

Smart Stay Tests

Generated by Doxygen 1.10.0

1 Namespace Index	1
1.1 Namespace List	1
2 Data Structure Index	3
2.1 Data Structures	3
3 File Index	5
3.1 File List	5
4 Namespace Documentation	7
4.1 SmartStay Namespace Reference	7
4.2 SmartStay.Core Namespace Reference	7
4.3 SmartStay.Core.Tests Namespace Reference	7
4.4 SmartStay.Core.Tests.Models Namespace Reference	7
4.4.1 Detailed Description	8
4.5 SmartStay.Core.Tests.Repositories Namespace Reference	9
4.5.1 Detailed Description	9
4.6 SmartStay.Core.Tests.Services Namespace Reference	9
4.6.1 Detailed Description	10
4.7 SmartStay.IO Namespace Reference	10
4.8 SmartStay.IO.Tests Namespace Reference	10
4.9 SmartStay.IO.Tests.Extensions Namespace Reference	10
4.9.1 Detailed Description	11
4.10 SmartStay.IO.Tests.FileOperations Namespace Reference	11
4.10.1 Detailed Description	11
4.11 SmartStay.Validation Namespace Reference	11
4.12 SmartStay.Validation.Tests Namespace Reference	11
4.12.1 Detailed Description	12
4.13 SmartStay.Validation.Tests.Validators Namespace Reference	12
4.13.1 Detailed Description	13
5 Data Structure Documentation	15
5.1 SmartStay.Core.Tests.Repositories.AccommodationsTests Class Reference	15
5.1.1 Detailed Description	15
5.1.2 Member Function Documentation	16
5.1.2.1 Add_ValidAccommodation_AddsAccommodationSuccessfully()	16
5.1.2.2 Export_ValidData_ExportsAccommodations()	16
5.1.2.3 FindAccommodationById_ExistingId_ReturnsAccommodation()	16
5.1.2.4 FindAccommodationById_NonExistingId_ReturnsNull()	16
5.1.2.5 Import_ValidData_ImportsAccommodations()	16
5.1.2.6 Load_ValidFile_LoadsAccommodations()	17
5.1.2.7 Remove_NonExistingAccommodation_ReturnsFalse()	17
5.1.2.8 Remove_ValidAccommodation_RemovesAccommodationSuccessfully()	17
5.1.2.9 Save_ValidData_SavesToFile()	17

5.2 SmartStay.Core.Tests.Models.AccommodationTests Class Reference	17
5.2.1 Detailed Description	18
5.2.2 Member Function Documentation	18
5.2.2.1 Constructor_InvalidName_ThrowsValidationException()	18
5.2.2.2 Constructor_ValidParameters_InitializesAccommodation()	18
5.2.2.3 FindRoomById_RoomDoesNotExist_ReturnsNull()	19
5.2.2.4 FindRoomById_RoomExists_ReturnsRoom()	19
5.2.2.5 Owner_ToString_ReturnsJson()	19
5.2.2.6 OwnerId_InvalidValue_ThrowsException()	19
5.2.2.7 OwnerId_SetAndGet_CorrectlySetsOwnerId()	19
5.3 SmartStay.Validation.Tests.Validators.AccommodationValidatorTests Class Reference	20
5.3.1 Detailed Description	20
5.3.2 Member Function Documentation	20
5.3.2.1 IsValidAccommodationId_InvalidAccommodationId_ReturnsFalse()	20
5.3.2.2 IsValidAccommodationId_ValidAccommodationId_ReturnsTrue()	21
5.3.2.3 IsValidAccommodationType_InvalidAccommodationType_ReturnsFalse()	21
5.3.2.4 IsValidAccommodationType_ValidAccommodationType_ReturnsTrue()	21
5.3.2.5 ValidateAccommodationId_InvalidAccommodationId_ThrowsValidationException()	21
5.3.2.6 ValidateAccommodationId_ValidAccommodationId_ReturnsAccommodationId()	21
5.3.2.7 ValidateAccommodationType_InvalidAccommodationType_ThrowsValidationException()	22
5.3.2.8 ValidateAccommodationType_ValidAccommodationType_ReturnsAccommodationType()	22
5.4 SmartStay.Validation.Tests.Validators.AddressValidatorTests Class Reference	22
5.4.1 Detailed Description	22
5.4.2 Member Function Documentation	23
5.4.2.1 IsValidAddress_InvalidAddress_ReturnsFalse()	23
5.4.2.2 IsValidAddress_ValidAddress_ReturnsTrue()	23
5.4.2.3 ValidateAddress_InvalidAddress_ThrowsValidationException()	23
5.4.2.4 ValidateAddress_ValidAddress_ReturnsAddress()	23
5.5 SmartStay.Core.Tests.Services.BookingManagerTests Class Reference	23
5.5.1 Detailed Description	25
5.5.2 Member Function Documentation	26
5.5.2.1 CreateAccommodation_CreatesAccommodation_WhenOwnerExists()	26
5.5.2.2 CreateAccommodation_ThrowsEntityNotFoundException_WhenOwnerDoesNotExist()	26
5.5.2.3 CreateBasicClient_CreatesClient_WhenValidInput()	26
5.5.2.4 CreateBasicClient_ThrowsClientCreationException_WhenValidationFails()	26
5.5.2.5 CreateBasicOwner_CreatesOwner_WhenValidDataIsProvided()	26
5.5.2.6 CreateBasicOwner_ThrowsException_WhenEmailsInvalid()	27
5.5.2.7 CreateCompleteClient_CreatesClient_WhenValidInput()	27
5.5.2.8 CreateCompleteClient_ThrowsClientCreationException_WhenValidationFails()	27
5.5.2.9 CreateCompleteOwner_CreatesOwner_WhenValidDataIsProvided()	27
5.5.2.10 CreateCompleteOwner_ThrowsException_WhenPhoneNumberIsInvalid()	27
5.5.2.11 FindClientById_ReturnsClient_WhenClientExists()	28

5.5.2.12 FindClientById_ThrowsArgumentException_WhenClientNotFound()	28
5.5.2.13 FindOwnerById_FindsOwner_WhenOwnerExists()	28
5.5.2.14 FindOwnerById_ThrowsException_WhenOwnerDoesNotExist()	28
5.5.2.15 RemoveAccommodation_ReturnsAccommodationNotFound_WhenAccommodationCannotBeFound()	28
5.5.2.16 RemoveAccommodation_ReturnsAccommodationNotFound_WhenAccommodationDoesNotExist()	29
5.5.2.17 RemoveAccommodation_SuccessfullyRemovesAccommodation_WhenAccommodationAndOwnerExist()	29
5.5.2.18 RemoveClient_RemovesClient_WhenClientExists()	29
5.5.2.19 RemoveClient_ReturnsFalse_WhenClientDoesNotExist()	29
5.5.2.20 RemoveOwner_RemovesOwner_WhenOwnerExists()	29
5.5.2.21 RemoveOwner_ReturnsFalse_WhenOwnerDoesNotExist()	30
5.5.2.22 SaveAll_CreatesFiles_WhenCalled()	30
5.5.2.23 UpdateAccommodation_ReturnsAccommodationNotFound_WhenAccommodationDoesNotExist()	30
5.5.2.24 UpdateAccommodation_ReturnsInvalidAddress_WhenInvalidAddressProvided()	30
5.5.2.25 UpdateAccommodation_ReturnsInvalidName_WhenInvalidNameProvided()	30
5.5.2.26 UpdateAccommodation_UpdatesAccommodation_WhenValidDataProvided()	31
5.5.2.27 UpdateClient_ReturnsClientNotFound_WhenClientDoesNotExist()	31
5.5.2.28 UpdateClient_ReturnsInvalidEmail_WhenEmailsInvalid()	31
5.5.2.29 UpdateClient_ReturnsInvalidFirstName_WhenFirstNamesInvalid()	31
5.5.2.30 UpdateClient_ReturnsInvalidLastName_WhenLastNamesInvalid()	31
5.5.2.31 UpdateClient_UpdatesClient_WhenValidData()	32
5.5.2.32 UpdateOwner_ReturnsInvalidEmail_WhenEmailsInvalid()	32
5.5.2.33 UpdateOwner_ReturnsInvalidFirstName_WhenFirstNamesInvalid()	32
5.5.2.34 UpdateOwner_ReturnsOwnerNotFound_WhenOwnerDoesNotExist()	32
5.5.2.35 UpdateOwner_UpdatesOwner_WhenValidDataProvided()	32
5.6 SmartStay.Core.Tests.Repositories.ClientsTests Class Reference	33
5.6.1 Detailed Description	33
5.6.2 Member Function Documentation	33
5.6.2.1 Add_ValidClient_AddsClientSuccessfully()	33
5.6.2.2 Export_ValidData_ExportsClients()	34
5.6.2.3 FindClientById_ExistingId_ReturnsClient()	34
5.6.2.4 FindClientById_NonExistingId_ReturnsNull()	34
5.6.2.5 Import_ValidData_ImportsClients()	34
5.6.2.6 Load_ValidFile_LoadsClients()	34
5.6.2.7 Remove_NonExistingClient_ReturnsFalse()	34
5.6.2.8 Remove_ValidClient_RemovesClientSuccessfully()	35
5.6.2.9 Save_ValidData_SavesToFile()	35
5.7 SmartStay.Core.Tests.Models.ClientTests Class Reference	35
5.7.1 Detailed Description	36
5.7.2 Member Function Documentation	36
5.7.2.1 Constructor_InvalidName_ThrowsValidationException()	36
5.7.2.2 Constructor_ValidParameters_InitializesClient()	36
5.7.2.3 Email_SetAndGet_ValidatesValue()	36

5.7.2.4	FirstName_SetAndGet_ValidatesValue()	36
5.7.2.5	Id_AutoGenerated_IsUniqueAndNonZero()	37
5.7.2.6	Payment_ToString_ReturnsJson()	37
5.7.2.7	PreferredPaymentMethod_SetAndGet_CorrectlyAssignsValue()	37
5.8	SmartStay.Validation.Tests.Validators.ClientValidatorTests Class Reference	37
5.8.1	Detailed Description	38
5.8.2	Member Function Documentation	38
5.8.2.1	IsValidClientId_InvalidId_ReturnsFalse()	38
5.8.2.2	IsValidClientId_ValidId_ReturnsTrue()	38
5.8.2.3	ValidateClientId_InvalidId_ThrowsValidationException()	38
5.8.2.4	ValidateClientId_ValidId_ReturnsClientId()	38
5.9	SmartStay.Validation.Tests.Validators.DateValidatorTests Class Reference	39
5.9.1	Detailed Description	39
5.9.2	Member Function Documentation	39
5.9.2.1	IsValidDateRange_InvalidDateRange_ReturnsFalse()	39
5.9.2.2	IsValidDateRange_ValidDateRange_ReturnsTrue()	40
5.9.2.3	IsValidFutureDate_InvalidDate_ReturnsFalse()	40
5.9.2.4	IsValidFutureDate_ValidDate_ReturnsTrue()	40
5.9.2.5	ValidateCheckInDate_InvalidDate_ThrowsValidationException()	40
5.9.2.6	ValidateCheckInDate_ValidDate_ReturnsCheckInDate()	40
5.9.2.7	ValidateCheckOutDate_InvalidDateRange_ThrowsValidationException()	41
5.9.2.8	ValidateCheckOutDate_ValidDateRange_ReturnsCheckOutDate()	41
5.10	SmartStay.Validation.Tests.Validators.EmailValidatorTests Class Reference	41
5.10.1	Detailed Description	42
5.10.2	Member Function Documentation	42
5.10.2.1	IsValidEmail_EmptyEmail_ReturnsFalse()	42
5.10.2.2	IsValidEmail_InvalidCharacters_ReturnsFalse()	42
5.10.2.3	IsValidEmail_MissingAtSymbol_ReturnsFalse()	42
5.10.2.4	IsValidEmail_MissingDomainExtension_ReturnsFalse()	42
5.10.2.5	IsValidEmail_NullEmail_ReturnsFalse()	43
5.10.2.6	IsValidEmail_ValidEmail_ReturnsTrue()	43
5.10.2.7	ValidateEmail_InvalidEmail_ThrowsValidationException()	43
5.10.2.8	ValidateEmail_ValidEmail_ReturnsEmail()	43
5.11	SmartStay.IO.Tests.Extensions.FileExtensionsTests Class Reference	43
5.11.1	Detailed Description	44
5.11.2	Member Function Documentation	44
5.11.2.1	EnsureDirectoryExists_DirectoryDoesNotExist_CreatesDirectory()	44
5.11.2.2	EnsureDirectoryExists_DirectoryExists_DoesNotThrowException()	44
5.11.2.3	EnsureDirectoryExists_NullOrEmptyPath_DoesNotThrowException()	44
5.11.2.4	EnsureDirectoryExists_RootDirectoryPath_DoesNotThrowException()	45
5.12	SmartStay.IO.Tests.FileOperations.FileHandlerTests Class Reference	45
5.12.1	Detailed Description	45

5.12.2 Member Function Documentation	45
5.12.2.1 ReadFile_EmptyPath_ThrowsArgumentException()	45
5.12.2.2 ReadFile_FileDoesNotExist_ThrowsFileNotFoundException()	46
5.12.2.3 ReadFile_ValidFilePath_ReturnsFileContent()	46
5.12.2.4 WriteFile_EmptyPath_ThrowsArgumentException()	46
5.12.2.5 WriteFile_NonExistentDirectory_CreatesDirectoryAndWritesFile()	46
5.12.2.6 WriteFile_ValidFilePath_WritesFileContent()	46
5.13 SmartStay.Validation.Tests.Validators.NameValidatorTests Class Reference	47
5.13.1 Detailed Description	47
5.13.2 Member Function Documentation	47
5.13.2.1 IsValidAccommodationName_InvalidName_ReturnsFalse()	47
5.13.2.2 IsValidAccommodationName_ValidName_ReturnsTrue()	48
5.13.2.3 IsValidName_InvalidName_ReturnsFalse()	48
5.13.2.4 IsValidName_ValidName_ReturnsTrue()	48
5.13.2.5 ValidateAccommodationName_TooLongName_ThrowsValidationException()	48
5.13.2.6 ValidateAccommodationName_ValidName_ReturnsName()	48
5.13.2.7 ValidateName_InvalidName_ThrowsValidationException()	49
5.13.2.8 ValidateName_TooLongName_ThrowsValidationException()	49
5.13.2.9 ValidateName_ValidName_ReturnsName()	49
5.14 SmartStay.Core.Tests.Repositories.OwnersTests Class Reference	49
5.14.1 Detailed Description	50
5.14.2 Member Function Documentation	50
5.14.2.1 Add_ValidOwner_AddsOwnerSuccessfully()	50
5.14.2.2 Export_ValidData_ExportsOwnersWithAccommodations()	50
5.14.2.3 FindOwnerById_ExistingId_ReturnsOwner()	51
5.14.2.4 FindOwnerById_NonExistingId_ReturnsNull()	51
5.14.2.5 Import_ValidData_ImportsOwners()	51
5.14.2.6 Load_ValidFile_LoadsOwners()	51
5.14.2.7 Remove_NonExistingOwner_ReturnsFalse()	51
5.14.2.8 Remove_ValidOwner_RemovesOwnerSuccessfully()	51
5.14.2.9 Save_ValidData_SavesToFile()	52
5.15 SmartStay.Core.Tests.Models.OwnerTests Class Reference	52
5.15.1 Detailed Description	52
5.15.2 Member Function Documentation	53
5.15.2.1 AddAccommodation_ValidAccommodation_AddsSuccessfully()	53
5.15.2.2 Constructor_AdditionalDetails_InitializesOwner()	53
5.15.2.3 Constructor_BasicDetails_InitializesOwner()	53
5.15.2.4 Email_SetAndGet_ValidatesValue()	53
5.15.2.5 FirstName_SetAndGet_ValidatesValue()	53
5.15.2.6 LastAssignedId_TracksCorrectly()	54
5.15.2.7 Owner_ToString_ReturnsJson()	54
5.15.2.8 PhoneNumber_SetAndGet_ValidatesValue()	54

5.15.2.9 RemoveAccommodation_ValidAccommodation_RemovesSuccessfully()	54
5.16 SmartStay.Validation.Tests.Validators.OwnerValidatorTests Class Reference	54
5.16.1 Detailed Description	55
5.16.2 Member Function Documentation	55
5.16.2.1 ValidateOwnerEmail_InvalidEmail_ThrowsValidationException()	55
5.16.2.2 ValidateOwnerEmail_ValidEmail_ReturnsEmail()	55
5.16.2.3 ValidateOwnerId_InvalidId_ThrowsValidationException()	56
5.16.2.4 ValidateOwnerId_ValidId_ReturnsId()	56
5.16.2.5 ValidateOwnerName_InvalidName_ThrowsValidationException()	56
5.16.2.6 ValidateOwnerName_ValidName_ReturnsName()	56
5.16.2.7 ValidateOwnerPhoneNumber_InvalidPhoneNumber_ThrowsValidationException()	56
5.16.2.8 ValidateOwnerPhoneNumber_ValidPhoneNumber_ReturnsPhoneNumber()	57
5.17 SmartStay.IO.Tests.FileOperations.PathValidatorTests Class Reference	57
5.17.1 Detailed Description	57
5.17.2 Member Function Documentation	58
5.17.2.1 FileExists_ExistingFile_ReturnsTrue()	58
5.17.2.2 FileExists_NonExistentFile_ReturnsFalse()	58
5.17.2.3 IsValidFileType_CaseInsensitiveExtensionComparison_ReturnsTrue()	58
5.17.2.4 IsValidFileType_InvalidExtension_ReturnsFalse()	58
5.17.2.5 IsValidFileType_NullOrEmptyFilePath_ThrowsArgumentException()	58
5.17.2.6 IsValidFileType_ValidExtension_ReturnsTrue()	59
5.18 SmartStay.Core.Tests.Models.PaymentTests Class Reference	59
5.18.1 Detailed Description	59
5.18.2 Member Function Documentation	59
5.18.2.1 Payment_InvalidAmount_ThrowsValidationException()	59
5.18.2.2 Payment_InvalidReservationId_ThrowsValidationException()	60
5.18.2.3 Payment_ToString_ReturnsJson()	60
5.18.2.4 Payment_UniqueIds_AssignsIncrementalIds()	60
5.18.2.5 Payment_UpdateInvalidStatus_ThrowsValidationException()	60
5.18.2.6 Payment_UpdateValidStatus_UpdatesStatus()	60
5.18.2.7 Payment_ValidData_CreatesPayment()	61
5.19 SmartStay.Validation.Tests.Validators.PaymentValidatorTests Class Reference	61
5.19.1 Detailed Description	62
5.19.2 Member Function Documentation	62
5.19.2.1 ValidatePayment_InvalidPayment_ThrowsValidationException()	62
5.19.2.2 ValidatePayment_ValidPayment_ReturnsPayment()	62
5.19.2.3 ValidatePaymentAmount_InvalidAmount_ThrowsValidationException()	62
5.19.2.4 ValidatePaymentAmount_ValidAmount_ReturnsAmount()	62
5.19.2.5 ValidatePaymentMethod_InvalidMethod_ThrowsValidationException()	63
5.19.2.6 ValidatePaymentMethod_ValidMethod_ReturnsMethod()	63
5.19.2.7 ValidatePaymentStatus_InvalidStatus_ThrowsValidationException()	63
5.19.2.8 ValidatePaymentStatus_ValidStatus_ReturnsStatus()	63

5.19.2.9 ValidatePrice_InvalidPrice_ThrowsValidationException()	63
5.19.2.10 ValidatePrice_ValidPrice_ReturnsPrice()	64
5.19.2.11 ValidateTotalCost_InvalidTotalCost_ThrowsValidationException()	64
5.19.2.12 ValidateTotalCost_ValidTotalCost_ReturnsTotalCost()	64
5.20 SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests Class Reference	64
5.20.1 Detailed Description	65
5.20.2 Member Function Documentation	65
5.20.2.1 IsValidPhoneNumber_InvalidPhoneNumber_ReturnsFalse()	65
5.20.2.2 IsValidPhoneNumber_ValidPhoneNumber_ReturnsTrue()	65
5.20.2.3 ValidatePhoneNumber_EmptyPhoneNumber_ThrowsValidationException()	65
5.20.2.4 ValidatePhoneNumber_InvalidPhoneNumber_ThrowsValidationException()	65
5.20.2.5 ValidatePhoneNumber_NullPhoneNumber_ThrowsValidationException()	66
5.20.2.6 ValidatePhoneNumber_ValidPhoneNumber_ReturnsPhoneNumber()	66
5.21 SmartStay.Core.Tests.Repositories.ReservationsTests Class Reference	66
5.21.1 Detailed Description	67
5.21.2 Member Function Documentation	67
5.21.2.1 Add_InvalidDateRange_ThrowsValidationException()	67
5.21.2.2 Add_InvalidTotalCost_ThrowsValidationException()	67
5.21.2.3 Add_NullReservation_ThrowsArgumentNullException()	67
5.21.2.4 Add_ValidReservation_AddsReservationSuccessfully()	68
5.21.2.5 Export_ValidData_ExportsReservationsWithPayments()	68
5.21.2.6 FindReservationById_ExistingId_ReturnsReservation()	68
5.21.2.7 FindReservationById_NonExistingId_ReturnsNull()	68
5.21.2.8 Import_ValidData_ImportsReservationsWithPayments()	68
5.21.2.9 Load_ValidFile_LoadsReservations()	69
5.21.2.10 Remove_NonExistingReservation_ReturnsFalse()	69
5.21.2.11 Remove_ValidReservation_RemovesReservationSuccessfully()	69
5.21.2.12 Save_ValidData_SavesToFile()	69
5.22 SmartStay.Core.Tests.Models.ReservationTests Class Reference	69
5.22.1 Detailed Description	70
5.22.2 Member Function Documentation	70
5.22.2.1 CheckIn_StatusNotPending_ReturnsFalse()	70
5.22.2.2 CheckIn_StatusPending_ChangesStatusToCheckedIn()	70
5.22.2.3 CheckOut_StatusCheckedIn_ChangesStatusToCheckedOut()	71
5.22.2.4 CheckOut_StatusNotCheckedIn_ReturnsFalse()	71
5.22.2.5 Constructor_ValidParameters_InitializesReservation()	71
5.22.2.6 IsFullyPaid_FullyPaid_ReturnsTrue()	71
5.22.2.7 MakePayment_ValidPayment_UpdatesAmountPaid()	71
5.22.2.8 ToString_ReturnsValidJson()	72
5.23 SmartStay.Validation.Tests.Validators.ReservationValidatorTests Class Reference	72
5.23.1 Detailed Description	72
5.23.2 Member Function Documentation	73

5.23.2.1 IsValidReservationId_InvalidReservationId_ReturnsFalse()	73
5.23.2.2 IsValidReservationId_ValidReservationId_ReturnsTrue()	73
5.23.2.3 IsValidReservationStatus_InvalidReservationStatus_ReturnsFalse()	73
5.23.2.4 IsValidReservationStatus_ValidReservationStatus_ReturnsTrue()	73
5.23.2.5 ValidateReservationId_InvalidReservationId_ThrowsValidationException()	73
5.23.2.6 ValidateReservationId_ValidReservationId_ReturnsReservationId()	74
5.23.2.7 ValidateReservationStatus_InvalidReservationStatus_ThrowsValidationException()	74
5.23.2.8 ValidateReservationStatus_ValidReservationStatus_ReturnsReservationStatus()	74
5.24 SmartStay.Core.Tests.Models.RoomTests Class Reference	74
5.24.1 Detailed Description	75
5.24.2 Member Function Documentation	75
5.24.2.1 AddReservation_ValidDates_AddsReservation()	75
5.24.2.2 CalculateTotalCost_EndDateBeforeStartDate_ThrowsArgumentException()	75
5.24.2.3 CalculateTotalCost_ValidDates_ReturnsCorrectCost()	75
5.24.2.4 Constructor_ValidParameters_InitializesRoom()	75
5.24.2.5 IsAvailable_NoOverlap_ReturnsTrue()	76
5.24.2.6 IsAvailable_Overlap_ReturnsFalse()	76
5.24.2.7 RemoveReservation_ExistingReservation_RemovesReservation()	76
5.24.2.8 ToString_ReturnsValidJson()	76
5.25 SmartStay.Validation.Tests.Validators.RoomValidatorTests Class Reference	76
5.25.1 Detailed Description	77
5.25.2 Member Function Documentation	77
5.25.2.1 IsValidAvailability_AnyStatus_ReturnsTrue()	77
5.25.2.2 IsValidRoomType_InvalidType_ReturnsFalse()	77
5.25.2.3 IsValidRoomType_ValidType_ReturnsTrue()	78
5.25.2.4 ValidateAvailability_ValidStatus_ReturnsAvailability()	78
5.25.2.5 ValidateRoomId_InvalidId_ThrowsValidationException()	78
5.25.2.6 ValidateRoomId_ValidId_ReturnsRoomId()	78
5.25.2.7 ValidateRoomType_InvalidType_ThrowsValidationException()	78
5.25.2.8 ValidateRoomType_ValidType_ReturnsRoomType()	79
5.26 SmartStay.Validation.Tests.ValidationErrorMessageTests Class Reference	79
5.26.1 Detailed Description	79
5.26.2 Member Function Documentation	79
5.26.2.1 GetErrorMessage_InvalidErrorCode_ReturnsFallbackMessage()	79
5.26.2.2 GetErrorMessage_SupportsLocalization()	80
5.26.2.3 GetErrorMessage_ValidErrorCode_ReturnsLocalizedMessage()	80
5.27 SmartStay.Validation.Tests.ValidationExceptionTests Class Reference	80
5.27.1 Detailed Description	80
5.27.2 Member Function Documentation	81
5.27.2.1 Constructor_WithErrorCode_SetsErrorCodeAndMessage()	81
5.27.2.2 Constructor_WithUnknownErrorCode_UsesFallbackMessage()	81

6 File Documentation	83
6.1 AccommodationTests.cs File Reference	83
6.2 AccommodationTests.cs	83
6.3 ClientTests.cs File Reference	85
6.4 ClientTests.cs	85
6.5 OwnerTests.cs File Reference	87
6.6 OwnerTests.cs	87
6.7 PaymentTests.cs File Reference	89
6.8 PaymentTests.cs	90
6.9 ReservationTests.cs File Reference	91
6.10 ReservationTests.cs	92
6.11 RoomTests.cs File Reference	94
6.12 RoomTests.cs	94
6.13 SmartStay.Core.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs File Reference	96
6.14 SmartStay.Core.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs	96
6.15 SmartStay.IO.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs File Reference	96
6.16 SmartStay.IO.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs	96
6.17 SmartStay.Validation.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs File Reference	96
6.18 SmartStay.Validation.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs	96
6.19 SmartStay.Core.Tests.AssemblyInfo.cs File Reference	97
6.20 SmartStay.Core.Tests.AssemblyInfo.cs	97
6.21 SmartStay.Core.Tests.GlobalUsings.g.cs File Reference	97
6.22 SmartStay.Core.Tests.GlobalUsings.g.cs	97
6.23 AccommodationsTests.cs File Reference	97
6.24 AccommodationsTests.cs	98
6.25 ClientsTests.cs File Reference	101
6.26 ClientsTests.cs	101
6.27 OwnersTests.cs File Reference	104
6.28 OwnersTests.cs	104
6.29 ReservationsTests.cs File Reference	107
6.30 ReservationsTests.cs	107
6.31 BookingManagerTests.cs File Reference	111
6.32 BookingManagerTests.cs	111
6.33 FileExtensionsTests.cs File Reference	122
6.34 FileExtensionsTests.cs	123
6.35 FileHandlerTests.cs File Reference	124
6.36 FileHandlerTests.cs	124
6.37 PathValidatorTests.cs File Reference	125
6.38 PathValidatorTests.cs	126
6.39 SmartStay.IO.Tests.AssemblyInfo.cs File Reference	127

6.40 SmartStay.IO.Tests.AssemblyInfo.cs	127
6.41 SmartStay.IO.Tests.GlobalUsings.g.cs File Reference	128
6.42 SmartStay.IO.Tests.GlobalUsings.g.cs	128
6.43 SmartStay.Validation.Tests.AssemblyInfo.cs File Reference	128
6.44 SmartStay.Validation.Tests.AssemblyInfo.cs	128
6.45 SmartStay.Validation.Tests.GlobalUsings.g.cs File Reference	128
6.46 SmartStay.Validation.Tests.GlobalUsings.g.cs	128
6.47 ValidationErrorMessageTests.cs File Reference	129
6.48 ValidationErrorMessageTests.cs	129
6.49 ValidationExceptionTests.cs File Reference	130
6.50 ValidationExceptionTests.cs	130
6.51 AccommodationValidatorTests.cs File Reference	131
6.52 AccommodationValidatorTests.cs	131
6.53 AddressValidatorTests.cs File Reference	132
6.54 AddressValidatorTests.cs	133
6.55 ClientValidatorTests.cs File Reference	134
6.56 ClientValidatorTests.cs	134
6.57 DateValidatorTests.cs File Reference	135
6.58 DateValidatorTests.cs	135
6.59 EmailValidatorTests.cs File Reference	137
6.60 EmailValidatorTests.cs	137
6.61 NameValidatorTests.cs File Reference	139
6.62 NameValidatorTests.cs	139
6.63 OwnerValidatorTests.cs File Reference	141
6.64 OwnerValidatorTests.cs	141
6.65 PaymentValidatorTests.cs File Reference	142
6.66 PaymentValidatorTests.cs	143
6.67 PhoneNumberValidatorTests.cs File Reference	145
6.68 PhoneNumberValidatorTests.cs	145
6.69 ReservationValidatorTests.cs File Reference	146
6.70 ReservationValidatorTests.cs	147
6.71 RoomValidatorTests.cs File Reference	148
6.72 RoomValidatorTests.cs	149

Index	151
--------------	------------

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

SmartStay	7
SmartStay.Core	7
SmartStay.Core.Tests	7
SmartStay.Core.Tests.Models The SmartStay.Core.Tests.Models namespace contains unit tests for the models used in the SmartStay application. These tests verify the correct behavior of the Accommodation class and its methods	7
SmartStay.Core.Tests.Repositories The SmartStay.Core.Tests.Repositories namespace contains unit tests for the repository classes that interact with the application data	9
SmartStay.Core.Tests.Services The SmartStay.Core.Tests.Services namespace contains unit tests for the services used in the SmartStay application. These tests verify the correct behavior of the Booking↔ Manager class and its methods	9
SmartStay.IO	10
SmartStay.IO.Tests	10
SmartStay.IO.Tests.Extensions The SmartStay.IO.Tests.Extensions namespace contains unit tests for the extension methods provided for file-related operations in the SmartStay application	10
SmartStay.IO.Tests.FileOperations The SmartStay.IO.Tests.FileOperations namespace contains unit tests for file op- erations used within the SmartStay application	11
SmartStay.Validation	11
SmartStay.Validation.Tests The SmartStay.Validation.Tests namespace contains unit tests for classes within the SmartStay.Validation namespace, ensuring correctness and reliability of validation functionalities	11
SmartStay.Validation.Tests.Validators The SmartStay.Validation.Tests.Validators namespace contains unit tests for the validation logic of different fields, specifically focusing on the AccommodationValidator class for validating accommodation types and IDs	12

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

SmartStay.Core.Tests.Repositories.AccommodationsTests	
Contains unit tests for the Accommodations repository class. Tests include adding, removing, importing, exporting accommodations, and serialization/deserialization processes	15
SmartStay.Core.Tests.Models.AccommodationTests	
Contains unit tests for the Accommodation class. Tests include validation, property assignments, room management, and string representation	17
SmartStay.Validation.Tests.Validators.AccommodationValidatorTests	
Contains unit tests for the AccommodationValidator class. Tests the validation logic for accommodation types and IDs	20
SmartStay.Validation.Tests.Validators.AddressValidatorTests	
Contains unit tests for the AddressValidator class. Tests the validation logic for addresses used in the SmartStay application	22
SmartStay.Core.Tests.Services.BookingManagerTests	
Contains unit tests for the BookingManager class. Tests the BookingManager.SaveAll method to ensure that it creates the necessary files when saving repositories	23
SmartStay.Core.Tests.Repositories.ClientsTests	
Contains unit tests for the Clients repository class. Tests include adding, removing, importing, exporting clients, and serialization/deserialization processes	33
SmartStay.Core.Tests.Models.ClientTests	
Contains unit tests for the Client class. Tests include validation, property assignments, and string representation	35
SmartStay.Validation.Tests.Validators.ClientValidatorTests	
Contains unit tests for the ClientValidator class. Tests the validation logic for client-related data in the SmartStay application	37
SmartStay.Validation.Tests.Validators.DateValidatorTests	
Contains unit tests for the DateValidator class. Tests the validation logic for dates such as check-in and check-out dates	39
SmartStay.Validation.Tests.Validators.EmailValidatorTests	
Contains unit tests for the EmailValidator class. Tests the validation logic for email addresses used in the SmartStay application	41
SmartStay.IO.Tests.Extensions.FileExtensionsTests	
Contains unit tests for the FileExtensions class. Tests the behavior of file-related extension methods	43
SmartStay.IO.Tests.FileOperations.FileHandlerTests	
Contains unit tests for the FileHandler class	45

SmartStay.Validation.Tests.Validators.NameValidatorTests	
Contains unit tests for the NameValidator class. Validates both general names and accommodation names, checking correct behavior when the names are valid or invalid	47
SmartStay.Core.Tests.Repositories.OwnersTests	
Contains unit tests for the Owners repository class. Tests include adding, removing, importing, exporting owners, and serialization/deserialization processes	49
SmartStay.Core.Tests.Models.OwnerTests	
Contains unit tests for the Owner class. Tests include validation, property assignments, and methods for managing accommodations	52
SmartStay.Validation.Tests.Validators.OwnerValidatorTests	
Contains unit tests for the OwnerValidator class. Tests the validation logic for owner-related data used in the SmartStay application	54
SmartStay.IO.Tests.FileOperations.PathValidatorTests	
Contains unit tests for the PathValidator class	57
SmartStay.Core.Tests.Models.PaymentTests	
Unit tests for the Payment class	59
SmartStay.Validation.Tests.Validators.PaymentValidatorTests	
Contains unit tests for the PaymentValidator class. Tests the validation logic for payment-related data in the SmartStay application	61
SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests	
Contains unit tests for the PhoneNumberValidator class. Tests the validation logic for phone numbers in the SmartStay application	64
SmartStay.Core.Tests.Repositories.ReservationsTests	
Contains unit tests for the Reservations repository class. Tests include adding, removing, importing, exporting reservations, and serialization/deserialization processes	66
SmartStay.Core.Tests.Models.ReservationTests	
Contains unit tests for the Reservation class. Tests include validation, property assignments, payment methods, and string representation	69
SmartStay.Validation.Tests.Validators.ReservationValidatorTests	
Contains unit tests for the ReservationValidator class. Tests the validation logic for reservation-related data in the SmartStay application	72
SmartStay.Core.Tests.Models.RoomTests	
Contains unit tests for the Room class. Tests include validation, property assignments, reservation management, cost calculation, and string representation	74
SmartStay.Validation.Tests.Validators.RoomValidatorTests	
Contains unit tests for the RoomValidator class. Tests the validation logic for room-related data used in the SmartStay application	76
SmartStay.Validation.Tests.ValidationErrorMessageTests	
Contains unit tests for the ValidationErrorMessage class. Ensures that error messages are correctly retrieved based on the provided error codes and that localization functions as expected	79
SmartStay.Validation.Tests.ValidationExceptionTests	
Contains unit tests for the ValidationException class. Ensures that exceptions are created with the expected error codes and messages	80

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

AccommodationTests.cs	83
ClientTests.cs	85
OwnerTests.cs	87
PaymentTests.cs	89
ReservationTests.cs	91
RoomTests.cs	94
SmartStay.Core.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs	96
SmartStay.IO.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs	96
SmartStay.Validation.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs	96
SmartStay.Core.Tests.AssemblyInfo.cs	97
SmartStay.Core.Tests.GlobalUsings.g.cs	97
AccommodationsTests.cs	97
ClientsTests.cs	101
OwnersTests.cs	104
ReservationsTests.cs	107
BookingManagerTests.cs	111
FileExtensionsTests.cs	122
FileHandlerTests.cs	124
PathValidatorTests.cs	125
SmartStay.IO.Tests.AssemblyInfo.cs	127
SmartStay.IO.Tests.GlobalUsings.g.cs	128
SmartStay.Validation.Tests.AssemblyInfo.cs	128
SmartStay.Validation.Tests.GlobalUsings.g.cs	128
ValidationErrorMessagesTests.cs	129
ValidationExceptionTests.cs	130
AccommodationValidatorTests.cs	131
AddressValidatorTests.cs	132
ClientValidatorTests.cs	134
DateValidatorTests.cs	135
EmailValidatorTests.cs	137
NameValidatorTests.cs	139
OwnerValidatorTests.cs	141
PaymentValidatorTests.cs	142
PhoneNumberValidatorTests.cs	145
ReservationValidatorTests.cs	146
RoomValidatorTests.cs	148

Chapter 4

Namespace Documentation

4.1 SmartStay Namespace Reference

Namespaces

- namespace [Core](#)
- namespace [IO](#)
- namespace [Validation](#)

4.2 SmartStay.Core Namespace Reference

Namespaces

- namespace [Tests](#)

4.3 SmartStay.Core.Tests Namespace Reference

Namespaces

- namespace [Models](#)
The [SmartStay.Core.Tests.Models](#) namespace contains unit tests for the models used in the [SmartStay](#) application. These tests verify the correct behavior of the Accommodation class and its methods.
- namespace [Repositories](#)
The [SmartStay.Core.Tests.Repositories](#) namespace contains unit tests for the repository classes that interact with the application data.
- namespace [Services](#)
The [SmartStay.Core.Tests.Services](#) namespace contains unit tests for the services used in the [SmartStay](#) application. These tests verify the correct behavior of the BookingManager class and its methods.

