WILLIAM R

T1A3

TERMINAL APP

SLIDE DECK

TO COVER:

01 APPLICATION FEATURES

02 APPLICATION STRUCTURE

03 CODE OVERVIEW

04 CODE DEEP DIVE

05 DEVELOPMENT REVIEW

APPLICATION FEATURES

My app, GreatReads is a command-line version of GoodReads, without the social network. It lets you create a library of shelves and add books to those shelves.

- ✔ Library persists between sessions
- ✔ User can perform CRUD on shelves
- ✔ User can perform CRUD on books in each shelf
- ✔ User can add books with a search feature using the OpenLibrary API

APPLICATION STRUCTURE

The app is structured with classes representing the main data objects (Library, Book, Shelf). These classes have methods like properties for retrieving data, to_JSON for serializing to a JSONable datatype, and allowing users to create, rename, and remove both shelves and books.

A main App file kicks off the interaction, allowing me to catch exceptions that need to be handled and ensure JSON is saved out with a final cleanup function. Then, I use a series of functions to handle menus, ensuring the user is never dropped out of the command-line environment.

There are a series of utility functions used for interacting with the **rich** library to prettyprint, as well as a separate class for running searches called **SearchHandler**.

The application is used exclusively from the command line. It takes input from the user in the form of letters and numbers to action menu items.

APPLICATION STRUCTURE

The source code directory is structured like this. I considered nesting the classes but it was proving easy enough to manage it cognitively, so I left them in the parent folder.

```
src
├── App.py
├── Book.py
├── exceptions.py
├── functions
│   ├── SearchHandler.py
│   ├── user_input.py
│   └── utils.py
├── Library.py
├── main.py
├── menus
│   ├── BookDetail.py
│   ├── library_menu.py
│   ├── shelf_detail_menu.py
│   └── welcome_menu.py
├── Shelf.py
└── tests
    ├── __init__.py
    ├── test_Book.py
    ├── test_Library.py
    └── test_Shelf.py

3 directories, 17 files
```

