C.A.R. Vision

Revision History

Version	Date	Description	Author
Iteration 2 Draft	Mar 27, 2019	Draft to be presented in Iteration 2.	Andrew Case
Inception Draft	Jan 22, 2019	First draft. To be refined during elaboration prior to iteration 1.	Andrew Case
Iteration 1 Draft	Feb 9, 2019	Draft to be presented in iteration 1.	Andrew Case Maggie Burton

Vision Statement

This system provides a simple, efficient, robust application for the provision and acquisition of a wide range of rental cars, promoting the reuse of vehicles.

Business Opportunity

Currently, the vehicle industry is riddled with problems pertaining to recalls and maintenance. This process promotes more complexity and involvement of a driver's busy lifestyle, not to mention the price associated with each repair. Throughout the life of a vehicle, general repairs can be easily avoided through the rental of vehicles. Renting a vehicle also allows drivers to operate exotic or newer model cars they would not otherwise get to use.

Summary of System Features

- Book a reservation for a specific vehicle and cancel a reservation, with real-time transactions and updates to the system
- Payment Authorization
- Option to purchase cars, with the sale being conducted by a representative of the company
- View catalog of vehicles, and allow administrators to update
- Option to purchase insurance from rental company or use personal insurance when renting cars
- Efficient billing and payroll system for company
- Customer assistance email service

Iteration 2

Project manager	
Project dates	Jan 17, 2019 - Mar 27, 2019
Completion	0%
Tasks	23
Resources	6

Tasks

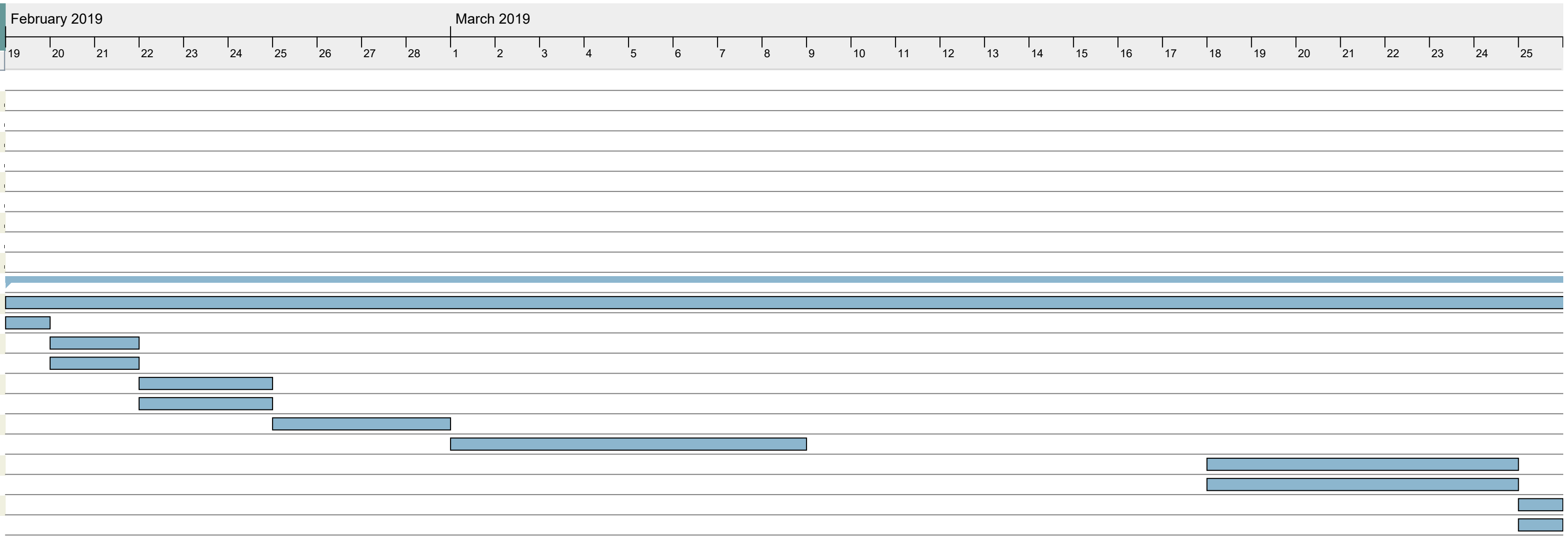
Name	Begin date	End date
Iteration 1	1/17/19	2/12/19
Timecards	1/17/19	2/11/19
Teamwork Plan	1/17/19	2/12/19
Project Vision	1/17/19	1/22/19
Requirements	1/23/19	2/9/19
Use Cases	1/24/19	2/9/19
SSDs	1/27/19	2/9/19
UI Wireframes	2/7/19	2/9/19
Domain Model	2/9/19	2/9/19
Actors	2/9/19	2/10/19
Iteration 2	2/19/19	3/26/19
Timecards	2/19/19	3/26/19
Teamwork Plan	2/19/19	2/19/19
Updated SSDs	2/20/19	2/21/19
Updated Domain Model	2/20/19	2/21/19
Updated UI Wireframes	2/22/19	2/24/19
Design Class Model	2/22/19	2/24/19
Test Coverage Plan	2/25/19	2/28/19
Design for Demo Code	3/1/19	3/8/19
Code Implementation	3/18/19	3/24/19
Quality Control for Iteration 1 and 2	3/18/19	3/24/19
Final Testing	3/25/19	3/25/19
Video	3/25/19	3/25/19

Resources

Name	Default role
Andrew Case	Project Lead
Matthew Darby	Coder
Mark Du	Designer
Maggie Burton	Quality Control
Stevie Damrel	Coder
Weston Straw	Designer

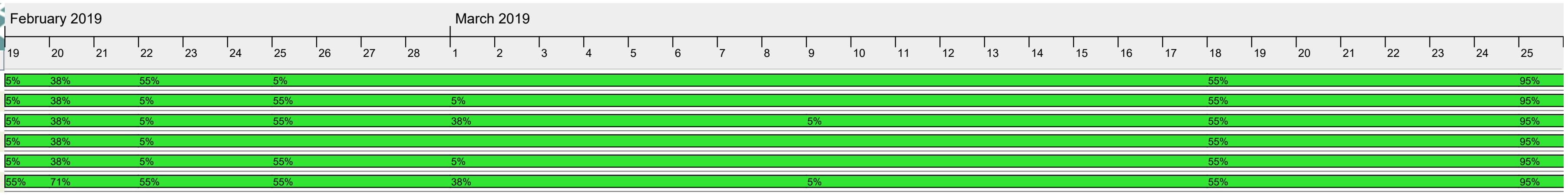
Gantt Chart

Name	Begin date	End date
Iteration 1	1/17/19	2/12/19
Timecards	1/17/19	2/11/19
Teamwork Plan	1/17/19	2/12/19
Project Vision	1/17/19	1/22/19
Requirements	1/23/19	2/9/19
Use Cases	1/24/19	2/9/19
SSDs	1/27/19	2/9/19
UI Wireframes	2/7/19	2/9/19
Domain Model	2/9/19	2/9/19
Actors	2/9/19	2/10/19
Iteration 2	2/19/19	3/26/19
Timecards	2/19/19	3/26/19
Teamwork Plan	2/19/19	2/19/19
Updated SSDs	2/20/19	2/21/19
Updated Domain Model	2/20/19	2/21/19
Updated UI Wireframes	2/22/19	2/24/19
Design Class Model	2/22/19	2/24/19
Test Coverage Plan	2/25/19	2/28/19
Design for Demo Code	3/1/19	3/8/19
Code Implementation	3/18/19	3/24/19
Quality Control for Itera...	3/18/19	3/24/19
Final Testing	3/25/19	3/25/19
Video	3/25/19	3/25/19