4.4 SmartStay.Core.Tests.Models Namespace Reference

The [SmartStay.Core.Tests.Models](#) namespace contains unit tests for the models used in the [SmartStay](#) application. These tests verify the correct behavior of the Accommodation class and its methods.

Data Structures

- class [AccommodationTests](#)
Contains unit tests for the Accommodation class. Tests include validation, property assignments, room management, and string representation.
- class [ClientTests](#)
Contains unit tests for the Client class. Tests include validation, property assignments, and string representation.
- class [OwnerTests](#)
Contains unit tests for the Owner class. Tests include validation, property assignments, and methods for managing accommodations.
- class [PaymentTests](#)
Unit tests for the Payment class.
- class [ReservationTests](#)
Contains unit tests for the Reservation class. Tests include validation, property assignments, payment methods, and string representation.
- class [RoomTests](#)
Contains unit tests for the Room class. Tests include validation, property assignments, reservation management, cost calculation, and string representation.

4.4.1 Detailed Description

The [SmartStay.Core.Tests.Models](#) namespace contains unit tests for the models used in the [SmartStay](#) application. These tests verify the correct behavior of the Accommodation class and its methods.

The [SmartStay.Core.Tests.Models](#) namespace contains unit tests for the models used in the [SmartStay](#) application. These tests verify the correct behavior of the Owner class and its methods.

The [SmartStay.Core.Tests.Models](#) namespace contains unit tests for the models used in the [SmartStay](#) application.

```
<copyright file="AccommodationTests.cs"> Copyright (c) 2024 Enrique Rodrigues. All Rights Reserved.
</copyright> <file> This file contains unit tests for the Accommodation class, which represents accommodations,
including their properties, methods, and validation logic in the SmartStay application. </file> <author>Enrique
Rodrigues</author> <date>03/12/2024</date>
```

```
<copyright file="ClientTests.cs"> Copyright (c) 2024 Enrique Rodrigues. All Rights Reserved. </copyright>
<file> This file contains unit tests for the Client class, which represents clients in the SmartStay applica-
tion, including their properties, methods, and validation logic. </file> <author>Enrique Rodrigues</author>
<date>03/12/2024</date>
```

```
<copyright file="OwnerTests.cs"> Copyright (c) 2024 Enrique Rodrigues. All Rights Reserved. </copyright>
<file> This file contains unit tests for the Owner class, which represents accommodation owners in the
SmartStay application, including their properties, methods, and validation logic. </file> <author>Enrique
Rodrigues</author> <date>03/12/2024</date>
```

```
<copyright file="PaymentTests.cs"> Copyright (c) 2024 All Rights Reserved. </copyright> <file> Contains unit
tests for the Payment class to verify its functionality and adherence to business rules. </file> <author>Enrique
Rodrigues</author> <date>03/12/2024</date>
```

```
<copyright file="ReservationTests.cs"> Copyright (c) 2024 Enrique Rodrigues. All Rights Reserved. </copyright>
<file> This file contains unit tests for the Reservation class, which stores reservation data in the SmartStay applica-
tion, including client IDs, accommodation types, check-in/check-out dates, payment statuses, and more. These tests
verify the correct functionality and validation of the class methods. </file> <author>Enrique Rodrigues</author>
<date>03/12/2024</date>
```

```
<copyright file="RoomTests.cs"> Copyright (c) 2024 Enrique Rodrigues. All Rights Reserved. </copyright>
<file> This file contains unit tests for the Room class, which represents an individual room within an accom-
modation. These tests verify the correct functionality of the class methods, including room initialization, avail-
ability checks, reservation management, price calculation, and string representation. </file> <author>Enrique
Rodrigues</author> <date>03/12/2024</date>
```

4.5 SmartStay.Core.Tests.Repositories Namespace Reference

The `SmartStay.Core.Tests.Repositories` namespace contains unit tests for the repository classes that interact with the application data.

Data Structures

- class `AccommodationsTests`
Contains unit tests for the Accommodations repository class. Tests include adding, removing, importing, exporting accommodations, and serialization/deserialization processes.
- class `ClientsTests`
Contains unit tests for the Clients repository class. Tests include adding, removing, importing, exporting clients, and serialization/deserialization processes.
- class `OwnersTests`
Contains unit tests for the Owners repository class. Tests include adding, removing, importing, exporting owners, and serialization/deserialization processes.
- class `ReservationsTests`
Contains unit tests for the Reservations repository class. Tests include adding, removing, importing, exporting reservations, and serialization/deserialization processes.

4.5.1 Detailed Description

The `SmartStay.Core.Tests.Repositories` namespace contains unit tests for the repository classes that interact with the application data.

```
<copyright file="AccommodationsTests.cs"> Copyright (c) 2024 All Rights Reserved. </copyright> <file> This file contains unit tests for the Accommodations class, which represents a collection of Accommodation objects managed within the SmartStay application. These tests verify the functionality of methods such as adding, removing, importing, exporting, and serializing accommodations. </file> <author>Enrique Rodrigues</author> <date>03/12/2024</date>
```

```
<copyright file="ClientsTests.cs"> Copyright (c) 2024 All Rights Reserved. </copyright> <file> This file contains unit tests for the Clients class, which manages a collection of Client objects. These tests verify the functionality of methods such as adding, removing, importing, exporting, and searching clients by their unique ID. </file> <author>Enrique Rodrigues</author> <date>03/12/2024</date>
```

```
<copyright file="OwnersTests.cs"> Copyright (c) 2024 All Rights Reserved. </copyright> <file> This file contains unit tests for the Owners class, which manages a collection of Owner objects. These tests verify the functionality of methods such as adding, removing, importing, exporting, and searching owners by their unique ID. </file> <author>Enrique Rodrigues</author> <date>03/12/2024</date>
```

```
<copyright file="ReservationsTests.cs"> Copyright (c) 2024 All Rights Reserved. </copyright> <file> This file contains unit tests for the Reservations class, which manages a collection of Reservation objects. These tests verify the functionality of methods such as adding, removing, importing, exporting, and searching reservations by their unique ID. </file> <author>Enrique Rodrigues</author> <date>03/12/2024</date>
```

4.6 SmartStay.Core.Tests.Services Namespace Reference

The `SmartStay.Core.Tests.Services` namespace contains unit tests for the services used in the `SmartStay` application. These tests verify the correct behavior of the `BookingManager` class and its methods.

Data Structures

- class [BookingManagerTests](#)

Contains unit tests for the BookingManager class. Tests the BookingManager.SaveAll method to ensure that it creates the necessary files when saving repositories.

4.6.1 Detailed Description

The [SmartStay.Core.Tests.Services](#) namespace contains unit tests for the services used in the [SmartStay](#) application. These tests verify the correct behavior of the BookingManager class and its methods.

```
<copyright file="BookingManagerTests.cs"> Copyright (c) 2024 Enrique Rodriguez. All Rights Reserved.
</copyright> <file> This file contains unit tests for the BookingManager class, specifically testing the file creation
functionality for saving data to files in the SmartStay system. </file> <author>Enrique Rodriguez</author>
<date>04/12/2024</date>
```

4.7 SmartStay.IO Namespace Reference

Namespaces

- namespace [Tests](#)

4.8 SmartStay.IO.Tests Namespace Reference

Namespaces

- namespace [Extensions](#)

The [SmartStay.IO.Tests.Extensions](#) namespace contains unit tests for the extension methods provided for file-related operations in the [SmartStay](#) application.

- namespace [FileOperations](#)

The [SmartStay.IO.Tests.FileOperations](#) namespace contains unit tests for file operations used within the [SmartStay](#) application.

4.9 SmartStay.IO.Tests.Extensions Namespace Reference

The [SmartStay.IO.Tests.Extensions](#) namespace contains unit tests for the extension methods provided for file-related operations in the [SmartStay](#) application.

Data Structures

- class [FileExtensionsTests](#)

Contains unit tests for the FileExtensions class. Tests the behavior of file-related extension methods.

4.9.1 Detailed Description

The [SmartStay.IO.Tests.Extensions](#) namespace contains unit tests for the extension methods provided for file-related operations in the [SmartStay](#) application.

```
<copyright file="FileExtensionsTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file> This file
contains unit tests for the FileExtensions class, ensuring the correct behavior of file-related extension methods in
the SmartStay application. </file> <author>Enrique Rodrigues</author> <date>25/11/2024</date>
```

4.10 SmartStay.IO.Tests.FileOperations Namespace Reference

The [SmartStay.IO.Tests.FileOperations](#) namespace contains unit tests for file operations used within the [SmartStay](#) application.

Data Structures

- class [FileHandlerTests](#)
Contains unit tests for the FileHandler class.
- class [PathValidatorTests](#)
Contains unit tests for the PathValidator class.

4.10.1 Detailed Description

The [SmartStay.IO.Tests.FileOperations](#) namespace contains unit tests for file operations used within the [SmartStay](#) application.

```
<copyright file="FileHandlerTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file> This file con-
tains unit tests for the FileHandler class, ensuring proper functionality for reading and writing files, including handling
invalid paths. </file> <author>Enrique Rodrigues</author> <date>25/11/2024</date>
```

```
<copyright file="PathValidatorTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file> This file
contains unit tests for the PathValidator class, validating file paths and file extensions in the SmartStay application.
</file> <author>Enrique Rodrigues</author> <date>25/11/2024</date>
```

4.11 SmartStay.Validation Namespace Reference

Namespaces

- namespace [Tests](#)
The [SmartStay.Validation.Tests](#) namespace contains unit tests for classes within the [SmartStay.Validation](#) namespace, ensuring correctness and reliability of validation functionalities.

4.12 SmartStay.Validation.Tests Namespace Reference

The [SmartStay.Validation.Tests](#) namespace contains unit tests for classes within the [SmartStay.Validation](#) namespace, ensuring correctness and reliability of validation functionalities.

Namespaces

- namespace [Validators](#)

The `SmartStay.Validation.Tests.Validators` namespace contains unit tests for the validation logic of different fields, specifically focusing on the `AccommodationValidator` class for validating accommodation types and IDs.

Data Structures

- class [ValidationErrorMessageTests](#)

Contains unit tests for the `ValidationErrorMessage` class. Ensures that error messages are correctly retrieved based on the provided error codes and that localization functions as expected.

- class [ValidationExceptionTests](#)

Contains unit tests for the `ValidationException` class. Ensures that exceptions are created with the expected error codes and messages.

4.12.1 Detailed Description

The `SmartStay.Validation.Tests` namespace contains unit tests for classes within the `SmartStay.Validation` namespace, ensuring correctness and reliability of validation functionalities.

```
<copyright file="ValidationErrorMessageTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file>
This file contains the implementation of the ValidationErrorMessageTests class, which provides unit tests for the
ValidationErrorMessage class to ensure the correct retrieval of localized validation error messages based on error
codes and current culture. </file> <author>Enrique Rodrigues</author> <date>24/11/2024</date>
```

```
<copyright file="ValidationExceptionTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file> This
file contains the implementation of the ValidationExceptionTests class, which provides unit tests for the Validation←
Exception class to ensure that validation exceptions are properly instantiated with the correct error codes and
messages. </file> <author>Enrique Rodrigues</author> <date>24/11/2024</date>
```

4.13 SmartStay.Validation.Tests.Validators Namespace Reference

The `SmartStay.Validation.Tests.Validators` namespace contains unit tests for the validation logic of different fields, specifically focusing on the `AccommodationValidator` class for validating accommodation types and IDs.

Data Structures

- class [AccommodationValidatorTests](#)

Contains unit tests for the `AccommodationValidator` class. Tests the validation logic for accommodation types and IDs.

- class [AddressValidatorTests](#)

Contains unit tests for the `AddressValidator` class. Tests the validation logic for addresses used in the `SmartStay` application.

- class [ClientValidatorTests](#)

Contains unit tests for the `ClientValidator` class. Tests the validation logic for client-related data in the `SmartStay` application.

- class [DateValidatorTests](#)

Contains unit tests for the `DateValidator` class. Tests the validation logic for dates such as check-in and check-out dates.

- class [EmailValidatorTests](#)

Contains unit tests for the `EmailValidator` class. Tests the validation logic for email addresses used in the [SmartStay](#) application.

- class [NameValidatorTests](#)

Contains unit tests for the `NameValidator` class. Validates both general names and accommodation names, checking correct behavior when the names are valid or invalid.

- class [OwnerValidatorTests](#)

Contains unit tests for the `OwnerValidator` class. Tests the validation logic for owner-related data used in the [SmartStay](#) application.

- class [PaymentValidatorTests](#)

Contains unit tests for the `PaymentValidator` class. Tests the validation logic for payment-related data in the [SmartStay](#) application.

- class [PhoneNumberValidatorTests](#)

Contains unit tests for the `PhoneNumberValidator` class. Tests the validation logic for phone numbers in the [SmartStay](#) application.

- class [ReservationValidatorTests](#)

Contains unit tests for the `ReservationValidator` class. Tests the validation logic for reservation-related data in the [SmartStay](#) application.

- class [RoomValidatorTests](#)

Contains unit tests for the `RoomValidator` class. Tests the validation logic for room-related data used in the [SmartStay](#) application.

4.13.1 Detailed Description

The [SmartStay.Validation.Tests.Validators](#) namespace contains unit tests for the validation logic of different fields, specifically focusing on the `AccommodationValidator` class for validating accommodation types and IDs.

The [SmartStay.Validation.Tests.Validators](#) namespace contains unit tests for the validation logic of different fields, including reservation data in the [SmartStay](#) application.

The [SmartStay.Validation.Tests.Validators](#) namespace contains unit tests for the validation logic of different fields, including phone numbers in the [SmartStay](#) application.

The [SmartStay.Validation.Tests.Validators](#) namespace contains unit tests for the validation logic of different fields.

```
<copyright file="AccommodationValidatorTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright>
<file> This file contains unit tests for the AccommodationValidator class, ensuring the correct validation of
accommodation-related data in the SmartStay application. </file> <author>Enrique Rodrigues</author>
<date>24/11/2024</date>
```

```
<copyright file="AddressValidatorTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file> This file
contains unit tests for the AddressValidator class, ensuring the correct validation of address-related data in the
SmartStay application. </file> <author>Enrique Rodrigues</author> <date>24/11/2024</date>
```

```
<copyright file="ClientValidatorTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file> This file
contains unit tests for the ClientValidator class, ensuring the correct validation of client-related data in the SmartStay
application. </file> <author>Enrique Rodrigues</author> <date>24/11/2024</date>
```

```
<copyright file="DateValidatorTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file> This file con-
tains unit tests for the DateValidator class, ensuring the correct validation of dates in the SmartStay application, such
as check-in and check-out dates. </file> <author>Enrique Rodrigues</author> <date>24/11/2024</date>
```

<copyright file="EmailValidatorTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file> This file contains unit tests for the EmailValidator class, ensuring the correct validation of email addresses in the [SmartStay](#) application. </file> <author>Enrique Rodrigues</author> <date>24/11/2024</date>

<copyright file="NameValidatorTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file> This file contains unit tests for the NameValidator class, ensuring that name validation logic for both general names and accommodation names works correctly under various scenarios. </file> <author>Enrique Rodrigues</author> <date>24/11/2024</date>

<copyright file="OwnerValidatorTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file> This file contains unit tests for the OwnerValidator class, ensuring the correct validation of owner-related data in the [SmartStay](#) application. </file> <author>Enrique Rodrigues</author> <date>03/12/2024</date>

<copyright file="PaymentValidatorTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file> This file contains unit tests for the PaymentValidator class, ensuring the correct validation of various aspects of payments within the [SmartStay](#) application. </file> <author>Enrique Rodrigues</author> <date>24/11/2024</date>

<copyright file="PhoneNumberValidatorTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file> This file contains unit tests for the PhoneNumberValidator class, ensuring the correct validation of phone numbers within the [SmartStay](#) application. </file> <author>Enrique Rodrigues</author> <date>24/11/2024</date>

<copyright file="ReservationValidatorTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file> This file contains unit tests for the ReservationValidator class, ensuring the correct validation of reservation-related data within the [SmartStay](#) application. </file> <author>Enrique Rodrigues</author> <date>24/11/2024</date>

<copyright file="RoomValidatorTests.cs"> Copyright (c) 2024 All Rights Reserved </copyright> <file> This file contains unit tests for the RoomValidator class, ensuring correct validation of room-related data within the [SmartStay](#) application. </file> <author>Enrique Rodrigues</author> <date>03/12/2024</date>

Chapter 5

Data Structure Documentation

5.1 SmartStay.Core.Tests.Repositories.AccommodationsTests Class Reference

Contains unit tests for the Accommodations repository class. Tests include adding, removing, importing, exporting accommodations, and serialization/deserialization processes.

Public Member Functions

- void [Add_ValidAccommodation_AddsAccommodationSuccessfully](#) ()
Tests the Accommodations.Add(Accommodation) method to ensure that an accommodation is successfully added.
- void [Remove_ValidAccommodation_RemovesAccommodationSuccessfully](#) ()
Tests the Accommodations.Remove(Accommodation) method to ensure that an accommodation is successfully removed.
- void [Remove_NonExistingAccommodation_ReturnsFalse](#) ()
Tests the Accommodations.Remove(Accommodation) method to ensure that attempting to remove a non-existing accommodation returns false.
- void [Import_ValidData_ImportsAccommodations](#) ()
Tests the Accommodations.Import(string) method to ensure that accommodations are imported correctly, including all fields and room data.
- void [Export_ValidData_ExportsAccommodations](#) ()
Tests the Accommodations.Export method to ensure that accommodations can be exported correctly.
- void [FindAccommodationById_ExistingId_ReturnsAccommodation](#) ()
Tests the Accommodations.FindAccommodationById(int) method to ensure that it finds an accommodation by its ID.
- void [FindAccommodationById_NonExistingId_ReturnsNull](#) ()
Tests the Accommodations.FindAccommodationById(int) method to ensure that it returns null when an accommodation with the specified ID does not exist.
- void [Save_ValidData_SavesToFile](#) ()
Tests the Accommodations.Save(string) method to ensure that accommodations can be saved to a file.
- void [Load_ValidFile_LoadsAccommodations](#) ()
Tests the Accommodations.Load(string) method to ensure that accommodations can be loaded from a file.

5.1.1 Detailed Description

Contains unit tests for the Accommodations repository class. Tests include adding, removing, importing, exporting accommodations, and serialization/deserialization processes.

Definition at line 29 of file [AccommodationsTests.cs](#).

5.1.2 Member Function Documentation

5.1.2.1 Add_ValidAccommodation_AddsAccommodationSuccessfully()

```
void SmartStay.Core.Tests.Repositories.AccommodationsTests.Add_ValidAccommodation_AddsAccommodation↵  
Successfully ( ) [inline]
```

Tests the Accommodations.Add(Accommodation) method to ensure that an accommodation is successfully added.

Definition at line 36 of file [AccommodationsTests.cs](#).

5.1.2.2 Export_ValidData_ExportsAccommodations()

```
void SmartStay.Core.Tests.Repositories.AccommodationsTests.Export_ValidData_ExportsAccommodations↵  
( ) [inline]
```

Tests the Accommodations.Export method to ensure that accommodations can be exported correctly.

Definition at line 165 of file [AccommodationsTests.cs](#).

5.1.2.3 FindAccommodationById_ExistingId_ReturnsAccommodation()

```
void SmartStay.Core.Tests.Repositories.AccommodationsTests.FindAccommodationById_ExistingId_↵  
ReturnsAccommodation ( ) [inline]
```

Tests the Accommodations.FindAccommodationById(int) method to ensure that it finds an accommodation by its ID.

Definition at line 246 of file [AccommodationsTests.cs](#).

5.1.2.4 FindAccommodationById_NonExistingId_ReturnsNull()

```
void SmartStay.Core.Tests.Repositories.AccommodationsTests.FindAccommodationById_NonExisting↵  
Id_ReturnsNull ( ) [inline]
```

Tests the Accommodations.FindAccommodationById(int) method to ensure that it returns null when an accommo-
dation with the specified ID does not exist.

Definition at line 266 of file [AccommodationsTests.cs](#).

5.1.2.5 Import_ValidData_ImportsAccommodations()

```
void SmartStay.Core.Tests.Repositories.AccommodationsTests.Import_ValidData_ImportsAccommodations↵  
( ) [inline]
```

Tests the Accommodations.Import(string) method to ensure that accommodations are imported correctly, including
all fields and room data.

Definition at line 93 of file [AccommodationsTests.cs](#).

5.1.2.6 Load_ValidFile_LoadsAccommodations()

```
void SmartStay.Core.Tests.Repositories.AccommodationsTests.Load_ValidFile_LoadsAccommodations  
( ) [inline]
```

Tests the Accommodations.Load(string) method to ensure that accommodations can be loaded from a file.

Definition at line 300 of file [AccommodationsTests.cs](#).

5.1.2.7 Remove_NonExistingAccommodation_ReturnsFalse()

```
void SmartStay.Core.Tests.Repositories.AccommodationsTests.Remove_NonExistingAccommodation_↵  
ReturnsFalse ( ) [inline]
```

Tests the Accommodations.Remove(Accommodation) method to ensure that attempting to remove a non-existing accommodation returns false.

Definition at line 75 of file [AccommodationsTests.cs](#).

5.1.2.8 Remove_ValidAccommodation_RemovesAccommodationSuccessfully()

```
void SmartStay.Core.Tests.Repositories.AccommodationsTests.Remove_ValidAccommodation_Removes↵  
AccommodationSuccessfully ( ) [inline]
```

Tests the Accommodations.Remove(Accommodation) method to ensure that an accommodation is successfully removed.

Definition at line 55 of file [AccommodationsTests.cs](#).

5.1.2.9 Save_ValidData_SavesToFile()

```
void SmartStay.Core.Tests.Repositories.AccommodationsTests.Save_ValidData_SavesToFile ( )  
[inline]
```

Tests the Accommodations.Save(string) method to ensure that accommodations can be saved to a file.

Definition at line 282 of file [AccommodationsTests.cs](#).

The documentation for this class was generated from the following file:

- [AccommodationsTests.cs](#)

5.2 SmartStay.Core.Tests.Models.AccommodationTests Class Reference

Contains unit tests for the Accommodation class. Tests include validation, property assignments, room management, and string representation.

Public Member Functions

- void [Constructor_ValidParameters_InitializesAccommodation](#) ()
Tests the constructor of the Accommodation class to ensure that it properly initializes an accommodation with the provided owner ID, type, name, and address.
- void [Constructor_InvalidName_ThrowsValidationException](#) ()
Tests the constructor of the Accommodation class to ensure that it throws a ValidationException if an invalid name is provided.
- void [FindRoomById_RoomExists_ReturnsRoom](#) ()
Tests the Accommodation.FindRoomById(int) method to ensure it correctly returns the room with the specified ID.
- void [FindRoomById_RoomDoesNotExist_ReturnsNull](#) ()
Tests the Accommodation.FindRoomById(int) method to ensure it returns null when a room with the specified ID does not exist.
- void [Owner_ToString_ReturnsJson](#) ()
Tests the Owner.ToString method to ensure it returns a valid JSON string representation of the owner.
- void [OwnerId_SetAndGet_CorrectlySetsOwnerId](#) ()
Tests the Accommodation.OwnerId property to ensure it correctly sets and retrieves the owner ID.
- void [OwnerId_InvalidValue_ThrowsException](#) ()
Tests the Accommodation.OwnerId property to ensure it throws an exception when an invalid ID is set.

5.2.1 Detailed Description

Contains unit tests for the Accommodation class. Tests include validation, property assignments, room management, and string representation.

Definition at line 28 of file [AccommodationTests.cs](#).

5.2.2 Member Function Documentation

5.2.2.1 Constructor_InvalidName_ThrowsValidationException()

```
void SmartStay.Core.Tests.Models.AccommodationTests.Constructor_InvalidName_ThrowsValidation←
Exception ( ) [inline]
```

Tests the constructor of the Accommodation class to ensure that it throws a ValidationException if an invalid name is provided.

Definition at line 60 of file [AccommodationTests.cs](#).

5.2.2.2 Constructor_ValidParameters_InitializesAccommodation()

```
void SmartStay.Core.Tests.Models.AccommodationTests.Constructor_ValidParameters_Initializes←
Accommodation ( ) [inline]
```

Tests the constructor of the Accommodation class to ensure that it properly initializes an accommodation with the provided owner ID, type, name, and address.

Exceptions

<i>ValidationException</i>	Thrown if any validation fails.
----------------------------	---------------------------------

Definition at line 36 of file [AccommodationTests.cs](#).

5.2.2.3 FindRoomById_RoomDoesNotExist_ReturnsNull()

```
void SmartStay.Core.Tests.Models.AccommodationTests.FindRoomById_RoomDoesNotExist_ReturnsNull  
( ) [inline]
```

Tests the Accommodation.FindRoomById(int) method to ensure it returns null when a room with the specified ID does not exist.

Definition at line 99 of file [AccommodationTests.cs](#).

5.2.2.4 FindRoomById_RoomExists_ReturnsRoom()

```
void SmartStay.Core.Tests.Models.AccommodationTests.FindRoomById_RoomExists_ReturnsRoom ( )  
[inline]
```

Tests the Accommodation.FindRoomById(int) method to ensure it correctly returns the room with the specified ID.

Definition at line 79 of file [AccommodationTests.cs](#).

5.2.2.5 Owner_ToString_ReturnsJson()

```
void SmartStay.Core.Tests.Models.AccommodationTests.Owner_ToString_ReturnsJson ( ) [inline]
```

Tests the Owner.ToString method to ensure it returns a valid JSON string representation of the owner.

Definition at line 116 of file [AccommodationTests.cs](#).

5.2.2.6 OwnerId_InvalidValue_ThrowsException()

```
void SmartStay.Core.Tests.Models.AccommodationTests.OwnerId_InvalidValue_ThrowsException ( )  
[inline]
```

Tests the Accommodation.OwnerId property to ensure it throws an exception when an invalid ID is set.

Definition at line 154 of file [AccommodationTests.cs](#).

5.2.2.7 OwnerId_SetAndGet_CorrectlySetsOwnerId()

```
void SmartStay.Core.Tests.Models.AccommodationTests.OwnerId_SetAndGet_CorrectlySetsOwnerId ( )  
[inline]
```

Tests the Accommodation.OwnerId property to ensure it correctly sets and retrieves the owner ID.

Definition at line 135 of file [AccommodationTests.cs](#).

The documentation for this class was generated from the following file:

- [AccommodationTests.cs](#)

5.3 SmartStay.Validation.Tests.Validators.AccommodationValidatorTests Class Reference

Contains unit tests for the AccommodationValidator class. Tests the validation logic for accommodation types and IDs.

Public Member Functions

- void [ValidateAccommodationType_ValidAccommodationType_ReturnsAccommodationType](#) ()
Tests the AccommodationValidator.ValidateAccommodationType(AccommodationType) method to ensure that it returns the accommodation type when the type is valid.
- void [ValidateAccommodationType_InvalidAccommodationType_ThrowsValidationException](#) ()
Tests the AccommodationValidator.ValidateAccommodationType(AccommodationType) method to ensure that it throws a ValidationException when the accommodation type is invalid.
- void [IsValidAccommodationType_ValidAccommodationType_ReturnsTrue](#) ()
Tests the AccommodationValidator.IsValidAccommodationType(AccommodationType) method to ensure it returns true when the accommodation type is valid.
- void [IsValidAccommodationType_InvalidAccommodationType_ReturnsFalse](#) ()
Tests the AccommodationValidator.IsValidAccommodationType(AccommodationType) method to ensure it returns false when the accommodation type is invalid.
- void [ValidateAccommodationId_ValidAccommodationId_ReturnsAccommodationId](#) ()
Tests the AccommodationValidator.ValidateAccommodationId(int) method to ensure that it returns the accommodation ID when the ID is valid.
- void [ValidateAccommodationId_InvalidAccommodationId_ThrowsValidationException](#) ()
Tests the AccommodationValidator.ValidateAccommodationId(int) method to ensure that it throws a ValidationException when the accommodation ID is invalid (non-positive).
- void [IsValidAccommodationId_ValidAccommodationId_ReturnsTrue](#) ()
Tests the AccommodationValidator.IsValidAccommodationId(int) method to ensure it returns true when the accommodation ID is valid (positive).
- void [IsValidAccommodationId_InvalidAccommodationId_ReturnsFalse](#) ()
Tests the AccommodationValidator.IsValidAccommodationId(int) method to ensure it returns false when the accommodation ID is invalid (non-positive).

5.3.1 Detailed Description

Contains unit tests for the AccommodationValidator class. Tests the validation logic for accommodation types and IDs.

Definition at line 27 of file [AccommodationValidatorTests.cs](#).

5.3.2 Member Function Documentation

5.3.2.1 IsValidAccommodationId_InvalidAccommodationId_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.AccommodationValidatorTests.IsValidAccommodationId_InvalidAccommodationId_ReturnsFalse ( ) [inline]
```

Tests the AccommodationValidator.IsValidAccommodationId(int) method to ensure it returns false when the accommodation ID is invalid (non-positive).

Definition at line 151 of file [AccommodationValidatorTests.cs](#).

5.3.2.2 IsValidAccommodationId_ValidAccommodationId_ReturnsTrue()

```
void SmartStay.Validation.Tests.Validators.AccommodationValidatorTests.IsValidAccommodation↵  
Id_ValidAccommodationId_ReturnsTrue ( ) [inline]
```

Tests the AccommodationValidator.IsValidAccommodationId(int) method to ensure it returns true when the accommodation ID is valid (positive).

Definition at line 134 of file [AccommodationValidatorTests.cs](#).

5.3.2.3 IsValidAccommodationType_InvalidAccommodationType_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.AccommodationValidatorTests.IsValidAccommodation↵  
Type_InvalidAccommodationType_ReturnsFalse ( ) [inline]
```

Tests the AccommodationValidator.IsValidAccommodationType(AccommodationType) method to ensure it returns false when the accommodation type is invalid.

Definition at line 84 of file [AccommodationValidatorTests.cs](#).

5.3.2.4 IsValidAccommodationType_ValidAccommodationType_ReturnsTrue()

```
void SmartStay.Validation.Tests.Validators.AccommodationValidatorTests.IsValidAccommodation↵  
Type_ValidAccommodationType_ReturnsTrue ( ) [inline]
```

Tests the AccommodationValidator.IsValidAccommodationType(AccommodationType) method to ensure it returns true when the accommodation type is valid.

Definition at line 67 of file [AccommodationValidatorTests.cs](#).

5.3.2.5 ValidateAccommodationId_InvalidAccommodationId_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.AccommodationValidatorTests.ValidateAccommodation↵  
Id_InvalidAccommodationId_ThrowsValidationException ( ) [inline]
```

Tests the AccommodationValidator.ValidateAccommodationId(int) method to ensure that it throws a Validation↵
Exception when the accommodation ID is invalid (non-positive).

Definition at line 118 of file [AccommodationValidatorTests.cs](#).

5.3.2.6 ValidateAccommodationId_ValidAccommodationId_ReturnsAccommodationId()

```
void SmartStay.Validation.Tests.Validators.AccommodationValidatorTests.ValidateAccommodation↵  
Id_ValidAccommodationId_ReturnsAccommodationId ( ) [inline]
```

Tests the AccommodationValidator.ValidateAccommodationId(int) method to ensure that it returns the accommoda-
tion ID when the ID is valid.

Definition at line 101 of file [AccommodationValidatorTests.cs](#).

5.3.2.7 ValidateAccommodationType_InvalidAccommodationType_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.AccommodationValidatorTests.ValidateAccommodation←
Type_InvalidAccommodationType_ThrowsValidationException ( ) [inline]
```

Tests the AccommodationValidator.ValidateAccommodationType(AccommodationType) method to ensure that it throws a ValidationException when the accommodation type is invalid.

Definition at line 51 of file [AccommodationValidatorTests.cs](#).

5.3.2.8 ValidateAccommodationType_ValidAccommodationType_ReturnsAccommodationType()

```
void SmartStay.Validation.Tests.Validators.AccommodationValidatorTests.ValidateAccommodation←
Type_ValidAccommodationType_ReturnsAccommodationType ( ) [inline]
```

Tests the AccommodationValidator.ValidateAccommodationType(AccommodationType) method to ensure that it returns the accommodation type when the type is valid.

Definition at line 34 of file [AccommodationValidatorTests.cs](#).

The documentation for this class was generated from the following file:

- [AccommodationValidatorTests.cs](#)

5.4 SmartStay.Validation.Tests.Validators.AddressValidatorTests Class Reference

Contains unit tests for the AddressValidator class. Tests the validation logic for addresses used in the [SmartStay](#) application.

Public Member Functions

- void [ValidateAddress_ValidAddress_ReturnsAddress](#) ()
Tests the AddressValidator.ValidateAddress(string) method to ensure that it returns the address when the address is valid.
- void [ValidateAddress_InvalidAddress_ThrowsValidationException](#) ()
Tests the AddressValidator.ValidateAddress(string) method to ensure that it throws a ValidationException when the address is invalid.
- void [IsValidAddress_ValidAddress_ReturnsTrue](#) ()
Tests the AddressValidator.IsValidAddress(string) method to ensure it returns true when the address is valid.
- void [IsValidAddress_InvalidAddress_ReturnsFalse](#) ()
Tests the AddressValidator.IsValidAddress(string) method to ensure it returns false when the address is invalid.

5.4.1 Detailed Description

Contains unit tests for the AddressValidator class. Tests the validation logic for addresses used in the [SmartStay](#) application.

Definition at line 25 of file [AddressValidatorTests.cs](#).

5.4.2 Member Function Documentation

5.4.2.1 IsValidAddress_InvalidAddress_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.AddressValidatorTests.IsValidAddress_InvalidAddress_ReturnsFalse ( ) [inline]
```

Tests the AddressValidator.IsValidAddress(string) method to ensure it returns false when the address is invalid.

Definition at line 81 of file [AddressValidatorTests.cs](#).

5.4.2.2 IsValidAddress_ValidAddress_ReturnsTrue()

```
void SmartStay.Validation.Tests.Validators.AddressValidatorTests.IsValidAddress_ValidAddress_ReturnsTrue ( ) [inline]
```

Tests the AddressValidator.IsValidAddress(string) method to ensure it returns true when the address is valid.

Definition at line 64 of file [AddressValidatorTests.cs](#).

5.4.2.3 ValidateAddress_InvalidAddress_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.AddressValidatorTests.ValidateAddress_InvalidAddress_ThrowsValidationException ( ) [inline]
```

Tests the AddressValidator.ValidateAddress(string) method to ensure that it throws a ValidationException when the address is invalid.

Definition at line 49 of file [AddressValidatorTests.cs](#).

5.4.2.4 ValidateAddress_ValidAddress_ReturnsAddress()

```
void SmartStay.Validation.Tests.Validators.AddressValidatorTests.ValidateAddress_ValidAddress_ReturnsAddress ( ) [inline]
```

Tests the AddressValidator.ValidateAddress(string) method to ensure that it returns the address when the address is valid.

Definition at line 32 of file [AddressValidatorTests.cs](#).

The documentation for this class was generated from the following file:

- [AddressValidatorTests.cs](#)

5.5 SmartStay.Core.Tests.Services.BookingManagerTests Class Reference

Contains unit tests for the BookingManager class. Tests the BookingManager.SaveAll method to ensure that it creates the necessary files when saving repositories.

