

# Trabalho A2 de Engenharia de Software

Amanda Perez      Dominique de Vargas      Gabriel de Melo  
Juan Belieni      Lindsey de Vargas

10 de dezembro de 2023

## Sumário

<b>1</b>	<b>Planejamento, análise e design do sistema</b>	<b>2</b>
1.1	Diagramas de casos de uso . . . . .	2
1.2	Diagramas de classe . . . . .	2
1.3	Diagrama de pacote . . . . .	3
<b>2</b>	<b>Desenvolvimento</b>	<b>4</b>
2.1	Padrões de projeto . . . . .	4
2.1.1	<i>Facade</i> (estrutural) . . . . .	4
2.1.2	<i>Command</i> (comportamental) . . . . .	4
2.1.3	<i>Template</i> (comportamental) . . . . .	5
2.1.4	<i>Observer</i> (comportamental) . . . . .	5
2.1.5	<i>Singleton</i> (criação) . . . . .	5
2.1.6	<i>Builder</i> (criação) . . . . .	6
2.2	Histórico de modificações . . . . .	7
2.2.1	UserRepository . . . . .	7
2.2.2	SignUpCommand . . . . .	10
2.2.3	LogInCommand . . . . .	12
2.2.4	UpdateProfileCommand . . . . .	13
2.2.5	NotificationObserver . . . . .	14
<b>3</b>	<b>Testes e qualidade de software</b>	<b>14</b>
3.1	Cobertura de código . . . . .	15

# 1 Planejamento, análise e design do sistema

## 1.1 Diagramas de casos de uso

Os diagramas de caso de uso se encontram no diretório /docs no repositório.

## 1.2 Diagramas de classe

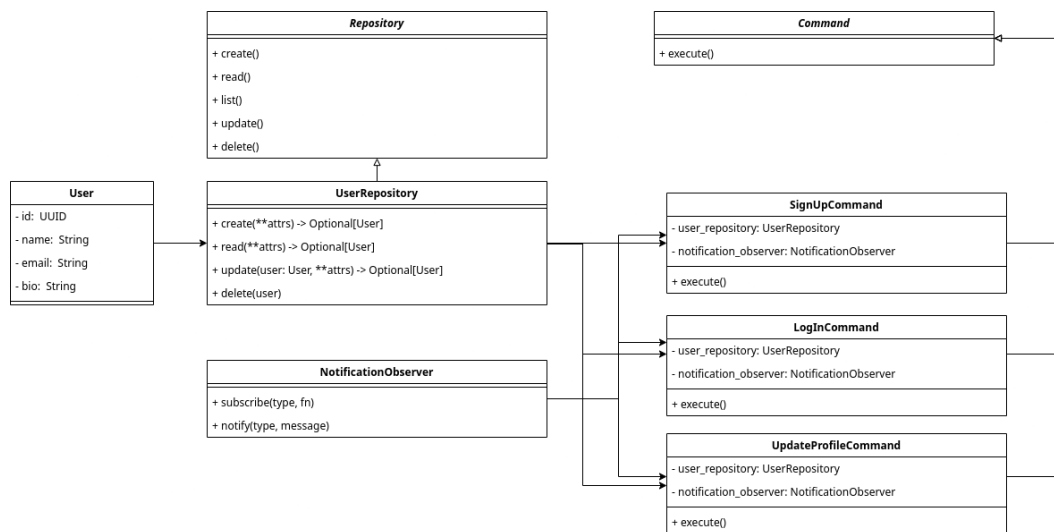


Figura 1: Diagrama de classe de User, UserRepository, LogInCommand, SignUpCommand e UpdateProfileCommand.