Resources Chart

Name	Default role
Andrew Case	Project Lead
Matthew Darby	Coder
Mark Du	Designer
Maggie Burton	Quality Cont...
Stevie Damrel	Coder
Weston Straw	Designer



C.A.R - GRASP Implementations

- **Creator** – Classes that edit other objects have creator status
  - The RepresentativeController is the creator for *Car* objects
  - The AdminController is the creator for *User* objects
  - The UserController is the creator for *Rental* objects
- **Low Coupling** – classes communicate minimally with each other
  - *Cars* have no knowledge of *Users*
  - *Cars* and *Representatives* have no knowledge of *Receipts*
  - *Users* (Customers) can only view *Cars*
  - *Receipts* have no knowledge of *Cars* or *Representatives*
  - *Receipts* only show data from *Users* and *Rentals*
- **High Cohesion** – classes have narrow and specific responsibilities
  - Controllers take input and create appropriate objects
  - *User* (and subtypes), *Receipt*, *Rental*, *Car* store only data pertinent to the individual object
- **Controller** – handles input and object creations
  - *RepresentativeController* handles input for new *Car*
  - *AdminController* takes input for new *User*
  - *UserController* takes input for new *Rental*
- **Polymorphism** – Subclasses are implemented to expand functionality of Superclasses
  - *Representative* extends *User*; it is a specialized type of *User*, with broader access to data in the system
  - *Administrator* extends *Representative*; it is a subtype of *Representative*, who has clearance to edit and view larger amounts of data
- **Indirection** – Classes call other classes to accomplish tasks
  - User calls *UserController* to view *Cars*
  - Administrator calls *AdminController* to edit *Users*

C.A.R. Testing Coverage Plan

In order to ensure correctness in the C.A.R. app, we will use Unit Testing, implemented with jUnit, to test the Create Account, Login, Select Reservation, Personal Insurance Policy, Payment/Billing System, Maintain Payroll and Manage Personnel functions. These being the functions where a user, whether customer or administrator, could enter data that is invalid. We will run the following tests:

Create Account:

- All fields filled, through bounds checking
- Credit Card Number correct length and format, through bounds checking and passing bad data
- Account created upon button click, through action listeners
- Password meets criteria, through checking for required characters

Login:

- Username and Password correspond and are valid, through validating existence in database
- Login initiated upon button click, through action listeners

Select Reservation:

- Car is available for selected dates, through searching in all active reservations

Personal Insurance Policy:

- All fields filled, through bounds checking
- Insurance current, through validating expiration date is in the future

Payment/Billing Information:

- Administrator can edit bill but cannot delete, through entering null data

Maintain Payroll:

- Payment meets federal minimum wage, through testing invalid input
- Administrator cannot edit his/her own payroll, through bounds checking
- Hours logged does not exceed possible hours, through bounds checking

Manage Personnel

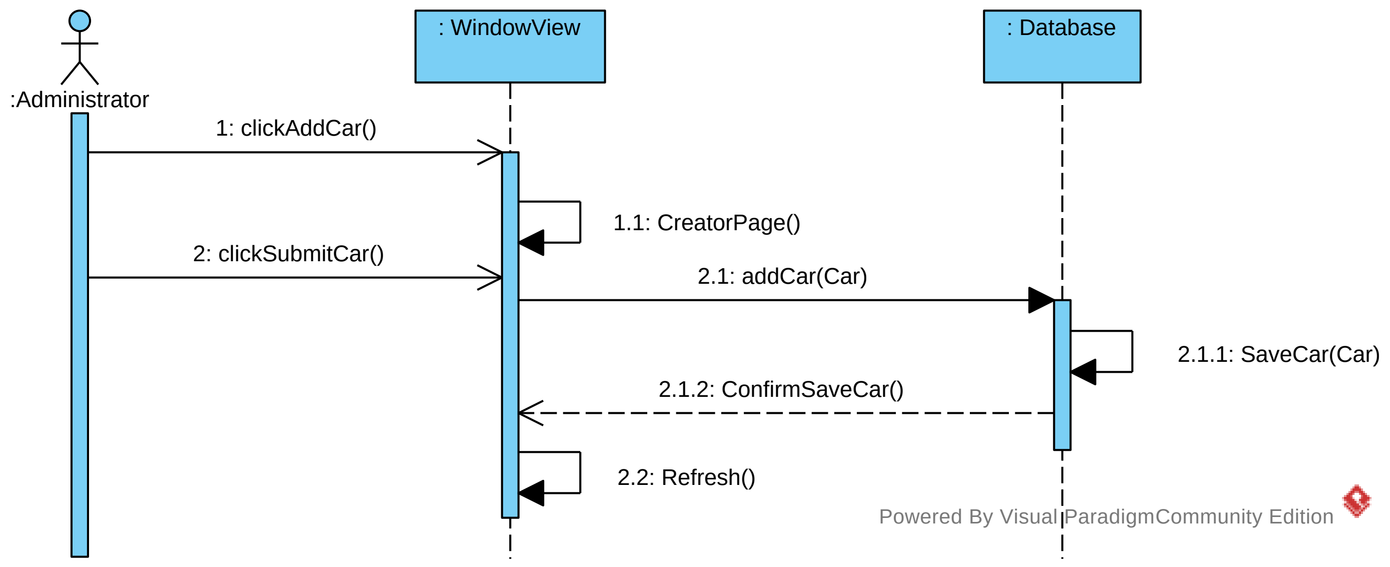
- Employee profiles cannot be duplicated, through testing invalid input
- New Employee profile must be complete, through bounds checking

Iteration 2 Suggested Points Distribution

All six team members participated and shared the workload evenly. Therefore, it is recommended that each team member be given equal points of the calculated grade.