Public Member Functions

- void [SaveAll_CreatesFiles_WhenCalled \(\)](#)
Tests the BookingManager.SaveAll method to ensure that it creates the necessary files in the specified directory when saving repositories.
- void [CreateBasicClient_CreatesClient_WhenValidInput \(\)](#)
Tests the BookingManager.CreateBasicClient method to ensure it creates a client with valid input data.
- void [CreateBasicClient_ThrowsClientCreationException_WhenValidationFails \(\)](#)
Tests the BookingManager.CreateBasicClient method to ensure it throws a ClientCreationException when invalid input data is provided.
- void [CreateCompleteClient_CreatesClient_WhenValidInput \(\)](#)
Tests the BookingManager.CreateCompleteClient method to ensure it creates a client with all information (first name, last name, email, phone number, and address) when valid input data is provided.
- void [CreateCompleteClient_ThrowsClientCreationException_WhenValidationFails \(\)](#)
Tests the BookingManager.CreateCompleteClient method to ensure it throws a ClientCreationException when invalid input data is provided.
- void [FindClientById_ReturnsClient_WhenClientExists \(\)](#)
Tests the BookingManager.FindClientById method to ensure it correctly finds a client by their ID.
- void [FindClientById_ThrowsArgumentException_WhenClientNotFound \(\)](#)
Tests the BookingManager.FindClientById method to ensure it throws an exception when no client with the specified ID is found.
- void [UpdateClient_UpdatesClient_WhenValidData \(\)](#)
Tests the BookingManager.UpdateClient method to ensure that it updates the client details correctly when valid data is provided.
- void [UpdateClient_ReturnsClientNotFound_WhenClientDoesNotExist \(\)](#)
Tests the BookingManager.UpdateClient method to ensure that it returns UpdateClientResult.ClientNotFound when the client does not exist.
- void [UpdateClient_ReturnsInvalidFirstName_WhenFirstNamesInvalid \(\)](#)
Tests the BookingManager.UpdateClient method to ensure it returns the correct validation error when the first name is invalid.
- void [UpdateClient_ReturnsInvalidLastName_WhenLastNamesInvalid \(\)](#)
Tests the BookingManager.UpdateClient method to ensure it returns the correct validation error when the last name is invalid.
- void [UpdateClient_ReturnsInvalidEmail_WhenEmailsInvalid \(\)](#)
Tests the BookingManager.UpdateClient method to ensure it returns the correct validation error when the email is invalid.
- void [RemoveClient_RemovesClient_WhenClientExists \(\)](#)
Tests the BookingManager.RemoveClient method to ensure that it removes a client from the system when the client exists.
- void [RemoveClient_ReturnsFalse_WhenClientDoesNotExist \(\)](#)
Tests the BookingManager.RemoveClient method to ensure that it returns false when attempting to remove a client that does not exist.
- void [CreateBasicOwner_CreatesOwner_WhenValidDataIsProvided \(\)](#)
Tests the BookingManager.CreateBasicOwner method to ensure that it creates a new owner with basic information when valid data is provided.
- void [CreateBasicOwner_ThrowsException_WhenEmailsInvalid \(\)](#)
Tests the BookingManager.CreateBasicOwner method to ensure that it throws an exception when invalid email is provided.
- void [CreateCompleteOwner_CreatesOwner_WhenValidDataIsProvided \(\)](#)
Tests the BookingManager.CreateCompleteOwner method to ensure that it creates a new owner with full details when valid data is provided.
- void [CreateCompleteOwner_ThrowsException_WhenPhoneNumberIsInvalid \(\)](#)
Tests the BookingManager.CreateCompleteOwner method to ensure that it throws an exception when invalid phone number is provided.

- void [FindOwnerByld_FindsOwner_WhenOwnerExists \(\)](#)
Tests the BookingManager.FindOwnerByld method to ensure that it finds an owner when the ID is valid.
- void [FindOwnerByld_ThrowsException_WhenOwnerDoesNotExist \(\)](#)
Tests the BookingManager.FindOwnerByld method to ensure that it throws an exception when the owner does not exist.
- void [UpdateOwner_UpdatesOwner_WhenValidDataIsProvided \(\)](#)
Tests the BookingManager.UpdateOwner method to ensure that it updates an owner when all input data is valid.
- void [UpdateOwner_ReturnsOwnerNotFound_WhenOwnerDoesNotExist \(\)](#)
Tests the BookingManager.UpdateOwner method to ensure that it returns OwnerNotFound when trying to update a non-existent owner.
- void [UpdateOwner_ReturnsInvalidFirstName_WhenFirstNamesInvalid \(\)](#)
Tests the BookingManager.UpdateOwner method to ensure that it returns InvalidFirstName when an invalid first name is provided.
- void [UpdateOwner_ReturnsInvalidEmail_WhenEmailsInvalid \(\)](#)
Tests the BookingManager.UpdateOwner method to ensure that it returns InvalidEmail when an invalid email is provided.
- void [RemoveOwner_RemovesOwner_WhenOwnerExists \(\)](#)
Tests the BookingManager.RemoveOwner method to ensure it removes an owner when the ID is valid.
- void [RemoveOwner_ReturnsFalse_WhenOwnerDoesNotExist \(\)](#)
Tests the BookingManager.RemoveOwner method to ensure it returns false when trying to remove an owner that does not exist.
- void [CreateAccommodation_CreatesAccommodation_WhenOwnerExists \(\)](#)
Tests the BookingManager.CreateAccommodation method to ensure it successfully creates an accommodation when the owner exists and all conditions are met.
- void [CreateAccommodation_ThrowsEntityNotFoundException_WhenOwnerDoesNotExist \(\)](#)
Tests the BookingManager.CreateAccommodation method to ensure it throws an exception when attempting to create accommodation for a non-existent owner.
- void [UpdateAccommodation_UpdatesAccommodation_WhenValidDataProvided \(\)](#)
Tests the BookingManager.UpdateAccommodation method to ensure that it successfully updates accommodation details when valid information is provided.
- void [UpdateAccommodation_ReturnsAccommodationNotFound_WhenAccommodationDoesNotExist \(\)](#)
Tests the BookingManager.UpdateAccommodation method to ensure it returns AccommodationNotFound when attempting to update an accommodation that does not exist.
- void [UpdateAccommodation_ReturnsInvalidName_WhenInvalidNameProvided \(\)](#)
Tests the BookingManager.UpdateAccommodation method to ensure it returns InvalidName when an invalid accommodation name is provided.
- void [UpdateAccommodation_ReturnsInvalidAddress_WhenInvalidAddressProvided \(\)](#)
Tests the BookingManager.UpdateAccommodation method to ensure it returns InvalidAddress when an invalid accommodation address is provided.
- void [RemoveAccommodation_SuccessfullyRemovesAccommodation_WhenAccommodationAndOwnerExist \(\)](#)
Tests the BookingManager.RemoveAccommodation method to ensure that it successfully removes an accommodation and disassociates it from the owner when valid data is provided.
- void [RemoveAccommodation_ReturnsAccommodationNotFound_WhenAccommodationDoesNotExist \(\)](#)
Tests the BookingManager.RemoveAccommodation method to ensure it returns AccommodationNotFound when the accommodation does not exist.
- void [RemoveAccommodation_ReturnsAccommodationNotFound_WhenAccommodationCannotBeFound \(\)](#)
Tests the BookingManager.RemoveAccommodation method to ensure it returns AccommodationNotFound when the accommodation cannot be found in the system.

5.5.1 Detailed Description

Contains unit tests for the BookingManager class. Tests the BookingManager.SaveAll method to ensure that it creates the necessary files when saving repositories.

Definition at line 31 of file [BookingManagerTests.cs](#).

5.5.2 Member Function Documentation

5.5.2.1 CreateAccommodation_CreatesAccommodation_WhenOwnerExists()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.CreateAccommodation_CreatesAccommodation_WhenOwnerExists ( ) [inline]
```

Tests the BookingManager.CreateAccommodation method to ensure it successfully creates an accommodation when the owner exists and all conditions are met.

Definition at line 779 of file [BookingManagerTests.cs](#).

5.5.2.2 CreateAccommodation_ThrowsEntityNotFoundException_WhenOwnerDoesNotExist()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.CreateAccommodation_ThrowsEntityNotFoundException_WhenOwnerDoesNotExist ( ) [inline]
```

Tests the BookingManager.CreateAccommodation method to ensure it throws an exception when attempting to create accommodation for a non-existent owner.

Definition at line 816 of file [BookingManagerTests.cs](#).

5.5.2.3 CreateBasicClient_CreatesClient_WhenValidInput()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.CreateBasicClient_CreatesClient_WhenValidInput ( ) [inline]
```

Tests the BookingManager.CreateBasicClient method to ensure it creates a client with valid input data.

Definition at line 70 of file [BookingManagerTests.cs](#).

5.5.2.4 CreateBasicClient_ThrowsClientCreationException_WhenValidationFails()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.CreateBasicClient_ThrowsClientCreationException_WhenValidationFails ( ) [inline]
```

Tests the BookingManager.CreateBasicClient method to ensure it throws a ClientCreationException when invalid input data is provided.

Definition at line 99 of file [BookingManagerTests.cs](#).

5.5.2.5 CreateBasicOwner_CreatesOwner_WhenValidDataIsProvided()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.CreateBasicOwner_CreatesOwner_WhenValidDataIsProvided ( ) [inline]
```

Tests the BookingManager.CreateBasicOwner method to ensure that it creates a new owner with basic information when valid data is provided.

Definition at line 442 of file [BookingManagerTests.cs](#).

5.5.2.6 CreateBasicOwner_ThrowsException_WhenEmailsInvalid()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.CreateBasicOwner_ThrowsException_WhenEmailsInvalid ( ) [inline]
```

Tests the BookingManager.CreateBasicOwner method to ensure that it throws an exception when invalid email is provided.

Definition at line 471 of file [BookingManagerTests.cs](#).

5.5.2.7 CreateCompleteClient_CreatesClient_WhenValidInput()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.CreateCompleteClient_CreatesClient_WhenValidInput ( ) [inline]
```

Tests the BookingManager.CreateCompleteClient method to ensure it creates a client with all information (first name, last name, email, phone number, and address) when valid input data is provided.

Definition at line 125 of file [BookingManagerTests.cs](#).

5.5.2.8 CreateCompleteClient_ThrowsClientCreationException_WhenValidationFails()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.CreateCompleteClient_ThrowsClientCreationException_WhenValidationFails ( ) [inline]
```

Tests the BookingManager.CreateCompleteClient method to ensure it throws a ClientCreationException when invalid input data is provided.

Definition at line 158 of file [BookingManagerTests.cs](#).

5.5.2.9 CreateCompleteOwner_CreatesOwner_WhenValidDataIsProvided()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.CreateCompleteOwner_CreatesOwner_WhenValidDataIsProvided ( ) [inline]
```

Tests the BookingManager.CreateCompleteOwner method to ensure that it creates a new owner with full details when valid data is provided.

Definition at line 496 of file [BookingManagerTests.cs](#).

5.5.2.10 CreateCompleteOwner_ThrowsException_WhenPhoneNumberIsInvalid()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.CreateCompleteOwner_ThrowsException_WhenPhoneNumberIsInvalid ( ) [inline]
```

Tests the BookingManager.CreateCompleteOwner method to ensure that it throws an exception when invalid phone number is provided.

Definition at line 529 of file [BookingManagerTests.cs](#).

5.5.2.11 FindClientById_ReturnsClient_WhenClientExists()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.FindClientById_ReturnsClient_When↵  
ClientExists ( ) [inline]
```

Tests the BookingManager.FindClientById method to ensure it correctly finds a client by their ID.

Definition at line 184 of file [BookingManagerTests.cs](#).

5.5.2.12 FindClientById_ThrowsArgumentException_WhenClientNotFound()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.FindClientById_ThrowsArgumentException↵  
_WhenClientNotFound ( ) [inline]
```

Tests the BookingManager.FindClientById method to ensure it throws an exception when no client with the specified ID is found.

Definition at line 216 of file [BookingManagerTests.cs](#).

5.5.2.13 FindOwnerById_FindsOwner_WhenOwnerExists()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.FindOwnerById_FindsOwner_WhenOwner↵  
Exists ( ) [inline]
```

Tests the BookingManager.FindOwnerById method to ensure that it finds an owner when the ID is valid.

Definition at line 556 of file [BookingManagerTests.cs](#).

5.5.2.14 FindOwnerById_ThrowsException_WhenOwnerDoesNotExist()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.FindOwnerById_ThrowsException_When↵  
OwnerDoesNotExist ( ) [inline]
```

Tests the BookingManager.FindOwnerById method to ensure that it throws an exception when the owner does not exist.

Definition at line 588 of file [BookingManagerTests.cs](#).

5.5.2.15 RemoveAccommodation_ReturnsAccommodationNotFound_WhenAccommodationCannotBeFound()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.RemoveAccommodation_ReturnsAccommodation↵  
NotFound_WhenAccommodationCannotBeFound ( ) [inline]
```

Tests the BookingManager.RemoveAccommodation method to ensure it returns AccommodationNotFound when the accommodation cannot be found in the system.

Definition at line 1041 of file [BookingManagerTests.cs](#).

5.5.2.16 RemoveAccommodation_ReturnsAccommodationNotFound_WhenAccommodationDoesNotExist()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.RemoveAccommodation_ReturnsAccommodation↵
NotFound_WhenAccommodationDoesNotExist ( ) [inline]
```

Tests the BookingManager.RemoveAccommodation method to ensure it returns AccommodationNotFound when the accommodation does not exist.

Definition at line 1016 of file [BookingManagerTests.cs](#).

5.5.2.17 RemoveAccommodation_SuccessfullyRemovesAccommodation_WhenAccommodationAndOwnerExist()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.RemoveAccommodation_Successfully↵
RemovesAccommodation_WhenAccommodationAndOwnerExist ( ) [inline]
```

Tests the BookingManager.RemoveAccommodation method to ensure that it successfully removes an accommodation and disassociates it from the owner when valid data is provided.

Definition at line 975 of file [BookingManagerTests.cs](#).

5.5.2.18 RemoveClient_RemovesClient_WhenClientExists()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.RemoveClient_RemovesClient_WhenClient↵
Exists ( ) [inline]
```

Tests the BookingManager.RemoveClient method to ensure that it removes a client from the system when the client exists.

Definition at line 388 of file [BookingManagerTests.cs](#).

5.5.2.19 RemoveClient_ReturnsFalse_WhenClientDoesNotExist()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.RemoveClient_ReturnsFalse_WhenClient↵
DoesNotExist ( ) [inline]
```

Tests the BookingManager.RemoveClient method to ensure that it returns false when attempting to remove a client that does not exist.

Definition at line 418 of file [BookingManagerTests.cs](#).

5.5.2.20 RemoveOwner_RemovesOwner_WhenOwnerExists()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.RemoveOwner_RemovesOwner_WhenOwner↵
Exists ( ) [inline]
```

Tests the BookingManager.RemoveOwner method to ensure it removes an owner when the ID is valid.

Definition at line 725 of file [BookingManagerTests.cs](#).

5.5.2.21 RemoveOwner_ReturnsFalse_WhenOwnerDoesNotExist()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.RemoveOwner_ReturnsFalse_WhenOwner↵  
DoesNotExist ( ) [inline]
```

Tests the BookingManager.RemoveOwner method to ensure it returns false when trying to remove an owner that does not exist.

Definition at line 755 of file [BookingManagerTests.cs](#).

5.5.2.22 SaveAll_CreatesFiles_WhenCalled()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.SaveAll_CreatesFiles_WhenCalled ( )  
[inline]
```

Tests the BookingManager.SaveAll method to ensure that it creates the necessary files in the specified directory when saving repositories.

Definition at line 38 of file [BookingManagerTests.cs](#).

5.5.2.23 UpdateAccommodation_ReturnsAccommodationNotFound_WhenAccommodationDoesNotExist()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.UpdateAccommodation_ReturnsAccommodation↵  
NotFound_WhenAccommodationDoesNotExist ( ) [inline]
```

Tests the BookingManager.UpdateAccommodation method to ensure it returns AccommodationNotFound when attempting to update an accommodation that does not exist.

Definition at line 886 of file [BookingManagerTests.cs](#).

5.5.2.24 UpdateAccommodation_ReturnsInvalidAddress_WhenInvalidAddressProvided()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.UpdateAccommodation_ReturnsInvalid↵  
Address_WhenInvalidAddressProvided ( ) [inline]
```

Tests the BookingManager.UpdateAccommodation method to ensure it returns InvalidAddress when an invalid accommodation address is provided.

Definition at line 942 of file [BookingManagerTests.cs](#).

5.5.2.25 UpdateAccommodation_ReturnsInvalidName_WhenInvalidNameProvided()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.UpdateAccommodation_ReturnsInvalid↵  
Name_WhenInvalidNameProvided ( ) [inline]
```

Tests the BookingManager.UpdateAccommodation method to ensure it returns InvalidName when an invalid accommodation name is provided.

Definition at line 910 of file [BookingManagerTests.cs](#).

5.5.2.26 UpdateAccommodation_UpdatesAccommodation_WhenValidDataProvided()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.UpdateAccommodation_UpdatesAccommodation_↵  
_WhenValidDataProvided ( ) [inline]
```

Tests the BookingManager.UpdateAccommodation method to ensure that it successfully updates accommodation details when valid information is provided.

Definition at line 844 of file [BookingManagerTests.cs](#).

5.5.2.27 UpdateClient_ReturnsClientNotFound_WhenClientDoesNotExist()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.UpdateClient_ReturnsClientNotFound_↵  
WhenClientDoesNotExist ( ) [inline]
```

Tests the BookingManager.UpdateClient method to ensure that it returns UpdateClientResult.ClientNotFound when the client does not exist.

Definition at line 277 of file [BookingManagerTests.cs](#).

5.5.2.28 UpdateClient_ReturnsInvalidEmail_WhenEmailIsInvalid()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.UpdateClient_ReturnsInvalidEmail_When↵  
EmailIsInvalid ( ) [inline]
```

Tests the BookingManager.UpdateClient method to ensure it returns the correct validation error when the email is invalid.

Definition at line 360 of file [BookingManagerTests.cs](#).

5.5.2.29 UpdateClient_ReturnsInvalidFirstName_WhenFirstNameIsInvalid()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.UpdateClient_ReturnsInvalidFirstName_↵  
WhenFirstNameIsInvalid ( ) [inline]
```

Tests the BookingManager.UpdateClient method to ensure it returns the correct validation error when the first name is invalid.

Definition at line 304 of file [BookingManagerTests.cs](#).

5.5.2.30 UpdateClient_ReturnsInvalidLastName_WhenLastNameIsInvalid()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.UpdateClient_ReturnsInvalidLastName_↵  
WhenLastNameIsInvalid ( ) [inline]
```

Tests the BookingManager.UpdateClient method to ensure it returns the correct validation error when the last name is invalid.

Definition at line 332 of file [BookingManagerTests.cs](#).

5.5.2.31 UpdateClient_UpdatesClient_WhenValidData()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.UpdateClient_UpdatesClient_WhenValidData ( ) [inline]
```

Tests the BookingManager.UpdateClient method to ensure that it updates the client details correctly when valid data is provided.

Definition at line 238 of file [BookingManagerTests.cs](#).

5.5.2.32 UpdateOwner_ReturnsInvalidEmail_WhenEmailsInvalid()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.UpdateOwner_ReturnsInvalidEmail_WhenEmailIsInvalid ( ) [inline]
```

Tests the BookingManager.UpdateOwner method to ensure that it returns InvalidEmail when an invalid email is provided.

Definition at line 698 of file [BookingManagerTests.cs](#).

5.5.2.33 UpdateOwner_ReturnsInvalidFirstName_WhenFirstNamesInvalid()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.UpdateOwner_ReturnsInvalidFirstName_WhenFirstNameIsInvalid ( ) [inline]
```

Tests the BookingManager.UpdateOwner method to ensure that it returns InvalidFirstName when an invalid first name is provided.

Definition at line 670 of file [BookingManagerTests.cs](#).

5.5.2.34 UpdateOwner_ReturnsOwnerNotFound_WhenOwnerDoesNotExist()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.UpdateOwner_ReturnsOwnerNotFound_WhenOwnerDoesNotExist ( ) [inline]
```

Tests the BookingManager.UpdateOwner method to ensure that it returns OwnerNotFound when trying to update a non-existent owner.

Definition at line 645 of file [BookingManagerTests.cs](#).

5.5.2.35 UpdateOwner_UpdatesOwner_WhenValidDataIsProvided()

```
void SmartStay.Core.Tests.Services.BookingManagerTests.UpdateOwner_UpdatesOwner_WhenValidDataIsProvided ( ) [inline]
```

Tests the BookingManager.UpdateOwner method to ensure that it updates an owner when all input data is valid.

Definition at line 610 of file [BookingManagerTests.cs](#).

The documentation for this class was generated from the following file:

- [BookingManagerTests.cs](#)

5.6 SmartStay.Core.Tests.Repositories.ClientsTests Class Reference

Contains unit tests for the Clients repository class. Tests include adding, removing, importing, exporting clients, and serialization/deserialization processes.

Public Member Functions

- void [Add_ValidClient_AddsClientSuccessfully](#) ()
Tests the Clients.Add(Client) method to ensure that a client is successfully added.
- void [Remove_ValidClient_RemovesClientSuccessfully](#) ()
Tests the Clients.Remove(Client) method to ensure that a client is successfully removed.
- void [Remove_NonExistingClient_ReturnsFalse](#) ()
Tests the Clients.Remove(Client) method to ensure that attempting to remove a non-existing client returns false.
- void [Import_ValidData_ImportsClients](#) ()
Tests the Clients.Import(string) method to ensure that clients are imported correctly, including all fields such as ID, name, email, phone number, address, and preferred payment method.
- void [Export_ValidData_ExportsClients](#) ()
Tests the Clients.Export method to ensure that clients can be exported correctly.
- void [FindClientById_ExistingId_ReturnsClient](#) ()
Tests the Clients.FindClientById(int) method to ensure that it finds a client by their ID.
- void [FindClientById_NonExistingId_ReturnsNull](#) ()
Tests the Clients.FindClientById(int) method to ensure that it returns null when a client with the specified ID does not exist.
- void [Save_ValidData_SavesToFile](#) ()
Tests the Clients.Save(string) method to ensure that the clients collection can be saved to a file.
- void [Load_ValidFile_LoadsClients](#) ()
Tests the Clients.Load(string) method to ensure that the clients collection can be loaded from a file.

5.6.1 Detailed Description

Contains unit tests for the Clients repository class. Tests include adding, removing, importing, exporting clients, and serialization/deserialization processes.

Definition at line 28 of file [ClientsTests.cs](#).

5.6.2 Member Function Documentation

5.6.2.1 Add_ValidClient_AddsClientSuccessfully()

```
void SmartStay.Core.Tests.Repositories.ClientsTests.Add_ValidClient_AddsClientSuccessfully ( )  
[inline]
```

Tests the Clients.Add(Client) method to ensure that a client is successfully added.

Definition at line 34 of file [ClientsTests.cs](#).

5.6.2.2 Export_ValidData_ExportsClients()

```
void SmartStay.Core.Tests.Repositories.ClientsTests.Export_ValidData_ExportsClients ( ) [inline]
```

Tests the Clients.Export method to ensure that clients can be exported correctly.

Definition at line 135 of file [ClientsTests.cs](#).

5.6.2.3 FindClientById_ExistingId_ReturnsClient()

```
void SmartStay.Core.Tests.Repositories.ClientsTests.FindClientById_ExistingId_ReturnsClient (
) [inline]
```

Tests the Clients.FindClientById(int) method to ensure that it finds a client by their ID.

Definition at line 186 of file [ClientsTests.cs](#).

5.6.2.4 FindClientById_NonExistingId_ReturnsNull()

```
void SmartStay.Core.Tests.Repositories.ClientsTests.FindClientById_NonExistingId_ReturnsNull (
) [inline]
```

Tests the Clients.FindClientById(int) method to ensure that it returns null when a client with the specified ID does not exist.

Definition at line 207 of file [ClientsTests.cs](#).

5.6.2.5 Import_ValidData_ImportsClients()

```
void SmartStay.Core.Tests.Repositories.ClientsTests.Import_ValidData_ImportsClients ( ) [inline]
```

Tests the Clients.Import(string) method to ensure that clients are imported correctly, including all fields such as ID, name, email, phone number, address, and preferred payment method.

Definition at line 90 of file [ClientsTests.cs](#).

5.6.2.6 Load_ValidFile_LoadsClients()

```
void SmartStay.Core.Tests.Repositories.ClientsTests.Load_ValidFile_LoadsClients ( ) [inline]
```

Tests the Clients.Load(string) method to ensure that the clients collection can be loaded from a file.

Definition at line 242 of file [ClientsTests.cs](#).

5.6.2.7 Remove_NonExistingClient_ReturnsFalse()

```
void SmartStay.Core.Tests.Repositories.ClientsTests.Remove_NonExistingClient_ReturnsFalse ( )
[inline]
```

Tests the Clients.Remove(Client) method to ensure that attempting to remove a non-existing client returns false.

Definition at line 72 of file [ClientsTests.cs](#).

5.6.2.8 Remove_ValidClient_RemovesClientSuccessfully()

```
void SmartStay.Core.Tests.Repositories.ClientsTests.Remove_ValidClient_RemovesClientSuccessfully  
( ) [inline]
```

Tests the Clients.Remove(Client) method to ensure that a client is successfully removed.

Definition at line 52 of file [ClientsTests.cs](#).

5.6.2.9 Save_ValidData_SavesToFile()

```
void SmartStay.Core.Tests.Repositories.ClientsTests.Save_ValidData_SavesToFile ( ) [inline]
```

Tests the Clients.Save(string) method to ensure that the clients collection can be saved to a file.

Definition at line 224 of file [ClientsTests.cs](#).

The documentation for this class was generated from the following file:

- [ClientsTests.cs](#)

5.7 SmartStay.Core.Tests.Models.ClientTests Class Reference

Contains unit tests for the Client class. Tests include validation, property assignments, and string representation.

Public Member Functions

- void [Constructor_ValidParameters_InitializesClient](#) ()
Tests the constructor of the Client class to ensure that it properly initializes a client with the provided first name, last name, and email.
- void [Constructor_InvalidName_ThrowsValidationException](#) ()
Tests the constructor of the Client class to ensure that it throws a ValidationException if an invalid name is provided.
- void [FirstName_SetAndGet_ValidatesValue](#) ()
Tests the Client.FirstName property to ensure that it correctly sets and retrieves the first name. Validates that invalid values throw a ValidationException.
- void [Email_SetAndGet_ValidatesValue](#) ()
Tests the Client.Email property to ensure that it correctly sets and retrieves the email. Validates that invalid email formats throw a ValidationException.
- void [Payment_ToString_ReturnsJson](#) ()
Tests the Client.ToString() method to ensure it returns a valid JSON string representation of the client.
- void [PreferredPaymentMethod_SetAndGet_CorrectlyAssignsValue](#) ()
Tests the Client.PreferredPaymentMethod property to ensure it correctly sets and retrieves the preferred payment method.
- void [Id_AutoGenerated_IsUniqueAndNonZero](#) ()
Tests that the Client.Id is a unique and non-zero value assigned during initialization.

5.7.1 Detailed Description

Contains unit tests for the Client class. Tests include validation, property assignments, and string representation.

Definition at line 26 of file [ClientTests.cs](#).

5.7.2 Member Function Documentation

5.7.2.1 Constructor_InvalidName_ThrowsValidationException()

```
void SmartStay.Core.Tests.Models.ClientTests.Constructor_InvalidName_ThrowsValidationException
( ) [inline]
```

Tests the constructor of the Client class to ensure that it throws a ValidationException if an invalid name is provided.

Definition at line 57 of file [ClientTests.cs](#).

5.7.2.2 Constructor_ValidParameters_InitializesClient()

```
void SmartStay.Core.Tests.Models.ClientTests.Constructor_ValidParameters_InitializesClient ( )
[inline]
```

Tests the constructor of the Client class to ensure that it properly initializes a client with the provided first name, last name, and email.

Exceptions

<i>ValidationException</i>	Thrown if any validation fails.
----------------------------	---------------------------------

Definition at line 34 of file [ClientTests.cs](#).

5.7.2.3 Email_SetAndGet_ValidatesValue()

```
void SmartStay.Core.Tests.Models.ClientTests.Email_SetAndGet_ValidatesValue ( ) [inline]
```

Tests the Client.Email property to ensure that it correctly sets and retrieves the email. Validates that invalid email formats throw a ValidationException.

Definition at line 94 of file [ClientTests.cs](#).

5.7.2.4 FirstName_SetAndGet_ValidatesValue()

```
void SmartStay.Core.Tests.Models.ClientTests.FirstName_SetAndGet_ValidatesValue ( ) [inline]
```

Tests the Client.FirstName property to ensure that it correctly sets and retrieves the first name. Validates that invalid values throw a ValidationException.

Definition at line 74 of file [ClientTests.cs](#).

5.7.2.5 Id_AutoGenerated_IsUniqueAndNonZero()

```
void SmartStay.Core.Tests.Models.ClientTests.Id_AutoGenerated_IsUniqueAndNonZero ( ) [inline]
```

Tests that the Client.Id is a unique and non-zero value assigned during initialization.

Definition at line 157 of file [ClientTests.cs](#).

5.7.2.6 Payment_ToString_ReturnsJson()

```
void SmartStay.Core.Tests.Models.ClientTests.Payment_ToString_ReturnsJson ( ) [inline]
```

Tests the Client.ToString() method to ensure it returns a valid JSON string representation of the client.

Definition at line 114 of file [ClientTests.cs](#).

5.7.2.7 PreferredPaymentMethod_SetAndGet_CorrectlyAssignsValue()

```
void SmartStay.Core.Tests.Models.ClientTests.PreferredPaymentMethod_SetAndGet_Correctly←  
AssignsValue ( ) [inline]
```

Tests the Client.PreferredPaymentMethod property to ensure it correctly sets and retrieves the preferred payment method.

Definition at line 141 of file [ClientTests.cs](#).

The documentation for this class was generated from the following file:

- [ClientTests.cs](#)

5.8 SmartStay.Validation.Tests.Validators.ClientValidatorTests Class Reference

Contains unit tests for the ClientValidator class. Tests the validation logic for client-related data in the [SmartStay](#) application.

Public Member Functions

- void [ValidateClientId_ValidId_ReturnsClientId](#) ()
Tests the ClientValidator.ValidateClientId(int) method to ensure that it returns the client ID when the ID is valid.
- void [ValidateClientId_InvalidId_ThrowsValidationException](#) ()
Tests the ClientValidator.ValidateClientId(int) method to ensure that it throws a ValidationException when the client ID is invalid.
- void [IsValidClientId_ValidId_ReturnsTrue](#) ()
Tests the ClientValidator.IsValidClientId(int) method to ensure it returns true when the client ID is valid.
- void [IsValidClientId_InvalidId_ReturnsFalse](#) ()
Tests the ClientValidator.IsValidClientId(int) method to ensure it returns false when the client ID is invalid (i.e., non-positive).

5.8.1 Detailed Description

Contains unit tests for the ClientValidator class. Tests the validation logic for client-related data in the [SmartStay](#) application.

Definition at line 25 of file [ClientValidatorTests.cs](#).

5.8.2 Member Function Documentation

5.8.2.1 IsValidClientId_InvalidId_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.ClientValidatorTests.IsValidClientId_InvalidId_ReturnsFalse ( ) [inline]
```

Tests the ClientValidator.IsValidClientId(int) method to ensure it returns false when the client ID is invalid (i.e., non-positive).

Definition at line 81 of file [ClientValidatorTests.cs](#).

5.8.2.2 IsValidClientId_ValidId_ReturnsTrue()

```
void SmartStay.Validation.Tests.Validators.ClientValidatorTests.IsValidClientId_ValidId_ReturnsTrue ( ) [inline]
```

Tests the ClientValidator.IsValidClientId(int) method to ensure it returns true when the client ID is valid.

Definition at line 64 of file [ClientValidatorTests.cs](#).

5.8.2.3 ValidateClientId_InvalidId_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.ClientValidatorTests.ValidateClientId_InvalidId_ThrowsValidationException ( ) [inline]
```

Tests the ClientValidator.ValidateClientId(int) method to ensure that it throws a ValidationException when the client ID is invalid.

Definition at line 49 of file [ClientValidatorTests.cs](#).

5.8.2.4 ValidateClientId_ValidId_ReturnsClientId()

```
void SmartStay.Validation.Tests.Validators.ClientValidatorTests.ValidateClientId_ValidId_ReturnsClientId ( ) [inline]
```

Tests the ClientValidator.ValidateClientId(int) method to ensure that it returns the client ID when the ID is valid.

Definition at line 32 of file [ClientValidatorTests.cs](#).

The documentation for this class was generated from the following file:

- [ClientValidatorTests.cs](#)

5.9 SmartStay.Validation.Tests.Validators.DateValidatorTests Class Reference

Contains unit tests for the DateValidator class. Tests the validation logic for dates such as check-in and check-out dates.

Public Member Functions

- void [ValidateCheckInDate_ValidDate_ReturnsCheckInDate](#) ()
Tests the DateValidator.ValidateCheckInDate(DateTime) method to ensure that it returns the check-in date when the date is valid (today or in the future).
- void [ValidateCheckInDate_InvalidDate_ThrowsValidationException](#) ()
Tests the DateValidator.ValidateCheckInDate(DateTime) method to ensure that it throws a ValidationException when the check-in date is in the past.
- void [ValidateCheckOutDate_ValidDateRange_ReturnsCheckOutDate](#) ()
Tests the DateValidator.ValidateCheckOutDate(DateTime, DateTime) method to ensure that it returns the check-out date when the date range is valid (check-out after check-in).
- void [ValidateCheckOutDate_InvalidDateRange_ThrowsValidationException](#) ()
Tests the DateValidator.ValidateCheckOutDate(DateTime, DateTime) method to ensure that it throws a ValidationException when the check-out date is before the check-in date.
- void [IsValidFutureDate_ValidDate_ReturnsTrue](#) ()
Tests the DateValidator.IsValidFutureDate(DateTime) method to ensure it returns true when the date is today or in the future.
- void [IsValidFutureDate_InvalidDate_ReturnsFalse](#) ()
Tests the DateValidator.IsValidFutureDate(DateTime) method to ensure it returns false when the date is in the past.
- void [IsValidDateRange_ValidDateRange_ReturnsTrue](#) ()
Tests the DateValidator.IsValidDateRange(DateTime, DateTime) method to ensure it returns true when the check-in date is earlier than the check-out date.
- void [IsValidDateRange_InvalidDateRange_ReturnsFalse](#) ()
Tests the DateValidator.IsValidDateRange(DateTime, DateTime) method to ensure it returns false when the check-in date is later than the check-out date.

5.9.1 Detailed Description

Contains unit tests for the DateValidator class. Tests the validation logic for dates such as check-in and check-out dates.

Definition at line 27 of file [DateValidatorTests.cs](#).

5.9.2 Member Function Documentation

5.9.2.1 IsValidDateRange_InvalidDateRange_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.DateValidatorTests.IsValidDateRange_InvalidDateRange_ReturnsFalse ( ) [inline]
```

Tests the DateValidator.IsValidDateRange(DateTime, DateTime) method to ensure it returns false when the check-in date is later than the check-out date.

Definition at line 153 of file [DateValidatorTests.cs](#).

5.9.2.2 IsValidDateRange_ValidDateRange_ReturnsTrue()

```
void SmartStay.Validation.Tests.Validators.DateValidatorTests.IsValidDateRange_ValidDate↵  
Range_ReturnsTrue ( ) [inline]
```

Tests the DateValidator.IsValidDateRange(DateTime, DateTime) method to ensure it returns true when the check-in date is earlier than the check-out date.

Definition at line 135 of file [DateValidatorTests.cs](#).

5.9.2.3 IsValidFutureDate_InvalidDate_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.DateValidatorTests.IsValidFutureDate_InvalidDate_↵  
ReturnsFalse ( ) [inline]
```

Tests the DateValidator.IsValidFutureDate(DateTime) method to ensure it returns false when the date is in the past.

Definition at line 118 of file [DateValidatorTests.cs](#).

5.9.2.4 IsValidFutureDate_ValidDate_ReturnsTrue()

```
void SmartStay.Validation.Tests.Validators.DateValidatorTests.IsValidFutureDate_ValidDate_↵  
ReturnsTrue ( ) [inline]
```

Tests the DateValidator.IsValidFutureDate(DateTime) method to ensure it returns true when the date is today or in the future.

