinhalten: Datenbankpfad, Ältester Eintrag, Neuster Eintrag, Anzahl der Einträge in der expense Tabelle, Summe aller Ausgaben und die durchschnittliche Ausgabe.
**Returns:** {
    db: str,
    expense_sum: float,
    expense_avg: float,
    expense_count: int
    oldest_entry: str,
    newest_entry: str
}

**yearly_expense(self, order: dict[str, bool]) -> list[tuple]:**
**Argumente:** order: {field: str, asc: bool}
**Beschreibung:** Gibt eine Liste von Tupeln mit der Gesamtsumme aller Ausgaben pro Jahr zurück.
**Returns:** list[tuple]

**monthly_expense(self, options: dict[str, dict]) -> list[tuple]:**
**Argumente:** options: {year: str, order: {field: str, asc: bool}}
**Beschreibung:** Gibt eine Liste von Tuplen mit der Gesamtsumme aller Ausgaben jeden Monats für ein bestimmtes Jahr zurück.

**switch_db(self, path: str) -> str:**
**Argumente:** path: str
**Beschreibung:** Wechselt auf die DB welche vom path Argument übergeben wird.
**Returns:** path: str

**create_new(self, path: str) -> bool:**
**Argumente:** path: str
**Beschreibung:** Legt eine neue Datenbank mit den default payment_method und category Tabellen an. Gibt bei Erfolg True zurück.
**Returns:** bool

**def expense_filter(self, filter: dict[str, list]) -> list[Expense]:**
**Argumente:** filter {
    start_date: str,
    end_date: str,
    min_amount: float,
    max_amount: float,
    category_list: list[int],
    payment_method_list: list[int],
    order: {field: str, asc: bool}
}
**Beschreibung:** Gibt eine Liste von Expense Objekten zurück die den Filterkriterien entsprechen, welche im filter Argument übergeben wurden.
**Returns:** list[Expense]

**insert_expense(self, expense: Expense) -> Expense:**
**Argumente:** expense: Expense
**Beschreibung:** Erstellt einen neuen Ausgabeneintrag in der Datenbank und gibt diesen zurück.

**Returns:** expense: Expense

**update_expense(self, expense: Expense) -> bool:**
**Argumente:** expense: Expense
**Beschreibung:** Bearbeitet eine Ausgabe. Gibt bei Erfolg True zurück
**Returns:** bool

**delete_expense(self, expense: Expense) -> Expense:**
**Argumente:** expense: Expense
**Beschreibung:** Löscht die gewünschte Ausgabe aus der Datenbank.
Gibt die gelöschte Ausgabe zurück.
**Returns:** expense: Expense

**insert_payment_method(self, payment_method: Payment_method) -> Payment_method:**
**Argumente:** payment_method: Payment_method
**Beschreibung:** Erstellt eine neue Bezahlmethodeneintrag in der Datenbank
**Returns:** payment_method: Payment_method

**payment_method_list(self) -> list[Payment_method]:**
**Argumente:** -
**Beschreibung:** Gibt eine Liste aller Bezahlmethoden zurück.
**Returns:** list[Payment_method]

**update_payment_method(self, payment_method: Payment_method) -> bool:**
**Argumente:** payment_method: Payment_method
**Beschreibung:** Bearbeitet eine Bezahlmethode. Gibt bei Erfolg True zurück
**Returns:** bool

**delete_payment_method(self, payment_method: Payment_method) -> Payment_method:**
**Argumente:** payment_method: Payment_method
**Beschreibung:** Löscht die gewünschte Bezahlmethode aus der Datenbank.
Gibt die gelöschte Bezahlmethode zurück.
**Returns:** payment_method: Payment_method

**insert_category(self, category: Category) -> Category:**
**Argumente:** category: Category
**Beschreibung:** Erstellt eine neue Kategorie in der Datenbank und gibt diese zurück.
**Returns:** category: Category

**find_parent_categories(self) -> list[Category]:**
**Argumente:** -
**Beschreibung:** Gibt eine Liste aller Hauptkategorien zurück.
**Returns:** list[Category]

**find_subcategories(self, parent_category_id: int) -> list[Category]:**
**Argumente:** parent_category_id: int
**Beschreibung:** Gibt eine Liste aller Kategorien einer Bestimmten Oberkategorie zurück.
**Returns:** list[Category]

# Extra_api

## Modul

modules/api.py

## Beschreibung

Extra_api ist die Schnittstelle zwischen UI und Backend. Alle Anfragen an die Extra_api müssen in JSON-Strings sein.