CODE OVERVIEW

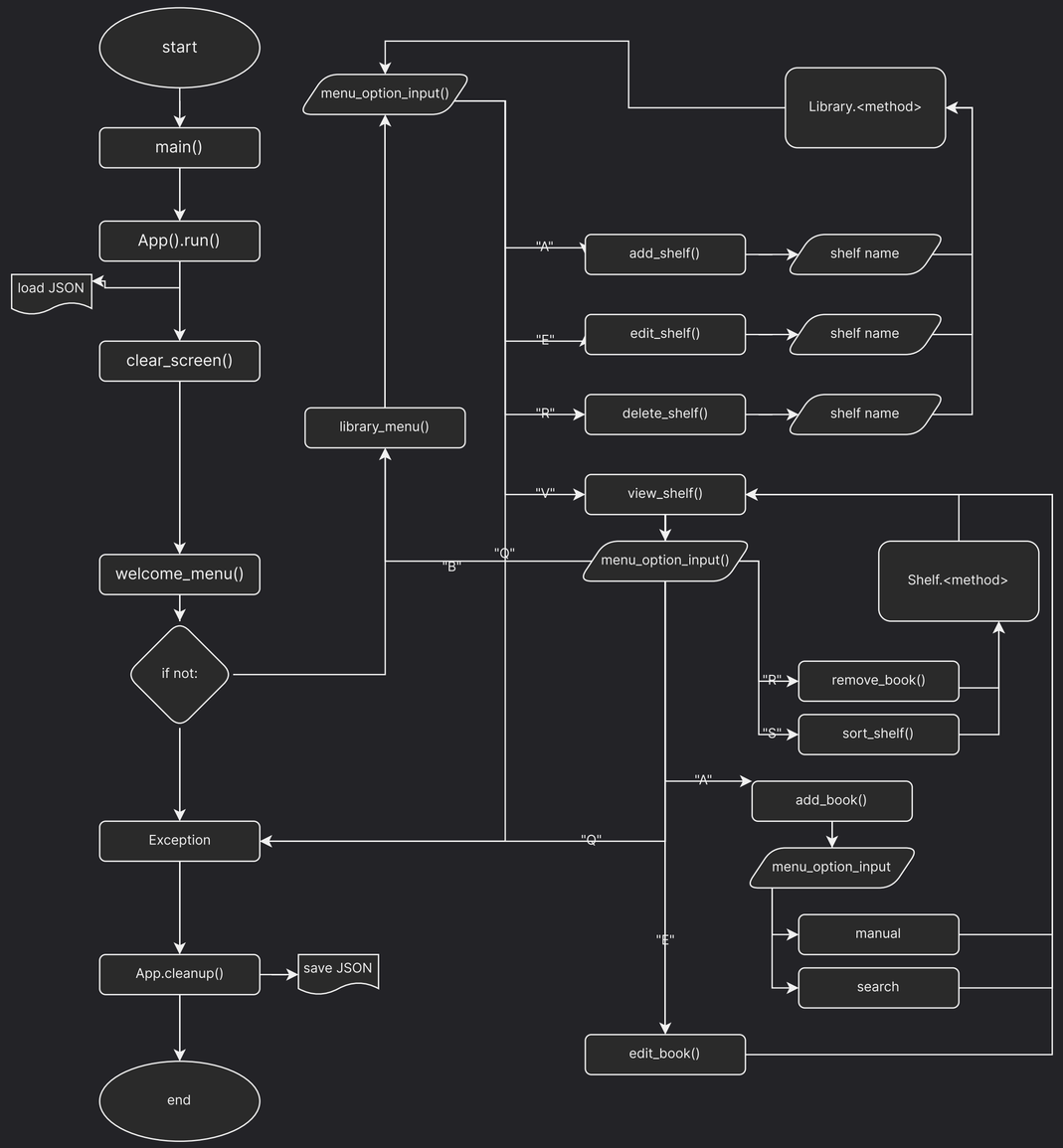
This diagram represents the main flow of the user through the app. For the sake of space, I've excluded any function that prints to the terminal.

The app begins, creates an App object that is responsible for attaching a Library to itself and then starting the main loop.

The welcome menu is a way to catch an accidental load, otherwise we go into the main library menu function. Input here calls one of a range of functions or returns false, which returns to the main loop and exits the program.

These functions print their options to the screen and then take further input to act on a Library object.

We can then step further into another nested loop, the view_shelf function, which shows a particular shelves' books.



