Prosjekt 2 – sykkelbutikk i python

# UML diagram:

|  |
| --- |
| User |
| + id: int  + name: str  + history: list[HistoryItem] |
| + save\_to\_history(rental\_bike\_id: int, bike\_id: int, bike\_name: str, from\_date: str, to\_date: str, price: int, action: str)  + return\_bike(rental\_bike\_id: int) |

|  |
| --- |
| HistoryItem |
| + user\_id: int  + rental\_bike\_id: int  + bike\_id: int  + bike\_name: str  + from\_date: str  + to\_date: str  + price: int  + action: str |
| + save\_to\_history(rental\_bike\_id: int, bike\_id: int, bike\_name: str, from\_date: str, to\_date: str, price: int, action: str)  + return\_bike(rental\_bike\_id: int) |

|  |
| --- |
| App |
| - global\_state  - screen\_manager |
| + run() |

|  |
| --- |
| Store |
| + name: str  + inventory: list[Bike]  + carts: dict  + id: int |
| + add\_to\_cart(user: User, item: dict)  + add\_inventory(bike\_id: int, quantity: int)  + get\_available\_bikes()  + rent(rental\_bike\_id: int, user: User, from\_date: datetime, to\_date: datetime)  + return\_bike(rental\_bike\_id: int)  + check\_and\_return\_bikes(time: datetime) |

|  |
| --- |
| RentalBike |
| + bike\_id: int  + is\_available: bool  + id: int  + renter: User  + from\_date: datetime  + to\_date: datetime |
|  |

|  |
| --- |
| GlobalState |
| - users: list[User]  + active\_user: User  + stores: list[Store]  + active\_store: Store  + bike\_types: list[Bike]  + time\_offset: datetime.timedelta |
| + get\_time()  + add\_time()  + reset\_time()  + load(filename: str)  + save(filename: str)  + add\_store(name: str)  + user\_exists(name: str)  + add\_user(name: str)  + set\_active\_user(user: User)  + get\_user\_by\_name(name: str)  + get\_user\_by\_id(id: int)  + get\_store(name: str)  + add\_bike\_type(name: str, description: str, hour\_price: int)  + get\_bike\_type\_by\_name(name: str)  + get\_bike\_type\_by\_id(id: int)  + get\_active\_cart()  + get\_cart(id: int)  + add\_to\_active\_cart(item: dict)  + clear\_active\_cart()  + check\_and\_return\_bikes()  + get\_user\_rentals(user: User) |

|  |
| --- |
| Bike |
| + id: int  + name: str  + description: str  + hour\_price: str |
|  |

|  |
| --- |
| ScreenManager |
| + global\_state: GlobalState  + clear\_display: bool  + history: list[str]  + start\_screen: callable |
| - clear\_display()  + set\_start\_screen(function: callable)  + start()  + go\_back()  + stay()  + \_<name>\_screen\_function() |

## Kommentarer

ScreenManager har veldig mange \*\_screen\_function() funksjoner, derfor har jeg ikke lagt dem til i UML-diagrammet. Fant ikke noen bedre måte å «simulere» nettsider på i python.

# Flowchart:Image

## Om Git

Git er et program for versjonskontroll. Det installeres på PCen og brukes gjennom command line. Versjonskontroll betyr å holde kontroll på historikken til en eller flere filer. I programmering får man ofte bugs og problemer når man prøver å kode en ny løsning eller funksjonalitet. Da vil ikke koden fungere og man kan få problemer med å fikse opp bugsa. I enkle tilfeller holder det å gå tilbake med ctrl+z, men hvis for mange endringer har blitt gjort så kan man risikere å ha brukt opp størrelsen på undo history. Versjonskontroll gir da muligheten til å «gå tilbake i tid» til et tidspunkt da koden fungerte. Git lar også folk samarbeide på samme kode, men da bruker man oftest andre verktøy som bygger på Git, som f.eks. Github eller BitBucket. Dette aspektet av Git er veldig viktig om man jobber sammen med andre. Da kan flere folk jobbe på samme codebase samtidig. Når man til slutt vil sette sammen arbeidet til en felles kode, så kan man gjøre det ved «merging».

Versjonskontroll, Git inkludert, er primitivt og lagrer ikke filene automatisk. Man må selv lagre filene med Git. Her snakker jeg om en annen type lagring enn ctrl+s. Filene er lagret på maskinen, men Git lagrer ikke filene dine i «commit history» før du ber den om det. «Commit history» er en tidslinje som består av «commits». En commit er en snapshot (øyeblikksbilde) av alle filene på det tidspunktet man «commiter» (lagrer med Git). Snapshoten lagrer alle filene, men den dupliserer ikke data om dataen er uendret. Om noen filer er uendret mellom to commits, så lagrer Git kun en referanse til den eksisterende filen. Derimot, hvis comitter en fil med endringer, så er Git smart og lagrer kun endringene som ble gjort og ikke hele filen. Prosessen av å commite filer med Git starter med at man «stager» dem. Det betyr at man markerer dem for å bli lagret, men de er ikke lagret enda. Så må man commite dem, og da lagrer Git dem til commit history. Om man vil gå tilbake til en tidligere commit så gjør man det med en «checkout». Da «sjekker» man ut på den tiden på tidslinjen.

Kilde:  
<https://stackoverflow.com/questions/8198105/how-does-git-store-files>