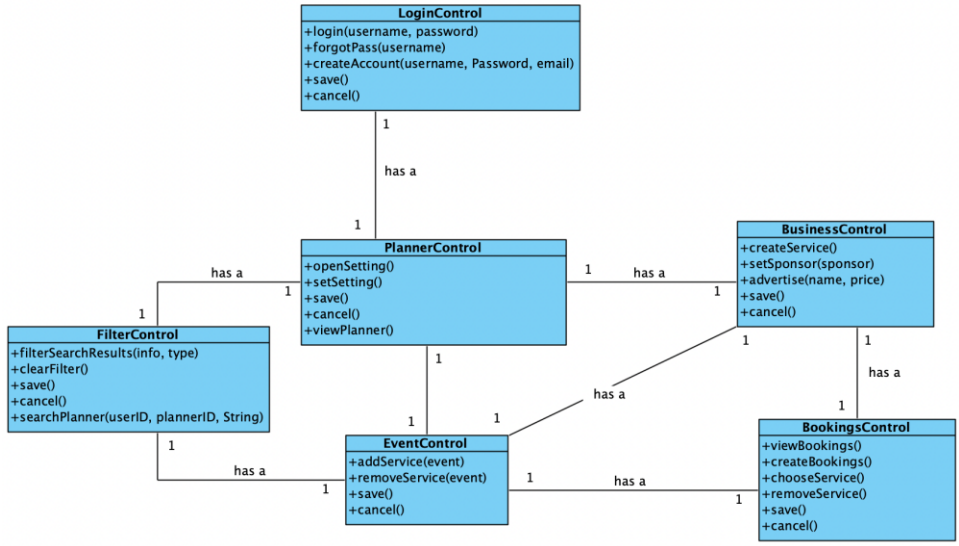
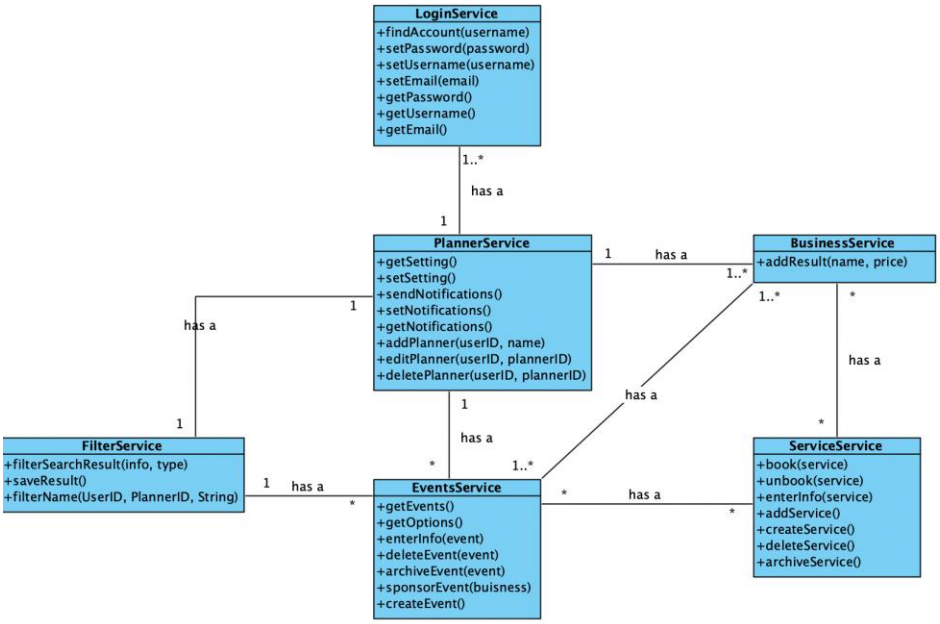
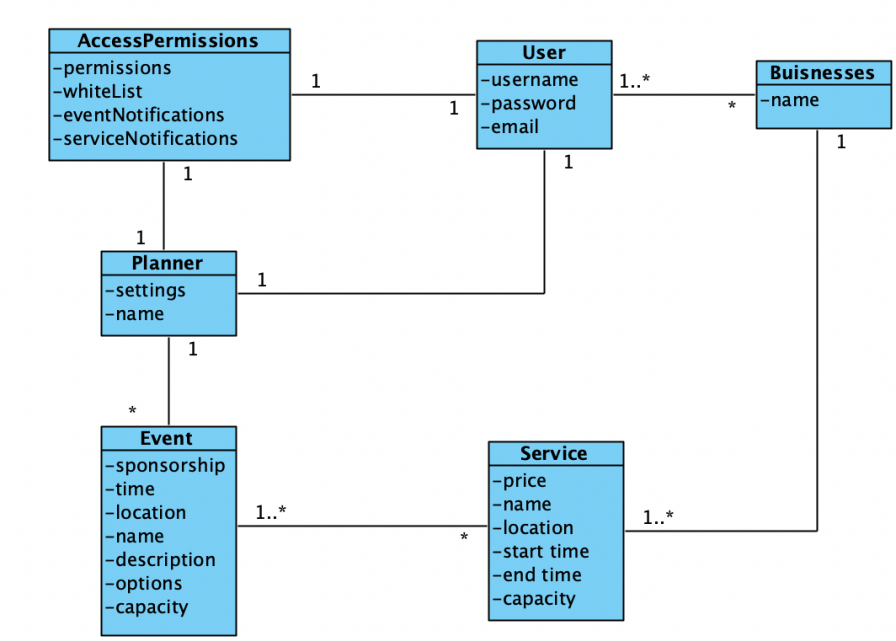
UI Design Diagram