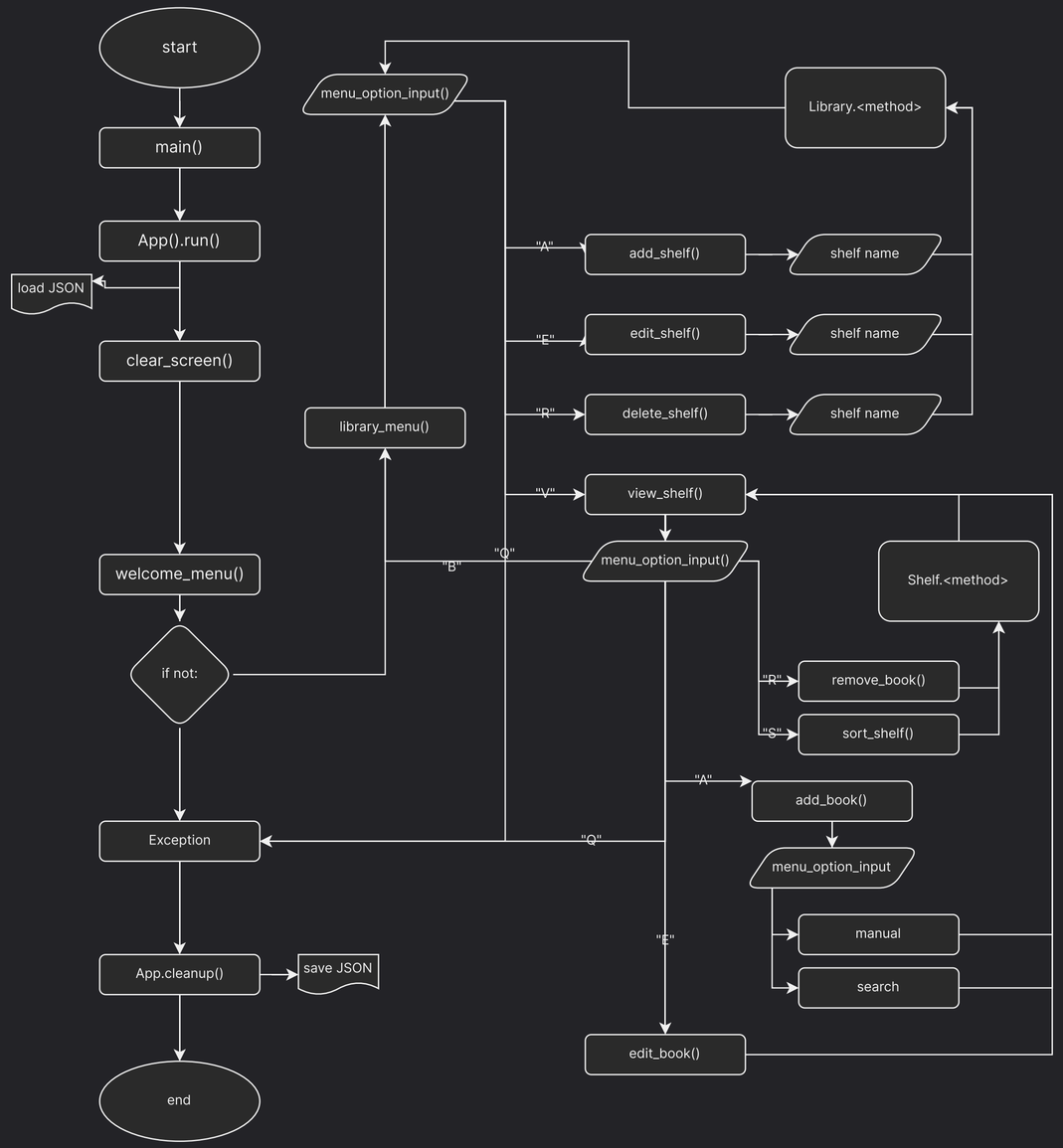
13774@CODERACADEMY.EDU.AU

CONT.

Inside the view shelf loop, we take input again. From there, there are two nested views - the edit view and the add book view. They use a class, BookDetailView, because they have some more complex internal logic. This class has 2 subclasses for each of the two views, with their own nested loops.

BookDetailAdd takes input and chooses either manual or search. Searching calls the SearchHandler which uses the OpenLibrary API, and processes the JSON to a printable table. Manual just enters at the prompt.

BookDetailEdit allows the user to select a field on the book to edit. It then calls the update method on the book object.



13774@CODERACADEMY.EDU.AU

DEEP DIVE: MAIN, APP

```python
def main():
    app = App()
    app.run()


if __name__ == "__main__":
    main()
```

```python
class App:
    def __init__(self):
        self.library = Library()

    def run(self):
        """Run main loop of the App."""
        clear_screen()
        try:
            if not welcome_menu():
                raise UserExited
            if not library_menu(self.library):
                raise UserExited
        except (UserExited, KeyboardInterrupt, EOFError):
            sys.exit(0)
        finally:
            self.cleanup()

    def cleanup(self):
        """Ensure safe shutdown."""
        print("\nClosing...")
        self.library.save_library()
```

