**Лабораторная работа №2**

Приложение спроектировано на основе Model-View-ViewModel шаблона.

**Model:**

* *LoginRequest*: запрос для авторизации пользователя.

Поля:

+ Email: строка, содержащая email пользователя.

+ Password: строка, содержащая пароль пользователя.

* *UserInfo*: содержит информацию о пользователе.

Поля:

+ Id: число для идентификации объекта.

+ FirstName: строка, содержащая имя.

+ LastName: строка, содержащая фамилию.

+ Email: строка, содержащая email.

+ PhoneNumber: строка, содержащая номер телефона.

+ Role: строка, содержащая роль пользователя (Admin, Manager, Worker).

* *LoginResponce*: ответ для авторизации пользователя.

Поля:

+ Token: строка, содержащая токен авторизации для идентификации пользователя при последующих запросах.

+ UserDetail: обьект класса UserInfo, содержащий информацию о пользователе.

* *RegisterRequest*: запрос для регистрации пользователя.

Поля:

+ UserDetail: обьект класса UserInfo, содержащий информацию о будущем пользователе.

+ Password: строка, содержащая пароль будущего пользователя.

* *Client*: информация о клиенте.

Поля:

+ Id: число для идентификации объекта.

+ FirstName: строка, содержащая имя.

+ LastName: строка, содержащая фамилию.

+ Email: строка, содержащая email.

+ PhoneNumber: строка, содержащая номер телефона.

+ Adress: строка, содержащая адрес проживания клиента.

+ IsProcessed: булево значение, содержащее информации об обработанности или необработанности заказа клиента.

+ UserId: число для идентификации пользователя, который работает с клиентом.

* *Report*: информация о жалобе.

Поля:

+ Id: число для идентификации объекта.

+ FirstName: строка, содержащая имя жалующегося.

+ LastName: строка, содержащая фамилию жалующегося.

+ Email: строка, содержащая email жалующегося.

+ PhoneNumber: строка, содержащая номер телефона жалующегося.

+ Text: строка, содержащая текст жалобы.

+ UserId: число для идентификации пользователя, которому отправлена жалоба.

+ IsAnswered: булево значение, содержащее информации обработанности или необработанности жалобы пользователем.

+ Answer: строка, содержащая ответ на жалобу.

* *Product*: информация о типе продукта.

Поля:

+ Id: число для идентификации объекта.

+ Name: строка, содержащая наименование продукта/товара.

+ Description: строка, содержащая описание продукта.

+ Price: вещественное число, содержащее цену продукта.

+ Amount: число, содержащее количество доступного на данный момент продукта.

+ ReorderLevel: число, содержащее количество продукта ниже которого необходимо делать новый заказ этого продукта.

* *Ceremony*: информация о церемонии/ритуале.

Поля:

+ Id: число для идентификации объекта.

+ Name: строка, содержащая наименование церемонии.

+ Description: строка, содержащая описание церемонии.

+ Price: вещественное число, содержащее цену проведения церемонии.

* *Order*: информация о заказе.

Поля:

+ Id: число для идентификации объекта.

+ Date: дата и время проведения церемонии.

+ Address: строка, содержащая адрес места проведения церемонии

+ CeremonyId: число для идентификации церемонии, которая заказана клиентом.

+ ClientId: число для идентификации клиента, которому принадлежит заказ.

* *ProductOrder*: информация о заказе продукта.

Поля:

+ Id: число для идентификации объекта.

+ Amount: число, содержащее количество заказанного продукта.

+ Comment: строка, содержащая информация о предпочтениях клиента по поводу продукта.

+ ProductId: число для идентификации типа продукта, который заказывается.

+ OrderId: число для идентификации заказа, в которым присутствует данный заказ продукта.

* *DeadPerson*: информация о погибшем человеке.

Поля:

+ Id: число для идентификации объекта.

+ FirstName: строка, содержащая имя.

+ LastName: строка, содержащая фамилию.

+ DateOfBirth: дата рождения.

+ DateOfDeath: дата смерти.

+ Details: строка, содержащая различную информацию о погибшем, которую предоставляет клиент.

+ OrderId: число для идентификации заказа, в котором будет оказана похоронная услуга для погибшего.

**Service:**

* *LoginService*: сервис для авторизации.