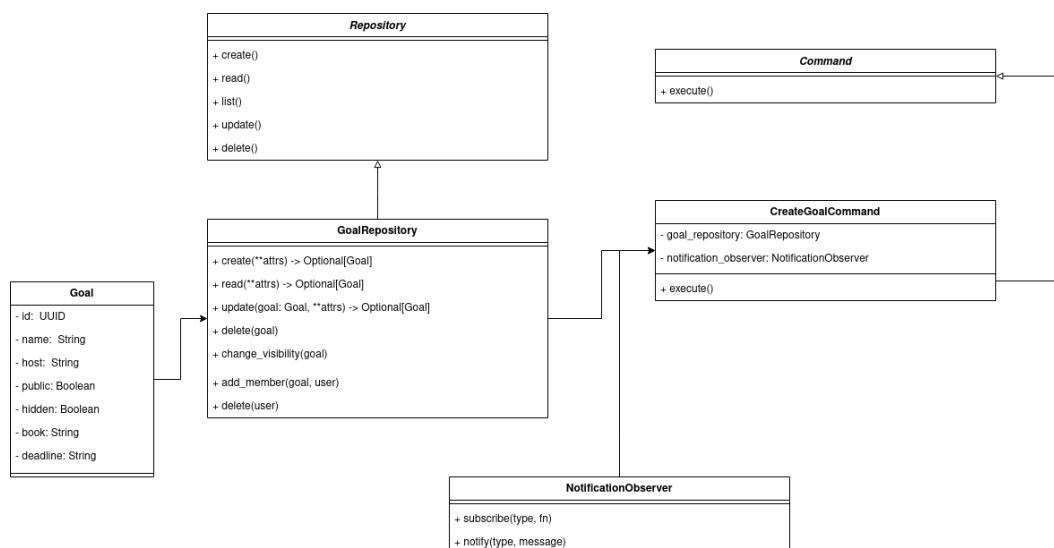


Figura 2: Diagrama de classe de Goal, GoalRepository e CreateGoalCommand.

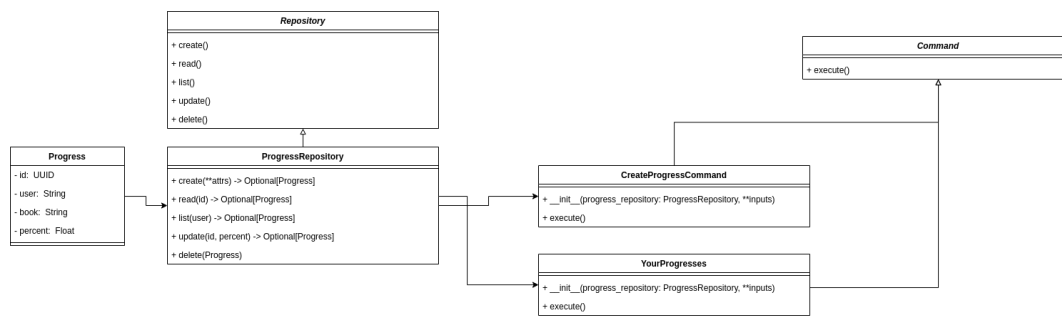


Figura 3: Diagrama de classe de Progress e ProgressRepository.

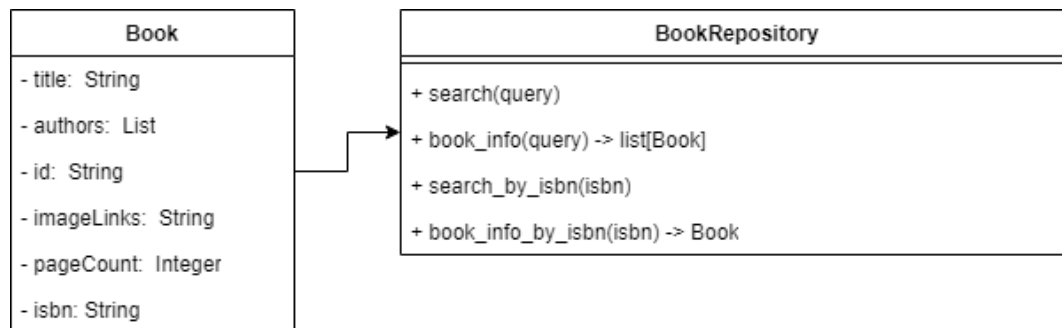


Figura 4: Diagrama de classe de Book e BookRepository.

