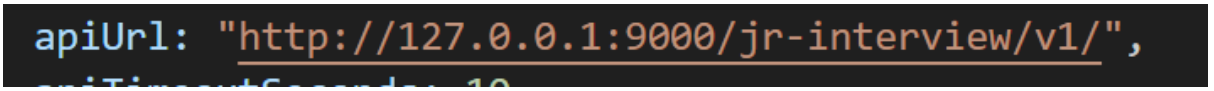
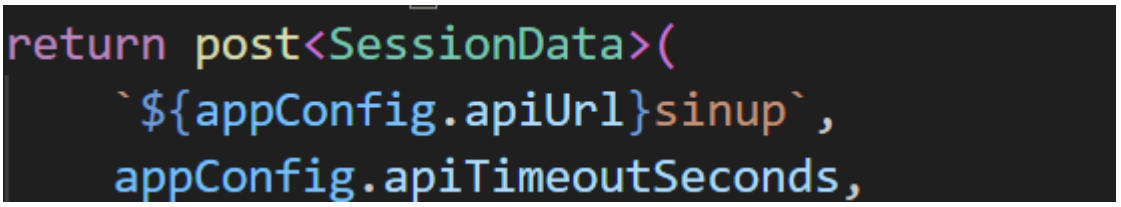
First task

Front bugs:

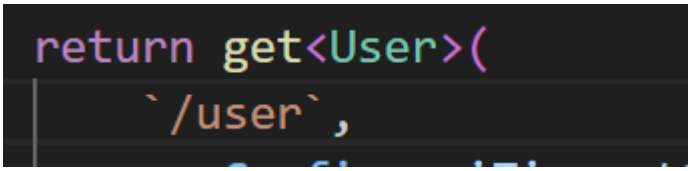
1. Wrong url for backend (v1 instead of v2)



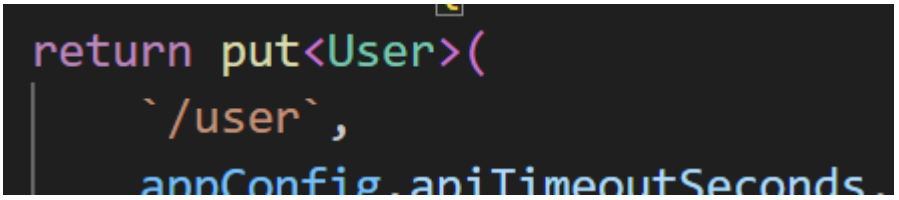
2. Wrong url for signup



3. Wrong url for user get method

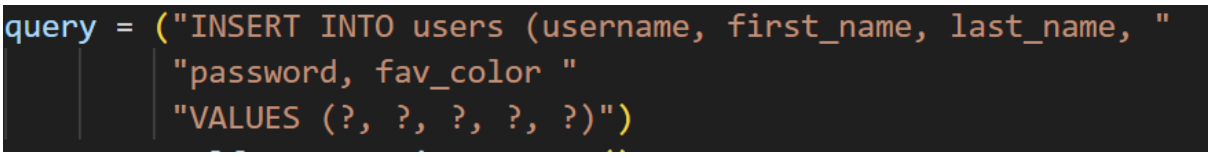


4. Same mistake in user put method

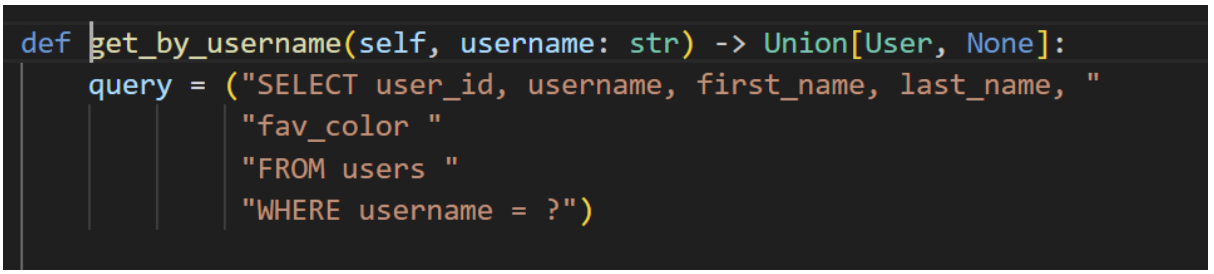


Backend bugs:

