Simple Authentication

Frank Walsh

Authentication

- *digital authentication:* where the confidence for user identity is established and presented via electronic methods
  - *Enrollment* – an individual applies to a credential service provider (CSP) to initiate the enrollment process. After successfully proving the applicant's identity, the CSP allows the applicant to become a subscriber.
  - *Authentication* – After becoming a subscriber, the user receives an authenticator e.g., a token and credentials, such as a user name.
  - He or she is then permitted to perform online transactions within an authenticated session with a relying party, where they must provide proof that he or she possesses one or more authenticators.
  - *Life-cycle maintenance* – the CSP is charged with the task of maintaining the user's credential of the course of its lifetime, while the subscriber is responsible for maintaining his or her authenticator(s)

Authentication

- For Pacemaker lab, we'll just focus on Authentication aspect.
- New requirement
  - Users must authenticate(log in) using their username and Password
  - Administrator users can do everything, Standard uses can only add activities/locations. They can only view data

Pacemaker API

- Create a log in/authentication method:
- Define a new field in the API (currentUser). We can use this to record the currentUser logged in.

```
Optional<User> currentUser;

// simplified login method
public boolean login(String email, String password) {
Optional<User> user = Optional.fromNullable(emailIndex.get(email));
if (user.isPresent() && user.get().password.equals(password)){
currentUser = user;
log.info(currentUser.get().email + " logged in...");
return true;
}
return false;
}
```

Pacemaker API

- Also add a log out method to the API:

```
public void logout() {

        if (currentUser.isPresent()){
           log.info(currentUser.get().firstName + " logged out...");
           currentUser = null;
        }
}
```

User Class

- Add a new field: **role**

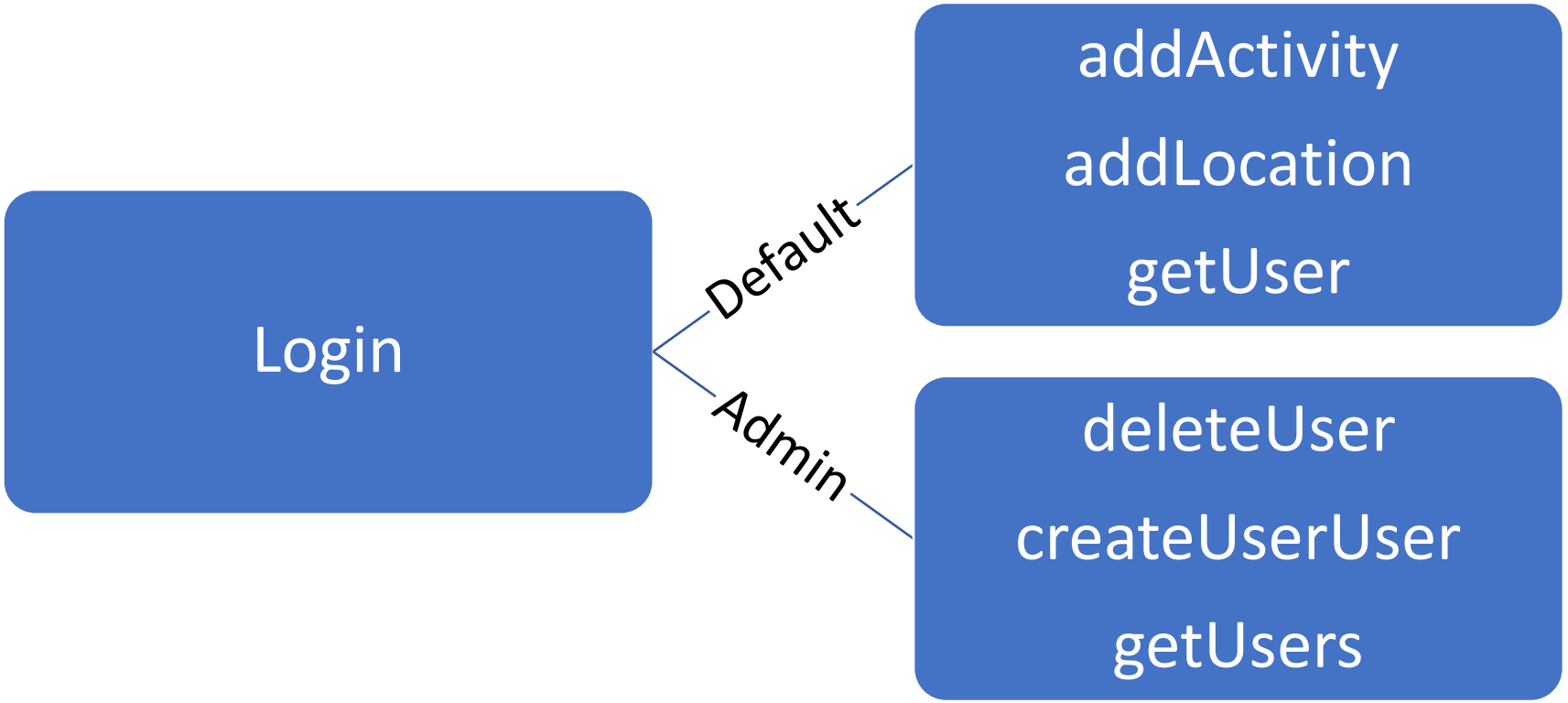
…

```
Public Optional<String> role role;
```