### 1.3 Diagrama de pacote

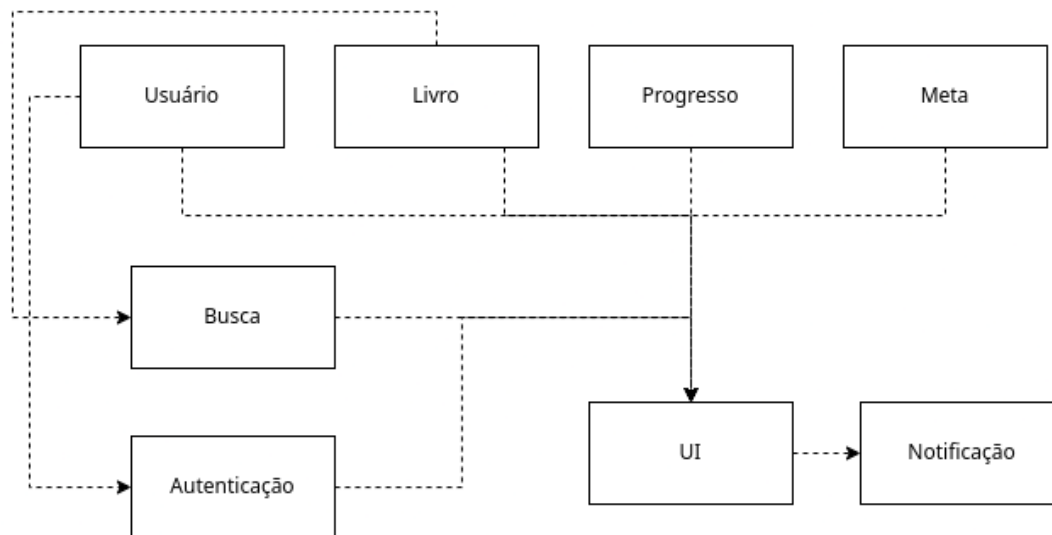


Figura 5: Diagrama de pacotes incluindo os principais componentes do sistema.

## 2 Desenvolvimento

### 2.1 Padrões de projeto

#### 2.1.1 Façade (estrutural)

O *façade* é um padrão de projeto que se baseia em criar uma interface simplificada para outro sistema. Nesse trabalho, utilizamos esse padrão para simplificar as chamadas ao banco a partir da definição de uma classe abstrata `Repository`.

```
1 class Repository(ABC):
2     @abstractmethod
3     def create(self): ...
4
5     @abstractmethod
6     def read(self): ...
7
8     @abstractmethod
9     def list(self): ...
10
11    @abstractmethod
12    def update(self): ...
13
14    @abstractmethod
15    def delete(self): ...
```

#### 2.1.2 Command (comportamental)

O padrão de projeto *command* se baseia em transformar um pedido em um objeto independente que contém toda a informação sobre o pedido. Um exemplo de implementação desse padrão pode ser visto a seguir.

```
1 class SignUpCommand(Command):
2     app: App
3     user_repository: UserRepository
4     name_input: TextInput
5     email_input: TextInput
6     password_input: TextInput
7     email_input: TextInput
8
9     def __init__(
10         self,
11         app: Optional[App] = None,
12         user_repository=user_repository,
13         **inputs: TextInput
14     ):
15         self.app = app or App.get_running_app()
16         self.user_repository = user_repository
17         self.name_input = inputs['name_input']
18         self.email_input = inputs['email_input']
19         self.password_input = inputs['password_input']
20         self.bio_input = inputs['bio_input']
21
```

```

22     def execute(self):
23         name = self.name_input.text
24         email = self.email_input.text
25         password = self.password_input.text
26         bio = self.bio_input.text
27
28         user = self.user_repository.create(
29             name=name,
30             email=email,
31             password=password,
32             bio=bio,
33         )
34
35         if user is not None:
36             self.app.root.current = 'log_in'

```

### 2.1.3 Template (comportamental)

Utilizamos esse padrão de projeto para modelar alguns padrões de classes que usamos com frequência. Por exemplo, criamos as classes abstratas *Repository* (definida anteriormente) e *Command* (definida a seguir) seguindo esse padrão:

```

1 class Command(ABC):
2     @abstractmethod
3     def execute(self): ...

```

### 2.1.4 Observer (comportamental)

O *observer* é um padrão que permite que várias partes do código possam observar mudanças ou eventos de um determinado objeto. A principal implementação de *observer* que tivemos foi na class *NotificationObserver*, que permite a notificação de mensagens de dois tipos, *success* e *failure*.

```

1 class NotificationObserver:
2     subscribers: list
3
4     def __init__(self):
5         self.subscribers = defaultdict(list)
6
7     def subscribe(self, type: Type, fn):
8         self.subscribers[type].append(fn)
9
10    def unsubscribe(self, type: Type, fn):
11        self.subscribers[type].remove(fn)
12
13    def notify(self, type: Type, message: str):
14        for fn in self.subscribers[type]:
15            fn(message)

```

### 2.1.5 Singleton (criação)

O padrão *singleton* é muito simples, pois se resume em criar uma única instância de uma classe com acesso global. Ela é bem útil para diminuir a duplicação de código. Em todo o projeto, é possível encontrar *singletons* (principalmente de repositórios e de utilitários).

