Sistema de Caronas - ShibaRides
Modelo de arquitetura de Software

Alunos
João Tito do Nascimento Silva
Pedro Fernandes

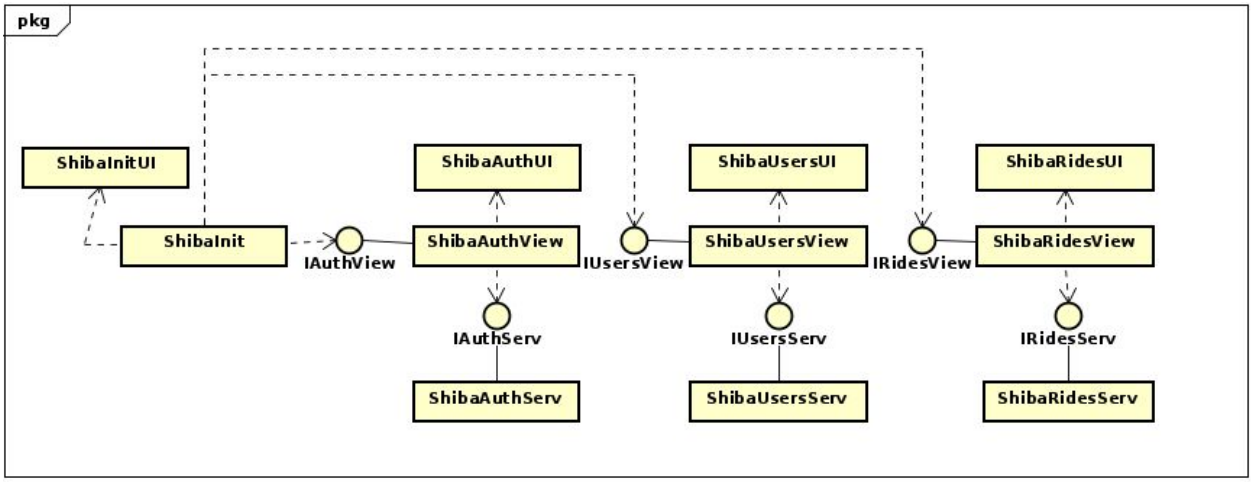
Novembro de 2019

Introdução

O sistema possui como objetivo proporcionar ao usuário a possibilidade de ofertar ou reservar caronas.

> O sistema de software a ser desenvolvido tem o objetivo de facilitar o transporte por meio de caronas entre cidades. Por meio desse sistema, qualquer usuário pode obter dados sobre caronas disponíveis. Para acessar esses dados, o usuário deve prover os seguintes dados quando de uma consulta: cidade de origem, estado de origem, cidade de destino, estado de destino e data de partida. Em resposta à consulta realizada, o sistema informa as caronas disponíveis. Para cada carona, são apresentados os seguintes dados: código da carona, nome do motorista, telefone do motorista, endereço de correio eletrônico do motorista, data de partida, duração do trajeto, número de vagas e preço. Para acessar os outros serviços providos pelo sistema, o usuário precisa antes ser autenticado. Para ser autenticado, o usuário precisa estar cadastrado. Ao se cadastrar, o usuário deve informar nome, telefone, endereço de correio eletrônico, senha e CPF. Também deve informar uma ou duas contas correntes para pagamentos de caronas. Para cada conta corrente, deve informar código de banco, número de agência e número de conta corrente. Para ser autenticado, o usuário deve informar endereço de correio eletrônico e senha. Uma vez autenticado, o usuário tem acesso aos seguintes serviços: cadastrar carona, descadastrar carona, reservar carona e cancelar reserva de carona. Para cadastrar uma carona, deve informar código da carona, cidade de origem, estado de origem, cidade de destino, estado de destino, data de partida, duração do trajeto, quantidade de vagas e preço. Um usuário não pode disponibilizar caronas com conflito de data, considerando-se data de partida e duração. O usuário que cadastrou uma carona pode listar as reservas associadas à mesma. Para isso, deve informar o código da carona. Para cada reserva associada à carona, o sistema informa, código da reserva, nome do passageiro, endereço de correio eletrônico do passageiro, preferência de assento e número de volumes da bagagem. Apenas o usuário que cadastrou uma carona pode descadastrá-la. Uma carona pode ser descadastrada desde que não existam reservas associadas à mesma. Para descadastrar uma carona, o usuário deve informar o código da carona. Quando uma carona é descadastrada, os seus dados são excluídos do sistema. Ao reservar uma carona, o usuário deve informar o código da carona, a preferência de assento e o número de volumes da sua bagagem. Ao confirmar uma solicitação de reserva, o sistema verifica o número de vagas disponíveis e informa, caso a reserva possa ser realizada, código de reserva, código(s) de banco(s), número(s) de agência(s) e número(s) de conta(s) corrente(s) para pagamento. Um usuário não pode efetuar reservas em caronas com conflito de data, considerando-se data de partida e duração. O cancelamento de reserva de carona só pode ocorrer se a data da carona for anterior à data atual. Cada usuário pode se descadastrar do sistema quando desejar, desde que não esteja associado a caronas ou a reservas. Quando do descadastramento de um usuário, os seus dados são excluídos.