## Abhängigkeiten

json
csv
datetime
pathlib

modules/models
modules/db import DBController
modules/plotter import Extra_plotter

## Argumente

Keine

## Methoden

### create_db(self, data:str) -> str:
**Argumente:** Erwartet ein JSON Encoded dict {name: str}
**Beschreibung:** Erstellt eine neue Datenbank im database Ordner mit default Kategorien und Bezahlmethoden
**Returns:** JSON Encoded dict {success: str, error: str}

### switch_db(self, name: str) -> str:
**Argumente:** JSON dict {name: str}
**Beschreibung:** Wechselt auf die Ausgewählte Datenbank. Datenbank muss im database Ordner existieren.
**Returns:** JSON encoded True/False

### delete_db(self, db_name: str) -> str:
**Argumente:** JSON dict {name: str}
**Beschreibung:** Löscht die Ausgewählte Datenbank aus dem database Ordner
**Returns:** JSON Encoded bool

### load_config(self) -> str:

**Argumente:** Keine
**Beschreibung:** Lädt den inhalt der Konfigurationsdatei config/config.json
**Returns:** config.json: str

**save_config(self, config: str) -> str:**
**Argumente:** JSON encoded dict
**Beschreibung:** Überschreibt die bestehende config/config.json mit der übergebenen config.
**Returns:** JSON encoded bool

**profile_info(self) -> str:**
**Argumente:** Keine
**Beschreibung:** Gibt eine Liste aller verfügbaren Profile (Datenbanken) und das zurzeit benutzte Profil zurück
**Returns:** JSON dict {active: str, profile_list: list}

**profile_details(self) -> str:**
**Argumente:** Keine
**Beschreibung:** Gibt ein JSON dict mit Informationen über das derzeit genutzte Profil zurück.
**Returns:** JSON dict {
    db: str,
    expense_sum: float,
    expense_avg: float,
    expense_count: int
    oldest_entry: str,
    newest_entry: str
}

**insert_expense(self, data: str) -> str:**
**Argumente:** data muss ein JSON Encoded dict sein
**Beschreibung:** Fügt der Datenbank eine neue Ausgabe hinzu.
**Returns:** JSON encoded dict der eingefügten Ausgabe

**expense_filter(self, data:str) -> str:**
**Argumente:** data: JSON encoded filter **{**
    start_date: str,
    end_date: str,
    min_amount: float,
    max_amount: float,
    category_list: list[int],
    payment_method_list: list[int],
    order: {field: str, asc: bool}
**}**
**Beschreibung:** Gibt eine Liste mit Ausgaben zurück, die den Filterkriterien entsprechen
**Returns:** JSON encoded list[dict]

**update_expense(self, data: str) -> str:**
**Argumente:** data: JSON dict
**Beschreibung:** Updated die gewünschte Ausgabe
**Returns:** JSON encoded bool

**delete_expense(self, data: str) -> str:**
**Argumente:** data: JSON encoded dict
**Beschreibung:** Löscht die gewünschte Ausgabe permanent aus der DB
**Returns:** Gibt die entfernte Ausgabe im JSON Format zurück

**insert_category(self, data:str) -> str:**
**Argumente:** data: json encoded dict
**Beschreibung:** Legt eine neue Kategorie in der Datenbank an.
**Returns:** JSON encoded dict der eingefügten Kategorie

**parent_category_list(self) -> str:**
**Argumente:** Keine
**Beschreibung:** Gibt eine Liste aller Hauptkategorien im JSON Format zurück.
**Returns:** JSON encoded list

**subcategory_list(self, data:str) -> str:**
**Argumente: data: JSON encoded dict der Hauptkategorie**
**Beschreibung:** Gibt eine Liste mit allen Subkategorien der übergeben Hauptkategorie zurück
**Returns:** JSON encoded list

**update_category(self, data: str) -> str:**
**Argumente:**
**Beschreibung:** Updated eine Kategorie mit den übergebenen Werten.
**Returns:** JSON encoded bool

**delete_category(self, data:str) -> str:**
**Argumente:** JSON dict der Kategorie die man löschen möchte.
**Beschreibung:** Löscht die Ausgwählte Kategorie permanent aus der Datenbank. Kategorie darf keine Unterkategorien oder Ausgaben enthalten.
**Returns:** JSON encoded dict der gelöschten Kategorie

**insert_payment_method(self, data:str) -> str:**
**Argumente:** JSON dict der Bezahlmethode die man einfügen möchte.
**Beschreibung:** Legt eine neue Bezahlmethode in der Datenbank an
**Returns:** Gibt die angelgte Bezahlmethode als JSON dict zurück.