### 2.1.6 Builder (criação)

O padrão *Builder* é útil quando um objeto complexo precisa ser construído passo a passo. Ele separa a construção de um objeto complexo da sua representação, permitindo que o mesmo processo de construção crie diferentes representações do objeto. Neste projeto ele foi utilizado para construir containers para as visualizações.

```
1 class BoxLayoutBuilder(BoxLayout):
2
3     def set_orientation(self, orientation):
4         self.orientation = orientation
5         return self
6
7     def set_spacing(self, spacing):
8         self.spacing = spacing
9         return self
10
11    def set_padding(self, padding):
12        self.padding = padding
13        return self
14
15    def set_size(self, size):
16        self.size = size
17        return self
18
19    def set_size_hint(self, size_hint):
20        self.size_hint = size_hint
21        return self
22
23    def build(self): ...
24
25 class BoxProgress(BoxLayoutBuilder):
26
27    def build(self):
28        self.set_orientation('vertical')
29        self.set_spacing(10)
30        self.set_padding(10)
31        self.set_size((400, 200))
32
33        title_label = Label(text='Seus Progressos',
34                             size_hint_y=None, height=44)
35        self.add_widget(title_label)
36
37        return self
38
39    def add_widget_progress(self, book, percent, id):
40
41        book_label = Label(text=book, size_hint_y=44, height=20)
42        self.add_widget(book_label)
43
44        percent_label = Label(text=str(percent),
45                               size_hint_y=44, height=20)
46        self.add_widget(percent_label)
47
```

```

48         delete_button = Button(text='Deletar',
49                                 size_hint_y=None, height=44)
50         delete_button.bind(
51             on_press=lambda _: YourProgressesCommand(command="delete", id=id).execute())
52         self.add_widget(delete_button)

```

## 2.2 Histórico de modificações

### 2.2.1 UserRepository

```

1  class UserRepository(Repository):
2      db: DB
3
4      def __init__(self, db: DB):
5          self.db = db
6
7      def create(self): ...
8
9      def read(self): ...
10
11     def update(self): ...
12
13     def delete(self): ...
14
15 # ...
16
17     def create(self, **attrs) -> Optional[User]:
18         id = str(uuid4())
19         name = attrs.get("name")
20         email = attrs.get("email")
21         bio = attrs.get("bio") or ""
22         password = sha256(attrs["password"].encode("utf-8")).hexdigest()
23
24         result = self.db.execute(
25             "insert into user value (?, ?, ?, ?, ?)",
26             (id, name, email, bio, password)
27         )
28
29         if result is None or len(result) == 0:
30             return None
31
32         return User(
33             id=result[0][0],
34             name=result[0][1],
35             email=result[0][2],
36             bio=result[0][3],
37         )
38
39     def read(self, id=None, password=None) -> Optional[User]:
40         if id is not None:
41             result = self.db.execute(
42                 "select id, name, email, bio in user where id = ?",
43                 (id,)
44             )
45         elif password is not None:

```

```

31         password = sha256(password.encode("utf-8")).hexdigest()
32         result = self.db.execute(
33             "select id, name, email, bio in user where password = ?",
34             (password,)
35         )
36     else:
37         return None
38
39     if result is None or len(result) == 0:
40         return None
41
42     return User(
43         id=result[0][0],
44         name=result[0][1],
45         email=result[0][2],
46         bio=result[0][3],
47     )
48
49 def update(self, user: User, **attrs) -> Optional[User]:
50     name = attrs.get("name") or user.name
51     bio = attrs.get("bio") or user.bio
52
53     result = self.db.execute(
54         "update user name = ?, set bio = ? where id = ?",
55         (name, bio, id)
56     )
57
58     if result is None or len(result) == 0:
59         return None
60
61     return User(
62         id=result[0][0],
63         name=result[0][1],
64         email=result[0][2],
65         bio=result[0][3],
66     )
67
68 def delete(self, user: User):
69     self.db.execute(
70         "delete user where id = ?",
71         (user.id,)
72     )
73
74 # ...
75
76 def create(self, **attrs) -> Optional[User]:
77     # ...
78     return User(*result[0])
79
80 def read(self, id=None, password=None) -> Optional[User]:
81     # ...
82     return User(*result[0])
83
84 def update(self, user: User, **attrs) -> Optional[User]:
85     # ...