Definition at line 101 of file [DateValidatorTests.cs](#).

5.9.2.5 ValidateCheckInDate_InvalidDate_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.DateValidatorTests.ValidateCheckInDate_Invalid↵  
Date_ThrowsValidationException ( ) [inline]
```

Tests the DateValidator.ValidateCheckInDate(DateTime) method to ensure that it throws a ValidationException when the check-in date is in the past.

Definition at line 51 of file [DateValidatorTests.cs](#).

5.9.2.6 ValidateCheckInDate_ValidDate_ReturnsCheckInDate()

```
void SmartStay.Validation.Tests.Validators.DateValidatorTests.ValidateCheckInDate_ValidDate_↵  
ReturnsCheckInDate ( ) [inline]
```

Tests the DateValidator.ValidateCheckInDate(DateTime) method to ensure that it returns the check-in date when the date is valid (today or in the future).

Definition at line 34 of file [DateValidatorTests.cs](#).

5.9.2.7 ValidateCheckOutDate_InvalidDateRange_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.DateValidatorTests.ValidateCheckOutDate_InvalidDateRange_ThrowsValidationException ( ) [inline]
```

Tests the DateValidator.ValidateCheckOutDate(DateTime, DateTime) method to ensure that it throws a ValidationException when the check-out date is before the check-in date.

Definition at line 84 of file [DateValidatorTests.cs](#).

5.9.2.8 ValidateCheckOutDate_ValidDateRange_ReturnsCheckOutDate()

```
void SmartStay.Validation.Tests.Validators.DateValidatorTests.ValidateCheckOutDate_ValidDateRange_ReturnsCheckOutDate ( ) [inline]
```

Tests the DateValidator.ValidateCheckOutDate(DateTime, DateTime) method to ensure that it returns the check-out date when the date range is valid (check-out after check-in).

Definition at line 66 of file [DateValidatorTests.cs](#).

The documentation for this class was generated from the following file:

- [DateValidatorTests.cs](#)

5.10 SmartStay.Validation.Tests.Validators.EmailValidatorTests Class Reference

Contains unit tests for the EmailValidator class. Tests the validation logic for email addresses used in the [SmartStay](#) application.

Public Member Functions

- void [ValidateEmail_ValidEmail_ReturnsEmail](#) ()
Tests the EmailValidator.ValidateEmail(string) method to ensure that it returns the email address when the address is valid.
- void [ValidateEmail_InvalidEmail_ThrowsValidationException](#) ()
Tests the EmailValidator.ValidateEmail(string) method to ensure that it throws a ValidationException when the email address is invalid.
- void [IsValidEmail_ValidEmail_ReturnsTrue](#) ()
Tests the EmailValidator.IsValidEmail(string) method to ensure that it returns true for a valid email address.
- void [IsValidEmail_MissingAtSymbol_ReturnsFalse](#) ()
Tests the EmailValidator.IsValidEmail(string) method to ensure that it returns false for an email address that is missing the '@' symbol.
- void [IsValidEmail_MissingDomainExtension_ReturnsFalse](#) ()
Tests the EmailValidator.IsValidEmail(string) method to ensure that it returns false for an email address that does not contain a domain extension.
- void [IsValidEmail_EmptyEmail_ReturnsFalse](#) ()
Tests the EmailValidator.IsValidEmail(string) method to ensure that it returns false for an empty email address.
- void [IsValidEmail_NullEmail_ReturnsFalse](#) ()
Tests the EmailValidator.IsValidEmail(string) method to ensure that it returns false for a null email address.
- void [IsValidEmail_InvalidCharacters_ReturnsFalse](#) ()
Tests the EmailValidator.IsValidEmail(string) method to ensure that it returns false for an email address with invalid characters.

5.10.1 Detailed Description

Contains unit tests for the EmailValidator class. Tests the validation logic for email addresses used in the [SmartStay](#) application.

Definition at line 25 of file [EmailValidatorTests.cs](#).

5.10.2 Member Function Documentation

5.10.2.1 IsValidEmail_EmptyEmail_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.EmailValidatorTests.IsValidEmail_EmptyEmail_↵  
ReturnsFalse ( ) [inline]
```

Tests the EmailValidator.IsValidEmail(string) method to ensure that it returns false for an empty email address.

Definition at line 115 of file [EmailValidatorTests.cs](#).

5.10.2.2 IsValidEmail_InvalidCharacters_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.EmailValidatorTests.IsValidEmail_InvalidCharacters↵  
_ReturnsFalse ( ) [inline]
```

Tests the EmailValidator.IsValidEmail(string) method to ensure that it returns false for an email address with invalid characters.

Definition at line 149 of file [EmailValidatorTests.cs](#).

5.10.2.3 IsValidEmail_MissingAtSymbol_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.EmailValidatorTests.IsValidEmail_MissingAtSymbol_↵  
ReturnsFalse ( ) [inline]
```

Tests the EmailValidator.IsValidEmail(string) method to ensure that it returns false for an email address that is missing the '@' symbol.

Definition at line 81 of file [EmailValidatorTests.cs](#).

5.10.2.4 IsValidEmail_MissingDomainExtension_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.EmailValidatorTests.IsValidEmail_MissingDomain↵  
Extension_ReturnsFalse ( ) [inline]
```

Tests the EmailValidator.IsValidEmail(string) method to ensure that it returns false for an email address that does not contain a domain extension.

Definition at line 98 of file [EmailValidatorTests.cs](#).

5.10.2.5 IsValidEmail_NullEmail_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.EmailValidatorTests.IsValidEmail_NullEmail_ReturnsFalse ( ) [inline]
```

Tests the EmailValidator.IsValidEmail(string) method to ensure that it returns false for a null email address.

Definition at line 132 of file [EmailValidatorTests.cs](#).

5.10.2.6 IsValidEmail_ValidEmail_ReturnsTrue()

```
void SmartStay.Validation.Tests.Validators.EmailValidatorTests.IsValidEmail_ValidEmail_ReturnsTrue ( ) [inline]
```

Tests the EmailValidator.IsValidEmail(string) method to ensure that it returns true for a valid email address.

Definition at line 64 of file [EmailValidatorTests.cs](#).

5.10.2.7 ValidateEmail_InvalidEmail_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.EmailValidatorTests.ValidateEmail_InvalidEmail_ThrowsValidationException ( ) [inline]
```

Tests the EmailValidator.ValidateEmail(string) method to ensure that it throws a ValidationException when the email address is invalid.

Definition at line 49 of file [EmailValidatorTests.cs](#).

5.10.2.8 ValidateEmail_ValidEmail_ReturnsEmail()

```
void SmartStay.Validation.Tests.Validators.EmailValidatorTests.ValidateEmail_ValidEmail_ReturnsEmail ( ) [inline]
```

Tests the EmailValidator.ValidateEmail(string) method to ensure that it returns the email address when the address is valid.

Definition at line 32 of file [EmailValidatorTests.cs](#).

The documentation for this class was generated from the following file:

- [EmailValidatorTests.cs](#)

5.11 SmartStay.IO.Tests.Extensions.FileExtensionsTests Class Reference

Contains unit tests for the FileExtensions class. Tests the behavior of file-related extension methods.

Public Member Functions

- void [EnsureDirectoryExists_DirectoryDoesNotExist_CreatesDirectory](#) ()
Tests the FileExtensions.EnsureDirectoryExists(string) method to ensure that it creates a new directory when the directory does not exist.
- void [EnsureDirectoryExists_DirectoryExists_DoesNotThrowException](#) ()
Tests the FileExtensions.EnsureDirectoryExists(string) method to ensure it does not throw an exception when the directory already exists.
- void [EnsureDirectoryExists_NullOrEmptyPath_DoesNotThrowException](#) (string filePath)
Tests the FileExtensions.EnsureDirectoryExists(string) method to ensure it does not throw an exception when given an empty file path.
- void [EnsureDirectoryExists_RootDirectoryPath_DoesNotThrowException](#) ()
Tests the FileExtensions.EnsureDirectoryExists(string) method to ensure it does not throw an exception when the file path points to the root directory.

5.11.1 Detailed Description

Contains unit tests for the FileExtensions class. Tests the behavior of file-related extension methods.

Definition at line 25 of file [FileExtensionsTests.cs](#).

5.11.2 Member Function Documentation

5.11.2.1 EnsureDirectoryExists_DirectoryDoesNotExist_CreatesDirectory()

```
void SmartStay.IO.Tests.Extensions.FileExtensionsTests.EnsureDirectoryExists_DirectoryDoesNotExist_CreatesDirectory ( ) [inline]
```

Tests the FileExtensions.EnsureDirectoryExists(string) method to ensure that it creates a new directory when the directory does not exist.

Definition at line 32 of file [FileExtensionsTests.cs](#).

5.11.2.2 EnsureDirectoryExists_DirectoryExists_DoesNotThrowException()

```
void SmartStay.IO.Tests.Extensions.FileExtensionsTests.EnsureDirectoryExists_DirectoryExists_DoesNotThrowException ( ) [inline]
```

Tests the FileExtensions.EnsureDirectoryExists(string) method to ensure it does not throw an exception when the directory already exists.

Definition at line 63 of file [FileExtensionsTests.cs](#).

5.11.2.3 EnsureDirectoryExists_NullOrEmptyPath_DoesNotThrowException()

```
void SmartStay.IO.Tests.Extensions.FileExtensionsTests.EnsureDirectoryExists_NullOrEmptyPath_DoesNotThrowException (
    string filePath ) [inline]
```

Tests the FileExtensions.EnsureDirectoryExists(string) method to ensure it does not throw an exception when given an empty file path.

Definition at line 94 of file [FileExtensionsTests.cs](#).

5.11.2.4 EnsureDirectoryExists_RootDirectoryPath_DoesNotThrowException()

```
void SmartStay.IO.Tests.Extensions.FileExtensionsTests.EnsureDirectoryExists_RootDirectory↵
Path_DoesNotThrowException ( ) [inline]
```

Tests the FileExtensions.EnsureDirectoryExists(string) method to ensure it does not throw an exception when the file path points to the root directory.

Definition at line 106 of file [FileExtensionsTests.cs](#).

The documentation for this class was generated from the following file:

- [FileExtensionsTests.cs](#)

5.12 SmartStay.IO.Tests.FileOperations.FileHandlerTests Class Reference

Contains unit tests for the FileHandler class.

Public Member Functions

- void [ReadFile_EmptyPath_ThrowsArgumentException](#) (string filePath)
Tests the FileHandler.ReadFile(string) method to ensure it throws an exception when the file path is empty.
- void [ReadFile_FileDoesNotExist_ThrowsFileNotFoundException](#) ()
Tests the FileHandler.ReadFile(string) method to ensure it throws a FileNotFoundException when the file does not exist.
- void [ReadFile_ValidFilePath_ReturnsFileContent](#) ()
Tests the FileHandler.ReadFile(string) method to ensure it reads the file content correctly.
- void [WriteFile_EmptyPath_ThrowsArgumentException](#) (string filePath)
Tests the FileHandler.WriteFile(string, string) method to ensure it throws an exception when the file path is empty.
- void [WriteFile_ValidFilePath_WritesFileContent](#) ()
Tests the FileHandler.WriteFile(string, string) method to ensure it writes content to the file.
- void [WriteFile_NonExistentDirectory_CreatesDirectoryAndWritesFile](#) ()
Tests the FileHandler.WriteFile(string, string) method to ensure it creates directories if they do not exist.

5.12.1 Detailed Description

Contains unit tests for the FileHandler class.

Definition at line 25 of file [FileHandlerTests.cs](#).

5.12.2 Member Function Documentation

5.12.2.1 ReadFile_EmptyPath_ThrowsArgumentException()

```
void SmartStay.IO.Tests.FileOperations.FileHandlerTests.ReadFile_EmptyPath_ThrowsArgument↵
Exception (
    string filePath ) [inline]
```

Tests the FileHandler.ReadFile(string) method to ensure it throws an exception when the file path is empty.

Definition at line 33 of file [FileHandlerTests.cs](#).

5.12.2.2 ReadFile_FileDoesNotExist_ThrowsFileNotFoundException()

```
void SmartStay.IO.Tests.FileOperations.FileHandlerTests.ReadFile_FileDoesNotExist_ThrowsFile↵  
NotFoundException ( ) [inline]
```

Tests the FileHandler.ReadFile(string) method to ensure it throws a FileNotFoundException when the file does not exist.

Definition at line 45 of file [FileHandlerTests.cs](#).

5.12.2.3 ReadFile_ValidFilePath_ReturnsFileContent()

```
void SmartStay.IO.Tests.FileOperations.FileHandlerTests.ReadFile_ValidFilePath_ReturnsFile↵  
Content ( ) [inline]
```

Tests the FileHandler.ReadFile(string) method to ensure it reads the file content correctly.

Definition at line 59 of file [FileHandlerTests.cs](#).

5.12.2.4 WriteFile_EmptyPath_ThrowsArgumentException()

```
void SmartStay.IO.Tests.FileOperations.FileHandlerTests.WriteFile_EmptyPath_ThrowsArgument↵  
Exception (   
    string filePath ) [inline]
```

Tests the FileHandler.WriteFile(string, string) method to ensure it throws an exception when the file path is empty.

Definition at line 87 of file [FileHandlerTests.cs](#).

5.12.2.5 WriteFile_NonExistentDirectory_CreatesDirectoryAndWritesFile()

```
void SmartStay.IO.Tests.FileOperations.FileHandlerTests.WriteFile_NonExistentDirectory_↵  
CreatesDirectoryAndWritesFile ( ) [inline]
```

Tests the FileHandler.WriteFile(string, string) method to ensure it creates directories if they do not exist.

Definition at line 125 of file [FileHandlerTests.cs](#).

5.12.2.6 WriteFile_ValidFilePath_WritesFileContent()

```
void SmartStay.IO.Tests.FileOperations.FileHandlerTests.WriteFile_ValidFilePath_WritesFile↵  
Content ( ) [inline]
```

Tests the FileHandler.WriteFile(string, string) method to ensure it writes content to the file.

Definition at line 98 of file [FileHandlerTests.cs](#).

The documentation for this class was generated from the following file:

- [FileHandlerTests.cs](#)

5.13 SmartStay.Validation.Tests.Validators.NameValidatorTests Class Reference

Contains unit tests for the NameValidator class. Validates both general names and accommodation names, checking correct behavior when the names are valid or invalid.

Public Member Functions

- void [ValidateName_ValidName_ReturnsName](#) ()
Tests the NameValidator.ValidateName(string) method to ensure that it returns the name as-is when the name is valid.
- void [ValidateName_InvalidName_ThrowsValidationException](#) ()
Tests the NameValidator.ValidateName(string) method to ensure that it throws a ValidationException when the name is invalid (empty).
- void [ValidateName_TooLongName_ThrowsValidationException](#) ()
Tests the NameValidator.ValidateName(string) method to ensure that it throws a ValidationException when the name is too long (greater than 50 characters).
- void [ValidateAccommodationName_ValidName_ReturnsName](#) ()
Tests the NameValidator.ValidateAccommodationName(string) method to ensure that it returns the accommodation name as-is when it is valid.
- void [ValidateAccommodationName_TooLongName_ThrowsValidationException](#) ()
Tests the NameValidator.ValidateAccommodationName(string) method to ensure that it throws a ValidationException when the accommodation name is too long (greater than 100 characters).
- void [IsValidName_ValidName_ReturnsTrue](#) ()
Tests the NameValidator.IsValidName(string) method to ensure it returns true when the name is valid.
- void [IsValidName_InvalidName_ReturnsFalse](#) ()
Tests the NameValidator.IsValidName(string) method to ensure it returns false when the name is invalid (empty).
- void [IsValidAccommodationName_ValidName_ReturnsTrue](#) ()
Tests the NameValidator.IsValidAccommodationName(string) method to ensure it returns true when the accommodation name is valid.
- void [IsValidAccommodationName_InvalidName_ReturnsFalse](#) ()
Tests the NameValidator.IsValidAccommodationName(string) method to ensure it returns false when the accommodation name is invalid (null).

5.13.1 Detailed Description

Contains unit tests for the NameValidator class. Validates both general names and accommodation names, checking correct behavior when the names are valid or invalid.

Definition at line 27 of file [NameValidatorTests.cs](#).

5.13.2 Member Function Documentation

5.13.2.1 IsValidAccommodationName_InvalidName_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.NameValidatorTests.IsValidAccommodationName_InvalidName_ReturnsFalse ( ) [inline]
```

Tests the NameValidator.IsValidAccommodationName(string) method to ensure it returns false when the accommodation name is invalid (null).

Definition at line 165 of file [NameValidatorTests.cs](#).

5.13.2.2 IsValidAccommodationName_ValidName_ReturnsTrue()

```
void SmartStay.Validation.Tests.Validators.NameValidatorTests.IsValidAccommodationName_ValidName_ReturnsTrue ( ) [inline]
```

Tests the NameValidator.IsValidAccommodationName(string) method to ensure it returns true when the accommodation name is valid.

Definition at line 148 of file [NameValidatorTests.cs](#).

5.13.2.3 IsValidName_InvalidName_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.NameValidatorTests.IsValidName_InvalidName_ReturnsFalse ( ) [inline]
```

Tests the NameValidator.IsValidName(string) method to ensure it returns false when the name is invalid (empty).

Definition at line 131 of file [NameValidatorTests.cs](#).

5.13.2.4 IsValidName_ValidName_ReturnsTrue()

```
void SmartStay.Validation.Tests.Validators.NameValidatorTests.IsValidName_ValidName_ReturnsTrue ( ) [inline]
```

Tests the NameValidator.IsValidName(string) method to ensure it returns true when the name is valid.

Definition at line 114 of file [NameValidatorTests.cs](#).

5.13.2.5 ValidateAccommodationName_TooLongName_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.NameValidatorTests.ValidateAccommodationName_TooLongName_ThrowsValidationException ( ) [inline]
```

Tests the NameValidator.ValidateAccommodationName(string) method to ensure that it throws a ValidationException when the accommodation name is too long (greater than 100 characters).

Definition at line 98 of file [NameValidatorTests.cs](#).

5.13.2.6 ValidateAccommodationName_ValidName_ReturnsName()

```
void SmartStay.Validation.Tests.Validators.NameValidatorTests.ValidateAccommodationName_ValidName_ReturnsName ( ) [inline]
```

Tests the NameValidator.ValidateAccommodationName(string) method to ensure that it returns the accommodation name as-is when it is valid.

Definition at line 81 of file [NameValidatorTests.cs](#).

5.13.2.7 ValidateName_InvalidName_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.NameValidatorTests.ValidateName_InvalidName_ThrowsValidationException ( ) [inline]
```

Tests the NameValidator.ValidateName(string) method to ensure that it throws a ValidationException when the name is invalid (empty).

Definition at line 51 of file [NameValidatorTests.cs](#).

5.13.2.8 ValidateName_TooLongName_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.NameValidatorTests.ValidateName_TooLongName_ThrowsValidationException ( ) [inline]
```

Tests the NameValidator.ValidateName(string) method to ensure that it throws a ValidationException when the name is too long (greater than 50 characters).

Definition at line 66 of file [NameValidatorTests.cs](#).

5.13.2.9 ValidateName_ValidName_ReturnsName()

```
void SmartStay.Validation.Tests.Validators.NameValidatorTests.ValidateName_ValidName_ReturnsName ( ) [inline]
```

Tests the NameValidator.ValidateName(string) method to ensure that it returns the name as-is when the name is valid.

Definition at line 34 of file [NameValidatorTests.cs](#).

The documentation for this class was generated from the following file:

- [NameValidatorTests.cs](#)

5.14 SmartStay.Core.Tests.Repositories.OwnersTests Class Reference

Contains unit tests for the Owners repository class. Tests include adding, removing, importing, exporting owners, and serialization/deserialization processes.

Public Member Functions

- void [Add_ValidOwner_AddsOwnerSuccessfully](#) ()
Tests the Owners.Add(Owner) method to ensure that an owner is successfully added.
- void [Remove_ValidOwner_RemovesOwnerSuccessfully](#) ()
Tests the Owners.Remove(Owner) method to ensure that an owner is successfully removed.
- void [Remove_NonExistingOwner_ReturnsFalse](#) ()
Tests the Owners.Remove(Owner) method to ensure that attempting to remove a non-existing owner returns false.
- void [Import_ValidData_ImportsOwners](#) ()
Tests the Owners.Import(string) method to ensure that owners are imported correctly, including all fields such as ID, first name, last name, email, phone number, address, and accommodations owned.
- void [Export_ValidData_ExportsOwnersWithAccommodations](#) ()
Tests the Owners.Export method to ensure that owners, accommodations, rooms, and reservation dates are exported correctly.
- void [FindOwnerById_ExistingId_ReturnsOwner](#) ()
Tests the Owners.FindOwnerById(int) method to ensure that it finds an owner by their ID.
- void [FindOwnerById_NonExistingId_ReturnsNull](#) ()
Tests the Owners.FindOwnerById(int) method to ensure that it returns null when an owner with the specified ID does not exist.
- void [Save_ValidData_SavesToFile](#) ()
Tests the Owners.Save(string) method to ensure that the owners collection can be saved to a file.
- void [Load_ValidFile_LoadsOwners](#) ()
Tests the Owners.Load(string) method to ensure that the owners collection can be loaded from a file.

5.14.1 Detailed Description

Contains unit tests for the Owners repository class. Tests include adding, removing, importing, exporting owners, and serialization/deserialization processes.

Definition at line 27 of file [OwnersTests.cs](#).

5.14.2 Member Function Documentation

5.14.2.1 Add_ValidOwner_AddsOwnerSuccessfully()

```
void SmartStay.Core.Tests.Repositories.OwnersTests.Add_ValidOwner_AddsOwnerSuccessfully ( )  
[inline]
```

Tests the Owners.Add(Owner) method to ensure that an owner is successfully added.

Definition at line 33 of file [OwnersTests.cs](#).

5.14.2.2 Export_ValidData_ExportsOwnersWithAccommodations()

```
void SmartStay.Core.Tests.Repositories.OwnersTests.Export_ValidData_ExportsOwnersWithAccommodations  
( ) [inline]
```

Tests the Owners.Export method to ensure that owners, accommodations, rooms, and reservation dates are exported correctly.

Definition at line 160 of file [OwnersTests.cs](#).

5.14.2.3 FindOwnerById_ExistingId_ReturnsOwner()

```
void SmartStay.Core.Tests.Repositories.OwnersTests.FindOwnerById_ExistingId_ReturnsOwner ( )  
[inline]
```

Tests the Owners.FindOwnerById(int) method to ensure that it finds an owner by their ID.

Definition at line 211 of file [OwnersTests.cs](#).

5.14.2.4 FindOwnerById_NonExistingId_ReturnsNull()

```
void SmartStay.Core.Tests.Repositories.OwnersTests.FindOwnerById_NonExistingId_ReturnsNull ( )  
[inline]
```

Tests the Owners.FindOwnerById(int) method to ensure that it returns null when an owner with the specified ID does not exist.

Definition at line 232 of file [OwnersTests.cs](#).

5.14.2.5 Import_ValidData_ImportsOwners()

```
void SmartStay.Core.Tests.Repositories.OwnersTests.Import_ValidData_ImportsOwners ( ) [inline]
```

Tests the Owners.Import(string) method to ensure that owners are imported correctly, including all fields such as ID, first name, last name, email, phone number, address, and accommodations owned.

Definition at line 89 of file [OwnersTests.cs](#).

5.14.2.6 Load_ValidFile_LoadsOwners()

```
void SmartStay.Core.Tests.Repositories.OwnersTests.Load_ValidFile_LoadsOwners ( ) [inline]
```

Tests the Owners.Load(string) method to ensure that the owners collection can be loaded from a file.

Definition at line 267 of file [OwnersTests.cs](#).

5.14.2.7 Remove_NonExistingOwner_ReturnsFalse()

```
void SmartStay.Core.Tests.Repositories.OwnersTests.Remove_NonExistingOwner_ReturnsFalse ( )  
[inline]
```

Tests the Owners.Remove(Owner) method to ensure that attempting to remove a non-existing owner returns false.

Definition at line 71 of file [OwnersTests.cs](#).

5.14.2.8 Remove_ValidOwner_RemovesOwnerSuccessfully()

```
void SmartStay.Core.Tests.Repositories.OwnersTests.Remove_ValidOwner_RemovesOwnerSuccessfully  
( ) [inline]
```

Tests the Owners.Remove(Owner) method to ensure that an owner is successfully removed.

Definition at line 51 of file [OwnersTests.cs](#).

5.14.2.9 Save_ValidData_SavesToFile()

```
void SmartStay.Core.Tests.Repositories.OwnersTests.Save_ValidData_SavesToFile ( ) [inline]
```

Tests the Owners.Save(string) method to ensure that the owners collection can be saved to a file.

Definition at line 249 of file [OwnersTests.cs](#).

The documentation for this class was generated from the following file:

- [OwnersTests.cs](#)

5.15 SmartStay.Core.Tests.Models.OwnerTests Class Reference

Contains unit tests for the Owner class. Tests include validation, property assignments, and methods for managing accommodations.

Public Member Functions

- void [Constructor_BasicDetails_InitializesOwner](#) ()
Tests the constructor of the Owner class with basic details to ensure proper initialization and validation.
- void [Constructor_AdditionalDetails_InitializesOwner](#) ()
Tests the constructor of the Owner class with additional details (phone number and address) to ensure proper initialization and validation.
- void [FirstName_SetAndGet_ValidatesValue](#) ()
Tests the Owner.FirstName property for proper value validation and assignment.
- void [Email_SetAndGet_ValidatesValue](#) ()
Tests the Owner.Email property for proper value validation and assignment.
- void [PhoneNumber_SetAndGet_ValidatesValue](#) ()
Tests the Owner.PhoneNumber property for proper value validation and assignment.
- void [Owner_ToString_ReturnsJson](#) ()
Tests the Owner.ToString() method to ensure it returns a valid JSON string representation of the owner.
- void [AddAccommodation_ValidAccommodation_AddsSuccessfully](#) ()
Tests the Owner.AddAccommodation method to ensure accommodations can be added successfully.
- void [RemoveAccommodation_ValidAccommodation_RemovesSuccessfully](#) ()
Tests the Owner.RemoveAccommodation method to ensure accommodations can be removed successfully.
- void [LastAssignedId_TracksCorrectly](#) ()
Tests the Owner.LastAssignedId property for proper tracking of the last assigned owner ID.

5.15.1 Detailed Description

Contains unit tests for the Owner class. Tests include validation, property assignments, and methods for managing accommodations.

Definition at line 27 of file [OwnerTests.cs](#).

5.15.2 Member Function Documentation

5.15.2.1 AddAccommodation_ValidAccommodation_AddsSuccessfully()

```
void SmartStay.Core.Tests.Models.OwnerTests.AddAccommodation_ValidAccommodation_AddsSuccessfully  
( ) [inline]
```

Tests the Owner.AddAccommodation method to ensure accommodations can be added successfully.

Definition at line 158 of file [OwnerTests.cs](#).

5.15.2.2 Constructor_AdditionalDetails_InitializesOwner()

```
void SmartStay.Core.Tests.Models.OwnerTests.Constructor_AdditionalDetails_InitializesOwner ( )  
[inline]
```

Tests the constructor of the Owner class with additional details (phone number and address) to ensure proper initialization and validation.

Definition at line 56 of file [OwnerTests.cs](#).

5.15.2.3 Constructor_BasicDetails_InitializesOwner()

```
void SmartStay.Core.Tests.Models.OwnerTests.Constructor_BasicDetails_InitializesOwner ( )  
[inline]
```

Tests the constructor of the Owner class with basic details to ensure proper initialization and validation.

Definition at line 34 of file [OwnerTests.cs](#).

5.15.2.4 Email_SetAndGet_ValidatesValue()

```
void SmartStay.Core.Tests.Models.OwnerTests.Email_SetAndGet_ValidatesValue ( ) [inline]
```

Tests the Owner.Email property for proper value validation and assignment.

Definition at line 99 of file [OwnerTests.cs](#).

5.15.2.5 FirstName_SetAndGet_ValidatesValue()

```
void SmartStay.Core.Tests.Models.OwnerTests.FirstName_SetAndGet_ValidatesValue ( ) [inline]
```

Tests the Owner.FirstName property for proper value validation and assignment.

Definition at line 80 of file [OwnerTests.cs](#).

5.15.2.6 LastAssignedId_TracksCorrectly()

```
void SmartStay.Core.Tests.Models.OwnerTests.LastAssignedId_TracksCorrectly ( ) [inline]
```

Tests the Owner.LastAssignedId property for proper tracking of the last assigned owner ID.

Definition at line 209 of file [OwnerTests.cs](#).

5.15.2.7 Owner_ToString_ReturnsJson()

```
void SmartStay.Core.Tests.Models.OwnerTests.Owner_ToString_ReturnsJson ( ) [inline]
```

Tests the Owner.ToString() method to ensure it returns a valid JSON string representation of the owner.

Definition at line 138 of file [OwnerTests.cs](#).

5.15.2.8 PhoneNumber_SetAndGet_ValidatesValue()

```
void SmartStay.Core.Tests.Models.OwnerTests.PhoneNumber_SetAndGet_ValidatesValue ( ) [inline]
```

Tests the Owner.PhoneNumber property for proper value validation and assignment.

Definition at line 118 of file [OwnerTests.cs](#).

5.15.2.9 RemoveAccommodation_ValidAccommodation_RemovesSuccessfully()

```
void SmartStay.Core.Tests.Models.OwnerTests.RemoveAccommodation_ValidAccommodation_Removes↵  
Successfully ( ) [inline]
```

Tests the Owner.RemoveAccommodation method to ensure accommodations can be removed successfully.

Definition at line 189 of file [OwnerTests.cs](#).

The documentation for this class was generated from the following file:

- [OwnerTests.cs](#)

5.16 SmartStay.Validation.Tests.Validators.OwnerValidatorTests Class Reference

Contains unit tests for the OwnerValidator class. Tests the validation logic for owner-related data used in the [SmartStay](#) application.

Public Member Functions

- void [ValidateOwnerId_ValidId_ReturnsId](#) ()
Tests the OwnerValidator.ValidateOwnerId(int) method to ensure that it returns the ID when the ID is valid.
- void [ValidateOwnerId_InvalidId_ThrowsValidationException](#) ()
Tests the OwnerValidator.ValidateOwnerId(int) method to ensure that it throws a ValidationException when the ID is invalid.
- void [ValidateOwnerName_ValidName_ReturnsName](#) ()
Tests the OwnerValidator.ValidateOwnerName(string) method to ensure that it returns the name when the name is valid.
- void [ValidateOwnerName_InvalidName_ThrowsValidationException](#) ()
Tests the OwnerValidator.ValidateOwnerName(string) method to ensure that it throws a ValidationException when the name is invalid.
- void [ValidateOwnerEmail_ValidEmail_ReturnsEmail](#) ()
Tests the OwnerValidator.ValidateOwnerEmail(string) method to ensure that it returns the email when the email is valid.
- void [ValidateOwnerEmail_InvalidEmail_ThrowsValidationException](#) ()
Tests the OwnerValidator.ValidateOwnerEmail(string) method to ensure that it throws a ValidationException when the email is invalid.
- void [ValidateOwnerPhoneNumber_ValidPhoneNumber_ReturnsPhoneNumber](#) ()
Tests the OwnerValidator.ValidateOwnerPhoneNumber(string) method to ensure that it returns the phone number when the number is valid.
- void [ValidateOwnerPhoneNumber_InvalidPhoneNumber_ThrowsValidationException](#) ()
Tests the OwnerValidator.ValidateOwnerPhoneNumber(string) method to ensure that it throws a ValidationException when the phone number is invalid.

5.16.1 Detailed Description

Contains unit tests for the OwnerValidator class. Tests the validation logic for owner-related data used in the [SmartStay](#) application.

Definition at line 25 of file [OwnerValidatorTests.cs](#).

5.16.2 Member Function Documentation

5.16.2.1 ValidateOwnerEmail_InvalidEmail_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.OwnerValidatorTests.ValidateOwnerEmail_InvalidEmail_ThrowsValidationException ( ) [inline]
```

Tests the OwnerValidator.ValidateOwnerEmail(string) method to ensure that it throws a ValidationException when the email is invalid.

Definition at line 113 of file [OwnerValidatorTests.cs](#).

5.16.2.2 ValidateOwnerEmail_ValidEmail_ReturnsEmail()

```
void SmartStay.Validation.Tests.Validators.OwnerValidatorTests.ValidateOwnerEmail_ValidEmail_ReturnsEmail ( ) [inline]
```

Tests the OwnerValidator.ValidateOwnerEmail(string) method to ensure that it returns the email when the email is valid.

Definition at line 96 of file [OwnerValidatorTests.cs](#).

5.16.2.3 ValidateOwnerId_InvalidId_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.OwnerValidatorTests.ValidateOwnerId_InvalidId_↵  
ThrowsValidationException ( ) [inline]
```

Tests the OwnerValidator.ValidateOwnerId(int) method to ensure that it throws a ValidationException when the ID is invalid.

Definition at line 49 of file [OwnerValidatorTests.cs](#).

5.16.2.4 ValidateOwnerId_ValidId_ReturnsId()

```
void SmartStay.Validation.Tests.Validators.OwnerValidatorTests.ValidateOwnerId_ValidId_↵  
ReturnsId ( ) [inline]
```

Tests the OwnerValidator.ValidateOwnerId(int) method to ensure that it returns the ID when the ID is valid.

Definition at line 32 of file [OwnerValidatorTests.cs](#).

5.16.2.5 ValidateOwnerName_InvalidName_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.OwnerValidatorTests.ValidateOwnerName_InvalidName_↵  
_ThrowsValidationException ( ) [inline]
```

Tests the OwnerValidator.ValidateOwnerName(string) method to ensure that it throws a ValidationException when the name is invalid.

Definition at line 81 of file [OwnerValidatorTests.cs](#).

5.16.2.6 ValidateOwnerName_ValidName_ReturnsName()

```
void SmartStay.Validation.Tests.Validators.OwnerValidatorTests.ValidateOwnerName_ValidName_↵  
ReturnsName ( ) [inline]
```

Tests the OwnerValidator.ValidateOwnerName(string) method to ensure that it returns the name when the name is valid.

Definition at line 64 of file [OwnerValidatorTests.cs](#).