System Design Diagram

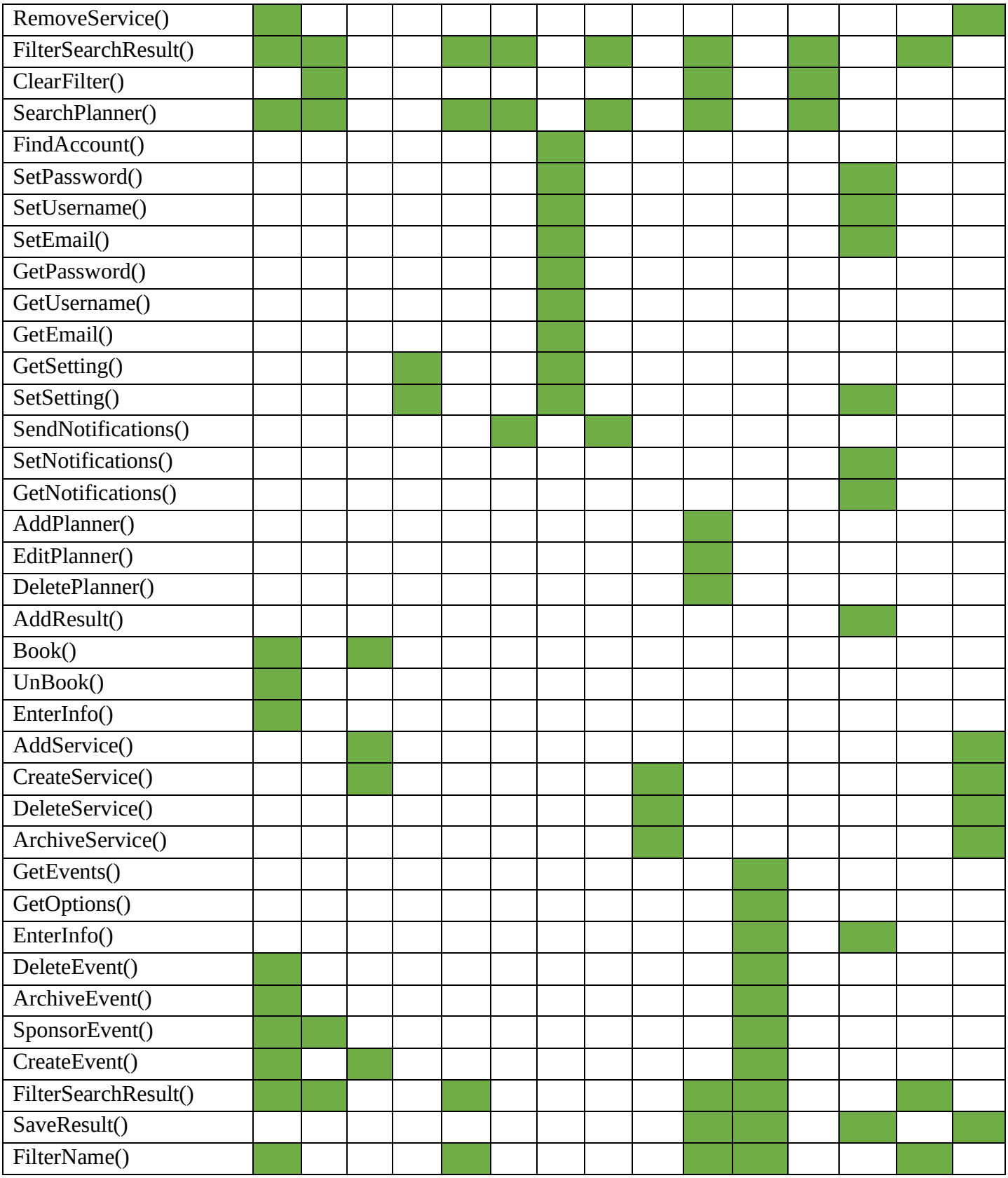


Persistence Design Diagram

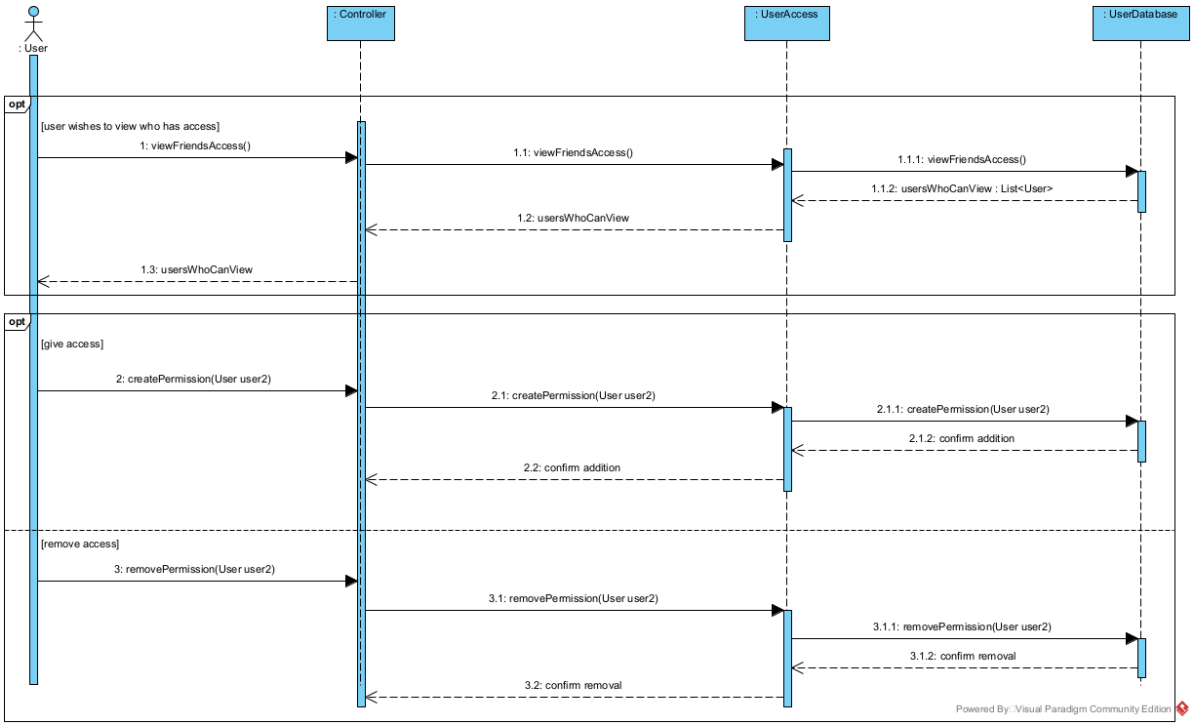


Testing Coverage

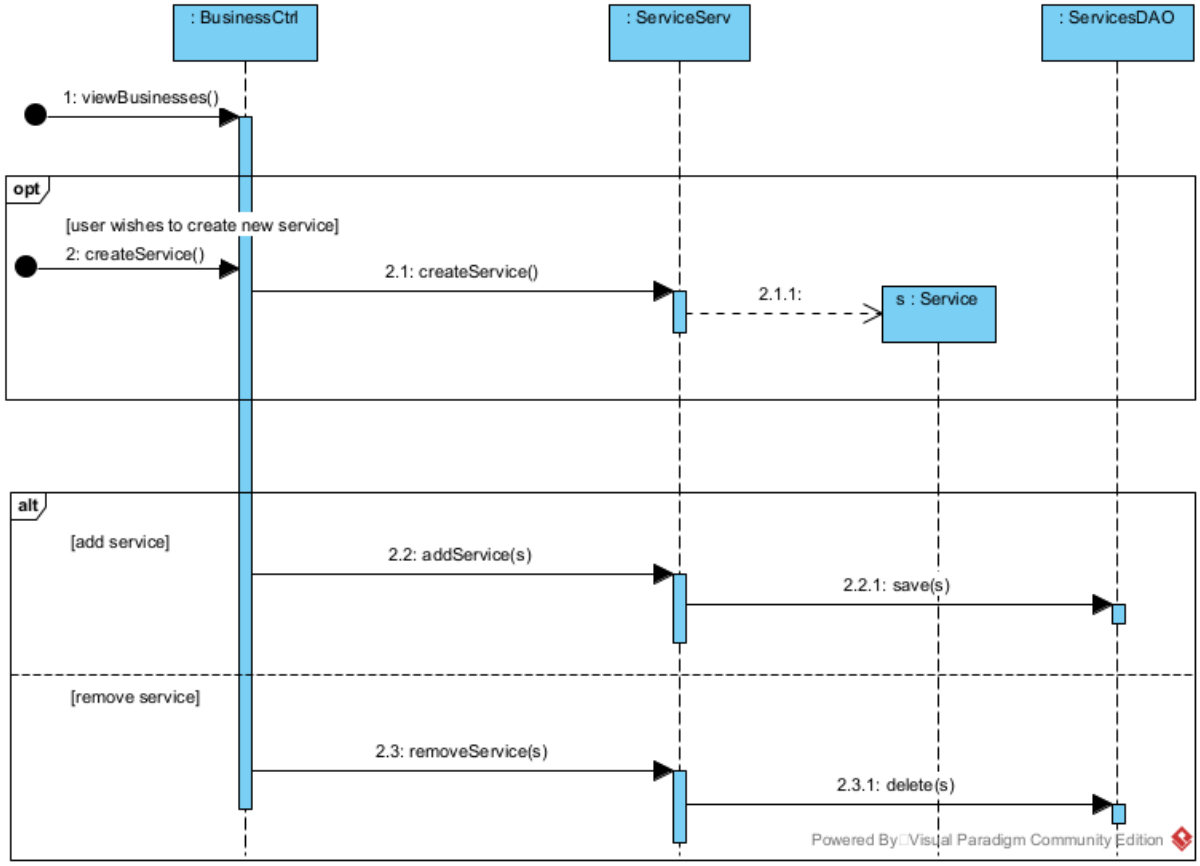
Requirements -> Functions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Login()							■								
ForgotPass()							■								
CreateAccount()							■								
Save()										■			■		
Cancel()										■					
OpenSetting()				■				■		■					
SetSetting()				■						■			■		
ViewPlanner()	■					■		■		■		■			
CreateService()			■						■						
SetSponsor()															■
Advertise()		■													■
ViewBookings()	■					■		■				■			
CreateBooking()	■		■												
ChooseBooking()	■							■							
ChooseService()	■		■						■						■
RemoveService()	■								■						■
AddService()	■		■						■						■