**payment_method_list(self) -> str:**
**Argumente:** Keine
**Beschreibung:** Gibt eine Liste aller Bezahlmethoden im JSON Format zurück
**Returns:** JSON encoded list

**update_payment_method(self, data:str) -> str:**
**Argumente:** JSON dict der Bezahlmethode, die man updaten möchte.
**Beschreibung:**
**Returns:** JSON encoded bool

**delete_payment_method(self, data:str) -> str:**
**Argumente:** JSON dict der Bezahlmethode, die man löschen möchte.
**Beschreibung:** Löscht eine Bezahlmethode permanent aus der Datenbank.
**Returns:** JSON dict der gelöschten Bezahlmethode

**plot_category_expense_png(self, data: str) -> bytes:**
**Argumente:** JSON dict {year: str}
**Beschreibung:** Gibt ein png mit einem Balkendiagramm der Summe aller Ausgaben pro Kategorie eines bestimmten Jahren zurück.
**Returns:** png in bytecode

**plot_monthly_expense_png(self, data: str) -> bytes:**
**Argumente:** JSON dict {year: str}
**Beschreibung:** Gibt ein png mit einem Balkendiagramm der Summe aller Ausgaben pro Kategorie eines bestimmten Jahren zurück.
**Returns:** png in bytecode

**yearly_expense(self, data: str) -> str:**
**Argumente:** JSON dict {year: str}
**Beschreibung:** Gibt die Summe aller Ausgaben eines Jahres zurück.
**Returns:** JSON encoded float

**monthly_expense(self, data: str) -> str:**
**Argumente:** JSON dict {year: str}
**Beschreibung:** Gibt die  Summe aller Ausgaben jeden Monats eines bestimmten Jahres zurück.
**Returns:** JSON list

**db_export_json(self) -> str:**
**Argumente:** Keine
**Beschreibung:** Exportiert alle Einträge der Datenbank im JSON Format und speichert diese im export Ordner
**Returns:** JSON encoded dict

**payment_method_export_csv(self) -> str:**
**Argumente:** Keine
**Beschreibung:** Legt eine neue Datei mit allen Bezahlmethoden im csv format im exports Ordner an.
**Returns:** JSON encoded bool

**category_export_csv(self) -> str:**
**Argumente:** Keine
**Beschreibung:** Legt eine neue Datei mit allen Kategorien im csv format im exports Ordner an.
**Returns:** JSON encoded bool

**expense_export_csv(self) -> str:**
**Argumente:** Keine
**Beschreibung:** Legt eine neue Datei mit allen Ausgaben im csv format im exports Ordner an.
**Returns:** JSON encoded bool

MainWindow(QMainWindow)

## Modul

ui/extra_pyside.py

## Beschreibung

Main Window ist das Hauptfenster der GUI. Hier wird das Hauptlayout festgelegt und die Kommunikation zwischen den unterschiedlichen GUI Elementen geregelt.

## Abhängigkeiten

sys
json
pathlib
datetime
pyside6
modules/api.py

## Argumente

Keine

## Attribute

**toolbar: QToolBar**
Menü am oberen Rand des Hauptfensters
**config_dict: dict**
Enthält Konfigurationen zum initialisieren der App
**expense_action: QAction**
Ausgaben Button in der Toolbar
**profile_action: QAction**
Profil Button in der Toolbar
**settings_action: QAction**
Einstellungs Button in der Toolbar
**stats_action: QAction**
Statistik Button in der Toolbar
**about_act: QAction**
Über Button in der Toolbar
**central: QWidget**
Zentrales Kontainerelment
**grid: QGridLayout**
Grid Layout des Hauptfensters
**left_stack: QStackedWidget**
Wird verwendet um die Sichtbarkeit der Menüs auf der linken Seite zu regeln.
**right_stack: QStackedWidget**
Wird verwendet um die Sichtbarkeit der Info Screens auf der rechten zu regeln.

**left_main_container: QWidget**
Linke Seite im Grid.
**left_main_layout: QGridLayout**
Layout für die Linke Seite des Grids.
**expense_insert_form: ExpenseInsertForm**
Verantwortlich für das Aussehen und die Funktion der Eingabemaske für neue Ausgaben.
**expense_update_form: ExpenseUpdateForm**
Verantwortlich für das Aussehen und die Funktion der Eingabemaske zum überarbeiten von Ausgaben.
**expense_filter_form: ExpenseFilterForm**
Verantwortlich für die Eingabemaske zum Filtern von Ausgaben.
**right_main_container: QWidget**
Kontainerelement für die Rechte Seite des Hauptfensters.
**right_main_layout: QGridLayout**
Layout für das Kontainerelement auf der rechten Seite.
**expense_table: ExpenseTable**
Tabelle zum Anzeigen der Ausgaben.
**left_profile_container: QWidget**
Kontainer für die Linke Seite des „Profile Screens“
**left_profile_layout: QGridLayout**
Layout für die linke Seite des „Profile Screens“
**profile_form: ProfileForm**
Verantwortlich für das Aussehen und die Funktion der Eingabemaske zum bearbeiten von Profilen.
**right_profile_container: QWidget**
Kontainer Element für die rechte Seite des „Profile Screens“
**right_profile_layout: QGridLayout**
Layout für die rechte Seite des „Profile Screens“
**profile_details: ProfileDetails**
Verantwortlich für die Darstellung der Profilübersicht.
**left_stats_container: QWidget**
Kontainer für die linke Seite des Statistik Screens
**left_stats_layout: QGridLayout**
Layout für die linke Seite des Statistik Screens
**stat_form: StatForm**
Verantwortlich für das Aussehen und die Funktion der Eingabemaske des Statistik Screens.
**right_stats_container: QWidget**
Kontainerelement für die rechte Seite des Statistik Screens
**right_stats_layout: QGridLayout**
Layout für die rechte Seite des Statistik Screens
**diagram_screen: DiagramScreen**
Hier werden die Diagramme dargestellt.
**left_settings_container: QWidget**
Linke Seite des Einstellungen Screens
**left_settings_layout: QGridLayout**
Layout für die linke Seite des Einstellungs Screens
**settings_form: SettingForm**
Verantwortlich für das Aussehen und die Funktion der Eingabemaske des „Einstellungs Screens“
**right_settings_container: Qwidget**