DEEP DIVE: LIBRARY

```python
class Library:
    def __init__(self, directory="greatreads", file="library.json"):
        self.contents = []
        self.json_path = self.initialise_library_file(directory, file)
        data = self.deserialize_library()
        self.create_library(data)

    @property
    def shelf_count(self):
        return len(self.contents)

    def initialise_library_file(self, directory, file):
        """Checks for the existence of the greatreads library. If it does
        data_path = platformdirs.user_data_path().joinpath(directory)
        json_path = data_path.joinpath(file)
        try:
            if not data_path.exists():
                data_path.mkdir()
            if not json_path.is_file():
                json_path.touch()
            return json_path
        except Exception as exception:
            print(
                f"Sorry, something completely unexpected went wrong: {exc
            )
            exit(2)
```

```python
def deserialize_library(self):
    """Loads the library from disk."""

    library_path = self.json_path

    try:
        with open(library_path, "r") as file:
            cache = file.read()
            # Catch if there's an empty JSON file
            if cache:
                json_data = orjson.loads(cache)
                return json_data
    except FileNotFoundError as exception:
        print(
            f"Something's gone horribly wrong. {exception}. Please file a bug on GitHub."
        )
        exit(1)
```

```python
def save_library(self):
    """Save the library to disk as JSON."""

    json_shelf_list = []
    for shelf in self.contents:
        json_shelf_list.append(shelf.to_JSON())

    with open(self.json_path, "wb") as file:
        file.write(orjson.dumps({"shelves": json_shelf_list}))

    print(f"saved to {self.json_path}")
```

DEEP DIVE: SHELF

```python
class Shelf:
    def __init__(self, data: dict = None):
        if not data:
            data = {}

        self.contents = []
        self.shelf_name = "Default Shelf"

        if data:
            self.shelf_name = data.get("shelf_name", "Default Shelf")
            self.add_initial_books(data)

    def __str__(self) -> str:
        return f"{self.shelf_name}"

    @property
    def length(self):
        return len(self.contents)

    def add_initial_books(self, data):
        if data.get("books", None):
            for book in data.get("books"):
                self.contents.append(Book(book))

    def add_new_book(self, new_book):
        if new_book:
            self.contents.append(new_book)

    def remove_book(self, book_to_remove):
        new_contents = [book for book in self.contents if book != book_to_remove]

        self.contents = new_contents

    def to_JSON(self):
        json_dict = {"shelf_name": self.shelf_name, "books": []}
        for book in self.contents:
            json_dict["books"].append(book.to_JSON())

        return json_dict
```

DEEP DIVE: SHELF MENU

```python
def shelf_detail_menu(library, active_shelf_name):
    """Main function that runs the shelf view menu."""
    shelf_detail_menu_choices = {
        "A": menu_add_book,
        "E": menu_edit_book,
        "R": menu_delete_book,
        "S": menu_sort_shelf,
    }

    console = Console()

    console.print(
        menu_banner(
            heading="Shelf Detail View",
            sub="Here you can see all your books on the shelf.",
        )
    )

    active_shelf = library.get_shelf(active_shelf_name)

    print_shelf_detail_menu(active_shelf=active_shelf, console=console)

    while True:
        user_choice = menu_option_input("Choose your option: ")
        if user_choice == "Q":
            raise UserExited
        if user_choice == "B":
            return False
        if user_choice not in shelf_detail_menu_choices.keys():
            print("Sorry, please select a valid choice.")
            continue
        shelf_detail_menu_choices[user_choice](
            active_shelf=active_shelf, console=console
        )
        print_shelf_detail_menu(active_shelf=active_shelf, console=console)
```

```python
def print_shelf_detail_menu(active_shelf, console):
    """Present menu options."""

    print_shelf_contents(active_shelf=active_shelf, console=console)

    table_options = {
        "A": "Add a Book",
        "E": "Edit a Book",
        "R": "Remove a Book",
        "S": "Sort Shelf",
        "B": "Go Back",
        "Q": "Quit",
    }

    console.print(create_menu_table(table_options, show_header=False, box=False))


def print_shelf_contents(active_shelf, console):
    """Print books in shelf."""
    table = Table()
    if active_shelf.length == 0:
        table.show_header = False
        table.add_row("Empty shelf!", style="italic")
    else:
        table.add_column("Book Title")
        table.add_column("Author")
        table.add_column("First Published")
        for book in active_shelf.contents:
            table.add_row(book.title, book.author, book.publication_year)
    console.print(table)
```