RemoveService()
FilterSearchResult()
ClearFilter()
SearchPlanner()
FindAccount()
SetPassword()
SetUsername()
SetEmail()
GetPassword()
GetUsername()
GetEmail()
GetSetting()
SetSetting()
SendNotifications()
SetNotifications()
GetNotifications()
AddPlanner()
EditPlanner()
DeletePlanner()
AddResult()
Book()
UnBook()
EnterInfo()
AddService()
CreateService()
DeleteService()
ArchiveService()
GetEvents()
GetOptions()
EnterInfo()
DeleteEvent()
ArchiveEvent()
SponsorEvent()
CreateEvent()
FilterSearchResult()
SaveResult()
FilterName()

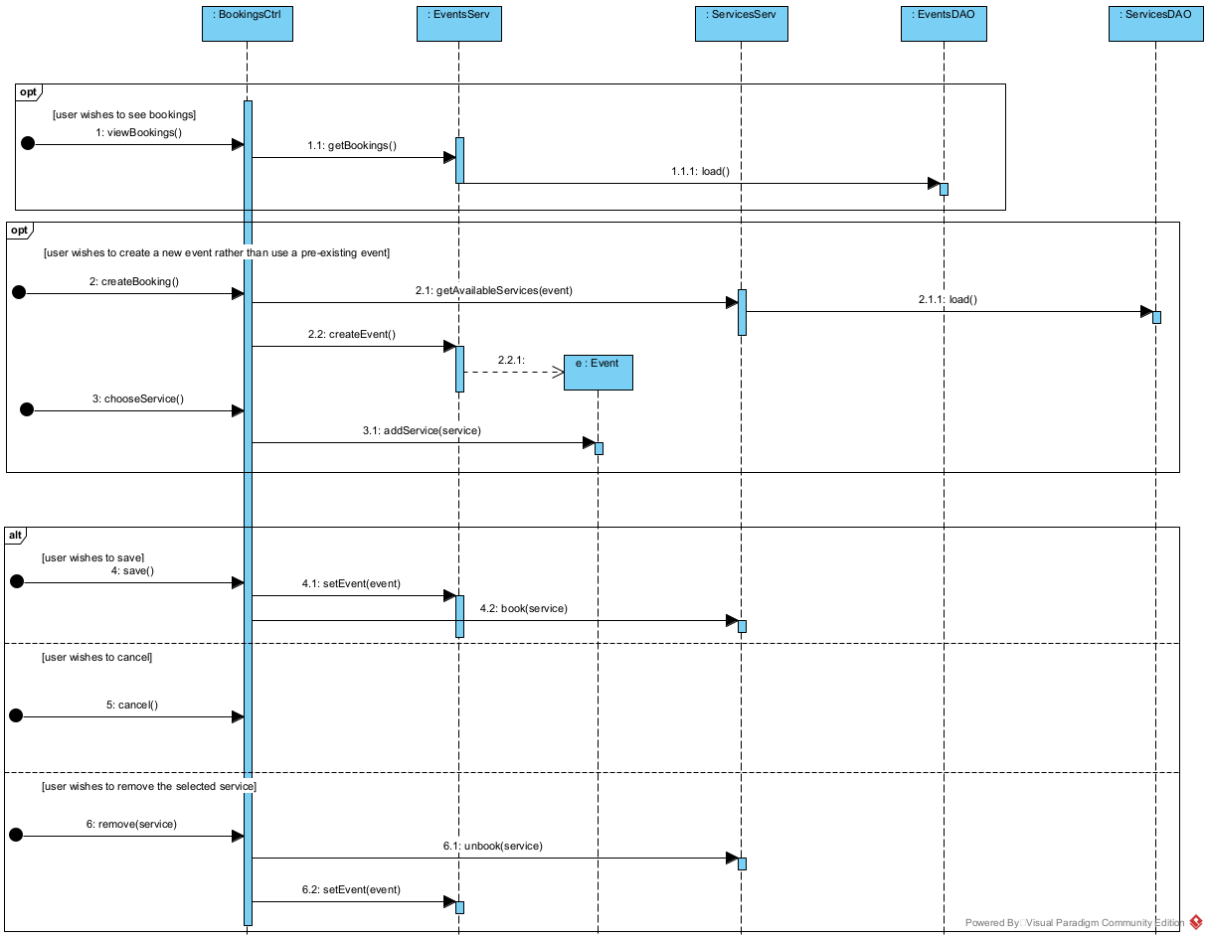
SD Manage Friends Access – Benjamin



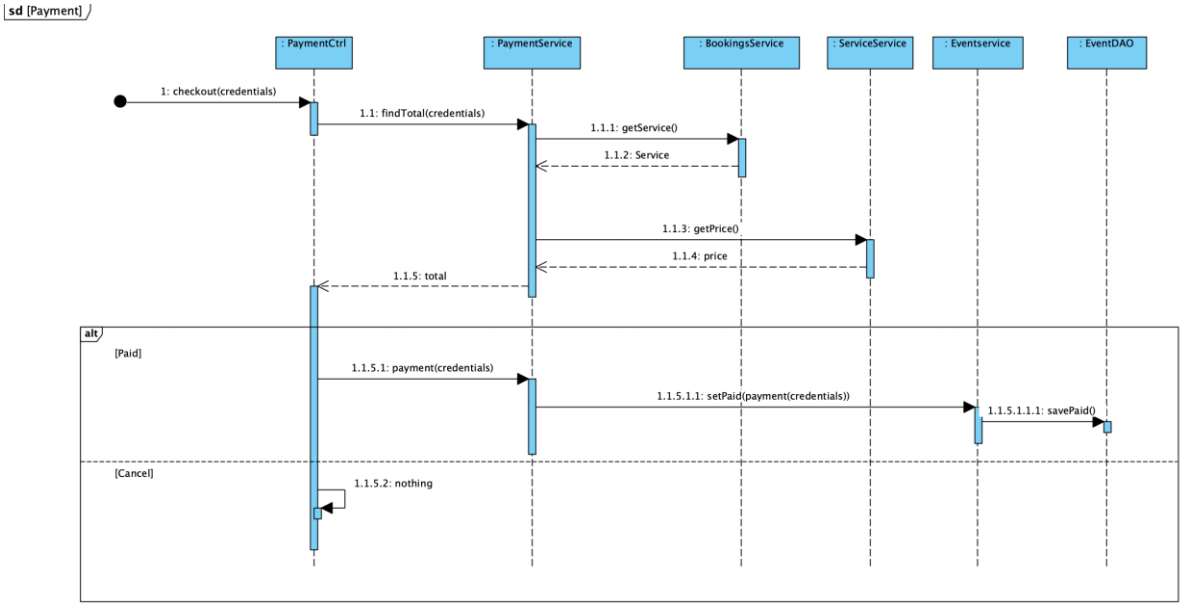
SD Manage Service - Benjamin



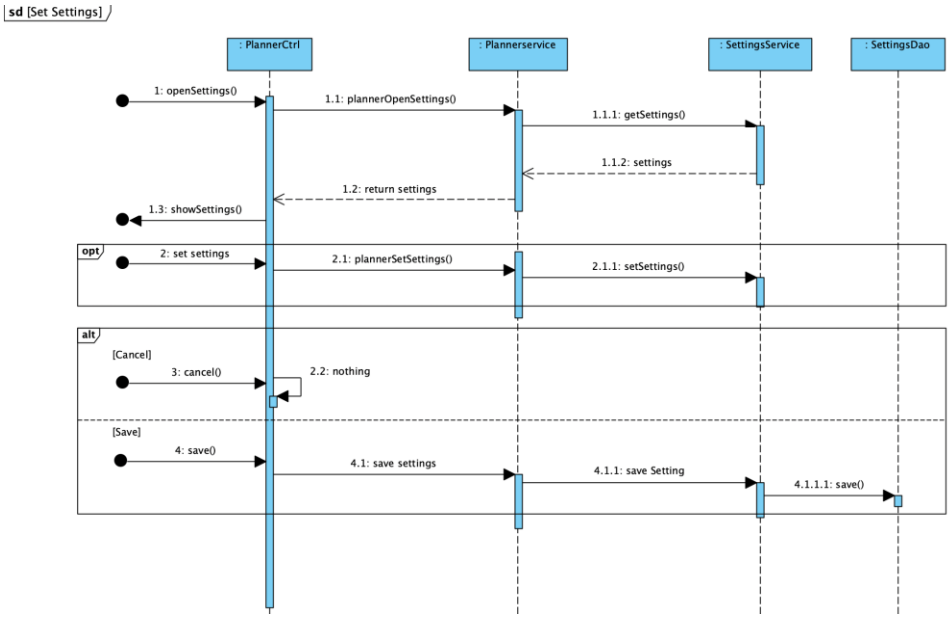
SD Manage Booking - Benjamin



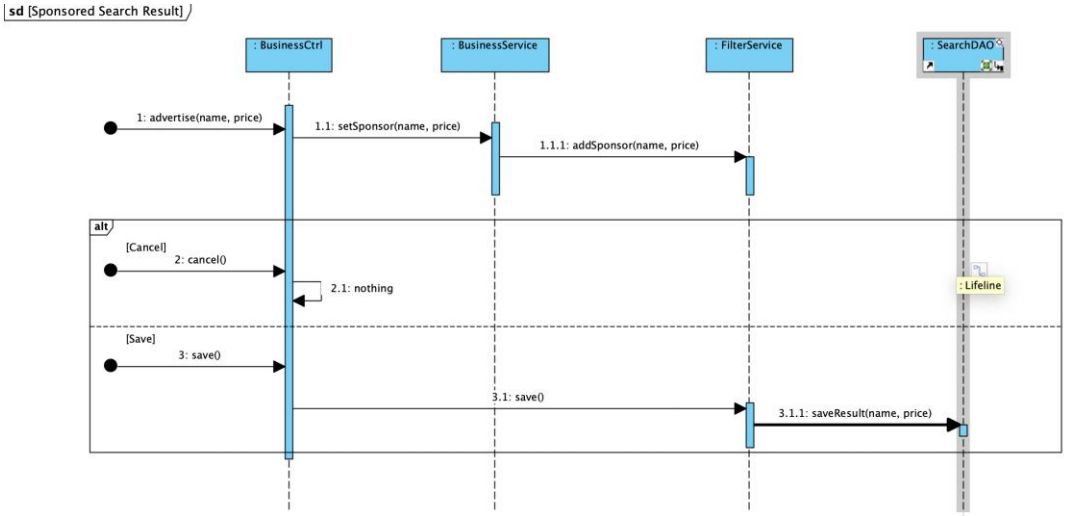
Payment – Yi Ding



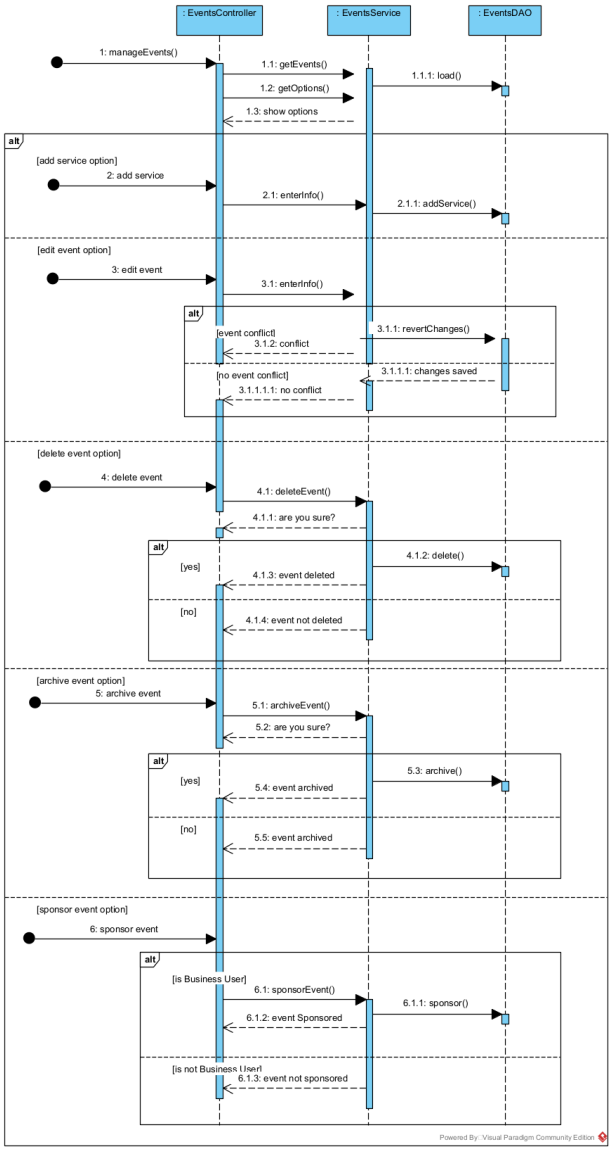
Settings – Yi Ding



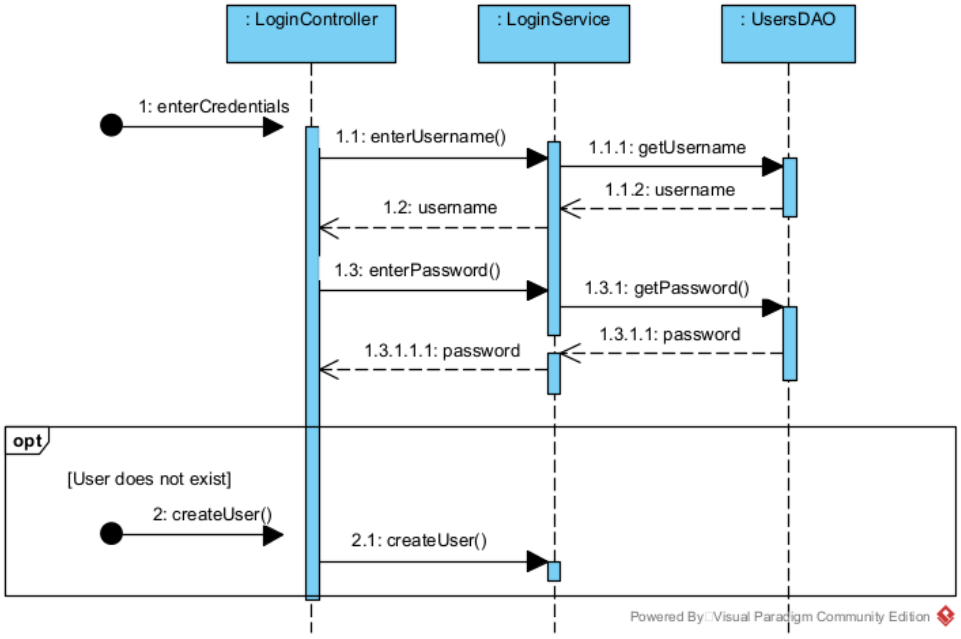