Kontainer Widget für die rechte Seite des Einstellungsfensters
**right_settings_layout: QgridLayout**
Layout für die rechte Seite des Einstellungs-Screens
**settings_screen: SettingsScreen**
Verantwortlich für das Aussehen und die Funktion der rechten Seite des Einstellungsfensters.

## Methoden

### draw_sum_per_category(self) -> bool:
**Argumente:** Keine
**Beschreibung:** Zeigt das PNG eines Balkendiagramms mit Ausgaben pro Kategorie des gewählten Jahres
**Returns:** bool

### draw_sum_per_month(self) -> bool:
**Argumente:** Keine
**Beschreibung:** Zeigt das PNG eines Balkendiagramms mit Ausgaben pro Monat des gewählten Jahres
**Returns:** bool

### about(self) -> None:
**Argumente:** Keine
**Beschreibung:** Öffnet eine Messagebox mit Informationen über die App.
**Returns:** -

### update_subcategory(self) -> bool:
**Argumente:** Keine
**Beschreibung:** Liest die Eingabemaske zum bearbeiten von Unterkategorien ein. Überprüft und evaluiert die eingegebenen Daten. Anschließend wird die Unterkategorie bearbeitet und informiert den Nutzer bei Erfolg oder Fehlschlag mit einer Messagebox. Bei Erfolg werden sämtliche Kategorie-Comboboxen aktualisiert.
**Returns:** bool

### update_category(self) -> bool:
**Argumente:** Keine
**Beschreibung:** Liest die Eingabemaske zum bearbeiten von Kategorien ein. Überprüft und evaluiert die eingegebenen Daten. Anschließend wird die Kategorie bearbeitet und informiert den Nutzer bei Erfolg oder Fehlschlag mit einer Messagebox. Bei Erfolg werden sämtliche Kategorie-Comboboxen aktualisiert.
**Returns:** bool

### update_payment_method(self) -> bool:
**Argumente:** Keine
**Beschreibung:** Liest die Eingabemaske zum bearbeiten von Bezahlmethoden ein. Überprüft und evaluiert die eingegebenen Daten. Anschließend wird die Bezahlmethode bearbeitet und informiert den Nutzer bei Erfolg oder Fehlschlag mit einer Messagebox. Bei Erfolg werden sämtliche Bezahlmethoden-Comboboxen aktualisiert.
**Returns:** bool

### delete_subcategory(self) -> bool:

**Argumente:** Keine
**Beschreibung:** Liest die ID der zu löschenden Unterkategorie aus der Combobox aus. Öffnet einen Bestätigungsdialog. Bei Bestätigung wird die Unterkategorie gelöscht und der Nutzer wird über Erfolg oder Fehlschlag mittels einer Messagebox informiert.
**Returns:** bool

### delete_category(self) -> bool:
**Argumente:** Keine
**Beschreibung:** Liest die ID der zu löschenden Kategorie aus der Combobox aus. Öffnet einen Bestätigungsdialog. Bei Bestätigung wird die Kategorie gelöscht und der Nutzer wird über Erfolg oder Fehlschlag mittels einer Messagebox informiert.
**Returns:** bool

### delete_payment_method(self) -> bool:
**Argumente:** Keine
**Beschreibung:** Liest die ID der zu löschenden Bezahlmethode aus der Combobox aus. Öffnet einen Bestätigungsdialog. Bei Bestätigung wird die Bezahlmethode gelöscht und der Nutzer wird über Erfolg oder Fehlschlag mittels einer Messagebox informiert.
**Returns:** bool

### create_payment_method(self) -> bool:
**Argumente:** Keine
**Beschreibung:** Liest die Daten der neuen Bezahlmethode aus der Eingabemaske aus, evaluiert die eingelesenen Daten. Sind die eingelesenen Daten ungültig wird der Nutzer informiert, ansonsten wird die neue Bezahlmethode angelegt und der Nutzer erhält ein Feedback via Messagebox.
**Returns:** bool