5.16.2.7 ValidateOwnerPhoneNumber_InvalidPhoneNumber_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.OwnerValidatorTests.ValidateOwnerPhoneNumber_↵  
InvalidPhoneNumber_ThrowsValidationException ( ) [inline]
```

Tests the OwnerValidator.ValidateOwnerPhoneNumber(string) method to ensure that it throws a ValidationException when the phone number is invalid.

Definition at line 145 of file [OwnerValidatorTests.cs](#).

5.16.2.8 ValidateOwnerPhoneNumber_ValidPhoneNumber_ReturnsPhoneNumber()

```
void SmartStay.Validation.Tests.Validators.OwnerValidatorTests.ValidateOwnerPhoneNumber_↔
ValidPhoneNumber_ReturnsPhoneNumber ( ) [inline]
```

Tests the OwnerValidator.ValidateOwnerPhoneNumber(string) method to ensure that it returns the phone number when the number is valid.

Definition at line 128 of file [OwnerValidatorTests.cs](#).

The documentation for this class was generated from the following file:

- [OwnerValidatorTests.cs](#)

5.17 SmartStay.IO.Tests.FileOperations.PathValidatorTests Class Reference

Contains unit tests for the PathValidator class.

Public Member Functions

- void [FileExists_NonExistentFile_ReturnsFalse](#) ()
Tests the PathValidator.FileExists(string) method to ensure it returns false for a non-existent file path.
- void [FileExists_ExistingFile_ReturnsTrue](#) ()
Tests the PathValidator.FileExists(string) method to ensure it returns true for an existing file path.
- void [IsValidFileType_NullOrEmptyFilePath_ThrowsArgumentException](#) (string filePath)
Tests the PathValidator.IsValidFileType(string, string) method to ensure it throws an exception when the file path is empty.
- void [IsValidFileType_ValidExtension_ReturnsTrue](#) ()
Tests the PathValidator.IsValidFileType(string, string) method to ensure it returns true for a file path with a matching extension.
- void [IsValidFileType_InvalidExtension_ReturnsFalse](#) ()
Tests the PathValidator.IsValidFileType(string, string) method to ensure it returns false for a file path with a non-matching extension.
- void [IsValidFileType_CaseInsensitiveExtensionComparison_ReturnsTrue](#) ()
Tests the PathValidator.IsValidFileType(string, string) method to ensure it performs a case-insensitive comparison of file extensions.

5.17.1 Detailed Description

Contains unit tests for the PathValidator class.

Definition at line 26 of file [PathValidatorTests.cs](#).

5.17.2 Member Function Documentation

5.17.2.1 FileExists_ExistingFile_ReturnsTrue()

```
void SmartStay.IO.Tests.FileOperations.PathValidatorTests.FileExists_ExistingFile_ReturnsTrue  
( ) [inline]
```

Tests the PathValidator.FileExists(string) method to ensure it returns true for an existing file path.

Definition at line 50 of file [PathValidatorTests.cs](#).

5.17.2.2 FileExists_NonExistentFile_ReturnsFalse()

```
void SmartStay.IO.Tests.FileOperations.PathValidatorTests.FileExists_NonExistentFile_Returns↵  
False ( ) [inline]
```

Tests the PathValidator.FileExists(string) method to ensure it returns false for a non-existent file path.

Definition at line 33 of file [PathValidatorTests.cs](#).

5.17.2.3 IsValidFileType_CaseInsensitiveExtensionComparison_ReturnsTrue()

```
void SmartStay.IO.Tests.FileOperations.PathValidatorTests.IsValidFileType_CaseInsensitive↵  
ExtensionComparison_ReturnsTrue ( ) [inline]
```

Tests the PathValidator.IsValidFileType(string, string) method to ensure it performs a case-insensitive comparison of file extensions.

Definition at line 129 of file [PathValidatorTests.cs](#).

5.17.2.4 IsValidFileType_InvalidExtension_ReturnsFalse()

```
void SmartStay.IO.Tests.FileOperations.PathValidatorTests.IsValidFileType_InvalidExtension_↵  
ReturnsFalse ( ) [inline]
```

Tests the PathValidator.IsValidFileType(string, string) method to ensure it returns false for a file path with a non-matching extension.

Definition at line 111 of file [PathValidatorTests.cs](#).

5.17.2.5 IsValidFileType_NullOrEmptyFilePath_ThrowsArgumentException()

```
void SmartStay.IO.Tests.FileOperations.PathValidatorTests.IsValidFileType_NullOrEmptyFile↵  
Path_ThrowsArgumentException (   
    string filePath ) [inline]
```

Tests the PathValidator.IsValidFileType(string, string) method to ensure it throws an exception when the file path is empty.

Definition at line 78 of file [PathValidatorTests.cs](#).

5.17.2.6 IsValidFileType_ValidExtension_ReturnsTrue()

```
void SmartStay.IO.Tests.FileOperations.PathValidatorTests.IsValidFileType_ValidExtension_↔
ReturnsTrue ( ) [inline]
```

Tests the PathValidator.IsValidFileType(string, string) method to ensure it returns true for a file path with a matching extension.

Definition at line 93 of file [PathValidatorTests.cs](#).

The documentation for this class was generated from the following file:

- [PathValidatorTests.cs](#)

5.18 SmartStay.Core.Tests.Models.PaymentTests Class Reference

Unit tests for the Payment class.

Public Member Functions

- void [Payment_ValidData_CreatesPayment](#) ()
Verifies that a valid payment is created successfully.
- void [Payment_InvalidReservationId_ThrowsValidationException](#) ()
Verifies that attempting to create a payment with an invalid reservation ID throws a ValidationException.
- void [Payment_InvalidAmount_ThrowsValidationException](#) ()
Verifies that attempting to create a payment with an invalid amount throws a ValidationException.
- void [Payment_UpdateValidStatus_UpdatesStatus](#) ()
Verifies that the payment status can be updated when valid.
- void [Payment_UpdateInvalidStatus_ThrowsValidationException](#) ()
Verifies that attempting to update the payment status with an invalid value throws a ValidationException.
- void [Payment_Uniquelds_AssignsIncrementalIds](#) ()
Verifies that the payment ID is generated uniquely for each payment.
- void [Payment_ToString_ReturnsJson](#) ()
Tests the Payment.ToString() method to ensure it returns a valid JSON string representation of the client.

5.18.1 Detailed Description

Unit tests for the Payment class.

Definition at line 26 of file [PaymentTests.cs](#).

5.18.2 Member Function Documentation

5.18.2.1 Payment_InvalidAmount_ThrowsValidationException()

```
void SmartStay.Core.Tests.Models.PaymentTests.Payment_InvalidAmount_ThrowsValidationException
( ) [inline]
```

Verifies that attempting to create a payment with an invalid amount throws a ValidationException.

Definition at line 77 of file [PaymentTests.cs](#).

5.18.2.2 Payment_InvalidReservationId_ThrowsValidationException()

```
void SmartStay.Core.Tests.Models.PaymentTests.Payment_InvalidReservationId_ThrowsValidationException ( ) [inline]
```

Verifies that attempting to create a payment with an invalid reservation ID throws a ValidationException.

Definition at line 58 of file [PaymentTests.cs](#).

5.18.2.3 Payment_ToString_ReturnsJson()

```
void SmartStay.Core.Tests.Models.PaymentTests.Payment_ToString_ReturnsJson ( ) [inline]
```

Tests the Payment.ToString() method to ensure it returns a valid JSON string representation of the client.

Definition at line 162 of file [PaymentTests.cs](#).

5.18.2.4 Payment_UniqueIds_AssignsIncrementalIds()

```
void SmartStay.Core.Tests.Models.PaymentTests.Payment_UniqueIds_AssignsIncrementalIds ( ) [inline]
```

Verifies that the payment ID is generated uniquely for each payment.

Definition at line 139 of file [PaymentTests.cs](#).

5.18.2.5 Payment_UpdateInvalidStatus_ThrowsValidationException()

```
void SmartStay.Core.Tests.Models.PaymentTests.Payment_UpdateInvalidStatus_ThrowsValidationException ( ) [inline]
```

Verifies that attempting to update the payment status with an invalid value throws a ValidationException.

Definition at line 120 of file [PaymentTests.cs](#).

5.18.2.6 Payment_UpdateValidStatus_UpdatesStatus()

```
void SmartStay.Core.Tests.Models.PaymentTests.Payment_UpdateValidStatus_UpdatesStatus ( ) [inline]
```

Verifies that the payment status can be updated when valid.

Definition at line 96 of file [PaymentTests.cs](#).

5.18.2.7 Payment_ValidData_CreatesPayment()

```
void SmartStay.Core.Tests.Models.PaymentTests.Payment_ValidData_CreatesPayment ( ) [inline]
```

Verifies that a valid payment is created successfully.

Definition at line 32 of file [PaymentTests.cs](#).

The documentation for this class was generated from the following file:

- [PaymentTests.cs](#)

5.19 SmartStay.Validation.Tests.Validators.PaymentValidatorTests Class Reference

Contains unit tests for the PaymentValidator class. Tests the validation logic for payment-related data in the [SmartStay](#) application.

Public Member Functions

- void [ValidatePrice_ValidPrice_ReturnsPrice](#) ()
Tests the PaymentValidator.ValidatePrice(decimal) method to ensure that it returns the price when the price is valid.
- void [ValidatePrice_InvalidPrice_ThrowsValidationException](#) ()
Tests the PaymentValidator.ValidatePrice(decimal) method to ensure that it throws a ValidationException when the price is invalid.
- void [ValidateTotalCost_ValidTotalCost_ReturnsTotalCost](#) ()
Tests the PaymentValidator.ValidateTotalCost(decimal) method to ensure that it returns the total cost when the total cost is valid.
- void [ValidateTotalCost_InvalidTotalCost_ThrowsValidationException](#) ()
Tests the PaymentValidator.ValidateTotalCost(decimal) method to ensure that it throws a ValidationException when the total cost is invalid.
- void [ValidatePaymentAmount_ValidAmount_ReturnsAmount](#) ()
Tests the PaymentValidator.ValidatePaymentAmount(decimal) method to ensure that it returns the payment amount when the amount is valid.
- void [ValidatePaymentAmount_InvalidAmount_ThrowsValidationException](#) ()
Tests the PaymentValidator.ValidatePaymentAmount(decimal) method to ensure that it throws a ValidationException when the payment amount is invalid.
- void [ValidatePaymentStatus_ValidStatus_ReturnsStatus](#) ()
Tests the PaymentValidator.ValidatePaymentStatus(PaymentStatus) method to ensure that it returns the payment status when the status is valid.
- void [ValidatePaymentStatus_InvalidStatus_ThrowsValidationException](#) ()
Tests the PaymentValidator.ValidatePaymentStatus(PaymentStatus) method to ensure that it throws a ValidationException when the payment status is invalid.
- void [ValidatePaymentMethod_ValidMethod_ReturnsMethod](#) ()
Tests the PaymentValidator.ValidatePaymentMethod(PaymentMethod) method to ensure that it returns the payment method when the method is valid.
- void [ValidatePaymentMethod_InvalidMethod_ThrowsValidationException](#) ()
Tests the PaymentValidator.ValidatePaymentMethod(PaymentMethod) method to ensure that it throws a ValidationException when the payment method is invalid.
- void [ValidatePayment_ValidPayment_ReturnsPayment](#) ()
Tests the PaymentValidator.ValidatePayment(decimal) method to ensure that it returns the payment value when the payment value is valid.
- void [ValidatePayment_InvalidPayment_ThrowsValidationException](#) ()
Tests the PaymentValidator.ValidatePayment(decimal) method to ensure that it throws a ValidationException when the payment value is invalid (negative).

5.19.1 Detailed Description

Contains unit tests for the `PaymentValidator` class. Tests the validation logic for payment-related data in the [SmartStay](#) application.

Definition at line 26 of file [PaymentValidatorTests.cs](#).

5.19.2 Member Function Documentation

5.19.2.1 `ValidatePayment_InvalidPayment_ThrowsValidationException()`

```
void SmartStay.Validation.Tests.Validators.PaymentValidatorTests.ValidatePayment_InvalidPayment_ThrowsValidationException ( ) [inline]
```

Tests the `PaymentValidator.ValidatePayment(decimal)` method to ensure that it throws a `ValidationException` when the payment value is invalid (negative).

Definition at line 210 of file [PaymentValidatorTests.cs](#).

5.19.2.2 `ValidatePayment_ValidPayment_ReturnsPayment()`

```
void SmartStay.Validation.Tests.Validators.PaymentValidatorTests.ValidatePayment_ValidPayment_ReturnsPayment ( ) [inline]
```

Tests the `PaymentValidator.ValidatePayment(decimal)` method to ensure that it returns the payment value when the payment value is valid.

Definition at line 193 of file [PaymentValidatorTests.cs](#).

5.19.2.3 `ValidatePaymentAmount_InvalidAmount_ThrowsValidationException()`

```
void SmartStay.Validation.Tests.Validators.PaymentValidatorTests.ValidatePaymentAmount_InvalidAmount_ThrowsValidationException ( ) [inline]
```

Tests the `PaymentValidator.ValidatePaymentAmount(decimal)` method to ensure that it throws a `ValidationException` when the payment amount is invalid.

Definition at line 114 of file [PaymentValidatorTests.cs](#).

5.19.2.4 `ValidatePaymentAmount_ValidAmount_ReturnsAmount()`

```
void SmartStay.Validation.Tests.Validators.PaymentValidatorTests.ValidatePaymentAmount_ValidAmount_ReturnsAmount ( ) [inline]
```

Tests the `PaymentValidator.ValidatePaymentAmount(decimal)` method to ensure that it returns the payment amount when the amount is valid.

Definition at line 97 of file [PaymentValidatorTests.cs](#).

5.19.2.5 ValidatePaymentMethod_InvalidMethod_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.PaymentValidatorTests.ValidatePaymentMethod_InvalidMethod_ThrowsValidationException ( ) [inline]
```

Tests the PaymentValidator.ValidatePaymentMethod(PaymentMethod) method to ensure that it throws a ValidationException when the payment method is invalid.

Definition at line 178 of file [PaymentValidatorTests.cs](#).

5.19.2.6 ValidatePaymentMethod_ValidMethod_ReturnsMethod()

```
void SmartStay.Validation.Tests.Validators.PaymentValidatorTests.ValidatePaymentMethod_ValidMethod_ReturnsMethod ( ) [inline]
```

Tests the PaymentValidator.ValidatePaymentMethod(PaymentMethod) method to ensure that it returns the payment method when the method is valid.

Definition at line 161 of file [PaymentValidatorTests.cs](#).

5.19.2.7 ValidatePaymentStatus_InvalidStatus_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.PaymentValidatorTests.ValidatePaymentStatus_InvalidStatus_ThrowsValidationException ( ) [inline]
```

Tests the PaymentValidator.ValidatePaymentStatus(PaymentStatus) method to ensure that it throws a ValidationException when the payment status is invalid.

Definition at line 146 of file [PaymentValidatorTests.cs](#).

5.19.2.8 ValidatePaymentStatus_ValidStatus_ReturnsStatus()

```
void SmartStay.Validation.Tests.Validators.PaymentValidatorTests.ValidatePaymentStatus_ValidStatus_ReturnsStatus ( ) [inline]
```

Tests the PaymentValidator.ValidatePaymentStatus(PaymentStatus) method to ensure that it returns the payment status when the status is valid.

Definition at line 129 of file [PaymentValidatorTests.cs](#).

5.19.2.9 ValidatePrice_InvalidPrice_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.PaymentValidatorTests.ValidatePrice_InvalidPrice_ThrowsValidationException ( ) [inline]
```

Tests the PaymentValidator.ValidatePrice(decimal) method to ensure that it throws a ValidationException when the price is invalid.

Definition at line 50 of file [PaymentValidatorTests.cs](#).

5.19.2.10 ValidatePrice_ValidPrice_ReturnsPrice()

```
void SmartStay.Validation.Tests.Validators.PaymentValidatorTests.ValidatePrice_ValidPrice_↵
ReturnsPrice ( ) [inline]
```

Tests the PaymentValidator.ValidatePrice(decimal) method to ensure that it returns the price when the price is valid.

Definition at line 33 of file [PaymentValidatorTests.cs](#).

5.19.2.11 ValidateTotalCost_InvalidTotalCost_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.PaymentValidatorTests.ValidateTotalCost_Invalid↵
TotalCost_ThrowsValidationException ( ) [inline]
```

Tests the PaymentValidator.ValidateTotalCost(decimal) method to ensure that it throws a ValidationException when the total cost is invalid.

Definition at line 82 of file [PaymentValidatorTests.cs](#).

5.19.2.12 ValidateTotalCost_ValidTotalCost_ReturnsTotalCost()

```
void SmartStay.Validation.Tests.Validators.PaymentValidatorTests.ValidateTotalCost_Valid↵
TotalCost_ReturnsTotalCost ( ) [inline]
```

Tests the PaymentValidator.ValidateTotalCost(decimal) method to ensure that it returns the total cost when the total cost is valid.

Definition at line 65 of file [PaymentValidatorTests.cs](#).

The documentation for this class was generated from the following file:

- [PaymentValidatorTests.cs](#)

5.20 SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests Class Reference

Contains unit tests for the PhoneNumberValidator class. Tests the validation logic for phone numbers in the [SmartStay](#) application.

Public Member Functions

- void [ValidatePhoneNumber_ValidPhoneNumber_ReturnsPhoneNumber](#) ()
Tests the PhoneNumberValidator.ValidatePhoneNumber(string) method to ensure that it returns the phone number when the phone number is valid.
- void [ValidatePhoneNumber_InvalidPhoneNumber_ThrowsValidationException](#) ()
Tests the PhoneNumberValidator.ValidatePhoneNumber(string) method to ensure that it throws a ValidationException when the phone number is invalid.
- void [IsValidPhoneNumber_ValidPhoneNumber_ReturnsTrue](#) ()
Tests the PhoneNumberValidator.IsValidPhoneNumber(string) method to ensure that it returns true for valid phone numbers.
- void [IsValidPhoneNumber_InvalidPhoneNumber_ReturnsFalse](#) ()
Tests the PhoneNumberValidator.IsValidPhoneNumber(string) method to ensure that it returns false for invalid phone numbers.
- void [ValidatePhoneNumber_EmptyPhoneNumber_ThrowsValidationException](#) ()
Tests the PhoneNumberValidator.ValidatePhoneNumber(string) method to ensure that it throws a ValidationException when the phone number is empty.
- void [ValidatePhoneNumber_NullPhoneNumber_ThrowsValidationException](#) ()
Tests the PhoneNumberValidator.ValidatePhoneNumber(string) method to ensure that it throws a ValidationException when the phone number is null.

5.20.1 Detailed Description

Contains unit tests for the `PhoneNumberValidator` class. Tests the validation logic for phone numbers in the [SmartStay](#) application.

Definition at line 25 of file [PhoneNumberValidatorTests.cs](#).

5.20.2 Member Function Documentation

5.20.2.1 IsValidPhoneNumber_InvalidPhoneNumber_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests.IsValidPhoneNumber_↵  
InvalidPhoneNumber_ReturnsFalse ( ) [inline]
```

Tests the `PhoneNumberValidator.IsValidPhoneNumber(string)` method to ensure that it returns false for invalid phone numbers.

Definition at line 82 of file [PhoneNumberValidatorTests.cs](#).

5.20.2.2 IsValidPhoneNumber_ValidPhoneNumber_ReturnsTrue()

```
void SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests.IsValidPhoneNumber_↵  
ValidPhoneNumber_ReturnsTrue ( ) [inline]
```

Tests the `PhoneNumberValidator.IsValidPhoneNumber(string)` method to ensure that it returns true for valid phone numbers.

Definition at line 65 of file [PhoneNumberValidatorTests.cs](#).

5.20.2.3 ValidatePhoneNumber_EmptyPhoneNumber_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests.ValidatePhoneNumber_↵  
EmptyPhoneNumber_ThrowsValidationException ( ) [inline]
```

Tests the `PhoneNumberValidator.ValidatePhoneNumber(string)` method to ensure that it throws a `Validation↵` Exception when the phone number is empty.

Definition at line 99 of file [PhoneNumberValidatorTests.cs](#).

5.20.2.4 ValidatePhoneNumber_InvalidPhoneNumber_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests.ValidatePhoneNumber_↵  
InvalidPhoneNumber_ThrowsValidationException ( ) [inline]
```

Tests the `PhoneNumberValidator.ValidatePhoneNumber(string)` method to ensure that it throws a `Validation↵` Exception when the phone number is invalid.

Definition at line 49 of file [PhoneNumberValidatorTests.cs](#).

5.20.2.5 ValidatePhoneNumber_NullPhoneNumber_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests.ValidatePhoneNumber_↵
NullPhoneNumber_ThrowsValidationException ( ) [inline]
```

Tests the PhoneNumberValidator.ValidatePhoneNumber(string) method to ensure that it throws a Validation↵Exception when the phone number is null.

Definition at line 115 of file [PhoneNumberValidatorTests.cs](#).

5.20.2.6 ValidatePhoneNumber_ValidPhoneNumber_ReturnsPhoneNumber()

```
void SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests.ValidatePhoneNumber_↵
ValidPhoneNumber_ReturnsPhoneNumber ( ) [inline]
```

Tests the PhoneNumberValidator.ValidatePhoneNumber(string) method to ensure that it returns the phone number when the phone number is valid.

Definition at line 32 of file [PhoneNumberValidatorTests.cs](#).

The documentation for this class was generated from the following file:

- [PhoneNumberValidatorTests.cs](#)

5.21 SmartStay.Core.Tests.Repositories.ReservationsTests Class Reference

Contains unit tests for the Reservations repository class. Tests include adding, removing, importing, exporting reservations, and serialization/deserialization processes.

Public Member Functions

- void [Add_ValidReservation_AddsReservationSuccessfully](#) ()
Tests the Reservations.Add(Reservation) method to ensure that a reservation is successfully added.
- void [Add_NullReservation_ThrowsArgumentNullException](#) ()
Tests the Reservations.Add(Reservation) method to ensure that attempting to add a null reservation throws an exception.
- void [Remove_ValidReservation_RemovesReservationSuccessfully](#) ()
Tests the Reservations.Remove(Reservation) method to ensure that a reservation is successfully removed.
- void [Remove_NonExistingReservation_ReturnsFalse](#) ()
Tests the Reservations.Remove(Reservation) method to ensure that attempting to remove a non-existing reservation returns false.
- void [Import_ValidData_ImportsReservationsWithPayments](#) ()
Tests the Reservations.Import(string) method to ensure that reservations with payments are imported correctly.
- void [Export_ValidData_ExportsReservationsWithPayments](#) ()
Tests the Reservations.Export method to ensure that reservations with payments are exported correctly.
- void [FindReservationById_ExistingId_ReturnsReservation](#) ()
Tests the Reservations.FindReservationById(int) method to ensure that it finds a reservation by its ID.
- void [FindReservationById_NonExistingId_ReturnsNull](#) ()

Tests the Reservations.FindReservationById(int) method to ensure that it returns null when a reservation with the specified ID does not exist.

- void [Save_ValidData_SavesToFile](#) ()

Tests the Reservations.Save(string) method to ensure that the reservations collection can be saved to a file.

- void [Load_ValidFile_LoadsReservations](#) ()

Tests the Reservations.Load(string) method to ensure that the reservations collection can be loaded from a file.

- void [Add_InvalidTotalCost_ThrowsValidationException](#) ()

Tests the Reservations.Add(Reservation) method to ensure that adding a reservation with invalid total cost throws an exception.

- void [Add_InvalidDateRange_ThrowsValidationException](#) ()

Tests the Reservations.Add(Reservation) method to ensure that adding a reservation with invalid dates throws an exception.

5.21.1 Detailed Description

Contains unit tests for the Reservations repository class. Tests include adding, removing, importing, exporting reservations, and serialization/deserialization processes.

Definition at line 29 of file [ReservationsTests.cs](#).

5.21.2 Member Function Documentation

5.21.2.1 Add_InvalidDateRange_ThrowsValidationException()

```
void SmartStay.Core.Tests.Repositories.ReservationsTests.Add_InvalidDateRange_ThrowsValidationException ( ) [inline]
```

Tests the Reservations.Add(Reservation) method to ensure that adding a reservation with invalid dates throws an exception.

Definition at line 325 of file [ReservationsTests.cs](#).

5.21.2.2 Add_InvalidTotalCost_ThrowsValidationException()

```
void SmartStay.Core.Tests.Repositories.ReservationsTests.Add_InvalidTotalCost_ThrowsValidationException ( ) [inline]
```

Tests the Reservations.Add(Reservation) method to ensure that adding a reservation with invalid total cost throws an exception.

Definition at line 311 of file [ReservationsTests.cs](#).

5.21.2.3 Add_NullReservation_ThrowsArgumentNullException()

```
void SmartStay.Core.Tests.Repositories.ReservationsTests.Add_NullReservation_ThrowsArgumentNullException ( ) [inline]
```

Tests the Reservations.Add(Reservation) method to ensure that attempting to add a null reservation throws an exception.

Definition at line 55 of file [ReservationsTests.cs](#).

5.21.2.4 Add_ValidReservation_AddsReservationSuccessfully()

```
void SmartStay.Core.Tests.Repositories.ReservationsTests.Add_ValidReservation_AddsReservation↵  
Successfully ( ) [inline]
```

Tests the Reservations.Add(Reservation) method to ensure that a reservation is successfully added.

Definition at line 35 of file [ReservationsTests.cs](#).

5.21.2.5 Export_ValidData_ExportsReservationsWithPayments()

```
void SmartStay.Core.Tests.Repositories.ReservationsTests.Export_ValidData_ExportsReservations↵  
WithPayments ( ) [inline]
```

Tests the Reservations.Export method to ensure that reservations with payments are exported correctly.

Definition at line 183 of file [ReservationsTests.cs](#).

5.21.2.6 FindReservationById_ExistingId_ReturnsReservation()

```
void SmartStay.Core.Tests.Repositories.ReservationsTests.FindReservationById_ExistingId_↵  
ReturnsReservation ( ) [inline]
```

Tests the Reservations.FindReservationById(int) method to ensure that it finds a reservation by its ID.

Definition at line 230 of file [ReservationsTests.cs](#).

5.21.2.7 FindReservationById_NonExistingId_ReturnsNull()

```
void SmartStay.Core.Tests.Repositories.ReservationsTests.FindReservationById_NonExistingId_↵  
ReturnsNull ( ) [inline]
```

Tests the Reservations.FindReservationById(int) method to ensure that it returns null when a reservation with the specified ID does not exist.

Definition at line 252 of file [ReservationsTests.cs](#).

5.21.2.8 Import_ValidData_ImportsReservationsWithPayments()

```
void SmartStay.Core.Tests.Repositories.ReservationsTests.Import_ValidData_ImportsReservations↵  
WithPayments ( ) [inline]
```

Tests the Reservations.Import(string) method to ensure that reservations with payments are imported correctly.

Definition at line 110 of file [ReservationsTests.cs](#).

5.21.2.9 Load_ValidFile_LoadsReservations()

```
void SmartStay.Core.Tests.Repositories.ReservationsTests.Load_ValidFile_LoadsReservations ( )  
[inline]
```

Tests the Reservations.Load(string) method to ensure that the reservations collection can be loaded from a file.

Definition at line 288 of file [ReservationsTests.cs](#).

5.21.2.10 Remove_NonExistingReservation_ReturnsFalse()

```
void SmartStay.Core.Tests.Repositories.ReservationsTests.Remove_NonExistingReservation_↵  
ReturnsFalse ( ) [inline]
```

Tests the Reservations.Remove(Reservation) method to ensure that attempting to remove a non-existing reservation returns false.

Definition at line 91 of file [ReservationsTests.cs](#).

5.21.2.11 Remove_ValidReservation_RemovesReservationSuccessfully()

```
void SmartStay.Core.Tests.Repositories.ReservationsTests.Remove_ValidReservation_Removes↵  
ReservationSuccessfully ( ) [inline]
```

Tests the Reservations.Remove(Reservation) method to ensure that a reservation is successfully removed.

Definition at line 70 of file [ReservationsTests.cs](#).

5.21.2.12 Save_ValidData_SavesToFile()

```
void SmartStay.Core.Tests.Repositories.ReservationsTests.Save_ValidData_SavesToFile ( ) [inline]
```

Tests the Reservations.Save(string) method to ensure that the reservations collection can be saved to a file.

Definition at line 269 of file [ReservationsTests.cs](#).

The documentation for this class was generated from the following file:

- [ReservationsTests.cs](#)

5.22 SmartStay.Core.Tests.Models.ReservationTests Class Reference

Contains unit tests for the Reservation class. Tests include validation, property assignments, payment methods, and string representation.

Public Member Functions

- void [Constructor_ValidParameters_InitializesReservation](#) ()
Tests the constructor of the Reservation class to ensure that it properly initializes a reservation with valid parameters.
- void [ToString_ReturnsValidJson](#) ()
Tests the Reservation.ToString method to ensure it returns a valid JSON representation.
- void [CheckIn_StatusPending_ChangesStatusToCheckedIn](#) ()
Tests the Reservation.CheckIn method to ensure that the reservation is correctly marked as CheckedIn when the status is Pending.
- void [CheckIn_StatusNotPending_ReturnsFalse](#) ()
Tests the Reservation.CheckIn method to ensure that the reservation does not change status if it is not Pending.
- void [CheckOut_StatusCheckedIn_ChangesStatusToCheckedOut](#) ()
Tests the Reservation.CheckOut method to ensure that the reservation is correctly marked as CheckedOut when the status is CheckedIn.
- void [CheckOut_StatusNotCheckedIn_ReturnsFalse](#) ()
Tests the Reservation.CheckOut method to ensure that the reservation does not change status if it is not CheckedIn.
- void [MakePayment_ValidPayment_UpdatesAmountPaid](#) ()
Tests the Reservation.MakePayment method to ensure that it correctly processes payments and updates the amount paid.
- void [IsFullyPaid_FullyPaid_ReturnsTrue](#) ()
Tests the Reservation.IsFullyPaid method to ensure it correctly identifies whether the reservation has been fully paid.

5.22.1 Detailed Description

Contains unit tests for the Reservation class. Tests include validation, property assignments, payment methods, and string representation.

Definition at line 27 of file [ReservationTests.cs](#).

5.22.2 Member Function Documentation

5.22.2.1 CheckIn_StatusNotPending_ReturnsFalse()

```
void SmartStay.Core.Tests.Models.ReservationTests.CheckIn_StatusNotPending_ReturnsFalse ( )  
[inline]
```

Tests the Reservation.CheckIn method to ensure that the reservation does not change status if it is not Pending.

Definition at line 108 of file [ReservationTests.cs](#).

5.22.2.2 CheckIn_StatusPending_ChangesStatusToCheckedIn()

```
void SmartStay.Core.Tests.Models.ReservationTests.CheckIn_StatusPending_ChangesStatusTo←  
CheckedIn ( ) [inline]
```

Tests the Reservation.CheckIn method to ensure that the reservation is correctly marked as CheckedIn when the status is Pending.

Definition at line 89 of file [ReservationTests.cs](#).

5.22.2.3 CheckOut_StatusCheckedIn_ChangesStatusToCheckedOut()

```
void SmartStay.Core.Tests.Models.ReservationTests.CheckOut_StatusCheckedIn_ChangesStatusToCheckedOut ( ) [inline]
```

Tests the Reservation.CheckOut method to ensure that the reservation is correctly marked as CheckedOut when the status is CheckedIn.

Definition at line 128 of file [ReservationTests.cs](#).

5.22.2.4 CheckOut_StatusNotCheckedIn_ReturnsFalse()

```
void SmartStay.Core.Tests.Models.ReservationTests.CheckOut_StatusNotCheckedIn_ReturnsFalse ( ) [inline]
```

Tests the Reservation.CheckOut method to ensure that the reservation does not change status if it is not CheckedIn.

Definition at line 148 of file [ReservationTests.cs](#).

5.22.2.5 Constructor_ValidParameters_InitializesReservation()

```
void SmartStay.Core.Tests.Models.ReservationTests.Constructor_ValidParameters_InitializesReservation ( ) [inline]
```

Tests the constructor of the Reservation class to ensure that it properly initializes a reservation with valid parameters.

Definition at line 34 of file [ReservationTests.cs](#).

5.22.2.6 IsFullyPaid_FullyPaid_ReturnsTrue()

```
void SmartStay.Core.Tests.Models.ReservationTests.IsFullyPaid_FullyPaid_ReturnsTrue ( ) [inline]
```

Tests the Reservation.IsFullyPaid method to ensure it correctly identifies whether the reservation has been fully paid.

Definition at line 187 of file [ReservationTests.cs](#).

5.22.2.7 MakePayment_ValidPayment_UpdatesAmountPaid()

```
void SmartStay.Core.Tests.Models.ReservationTests.MakePayment_ValidPayment_UpdatesAmountPaid ( ) [inline]
```

Tests the Reservation.MakePayment method to ensure that it correctly processes payments and updates the amount paid.

Definition at line 168 of file [ReservationTests.cs](#).

5.22.2.8 ToString_ReturnsValidJson()

```
void SmartStay.Core.Tests.Models.ReservationTests.ToString_ReturnsValidJson ( ) [inline]
```

Tests the Reservation.ToString method to ensure it returns a valid JSON representation.

Definition at line 64 of file [ReservationTests.cs](#).

The documentation for this class was generated from the following file:

- [ReservationTests.cs](#)

5.23 SmartStay.Validation.Tests.Validators.ReservationValidatorTests Class Reference

Contains unit tests for the ReservationValidator class. Tests the validation logic for reservation-related data in the [SmartStay](#) application.

Public Member Functions

- void [ValidateReservationStatus_ValidReservationStatus_ReturnsReservationStatus](#) ()
Tests the ReservationValidator.ValidateReservationStatus(ReservationStatus) method to ensure it returns the reservation status when the status is valid.
- void [ValidateReservationStatus_InvalidReservationStatus_ThrowsValidationException](#) ()
Tests the ReservationValidator.ValidateReservationStatus(ReservationStatus) method to ensure it throws a ValidationException when the status is invalid.
- void [IsValidReservationStatus_ValidReservationStatus_ReturnsTrue](#) ()
Tests the ReservationValidator.IsValidReservationStatus(ReservationStatus) method to ensure it returns true for valid reservation statuses.
- void [IsValidReservationStatus_InvalidReservationStatus_ReturnsFalse](#) ()
Tests the ReservationValidator.IsValidReservationStatus(ReservationStatus) method to ensure it returns false for invalid reservation statuses.
- void [ValidateReservationId_ValidReservationId_ReturnsReservationId](#) ()
Tests the ReservationValidator.ValidateReservationId(int) method to ensure it returns the reservation ID when the ID is valid.
- void [ValidateReservationId_InvalidReservationId_ThrowsValidationException](#) ()
Tests the ReservationValidator.ValidateReservationId(int) method to ensure it throws a ValidationException when the reservation ID is invalid.
- void [IsValidReservationId_ValidReservationId_ReturnsTrue](#) ()
Tests the ReservationValidator.IsValidReservationId(int) method to ensure it returns true for valid reservation IDs.
- void [IsValidReservationId_InvalidReservationId_ReturnsFalse](#) ()
Tests the ReservationValidator.IsValidReservationId(int) method to ensure it returns false for invalid reservation IDs.

5.23.1 Detailed Description

Contains unit tests for the ReservationValidator class. Tests the validation logic for reservation-related data in the [SmartStay](#) application.

Definition at line 26 of file [ReservationValidatorTests.cs](#).

5.23.2 Member Function Documentation

5.23.2.1 IsValidReservationId_InvalidReservationId_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.ReservationValidatorTests.IsValidReservationId_↵  
InvalidReservationId_ReturnsFalse ( ) [inline]
```

Tests the ReservationValidator.IsValidReservationId(int) method to ensure it returns false for invalid reservation IDs.

Definition at line 149 of file [ReservationValidatorTests.cs](#).

5.23.2.2 IsValidReservationId_ValidReservationId_ReturnsTrue()

```
void SmartStay.Validation.Tests.Validators.ReservationValidatorTests.IsValidReservationId_↵  
ValidReservationId_ReturnsTrue ( ) [inline]
```

Tests the ReservationValidator.IsValidReservationId(int) method to ensure it returns true for valid reservation IDs.

Definition at line 132 of file [ReservationValidatorTests.cs](#).

5.23.2.3 IsValidReservationStatus_InvalidReservationStatus_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.ReservationValidatorTests.IsValidReservation_↵  
Status_InvalidReservationStatus_ReturnsFalse ( ) [inline]
```

Tests the ReservationValidator.IsValidReservationStatus(ReservationStatus) method to ensure it returns false for invalid reservation statuses.

Definition at line 83 of file [ReservationValidatorTests.cs](#).

5.23.2.4 IsValidReservationStatus_ValidReservationStatus_ReturnsTrue()

```
void SmartStay.Validation.Tests.Validators.ReservationValidatorTests.IsValidReservation_↵  
Status_ValidReservationStatus_ReturnsTrue ( ) [inline]
```

Tests the ReservationValidator.IsValidReservationStatus(ReservationStatus) method to ensure it returns true for valid reservation statuses.

Definition at line 66 of file [ReservationValidatorTests.cs](#).

5.23.2.5 ValidateReservationId_InvalidReservationId_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.ReservationValidatorTests.ValidateReservationId_↵  
InvalidReservationId_ThrowsValidationException ( ) [inline]
```

Tests the ReservationValidator.ValidateReservationId(int) method to ensure it throws a ValidationException when the reservation ID is invalid.

Definition at line 117 of file [ReservationValidatorTests.cs](#).

5.23.2.6 ValidateReservationId_ValidReservationId_ReturnsReservationId()

```
void SmartStay.Validation.Tests.Validators.ReservationValidatorTests.ValidateReservationId↵
ValidReservationId_ReturnsReservationId ( ) [inline]
```

Tests the ReservationValidator.ValidateReservationId(int) method to ensure it returns the reservation ID when the ID is valid.

Definition at line 100 of file [ReservationValidatorTests.cs](#).