Sponsored Search – Yi Ding



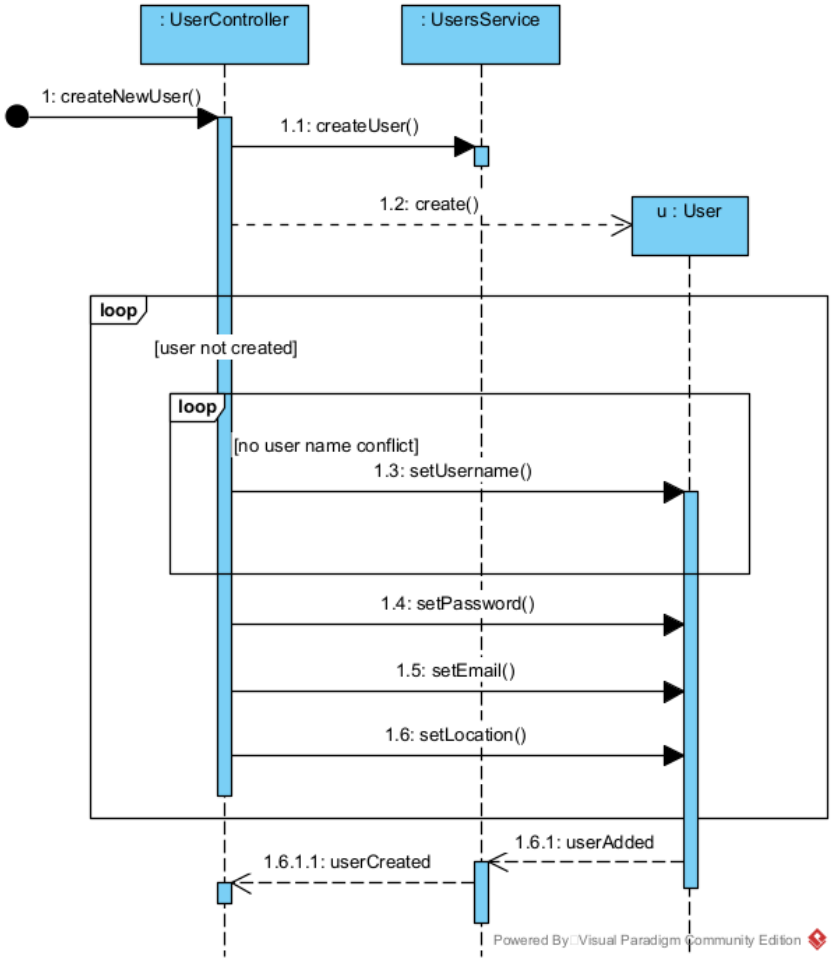
Manage Events – Bryce



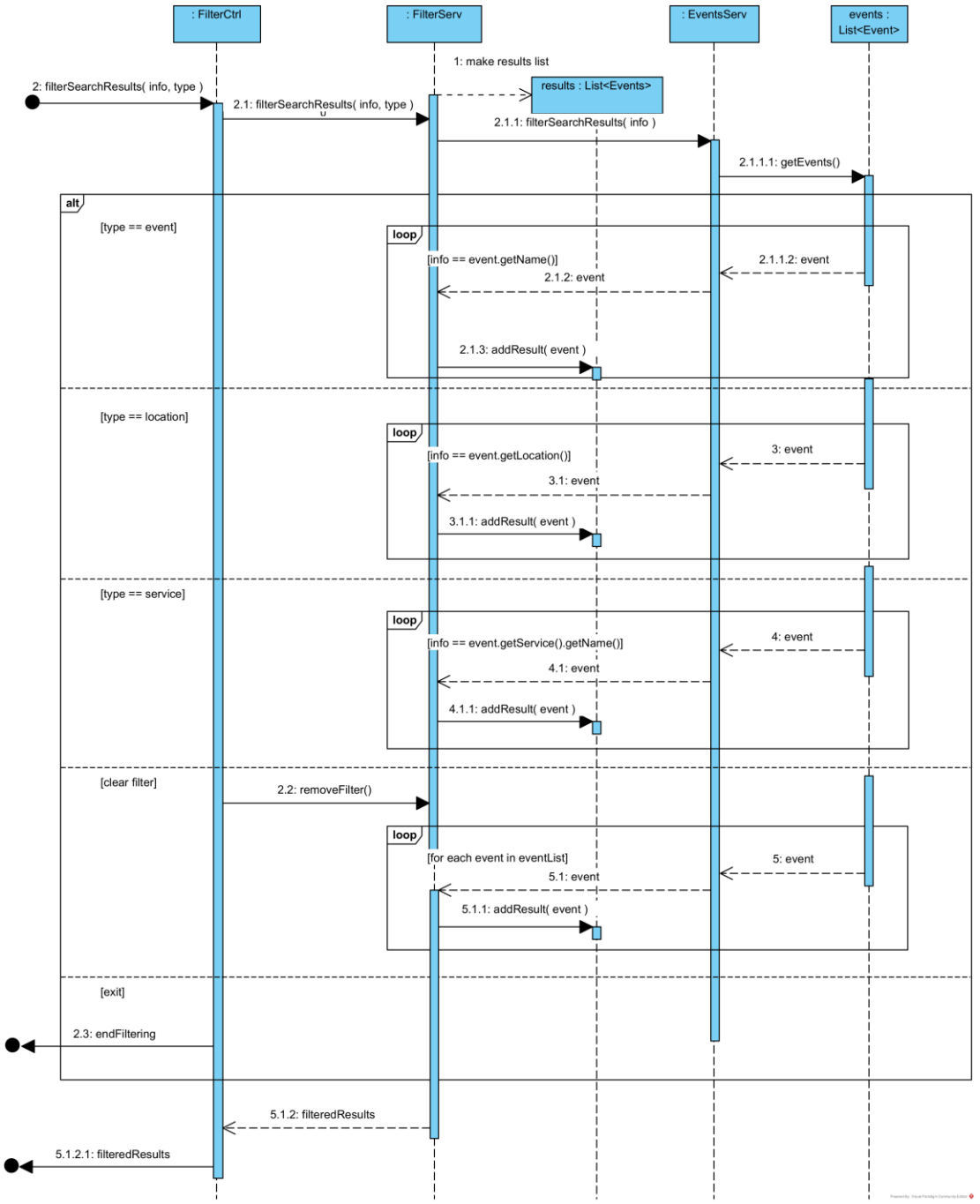
Login – Bryce


: LoginController
: LoginService
: UsersDAO
1: enterCredentials
1.1: enterUsername()
1.1.1: getUsername
1.1.2: username
1.2: username
1.3: enterPassword()
1.3.1: getPassword()
1.3.1.1: password
1.3.1.1.1: password
opt
[User does not exist]
2: createUser()
2.1: createUser()
Powered By: Visual Paradigm Community Edition