### create_subcategory(self) -> bool:
**Argumente:** Keine
**Beschreibung:** Liest die Daten der neuen Unterkategorie aus der Eingabemaske aus, evaluiert die eingelesenen Daten. Sind die eingelesenen Daten ungültig wird der Nutzer informiert, ansonsten wird die neue Unterkategorie angelegt und der Nutzer erhält ein Feedback via Messagebox.
**Returns:** bool

### create_category(self) -> bool:
**Argumente:** Keine
**Beschreibung:** Liest die Daten der neuen Kategorie aus der Eingabemaske aus, evaluiert die eingelesenen Daten. Sind die eingelesenen Daten ungültig wird der Nutzer informiert, ansonsten wird die neue Kategorie angelegt und der Nutzer erhält ein Feedback via Messagebox.
**Returns:** bool

### confirmation_dialog(self, title, text) -> bool:
**Argumente:** title: str, text: text
**Beschreibung:** Erzeugt eine Dialogbox mit Titel und Text. Wird für „sind sie sicher“ Dialoge verwendet
**Returns:** bool

**delete_profile(self) -> bool:**
**Argumente:** Keine
**Beschreibung:** Liest das zu löschende Profil aus der Combobox aus. Anschließend wird das Profil samt Datenbank gelöscht und aus der Combobox entfernt. Funktioniert nur wenn das zu löschende Profil nicht das zurzeit aktive Profil ist.
**Returns:** bool

**create_profile(self) -> bool:**
**Argumente:** Keine
**Beschreibung:** Liest den gewünschten Profilnamen aus der Eingabemaske aus. Ist der Name gültig wird ein neues Profil samt Datenbank angelegt.
**Returns:** bool

**disconnect_category_combo(self) -> None:**
**Argumente:** Keine
**Beschreibung:** Entfernt alle Eventlistener von Kategorie-Comboboxen. Wird benötigt um Comboboxen zu aktualisieren.
**Returns:** -

**init_comboboxes(self) -> None:**
**Argumente:** Keine
**Beschreibung:** Erstellt / Aktualisiert alle Comboboxen
**Returns:** -

**switch_profile(self) -> None:**
**Argumente:** Keine
**Beschreibung:** Wechselt das derzeitig aktive Profil auf das ausgewählte Profil.
**Returns:** -

**main_screen(self) -> None:**
**Argumente:** Keine
**Beschreibung:** Wechselt den Display Stack zum Hauptfenster
**Returns:** -

**profile_screen(self) -> None:**
**Argumente:** Keine
**Beschreibung:** Wechselt den Display Stack zum Profilfenster
**Returns:** -

**stat_screen(self) -> None:**
**Argumente:** Keine
**Beschreibung:** Wechselt den Display Stack zum Statistikfenster
**Returns:** -

**setting_screen(self) -> None:**
**Argumente:** Keine
**Beschreibung:** Wechselt den Display Stack zum Einstellungsfenster
**Returns:** Platzhalter

**abort_expense_update(self)  -> None:**
**Argumente:** Keine

**Beschreibung:** Wechselt die Ausgabe bearbeiten Maske zur Ausgabe erstellen Maske.
**Returns:** -

**get_expense_list(self) -> None:**
**Argumente:** Keine
**Beschreibung:** Liest die Maske zum filtern von Ausgaben aus, leitet die Anfrage an das Backend weiter und erstellt aus der Antwort die Ausgabentabelle.
**Returns:** -

**click_expense_table_cell(self, row, column) -> None:**
**Argumente:** row: int, column: int
**Beschreibung:** Wertet aus welche Zelle in der Ausgabentabelle angeklickt wurde und verarbeitet die Anfrage.
**Returns:** -

**init_update_inputs(self, data) -> None:**
**Argumente:** data: dict
**Beschreibung:** Füllt die Eingabemaske zum bearbeiten von Ausgaben mit den Werten der zu bearbeitenden Ausgabe.
**Returns:** -

**update_expense(self) -> None:**
**Argumente:** Keine
**Beschreibung:** Wertet die Eingabemaske zum bearbeiten von Ausgaben aus. Bei gültigen Eingaben wird die Anfrage zum bearbeiten ans Backend gesendet. Der Nutzer wird per Messagebox informiert, wenn seine Eingaben nicht gültig sind oder wenn die Ausgabe erfolgreich bearbeitet wurde.
**Returns:** -

## StatForm(QWidget)

### Modul

ui.py

### Beschreibung

Zuständig für Positionierung und Aussehen der Eingabemaske zum Erstellen von Diagrammen.