Методы:

+ Authenticate(LoginRequest): отправляет на сервер запрос авторизации, возвращает LoginResponce.

* *EmaiSender*: сервис для отправки email сообщения.

Методы:

+ SendEmail(string, string): первый параметр представляет собой email получателя, второй параметр ­­– текст сообщения. Отправляет сообщение по переданному email.

* *UserService, ClientService, ReportService, ProductService, CeremonyService, OrderService, ProductOrderService, DeadPersonService*: сервисы для CRUD операциями над соответствующими моделями.

Методы (<Model> замещает модели над которыми работают сервисы):

+ Get<Model>s(): отправляет запрос на сервер и возвращает список объектов <Model>.

+ Get<Model>(int id): отправляет запрос на сервер и возвращает объект <Model> по переданному id.

+ Delete<Model>(int id): отправляет запрос на сервер по удалению <Model> по переданному id.

+ Update<Model>(<Model>): отправляет запрос на сервер по обновление переданного <Model>.

+ Add<Model>(<Model>): отправляет запрос на сервер по добавлению переданного <Model>.

***Разница в функциональности исходя из роли:***

* UserService: к удалению, добавлению и регистрации имеет доступ только Admin.
* ClientService: к удалению, добавлению и регистрации имеет доступ только Manager и Admin. Worker получает только заданных ему клиентов при использовании GetClients().
* ReportService: к удалению, добавлению и добавлению имеет доступ только Admin и Manager.
* ProductService: к удалению, добавлению и добавлению имеет доступ только Admin и Manager.
* CeremonyService: к удалению, добавлению и добавлению имеет доступ только Admin и Manager.

**View:**

***Общие страницы:***

* *AppShell*: представляет меню приложения через которое можно перейти на необходимую страничку. Изначально содержит функционал по переходу на страницу настроек соединения (ConnectionSettingsPage) и страницы для авторизации (LoginPage). Пользуется классом AppShellViewModel.
* *LoadingPage*: стартовая страница приложения, которая в зависимости от содержащихся в локальном хранилище настроек соединения и авторизации, пересылает на конкретные страницы. Пользуется классом LoadingViewModel.
* *LoginPage*: страница для авторизации. Пользуется классом LoginViewModel.
* *ConnectionSettingsPage*: страница для настроек соединения с сервером. Пользуется классом SettingsViewModel.
* *CalendarPage*: страница для просмотра расписания заказов (Orders). Пользуется классом CalendarViewModel.

***Страницы Админа:***

* *AdminUsersPage*: страница для просмотра, удаления, выбора для модификации, перехода на страницу добавления пользователей системы. Содержит список пользователей с личной информацией, контактными данными и ролью. Пользуется классом AdminUsersViewModel.
* *AdminUserDetailsPage*: страница для измения либо удаления информации о пользователе (UserInfo). Пользуется классом AdminUserDetailsViewModel.
* *AdminRegistrationPage*: страница для добавление аккаунта пользователю. Необходима информация о пользователе (UserInfo) и пароль. Пользуется AdminRegistrationViewModel.

***Страницы Менеджера:***

* *ManagerClientsPage*: страница для просмотра списка клиентов (Client), их удаление, перехода на страницу модификации и добавления. Также отображается работник (UserInfo) который назначен клиенту. Пользуется классом ManagerClientsViewModel.
* *ManagerClientsDetailsPage*: страница для измения либо удаления информации о клиенте (Client), назначения работника (UserInfo) для клиента, отправки информации клиенту о контактных данных работника по email. Пользуется классом ManagerClientDetailsViewModel.
* *ManagerClientRegistrationPage*: страница для добавление клиента (Client) и работника для него (UserInfo). Пользуется ManagerClientRegistrationViewModel.
* *MangerReportsPage*: страница для просмотра списка жалоб (Report), их удаление, обновление, добавление, содержит информации о работнике которому адресована жалоба, ее статус (Рассмотрена или нет) и возможность отправки email сообщения с ответом. Пользуется классом ManagerReportsViewModel.
* *ManagerInventoryPage*: страница для CRUD операций над типами продуктов (Product). Пользуется классом ManagerInventoryViewModel.
* *ManagerCeremonyPage*: страница для CRUD операций над типами цереионий (Ceremony). Пользуется классом ManagerCeremonyViewModel.