5.23.2.7 ValidateReservationStatus_InvalidReservationStatus_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.ReservationValidatorTests.ValidateReservation↵
Status_InvalidReservationStatus_ThrowsValidationException ( ) [inline]
```

Tests the ReservationValidator.ValidateReservationStatus(ReservationStatus) method to ensure it throws a ValidationException when the status is invalid.

Definition at line 50 of file [ReservationValidatorTests.cs](#).

5.23.2.8 ValidateReservationStatus_ValidReservationStatus_ReturnsReservationStatus()

```
void SmartStay.Validation.Tests.Validators.ReservationValidatorTests.ValidateReservation↵
Status_ValidReservationStatus_ReturnsReservationStatus ( ) [inline]
```

Tests the ReservationValidator.ValidateReservationStatus(ReservationStatus) method to ensure it returns the reservation status when the status is valid.

Definition at line 33 of file [ReservationValidatorTests.cs](#).

The documentation for this class was generated from the following file:

- [ReservationValidatorTests.cs](#)

5.24 SmartStay.Core.Tests.Models.RoomTests Class Reference

Contains unit tests for the Room class. Tests include validation, property assignments, reservation management, cost calculation, and string representation.

Public Member Functions

- void [Constructor_ValidParameters_InitializesRoom](#) ()
Tests the constructor of the Room class to ensure that it properly initializes a room with valid parameters.
- void [ToString_ReturnsValidJson](#) ()
Tests the Room.ToString method to ensure it returns a valid JSON representation.
- void [IsAvailable_NoOverlap_ReturnsTrue](#) ()
Tests the Room.IsAvailable method to ensure it correctly checks availability for a room when the given date range is not overlapping with existing reservations.
- void [IsAvailable_Overlap_ReturnsFalse](#) ()
Tests the Room.IsAvailable method to ensure it correctly detects overlap with existing reservations.
- void [AddReservation_ValidDates_AddsReservation](#) ()
Tests the Room.AddReservation method to ensure a reservation can be added successfully.
- void [RemoveReservation_ExistingReservation_RemovesReservation](#) ()
Tests the Room.RemoveReservation method to ensure a reservation can be removed successfully.
- void [CalculateTotalCost_ValidDates_ReturnsCorrectCost](#) ()
Tests the Room.CalculateTotalCost method to ensure it calculates the correct total cost for a stay.
- void [CalculateTotalCost_EndDateBeforeStartDate_ThrowsArgumentException](#) ()
Tests the Room.CalculateTotalCost method to ensure it throws an exception when the end date is before the start date.

5.24.1 Detailed Description

Contains unit tests for the Room class. Tests include validation, property assignments, reservation management, cost calculation, and string representation.

Definition at line 27 of file [RoomTests.cs](#).

5.24.2 Member Function Documentation

5.24.2.1 AddReservation_ValidDates_AddsReservation()

```
void SmartStay.Core.Tests.Models.RoomTests.AddReservation_ValidDates_AddsReservation ( ) [inline]
```

Tests the Room.AddReservation method to ensure a reservation can be added successfully.

Definition at line 111 of file [RoomTests.cs](#).

5.24.2.2 CalculateTotalCost_EndDateBeforeStartDate_ThrowsArgumentException()

```
void SmartStay.Core.Tests.Models.RoomTests.CalculateTotalCost_EndDateBeforeStartDate_Throws←  
ArgumentException ( ) [inline]
```

Tests the Room.CalculateTotalCost method to ensure it throws an exception when the end date is before the start date.

Definition at line 170 of file [RoomTests.cs](#).

5.24.2.3 CalculateTotalCost_ValidDates_ReturnsCorrectCost()

```
void SmartStay.Core.Tests.Models.RoomTests.CalculateTotalCost_ValidDates_ReturnsCorrectCost (   
 ) [inline]
```

Tests the Room.CalculateTotalCost method to ensure it calculates the correct total cost for a stay.

Definition at line 151 of file [RoomTests.cs](#).

5.24.2.4 Constructor_ValidParameters_InitializesRoom()

```
void SmartStay.Core.Tests.Models.RoomTests.Constructor_ValidParameters_InitializesRoom ( )   
 [inline]
```

Tests the constructor of the Room class to ensure that it properly initializes a room with valid parameters.

Definition at line 34 of file [RoomTests.cs](#).

5.24.2.5 `IsAvailable_NoOverlap_ReturnsTrue()`

```
void SmartStay.Core.Tests.Models.RoomTests.IsAvailable_NoOverlap_ReturnsTrue ( ) [inline]
```

Tests the `Room.IsAvailable` method to ensure it correctly checks availability for a room when the given date range is not overlapping with existing reservations.

Definition at line 72 of file [RoomTests.cs](#).

5.24.2.6 `IsAvailable_Overlap_ReturnsFalse()`

```
void SmartStay.Core.Tests.Models.RoomTests.IsAvailable_Overlap_ReturnsFalse ( ) [inline]
```

Tests the `Room.IsAvailable` method to ensure it correctly detects overlap with existing reservations.

Definition at line 92 of file [RoomTests.cs](#).

5.24.2.7 `RemoveReservation_ExistingReservation_RemovesReservation()`

```
void SmartStay.Core.Tests.Models.RoomTests.RemoveReservation_ExistingReservation_Removes↵  
Reservation ( ) [inline]
```

Tests the `Room.RemoveReservation` method to ensure a reservation can be removed successfully.

Definition at line 130 of file [RoomTests.cs](#).

5.24.2.8 `ToString_ReturnsValidJson()`

```
void SmartStay.Core.Tests.Models.RoomTests.ToString_ReturnsValidJson ( ) [inline]
```

Tests the `Room.ToString` method to ensure it returns a valid JSON representation.

Definition at line 53 of file [RoomTests.cs](#).

The documentation for this class was generated from the following file:

- [RoomTests.cs](#)

5.25 `SmartStay.Validation.Tests.Validators.RoomValidatorTests` Class Reference

Contains unit tests for the `RoomValidator` class. Tests the validation logic for room-related data used in the [SmartStay](#) application.

Public Member Functions

- void [ValidateRoomType_ValidType_ReturnsRoomType](#) ()
Tests the RoomValidator.ValidateRoomType(RoomType) method to ensure that it returns the room type when it is valid.
- void [ValidateRoomType_InvalidType_ThrowsValidationException](#) ()
Tests the RoomValidator.ValidateRoomType(RoomType) method to ensure that it throws a ValidationException when the room type is invalid.
- void [ValidateAvailability_ValidStatus_ReturnsAvailability](#) ()
Tests the RoomValidator.ValidateAvailability(bool) method to ensure that it returns the availability status when it is valid.
- void [ValidateRoomId_ValidId_ReturnsRoomId](#) ()
Tests the RoomValidator.ValidateRoomId(int) method to ensure that it returns the room ID when it is valid.
- void [ValidateRoomId_InvalidId_ThrowsValidationException](#) ()
Tests the RoomValidator.ValidateRoomId(int) method to ensure that it throws a ValidationException when the room ID is invalid.
- void [IsValidRoomType_ValidType_ReturnsTrue](#) ()
Tests the RoomValidator.IsValidRoomType(RoomType) method to ensure that it returns true for a valid room type.
- void [IsValidRoomType_InvalidType_ReturnsFalse](#) ()
Tests the RoomValidator.IsValidRoomType(RoomType) method to ensure that it returns false for an invalid room type.
- void [IsValidAvailability_AnyStatus_ReturnsTrue](#) ()
Tests the RoomValidator.IsValidAvailability(bool) method to ensure that it always returns true (logic is tautological).

5.25.1 Detailed Description

Contains unit tests for the RoomValidator class. Tests the validation logic for room-related data used in the [SmartStay](#) application.

Definition at line 26 of file [RoomValidatorTests.cs](#).

5.25.2 Member Function Documentation

5.25.2.1 IsValidAvailability_AnyStatus_ReturnsTrue()

```
void SmartStay.Validation.Tests.Validators.RoomValidatorTests.IsValidAvailability_AnyStatus_↵
ReturnsTrue ( ) [inline]
```

Tests the RoomValidator.IsValidAvailability(bool) method to ensure that it always returns true (logic is tautological).

Definition at line 148 of file [RoomValidatorTests.cs](#).

5.25.2.2 IsValidRoomType_InvalidType_ReturnsFalse()

```
void SmartStay.Validation.Tests.Validators.RoomValidatorTests.IsValidRoomType_InvalidType_↵
ReturnsFalse ( ) [inline]
```

Tests the RoomValidator.IsValidRoomType(RoomType) method to ensure that it returns false for an invalid room type.

Definition at line 131 of file [RoomValidatorTests.cs](#).

5.25.2.3 IsValidRoomType_ValidType_ReturnsTrue()

```
void SmartStay.Validation.Tests.Validators.RoomValidatorTests.IsValidRoomType_ValidType_ReturnsTrue ( ) [inline]
```

Tests the RoomValidator.IsValidRoomType(RoomType) method to ensure that it returns true for a valid room type.

Definition at line 114 of file [RoomValidatorTests.cs](#).

5.25.2.4 ValidateAvailability_ValidStatus_ReturnsAvailability()

```
void SmartStay.Validation.Tests.Validators.RoomValidatorTests.ValidateAvailability_ValidStatus_ReturnsAvailability ( ) [inline]
```

Tests the RoomValidator.ValidateAvailability(bool) method to ensure that it returns the availability status when it is valid.

Definition at line 65 of file [RoomValidatorTests.cs](#).

5.25.2.5 ValidateRoomId_InvalidId_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.RoomValidatorTests.ValidateRoomId_InvalidId_ThrowsValidationException ( ) [inline]
```

Tests the RoomValidator.ValidateRoomId(int) method to ensure that it throws a ValidationException when the room ID is invalid.

Definition at line 99 of file [RoomValidatorTests.cs](#).

5.25.2.6 ValidateRoomId_ValidId_ReturnsRoomId()

```
void SmartStay.Validation.Tests.Validators.RoomValidatorTests.ValidateRoomId_ValidId_ReturnsRoomId ( ) [inline]
```

Tests the RoomValidator.ValidateRoomId(int) method to ensure that it returns the room ID when it is valid.

Definition at line 82 of file [RoomValidatorTests.cs](#).

5.25.2.7 ValidateRoomType_InvalidType_ThrowsValidationException()

```
void SmartStay.Validation.Tests.Validators.RoomValidatorTests.ValidateRoomType_InvalidType_ThrowsValidationException ( ) [inline]
```

Tests the RoomValidator.ValidateRoomType(RoomType) method to ensure that it throws a ValidationException when the room type is invalid.

Definition at line 50 of file [RoomValidatorTests.cs](#).

5.25.2.8 ValidateRoomType_ValidType_ReturnsRoomType()

```
void SmartStay.Validation.Tests.Validators.RoomValidatorTests.ValidateRoomType_ValidType_↵  
ReturnsRoomType ( ) [inline]
```

Tests the RoomValidator.ValidateRoomType(RoomType) method to ensure that it returns the room type when it is valid.

Definition at line 33 of file [RoomValidatorTests.cs](#).

The documentation for this class was generated from the following file:

- [RoomValidatorTests.cs](#)

5.26 SmartStay.Validation.Tests.ValidationErrorMessageTests Class Reference

Contains unit tests for the ValidationErrorMessage class. Ensures that error messages are correctly retrieved based on the provided error codes and that localization functions as expected.

Public Member Functions

- void [GetErrorMessage_ValidErrorCode_ReturnsLocalizedMessage](#) ()
Verifies that providing a valid ValidationErrorCode returns the correct localized error message.
- void [GetErrorMessage_InvalidErrorCode_ReturnsFallbackMessage](#) ()
Verifies that providing an invalid or undefined error code returns the fallback error message.
- void [GetErrorMessage_SupportsLocalization](#) ()
Verifies that error messages are correctly localized based on the current culture.

5.26.1 Detailed Description

Contains unit tests for the ValidationErrorMessage class. Ensures that error messages are correctly retrieved based on the provided error codes and that localization functions as expected.

Definition at line 27 of file [ValidationErrorMessageTests.cs](#).

5.26.2 Member Function Documentation

5.26.2.1 GetErrorMessage_InvalidErrorCode_ReturnsFallbackMessage()

```
void SmartStay.Validation.Tests.ValidationErrorMessageTests.GetErrorMessage_InvalidError↵  
Code_ReturnsFallbackMessage ( ) [inline]
```

Verifies that providing an invalid or undefined error code returns the fallback error message.

Definition at line 50 of file [ValidationErrorMessageTests.cs](#).

5.26.2.2 GetErrorMessage_SupportsLocalization()

```
void SmartStay.Validation.Tests.ValidationErrorMessageTests.GetErrorMessage_SupportsLocalization  
( ) [inline]
```

Verifies that error messages are correctly localized based on the current culture.

Definition at line 67 of file [ValidationErrorMessageTests.cs](#).

5.26.2.3 GetErrorMessage_ValidErrorCode_ReturnsLocalizedMessage()

```
void SmartStay.Validation.Tests.ValidationErrorMessageTests.GetErrorMessage_ValidErrorCode_↵  
ReturnsLocalizedMessage ( ) [inline]
```

Verifies that providing a valid ValidationErrorCode returns the correct localized error message.

Definition at line 33 of file [ValidationErrorMessageTests.cs](#).

The documentation for this class was generated from the following file:

- [ValidationErrorMessageTests.cs](#)

5.27 SmartStay.Validation.Tests.ValidationExceptionTests Class Reference

Contains unit tests for the ValidationException class. Ensures that exceptions are created with the expected error codes and messages.

Public Member Functions

- void [Constructor_WithErrorCode_SetsErrorCodeAndMessage](#) ()
Verifies that the ValidationException constructor correctly sets the error code and retrieves the corresponding error message.
- void [Constructor_WithUnknownErrorCode_UsesFallbackMessage](#) ()
Verifies that the ValidationException constructor handles unknown error codes by using the fallback error message while retaining the provided error code.

5.27.1 Detailed Description

Contains unit tests for the ValidationException class. Ensures that exceptions are created with the expected error codes and messages.

Definition at line 25 of file [ValidationExceptionTests.cs](#).

5.27.2 Member Function Documentation

5.27.2.1 Constructor_WithErrorCode_SetsErrorCodeAndMessage()

```
void SmartStay.Validation.Tests.ValidationExceptionTests.Constructor_WithErrorCode_SetsError↵  
CodeAndMessage ( ) [inline]
```

Verifies that the ValidationException constructor correctly sets the error code and retrieves the corresponding error message.

Definition at line 32 of file [ValidationExceptionTests.cs](#).

5.27.2.2 Constructor_WithUnknownErrorCode_UsesFallbackMessage()

```
void SmartStay.Validation.Tests.ValidationExceptionTests.Constructor_WithUnknownErrorCode_↵  
UsesFallbackMessage ( ) [inline]
```

Verifies that the ValidationException constructor handles unknown error codes by using the fallback error message while retaining the provided error code.

Definition at line 51 of file [ValidationExceptionTests.cs](#).

The documentation for this class was generated from the following file:

- [ValidationExceptionTests.cs](#)

Chapter 6

File Documentation

6.1 AccommodationTests.cs File Reference

Data Structures

- class [SmartStay.Core.Tests.Models.AccommodationTests](#)
Contains unit tests for the Accommodation class. Tests include validation, property assignments, room management, and string representation.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Core](#)
- namespace [SmartStay.Core.Tests](#)
- namespace [SmartStay.Core.Tests.Models](#)
The [SmartStay.Core.Tests.Models](#) namespace contains unit tests for the models used in the [SmartStay](#) application. These tests verify the correct behavior of the Accommodation class and its methods.

6.2 AccommodationTests.cs

[Go to the documentation of this file.](#)

```
00001
00010
00015 namespace SmartStay.Core.Tests.Models
00016 {
00017     using SmartStay.Core.Models;
00018     using SmartStay.Common.Enums;
00019     using SmartStay.Validation;
00020     using System;
00021     using System.Collections.Generic;
00022     using Xunit;
00023
00028 public class AccommodationTests
00029 {
00035     [Fact]
00036     public void Constructor_ValidParameters_InitializesAccommodation()
00037     {
00038         // Arrange
00039         var ownerId = 1;
00040         var type = AccommodationType.Hotel;
00041         var name = "Sunset Hotel";
00042         var address = "123 Beach Ave, Ocean City";
00043
00044         // Act
```

```

00045         var accommodation = new Accommodation(ownerId, type, name, address);
00046
00047         // Assert
00048         Assert.Equal(ownerId, accommodation.OwnerId);
00049         Assert.Equal(type, accommodation.Type);
00050         Assert.Equal(name, accommodation.Name);
00051         Assert.Equal(address, accommodation.Address);
00052         Assert.Empty(accommodation.Rooms); // No rooms by default
00053     }
00054
00055     [Fact]
00060     public void Constructor_InvalidName_ThrowsValidationException()
00061     {
00062         // Arrange
00063         var ownerId = 1;
00064         var type = AccommodationType.Hotel;
00065         var name = ""; // Invalid name
00066         var address = "123 Beach Ave, Ocean City";
00067
00068         // Act & Assert
00069         var exception = Assert.Throws<ValidationException>(() => new Accommodation(ownerId, type,
name, address));
00070
00071         Assert.Equal("The provided accommodation name is invalid.", exception.Message);
00072     }
00073
00078     [Fact]
00079     public void FindRoomById_RoomExists_ReturnsRoom()
00080     {
00081         // Arrange
00082         var accommodation = new Accommodation(1, AccommodationType.Hotel, "Enrique Hotel", "123 Beach
Ave");
00083         var room = new Room(RoomType.Single, 100.0m);
00084         accommodation.AddRoom(room);
00085
00086         // Act
00087         var foundRoom = accommodation.FindRoomById(room.Id);
00088
00089         // Assert
00090         Assert.NotNull(foundRoom);
00091         Assert.Equal(room.Id, foundRoom.Id);
00092     }
00093
00098     [Fact]
00099     public void FindRoomById_RoomDoesNotExist_ReturnsNull()
00100     {
00101         // Arrange
00102         var accommodation = new Accommodation(1, AccommodationType.Hotel, "Sunset Hotel", "123 Beach
Ave");
00103
00104         // Act
00105         var foundRoom = accommodation.FindRoomById(101); // Room ID does not exist
00106
00107         // Assert
00108         Assert.Null(foundRoom);
00109     }
00110
00115     [Fact]
00116     public void Owner_ToString_ReturnsJson()
00117     {
00118         // Arrange
00119         var owner = new Owner("Alice", "Johnson", "alice.johnson@example.com");
00120
00121         // Act
00122         var jsonString = owner.ToString();
00123
00124         // Assert
00125         Assert.Contains($"\"Id\": {owner.Id}", jsonString);
00126         Assert.Contains($"\"FirstName\": \"{owner.FirstName}\"", jsonString);
00127         Assert.Contains($"\"LastName\": \"{owner.LastName}\"", jsonString);
00128         Assert.Contains($"\"Email\": \"{owner.Email}\"", jsonString);
00129     }
00130
00134     [Fact]
00135     public void OwnerId_SetAndGet_CorrectlySetsOwnerId()
00136     {
00137         // Arrange
00138         #pragma warning disable IDE0017 // Simplify object initialization
00139         var accommodation = new Accommodation(1, AccommodationType.Hotel, "Sunset Hotel", "123 Beach
Ave");
00140         #pragma warning restore IDE0017 // Simplify object initialization
00141
00142         // Act
00143         accommodation.OwnerId = 2;
00144
00145         // Assert
00146         Assert.Equal(2, accommodation.OwnerId);

```

```

00147     }
00148
00153     [Fact]
00154     public void OwnerId_InvalidValue_ThrowsException()
00155     {
00156         // Arrange
00157         var accommodation = new Accommodation(1, AccommodationType.Hotel, "Sunset Hotel", "123 Beach
Ave");
00158
00159         // Act & Assert
00160         var exception = Assert.Throws<ValidationException>(() => accommodation.OwnerId = -1);
00161         Assert.Equal("The provided ID is invalid.", exception.Message);
00162     }
00163 }
00164 }

```

6.3 ClientTests.cs File Reference

Data Structures

- class [SmartStay.Core.Tests.Models.ClientTests](#)

Contains unit tests for the Client class. Tests include validation, property assignments, and string representation.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Core](#)
- namespace [SmartStay.Core.Tests](#)
- namespace [SmartStay.Core.Tests.Models](#)

The [SmartStay.Core.Tests.Models](#) namespace contains unit tests for the models used in the [SmartStay](#) application. These tests verify the correct behavior of the Accommodation class and its methods.

6.4 ClientTests.cs

[Go to the documentation of this file.](#)

```

00001
00010
00015 namespace SmartStay.Core.Tests.Models
00016 {
00017     using Xunit;
00018     using SmartStay.Core.Models;
00019     using SmartStay.Common.Enums;
00020     using SmartStay.Validation;
00021
00026 public class ClientTests
00027 {
00033     [Fact]
00034     public void Constructor_ValidParameters_InitializesClient()
00035     {
00036         // Arrange
00037         var firstName = "Enrique";
00038         var lastName = "Rodrigues";
00039         var email = "Enrique.Rodrigues@example.com";
00040
00041         // Act
00042         var client = new Client(firstName, lastName, email);
00043
00044         // Assert
00045         Assert.Equal(firstName, client.FirstName);
00046         Assert.Equal(lastName, client.LastName);
00047         Assert.Equal(email, client.Email);
00048         Assert.Equal(PaymentMethod.None, client.PreferredPaymentMethod); // Default value
00049         Assert.NotEqual(0, client.Id); // ID should be auto-assigned
00050     }
00051
00056     [Fact]
00057     public void Constructor_InvalidName_ThrowsValidationException()

```

```

00058     {
00059         // Arrange
00060         var invalidName = "";
00061         var validEmail = "enrique.rodriques@example.com";
00062
00063         // Act & Assert
00064         var exception = Assert.Throws<ValidationException>(() => new Client(invalidName, "Rodrigues",
validEmail));
00065
00066         Assert.Equal("The provided name is invalid.", exception.Message);
00067     }
00068
00073     [Fact]
00074     public void FirstName_SetAndGet_ValidatesValue()
00075     {
00076         // Arrange
00077         var client = new Client("Enrique", "Rodrigues", "enrique.rodriques@example.com");
00078
00079         // Act
00080         client.FirstName = "Jane"; // Valid name
00081
00082         // Assert
00083         Assert.Equal("Jane", client.FirstName);
00084
00085         // Act & Assert for invalid value
00086         Assert.Throws<ValidationException>(() => client.FirstName = "");
00087     }
00088
00093     [Fact]
00094     public void Email_SetAndGet_ValidatesValue()
00095     {
00096         // Arrange
00097         var client = new Client("Enrique", "Rodrigues", "enrique.rodriques@example.com");
00098
00099         // Act
00100         client.Email = "jane.doe@example.com"; // Valid email
00101
00102         // Assert
00103         Assert.Equal("jane.doe@example.com", client.Email);
00104
00105         // Act & Assert for invalid value
00106         Assert.Throws<ValidationException>(() => client.Email = "invalid-email");
00107     }
00108
00113     [Fact]
00114     public void Payment_ToString_ReturnsJson()
00115     {
00116         // Arrange
00117         int reservationId = 101;
00118         decimal amount = 300.00m;
00119         DateTime date = DateTime.UtcNow;
00120         PaymentMethod method = PaymentMethod.PayPal;
00121         PaymentStatus status = PaymentStatus.Completed;
00122
00123         var payment = new Payment(reservationId, amount, date, method, status);
00124
00125         // Act
00126         var jsonString = payment.ToString();
00127
00128         // Assert
00129         Assert.Contains("\"Id\":", jsonString);
00130         Assert.Contains($"\"ReservationId\": {reservationId}", jsonString);
00131         Assert.Contains($"\"Amount\": {amount}", jsonString);
00132         Assert.Contains($"\"Method\": \"{method}\"", jsonString);
00133         Assert.Contains($"\"Status\": \"{status}\"", jsonString);
00134     }
00135
00140     [Fact]
00141     public void PreferredPaymentMethod_SetAndGet_CorrectlyAssignsValue()
00142     {
00143         // Arrange
00144         var client = new Client("Enrique", "Rodrigues", "enrique.rodriques@example.com");
00145
00146         // Act
00147         client.PreferredPaymentMethod = PaymentMethod.MultiBanco;
00148
00149         // Assert
00150         Assert.Equal(PaymentMethod.MultiBanco, client.PreferredPaymentMethod);
00151     }
00152
00156     [Fact]
00157     public void Id_AutoGenerated_IsUniqueAndNonZero()
00158     {
00159         // Arrange
00160         var client1 = new Client("John", "Doe", "john.doe@example.com");
00161         var client2 = new Client("Jane", "Smith", "jane.smith@example.com");
00162

```

```

00163         // Act
00164         var id1 = client1.Id;
00165         var id2 = client2.Id;
00166
00167         // Assert
00168         Assert.NotEqual(0, id1);
00169         Assert.NotEqual(0, id2);
00170         Assert.NotEqual(id1, id2); // Ensure unique IDs
00171     }
00172 }
00173 }

```

6.5 OwnerTests.cs File Reference

Data Structures

- class [SmartStay.Core.Tests.Models.OwnerTests](#)

Contains unit tests for the Owner class. Tests include validation, property assignments, and methods for managing accommodations.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Core](#)
- namespace [SmartStay.Core.Tests](#)
- namespace [SmartStay.Core.Tests.Models](#)

The [SmartStay.Core.Tests.Models](#) namespace contains unit tests for the models used in the [SmartStay](#) application. These tests verify the correct behavior of the Accommodation class and its methods.

6.6 OwnerTests.cs

[Go to the documentation of this file.](#)

```

00001
00011
00016 namespace SmartStay.Core.Tests.Models
00017 {
00018     using Xunit;
00019     using SmartStay.Core.Models;
00020     using SmartStay.Validation;
00021     using SmartStay.Common.Enums;
00022
00027 public class OwnerTests
00028 {
00033     [Fact]
00034     public void Constructor_BasicDetails_InitializesOwner()
00035     {
00036         // Arrange
00037         var firstName = "Enrique";
00038         var lastName = "Rodrigues";
00039         var email = "enrique.rodrigues@example.com";
00040
00041         // Act
00042         var owner = new Owner(firstName, lastName, email);
00043
00044         // Assert
00045         Assert.Equal(firstName, owner.FirstName);
00046         Assert.Equal(lastName, owner.LastName);
00047         Assert.Equal(email, owner.Email);
00048         Assert.NotEqual(0, owner.Id); // ID should be auto-assigned
00049     }
00050
00055     [Fact]
00056     public void Constructor_AdditionalDetails_InitializesOwner()
00057     {
00058         // Arrange
00059         var firstName = "Bob";

```

```

00060         var lastName = "Smith";
00061         var email = "bob.smith@example.com";
00062         var phoneNumber = "+351777888999";
00063         var address = "123 Main Street";
00064
00065         // Act
00066         var owner = new Owner(firstName, lastName, email, phoneNumber, address);
00067
00068         // Assert
00069         Assert.Equal(firstName, owner.FirstName);
00070         Assert.Equal(lastName, owner.LastName);
00071         Assert.Equal(email, owner.Email);
00072         Assert.Equal(phoneNumber, owner.PhoneNumber);
00073         Assert.Equal(address, owner.Address);
00074     }
00075
00076     [Fact]
00077     public void FirstName_SetAndGet_ValidatesValue()
00078     {
00079         // Arrange
00080         var owner = new Owner("Alice", "Johnson", "alice.johnson@example.com");
00081
00082         // Act
00083         owner.FirstName = "Eve"; // Valid name
00084
00085         // Assert
00086         Assert.Equal("Eve", owner.FirstName);
00087
00088         // Act & Assert for invalid value
00089         Assert.Throws<ValidationException>(() => owner.FirstName = "");
00090     }
00091
00092     [Fact]
00093     public void Email_SetAndGet_ValidatesValue()
00094     {
00095         // Arrange
00096         var owner = new Owner("Alice", "Johnson", "alice.johnson@example.com");
00097
00098         // Act
00099         owner.Email = "eve.johnson@example.com"; // Valid email
00100
00101         // Assert
00102         Assert.Equal("eve.johnson@example.com", owner.Email);
00103
00104         // Act & Assert for invalid value
00105         Assert.Throws<ValidationException>(() => owner.Email = "invalid-email");
00106     }
00107
00108     [Fact]
00109     public void PhoneNumber_SetAndGet_ValidatesValue()
00110     {
00111         // Arrange
00112         var owner = new Owner("Alice", "Johnson", "alice.johnson@example.com");
00113
00114         // Act
00115         owner.PhoneNumber = "+9876543210"; // Valid phone number
00116
00117         // Assert
00118         Assert.Equal("+9876543210", owner.PhoneNumber);
00119
00120         // Act & Assert for invalid value
00121         Assert.Throws<ValidationException>(() => owner.PhoneNumber = "1234");
00122     }
00123
00124     [Fact]
00125     public void Owner_ToString_ReturnsJson()
00126     {
00127         // Arrange
00128         var owner = new Owner("Alice", "Johnson", "alice.johnson@example.com");
00129
00130         // Act
00131         var jsonString = owner.ToString();
00132
00133         // Assert
00134         Assert.Contains($"\"Id\": {owner.Id}", jsonString);
00135         Assert.Contains($"\"FirstName\": \"{owner.FirstName}\"", jsonString);
00136         Assert.Contains($"\"LastName\": \"{owner.LastName}\"", jsonString);
00137         Assert.Contains($"\"Email\": \"{owner.Email}\"", jsonString);
00138     }
00139
00140     [Fact]
00141     public void AddAccommodation_ValidAccommodation_AddsSuccessfully()
00142     {
00143         // Arrange
00144         var owner = new Owner("Alice", "Johnson", "alice.johnson@example.com");
00145         var accommodation = new Accommodation(ownerId: owner.Id, // Use the owner's ID
00146                                             type: AccommodationType.Hotel, // Specify a valid type

```

```

00164                                     name: "Hotel Paradise",           // Provide the name
00165                                     address: "123 Paradise Street" // Provide the address
00166     );
00167
00168     // Act
00169     var result = owner.AddAccommodation(accommodation);
00170
00171     // Assert
00172     Assert.True(result);
00173
00174     // Assert that the accommodation was added by checking multiple properties
00175     var addedAccommodation = owner.AccommodationsOwned.Find(a => a.Name == accommodation.Name);
00176
00177     Assert.NotNull(addedAccommodation); // Ensure an accommodation with matching name was added
00178     Assert.Equal(accommodation.Name, addedAccommodation.Name); // Verify Name
00179     Assert.Equal(accommodation.Address, addedAccommodation.Address); // Verify Address
00180     Assert.Equal(accommodation.OwnerId, addedAccommodation.OwnerId); // Verify OwnerId
00181     Assert.Equal(accommodation.Type, addedAccommodation.Type); // Verify Type
00182 }
00183
00184 [Fact]
00185 public void RemoveAccommodation_ValidAccommodation_RemovesSuccessfully()
00186 {
00187     // Arrange
00188     var owner = new Owner("Alice", "Johnson", "alice.johnson@example.com");
00189     var accommodation = new Accommodation { Name = "Hotel Paradise" };
00190     owner.AddAccommodation(accommodation);
00191
00192     // Act
00193     var result = owner.RemoveAccommodation(accommodation);
00194
00195     // Assert
00196     Assert.True(result);
00197     Assert.DoesNotContain(accommodation, owner.AccommodationsOwned);
00198 }
00199
00200 [Fact]
00201 public void LastAssignedId_TracksCorrectly()
00202 {
00203     // Arrange
00204     Owner.LastAssignedId = 0; // Reset for testing
00205     _ = new Owner("Alice", "Johnson", "alice.johnson@example.com");
00206     var owner2 = new Owner("Bob", "Smith", "bob.smith@example.com");
00207
00208     // Act
00209     var lastId = Owner.LastAssignedId;
00210
00211     // Assert
00212     Assert.Equal(owner2.Id, lastId);
00213 }
00214 }
00215 }
00216 }

```

6.7 PaymentTests.cs File Reference

Data Structures

- class [SmartStay.Core.Tests.Models.PaymentTests](#)

Unit tests for the Payment class.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Core](#)
- namespace [SmartStay.Core.Tests](#)
- namespace [SmartStay.Core.Tests.Models](#)

The [SmartStay.Core.Tests.Models](#) namespace contains unit tests for the models used in the [SmartStay](#) application. These tests verify the correct behavior of the [Accommodation](#) class and its methods.

6.8 PaymentTests.cs

[Go to the documentation of this file.](#)

```

00001
00010
00015 namespace SmartStay.Core.Tests.Models
00016 {
00017     using System;
00018     using SmartStay.Common.Enums;
00019     using SmartStay.Core.Models;
00020     using SmartStay.Validation;
00021     using Xunit;
00022
00026 public class PaymentTests
00027 {
00031     [Fact]
00032     public void Payment_ValidData_CreatesPayment()
00033     {
00034         // Arrange
00035         int reservationId = 101;
00036         decimal amount = 250.50m;
00037         DateTime date = DateTime.UtcNow;
00038         PaymentMethod method = PaymentMethod.PayPal;
00039         PaymentStatus status = PaymentStatus.Completed;
00040
00041         // Act
00042         var payment = new Payment(reservationId, amount, date, method, status);
00043
00044         // Assert
00045         Assert.Equal(reservationId, payment.ReservationId);
00046         Assert.Equal(amount, payment.Amount);
00047         Assert.Equal(date, payment.Date);
00048         Assert.Equal(method, payment.Method);
00049         Assert.Equal(status, payment.Status);
00050         Assert.True(payment.Id > 0);
00051     }
00052
00057     [Fact]
00058     public void Payment_InvalidReservationId_ThrowsValidationException()
00059     {
00060         // Arrange
00061         int invalidReservationId = -1;
00062         decimal amount = 100.00m;
00063         DateTime date = DateTime.UtcNow;
00064         PaymentMethod method = PaymentMethod.BankTransfer;
00065         PaymentStatus status = PaymentStatus.Pending;
00066
00067         // Act & Assert
00068         var exception =
00069             Assert.Throws<ValidationException>(() => new Payment(invalidReservationId, amount, date,
method, status));
00070         Assert.Equal(ValidationError.InvalidId, exception.ErrorCode);
00071     }
00072
00076     [Fact]
00077     public void Payment_InvalidAmount_ThrowsValidationException()
00078     {
00079         // Arrange
00080         int reservationId = 101;
00081         decimal invalidAmount = -50.00m;
00082         DateTime date = DateTime.UtcNow;
00083         PaymentMethod method = PaymentMethod.PayPal;
00084         PaymentStatus status = PaymentStatus.Completed;
00085
00086         // Act & Assert
00087         var exception =
00088             Assert.Throws<ValidationException>(() => new Payment(reservationId, invalidAmount, date,
method, status));
00089         Assert.Equal(ValidationError.InvalidPaymentValue, exception.ErrorCode);
00090     }
00091
00095     [Fact]
00096     public void Payment_UpdateValidStatus_UpdatesStatus()
00097     {
00098         // Arrange
00099         int reservationId = 101;
00100         decimal amount = 150.00m;
00101         DateTime date = DateTime.UtcNow;
00102         PaymentMethod method = PaymentMethod.BankTransfer;
00103         PaymentStatus initialStatus = PaymentStatus.Pending;
00104         PaymentStatus updatedStatus = PaymentStatus.Completed;
00105
00106         var payment = new Payment(reservationId, amount, date, method, initialStatus);
00107
00108         // Act

```

```

00109         payment.Status = updatedStatus;
00110
00111         // Assert
00112         Assert.Equal(updatedStatus, payment.Status);
00113     }
00114
00119     [Fact]
00120     public void Payment_UpdateInvalidStatus_ThrowsValidationException()
00121     {
00122         // Arrange
00123         int reservationId = 101;
00124         decimal amount = 200.00m;
00125         DateTime date = DateTime.UtcNow;
00126         PaymentMethod method = PaymentMethod.MultiBanco;
00127         PaymentStatus initialStatus = PaymentStatus.Pending;
00128         var payment = new Payment(reservationId, amount, date, method, initialStatus);
00129
00130         // Act & Assert
00131         var exception = Assert.Throws<ValidationException>(() => payment.Status =
00132             (PaymentStatus) (-1));
00133         Assert.Equal(ValidationErrorCode.InvalidPaymentStatus, exception.ErrorCode);
00134
00138     [Fact]
00139     public void Payment_UniqueIds_AssignsIncrementalIds()
00140     {
00141         // Arrange
00142         int reservationId = 101;
00143         decimal amount = 100.00m;
00144         DateTime date = DateTime.UtcNow;
00145         PaymentMethod method = PaymentMethod.MultiBanco;
00146         PaymentStatus status = PaymentStatus.Completed;
00147
00148         // Act
00149         var payment1 = new Payment(reservationId, amount, date, method, status);
00150         var payment2 = new Payment(reservationId, amount, date, method, status);
00151
00152         // Assert
00153         Assert.NotEqual(payment1.Id, payment2.Id);
00154         Assert.True(payment2.Id > payment1.Id);
00155     }
00156
00161     [Fact]
00162     public void Payment_ToString_ReturnsJson()
00163     {
00164         // Arrange
00165         int reservationId = 101;
00166         decimal amount = 300.00m;
00167         DateTime date = DateTime.UtcNow;
00168         PaymentMethod method = PaymentMethod.PayPal;
00169         PaymentStatus status = PaymentStatus.Completed;
00170
00171         var payment = new Payment(reservationId, amount, date, method, status);
00172
00173         // Act
00174         var jsonString = payment.ToString();
00175
00176         // Assert
00177         Assert.Contains($"\"Id\": {payment.Id}", jsonString);
00178         Assert.Contains($"\"ReservationId\": {reservationId}", jsonString);
00179         Assert.Contains($"\"Amount\": {amount}", jsonString);
00180         Assert.Contains($"\"Method\": \"{method}\"", jsonString);
00181         Assert.Contains($"\"Status\": \"{status}\"", jsonString);
00182     }
00183 }
00184 }

```

6.9 ReservationTests.cs File Reference

Data Structures

- class [SmartStay.Core.Tests.Models.ReservationTests](#)

Contains unit tests for the Reservation class. Tests include validation, property assignments, payment methods, and string representation.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Core](#)
- namespace [SmartStay.Core.Tests](#)
- namespace [SmartStay.Core.Tests.Models](#)

The [SmartStay.Core.Tests.Models](#) namespace contains unit tests for the models used in the [SmartStay](#) application. These tests verify the correct behavior of the [Accommodation](#) class and its methods.