### Abhängigkeiten

sys
json
pathlib
datetime
pyside6
modules/api.py

### Argumente

Keine

### Attribute

**grid: QgridLayout**
Layout der Eingabemaske für Statistiken
**self.year_input: QlineEdit**
Eingabefeld für das gewünschte Jahr.
**validator: QIntValidator**
Stellt sicher, dass ein gültiges Jahr eingegeben wurde.
**btn_sum_per_month: QPushButton**
Button zum erstellen des Monatsdiagramms
**btn_sum_per_category: QPushButton**
Button zum erstellen des Kategoriendiagramms

### Methoden

**plot_category_expense_bars(self) -> bytes:**
**Argumente**: Keine
**Beschreibung**: Liest die Eingabe aus dem Eingabefeld für das Jahr aus. Sendet die Anfrage an das Backend und gibt des erhaltene PNG zurück
**Returns**: bytes

**plot_monthly_expense_bars(self) -> bytes:**
**Argumente**: Keine
**Beschreibung**: Liest die Eingabe aus dem Eingabefeld für das Jahr aus. Sendet die Anfrage an das Backend und gibt des erhaltene PNG zurück
**Returns**: bytes

## SettingForm(QWidget)

### Modul

ui.py

### Beschreibung

Kümmert sich um die Darstellung, Anordnung und Funktion der Einstellungsoptionen.

### Abhängigkeiten

sys
json
pathlib
datetime
pyside6
modules/api.py

## Argumente

setting_screen: SettingsScreen

## Attribute

**self.grid: QgridLayout**
Layout für die Einstellungseingabemaske.
**self.settings_menu_container: Qwidget**
Kontainerelement für das Einstellungsmenü.
**self.settings_menu_layout: QgridLayout**
Layout für das Einstellungsmenü.
**self.btn_category_settings: QPushButton**
Button um die Einstellungsoptionen für Kategorien zu öffnen.
**self.btn_payment_settings: QPushButton**
Button um die Einstellungsoptionen für Bezahlmethoden zu öffnen.
**self.btn_export_settings: QPushButton**
Button um die Exportoptionen zu öffnen.
**self.setting_stack: QstackedWidget**
Stack für die Optionsmenüs
**self.category_settings_container: Qwidget**
Kontainer für die Kategorieoptionen
**self.category_settings_layout: QgridLayout**
Layout für den Kontainer für die Kategorieoptionen
**self.combo_category: QcomboBox**
Dropdownmenü für Kategorien
**self.category_name_input: QlineEdit**
Eingabefeld für den Kategorienamen
**self.btn_create_category: QPushButton**
Button zum anlegen von Kategorien
**self.btn_update_category: QPushButton**
Button zum bearbeiten von Kategorien
**self.btn_delete_category: QPushButton**
Button zum löschen von Kategorien
**self.combo_subcategory: QcomboBox**
Dropdownmenü für Unterkategorien
**self.subcategory_name_input: QlineEdit**
Eingabefeld für den Unterkategorienamen
**self.btn_create_subcategory: QPushButton**
Button zum erstellen von Unterkategorien
**self.btn_update_subcategory: QPushButton**
Button zum bearbeiten von Unterkategorien.
**self.btn_delete_subcategory: QPushButton**

Button zum löschen von Unterkategorien.
**self.payment_settings_container: Qwidget**
Kontainerelement für die Bezahlmethodenoptionen.
**self.payment_settings_layout: QgridLayout**
Layout für das Kontainerelement der Bezahlmethodenoptionen.
**self.combo_payment: QComboBox**
Dropdownmenü für Bezahlmethoden
**self.payment_name_input: QLineEdit**
Eingabefeld für die Bezeichnung von Bezahlmethoden.
**self.btn_create_payment: QPushButton**
Button zum erstellen von Bezahlmethoden.
**self.btn_update_payment: QPushButton**
Button zum bearbeiten von Bezahlmethoden.
**self.btn_delete_payment: QPushButton**
Button zum löschen von Bezahlmethoden.
**self.export_settings_container: QWidget**
Kontainerelement für Exportoptionen.
**self.export_settings_layout: QGridLayout**
Layout für das Kontainerelement der Exportoptionen.
**self.btn_export_db_json: QPushButton**
Button zum exportieren der kompletten Datenbank im JSON Format.
**self.btn_export_exp_csv: QPushButton**
Button zum exportieren der Ausgabentabelle im CSV Format.
**self.btn_export_cat_csv: QPushButton**
Button zum exportieren der Kategorientabelle im CSV Format.
**self.btn_export_pay_csv: QPushButton**
Button zum exportieren der Bezahlmethodentabelle im CSV Format.