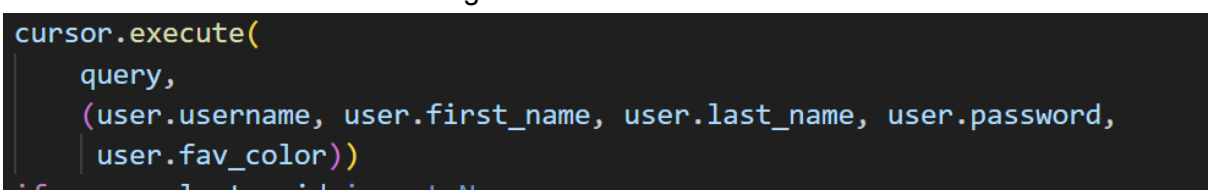
1. Missed bracket in db add query



2. No password from db to check it in login.



3. Password isn't hashed before storing in db



4. Mistake in word fav_color in update user method

```
query = ("UPDATE users "
         "SET username = ?, first_name = ?, last_name = ?, "
         "fav_colour = ? "
         "WHERE user_id = ?")
```

5. Hashed session token in db and session token that returns are not the same

```
def create(self, user_id) -> Session:
    token = token_urlsafe(CONFIG.token_length)
    hashed_token = sha256(token.encode("UTF-8")).hexdigest()
    session = Session(user_id=user_id)
    session.token = hashed_token
    self.db.add(session)
    session.token = token
    return session
```

6. No commit in user's update method

```
def update(user: User, token: str) -> User:
    with Transaction(session_token=token) as transaction:
        logic = UsersLogic(transaction)
        updated_user = logic.update(user)

    return updated_user
```

Second task

1. Creating new column in the database

```
ALTER TABLE users ADD COLUMN city TEXT NOT NULL DEFAULT '';
```

2. Adding country field into shema model

```
city = fields.String(required=True)
```

3. Same into model

```
            city: Union[str, None] = None,):

    self.user_id = user_id
    self.username = username
    self.first_name = first_name
    self.last_name = last_name
    self.password = password
    self.fav_color = fav_color
    self.city = city
```

4. Some changes in queries in UsersBb class

```
def update(self, user: User):
    query = ("UPDATE users "
             "SET username = ?, first_name = ?, last_name = ?, "
             "fav_color = ?, city = ? "
             "WHERE user_id = ?")
```

5. Changes into ts type of user

```
export interface User {
    userId?: number;
    username: string;
    firstName: string;
    lastName: string;
    city: string;
    favColor: string;
    password?: string;
}
```

6. Creating this field in signup page and in profile page

```
const [user, setUser] = useState<User>({
    firstName: "",
    lastName: "",
    username: "",
    city: "",
    password: "",
    favColor: "",
});
const [errors, setErrors] = useState<User>({
    firstName: "",
    lastName: "",
    username: "",
    city: "",
    password: "",
    favColor: "",
});
```

7. To make users that were registered before this field was added, I created redirection to proafter they login into application

```
let gotoProfile = "/profile";
```

```
const onSuccess = (sessionData: SessionData): void => {
    setWorking(false);
    loginAC.current = null;
    session.updateSessionData(sessionData);
    console.log(sessionData);
    if (!sessionData.city) {
        navigate(gotoProfile)
    } else {
        navigate(gotoAfterLogin, { replace: replaceLocationInRouter });
    }
};
```

Third task:

1. To make a reset password page I created a component on the frontend part with two input fields: old password and new password.

```tsx
export const ResetPassword = (): JSX.Element => {
    const [passwords, setPasswords] = useState<Passwords>({
        oldPassword: "",
        newPassword: "",
    });
    const [errors, setErrors] = useState<Passwords>({
        oldPassword: "",
        newPassword: "",
    });

    const [working, setWorking] = useState<boolean>(false);
    const [toastProps, setToastProps] = useState<ToastPropsData | null>(null);
    const passwordsAC = useRef<AbortController | null>(null);
    const navigate = useNavigate();

    let gotoHomePage = "/user";
```

2. When a user clicks submit it goes to the backend and if the old password is correct then it changes the old password to the new password. If the new password and old password are equal, it raises an exception.

```python
class ResetPasswords(MethodView):
    def post(self):
        try:
            if not request.is_json:
                raise Exception("Request is not JSON")
            token = get_session_token()
            user = get(token)
            json_data = request.data.decode("utf-8")
            schema = PasswordsSchema(external_casing=Casing.CAMEL)
            passwords: Passwords = cast(Passwords, schema.loads(json_data))
            reset_passwords(user, token, passwords)

            return success_response(data={"message": "Password reseted successfully"},
                                    convert_keys_to_camel=True)

        except Exception as e:
            return error_reponse(e)
```

```python
class ResetPasswords(MethodView):
    def post(self):
        try:
            if not request.is_json:
                raise Exception("Request is not JSON")
            token = get_session_token()
            user = get(token)
            json_data = request.data.decode("utf-8")
            schema = PasswordsSchema(external_casing=Casing.CAMEL)
            passwords: Passwords = cast(Passwords, schema.loads(json_data))
            reset_passwords(user, token, passwords)

            return success_response(data={"message": "Password reseted successfully"},
                                    convert_keys_to_camel=True)

        except Exception as e:
            return error_reponse(e)
```

Fourth task

1. Password hashing
2. Changing database from sqlite to postgres/mysql
3. Testing
4. More validation in models
5. logging