6.10 ReservationTests.cs

[Go to the documentation of this file.](#)

```

00001
00011
00016 namespace SmartStay.Core.Tests.Models
00017 {
00018     using SmartStay.Core.Models;
00019     using SmartStay.Common.Enums;
00020     using System;
00021     using Xunit;
00022
00027 public class ReservationTests
00028 {
00033     [Fact]
00034     public void Constructor_ValidParameters_InitializesReservation()
00035     {
00036         // Arrange
00037         var clientId = 1;
00038         var accommodationId = 2;
00039         var roomId = 3;
00040         var accommodationType = AccommodationType.Hotel;
00041         var checkInDate = new DateTime(2024, 12, 1);
00042         var checkOutDate = new DateTime(2024, 12, 5);
00043         var totalCost = 500.0m;
00044
00045         // Act
00046         var reservation =
00047             new Reservation(clientId, accommodationId, roomId, accommodationType, checkInDate,
00048                 checkOutDate, totalCost);
00049
00049         // Assert
00050         Assert.Equal(clientId, reservation.ClientId);
00051         Assert.Equal(accommodationId, reservation.AccommodationId);
00052         Assert.Equal(roomId, reservation.RoomId);
00053         Assert.Equal(accommodationType, reservation.AccommodationType);
00054         Assert.Equal(checkInDate, reservation.CheckInDate);
00055         Assert.Equal(checkOutDate, reservation.CheckOutDate);
00056         Assert.Equal(totalCost, reservation.TotalCost);
00057         Assert.Equal(ReservationStatus.Pending, reservation.Status); // Default status
00058     }
00059
00063     [Fact]
00064     public void ToString_ReturnsValidJson()
00065     {
00066         // Arrange
00067         var reservation = new Reservation(1, 2, 3, AccommodationType.Hotel, new DateTime(2024, 12, 1),
00068             new DateTime(2024, 12, 5), 500.0m);
00069
00070         // Act
00071         var jsonString = reservation.ToString();
00072
00073         // Assert
00074         Assert.Contains($"\"ClientId\": {reservation.ClientId}", jsonString);
00075         Assert.Contains($"\"AccommodationId\": {reservation.AccommodationId}", jsonString);
00076         Assert.Contains($"\"RoomId\": {reservation.RoomId}", jsonString);
00077         Assert.Contains($"\"AccommodationType\": \"{reservation.AccommodationType}\"", jsonString);
00078         Assert.Contains($"\"CheckInDate\": \"{reservation.CheckInDate:yyyy-MM-ddTHH:mm:ss}\"",
00079             jsonString);
00079         Assert.Contains($"\"CheckOutDate\": \"{reservation.CheckOutDate:yyyy-MM-ddTHH:mm:ss}\"",
00080             jsonString);
00080         Assert.Contains($"\"TotalCost\": {reservation.TotalCost}", jsonString);
00081         Assert.Contains($"\"Status\": \"{reservation.Status}\"", jsonString);
00082     }
00083
00088     [Fact]
00089     public void CheckIn_StatusPending_ChangesStatusToCheckedIn()
00090     {

```

```
00091         // Arrange
00092         var reservation = new Reservation(1, 2, 3, AccommodationType.Hotel, new DateTime(2024, 12, 1),
00093             new DateTime(2024, 12, 5), 500.0m);
00094
00095         // Act
00096         var result = reservation.CheckIn();
00097
00098         // Assert
00099         Assert.True(result);
00100         Assert.Equal(ReservationStatus.CheckedIn, reservation.Status);
00101     }
00102
00107     [Fact]
00108     public void CheckIn_StatusNotPending_ReturnsFalse()
00109     {
00110         // Arrange
00111         var reservation = new Reservation(1, 2, 3, AccommodationType.Hotel, new DateTime(2024, 12, 1),
00112             new DateTime(2024, 12, 5), 500.0m);
00113         reservation.Status = ReservationStatus.CheckedOut;
00114
00115         // Act
00116         var result = reservation.CheckIn();
00117
00118         // Assert
00119         Assert.False(result);
00120         Assert.Equal(ReservationStatus.CheckedOut, reservation.Status);
00121     }
00122
00127     [Fact]
00128     public void CheckOut_StatusCheckedIn_ChangesStatusToCheckedOut()
00129     {
00130         // Arrange
00131         var reservation = new Reservation(1, 2, 3, AccommodationType.Hotel, new DateTime(2024, 12, 1),
00132             new DateTime(2024, 12, 5), 500.0m);
00133         reservation.Status = ReservationStatus.CheckedIn;
00134
00135         // Act
00136         var result = reservation.CheckOut();
00137
00138         // Assert
00139         Assert.True(result);
00140         Assert.Equal(ReservationStatus.CheckedOut, reservation.Status);
00141     }
00142
00147     [Fact]
00148     public void CheckOut_StatusNotCheckedIn_ReturnsFalse()
00149     {
00150         // Arrange
00151         var reservation = new Reservation(1, 2, 3, AccommodationType.Hotel, new DateTime(2024, 12, 1),
00152             new DateTime(2024, 12, 5), 500.0m);
00153         reservation.Status = ReservationStatus.Pending;
00154
00155         // Act
00156         var result = reservation.CheckOut();
00157
00158         // Assert
00159         Assert.False(result);
00160         Assert.Equal(ReservationStatus.Pending, reservation.Status);
00161     }
00162
00167     [Fact]
00168     public void MakePayment_ValidPayment_UpdatesAmountPaid()
00169     {
00170         // Arrange
00171         var reservation = new Reservation(1, 2, 3, AccommodationType.Hotel, new DateTime(2024, 12, 1),
00172             new DateTime(2024, 12, 5), 500.0m);
00173
00174         // Act
00175         var paymentResult = reservation.MakePayment(200.0m, PaymentMethod.BankTransfer);
00176
00177         // Assert
00178         Assert.Equal(PaymentResult.Success, paymentResult);
00179         Assert.Equal(200.0m, reservation.AmountPaid);
00180     }
00181
00186     [Fact]
00187     public void IsFullyPaid_FullyPaid_ReturnsTrue()
00188     {
00189         // Arrange
00190         var reservation = new Reservation(1, 2, 3, AccommodationType.Hotel, new DateTime(2024, 12, 1),
00191             new DateTime(2024, 12, 5), 500.0m);
00192         reservation.AmountPaid = 500.0m;
00193
00194         // Act
00195         var result = reservation.IsFullyPaid();
00196
00197         // Assert
```

```

00198         Assert.True(result);
00199     }
00200 }
00201 }

```

6.11 RoomTests.cs File Reference

Data Structures

- class [SmartStay.Core.Tests.Models.RoomTests](#)

Contains unit tests for the Room class. Tests include validation, property assignments, reservation management, cost calculation, and string representation.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Core](#)
- namespace [SmartStay.Core.Tests](#)
- namespace [SmartStay.Core.Tests.Models](#)

The [SmartStay.Core.Tests.Models](#) namespace contains unit tests for the models used in the [SmartStay](#) application. These tests verify the correct behavior of the Accommodation class and its methods.

6.12 RoomTests.cs

[Go to the documentation of this file.](#)

```

00001
00011
00016 namespace SmartStay.Core.Tests.Models
00017 {
00018     using SmartStay.Core.Models;
00019     using SmartStay.Common.Enums;
00020     using System;
00021     using Xunit;
00022
00027 public class RoomTests
00028 {
00033     [Fact]
00034     public void Constructor_ValidParameters_InitializesRoom()
00035     {
00036         // Arrange
00037         var roomType = RoomType.Single;
00038         var pricePerNight = 100.0m;
00039
00040         // Act
00041         var room = new Room(roomType, pricePerNight);
00042
00043         // Assert
00044         Assert.Equal(roomType, room.Type);
00045         Assert.Equal(pricePerNight, room.PricePerNight);
00046         Assert.True(room.Id > 0); // ID should be positive and unique
00047     }
00048
00052     [Fact]
00053     public void ToString_ReturnsValidJson()
00054     {
00055         // Arrange
00056         var room = new Room(RoomType.Single, 100.0m);
00057
00058         // Act
00059         var jsonString = room.ToString();
00060
00061         // Assert
00062         Assert.Contains($"\"Id\": {room.Id}", jsonString);
00063         Assert.Contains($"\"Type\": \"{room.Type}\"", jsonString);
00064         Assert.Contains($"\"PricePerNight\": {room.PricePerNight}", jsonString);
00065     }

```

```

00066
00071 [Fact]
00072 public void IsAvailable_NoOverlap_ReturnsTrue()
00073 {
00074     // Arrange
00075     var room = new Room(RoomType.Single, 100.0m);
00076     var startDate = new DateTime(2024, 12, 1);
00077     var endDate = new DateTime(2024, 12, 5);
00078     room.AddReservation(startDate, endDate); // Add reservation to this room
00079
00080     // Act
00081     var isAvailable = room.IsAvailable(new DateTime(2024, 12, 6), new DateTime(2024, 12, 10));
00082
00083     // Assert
00084     Assert.True(isAvailable);
00085 }
00086
00091 [Fact]
00092 public void IsAvailable_Overlap_ReturnsFalse()
00093 {
00094     // Arrange
00095     var room = new Room(RoomType.Single, 100.0m);
00096     var existingStartDate = new DateTime(2024, 12, 1);
00097     var existingEndDate = new DateTime(2024, 12, 5);
00098     room.AddReservation(existingStartDate, existingEndDate); // Add reservation to this room
00099
00100     // Act
00101     var isAvailable = room.IsAvailable(new DateTime(2024, 12, 4), new DateTime(2024, 12, 6));
00102
00103     // Assert
00104     Assert.False(isAvailable);
00105 }
00106
00110 [Fact]
00111 public void AddReservation_ValidDates_AddsReservation()
00112 {
00113     // Arrange
00114     var room = new Room(RoomType.Single, 100.0m);
00115     var startDate = new DateTime(2024, 12, 1);
00116     var endDate = new DateTime(2024, 12, 5);
00117
00118     // Act
00119     var result = room.AddReservation(startDate, endDate);
00120
00121     // Assert
00122     Assert.True(result);
00123     Assert.Single(room.ReservationDates);
00124 }
00125
00129 [Fact]
00130 public void RemoveReservation_ExistingReservation_RemovesReservation()
00131 {
00132     // Arrange
00133     var room = new Room(RoomType.Single, 100.0m);
00134     var startDate = new DateTime(2024, 12, 1);
00135     var endDate = new DateTime(2024, 12, 5);
00136     room.AddReservation(startDate, endDate); // Add reservation to this room
00137
00138     // Act
00139     var result = room.RemoveReservation(startDate, endDate);
00140
00141     // Assert
00142     Assert.True(result);
00143     Assert.Empty(room.ReservationDates);
00144 }
00145
00150 [Fact]
00151 public void CalculateTotalCost_ValidDates_ReturnsCorrectCost()
00152 {
00153     // Arrange
00154     var room = new Room(RoomType.Single, 100.0m);
00155     var startDate = new DateTime(2024, 12, 1);
00156     var endDate = new DateTime(2024, 12, 5);
00157
00158     // Act
00159     var totalCost = room.CalculateTotalCost(startDate, endDate);
00160
00161     // Assert
00162     Assert.Equal(400.0m, totalCost); // 4 nights at 100.0m per night
00163 }
00164
00169 [Fact]
00170 public void CalculateTotalCost_EndDateBeforeStartDate_ThrowsArgumentException()
00171 {
00172     // Arrange
00173     var room = new Room(RoomType.Single, 100.0m);
00174     var startDate = new DateTime(2024, 12, 5);

```

```

00175         var endDate = new DateTime(2024, 12, 1);
00176
00177         // Act & Assert
00178         Assert.Throws<ArgumentException>(() => room.CalculateTotalCost(startDate, endDate));
00179     }
00180 }
00181 }

```

6.13 SmartStay.Core.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs File Reference

6.14 SmartStay.Core.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs

[Go to the documentation of this file.](#)

```

00001 // <autogenerated />
00002 using System;
00003 using System.Reflection;
00004 [assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v8.0",
    FrameworkDisplayName = ".NET 8.0")]

```

6.15 SmartStay.IO.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs File Reference

6.16 SmartStay.IO.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs

[Go to the documentation of this file.](#)

```

00001 // <autogenerated />
00002 using System;
00003 using System.Reflection;
00004 [assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v8.0",
    FrameworkDisplayName = ".NET 8.0")]

```

6.17 SmartStay.Validation.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs File Reference

6.18 SmartStay.Validation.Tests/obj/Debug/net8.0/.NETCoreApp,Version=v8.0.AssemblyAttributes.cs

[Go to the documentation of this file.](#)

```

00001 // <autogenerated />
00002 using System;
00003 using System.Reflection;
00004 [assembly: global::System.Runtime.Versioning.TargetFrameworkAttribute(".NETCoreApp,Version=v8.0",
    FrameworkDisplayName = ".NET 8.0")]

```

6.19 SmartStay.Core.Tests.AssemblyInfo.cs File Reference

6.20 SmartStay.Core.Tests.AssemblyInfo.cs

[Go to the documentation of this file.](#)

```

00001 //-----
00002 // <auto-generated>
00003 //     This code was generated by a tool.
00004 //     Runtime Version:4.0.30319.42000
00005 //
00006 //     Changes to this file may cause incorrect behavior and will be lost if
00007 //     the code is regenerated.
00008 // </auto-generated>
00009 //-----
00010
00011 using System;
00012 using System.Reflection;
00013
00014 [assembly: System.Reflection.AssemblyCompanyAttribute("SmartStay.Core.Tests")]
00015 [assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]
00016 [assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
00017 [assembly:
    System.Reflection.AssemblyInformationalVersionAttribute("1.0.0+c366ac03947932e5126b804e73253b4d5f5e0e8d")]
00018 [assembly: System.Reflection.AssemblyProductAttribute("SmartStay.Core.Tests")]
00019 [assembly: System.Reflection.AssemblyTitleAttribute("SmartStay.Core.Tests")]
00020 [assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
00021
00022 // Generated by the MSBuild WriteCodeFragment class.
00023

```

6.21 SmartStay.Core.Tests.GlobalUsings.g.cs File Reference

6.22 SmartStay.Core.Tests.GlobalUsings.g.cs

[Go to the documentation of this file.](#)

```

00001 // <auto-generated/>
00002 global using global::System;
00003 global using global::System.Collections.Generic;
00004 global using global::System.IO;
00005 global using global::System.Linq;
00006 global using global::System.Net.Http;
00007 global using global::System.Threading;
00008 global using global::System.Threading.Tasks;
00009 global using global::Xunit;

```

6.23 AccommodationsTests.cs File Reference

Data Structures

- class [SmartStay.Core.Tests.Repositories.AccommodationsTests](#)
Contains unit tests for the Accommodations repository class. Tests include adding, removing, importing, exporting accommodations, and serialization/deserialization processes.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Core](#)
- namespace [SmartStay.Core.Tests](#)
- namespace [SmartStay.Core.Tests.Repositories](#)

The [SmartStay.Core.Tests.Repositories](#) namespace contains unit tests for the repository classes that interact with the application data.

6.24 AccommodationsTests.cs

[Go to the documentation of this file.](#)

```

00001
00011
00016 namespace SmartStay.Core.Tests.Repositories
00017 {
00018     using System.Globalization;
00019     using SmartStay.Common.Enums;
00020     using SmartStay.Core.Models;
00021     using SmartStay.Core.Repositories;
00022     using SmartStay.Core.Utilities;
00023     using Xunit;
00024
00029 public class AccommodationsTests
00030 {
00035     [Fact]
00036     public void Add_ValidAccommodation_AddsAccommodationSuccessfully()
00037     {
00038         // Arrange
00039         var accommodationRepo = new Accommodations();
00040         var accommodation = new Accommodation(1, AccommodationType.Hotel, "Grand Hotel", "456 Luxury
Ave");
00041
00042         // Act
00043         var result = accommodationRepo.Add(accommodation);
00044
00045         // Assert
00046         Assert.True(result);
00047         Assert.Equal(1, accommodationRepo.CountAccommodations());
00048     }
00049
00054     [Fact]
00055     public void Remove_ValidAccommodation_RemovesAccommodationSuccessfully()
00056     {
00057         // Arrange
00058         var accommodationRepo = new Accommodations();
00059         var accommodation = new Accommodation(1, AccommodationType.Hotel, "Grand Hotel", "456 Luxury
Ave");
00060         accommodationRepo.Add(accommodation);
00061
00062         // Act
00063         var result = accommodationRepo.Remove(accommodation);
00064
00065         // Assert
00066         Assert.True(result);
00067         Assert.Equal(0, accommodationRepo.CountAccommodations()); // No accommodations should remain
00068     }
00069
00074     [Fact]
00075     public void Remove_NonExistingAccommodation_ReturnsFalse()
00076     {
00077         // Arrange
00078         var accommodationRepo = new Accommodations();
00079         var accommodation = new Accommodation(1, AccommodationType.Hotel, "Grand Hotel", "456 Luxury
Ave");
00080
00081         // Act
00082         var result = accommodationRepo.Remove(accommodation);
00083
00084         // Assert
00085         Assert.False(result); // Accommodation does not exist, should return false
00086     }
00087
00092     [Fact]
00093     public void Import_ValidData_ImportsAccommodations()
00094     {
00095         // Arrange
00096         var accommodationRepo = new Accommodations();
00097
00098         // Example JSON data representing multiple accommodations with rooms
00099         var jsonData = @"
00100         [
00101             {
00102                 "Id": 1,
00103                 "OwnerId": 1,
00104                 "Type": 1,
00105                 "Name": "Grand Hotel",
00106                 "Address": "456 Luxury Ave",
00107                 "Rooms": [
00108                     {
00109                         "Id": 101,
00110                         "Type": 1,
00111                         "PricePerNight": 250.0,
00112                         "ReservationDates": [

```

```

00113         {
00114             "Start": "2024-12-10T14:00:00",
00115             "End": "2024-12-15T11:00:00"
00116         }
00117     ]
00118 }
00119 ]
00120 }
00121 ];
00122 // Act
00123 var result = accommodationRepo.Import(jsonData);
00124
00125 // Assert: Verify the import counts
00126 Assert.Equal(1, result.ImportedCount); // Only 1 accommodation is imported
00127 Assert.Equal(0, result.ReplacedCount); // No existing accommodations should be replaced
00128
00129 // Assert: Verify the number of accommodations in the repository
00130 Assert.Equal(1, accommodationRepo.CountAccommodations());
00131
00132 // Assert: Verify the details of the imported accommodation
00133 var importedAccommodation = accommodationRepo.FindAccommodationById(1);
00134 Assert.NotNull(importedAccommodation);
00135 Assert.Equal(1, importedAccommodation.Id);
00136 Assert.Equal(1, importedAccommodation.OwnerId);
00137 Assert.Equal(AccommodationType.Hotel, importedAccommodation.Type);
00138 Assert.Equal("Grand Hotel", importedAccommodation.Name);
00139 Assert.Equal("456 Luxury Ave", importedAccommodation.Address);
00140
00141 // Assert: Verify that rooms are properly imported
00142 Assert.Single(importedAccommodation.Rooms); // Only 1 room
00143 var importedRoom = importedAccommodation.Rooms[0];
00144 Assert.Equal(101, importedRoom.Id);
00145 Assert.Equal(RoomType.Single, importedRoom.Type);
00146 Assert.Equal(250.00m, importedRoom.PricePerNight);
00147
00148 // Assert: Verify ReservationDates for the room
00149 Assert.Single(importedRoom.ReservationDates); // Only 1 reservation date for the room
00150 var reservationDate = importedRoom.ReservationDates.First();
00151 var format = "yyyy-MM-ddTHH:mm:ss"; // Define the expected format
00152 var culture = CultureInfo.InvariantCulture; // Use a culture-insensitive format provider
00153 Assert.Equal(DateTime.ParseExact("2024-12-10T14:00:00", format, culture),
00154 reservationDate.Start);
00155 Assert.Equal(DateTime.ParseExact("2024-12-15T11:00:00", format, culture),
00156 reservationDate.End);
00157
00158 // Verify the total number of rooms
00159 Assert.Single(importedAccommodation.Rooms);
00160 }
00161 [Fact]
00162 public void Export_ValidData_ExportsAccommodations()
00163 {
00164     // Arrange
00165     var accommodationRepo = new Accommodations();
00166
00167     // Create DateRange objects for the rooms' reservation dates
00168     var format = "yyyy-MM-ddTHH:mm:ss"; // Define the expected format
00169     var culture = CultureInfo.InvariantCulture; // Use a culture-insensitive format provider
00170
00171     var reservationDate1 = new DateRange(DateTime.ParseExact("2024-12-10T14:00:00", format,
00172 culture),
00173                                     DateTime.ParseExact("2024-12-15T11:00:00", format,
00174 culture));
00175
00176     var reservationDate2 = new DateRange(DateTime.ParseExact("2024-12-20T14:00:00", format,
00177 culture),
00178                                     DateTime.ParseExact("2024-12-25T11:00:00", format,
00179 culture));
00180
00181     // Create rooms and add reservation dates as SortedSet<DateRange>
00182     var room1 = new Room(101, RoomType.Single, 250.0m, [reservationDate1]);
00183     var room2 = new Room(201, RoomType.Double, 350.0m, [reservationDate2]);
00184
00185     // Create accommodations and add rooms
00186     var accommodation1 = new Accommodation(1, 1, AccommodationType.Hotel, "Grand Hotel", "456
Luxury Ave", [room1]);
00187     var accommodation2 =
00188         new Accommodation(2, 2, AccommodationType.Apartment, "Luxury Suites", "789 Elite St",
[room2]);
00189
00190     accommodationRepo.Add(accommodation1);
00191     accommodationRepo.Add(accommodation2);
00192
00193     // Act
00194     var jsonData = accommodationRepo.Export();

```

```

00195         // Assert: Verify that all accommodations are exported
00196         Assert.Contains("Grand Hotel", jsonData);
00197         Assert.Contains("456 Luxury Ave", jsonData);
00198         Assert.Contains("Luxury Suites", jsonData);
00199         Assert.Contains("789 Elite St", jsonData);
00200
00201         // Assert: Verify that all fields in accommodation1 are exported
00202         Assert.Contains("\"Id\": 1", jsonData);
00203         Assert.Contains("\"OwnerId\": 1", jsonData);
00204         Assert.Contains("\"Type\": 1", jsonData); // Ensure the type is properly serialized
00205         Assert.Contains("\"Name\": \"Grand Hotel\"", jsonData);
00206         Assert.Contains("\"Address\": \"456 Luxury Ave\"", jsonData);
00207
00208         // Assert: Verify that rooms and reservation dates for accommodation1 are exported
00209         Assert.Contains("\"Rooms\":", jsonData);
00210         Assert.Contains("\"Id\": 101", jsonData); // Room ID
00211         Assert.Contains("\"Type\": 1", jsonData); // Room type (Single)
00212         Assert.Contains("\"PricePerNight\": 250.0", jsonData); // Room price
00213         Assert.Contains("\"ReservationDates\":", jsonData); // Reservation dates array
00214
00215         // Assert: Verify that the reservation dates are correctly serialized and sorted (for
accommodation1)
00216         Assert.Contains("\"Start\": \"2024-12-10T14:00:00\"", jsonData);
00217         Assert.Contains("\"End\": \"2024-12-15T11:00:00\"", jsonData);
00218
00219         // Assert: Verify that accommodation2 data is exported correctly
00220         Assert.Contains("\"Id\": 2", jsonData);
00221         Assert.Contains("\"OwnerId\": 2", jsonData);
00222         Assert.Contains("\"Type\": 2", jsonData); // Ensure the type is properly serialized
00223         Assert.Contains("\"Name\": \"Luxury Suites\"", jsonData);
00224         Assert.Contains("\"Address\": \"789 Elite St\"", jsonData);
00225
00226         // Assert: Verify that rooms and reservation dates for accommodation2 are exported
00227         Assert.Contains("\"Id\": 201", jsonData); // Room ID for accommodation2
00228         Assert.Contains("\"Type\": 2", jsonData); // Room type (Double)
00229         Assert.Contains("\"PricePerNight\": 350.0", jsonData); // Room price
00230         Assert.Contains("\"ReservationDates\":", jsonData); // Reservation dates array
00231
00232         // Assert: Verify that the reservation dates for accommodation2 are correctly serialized and
sorted
00233         Assert.Contains("\"Start\": \"2024-12-20T14:00:00\"", jsonData);
00234         Assert.Contains("\"End\": \"2024-12-25T11:00:00\"", jsonData);
00235
00236         // Assert: Verify that the JSON data contains the correct number of accommodations
00237         Assert.Contains("[", jsonData); // Should be an array of accommodations
00238         Assert.Contains("]", jsonData); // Should be an array of accommodations
00239     }
00240
00241     [Fact]
00242     public void FindAccommodationById_ExistingId_ReturnsAccommodation()
00243     {
00244         // Arrange
00245         var accommodationRepo = new Accommodations();
00246         var accommodation = new Accommodation(1, AccommodationType.Hotel, "Grand Hotel", "456 Luxury
Ave");
00247         accommodationRepo.Add(accommodation);
00248
00249         // Act
00250         var foundAccommodation = accommodationRepo.FindAccommodationById(accommodation.Id);
00251
00252         // Assert
00253         Assert.NotNull(foundAccommodation);
00254         Assert.Equal("Grand Hotel", foundAccommodation.Name);
00255     }
00256
00257     [Fact]
00258     public void FindAccommodationById_NonExistingId_ReturnsNull()
00259     {
00260         // Arrange
00261         var accommodationRepo = new Accommodations();
00262
00263         // Act
00264         var foundAccommodation = accommodationRepo.FindAccommodationById(1); // ID does not exist
00265
00266         // Assert
00267         Assert.Null(foundAccommodation);
00268     }
00269
00270     [Fact]
00271     public void Save_ValidData_SavesToFile()
00272     {
00273         // Arrange
00274         var accommodationRepo = new Accommodations();
00275         var accommodation = new Accommodation(1, AccommodationType.Hotel, "Grand Hotel", "456 Luxury
Ave");
00276         accommodationRepo.Add(accommodation);
00277         var filePath = "accommodations_test.dat";

```

```

00289
00290     // Act & Assert
00291     var exception = Record.Exception(() => accommodationRepo.Save(filePath));
00292     Assert.Null(exception); // No exceptions should occur during save
00293 }
00294
00299 [Fact]
00300 public void Load_ValidFile_LoadsAccommodations()
00301 {
00302     // Arrange
00303     var accommodationRepo = new Accommodations();
00304     var accommodation = new Accommodation(1, AccommodationType.Hotel, "Grand Hotel", "456 Luxury
Ave");
00305     accommodationRepo.Add(accommodation);
00306     var filePath = "accommodations_test.dat";
00307     accommodationRepo.Save(filePath); // Save before loading
00308
00309     // Act
00310     var newRepo = new Accommodations();
00311     newRepo.Load(filePath);
00312
00313     // Assert
00314     Assert.Equal(1, newRepo.CountAccommodations());
00315 }
00316 }
00317 }

```

6.25 ClientsTests.cs File Reference

Data Structures

- class [SmartStay.Core.Tests.Repositories.ClientsTests](#)

Contains unit tests for the Clients repository class. Tests include adding, removing, importing, exporting clients, and serialization/deserialization processes.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Core](#)
- namespace [SmartStay.Core.Tests](#)
- namespace [SmartStay.Core.Tests.Repositories](#)

The [SmartStay.Core.Tests.Repositories](#) namespace contains unit tests for the repository classes that interact with the application data.

6.26 ClientsTests.cs

[Go to the documentation of this file.](#)

```

00001
00011
00016 namespace SmartStay.Core.Tests.Repositories
00017 {
00018     using SmartStay.Common.Enums;
00019     using SmartStay.Core.Models;
00020     using SmartStay.Core.Repositories;
00021     using SmartStay.Core.Utilities;
00022     using Xunit;
00023
00028 public class ClientsTests
00029 {
00033     [Fact]
00034     public void Add_ValidClient_AddsClientSuccessfully()
00035     {
00036         // Arrange
00037         var clientRepo = new Clients();
00038         var client = new Client("John", "Doe", "johndoe@example.com");
00039

```

```

00040         // Act
00041         var result = clientRepo.Add(client);
00042
00043         // Assert
00044         Assert.True(result);
00045         Assert.Equal(1, clientRepo.CountClients());
00046     }
00047
00051     [Fact]
00052     public void Remove_ValidClient_RemovesClientSuccessfully()
00053     {
00054         // Arrange
00055         var clientRepo = new Clients();
00056         var client = new Client("John", "Doe", "johndoe@example.com");
00057         clientRepo.Add(client);
00058
00059         // Act
00060         var result = clientRepo.Remove(client);
00061
00062         // Assert
00063         Assert.True(result);
00064         Assert.Equal(0, clientRepo.CountClients()); // No clients should remain
00065     }
00066
00071     [Fact]
00072     public void Remove_NonExistingClient_ReturnsFalse()
00073     {
00074         // Arrange
00075         var clientRepo = new Clients();
00076         var client = new Client("John", "Doe", "johndoe@example.com");
00077
00078         // Act
00079         var result = clientRepo.Remove(client);
00080
00081         // Assert
00082         Assert.False(result); // Client does not exist, should return false
00083     }
00084
00089     [Fact]
00090     public void Import_ValidData_ImportsClients()
00091     {
00092         // Arrange
00093         var clientRepo = new Clients();
00094
00095         // Example JSON data representing multiple clients
00096         var jsonData = @"
00097         [
00098             {
00099                 "Id": 1,
00100                 "FirstName": "John",
00101                 "LastName": "Doe",
00102                 "Email": "johndoe@example.com",
00103                 "PhoneNumber": "+351222333444",
00104                 "Address": "123 Main St",
00105                 "PreferredPaymentMethod": 2
00106             }
00107         ]";
00108
00109         // Act
00110         var result = clientRepo.Import(jsonData);
00111
00112         // Assert: Verify the import counts
00113         Assert.Equal(1, result.ImportedCount); // Only 1 client is imported
00114         Assert.Equal(0, result.ReplacedCount); // No existing clients should be replaced
00115
00116         // Assert: Verify the number of clients in the repository
00117         Assert.Equal(1, clientRepo.CountClients());
00118
00119         // Assert: Verify the details of the imported client
00120         var importedClient = clientRepo.FindClientById(1);
00121         Assert.NotNull(importedClient);
00122         Assert.Equal(1, importedClient.Id);
00123         Assert.Equal("John", importedClient.FirstName);
00124         Assert.Equal("Doe", importedClient.LastName);
00125         Assert.Equal("johndoe@example.com", importedClient.Email);
00126         Assert.Equal("+351222333444", importedClient.PhoneNumber);
00127         Assert.Equal("123 Main St", importedClient.Address);
00128         Assert.Equal(PaymentMethod.PayPal, importedClient.PreferredPaymentMethod);
00129     }
00130
00134     [Fact]
00135     public void Export_ValidData_ExportsClients()
00136     {
00137         // Arrange
00138         var clientRepo = new Clients();
00139
00140         // Create clients

```

```

00141         var client1 =
00142             new Client(1, "John", "Doe", "johndoe@example.com", "+351222333444", "123 Main St",
PaymentMethod.PayPal);
00143         var client2 = new Client(2, "Jane", "Smith", "janesmith@example.com", "+351222333444", "456
Oak St",
                                PaymentMethod.PayPal);
00144
00145         clientRepo.Add(client1);
00146         clientRepo.Add(client2);
00147
00148         // Act
00149         var jsonData = clientRepo.Export();
00150
00151         // Assert: Verify that all clients are exported
00152         Assert.Contains("John", jsonData);
00153         Assert.Contains("Doe", jsonData);
00154         Assert.Contains("johndoe@example.com", jsonData);
00155         Assert.Contains("123 Main St", jsonData);
00156
00157         Assert.Contains("Jane", jsonData);
00158         Assert.Contains("Smith", jsonData);
00159         Assert.Contains("janesmith@example.com", jsonData);
00160         Assert.Contains("456 Oak St", jsonData);
00161
00162         // Assert: Verify that all fields in client1 are exported
00163         Assert.Contains("\"Id\": 1", jsonData);
00164         Assert.Contains("\"FirstName\": \"John\"", jsonData);
00165         Assert.Contains("\"LastName\": \"Doe\"", jsonData);
00166         Assert.Contains("\"Email\": \"johndoe@example.com\"", jsonData);
00167         Assert.Contains("\"PhoneNumber\": \"\\u002B351222333444\"", jsonData);
00168         Assert.Contains("\"Address\": \"123 Main St\"", jsonData);
00169         Assert.Contains("\"PreferredPaymentMethod\": 2", jsonData);
00170
00171         // Assert: Verify that all fields in client2 are exported
00172         Assert.Contains("\"Id\": 2", jsonData);
00173         Assert.Contains("\"FirstName\": \"Jane\"", jsonData);
00174         Assert.Contains("\"LastName\": \"Smith\"", jsonData);
00175         Assert.Contains("\"Email\": \"janesmith@example.com\"", jsonData);
00176         Assert.Contains("\"PhoneNumber\": \"\\u002B351222333444\"", jsonData);
00177         Assert.Contains("\"Address\": \"456 Oak St\"", jsonData);
00178         Assert.Contains("\"PreferredPaymentMethod\": 2", jsonData); // Assuming PayPal is serialized
00179     as 2
00180     }
00181
00182     [Fact]
00183     public void FindClientById_ExistingId_ReturnsClient()
00184     {
00185         // Arrange
00186         var clientRepo = new Clients();
00187         var client = new Client("John", "Doe", "johndoe@example.com");
00188         clientRepo.Add(client);
00189
00190         // Act
00191         var foundClient = clientRepo.FindClientById(client.Id);
00192
00193         // Assert
00194         Assert.NotNull(foundClient);
00195         Assert.Equal("John", foundClient.FirstName);
00196         Assert.Equal("Doe", foundClient.LastName);
00197     }
00198
00199     [Fact]
00200     public void FindClientById_NonExistingId_ReturnsNull()
00201     {
00202         // Arrange
00203         var clientRepo = new Clients();
00204
00205         // Act
00206         var foundClient = clientRepo.FindClientById(1); // ID does not exist
00207
00208         // Assert
00209         Assert.Null(foundClient);
00210     }
00211
00212     [Fact]
00213     public void Save_ValidData_SavesToFile()
00214     {
00215         // Arrange
00216         var clientRepo = new Clients();
00217         var client = new Client("John", "Doe", "johndoe@example.com");
00218         clientRepo.Add(client);
00219         var filePath = "clients_test.dat";
00220
00221         // Act & Assert
00222         var exception = Record.Exception(() => clientRepo.Save(filePath));
00223         Assert.Null(exception); // No exceptions should occur during save
00224     }
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235

```

```

00236
00241     [Fact]
00242     public void Load_ValidFile_LoadsClients()
00243     {
00244         // Arrange
00245         var clientRepo = new Clients();
00246         var client = new Client("John", "Doe", "johndoe@example.com");
00247         clientRepo.Add(client);
00248         var filePath = "clients_test.dat";
00249         clientRepo.Save(filePath); // Save before loading
00250
00251         // Act
00252         var newRepo = new Clients();
00253         newRepo.Load(filePath);
00254
00255         // Assert
00256         Assert.Equal(1, newRepo.CountClients());
00257     }
00258 }
00259 }

```

6.27 OwnersTests.cs File Reference

Data Structures

- class [SmartStay.Core.Tests.Repositories.OwnersTests](#)

Contains unit tests for the Owners repository class. Tests include adding, removing, importing, exporting owners, and serialization/deserialization processes.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Core](#)
- namespace [SmartStay.Core.Tests](#)
- namespace [SmartStay.Core.Tests.Repositories](#)

The [SmartStay.Core.Tests.Repositories](#) namespace contains unit tests for the repository classes that interact with the application data.