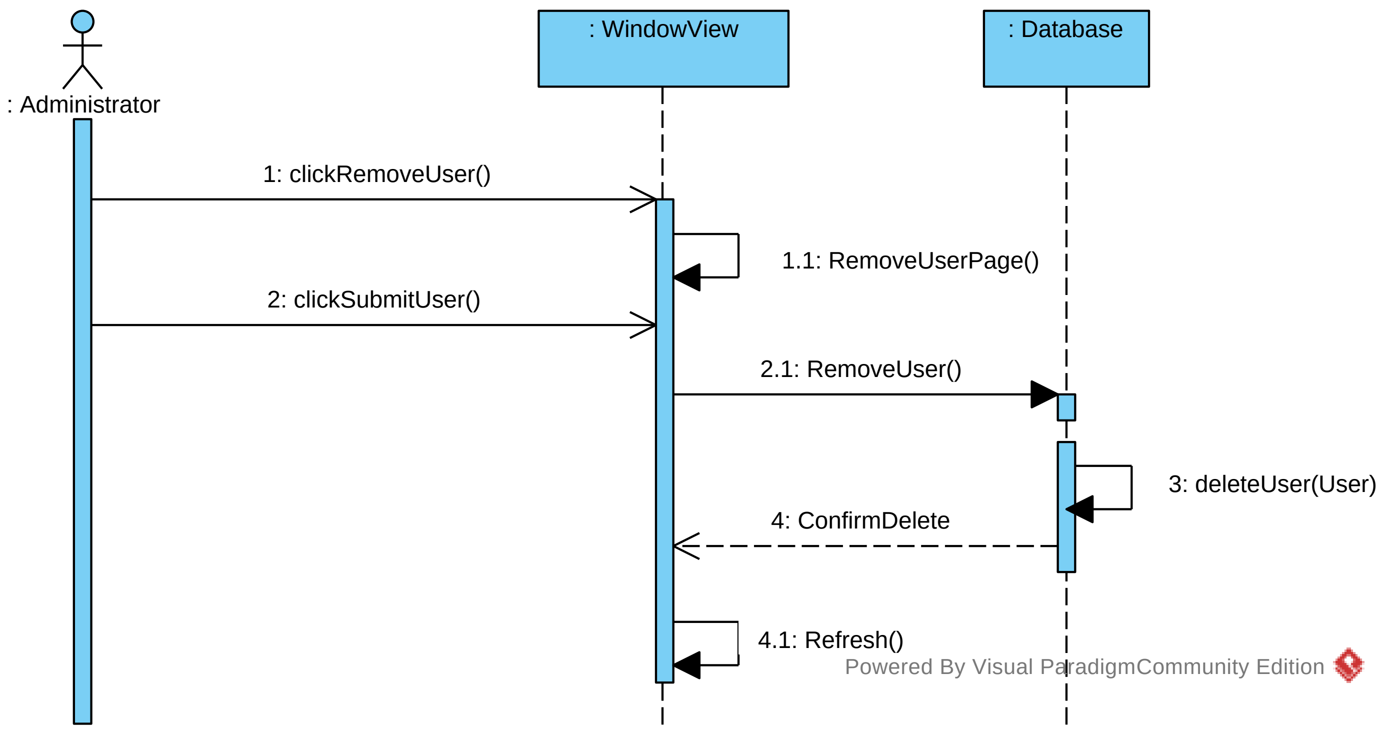
Iteration 2 Timecards

Name	Time Spent
Andrew Case	37h 00m
Matthew Darby	22h 15m
Mark Du	22h 37m
Maggie Burton	19h 30m
Stevie Damrel	14h 30m
Weston Straw	27h 14m



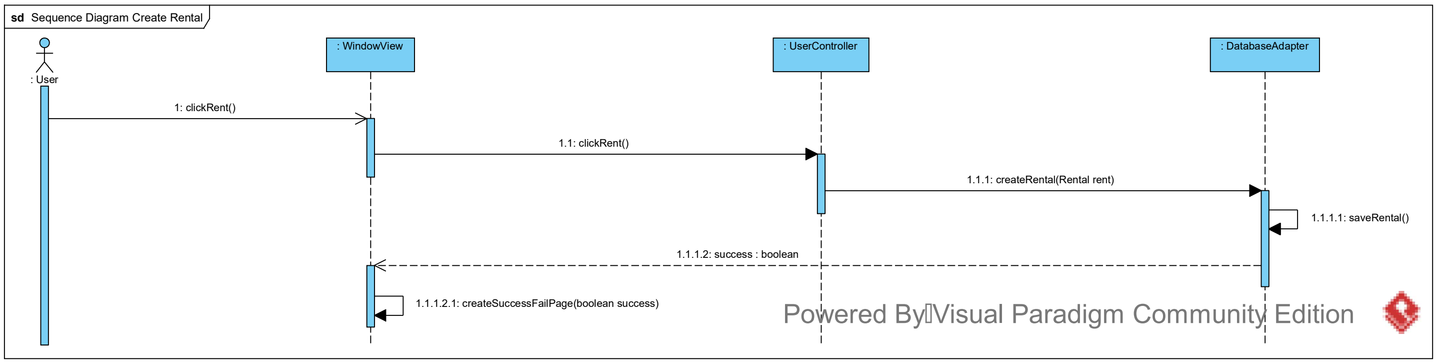
:Administrator
: WindowView
: Database
1: clickAddCar()
1.1: CreatorPage()
2: clickSubmitCar()
2.1: addCar(Car)
2.1.1: SaveCar(Car)
2.1.2: ConfirmSaveCar()
2.2: Refresh()
Powered By Visual ParadigmCommunity Edition

Add Car

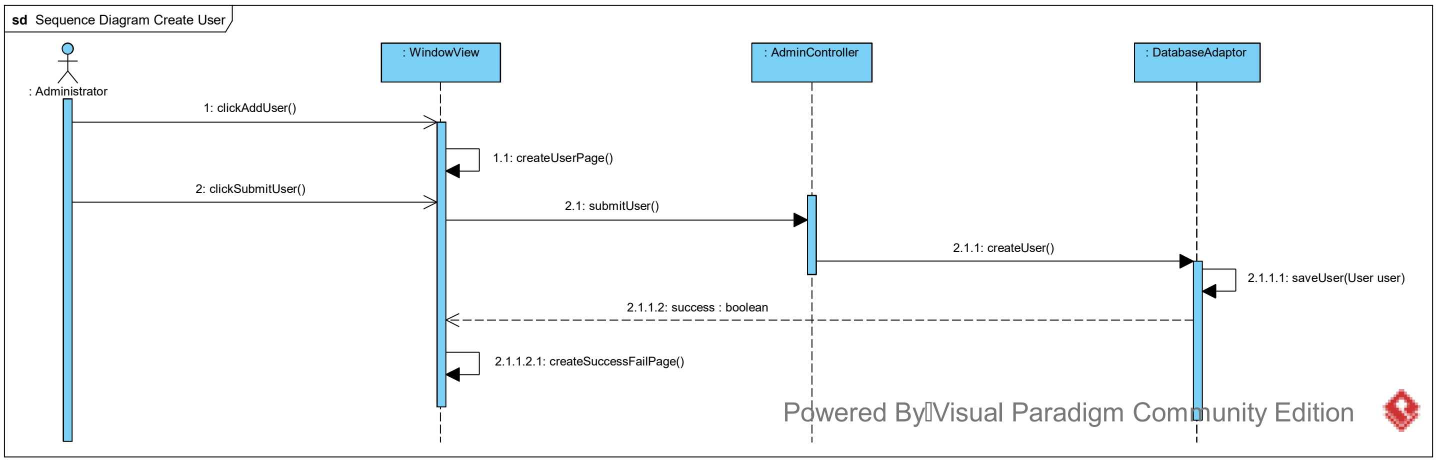


: Administrator
: WindowView
: Database
1: clickRemoveUser()
1.1: RemoveUserPage()
2: clickSubmitUser()
2.1: RemoveUser()
3: deleteUser(User)
4: ConfirmDelete
4.1: Refresh()
Powered By Visual ParadigmCommunity Edition