…

Menu

- Change the CLI interface so that you are forced to login
  - Use Sub Menus:



Cliché SubMenus

- Cliche Subshells offer a way to navigate through tree-like structures.
- Main menu must implement `ShellDependent`
  - Must give subshell a reference to the parent shell

```
public class Main implements ShellDependent {

private Shell theShell;

public void cliSetShell(Shell theShell) {
        this.theShell = theShell;
}
…

ShellFactory.createSubshell(user.firstName, theShell, "Admin",
adminMenu).commandLoop();
```

Menu Class

- Update the menu class to just contain the logIn command

```
@Command(description = "Log in")
public void logIn(@Param(name = "user name") String userName, @Param(name = "password") String pass)
throws IOException {

if (paceApi.login(userName, pass)) {
  User user = paceApi.currentUser;
  System.out.println("You are logged in as " + user.email);
  Optional<String> role = Optional.fromNullable(user.role);
  if (role.isPresent() && role.get().equals(ADMIN)) {
    adminMenu = new AdminMenu(paceApi, user.firstName);
    ShellFactory.createSubshell(user.firstName, theShell, "Admin", adminMenu).commandLoop();
  } else {
    defaultMenu = new DefaultMenu(paceApi, user);
    ShellFactory.createSubshell(user.firstName, theShell, "Default", defaultMenu).commandLoop();
  }
} else
    System.out.println("Unknown username/password.");
}
```

Default Menu

- Provides the default commands for the logged in user:

```
public class DefaultMenu {

  private String name;
  private User user;
  private PacemakerAPI paceApi;

  public DefaultMenu(PacemakerAPI paceApi, User user) {
    this.paceApi = paceApi;
    this.setName(user.firstName);
    this.user=user;
  }
```

Default Menu

- User ID already know from log in – use it in addActivity

```
@Command(description = "Add an activity")
public void addActivity(@Param(name = "type") String type,
                        @Param(name = "location") String location,
                        @Param(name = "distance") double distance) {
        paceApi.createActivity(user.id, type, location, distance);
}

@Command(description = "Add Location to an activity")
public void addLocation(@Param(name = "activity-id") Long id,
                        @Param(name = "latitude") float latitude,
                        @Param(name = "longitude") float longitude) {
        Optional<Activity> activity = Optional.fromNullable(paceApi.getActivity(id));
        if (activity.isPresent()) {
                paceApi.addLocation(activity.get().id, latitude, longitude);
        }
}
```

Admin Menu

- All the commands from the original Menu.

```
public class AdminMenu implements ShellDependent  {

        private String name;
        private PacemakerAPI paceApi;


        public AdminMenu(PacemakerAPI paceApi, String userName) {
                this.paceApi = paceApi;
                this.setName(userName);
        }
…
```