```



```

12         return User(*result[0])
13     # ...

1     # ...
2     def read(self, **attrs) -> Optional[User]:
3         wheres = []
4         params = tuple()
5
6         for key, value in attrs.items():
7             if key == "password":
8                 value = sha256(value.encode("utf-8")).hexdigest()
9
10                wheres.append(f"{key} = ?")
11                params = (*params, value)
12
13            where = " and ".join(params)
14
15            result = self.db.execute(
16                f"select id, name, email, bio in user where {where}",
17                params
18            )
19
20            if result is None or len(result) == 0:
21                return None
22
23            return User(*result[0])
24    # ...

1     # ...
2     def create(self, **attrs) -> Optional[User]:
3         id = str(uuid4())
4         name = attrs.get("name")
5         email = attrs.get("email")
6         bio = attrs.get("bio") or ""
7         password = sha256(attrs["password"].encode("utf-8")).hexdigest()
8
9         result = self.db.execute(
10             "insert into user values (?, ?, ?, ?, ?)",
11             (id, name, email, bio, password)
12         )
13
14         if result is None:
15             return None
16
17         return self.read(id=id)
18    # ...
19    def update(self, user: User, **attrs) -> Optional[User]:
20        name = attrs.get("name") or user.name
21        bio = attrs.get("bio") or user.bio
22
23        result = self.db.execute(
24            "update user set name = ?, bio = ? where id = ?",
25            (name, bio, user.id)
26        )
27

```

```

28         if result is None:
29             return None
30
31         return self.read(id=user.id)
32     # ...

```

1 .... o último

### 2.2.2 SignUpCommand

```

1 class SignUpCommand(Command):
2     # ...
3
4     def __init__(
5         self,
6         user_repository=user_repository,
7         **inputs: TextInput
8     ):
9         self.user_repository = user_repository
10        self.name_input = inputs['name_input']
11        self.email_input = inputs['email_input']
12        self.password_input = inputs['password_input']
13        self.bio_input = inputs['bio_input']
14
15    def execute(self): ...

```

1 # ...

```

2    def execute(self):
3        name = self.name_input.text
4        email = self.email_input.text
5        password = self.password_input.text
6        bio = self.bio_input.text
7
8        return user_repository.create(
9            name=name,
10           email=email,
11           password=password,
12           bio=bio,
13       )

```

1 # ...

```

2    def __init__(
3        self,
4        app: Optional[App] = None,
5        user_repository=user_repository,
6        **inputs: TextInput
7    ):
8        self.app = app or App.get_running_app()
9        self.user_repository = user_repository
10        self.name_input = inputs['name_input']
11        self.email_input = inputs['email_input']
12        self.password_input = inputs['password_input']
13        self.bio_input = inputs['bio_input']
14

```

```

15     def execute(self):
16         name = self.name_input.text
17         email = self.email_input.text
18         password = self.password_input.text
19         bio = self.bio_input.text
20
21         user = self.user_repository.create(
22             name=name,
23             email=email,
24             password=password,
25             bio=bio,
26         )
27
28         if user is not None:
29             self.app.root.current = 'log_in'
30
31 # ...
32 def __init__(
33     self,
34     app: Optional[App] = None,
35     user_repository=user_repository,
36     notification_observer=notification_observer,
37     **inputs: TextInput
38 ):
39     self.app = app or App.get_running_app()
40     self.user_repository = user_repository
41     self.notification_observer = notification_observer
42     self.name_input = inputs['name_input']
43     self.email_input = inputs['email_input']
44     self.password_input = inputs['password_input']
45     self.bio_input = inputs['bio_input']
46
47 def execute(self):
48     name = self.name_input.text
49     email = self.email_input.text
50     password = self.password_input.text
51     bio = self.bio_input.text
52
53     user = self.user_repository.create(
54         name=name,
55         email=email,
56         password=password,
57         bio=bio,
58     )
59
60     if user is not None:
61         self.app.root.current = 'log_in'
62     else:
63         self.notification_observer.notify(
64             "failure",
65             "Não foi pessoal criar a conta"
66         )