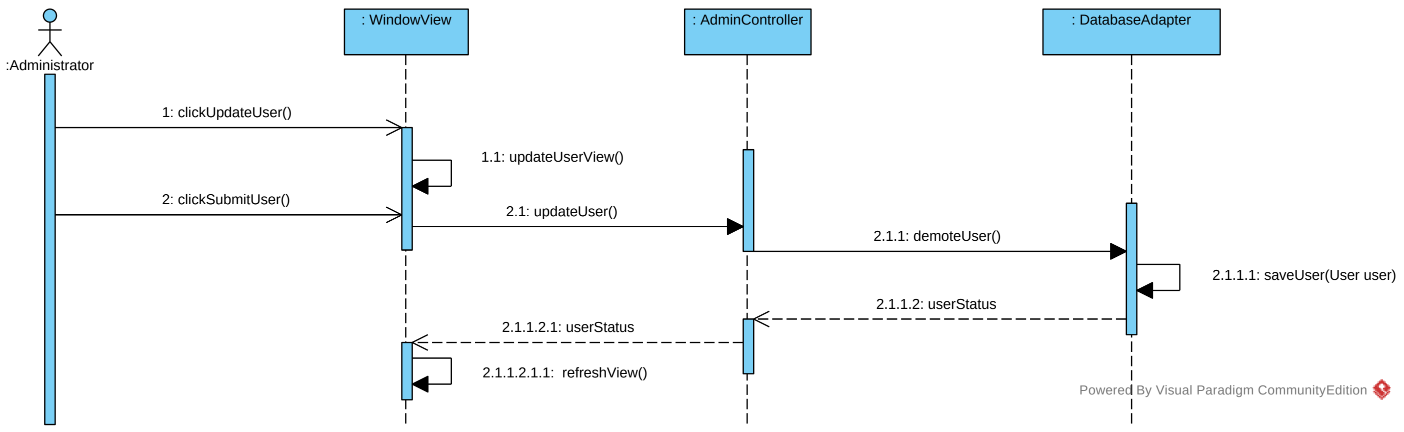
Remove User


sd Sequence Diagram Create Rental
: User
: WindowView
: UserController
: DatabaseAdapter
1: clickRent()
1.1: clickRent()
1.1.1: createRental(Rental rent)
1.1.1.1: saveRental()
1.1.1.2: success : boolean
1.1.1.2.1: createSuccessFailPage(boolean success)

Rent Car

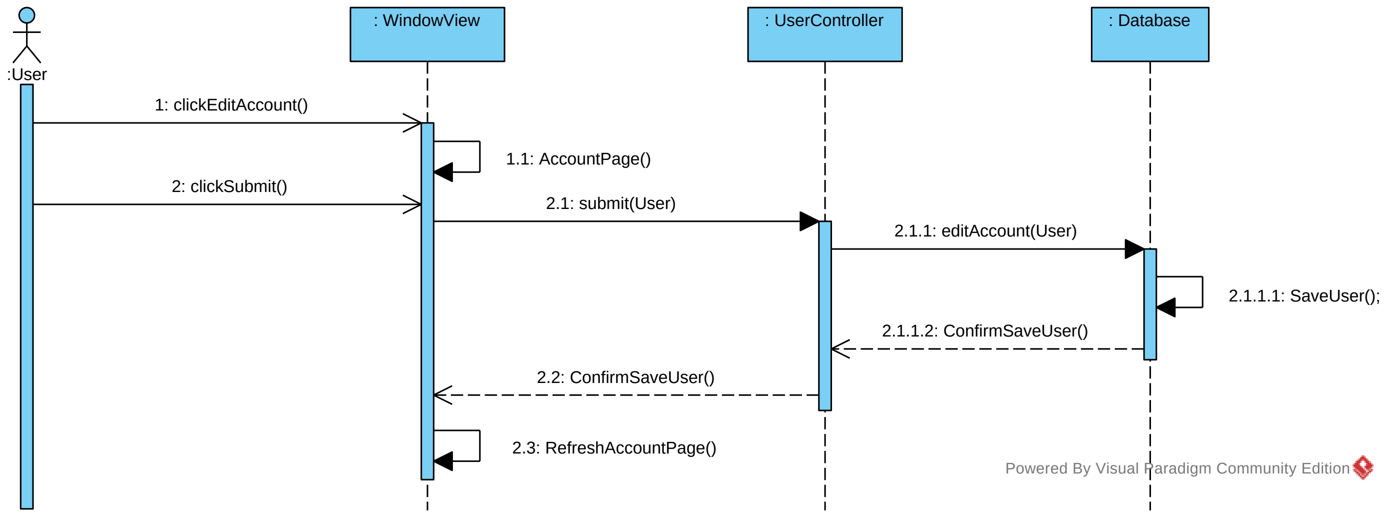

sd Sequence Diagram Create User
: Administrator
: WindowView
: AdminController
: DatabaseAdaptor
1: clickAddUser()
1.1: createUserPage()
2: clickSubmitUser()
2.1: submitUser()
2.1.1: createUser()
2.1.1.1: saveUser(User user)
2.1.1.2: success : boolean
2.1.1.2.1: createSuccessFailPage()
Powered By Visual Paradigm Community Edition

Add User

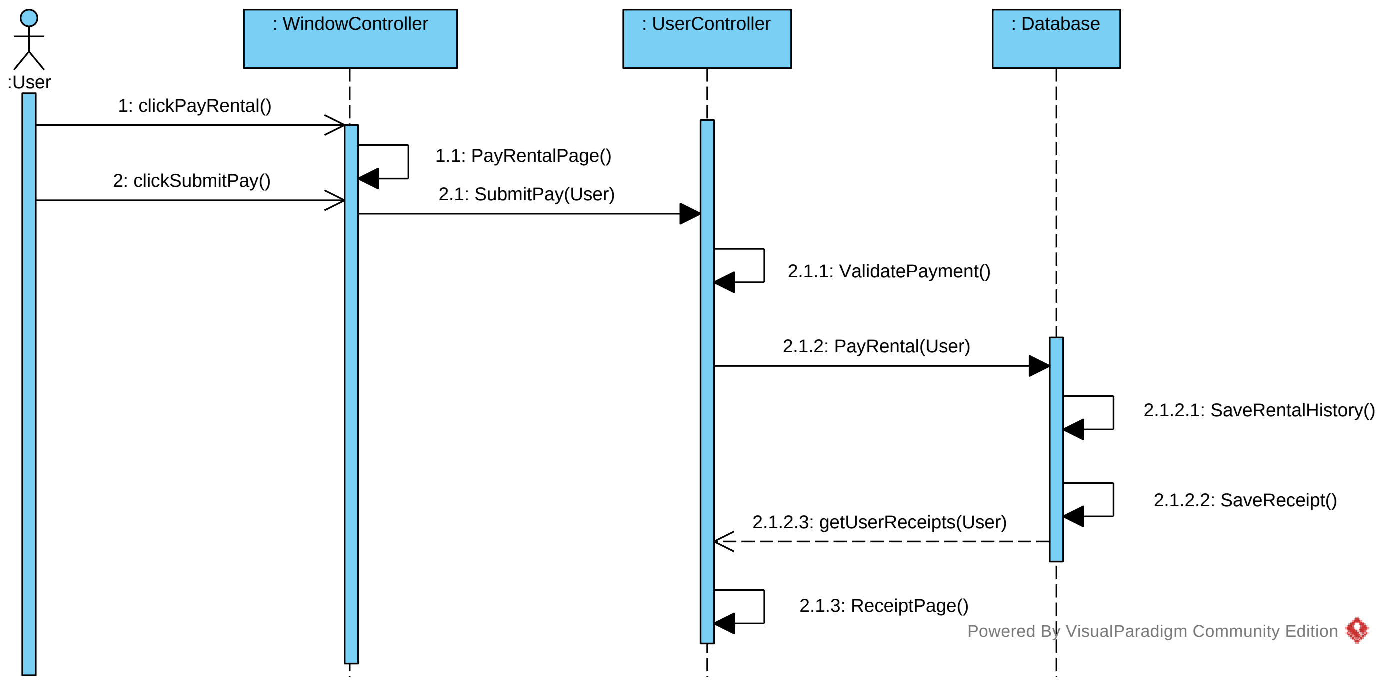


:Administrator
: WindowView
: AdminController
: DatabaseAdapter
1: clickUpdateUser()
1.1: updateUserView()
2: clickSubmitUser()
2.1: updateUser()
2.1.1: demoteUser()
2.1.1.1: saveUser(User user)
2.1.1.2: userStatus
2.1.1.2.1: userStatus
2.1.1.2.1.1: refreshView()
Powered By Visual Paradigm CommunityEdition