Para que o sistema seja funcional, optamos por dividi-lo em três subsistemas: autenticação(auth), usuários(users) e caronas(rides). Conforme os requisitos, separamos o sistema em duas camadas: a camada de apresentação e a camada de serviço. Dessa maneira, obtivemos um sistema composto por módulos, cujo modelo é apresentado na figura abaixo.



Os nomes dos módulos foram decididos de modo a facilitar a compreensão do sistema. Cada módulo se inicia por “Shiba”, seguido pelo subsistema correspondente, como citado anteriormente e, por fim, pela camada correspondente, que pode ser “View”, para a camada de apresentação, ou “Serv”, para a camada de serviço. As interfaces seguem o mesmo padrão, porém alterando o prefixo “Shiba” por “I”.

Os módulos da camada de serviço são ShibaAuthServ, ShibaUsersServ e ShibaRidesServ, que implementam as interfaces IAuthServ, IUsersServ e IRidesServ. Esses módulos são responsáveis por fornecer a persistência dos dados e a lógica de negócio.

Os módulos ShibaAuthView, ShibaUsersView e ShibaRidesView são os módulos da camada de apresentação e são responsáveis pela interface com o usuário e pela validação dos dados, que ocorre por meio dos domínios. Esses módulos implementam as interfaces IAuthView, IUsersView e IRidesView. A interface gráfica é implementada pelos módulos ShibaAuthUI, ShibaUsersUI e ShibaRidesUI por meio da utilização da biblioteca PDCurses(ncurses, no caso dos sistemas Linux).

Há ainda um módulo de inicialização, chamado ShibaInit, responsável por montar o sistema e iniciar a execução. Esse módulo utiliza o módulo ShibaInitUI para fornecer a interface gráfica da página principal e páginas de seleção e utiliza as interfaces providas pelos módulos da camada de apresentação para organizar o fluxo da aplicação pelos subsistemas.

A implementação das interfaces ocorre por meio de controladoras. Cada interface contém métodos virtuais puros que são sobrescritos pelos métodos das controladoras, as quais herdam das classes dessas interfaces.

Módulos

A partir da noção geral sobre cada subsistema e das camadas, podemos analisar as responsabilidades de cada módulo individualmente.

- Subsistema de autenticação(ShibaAuth): subsistema responsável pelo processo de autenticação do usuário no sistema para sua utilização.
    - Módulo ShibaAuthView: módulo da camada de apresentação do subsistema de autenticação. Esse módulo implementa as funções da interface IAuthView, utilizada pelo módulo de inicialização, e é responsável por apresentar ao usuário a tela de autenticação e tentar realizar o login por meio da utilização da interface IAuthServ.
    - Módulo ShibaAuthServ: módulo da camada de serviço do subsistema de autenticação. Esse módulo implementa as chamadas da interface IAuthServ. Sua responsabilidade é acessar a banco de dados que contém as informações sobre a autenticação dos usuários e verificar, a partir das informações dadas pela camada superior, se o usuário pode ser autenticado.
    - Módulo ShibaAuthUI: módulo responsável pela implementação da interface gráfica de autenticação. Por meio de classes com métodos estáticos, esse módulo constrói a tela de autenticação e garante que as informações dadas pelo usuário sejam retornadas para serem utilizadas pelo módulo ShibaAuthView.
- Subsistema de usuários(ShibaUsers): subsistema responsável pela edição de dados de usuário e cadastramento de novos usuários.
    - Módulo ShibaUsersView: módulo da camada de apresentação do subsistema de usuários. Esse módulo é responsável por chamar as funções que implementam as interfaces gráficas e, por meio da interface da camada inferior, realizar o cadastramento ou edição de um usuário no banco de dados.
    - Módulo ShibaUsersServ: módulo da camada de serviço do subsistema de usuários. Esse módulo é responsável por acessar a base de dados, quando solicitado, e tentar garantir que as mudanças solicitadas ocorram, que no caso se trata da edição ou cadastramento de um usuário. A lógica de negócio deve ser implementada dentro desse módulo. O módulo

