Листинг А.1 - Метод «быстрого» поиска докторов

```
	public async Task<IEnumerable<Doctor>>
GetFastSearchedDoctorsAsync(string 		searchTerm)
	{
		var values = searchTerm.Trim().Split(" ");

		var doctorComparer = new DoctorComparer();

			var doctors = new HashSet<Doctor>(doctorComparer);

		foreach(var value in values)
			{
				IEnumerable<Doctor> temp = await FindByCondition(doc
=>
	doc.User.FirstName.ToLower().Contains(value.ToLower())
				||
doc.User.LastName.ToLower().Contains(value.ToLower())
			|| doc.User.Country.ToLower().Contains(value.ToLower())
				||
doc.User.Country.ToLower().Contains(value.ToLower())
			|| doc.Specialization.Name.ToLower()
			.Contains(value.ToLower())).ToListAsync();

				foreach (var item in temp) doctors.Add(item);
		}

		return doctors;
	}
```

Листинг А.2 - Метод регистрации нового пользователя

```
	public async Task<AuthenticationResultDto>
		RegisterAsync(UserRegistrationDto registrationDto)
	{
		var existingUser = await
			_userManager.FindByEmailAsync(registrationDto.Email);

		if (existingUser != null) return new AuthenticationResultDto
		{
			ErrorMessages = new List<string> { "Данная электронная
почта 				уже зарегистрирована, попробуйте другую!" }
		};

		existingUser = await _userManager.
			FindByNameAsync(registrationDto.UserName);

		if (existingUser != null) return new AuthenticationResultDto
		{
```

```
                ErrorMessages = new List<string> { "Данное имя пользователя
                    уже зарегистрировано!" }
            };

            var newUser = _mapper.Map<User>(registrationDto);
            if (registrationDto.Password != registrationDto.ConfirmPassword)
                return new AuthenticationResultDto
                {
                    ErrorMessages = new List<string> { "Пароль
подтверждён                         неверно!" }
                };

            var createdUser = await _userManager.CreateAsync(newUser,
                registrationDto.Password);

            if (!createdUser.Succeeded)
                return new AuthenticationResultDto
                {
                    ErrorMessages = createdUser.Errors.Select(err =>
                        err.Description)
                };

            if (!string.IsNullOrEmpty(newUser.Type))
            {
                var result = await _userManager.AddToRoleAsync(newUser,
                    newUser.Type);
                if(!result.Succeeded)
                return new AuthenticationResultDto
                {
                    ErrorMessages = result.Errors.Select(err =>
                        err.Description)
                };
            }

            var token = await
                _userManager.GenerateEmailConfirmationTokenAsync(newUser);
            var confirmationLink = $"http://localhost:3000/emailConfirmed?
                token={token}&email={newUser.Email}";

            var message = new MessageOptions(new string[] { newUser.Email },
                "Письмо подтверждения", confirmationLink);

            await _emailService.SendConfirmEmail(message);

            return await GenerateAuthResultForUser(newUser);
        }
```