***Страницы Работника:***

* *WorkerClientsPage*: страница для просмотра клиентов (Client) и перехода на конкретного клиента для изменения его заказа (Order). Пользуется классом WorkerClientsViewModel.
* *WorkerClientDetailsPage*: страница для изменения заказа (Order) клиента (Client), добавление заказанных продуктов (ProductOrder), типа церемонии (Cermony), информации о погибшем человеке (DeadPerson) и отправка информации о заказе на email клиента. Пользуется классом WorkerClientDetailsViewModel.
* *WorkerReportsPage*: страница для просмотра жалоб (Report) на работника и перехода на конкретную для изменения. Пользуется классом WorkerReportsViewModel.
* *WorkerReportDetailsPage*: страница для просмотра информации о жалобе (Report), добавлении ответа на жалобу и изменении ее статуса. Пользуется классом WorkerReportDetailsViewModel.

**Вспомогательные классы:**

* AppRoleRouter: класс для добавление страничек в меню исходя из роли.

Методы:

+ AddFlyoutMenusDetails(): исходя из роли добавляет в меню соответствующие странички. Admin: AdminUsersPage, CalendarPage. Manager: ManagerClientsPage, ManagerReportsPage, ManagerInventoryPage, ManagerCeremonyPage, Calendar. Worker: WorkerClientsPage, WorkerReportsPage, Calendar.

**ViewModel:**

* *AppShellViewModel*:

Методы:

+ SignOut(): очищает локальное хранилище от информации пользователя, из меню убираются страницы соответсвующей роли, перемещает на страницу LoginPage.

+ GoToConnectionSettings(): перемещает на страницу ConnectionSettingsPage.

* *LoadingViewModel*:

Методы:

+ CheckUserLoginConnectionDetails(): проверяет авторизован ли пользователь и установлены ли настройки соединения, в зависимости от этого перемещает на соответствующие страницы. Если все установлено использует класс AppRoleRouter для добавление соответствующих ролям страниц.

* *LoginViewModel*:

Поля:

+ LoginInfo: объект класса LoginRequest, получает информацию от страницы для последующей авторизацией.

- \_service: объект класса, реализующего интерфейс ILoginService, используется для авторизации пользователя

Методы:

+ LoginViewModel(ILoginService): конструктор класса, переданный аргумент присваивается полю \_service.

+ Login(): авторизует пользователя используя метод ILoginService передавая информацию из поля LoginInfo. Использует класс AppRoleRouter для добавление соответствующих ролям страниц.

* *SettingsViewModel*:

Поля:

+ Host: строка, содержит ip адрес либо доменное имя хоста.

+ Port: строка, содержит порт который прослушивает сервер.

Методы:

+ Save(): сохраняет данные из полей Host и Port в локальное хранилище.

* *AdminUsersViewModel*:

Поля:

+ Users: список объектов класса UserInfo, к которому страница имеет доступ.

- \_service: объект класса, реализующего интерфейс IUserService, используется для CRUD операций над моделями пользователей.

Методы:

+ AdminUsersViewModel(IUserService): конструктор класса, переданный аргумент присваивается полю \_service.

+ ChooseUser(UserInfo): совершает переход на страницу AdminUserDetailsPage с пользователем, переданном в параметре.

+ GoToRegistration(): совершает переход на страницу добавление нового пользователя AdminRegistrationPage.

+ GetUsers(): присваивает полю Users список пользователей полученного из сервиса из поля \_service.

+ DeleteUser(UserInfo): использует сервис из поля \_service для удаление пользователя, который передан в параметр.

* *AdminUserDetailsViewModel*:

Поля:

+ User: содержит объект класса UserInfo, представляет из себя редактируемого пользователя.

- \_service: объект класса, реализующего интерфейс IUserService, используется для CRUD операций над моделями пользователей.

Методы:

+ AdminUserDetailsViewModel(IUserService): конструктор класса, переданный аргумент присваивается полю \_service.

+ UpdateUser(): использует сервис из поля \_service для обновления пользователя из поля User.

+ DeleteUser(): использует сервис из поля \_service для удаление пользователя из поля User.

* *AdminRegistrationViewModel*:

Поля:

+ User: содержит объект класса UserInfo, представляет из себя добавляемого пользователя.