DEEP DIVE: SHELF MENU CONT.

```python
def menu_add_book(active_shelf, console):
    """Opens the add menu screen."""
    BookDetailAdd(active_shelf=active_shelf, console=console).add_book()


def menu_edit_book(active_shelf, console):
    """Asks for user input and if book is found, opens the edit menu."""
    if active_shelf.length == 0:
        console.print("Sorry, there are no books to edit.", style="b")
        sleep(1)
    else:
        console.print("Which book would you like to edit?", style="b")
        book = find_book_in_shelf(active_shelf=active_shelf)
        if not book:
            console.print(f"Sorry, I can't find that book.", style="red b")
            return
        BookDetailEdit(console=console, book=book).edit_book()
```

```python
def menu_sort_shelf(active_shelf, console):
    """Prints a menu and sorts depending on user input."""
    print_sort_menu(console)
    user_choice = menu_option_input("Sort by what?")
    if user_choice == "T":
        active_shelf.contents.sort(key=lambda book: book.title)
    if user_choice == "A":
        active_shelf.contents.sort(key=lambda book: book.author)
    if user_choice == "Y":
        active_shelf.contents.sort(
            key=lambda book: book.first_publish_year, reverse=True
        )
```

DEEP DIVE: BOOK SEARCH

BOOKDETAIL.PY

```python
def add_book_search(self) -> Book | bool:
    """Get a search string from a user, uses a SearchHandler
    to discover the book, present the results and take their selection."""
    self.console.print("Search Mode", style="i blue underline")
    term_to_search = handle_string_input("Search term: ")
    search_handler = SearchHandler(term_to_search)
    try:
        search_handler.run_search()
    except NetworkConnectivityError:
        self.console.print(
            "Sorry, there's been a network error. Check your internet connection a
            style="red b",
        )
        return False

    self.console.print(search_handler.search_results)
    while True:
        self.console.print("Select your result from 1-10, or \cancel to cancel.")
        chosen_result = handle_string_input("Selection: ")
        if chosen_result == "\cancel":
            break
        if not chosen_result.isnumeric():
            continue
        if 1 > int(chosen_result) < 10:
            found_book = search_handler.make_user_choice(chosen_result)
            return found_book
```

SEARCHHANDLER.PY

```python
class SearchHandler:

    API_URL = "http://openlibrary.org/search.json"

    def __init__(self, search_term: str):
        self.search_term = search_term

    def run_search(self):
        """Method responsible for running the search from start to finish."""
        try:
            self.search_response = self._perform_search()
            self.results_list = self._process_results()
            self._search_results = self._create_result_table()
        except (requests.exceptions.ConnectionError, requests.exceptions.HTTPError
            raise NetworkConnectivityError

    @property
    def search_results(self):
        return self._search_results

    def _perform_search(self):
        search_term = self.search_term
        split_by_plus = search_term.replace(" ", "+")
        response = requests.get(
            SearchHandler.API_URL, params={"q": split_by_plus, "limit": 10}
        )
        return response.content

    def _process_results(self):
        search_response = self.search_response
        results = []
        processed = orjson.loads(search_response)
        for book_entry in processed["docs"]:
            new_book = Book(
                {
                    "title": book_entry.get("title"),
                    "author": book_entry.get("author_name", [""])[0],
                    "first_publish_year": book_entry.get("first_publish_year"),
                }
            )
            results.append(new_book)
        return results
```

DEVELOPMENT REVIEW

I had some difficulties at the start of this project - I ambitiously chose to use a TUI library that I ended up abandoning... Twice. Learned my lesson about doing further requirements gathering before breaking ground!

Once I got my head around the code, things went smoothly. The most complicated thing turned out to be menu management - the actual logic of the classes came together quite easily.

Managing the project through Trello was very useful and exciting to see how it flowed as I finished more features.

Testing was also easier than I thought it would be, pytest is a lot more intuitive than other JS frameworks I've used.

Thanks for watching.