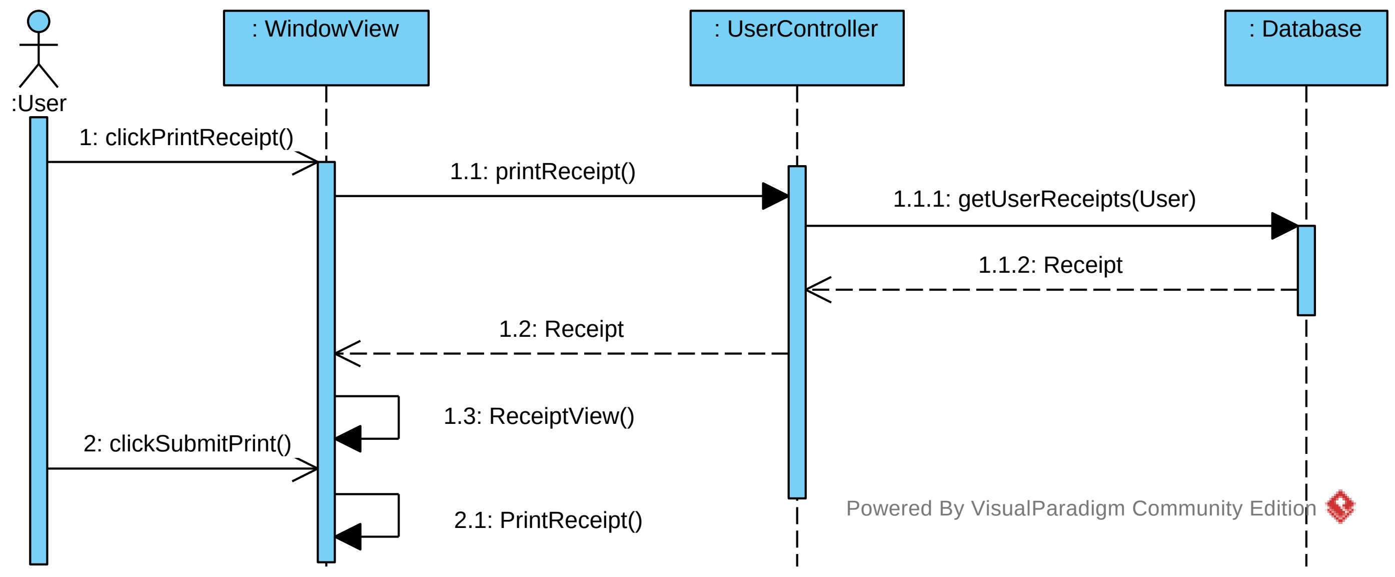
Demote User



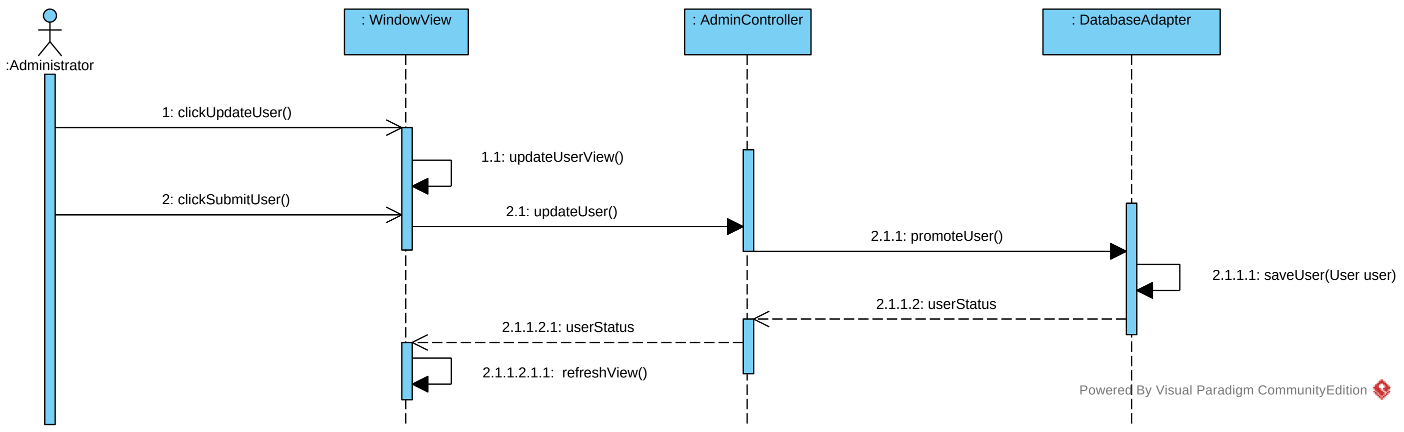
Edit Account


:User
: WindowController
: UserController
: Database
1: clickPayRental()
1.1: PayRentalPage()
2: clickSubmitPay()
2.1: SubmitPay(User)
2.1.1: ValidatePayment()
2.1.2: PayRental(User)
2.1.2.1: SaveRentalHistory()
2.1.2.2: SaveReceipt()
2.1.2.3: getUserReceipts(User)
2.1.3: ReceiptPage()
Powered By VisualParadigm Community Edition