+ Password: строка, содержит пароль для регистрации пользователя.

- \_service: объект класса, реализующего интерфейс IUserService, используется для CRUD операций над моделями пользователей.

Методы:

+ AdminRegistrationViewModel(IUserService): конструктор класса, переданный аргумент присваивается полю \_service.

+ RegisterUser(): использует сервис из поля \_service для регистрации пользователя из поля User, используя пароль из поля Password.

* *ManagerClientsViewModel*:

Поля:

+ Clients: список пар объектов класса Client и string, представляет из себя список клиентов и полное имя пользователя, который работает с клиентом.

- \_userService: объект класса, реализующего интерфейс IUserService, используется для получения списка пользователей.

- \_clientService: объект класса, реализующего интерфейс IClientService, используется для CRUD операций над моделями клиентов.

Методы:

+ ManagerClientsViewModel(IUserService, IClientService): конструктор класса, переданные аргументы присваивается полям \_userService, \_clientService соответственно.

+ ChooseClient(Client): совершает переход на страницу ManagerClientDetailsPage с клиентом, переданном в параметре.

+ GoToRegistration(): совершает переход на страницу добавление нового клиента ManagerClientRegistrationPage.

+ GetClients(): присваивает полю Clients список пользователей и имен пользователей, которые работает с соответствующими клиентами. Получает список клиентов используя поле \_clientService и соответствующие имена пользователей из \_userService.

+ DeleteClient(Client): использует поле \_clientService для удаление пользователя, который передан в параметр.

* *ManagerClientDetailsPage*:

Поля:

+ ClientDetails: содержит объект класса Client, представляет из себя редактируемого клиента.

+ Worker: содержит объект класса UserInfo, представляет из себя пользователя, который работает с клиентом.

+ Workers: список объектов класса UserInfo, представляет из себя список пользователей для выбора, кто будет работать с клиентом.

- \_userService: объект класса, реализующего интерфейс IUserService, используется для получения списка пользователей.

- \_clientService: объект класса, реализующего интерфейс IClientService, используется для CRUD операций над моделями клиентов.

- \_emailSender: объект класса, реализующего интерфейс IEmailSender, используется для отправки email сообщений.

Методы:

+ ManagerClientDetailsViewModel(IUserService, IClientService, IEmailSender): конструктор класса, переданные аргументы присваивается полям \_userService, \_clientService, \_emailSender соответственно.

+ UpdateClient(): использует поле \_clientService для обновления клиента из поля ClientDetails.

+ DeleteClient(): использует поле \_clientService для удаления клиента из поля ClientDetails.

+ GetWorkers(): использует поле \_userService для получения списка пользователей для выбора. Полученный список присваивает полю Workers.

+ SendDetailsEmail(): отправляет письмо клиенту о том, к какому пользователю следует обращаться для заказа. Использует поле \_emailSender.

* *ManagerClientRegistrationViewModel*:

Поля:

+ ClientDetails: содержит объект класса Client, представляет из себя добавляемого клиента.

+ Workers: список объектов класса UserInfo, представляет из себя список пользователей для выбора, кто будет работать с клиентом.

- \_userService: объект класса, реализующего интерфейс IUserService, используется для получения списка пользователей.

- \_clientService: объект класса, реализующего интерфейс IClientService, используется для CRUD операций над моделями клиентов.

Методы:

+ ManagerClientRegistrationViewModel(IUserService, IClientService): конструктор класса, переданные аргументы присваивается полям \_userService, \_clientService соответственно.

+ RegisterClient(): использует поле \_clientService для регистрации нового клиента из поля ClientDetails.

+ GetWorkers(): использует поле \_userService для получения списка пользователей для выбора. Полученный список присваивает полю Workers.

* *ManagerReportsViewMode*:

Поля:

+ Reports: список пар объектов класса Report и string, представляет из себя список жалоб и имя пользователя, кому адресована жалоба.

+ ReportDetails: содержит объект класса Report, представляет из себя изменяемую или добавляемую жалобу.

+ Workers: список объектов класса UserInfo, представляет из себя список пользователей для выбора, кому отправлена жалоба.

- \_userService: объект класса, реализующего интерфейс IUserService, используется для получения списка пользователей.

- \_reportService: объект класса, реализующего интерфейс IReportService, используется для CRUD операций над моделями жалоб.