## Methoden

### change_parent_category(self, index) -> None:
**Argumente**: index: int
**Beschreibung**: Ändert den Inhalt des Dropdownmenüs für Unterkategorie wenn eine andere Hauptkategorie ausgewählt wird.
**Returns**: -

### show_category_settings(self) -> None:
**Argumente**: Keine
**Beschreibung**: Zeigt das Einstellungsmenü für Kategorien an.
**Returns**: -

### show_payment_settings(self) -> None:
**Argumente**: Keine
**Beschreibung**: Zeigt das Einstellungsmenü für Bezahlmethoden an.
**Returns**: -

### show_export_settings(self) -> None:
**Argumente**: Keine
**Beschreibung**: Zeigt das Menü für Exportoptionen an.
**Returns**: -

**db_export_json(self) -> bool:**
**Argumente**: Keine
**Beschreibung**: Exportiert den Inhalt der gesamten Datenbank im JSON Format und speichert die Datei im *export* Ordner.
**Returns**: bool

**exp_export_csv(self) -> bool:**
**Argumente**: Keine
**Beschreibung**: Exportiert den Inhalt der Tabelle expense im CSV Format und speichert die Datei im *export* Ordner.
**Returns**: bool

**cat_export_csv(self) -> bool:**
**Argumente**: Keine
**Beschreibung**: Exportiert den Inhalt der Tabelle category im CSV Format und speichert die Datei im *export* Ordner.
**Returns**: -

**pay_export_csv(self) -> bool:**
**Argumente**: Keine
**Beschreibung**: Exportiert den Inhalt der Tabelle payment_method im CSV Format und speichert die Datei im *export* Ordner.
**Returns**: -

## SettingsScreen(QWidget)

### Modul

ui.py

### Beschreibung

Verwantwortlich für die Darstellung von Hilfstexten zu den gewünschten Einstellungsmenüs.

### Abhängigkeiten

sys
json
pathlib
datetime
pyside6
modules/api.py

### Argumente

## Attribute

**self.grid: QGridLayout**
Layout für den "Settingsscreen"
**self.setting_stack: QStackedWidget**
Stack zum anzeigen des gewünschten Kontainers.
**self.category_settings_container: QWidget**
Kontainerlement für Hilfstexte zu den Kategorieoptionen.
**self.payment_settings_container: QWidget**
Kontainerelement für Hilfstexte zu den Bezahlmethodenoptionen.
**self.payment_settings_layout: QGridLayout**
Layout für den Bezahlmethodenkontainer.
**self.export_settings_container: QWidget**
Kontainerelement für Hilfstexte zu den Exportoptionen.
**self.export_settings_layout: QGridLayout**
Layout für das Exportkontainerelement
**export_help_label: QLabel**
Label für den Hilfstext zu den Exportoptionen.
**payment_help_label: QLabel**
Label für den Hilfstext zu den Bezahlmethodenoptionen.
**category_help_label: QLabel**
Label für den Hilfstext zu den Kategorieoptionen.

## Methoden

Keine

DiagramScreen(QWidget)

## Modul

ui.py

## Beschreibung

Verantwortlich für das Darstellen von Diagrammen.

## Abhängigkeiten

sys
json
pathlib
datetime
pyside6
modules/api.py

### Argumente

### Attribute

grid = QGridLayout()
self.image_label = QLabel()

### Methoden

**pixmap_image(self, png_bytes: bytes):**
**Argumente**: png: bytes
**Beschreibung**: Rendert das erhaltene PNG auf das image_label
**Returns**: -

## ExpenseInsertForm(QWidget)

### Modul

ui.py

### Beschreibung

Eingabemaske zum Anlegen neuer Ausgaben.

### Abhängigkeiten

sys
json
pathlib
datetime
pyside6
modules/api.py

### Argumente

Keine

### Attribute

**grid: QGridLayout**
Layout für die Eingabemaske
**self.amount_input: QLineEdit**
Eingabefeld für den Betrag der Ausgabe.

**validator: QDoubleValidator**
Überprüft ob ein gültiger Wert für den Betrag eingeben worden ist.
**self.combo_category: QComboBox**
Dropdownmenü für Kategorien
**self.combo_subcategory: QComboBox**
Dropdownmenü für Unterkategorien
**self.combo_payment: QComboBox**
Dropdownmenü für Bezahlmethoden.
**self.date_edit: QDateEdit**
Eingabefeld für das Datum der Ausgabe.
**self.note_input: QTextEdit**
Eingabefeld für eine optionale Notiz zur Ausgabe.
**self.btn_save_expense: QPushButton**
Button zum speichern der Ausgabe

## Methoden

### change_parent_category(self, index) -> None:
**Argumente**: index: int
**Beschreibung**: Ändert das Dropdownmenü für Unterkategorien wenn die Hauptkategorie gewechselt wird.
**Returns**: -

### save_expense(self) -> bool:
**Argumente**: Keine
**Beschreibung**: Sendet die Eingegebenen Daten an das Backend damit eine neue Ausgabe angelegt wird.
**Returns**: -

# ProfileForm(QWidget)

## Modul

ui.py

## Beschreibung

Eingabemaske zum anlegen neuer Profile.