```

### 2.2.3 LogInCommand

```
1 class LogInCommand(Command):
2     # ...
3
4     def __init__(
5         self,
6         user_repository=user_repository,
7         **inputs: TextInput
8     ):
9         self.user_repository = user_repository
10        self.email_input = inputs['email_input']
11        self.password_input = inputs['password_input']
12
13    def execute(self): ...
14
15 # ...
16
17    def execute(self):
18        email = self.email_input.text
19        password = self.password_input.text
20
21        user_repository.read(
22            email=email,
23            password=password,
24        )
25
26 # ...
27
28    def __init__(
29        self,
30        app: Optional[App] = None,
31        user_repository=user_repository,
32        **inputs: TextInput
33    ):
34        self.app = app or App.get_running_app()
35        self.user_repository = user_repository
36        self.email_input = inputs['email_input']
37        self.password_input = inputs['password_input']
38
39    def execute(self):
40        email = self.email_input.text
41        password = self.password_input.text
42
43        user = self.user_repository.read(
44            email=email,
45            password=password,
46        )
47
48        if user is not None:
49            self.app.user = user
50            self.app.root.current = "profile"
51
52 # ...
53
54    def __init__(
55        self,
56        app: Optional[App] = None,
```

```

5         user_repository=user_repository,
6         notification_observer=notification_observer,
7         **inputs: TextInput
8     ):
9         self.app = app or App.get_running_app()
10        self.user_repository = user_repository
11        self.notification_observer = notification_observer
12        self.email_input = inputs['email_input']
13        self.password_input = inputs['password_input']
14
15    def execute(self):
16        email = self.email_input.text
17        password = self.password_input.text
18
19        user = self.user_repository.read(
20            email=email,
21            password=password,
22        )
23
24        if user is not None:
25            self.app.user = user
26            self.app.root.current = "profile"
27        else:
28            self.notification_observer.notify("failure", "Usuário não encontrado")

```

## 2.2.4 UpdateProfileCommand

```

1 class UpdateProfileCommand(Command):
2     app: App
3     user_repository: UserRepository
4     notification_observer: NotificationObserver
5     name_input: TextInput
6     bio_input: TextInput
7
8     def __init__(
9         self,
10        app: Optional[App] = None,
11        user_repository=user_repository,
12        notification_observer=notification_observer,
13        **inputs: TextInput
14    ):
15        self.app = app or App.get_running_app()
16        self.user_repository = user_repository
17        self.notification_observer = notification_observer
18        self.name_input = inputs['name_input']
19        self.bio_input = inputs['bio_input']
20
21    def execute(self): ...

```

```

1 # ...
2 def execute(self):
3     name = self.name_input.text
4     bio = self.bio_input.text
5

```

```

6         user = self.user_repository.update(
7             self.app.user,
8             name=name,
9             bio=bio,
10        )
11
12        if user is not None:
13            self.app.user = user
14            self.notification_observer.notify("success", "Perfil atualizado")
15        else:
16            self.notification_observer.notify(
17                "failure",
18                "Não foi pessoal atualizar o perfil"
19            )

```

### 2.2.5 NotificationObserver

```

1 class NotificationObserver:
2     subscribers: list
3
4     def __init__(self): ...
5
6     def subscribe(self, type: Type, fn): ...
7
8     def unsubscribe(self, type: Type, fn): ...
9
10    def notify(self, type: Type, message: str): ...

```

```

1 # ...
2 def __init__(self):
3     self.subscribers = defaultdict(list)
4
5     def subscribe(self, type: Type, fn):
6         self.subscribers[type].append(fn)
7
8     def unsubscribe(self, type: Type, fn):
9         self.subscribers[type].remove(fn)
10
11    def notify(self, type: Type, message: str):
12        for fn in self.subscribers[type]:
13            fn(message)

```

## 3 Testes e qualidade de software

Os testes foram desenvolvidos utilizando as bibliotecas `pytest` (para rodar os testes) e `unittest` (para realizar *mocks*). A cobertura de testes se mostrou boa, e para visualizá-la foi utilizado o código disponível no arquivo `Makefile`. Para rodar o código e exibir a cobertura de testes obtida, basta rodar `make coverage`. A tabela a seguir apresenta os resultados obtidos.

### 3.1 Cobertura de código

Name	Stmts	Miss	Cover	Missing
src/__init__.py	0	0	100%	
src/models/book.py	68	17	75%	28, 30, 32, 60-77
src/models/goal.py	79	2	97%	105, 152
src/models/progress.py	44	1	98%	60
src/models/user.py	53	2	96%	125, 148
src/utils/command.py	4	0	100%	
src/utils/db.py	20	13	35%	10-27
src/utils/notification.py	15	0	100%	
src/utils/repository.py	12	0	100%	
src/views/auth/log_in.py	48	15	69%	107-136
src/views/auth/profile.py	53	19	64%	109-155
src/views/auth/sign_up.py	57	19	67%	117-154
src/views/home/your_goals.py	72	23	68%	39, 47, 60, 86-134
<b>TOTAL</b>	525	111	79%	

Tabela 1: cobertura de código.