- \_emailSender: объект класса, реализующего интерфейс IEmailSender, используется для отправки email сообщений.

Методы:

+ ManagerClientDetailsViewModel(IUserService, IReportService, IEmailSender): конструктор класса, переданные аргументы присваивается полям \_userService, \_reportService, \_emailSender соответственно.

+ GetWorkers(): использует поле \_userService для получения списка пользователей для выбора. Полученный список присваивает полю Workers.

+ GetReports(): присваивает полю Reports список жалоб и имен пользователей, к которым адресованы жалобы. Получает список жалоб используя поле \_reportService и соответствующие имена пользователей из \_userService.

+ AddReport(): использует поле \_reportService для добавления новой жалобы из поля ReportDetails.

+ UpdateReport(): использует поле \_reportService для обновления жалобы из поля ReportDetails.

+ DeleteReport(): использует поле \_reportService для удаления жалобы из поля ReportDetails.

+ ChooseReport(Report): присваивает полю ReportDetails жалобу, переданную в параметр.

+ SendAnsweEmail(Report): отправляет письмо жалующемуся с ответом на жалобу. Использует поле \_emailSender.

* *ManagerInventoryViewMode*:

Поля:

+ Products: список объектов класса Product, представляет из себя список доступных для заказа продуктов.

+ ProductDetails: содержит объект класса Product, представляет из себя изменяемый или добавляемый продукт.

- \_service: объект класса, реализующего интерфейс IProductService, используется для CRUD операций над моделями продуктов.

Методы:

+ ManagerInventoryViewModel(IProductService): конструктор класса, переданный аргумент присваивается полю \_service.

+ GetProducts(): использует поле \_service для получения списка продуктов. Полученный список присваивает полю Products.

+ AddProduct(): использует поле \_service для добавления нового продукта из поля ProductDetails.

+ UpdateProduct(): использует поле \_service для обновления продукта из поля ProductDetails.

+ DeleteProduct(): использует поле \_service для удаления продукта из поля ProductDetails.

+ ChooseProduct(Product): присваивает полю ProductDetails продукт, переданный в параметр.

* *ManagerCeremonyViewMode*:

Поля:

+ Ceremonies: список объектов класса Ceremony, представляет из себя список доступных для заказа церемоний.

+ CeremonyDetails: содержит объект класса Ceremony, представляет из себя изменяемую или добавляемую церемонию.

- \_service: объект класса, реализующего интерфейс ICeremonyService, используется для CRUD операций над моделями церемоний.

Методы:

+ ManagerInventoryViewModel(ICeremonyService): конструктор класса, переданный аргумент присваивается полю \_service.

+ GetCeremonies(): использует поле \_service для получения списка церемоний. Полученный список присваивает полю Ceremonies.

+ AddCeremony(): использует поле \_service для добавления новой церемонии из поля CeremonyDetails.

+ UpdateCeremony(): использует поле \_service для обновления церемонии из поля CeremonyDetails.

+ DeleteCeremony(): использует поле \_service для удаления церемонии из поля CeremonyDetails.

+ ChooseCeremony(Ceremony): присваивает полю CeremonyDetails продукт, переданный в параметр.

* *CalendarViewModel*:

Поля:

+ Orders: список объектов класса Order, представляет из себя список заказов.

- \_service: объект класса, реализующего интерфейс IOrderService, используется для получения списка заказов.

Методы:

+ CalendarViewModel(ICalendarService): конструктор класса, переданный аргумент присваивается полю \_service.

+ GetOrders(): использует поле \_service для получения списка заказов. Полученный список присваивает полю Ceremonies.

* *WorkerClientsViewModel*:

Поля:

+ Clients: список объектов класса Client, представляет из себя список клиентов, предоставленных пользователю.

- \_service: объект класса, реализующего интерфейс IClientService, используется для получения списка клиентов пользователя.

Методы:

+ WorkerClientsViewModel(IClientService): конструктор класса, переданный аргумент присваивается полю \_service.

+ GetClients(): использует поле \_service для получения списка клиентов, с которыми работает пользователь. Полученный список присваивает полю Clients.

+ ChooseClient(Client): совершает переход на страницу WorkerClientDetailsPage с клиентом, переданном в параметре.