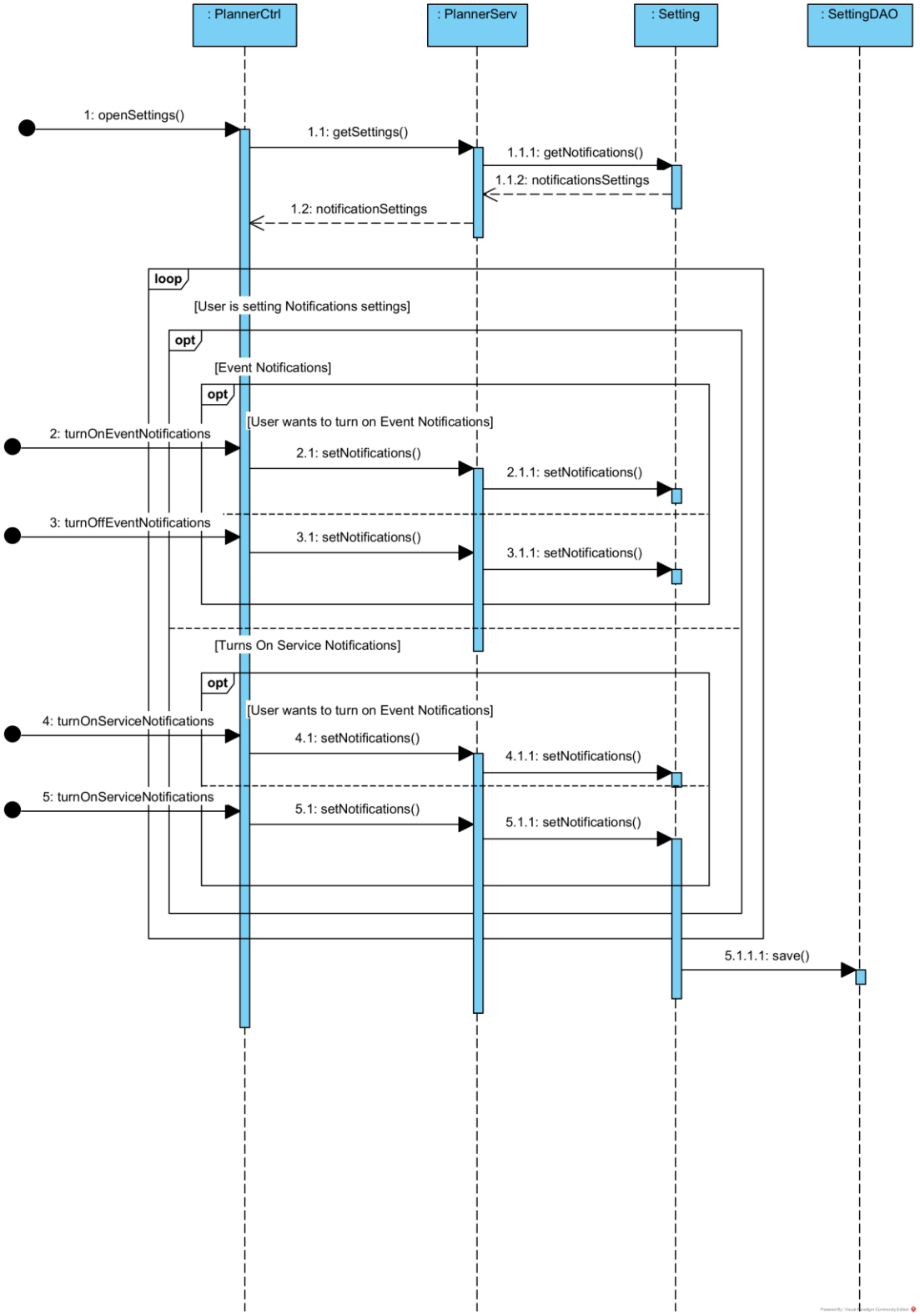
Create new user – Bryce



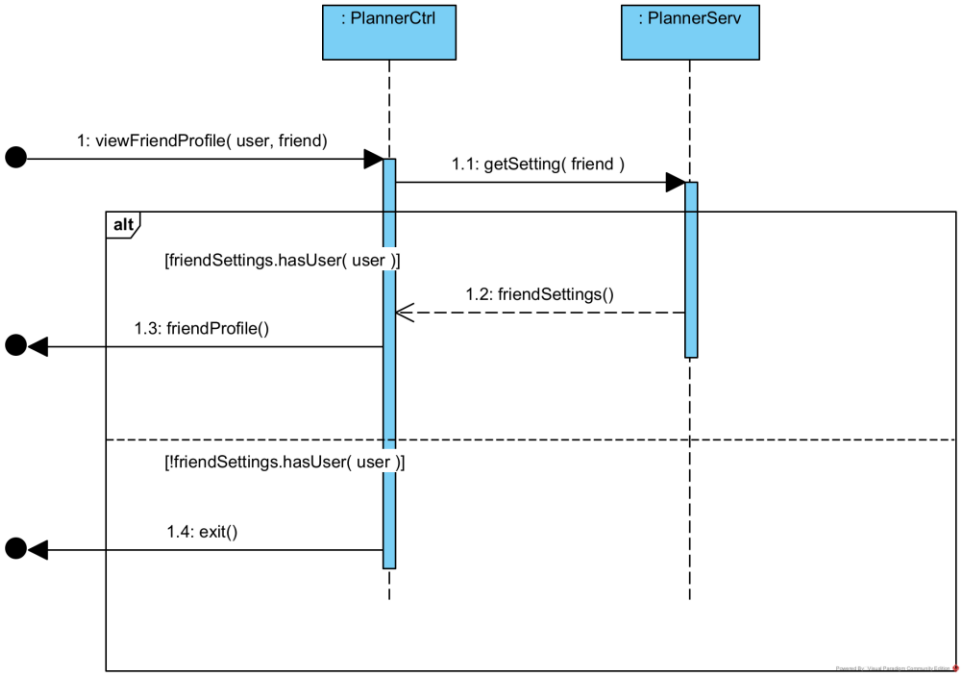
Filter Search Results – Jason



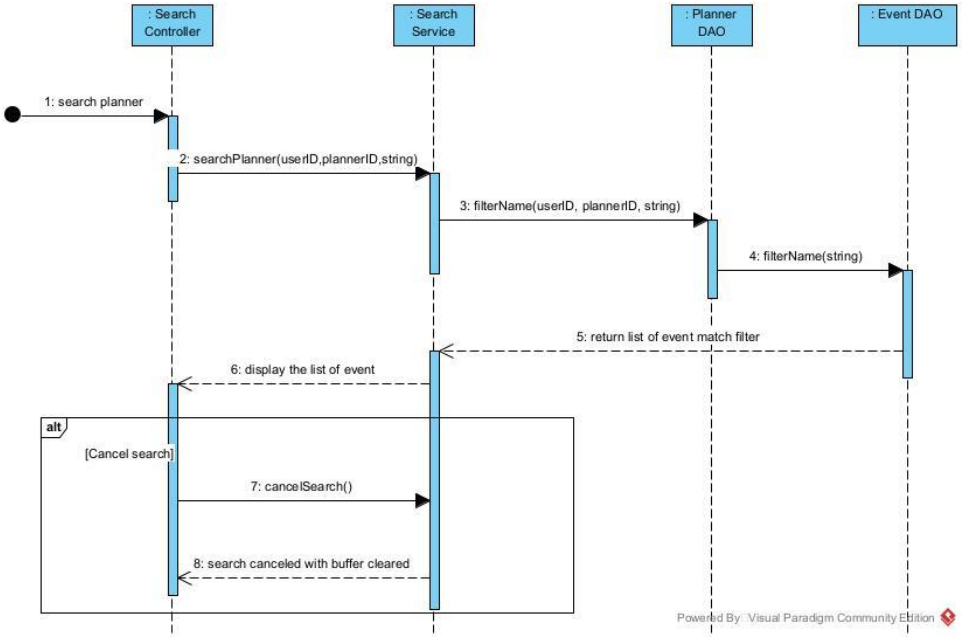
Set Notifications – Jason



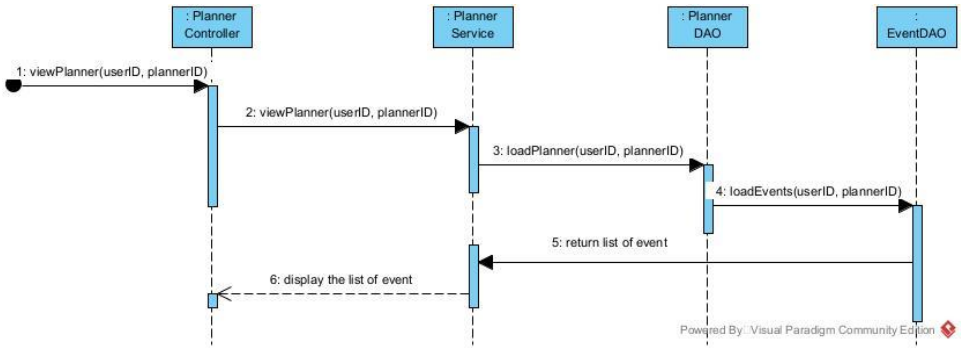
ViewProfiles – Jason



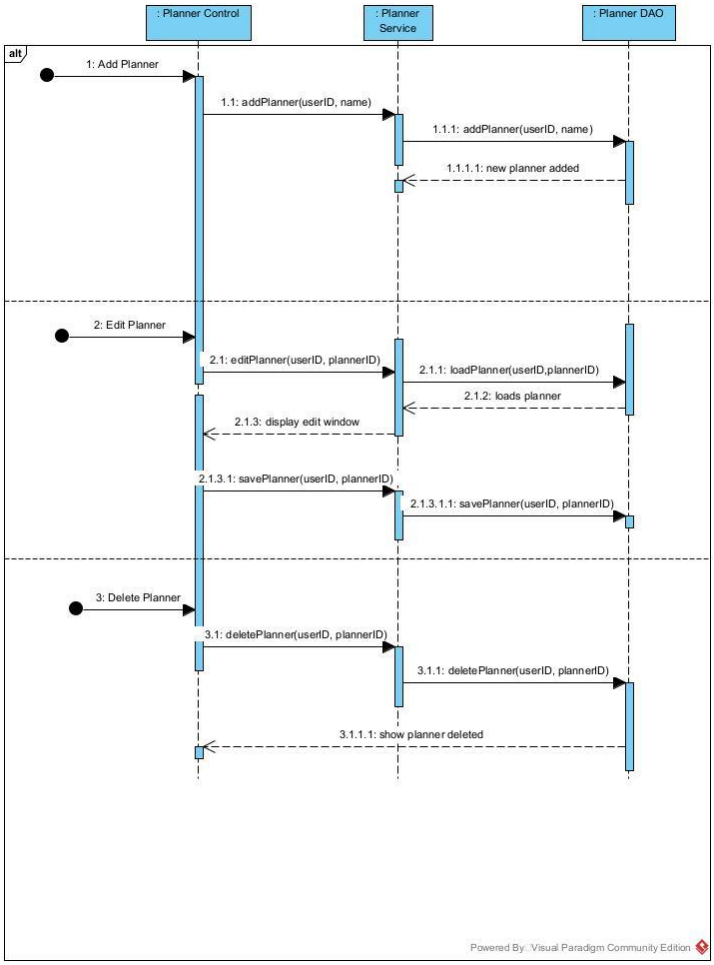
Search – Yutai



ViewPlanner – Yutai



Manage Planner – Yutai



GRASP Patterns

Pattern: Information Experts

These classes are responsible for the management of other classes. The Planner allows interaction between the User and Event classes. The Event's job stores the relation between Users, Dates, and Services. The Planner's job is to store the relation between Users, Events, and Businesses. The Planner and Event classes are Information Experts, as they hold information on all the other classes. We plan to implement them so that each Planner will know what Events it has, and each Event will know the Users and Services connected to it.