## Abhängigkeiten

sys
json
pathlib
datetime
pyside6
modules/api.py

### Argumente

Keine

### Attribute

**grid: QGridLayout**
Layout für die Eingabemaske zum Anlegen neuer Profile.
**self.combo_profile: QComboBox**
Dropdownmenü aller Profile.
**self.profile_name_input: QLineEdit**
Eingabefeld für den Profilnamen.
**self.btn_switch_profile: QPushButton**
Button zum ändern des derzeitig aktiven Profils.
**self.btn_create_profile: QPushButton**
Button zum anlegen eines neuen Profils.
**self.btn_delete_profile: QPushButton**
Button zum löschen eines vorhandenen Profils.
**self.btn_close_menu: QPushButton**
Button um zurück zur Hauptansicht zu gelangen.

### Methoden

**init_combo_profile(self) -> None:**
**Argumente**: Keine
**Beschreibung**: Initialisiert das Dropdownmenü für Profile
**Returns**: -

## ProfileDetails(QWidget)

### Modul

ui.py

### Beschreibung

Gibt einen Überblick über das gewählte Profil. Ältester Eintrag, neuster Eintrag, Summe aller Ausgaben etc.

### Abhängigkeiten

sys
json
pathlib

datetime
pyside6
modules/api.py

## Argumente

Keine

## Attribute

**profile_info: dict**
Dict über Profilinformationen.
**grid: QGridLayout**
Layout zur Darstellung von Profilinformationen.
**self.detail_list: QListWidget()**
Listenelement zur Darstellung von Profilinformationen.

## Methoden

**update_detail_list(self) -> None:**
**Argumente**: Keine
**Beschreibung**: Aktualisiert die Profilinformationen wenn das Profil gewechselt wird.
**Returns**: -

```
ExpenseUpdateForm(QWidget)
```

## Modul

ui.py

## Beschreibung

Eingabemaske zum bearbeiten von Ausgaben.

## Abhängigkeiten

sys
json
pathlib
datetime
pyside6
modules/api.py

## Argumente

Keine

## Attribute

**grid: QGridLayout**
Layout für die Eingabemaske
**self.amount_input: QLineEdit**
Eingabefeld für den Betrag der Ausgabe.
**validator: QDoubleValidator**
Validiert das Eingabfeld für den Betrag der Ausgabe.
**self.combo_category: QComboBox**
Dropdownmenü für Kategorien.
**self.combo_subcategory: QComboBox**
Dropdownmenü für Unterkategorien
**self.combo_payment: QComboBox**
Dropdownmenü für Bezahlmethoden.
**self.date_edit: QDateEdit**
Eingabefeld für das Datum der Ausgabe.
**self.note_input: QTextEdit**
Eingabefeld für die Notiz zur Ausgabe.
**self.btn_update_expense: QPushButton**
Button zum bearbeiten der Ausgabe.
**self.btn_abort_update: QPushButton**
Button um das bearbeiten abzubrechen und zum Anlegen von Ausgaben zurückzukehren.
**self.id_input: QLineEdit**
Verstecktes Eingabefeld. Wird mit der ID der zu bearbeitenden Ausgabe initialisiert.

## Methoden

### def change_parent_category(self, index):

**Argumente**: Keine
**Beschreibung**: Aktualisiert das Dropdownmenü für Unterkategorien wenn die Hauptkategorie gewechselt wird.
**Returns**: -

ExpenseFilterForm(QWidget)

## Modul

ui.py

## Beschreibung

Eingabemaske zum Filtern von Ausgaben.

## Abhängigkeiten

sys
json
pathlib
datetime
pyside6
modules/api.py

## Argumente

Keine

## Attribute

**grid: QGridLayout**
Layout für die Filtermaske
**validator: QDoubleValidator**
Validiert die eingebenen Daten bei minamount und maxamount.
**self.minamount_input: QLineEdit**
Eingabefeld für die Untergrenze des Betrages der zu filternden Ausgabe.
**self.maxamount_input: QLineEdit**
Eingabefeld für die Obergrenze des Betrages der zu filternden Ausgabe.
**self.ordercombo: QComboBox**
Dropdownmenü für die gewünschte Sortierreihenfolge.

## Methoden

### search_expense(self) -> list:

**Argumente**: expense_list: list
**Beschreibung**: Sendet ein Dict mit den Eingegebenen Filterkriterien an das Backend und gibt eine Liste mit den gefundenen Ausgaben zurück.
**Returns**: list

ExpenseTable(QWidget)

## Modul

ui.py

## Beschreibung

Stellt die gesuchten Ausgaben in einer Tabelle dar.

## Abhängigkeiten

sys
json
pathlib
datetime
pyside6
modules/api.py

## Argumente

Keine

## Attribute

**self.expense_table: QTableWidget**
Tabelle zur Darstellung von Ausgaben.

## Methoden

### render_table(self, expense_list:list) -> None

**Argumente**: expense_list: list
**Beschreibung**: Erstellt die Tabelle mit den Werten aus der übergebenen Liste.
**Returns**: -