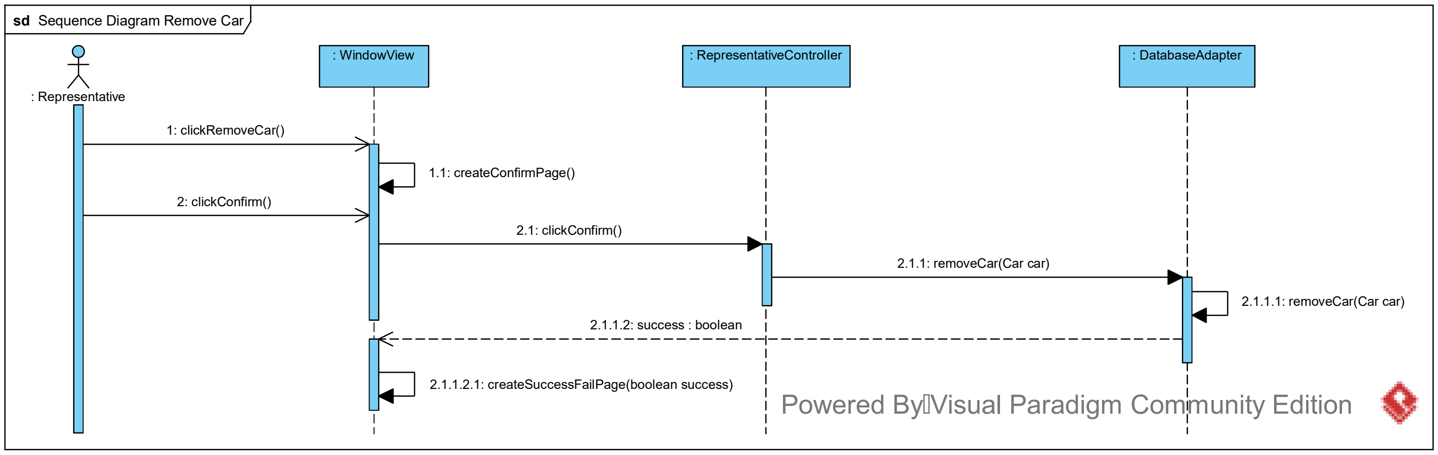
Pay Rental



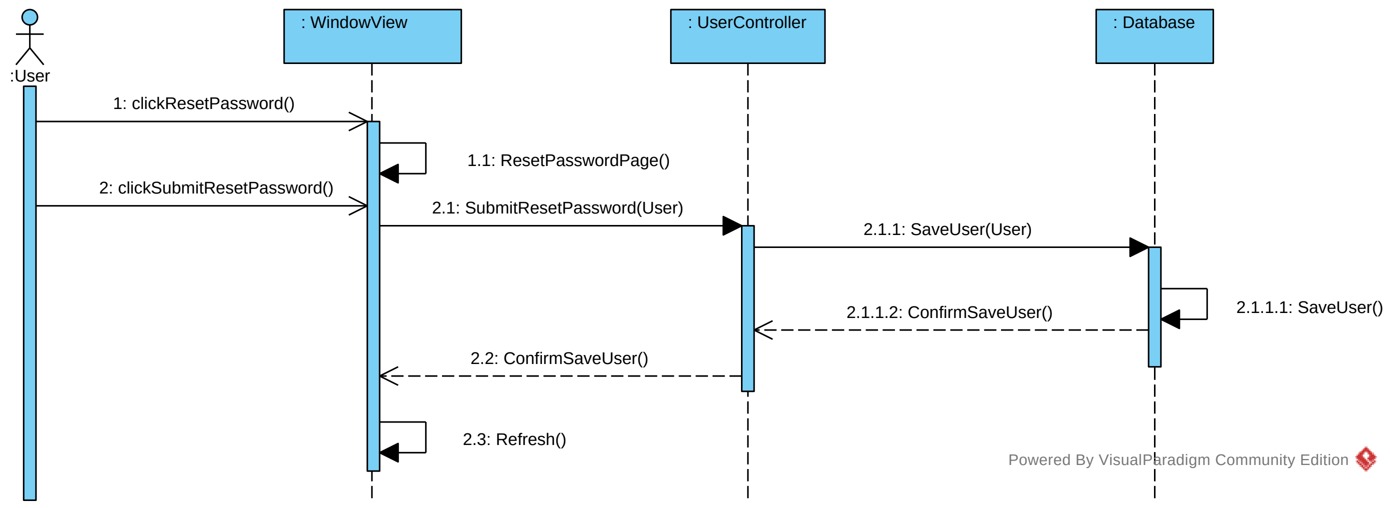
Print Receipt



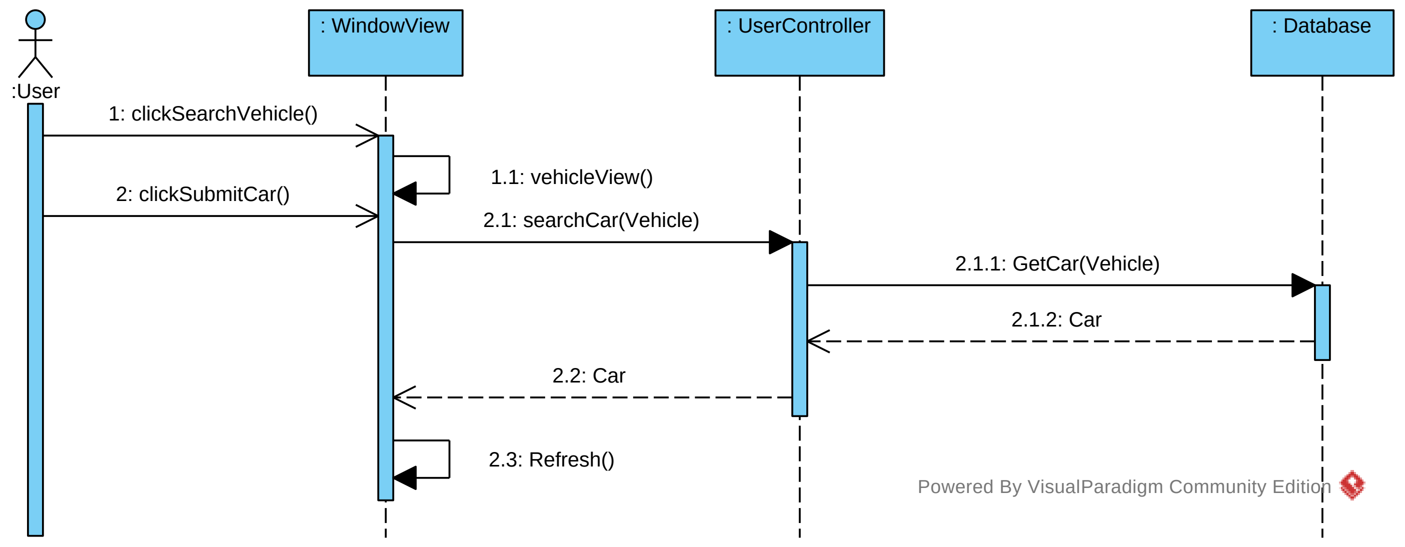
Promote User


sd Sequence Diagram Remove Car
: Representative
: WindowView
: RepresentativeController
: DatabaseAdapter
1: clickRemoveCar()
1.1: createConfirmPage()
2: clickConfirm()
2.1: clickConfirm()
2.1.1: removeCar(Car car)
2.1.1.1: removeCar(Car car)
2.1.1.2: success : boolean
2.1.1.2.1: createSuccessFailPage(boolean success)
Powered By Visual Paradigm Community Edition