* *WorkerClientDetailsViewModel*:

Поля:

+ ClientDetails: объект класса Client, представляет из себя изменяемого клиента.

+ OrderDetails: объект класса Order, представляет из себя изменяемый заказ, который принадлежит клиенту.

+ DeadPersonDetails: объект класса DeadPerson, представляет из себя погибшего человека, который указан в заказе.

+ ProductOrders: список объектов класса ProductOrder, представляет из себя список заказанных продуктов.

+ Products: список объектов класса Product, представляет из себя список доступных для заказа продуктов.

+ Ceremonies: список объектов класса Ceremony, представляет из себя список доступных для заказа церемоний.

+ IsPossible: булево значение, представляет из себя возможность или невозможность совершить заказ исходя из даты мероприятия и количества продуктов на складе.

+ TotalPrice: вещественное число, содержит общую стоимость заказанных продуктов и церемонии.

- \_clientService: объект класса, реализующего интерфейс IClientService, используется для получения и изменения информации о клиенте.

- \_orderService: объект класса, реализующего интерфейс IOrderService, используется для получения и обновления заказа клиента.

- \_deadPersonService: объект класса, реализующего интерфейс IDeadPersonService, используется для получения информации о погибшем человеке, указанном в заказе.

- \_productOrderService: объект класса, реализующего интерфейс IProductOrderService, используется для получения списка заказанных продуктов.

- \_productService: объект класса, реализующего интерфейс IProductService, используется для получения списка продуктов, которые возможно заказать.

- \_ceremonyService: объект класса, реализующего интерфейс ICeremonyService, используется для получения церемонии указанной в заказе клиента.

- \_emailSender: объект класса, реализующего интерфейс IEmailSender, используется для отправки email сообщений.

Методы:

+ WorkerClientDetails(IClientService, IOrderService, IDeadPersonService, IProductOrderService, IProductService, ICeremonyService, IEmailSender): конструктор класса, переданные аргументы присваивается полям \_clientService, \_orderService, \_deadPersonService, \_productOrderService, \_productService, \_ceremonyService, \_emailSender соответственно.

+ GetOrderDetails(): использует поле \_OrderService для получения заказа клиента. Полученный заказ присваивает полю OrderDetails.

+ GetDeadPersonDetails(): использует поле \_deadPersonService для получения информации о погибшем человеке, который указан в заказе клиента. Полученная информация присваивается полю DeadPersonDetails.

+ GetProductOrders(): использует поле \_ProductOrderService для получения списка заказанных продуктов клиента. Полученная информация присваивается полю ProductOrders.

+ GetProducts(): использует поле \_ProductService для получения списка возможных для заказа продуктов. Полученная информация присваивается полю Products.

+ GetCeremonies(): использует поле \_CeremonyService для получения списка возможных для заказа церемоний. Полученная информация присваивается полю Ceremonies.

+ UpdateClientOrder(): использует поле \_orderService, \_productOrdersService, \_deadPersonService для обновления данных о заказе. Если не возможно завершить заказ полю IsPossible присваивается значение False, иначе используя поле \_clientService обновляет информацию о завершенности заказа клиента.

+ SendDetailsEmail(): отправляет письмо клиенту с информацией о заказе. Использует поле \_emailSender.

* *WorkerReportsViewModel*:

Поля:

+ Reports: список объектов класса Report, представляет из себя список жалоб на пользователя.

- \_service: объект класса, реализующего интерфейс IReportService, используется для получения списка жалоб.

Методы:

+ WorkerReportsViewModel(IReportService): конструктор класса, переданный аргумент присваивается полю \_service.

+ GetReports(): использует поле \_service для получения списка жалоб. Полученный список присваивает полю Reports.

+ ChooseReport(Report): совершает переход на страницу WorkerReportDetailsPage с жалобой, переданном в параметре.

* *WorkerReportDetailsViewModel*:

Поля:

+ ReportDetails: объект класса Report, представляет из себя изменяемую жалобу.

- \_service: объект класса, реализующего интерфейс IReportService, используется для обновления жалоб.

Методы:

+ WorkerReportDetailsViewModel(IReportService): конструктор класса, переданный аргумент присваивается полю \_service.

+ UpdateReport(Report): использует поле \_service для обновления жалобы из поля ReportDetails.