Pattern: Creator

The main Creator in this software is the Planner, which creates Events, Services, Businesses, Users, and Permissions. Additionally, we have some minor creators:

- Businesses create Services.
- Services create Bookings
- Events and Business

Pattern: Low Coupling & High Cohesion

Originally, Login and Registration functions were directly connected to the planner, but we decided to split the Login function into its own class to maintain Lower Coupling and High Cohesion. Most of our classes are self-contained, and don't require explicit knowledge of other classes to work as intended.

High coupling exists between our Planner, Event and Business classes. We have set it up this way so that we could decouple the User class from Events and Businesses and increase cohesion by separating Users from the Events and Services. The Planner has a list of Events, which also has a list of Services. Planners are directly connected to Users, so each member of Planner is, by proxy, a member of User. Both Events and Services cannot function without the existence of the User class. However, the other classes seem to be coupled only for the sake of a particular Use Case, so I believe the application has low coupling overall.

Pattern: Controller

The Controllers for this app will be the UI classes, which handle and react to input for different use cases. We chose to have controllers for Login, Planners, Filters, Business, Bookings, and Events. Bookings are closely tied to Services, so we set up a Controller to help visualize the ways in which booking works.

Pattern: Polymorphism

Polymorphism is demonstrated through our UI.

Pattern: Dynamic Binding

This exists because the UI all have createAndShowGUI() functions and extend JFrame through different implementations.

Pattern: Pure Fabrication

We have multiple classes that demonstrate this pattern. The first is the Event class, which represents the connection between Users, dates, locations, and Services. The class that exemplifies this GRASP pattern is the Event class, which doesn't really represent anything that exists in reality. Event represents the relationship between Users and Dates.

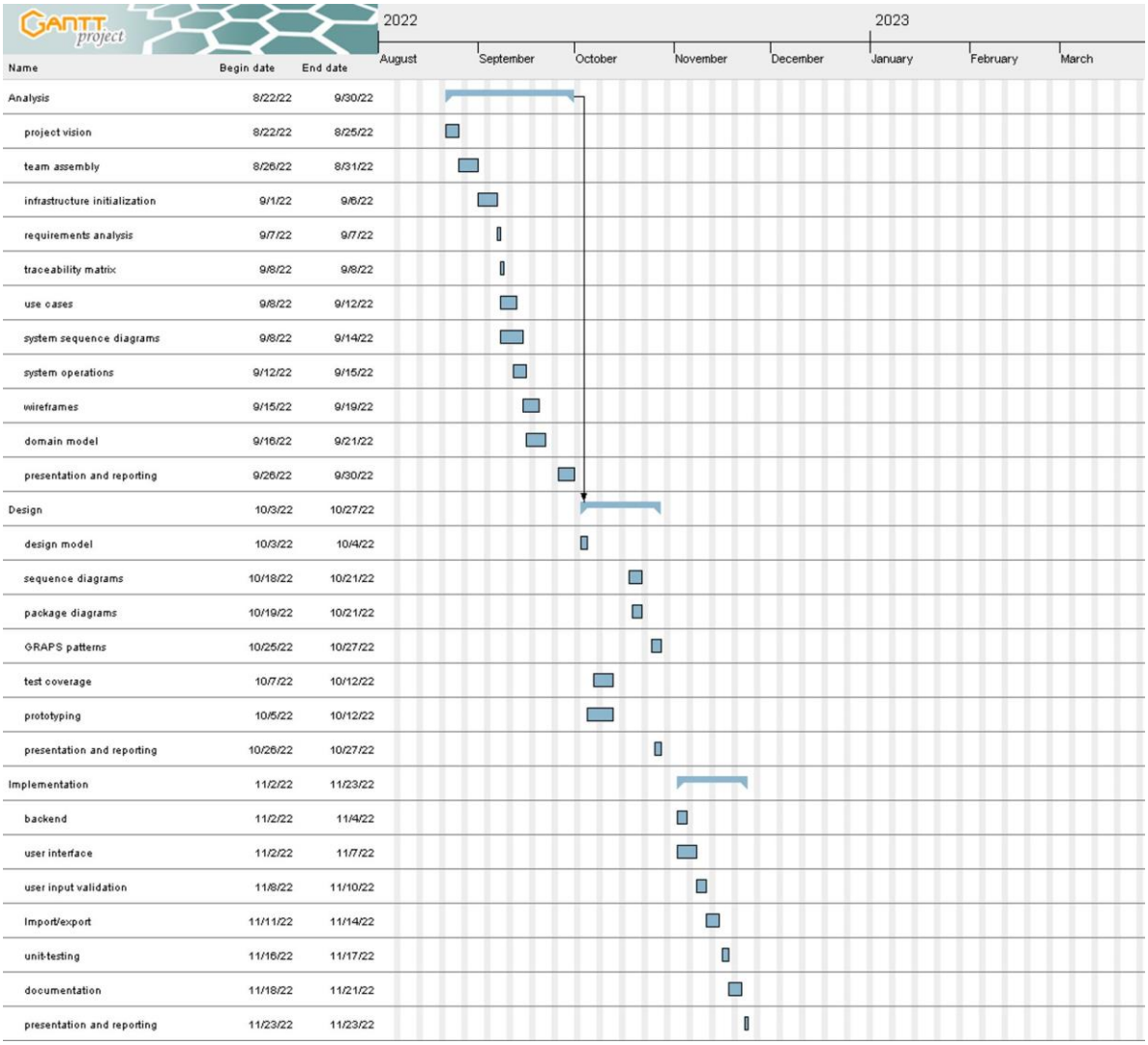
Pattern: Indirection

Indirection is used many times throughout this software. One example is the Planner class which mediates between the Users and Events. There's also a lot of indirection through the UI, which handles User input and directs it to the other classes.

Pattern: Law of Demeter

This law is satisfied in multiple ways. Firstly, by the indirection between Users and Services, and between Services and Events.

Updated Gantt



Name	Begin date	End date
Analysis	8/22/22	9/30/22
project vision	8/22/22	8/25/22
team assembly	8/26/22	8/31/22
infrastructure initialization	9/1/22	9/6/22
requirements analysis	9/7/22	9/7/22
traceability matrix	9/8/22	9/8/22
use cases	9/8/22	9/12/22
system sequence diagrams	9/8/22	9/14/22
system operations	9/12/22	9/15/22
wireframes	9/15/22	9/19/22
domain model	9/16/22	9/21/22
presentation and reporting	9/26/22	9/30/22
Design	10/3/22	10/27/22
design model	10/3/22	10/4/22
sequence diagrams	10/18/22	10/21/22
package diagrams	10/19/22	10/21/22
GRAPS patterns	10/25/22	10/27/22
test coverage	10/7/22	10/12/22
prototyping	10/5/22	10/12/22
presentation and reporting	10/26/22	10/27/22
Implementation	11/2/22	11/23/22
backend	11/2/22	11/4/22
user interface	11/2/22	11/7/22
user input validation	11/8/22	11/10/22
Import/export	11/11/22	11/14/22
unit-testing	11/16/22	11/17/22
documentation	11/18/22	11/21/22
presentation and reporting	11/23/22	11/23/22

GitHub: https://potatofishes.github.io/MosesTravelGuide/

Check out our website and progress tracking tab to see hours and trello