Листинг А.3 - Метод вычисления списка доступного времени для записи к доктору на определённую дату

```
        public async Task<IEnumerable<AvailableAppointmentTimeDto>>
            Handle(GetAvailableAppointmentTimeForDateCommand command,
```

```
            CancellationToken cancellationToken)
      {
            var availableTime = new List<AvailableAppointmentTimeDto>();

            var doctor = await

      _repository.Doctor.GetDoctorByDoctorIdAsync(command.doctorId);

            if (doctor == null) return availableTime;

            var doctorAppointments = await _repository.Appointment.
                  GetAllAppointmentsForDoctorAsync(command.doctorId);

            var appointmentsTime = doctorAppointments.Where(app =>
                  app.StartDate.Date.Equals(command.date.Date)).
                  Select(app => app.StartTime);

            var doctorScedule = await _repository.DoctorsScedule.
                  GetDoctorsScedule(command.doctorId);

            var doctorWorkingDays = string.Join(" ", doctorScedule.
                  Select(sced => sced.dayOfWeek.ToString()));

           if  (!doctorWorkingDays.Contains(command.date.DayOfWeek.ToString())
                  || command.date.Date < DateTime.UtcNow.Date)
                  return availableTime;

            var doctorDayOffs = await _repository.DoctorDayOff.
                  GetDoctorsDayOffsByDoctorId(command.doctorId);

            var unavailableDays = doctorDayOffs.Where(day => day.Status !=
                  DoctorStatusEnum.Available).Select(day => day.Date);

            foreach (var date in unavailableDays)
                  if (command.date.Date.Equals(date.Date)) return
availableTime;

            var time = doctor.WorkDayStart;

            while(time < doctor.WorkDayEnd)
            {
                  if (DateTime.UtcNow.Date == command.date.Date &&
                        DateTime.Now.TimeOfDay > time)
                  {
                        time = time.Add(new TimeSpan(0,
                              doctor.AverageAppointmentTime, 0));
                        continue;
                  }

                  bool available = true;

                  foreach (var appTime in appointmentsTime)
                        if (time >= appTime && time <= appTime.Add(new
                        TimeSpan(0, doctor.AverageAppointmentTime, 0)))
                                    available = false;
```

```
            if (available && time > doctor.DinnerStart.Subtract(new
            TimeSpan(0, doctor.AverageAppointmentTime, 0)) && time <
            doctor.DinnerEnd) available = false;

            if (available) availableTime.Add(new
            AvailableAppointmentTimeDto
            {
                  doctorId = command.doctorId,
                  Time = $"{time.Hours}:{time.Minutes}"
            });

            time = time.Add(new TimeSpan(0,
                  doctor.AverageAppointmentTime, 0));
      }

      return availableTime;
}
```

Листинг А.4 - Тест GetDoctorsCommand_ShouldReturnAllDoctors

```
[Fact]
public async Task GetDoctorsCommand_ShouldReturnAllDoctors()
{
      var command = new GetDoctorsCommand()
      {
            doctorParameters = new DoctorParameters { }
      };

      var doctorMock = new Mock<IDoctorRepository>();
      doctorMock.Setup(repo =>
            repo.GetAllDoctorsAsync(command.doctorParameters,
            false)).Returns(TestDoctors());

      var mockRepoManager = new Mock<IRepositoryManager>();
      mockRepoManager.Setup(repo =>
            repo.Doctor).Returns(doctorMock.Object);

      var mockConfigurationRoot = new Mock<IConfigurationRoot>();
      mockConfigurationRoot.SetupGet(config =>
            config[It.IsAny<string>()]).Returns("some setting");

      var mapperMock = new Mock<IMapper>();
      mapperMock.Setup(mapper => mapper.Map<IEnumerable<DoctorDto>>(It.
            IsAny<IEnumerable<Doctor>>())).Returns(TestDoctorsDto());

      var commandHandler = new
            GetDoctorsCommandHandler(mockRepoManager.Object,
            mapperMock.Object);

      var doctorsDto = await commandHandler.Handle(command,
            CancellationToken.None);

      Assert.NotNull(doctorsDto);
```

```
        Assert.IsAssignableFrom<IEnumerable<DoctorDto>>(doctorsDto);
        Assert.Equal(TestDoctorsDto().Count(), doctorsDto.Count());
    }
```

Листинг А.5 - Тест GetDoctorCommand_ShouldReturnNull

```
    [Fact]
    public async Task GetDoctorCommand_ShouldReturnNull()
    {
        var command = new GetDoctorCommand()
        {
            doctorId = Guid.NewGuid()
        };

        var doctorMock = new Mock<IDoctorRepository>();
        doctorMock.Setup(repo =>
            repo.GetDoctorByDoctorIdAsync(command.doctorId,
            false)).Returns(GetTestDoctor(command.doctorId));

        var mockRepoManager = new Mock<IRepositoryManager>();
        mockRepoManager.Setup(repo =>
            repo.Doctor).Returns(doctorMock.Object);

        var mockConfigurationRoot = new Mock<IConfigurationRoot>();
        mockConfigurationRoot.SetupGet(config =>
            config[It.IsAny<string>()]).Returns("some setting");

        var mapperMock = new Mock<IMapper>();
        mapperMock.Setup(mapper =>
            mapper.Map<DoctorDto>(It.IsAny<Doctor>())).
            Returns(GetTestDoctorDto(command.doctorId));

        var commandHandler = new
            GetDoctorCommandHandler(mockRepoManager.Object,
            mapperMock.Object);

        var doctorDto = await commandHandler.Handle(command,
            CancellationToken.None);

        Assert.Null(doctorDto);
    }
```