também será responsável por retornar informações sobre um usuário já cadastrado.
    - Módulo ShibaUsersUI: módulo responsável pela interface gráfica do subsistema de usuários. Por meio de classes estáticas, o módulo constrói as interfaces e retorna os resultados fornecidos pelo usuário. Nesse caso, teremos uma tela de cadastramento e outra de edição de dados.
- Subsistema de caronas(ShibaRides): subsistema responsável pela reserva e fornecimento de caronas por um usuário, que inclui a pesquisa por caronas e outras modalidades.
    - Módulo ShibaRidesView: módulo da camada de apresentação no subsistema de caronas. Sua função é chamar as funções de construção da interface gráfica e entregar ou receber dados sobre caronas da camada inferior, por meio da interface IRidesServ. No caso desse módulo, temos tanto a entrega quanto o recebimento de dados, pois as caronas devem ser listadas e suas informações devem ser exibidas.
    - Módulo ShibaRidesServ: módulo da camada de serviço no subsistema de caronas. Sua função é retornar ou guardar informações sobre caronas em uma estrutura de persistência, que se trata do banco de dados. Esse módulo também deve implementar corretamente a lógica de negócio.
    - Módulo ShibaRidesUI: módulo responsável pela implementação das interfaces gráficas do subsistema de caronas. As interfaces deverão ser capazes de listar caronas e suas informações, de modo que, por vezes, será necessário interagir anteriormente com a camada de serviço.

Interfaces

A utilização de interfaces no sistema permite que haja certa abstração entre os módulos. Isso é vantajoso pois auxilia na manutenção do sistema, além de facilitar o desenvolvimento.

O módulo de inicialização interage com os módulos da camada de apresentação de cada subsistema por meio de interfaces. Do mesmo modo, os módulos da camada de apresentação de cada subsistema interagem com os módulos da cada de serviço de seus subsistemas correspondentes por meio de interfaces.

Podemos analisar as interfaces já citadas individualmente, de modo a expor os seus métodos. Os métodos que utilizam de alguma maneira o banco de dados, seja direta ou indiretamente, podem lançar uma exceção *runtime_error*, que indica que o banco de dados está indisponível. Nesse caso, é apresentada uma tela indicando o problema.

- IAuthView: interface da camada de apresentação do subsistema de autenticação. Métodos:
  - *virtual bool login(Email &email) throw (runtime_error) = 0;*
    Método responsável por mostrar a tela de login e efetuar a autenticação do usuário. Deve ser passado um objeto email, que armazenará o email do usuário caso esse seja autenticado, para posterior utilização no sistema. É retornado True caso a autenticação seja realizada, e False caso o usuário desista da autenticação.

  - *virtual void setServiceController(IAuthServ *) = 0;*
    Método invocado na construção do sistema. A chamada desse método serve para definir uma referência a uma controladora da camada de serviço, pois a camada de apresentação dependerá de seus métodos.

- IAuthServ:
  - *virtual bool authenticate(Email email, Senha senha) throw (runtime_error)=0;*
    Método responsável por acessar o banco de dados e verificar se as informações dadas nos parâmetros correspondem a um usuário válido, de modo a efetuar ou não a autenticação. Caso a autenticação seja bem sucedida, é retornado True. Caso contrário, é retornado False.

- IUsersView:
    - *virtual void signup(void) throw (runtime_error) = 0;*
      Método responsável pelo cadastramento de um usuário. A chamada desse método leva à exibição de uma tela de cadastramento. Por meio do módulo ShibaUsersUI, o módulo ShibaUsersView constrói a tela e obtém os dados do usuário. Esses dados são passados ao módulo de serviço, por meio da interface IUsersServ, e esse módulo verifica se é possível criar o usuário.

    - *virtual bool deletion(Email email) throw (runtime_error) = 0;*
      Método pelo qual o usuário solicitará sua deleção do sistema.

- IUsersServ:
    - *virtual bool registerUser(Nome nome, Email email, Senha senha, Telefone telefone, CPF cpf) throw (runtime_error) = 0;*
      Método da camada de serviço, responsável pela criação de um novo usuário no banco de dados. Ao ser invocado, esse método utiliza os dados passados e verifica se é possível criar o usuário. Em caso afirmativo, é retornado True. Em caso negativo, é retornado False.