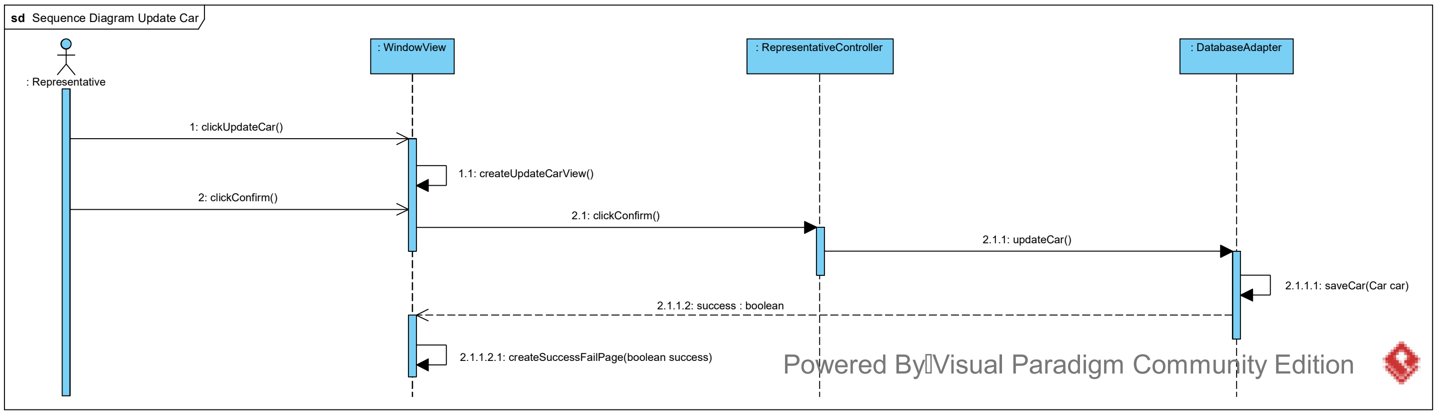
Remove Car



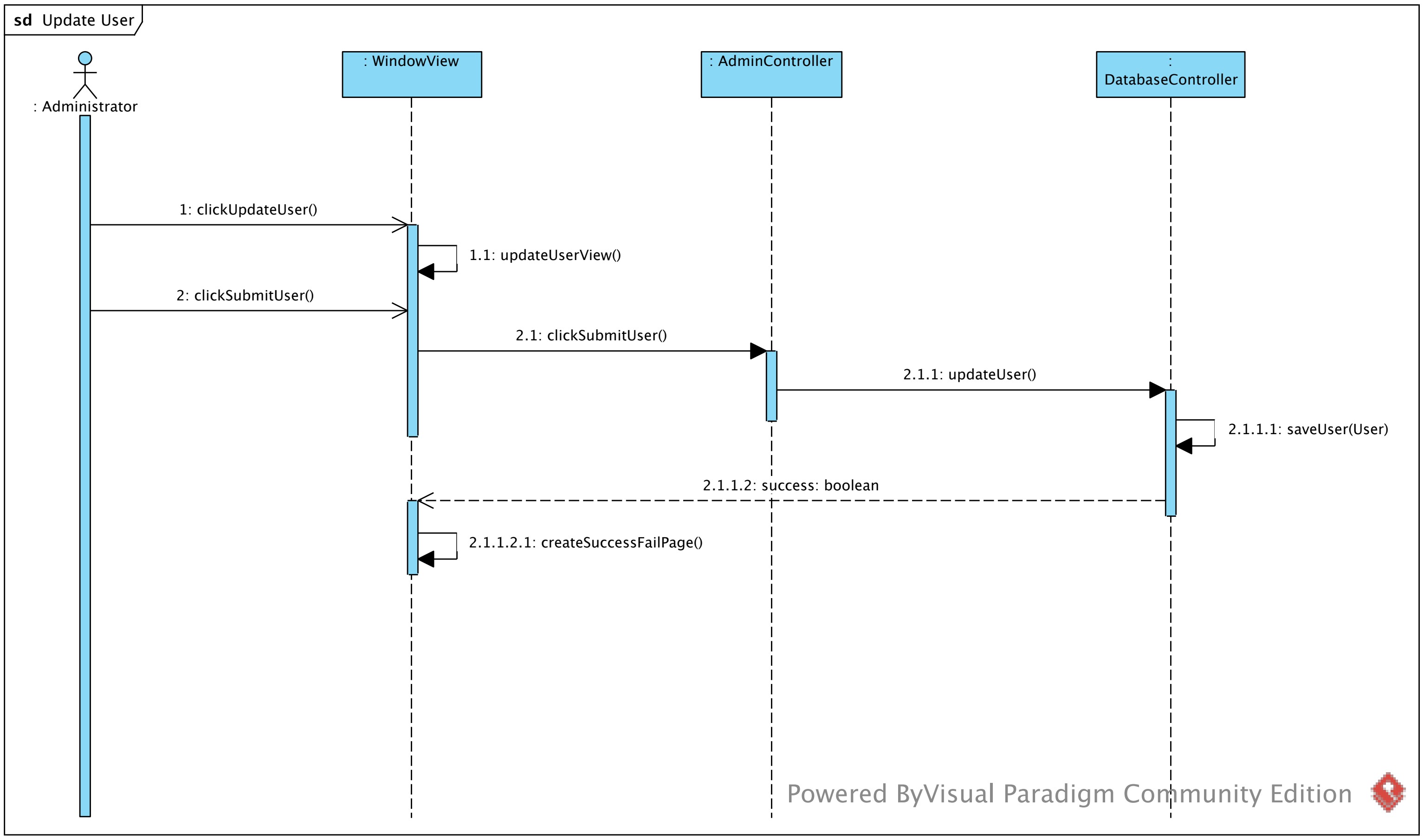
Reset Password


:User
: WindowView
: UserController
: Database
1: clickSearchVehicle()
1.1: vehicleView()
2: clickSubmitCar()
2.1: searchCar(Vehicle)
2.1.1: GetCar(Vehicle)
2.1.2: Car
2.2: Car
2.3: Refresh()
Powered By VisualParadigm Community Edition

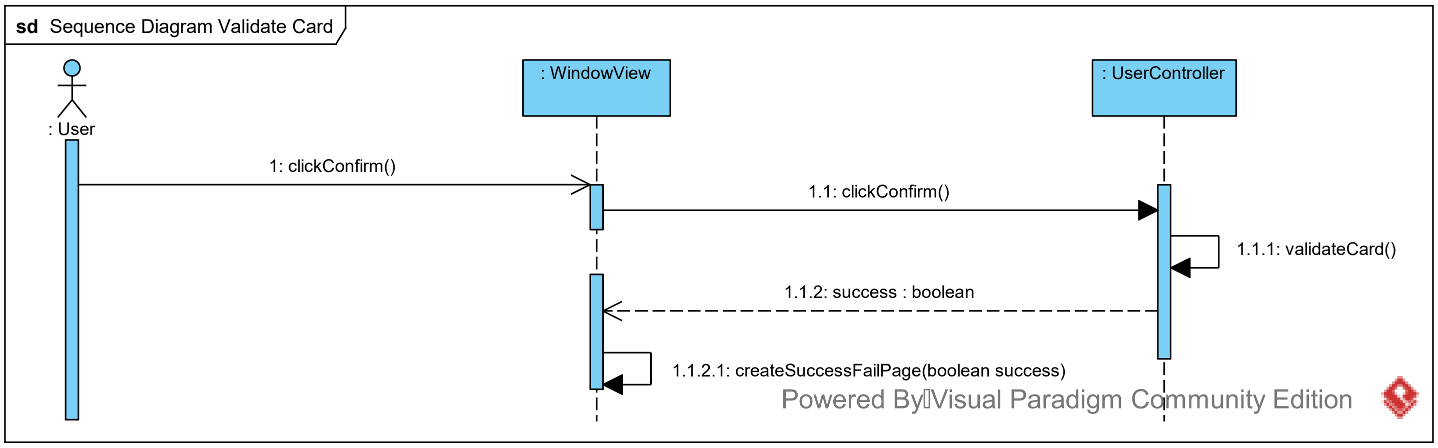
Search Car


sd Sequence Diagram Update Car
: Representative
: WindowView
: RepresentativeController
: DatabaseAdapter
1: clickUpdateCar()
1.1: createUpdateCarView()
2: clickConfirm()
2.1: clickConfirm()
2.1.1: updateCar()
2.1.1.1: saveCar(Car car)
2.1.1.2: success : boolean
2.1.1.2.1: createSuccessFailPage(boolean success)
Powered By Visual Paradigm Community Edition