Листинг А.6 - Тест CreateDoctorCommand_ShouldCreateNewDoctor

```
    [Fact]
    public async Task CreateDoctorCommand_ShouldCreateNewDoctor()
    {
        var command = new CreateDoctorCommand()
        {
            doctor = new DoctorForCreationDto { }
        };

        var doctorId = Guid.NewGuid();
```

```
        var doctorMock = new Mock<IDoctorRepository>();
        doctorMock.Setup(repo => repo.CreateDoctor(new Doctor { Id =
              doctorId }));

        var mockRepoManager = new Mock<IRepositoryManager>();
        mockRepoManager.Setup(repo =>
              repo.Doctor).Returns(doctorMock.Object);
        mockRepoManager.Setup(repo =>
              repo.SaveAsync()).Returns(Task.CompletedTask);

        var mockConfigurationRoot = new Mock<IConfigurationRoot>();
        mockConfigurationRoot.SetupGet(config =>
              config[It.IsAny<string>()]).Returns("some setting");

        var mapperMock = new Mock<IMapper>();
        mapperMock.Setup(mapper =>
              mapper.Map<Doctor>(It.IsAny<DoctorForCreationDto>())).
              Returns(new Doctor { Id = doctorId });

        var commandHandler = new
              CreateDoctorCommandHandler(mockRepoManager.Object,
              mapperMock.Object);

        var id = await commandHandler.Handle(command,
              CancellationToken.None);

        Assert.True(id == doctorId);
    }
```

Листинг А.7 - Тест When doctor go to notifications page, notifications page should appear

```
  it("When doctor go to notifications page, notifications page should appear",
      () => {
            cy.get("body").then((body) => { if (
                  body.find("button[type=submit]")[0] &&
                        body.find("button[type=submit]")[0].innerText ===
                        "Войти" )
                  {
                        cy.findByLabelText("Имя пользователя или эл.
                                          почта:").type( "TestDoctor", );
                        cy.findByLabelText("Пароль:").type("Password123!");
                                          cy.contains("Войти").click(); } });
                        cy.contains("Добро пожаловать!");
```

```
                    cy.findByText("Уведомления").click();
                    cy.url().should("include", "notifications");
                }
    );
```

Листинг А.8 - Тест When trying sign in with blank inputs, error messsage should appear

```
it("When trying sign in with blank inputs, error messsage should appear", ()
    => {
          cy.contains("Вход в аккаунт");
          cy.contains("Войти").click();
          cy.contains("Логин обязателен!");
          cy.contains("Пароль обязателен!");
          cy.findByLabelText("Имя пользователя или эл. почта:").type("A");
          cy.findByLabelText("Пароль:").type("A");
          cy.contains("Войти").click();
          cy.contains("Неверные данные пользователя!");
       }
   );
```