    - *virtual bool deleteUser(Email email) throw(runtime_error) = 0;*
      Método responsável pela deleção do usuário da base de dados. Dada a informação de email do usuário, o módulo de serviço acessa a base de dados e remove o usuário, caso não haja caronas ou reservas associadas a este.

- IRidesView:
    - *virtual void queryRides(void) throw (runtime_error)=0;*
      A chamada desse método ocasiona, pela utilização do módulo ShibaRidesUI, a exibição de uma tela para fazer pesquisa por caronas. O usuário deverá digitar as informações(cidade de origem, estado de origem, cidade de destino, estado de destino e data de partida) e serão listadas as caronas disponíveis com esses dados. Por meio do método getRides da interface IRidesServ, são obtidas as caronas com as informações dadas associadas, e estas são exibidas ao usuário.

    - *virtual void reserveRide(Email email) throw (runtime_error)=0;*
      Esse método exibe, por meio do módulo ShibaRidesUI, uma tela que permite que o usuário digite um código de carona para criar uma reserva.

Deve ser passado como parâmetro um email, que será uma referência para registrar, no banco de dados, a associação entre a reserva e o usuário que a solicitou. O método reserveRide da interface IRidesServ é invocado para guardar, no banco de dados, a nova reserva.

- *virtual void registerRide(Email email) throw (runtime_error)=0;*
  Cadastramento de uma carona. Quando esse método é invocado, por meio do módulo ShibaRidesUI, é exibida uma tela com vários campos onde o usuário deverá inserir informações a serem usadas para criar uma nova carona. A carona é registrada no banco de dados por meio do método createRide, da interface IRidesServ.

- *virtual void listUserReservedRides(Email email) throw (runtime_error)=0;*
  Lista as caronas que determinado usuário, indicado pelo email, já reservou. Quando esse método é invocado, a controladora utiliza as classes do módulo ShibaRidesUI para construir a tela, usando as informações obtidas da camada inferior por meio da interface IRidesServ.

- *virtual void listUserOfferedRides(Email email) throw (runtime_error)=0;*
  Lista as caronas que determinado usuário ofertou para reserva. Quando esse método é chamado, a controladora utiliza a interface com a camada inferior para obter as caronas oferecidas pelo usuário, e exibe-as na tela por meio das classes do módulo ShibaRidesUI.

- *virtual void setServiceController(IRidesServ *)= 0;*
  Método invocado na construção do sistema. A chamada desse método serve para definir uma referência a uma controladora da camada de serviço, pois a camada de apresentação dependerá de seus métodos.

- IRidesServ:
  - *virtual bool getRides(Estado estorigem, Cidade cidorigem, Estado estdest, Cidade ciddest, std::vector<Carona> &caronavec) throw (runtime_error)=0;*
    Acessa o banco de dados em busca de caronas que atendam às informações especificadas nos parâmetros. O método retorna True caso hajam caronas disponíveis com os dados informados e False caso contrário. Se há caronas disponíveis, elas são armazenadas no vetor *caronavec*, que é utilizado pela camada superior para exibir os resultados ao usuário pela da interface gráfica.

- *virtual bool createRide(Email email, CodDeCarona cod, Estado estorigem, Cidade cidorigem, Estado estdest, Cidade ciddest, Data leaving, Duracao time, Vagas available, Preco price) throw (runtime_error)=0;*
  Método invocado durante a criação de uma nova carona. O método acessa a base de dados e, caso seja possível criar a carona, retorna True. Caso contrário, retorna False.

- *virtual bool getReserves(Email email, std::vector<Reserva> &reservesvec) throw (runtime_error)=0;*
  Método que obtém as reservas associadas ao usuário indicado pelo email. Caso haja reservas associadas, é retornado True e as reservas são armazenadas no vetor *reservesvec* para utilização na camada superior. Caso contrário, é retornado False.

- *virtual bool getOfferedRides(Email email, std::vector<Carona> &caronavec) throw (runtime_error)=0;*
  Obtém as caronas ofertadas pelo usuário indicado pelo email. Caso haja caronas ofertadas, retorna True e salva as caronas no vetor *caronavec* para exibição na interface gráfica. Caso contrário, retorna False.

- *virtual bool reserveRide(CodDeReserva codreserva, Assento assento, Bagagem bagagem) throw (runtime_error)=0;*
  Cria, no banco de dados, a nova reserva solicitada, associando-a ao usuário. Retorna True caso seja possível realizar a reserva, e false em caso contrário.