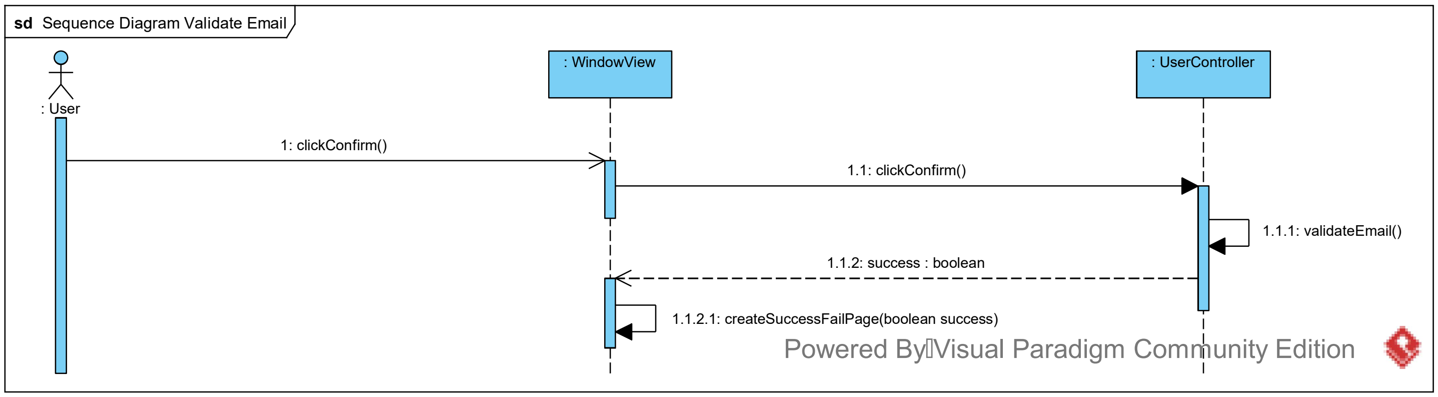
Update Car


sd Update User
: Administrator
: WindowView
: AdminController
: DatabaseController
1: clickUpdateUser()
1.1: updateUserView()
2: clickSubmitUser()
2.1: clickSubmitUser()
2.1.1: updateUser()
2.1.1.1: saveUser(User)
2.1.1.2: success: boolean
2.1.1.2.1: createSuccessFailPage()
Powered ByVisual Paradigm Community Edition

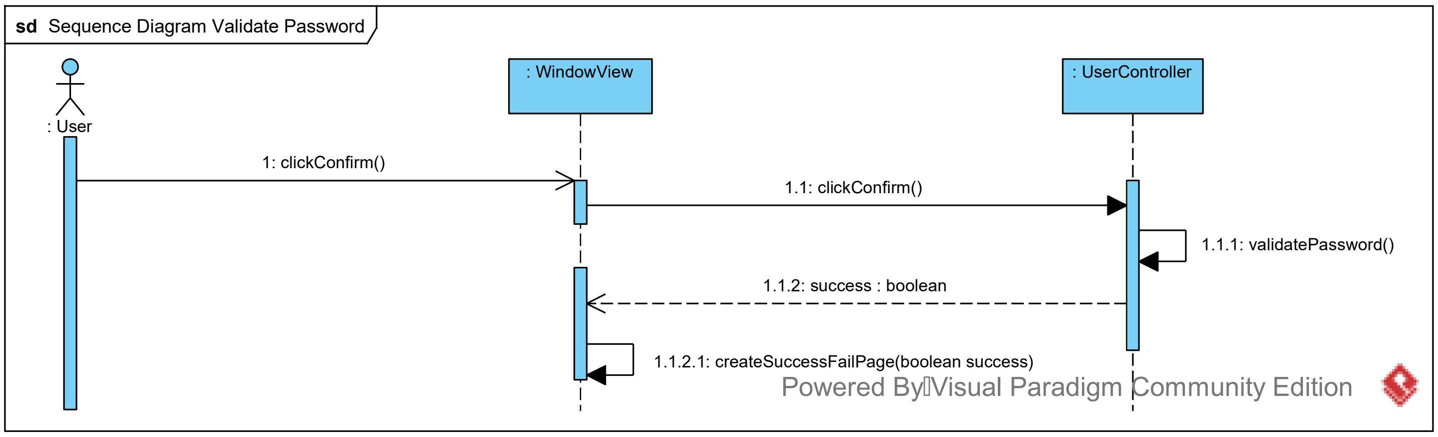
Update User


sd Sequence Diagram Validate Card
: User
: WindowView
: UserController
1: clickConfirm()
1.1: clickConfirm()
1.1.1: validateCard()
1.1.2: success : boolean
1.1.2.1: createSuccessFailPage(boolean success)
Powered By Visual Paradigm Community Edition

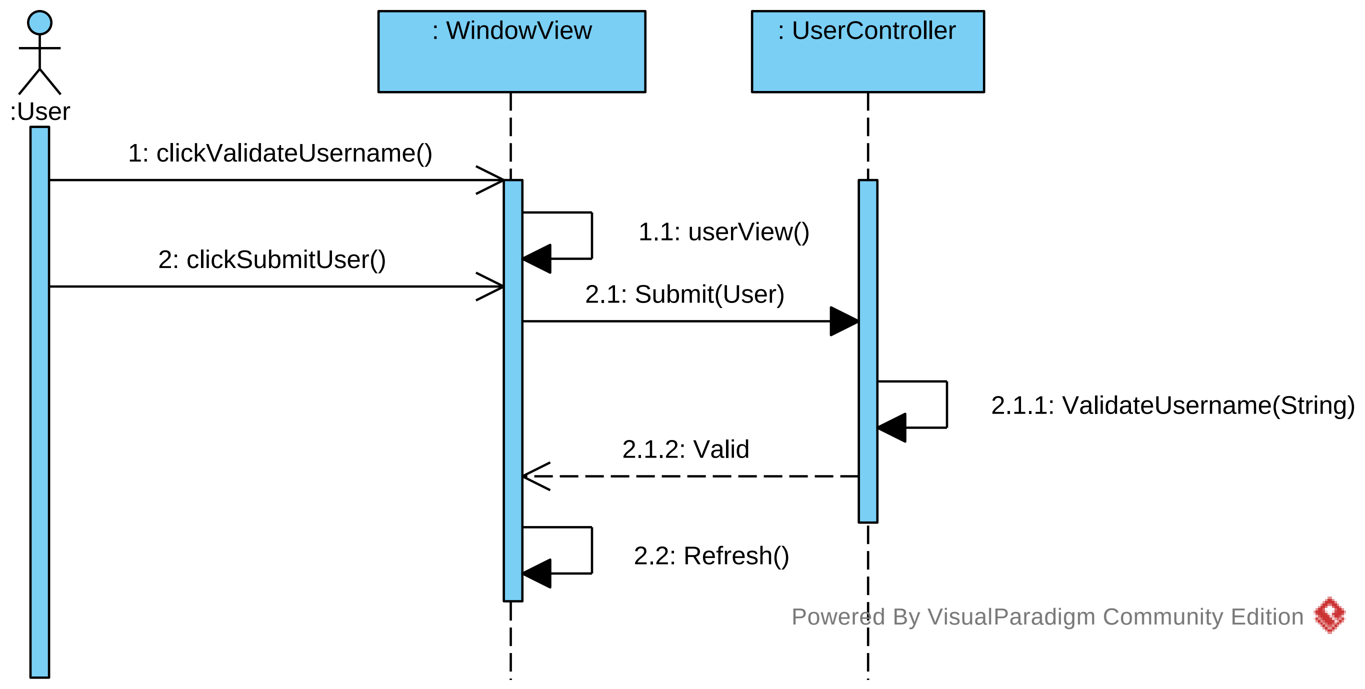
Validate Card


sd Sequence Diagram Validate Email
: User
: WindowView
: UserController
1: clickConfirm()
1.1: clickConfirm()
1.1.1: validateEmail()
1.1.2: success : boolean
1.1.2.1: createSuccessFailPage(boolean success)
Powered By Visual Paradigm Community Edition

Validate Email


sd Sequence Diagram Validate Password
: User
: WindowView
: UserController
1: clickConfirm()
1.1: clickConfirm()
1.1.1: validatePassword()
1.1.2: success : boolean
1.1.2.1: createSuccessFailPage(boolean success)
Powered By Visual Paradigm Community Edition

Validate Password



Validate Username