Листинг А.9 - Тест When trying register doctor with wrong data in inputs, error messsages should appear

```
it("When trying register doctor with wrong data in inputs, error messsages
    should appear",
() => {
          cy.contains("Вход в аккаунт");
          cy.contains("Зарегистрироваться").click();
          cy.contains("Зарегистрироваться как:");
          cy.contains("Доктор").click();
          cy.get("form").submit();
          cy.contains("Имя обязательно!");
          cy.contains("Фамилия обязательна!");
          cy.contains("Дата рождения обязательна!");
          cy.contains("Дата начала работы обязательна!");
          cy.contains("Почтовый код обязателен!");
          cy.contains("Номер телефона обязателен!");
```

```
        cy.contains("Страна обязательна!");
        cy.contains("Город обязателен!");
        cy.contains("Специализация обязательна!");
        cy.contains("Продолжительность приёма обязательна!");
        cy.contains("Время начала рабочего дня обязательно!");
        cy.contains("Время конца рабочего дня обязательно!");
        cy.contains("Время начала обеденного перерыва обязательно!");
        cy.contains("Время конца обеденного перерыва обязательно!");
        cy.contains("Номер лицензии обязателен!");
        cy.contains("Адрес работы обязателен!");
        cy.contains("Имя пользователя обязательно!");
        cy.contains("Почта обязательна!");
        cy.contains("Пароль обязателен!");
        cy.contains("Подтверждение пароля обязательно!");
        cy.findByLabelText("Имя").type("A", { force: true });
        cy.findByLabelText("Фамилия").type("A", { force: true });
        cy.findByLabelText("Почтовый код").type("A", { force: true });
        cy.findByLabelText("Номер телефона").type("A", { force: true });
        cy.findByLabelText("Страна").type("A", { force: true });
        cy.findByLabelText("Город").type("A", { force: true });
        cy.findByLabelText( "Примерная продолжительность вашего приёма(в
              минутах):", ).type("1", { force: true });
        cy.findByLabelText("Номер лицензии").type("A", { force: true });
        cy.findByLabelText("Адрес работы").type("A", { force: true });
       cy.findByLabelText("Имя пользователя").type("A", { force: true });
        cy.findByLabelText("Эл. почта").type("A", { force: true });
        cy.findByLabelText("Пароль").type("A", { force: true });
        cy.findByLabelText("Подтвердите пароль").type("A", { force:
true });    cy.contains("Имя должно состоять минимум из 2 символов!");
        cy.contains("Фамилия должна состоять минимум из 2 символов!");
        cy.contains("Дата рождения обязательна!"); cy.contains("Дата
начала            работы обязательна!");
        cy.contains("Код должен состоять из 5 цифр!");
        cy.contains("Неверный формат номера!");
        cy.contains("Название страны не может быть меньше 3 символов!");
        cy.contains("Название города не может быть меньше 3 символов!");
        cy.contains("5 минут - минимальное время");
```

```
            cy.contains("Имя пользователя должно состоять минимум из 2
                  символов!");
            cy.contains("Неверный формат адреса почты!");
            cy.contains("Пароль должен состоять минимум из 8 символов!");
            cy.findByLabelText("Имя").type( "Aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
      aa          aaaaaaaaaaaaaaaaaaaaaaaaaaaaa", { force: true }, );
            cy.findByLabelText("Фамилия").type( "Aaaaaaaaaaaaaaaaaaaaaaaaaaa
      aa
                  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa", { force: true }, );
            cy.findByLabelText("Страна").type( "Aaaaaaaaaaaaaaaaaaaaaaaaaaaa
      aa          aaaaaaaaaaaaaaaaaaaaaaaaaaaaa", { force: true }, );
            cy.findByLabelText("Город").type( "Aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
      aa          aaaaaaaaaaaaaaaaaaaaaaaaaaaaa", { force: true }, );
            cy.findByLabelText( "Примерная продолжительность вашего приёма(в
                  минутах):", ).type("100", { force: true });
            cy.findByLabelText("Имя

            пользователя").type( "Aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
                  aaaaaaaaaaaaaaaaaaaaaaaaa", { force: true }, );
            cy.findByLabelText("Пароль").type( "Aaaaaaaaaaaaaaaaaaaaaaaaaaaa
            aa    aaaaaaaaaaaaaaaaaaaaaaaaaaaaa", { force: true }, );
            cy.contains("Имя должно состоять максимум из 50 символов!");
            cy.contains("Фамилия должна состоять максимум из 50 символов!");
            cy.contains("Название страны не может быть больше 50 символов!");
            cy.contains("Название города не может быть больше 50 символов!");
            cy.contains("45 минут - максимальное время");
            cy.contains("Имя пользователя должно состоять максимум из 20
                  символов!");
            cy.contains("Пароль должен содержать минимум 1 цифру");
      }
);
```