6.28 OwnersTests.cs

[Go to the documentation of this file.](#)

```

00001
00011
00016 namespace SmartStay.Core.Tests.Repositories
00017 {
00018     using SmartStay.Core.Models;
00019     using SmartStay.Core.Repositories;
00020     using SmartStay.Core.Utilities;
00021     using Xunit;
00022
00027 public class OwnersTests
00028 {
00032     [Fact]
00033     public void Add_ValidOwner_AddsOwnerSuccessfully()
00034     {
00035         // Arrange
00036         var ownerRepo = new Owners();
00037         var owner = new Owner("John", "Doe", "johndoe@example.com");
00038
00039         // Act
00040         var result = ownerRepo.Add(owner);
00041
00042         // Assert
00043         Assert.True(result);
00044         Assert.Equal(1, ownerRepo.CountOwners());
00045     }

```

```

00046
00050 [Fact]
00051 public void Remove_ValidOwner_RemovesOwnerSuccessfully()
00052 {
00053     // Arrange
00054     var ownerRepo = new Owners();
00055     var owner = new Owner("John", "Doe", "johndoe@example.com");
00056     ownerRepo.Add(owner);
00057
00058     // Act
00059     var result = ownerRepo.Remove(owner);
00060
00061     // Assert
00062     Assert.True(result);
00063     Assert.Equal(0, ownerRepo.CountOwners()); // No owners should remain
00064 }
00065
00070 [Fact]
00071 public void Remove_NonExistingOwner_ReturnsFalse()
00072 {
00073     // Arrange
00074     var ownerRepo = new Owners();
00075     var owner = new Owner("John", "Doe", "johndoe@example.com");
00076
00077     // Act
00078     var result = ownerRepo.Remove(owner);
00079
00080     // Assert
00081     Assert.False(result); // Owner does not exist, should return false
00082 }
00083
00088 [Fact]
00089 public void Import_ValidData_ImportsOwners()
00090 {
00091     // Arrange
00092     var ownerRepo = new Owners();
00093
00094     // Example JSON data representing multiple owners
00095     var jsonData = @"
00096     [
00097     {
00098         ""Id"": 1,
00099         ""FirstName"": ""Alice"",
00100         ""LastName"": ""Johnson"",
00101         ""Email"": ""alicejohnson@example.com"",
00102         ""PhoneNumber"": ""+351333444555"",
00103         ""Address"": ""789 Pine St"",
00104         ""AccommodationsOwned"": [
00105             {
00106                 ""Id"": 1,
00107                 ""OwnerId"": 1,
00108                 ""Type"": 1,
00109                 ""Name"": ""Grand Hotel"",
00110                 ""Address"": ""456 Luxury Ave"",
00111                 ""Rooms"": [
00112                     {
00113                         ""Id"": 101,
00114                         ""Type"": 1,
00115                         ""PricePerNight"": 250.0,
00116                         ""ReservationDates"": [
00117                             {
00118                                 ""Start"": ""2024-12-10T14:00:00"",
00119                                 ""End"": ""2024-12-15T11:00:00""
00120                             }
00121                         ]
00122                     }
00123                 ]
00124             }
00125         ]
00126     }
00127 ];
00128
00129     // Act
00130     var result = ownerRepo.Import(jsonData);
00131
00132     // Assert: Verify the import counts
00133     Assert.Equal(1, result.ImportedCount); // Only 1 owner is imported
00134     Assert.Equal(0, result.ReplacedCount); // No existing owners should be replaced
00135
00136     // Assert: Verify the number of owners in the repository
00137     Assert.Equal(1, ownerRepo.CountOwners());
00138
00139     // Assert: Verify the details of the imported owner
00140     var importedOwner = ownerRepo.FindOwnerById(1);
00141     Assert.NotNull(importedOwner);
00142     Assert.Equal(1, importedOwner.Id);
00143     Assert.Equal("Alice", importedOwner.FirstName);

```

```

00144         Assert.Equal("Johnson", importedOwner.LastName);
00145         Assert.Equal("alicejohnson@example.com", importedOwner.Email);
00146         Assert.Equal("+351333444555", importedOwner.PhoneNumber);
00147         Assert.Equal("789 Pine St", importedOwner.Address);
00148
00149         // Assert: Verify the accommodations owned
00150         Assert.NotNull(importedOwner.AccommodationsOwned);
00151         Assert.Single(importedOwner.AccommodationsOwned);
00152         Assert.Contains(importedOwner.AccommodationsOwned, a => a.Name == "Grand Hotel");
00153     }
00154
00155     [Fact]
00156     public void Export_ValidData_ExportsOwnersWithAccommodations()
00157     {
00158         // Arrange
00159         var ownerRepo = new Owners();
00160
00161         // Create accommodations with rooms and reservation dates
00162         var accommodation1 = new Accommodation(
00163             id: 1, ownerId: 1, type: Common.Enums.AccommodationType.Hotel, name: "Grand Hotel",
00164             address: "456 Luxury Ave",
00165             rooms: new List<Room> { new Room(
00166                 id: 101, type: Common.Enums.RoomType.Double, pricePerNight: 250.0m,
00167                 reservationDates: new SortedSet<DateRange> { new DateRange(
00168                     start: new DateTime(2024, 12, 10, 14, 0, 0), end: new DateTime(2024, 12, 15, 11,
00169                     0, 0) ) } } });
00170
00171         var owner1 = new Owner(id: 1, firstName: "Alice", lastName: "Johnson", email:
00172             "alicejohnson@example.com",
00173             phoneNumber: "+351333444555", address: "789 Pine St",
00174             accommodationsOwned: new List<Accommodation> { accommodation1 });
00175
00176         ownerRepo.Add(owner1);
00177
00178         // Act
00179         var jsonData = ownerRepo.Export();
00180
00181         // Assert: Verify that all owners are exported
00182         Assert.Contains("Alice", jsonData);
00183         Assert.Contains("Johnson", jsonData);
00184         Assert.Contains("alicejohnson@example.com", jsonData);
00185         Assert.Contains("789 Pine St", jsonData);
00186
00187         // Assert: Verify that the accommodation details are exported
00188         Assert.Contains("\"Id\": 1", jsonData); // Accommodation Id
00189         Assert.Contains("\"OwnerId\": 1", jsonData); // OwnerId of the accommodation
00190         Assert.Contains("\"Type\": 1", jsonData); // Accommodation type
00191         Assert.Contains("\"Name\": \"Grand Hotel\"", jsonData); // Accommodation name
00192         Assert.Contains("\"Address\": \"456 Luxury Ave\"", jsonData); // Accommodation address
00193
00194         // Assert: Verify that rooms inside the accommodation are exported
00195         Assert.Contains("\"Id\": 101", jsonData); // Room Id
00196         Assert.Contains("\"Type\": 1", jsonData); // Room type
00197         Assert.Contains("\"PricePerNight\": 250.0", jsonData); // Room price
00198         Assert.Contains("\"ReservationDates\": \"", jsonData); // ReservationDates key
00199
00200         // Assert: Verify that the reservation dates are exported
00201         Assert.Contains("\"Start\": \"2024-12-10T14:00:00\"", jsonData); // Reservation start date
00202         Assert.Contains("\"End\": \"2024-12-15T11:00:00\"", jsonData); // Reservation end date
00203     }
00204
00205     [Fact]
00206     public void FindOwnerById_ExistingId_ReturnsOwner()
00207     {
00208         // Arrange
00209         var ownerRepo = new Owners();
00210         var owner = new Owner("John", "Doe", "johndoe@example.com");
00211         ownerRepo.Add(owner);
00212
00213         // Act
00214         var foundOwner = ownerRepo.FindOwnerById(owner.Id);
00215
00216         // Assert
00217         Assert.NotNull(foundOwner);
00218         Assert.Equal("John", foundOwner.FirstName);
00219         Assert.Equal("Doe", foundOwner.LastName);
00220     }
00221
00222     [Fact]
00223     public void FindOwnerById_NonExistingId_ReturnsNull()
00224     {
00225         // Arrange
00226         var ownerRepo = new Owners();
00227
00228         // Act
00229         var foundOwner = ownerRepo.FindOwnerById(1); // ID does not exist
00230     }

```

```

00240         // Assert
00241         Assert.Null(foundOwner);
00242     }
00243
00244     [Fact]
00249     public void Save_ValidData_SavesToFile()
00250     {
00251         // Arrange
00252         var ownerRepo = new Owners();
00253         var owner = new Owner("John", "Doe", "johndoe@example.com");
00254         ownerRepo.Add(owner);
00255         var filePath = "owners_test.dat";
00256
00257         // Act & Assert
00258         var exception = Record.Exception(() => ownerRepo.Save(filePath));
00259         Assert.Null(exception); // No exceptions should occur during save
00260     }
00261
00262     [Fact]
00267     public void Load_ValidFile_LoadsOwners()
00268     {
00269         // Arrange
00270         var ownerRepo = new Owners();
00271         var owner = new Owner("John", "Doe", "johndoe@example.com");
00272         ownerRepo.Add(owner);
00273         var filePath = "owners_test.dat";
00274         ownerRepo.Save(filePath); // Save before loading
00275
00276         // Act
00277         var newRepo = new Owners();
00278         newRepo.Load(filePath);
00279
00280         // Assert
00281         Assert.Equal(1, newRepo.CountOwners());
00282     }
00283 }
00284 }

```

6.29 ReservationsTests.cs File Reference

Data Structures

- class [SmartStay.Core.Tests.Repositories.ReservationsTests](#)

Contains unit tests for the Reservations repository class. Tests include adding, removing, importing, exporting reservations, and serialization/deserialization processes.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Core](#)
- namespace [SmartStay.Core.Tests](#)
- namespace [SmartStay.Core.Tests.Repositories](#)

The [SmartStay.Core.Tests.Repositories](#) namespace contains unit tests for the repository classes that interact with the application data.

6.30 ReservationsTests.cs

[Go to the documentation of this file.](#)

```

00001
00011
00016 namespace SmartStay.Core.Tests.Repositories
00017 {
00018     using SmartStay.Core.Models;
00019     using SmartStay.Core.Repositories;
00020     using Xunit;
00021     using System;

```

```

00022 using SmartStay.Common.Enums;
00023 using SmartStay.Validation;
00024
00029 public class ReservationsTests
00030 {
00034     [Fact]
00035     public void Add_ValidReservation_AddsReservationSuccessfully()
00036     {
00037         // Arrange
00038         var reservationsRepo = new Reservations();
00039         var reservation =
00040             new Reservation(1, 1, 101, AccommodationType.Hotel, DateTime.Now.AddDays(1),
00041                 DateTime.Now.AddDays(2), 100m);
00042         // Act
00043         var result = reservationsRepo.Add(reservation);
00044
00045         // Assert
00046         Assert.True(result);
00047         Assert.Equal(1, reservationsRepo.CountReservations());
00048     }
00049
00054     [Fact]
00055     public void Add_NullReservation_ThrowsArgumentNullException()
00056     {
00057         // Arrange
00058         var reservationsRepo = new Reservations();
00059
00060         // Act & Assert
00061         var exception = Assert.Throws<ArgumentNullException>(() => reservationsRepo.Add(null!));
00062         Assert.Equal("Reservation cannot be null (Parameter 'reservation')", exception.Message);
00063     }
00064
00069     [Fact]
00070     public void Remove_ValidReservation_RemovesReservationSuccessfully()
00071     {
00072         // Arrange
00073         var reservationsRepo = new Reservations();
00074         var reservation =
00075             new Reservation(1, 1, 101, AccommodationType.Hotel, DateTime.Now.AddDays(1),
00076                 DateTime.Now.AddDays(2), 100m);
00077         reservationsRepo.Add(reservation);
00078
00079         // Act
00080         var result = reservationsRepo.Remove(reservation);
00081
00082         // Assert
00083         Assert.True(result);
00084         Assert.Equal(0, reservationsRepo.CountReservations()); // No reservations should remain
00085     }
00086
00090     [Fact]
00091     public void Remove_NonExistingReservation_ReturnsFalse()
00092     {
00093         // Arrange
00094         var reservationsRepo = new Reservations();
00095         var reservation =
00096             new Reservation(1, 1, 101, AccommodationType.Hotel, DateTime.Now.AddDays(1),
00097                 DateTime.Now.AddDays(2), 100m);
00098
00099         // Act
00100         var result = reservationsRepo.Remove(reservation);
00101
00102         // Assert
00103         Assert.False(result); // Reservation does not exist, should return false
00104     }
00105
00109     [Fact]
00110     public void Import_ValidData_ImportsReservationsWithPayments()
00111     {
00112         // Arrange
00113         var reservationRepo = new Reservations();
00114
00115         // Example JSON data representing multiple reservations with payments
00116         var jsonData = @"
00117         [
00118             {
00119                 "Id": 1,
00120                 "ClientId": 101,
00121                 "AccommodationId": 202,
00122                 "RoomId": 303,
00123                 "AccommodationType": 1,
00124                 "CheckInDate": "2024-12-10T14:00:00",
00125                 "CheckOutDate": "2024-12-15T11:00:00",
00126                 "Status": 1,
00127                 "TotalCost": 1250.00,
00128                 "AmountPaid": 500.00,

```

```

00129         "Payments": [
00130             {
00131                 "Id": 1,
00132                 "ReservationId": 1,
00133                 "Amount": 500.00,
00134                 "Date": "2024-12-01T10:00:00",
00135                 "Method": 2,
00136                 "Status": 2
00137             }
00138         ]
00139     };
00140 ];
00141
00142 // Act
00143 var result = reservationRepo.Import(jsonData);
00144
00145 // Assert: Verify the import counts
00146 Assert.Equal(1, result.ImportedCount); // Only 1 reservation is imported
00147 Assert.Equal(0, result.ReplacedCount); // No existing reservations should be replaced
00148
00149 // Assert: Verify the number of reservations in the repository
00150 Assert.Equal(1, reservationRepo.CountReservations());
00151
00152 // Assert: Verify the details of the imported reservation
00153 var importedReservation = reservationRepo.FindReservationById(1);
00154 Assert.NotNull(importedReservation);
00155 Assert.Equal(1, importedReservation.Id);
00156 Assert.Equal(101, importedReservation.ClientId);
00157 Assert.Equal(202, importedReservation.AccommodationId);
00158 Assert.Equal(303, importedReservation.RoomId);
00159 Assert.Equal(AccommodationType.Hotel, importedReservation.AccommodationType); // Assuming 1
maps to "Hotel"
00160 Assert.Equal(new DateTime(2024, 12, 10, 14, 0, 0), importedReservation.CheckInDate);
00161 Assert.Equal(new DateTime(2024, 12, 15, 11, 0, 0), importedReservation.CheckOutDate);
00162 Assert.Equal(ReservationStatus.CheckedIn, importedReservation.Status);
00163 Assert.Equal(1250.00m, importedReservation.TotalCost);
00164 Assert.Equal(500.00m, importedReservation.AmountPaid);
00165
00166 // Assert: Verify the payment details
00167 Assert.NotNull(importedReservation.Payments);
00168 Assert.Single(importedReservation.Payments);
00169 var payment = importedReservation.Payments.First();
00170 Assert.Equal(1, payment.Id);
00171 Assert.Equal(1, payment.ReservationId);
00172 Assert.Equal(500.00m, payment.Amount);
00173 Assert.Equal(new DateTime(2024, 12, 1, 10, 0, 0), payment.Date);
00174 Assert.Equal(PaymentMethod.PayPal, payment.Method);
00175 Assert.Equal(PaymentStatus.Completed, payment.Status);
00176 }
00177
00182 [Fact]
00183 public void Export_ValidData_ExportsReservationsWithPayments()
00184 {
00185     // Arrange
00186     var reservationRepo = new Reservations();
00187
00188     // Create reservation with a payment
00189     var payment1 = new Payment(1, 500.0m, new DateTime(2024, 12, 1, 10, 0, 0),
PaymentMethod.BankTransfer,
00190         PaymentStatus.Completed);
00191
00192     var reservation1 = new Reservation(
00193         id: 1, clientId: 101, accommodationId: 202, roomId: 303, accommodationType:
AccommodationType.Hotel,
00194         checkInDate: new DateTime(2024, 12, 10, 14, 0, 0), checkOutDate: new DateTime(2024, 12,
15, 11, 0, 0),
00195         status: ReservationStatus.CheckedIn, totalCost: 1250.0m, amountPaid: 500.0m,
payments: new List<Payment> { payment1 });
00196
00197     reservationRepo.Add(reservation1);
00198
00199     // Act
00200     var jsonData = reservationRepo.Export();
00201
00202     // Assert: Verify that all reservation fields are exported
00203     Assert.Contains("\"Id\": 1", jsonData);
00204     Assert.Contains("\"ClientId\": 101", jsonData);
00205     Assert.Contains("\"AccommodationId\": 202", jsonData);
00206     Assert.Contains("\"RoomId\": 303", jsonData);
00207     Assert.Contains("\"AccommodationType\": 1", jsonData); // Assuming 1 is "Hotel"
00208     Assert.Contains("\"CheckInDate\": \"2024-12-10T14:00:00\"", jsonData);
00209     Assert.Contains("\"CheckOutDate\": \"2024-12-15T11:00:00\"", jsonData);
00210     Assert.Contains("\"Status\": 2", jsonData); // Assuming 2 is "CheckedIn"
00211     Assert.Contains("\"TotalCost\": 1250.0", jsonData);
00212     Assert.Contains("\"AmountPaid\": 500.0", jsonData);
00213
00214     // Assert: Verify that the payments are exported correctly
00215

```

```

00216         Assert.Contains("\"Payments\":", jsonData);
00217         Assert.Contains("\"Id\": 1", jsonData);
00218         Assert.Contains("\"ReservationId\": 1", jsonData);
00219         Assert.Contains("\"Amount\": 500.0", jsonData);
00220         Assert.Contains("\"Date\": \"2024-12-01T10:00:00\"", jsonData);
00221         Assert.Contains("\"Method\": 4", jsonData);
00222         Assert.Contains("\"Status\": 2", jsonData);
00223     }
00224
00229     [Fact]
00230     public void FindReservationById_ExistingId_ReturnsReservation()
00231     {
00232         // Arrange
00233         var reservationsRepo = new Reservations();
00234         var reservation =
00235             new Reservation(1, 1, 101, AccommodationType.Hotel, DateTime.Now.AddDays(1),
00236                 DateTime.Now.AddDays(2), 100m);
00237         reservationsRepo.Add(reservation);
00238
00239         // Act
00240         var foundReservation = reservationsRepo.FindReservationById(reservation.Id);
00241
00242         // Assert
00243         Assert.NotNull(foundReservation);
00244         Assert.Equal(1, foundReservation.ClientId);
00245         Assert.Equal(1, foundReservation.AccommodationId);
00246     }
00251     [Fact]
00252     public void FindReservationById_NonExistingId_ReturnsNull()
00253     {
00254         // Arrange
00255         var reservationsRepo = new Reservations();
00256
00257         // Act
00258         var foundReservation = reservationsRepo.FindReservationById(1); // ID does not exist
00259
00260         // Assert
00261         Assert.Null(foundReservation);
00262     }
00263
00268     [Fact]
00269     public void Save_ValidData_SavesToFile()
00270     {
00271         // Arrange
00272         var reservationsRepo = new Reservations();
00273         var reservation =
00274             new Reservation(1, 1, 101, AccommodationType.Hotel, DateTime.Now.AddDays(1),
00275                 DateTime.Now.AddDays(2), 100m);
00276         reservationsRepo.Add(reservation);
00277         var filePath = "reservations_test.dat";
00278
00279         // Act & Assert
00280         var exception = Record.Exception(() => reservationsRepo.Save(filePath));
00281         Assert.Null(exception); // No exceptions should occur during save
00282     }
00287     [Fact]
00288     public void Load_ValidFile_LoadsReservations()
00289     {
00290         // Arrange
00291         var reservationsRepo = new Reservations();
00292         var reservation =
00293             new Reservation(1, 1, 101, AccommodationType.Hotel, DateTime.Now.AddDays(1),
00294                 DateTime.Now.AddDays(2), 100m);
00295         reservationsRepo.Add(reservation);
00296         var filePath = "reservations_test.dat";
00297         reservationsRepo.Save(filePath); // Save before loading
00298
00299         // Act
00300         var newRepo = new Reservations();
00301         newRepo.Load(filePath);
00302
00303         // Assert
00304         Assert.Equal(1, newRepo.CountReservations());
00305     }
00310     [Fact]
00311     public void Add_InvalidTotalCost_ThrowsValidationException()
00312     {
00313         // Arrange & Act & Assert
00314         var exception = Assert.Throws<ValidationException>(() => new Reservation(1, 1, 101,
00315             AccommodationType.Hotel,
00316             DateTime.Now.AddDays(1),
00317             DateTime.Now.AddDays(2), -100m));

```

```

00317         Assert.Equal(ValidationErrorCode.InvalidTotalCost, exception.ErrorCode);
00318     }
00319
00324     [Fact]
00325     public void Add_InvalidDateRange_ThrowsValidationException()
00326     {
00327         // Arrange & Act & Assert
00328         var exception = Assert.Throws<ValidationException>(() => new Reservation(1, 1, 101,
00329             AccommodationType.Hotel,
00329             DateTime.Now.AddDays(3),
00330             DateTime.Now.AddDays(2), 100m));
00331         Assert.Equal(ValidationErrorCode.InvalidDateRange, exception.ErrorCode);
00332     }
00333 }
00334 }

```

6.31 BookingManagerTests.cs File Reference

Data Structures

- class [SmartStay.Core.Tests.Services.BookingManagerTests](#)

Contains unit tests for the BookingManager class. Tests the BookingManager.SaveAll method to ensure that it creates the necessary files when saving repositories.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Core](#)
- namespace [SmartStay.Core.Tests](#)
- namespace [SmartStay.Core.Tests.Services](#)

The [SmartStay.Core.Tests.Services](#) namespace contains unit tests for the services used in the [SmartStay](#) application. These tests verify the correct behavior of the BookingManager class and its methods.

6.32 BookingManagerTests.cs

[Go to the documentation of this file.](#)

```

00001
00010
00015 namespace SmartStay.Core.Tests.Services
00016 {
00017     using SmartStay.Core.Services;
00018     using Microsoft.Extensions.Logging;
00019     using System;
00020     using System.IO;
00021     using Xunit;
00022     using SmartStay.Common.Exceptions;
00023     using SmartStay.Core.Models;
00024     using SmartStay.Common.Enums;
00025
00031 public class BookingManagerTests
00032 {
00037     [Fact]
00038     public void SaveAll_CreatesFiles_WhenCalled()
00039     {
00040         // Arrange: Set up a temporary folder path for testing
00041         string dataFolder = Path.Combine(Path.GetTempPath(), Guid.NewGuid().ToString());
00042         Directory.CreateDirectory(dataFolder); // Create the temp directory
00043
00044         // Create a real logger using LoggerFactory
00045         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00046         var logger = loggerFactory.CreateLogger<BookingManager>();
00047
00048         // Create an instance of BookingManager with the logger
00049         var bookingManager = new BookingManager(logger);

```

```

00050
00051 // Act: Call SaveAll to create files in the temp directory
00052 bookingManager.SaveAll(dataFolder);
00053
00054 // Assert: Verify that the expected files were created
00055 Assert.True(File.Exists(Path.Combine(dataFolder, "clients.dat")), "Clients file was not
created.");
00056 Assert.True(File.Exists(Path.Combine(dataFolder, "accommodations.dat")),
00057 "Accommodations file was not created.");
00058 Assert.True(File.Exists(Path.Combine(dataFolder, "reservations.dat")), "Reservations file was
not created.");
00059 Assert.True(File.Exists(Path.Combine(dataFolder, "owners.dat")), "Owners file was not
created.");
00060
00061 // Clean up: Delete the temporary directory and its contents after the test
00062 Directory.Delete(dataFolder, true); // Remove the temp folder and files
00063 }
00064
00065 [Fact]
00070 public void CreateBasicClient_CreatesClient_WhenValidInput()
00071 {
00072 // Arrange: Create input data for client creation
00073 string firstName = "John";
00074 string lastName = "Doe";
00075 string email = "john.doe@example.com";
00076
00077 // Create a real logger using LoggerFactory
00078 var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00079 var logger = loggerFactory.CreateLogger<BookingManager>();
00080
00081 // Create an instance of BookingManager with the logger
00082 var bookingManager = new BookingManager(logger);
00083
00084 // Act: Call CreateBasicClient to create a new client
00085 var client = bookingManager.CreateBasicClient(firstName, lastName, email);
00086
00087 // Assert: Verify that the client was created with the expected values
00088 Assert.NotNull(client);
00089 Assert.Equal(firstName, client.FirstName);
00090 Assert.Equal(lastName, client.LastName);
00091 Assert.Equal(email, client.Email);
00092 }
00093
00098 [Fact]
00099 public void CreateBasicClient_ThrowsClientCreationException_WhenValidationFails()
00100 {
00101 // Arrange: Create invalid input data (e.g., invalid email)
00102 string firstName = "John";
00103 string lastName = "Doe";
00104 string email = "invalid-email"; // Invalid email
00105
00106 // Create a real logger using LoggerFactory
00107 var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00108 var logger = loggerFactory.CreateLogger<BookingManager>();
00109
00110 // Create an instance of BookingManager with the logger
00111 var bookingManager = new BookingManager(logger);
00112
00113 // Act & Assert: Verify that creating a client with invalid input throws an exception
00114 var exception =
00115 Assert.Throws<ClientCreationException>(() => bookingManager.CreateBasicClient(firstName,
lastName, email));
00116 Assert.Contains("An error occurred while creating the client", exception.Message);
00117 }
00118
00124 [Fact]
00125 public void CreateCompleteClient_CreatesClient_WhenValidInput()
00126 {
00127 // Arrange: Create input data for complete client creation
00128 string firstName = "Jane";
00129 string lastName = "Doe";
00130 string email = "jane.doe@example.com";
00131 string phoneNumber = "+351222333444";
00132 string address = "123 Main St";
00133
00134 // Create a real logger using LoggerFactory
00135 var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00136 var logger = loggerFactory.CreateLogger<BookingManager>();
00137
00138 // Create an instance of BookingManager with the logger
00139 var bookingManager = new BookingManager(logger);
00140
00141 // Act: Call CreateCompleteClient to create a new client
00142 var client = bookingManager.CreateCompleteClient(firstName, lastName, email, phoneNumber,
address);
00143
00144 // Assert: Verify that the client was created with the expected values

```

```

00145         Assert.NotNull(client);
00146         Assert.Equal(firstName, client.FirstName);
00147         Assert.Equal(lastName, client.LastName);
00148         Assert.Equal(email, client.Email);
00149         Assert.Equal(phoneNumber, client.PhoneNumber);
00150         Assert.Equal(address, client.Address);
00151     }
00152
00153     [Fact]
00154     public void CreateCompleteClient_ThrowsClientCreationException_WhenValidationFails()
00155     {
00156         // Arrange: Create invalid input data (e.g., invalid email)
00157         string firstName = "Jane";
00158         string lastName = "Doe";
00159         string email = "invalid-email"; // Invalid email
00160         string phoneNumber = "+351222333444";
00161         string address = "123 Main St";
00162
00163         // Create a real logger using LoggerFactory
00164         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00165         var logger = loggerFactory.CreateLogger<BookingManager>();
00166
00167         // Create an instance of BookingManager with the logger
00168         var bookingManager = new BookingManager(logger);
00169
00170         // Act & Assert: Verify that creating a client with invalid input throws an exception
00171         var exception = Assert.Throws<ClientCreationException>(
00172             () => bookingManager.CreateCompleteClient(firstName, lastName, email, phoneNumber,
00173                 address));
00174         Assert.Contains("An error occurred while creating the client", exception.Message);
00175     }
00176
00177     [Fact]
00178     public void FindClientById_ReturnsClient_WhenClientExists()
00179     {
00180         // Arrange: Create a client and add them to the system
00181         var firstName = "John";
00182         var lastName = "Doe";
00183         var email = "john.doe@example.com";
00184
00185         // Create a real logger using LoggerFactory
00186         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00187         var logger = loggerFactory.CreateLogger<BookingManager>();
00188
00189         // Create an instance of BookingManager and add the client to the system
00190         var bookingManager = new BookingManager(logger);
00191         var client = new Client(firstName, lastName, email);
00192         bookingManager.Clients.Add(client);
00193
00194         // Act: Call FindClientById to find the client by ID
00195         var foundClient = bookingManager.FindClientById(client.Id);
00196
00197         // Assert: Verify that the correct client is returned
00198         Assert.NotNull(foundClient);
00199         Assert.Equal(client.Id, foundClient.Id);
00200         Assert.Equal(firstName, foundClient.FirstName);
00201         Assert.Equal(lastName, foundClient.LastName);
00202         Assert.Equal(email, foundClient.Email);
00203     }
00204
00205     [Fact]
00206     public void FindClientById_ThrowsArgumentException_WhenClientNotFound()
00207     {
00208         // Arrange: Use a non-existent client ID
00209         var nonExistentClientId = 999;
00210
00211         // Create a real logger using LoggerFactory
00212         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00213         var logger = loggerFactory.CreateLogger<BookingManager>();
00214
00215         // Create an instance of BookingManager
00216         var bookingManager = new BookingManager(logger);
00217
00218         // Act & Assert: Verify that an exception is thrown when no client is found with the given ID
00219         var exception = Assert.Throws<ArgumentException>( () =>
00220             bookingManager.FindClientById(nonExistentClientId));
00221         Assert.Contains($"Client with ID {nonExistentClientId} not found.", exception.Message);
00222     }
00223
00224     [Fact]
00225     public void UpdateClient_UpdatesClient_WhenValidData()
00226     {
00227         // Arrange: Create a client and add them to the system
00228         var firstName = "John";
00229         var lastName = "Doe";
00230         var email = "john.doe@example.com";
00231         var phoneNumber = "+351222333444";

```

```

00245         var address = "123 Main St";
00246         var paymentMethod = PaymentMethod.BankTransfer;
00247
00248         // Create a real logger using LoggerFactory
00249         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00250         var logger = loggerFactory.CreateLogger<BookingManager>();
00251
00252         // Create an instance of BookingManager and add the client to the system
00253         var bookingManager = new BookingManager(logger);
00254         var client = new Client(firstName, lastName, email);
00255         bookingManager.Clients.Add(client); // Assuming AddClient is a method to add clients to the
system
00256
00257         // Act: Call UpdateClient to update the client information
00258         var result =
00259             bookingManager.UpdateClient(client.Id, firstName, lastName, email, phoneNumber, address,
paymentMethod);
00260
00261         // Assert: Verify that the update was successful and the client details were updated
00262         Assert.Equal(UpdateClientResult.Success, result);
00263         var updatedClient = bookingManager.FindClientById(client.Id);
00264         Assert.Equal(firstName, updatedClient.FirstName);
00265         Assert.Equal(lastName, updatedClient.LastName);
00266         Assert.Equal(email, updatedClient.Email);
00267         Assert.Equal(phoneNumber, updatedClient.PhoneNumber);
00268         Assert.Equal(address, updatedClient.Address);
00269         Assert.Equal(paymentMethod, updatedClient.PreferredPaymentMethod);
00270     }
00271
00272     [Fact]
00273     public void UpdateClient_ReturnsClientNotFound_WhenClientDoesNotExist()
00274     {
00275         // Arrange: Use a non-existent client ID
00276         var nonExistentClientId = 999;
00277         var firstName = "Jane";
00278         var lastName = "Doe";
00279         var email = "jane.doe@example.com";
00280
00281         // Create a real logger using LoggerFactory
00282         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00283         var logger = loggerFactory.CreateLogger<BookingManager>();
00284
00285         // Create an instance of BookingManager
00286         var bookingManager = new BookingManager(logger);
00287
00288         // Act: Call UpdateClient with a non-existent client ID
00289         var result = bookingManager.UpdateClient(nonExistentClientId, firstName, lastName, email);
00290
00291         // Assert: Verify that the result is ClientNotFound
00292         Assert.Equal(UpdateClientResult.ClientNotFound, result);
00293     }
00294
00295     [Fact]
00296     public void UpdateClient_ReturnsInvalidFirstName_WhenFirstNameIsInvalid()
00297     {
00298         // Arrange: Create a client and add them to the system
00299         var invalidFirstName = ""; // Invalid first name (empty string)
00300         var lastName = "Doe";
00301         var email = "john.doe@example.com";
00302
00303         // Create a real logger using LoggerFactory
00304         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00305         var logger = loggerFactory.CreateLogger<BookingManager>();
00306
00307         // Create an instance of BookingManager and add the client to the system
00308         var bookingManager = new BookingManager(logger);
00309         var client = new Client("John", lastName, email);
00310         bookingManager.Clients.Add(client); // Assuming AddClient is a method to add clients to the
system
00311
00312         // Act: Call UpdateClient with an invalid first name
00313         var result = bookingManager.UpdateClient(client.Id, invalidFirstName, lastName, email);
00314
00315         // Assert: Verify that the result is InvalidFirstName
00316         Assert.Equal(UpdateClientResult.InvalidFirstName, result);
00317     }
00318
00319     [Fact]
00320     public void UpdateClient_ReturnsInvalidLastName_WhenLastNameIsInvalid()
00321     {
00322         // Arrange: Create a client and add them to the system
00323         var firstName = "John";
00324         var invalidLastName = ""; // Invalid last name (empty string)
00325         var email = "john.doe@example.com";
00326
00327         // Create a real logger using LoggerFactory
00328         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340

```

```

00341         var logger = loggerFactory.CreateLogger<BookingManager>();
00342
00343         // Create an instance of BookingManager and add the client to the system
00344         var bookingManager = new BookingManager(logger);
00345         var client = new Client(firstName, "Doe", email);
00346         bookingManager.Clients.Add(client); // Assuming AddClient is a method to add clients to the
system
00347
00348         // Act: Call UpdateClient with an invalid last name
00349         var result = bookingManager.UpdateClient(client.Id, firstName, invalidLastName, email);
00350
00351         // Assert: Verify that the result is InvalidLastName
00352         Assert.Equal(UpdateClientResult.InvalidLastName, result);
00353     }
00354
00355     [Fact]
00360     public void UpdateClient_ReturnsInvalidEmail_WhenEmailIsInvalid()
00361     {
00362         // Arrange: Create a client and add them to the system
00363         var firstName = "John";
00364         var lastName = "Doe";
00365         var invalidEmail = "invalid-email"; // Invalid email format
00366
00367         // Create a real logger using LoggerFactory
00368         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00369         var logger = loggerFactory.CreateLogger<BookingManager>();
00370
00371         // Create an instance of BookingManager and add the client to the system
00372         var bookingManager = new BookingManager(logger);
00373         var client = new Client(firstName, lastName, "john.doe@example.com");
00374         bookingManager.Clients.Add(client); // Assuming AddClient is a method to add clients to the
system
00375
00376         // Act: Call UpdateClient with an invalid email
00377         UpdateClientResult result = bookingManager.UpdateClient(client.Id, firstName, lastName,
invalidEmail);
00378
00379         // Assert: Verify that the result is InvalidEmail
00380         Assert.Equal(UpdateClientResult.InvalidEmail, result);
00381     }
00382
00383     [Fact]
00388     public void RemoveClient_RemovesClient_WhenClientExists()
00389     {
00390         // Arrange: Create a client and add them to the system
00391         var firstName = "John";
00392         var lastName = "Doe";
00393         var email = "john.doe@example.com";
00394         var client = new Client(firstName, lastName, email);
00395
00396         // Create a real logger using LoggerFactory
00397         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00398         var logger = loggerFactory.CreateLogger<BookingManager>();
00399
00400         // Create an instance of BookingManager and add the client to the system
00401         var bookingManager = new BookingManager(logger);
00402         bookingManager.Clients.Add(client); // Add the client to the system
00403
00404         // Act: Call RemoveClient to remove the client
00405         var result = bookingManager.RemoveClient(client.Id);
00406
00407         // Assert: Verify that the result is true and the client has been removed
00408         Assert.True(result);
00409         var removedClient = bookingManager.Clients.FindClientById(client.Id);
00410         Assert.Null(removedClient); // Ensure the client no longer exists in the system
00411     }
00412
00413     [Fact]
00418     public void RemoveClient_ReturnsFalse_WhenClientDoesNotExist()
00419     {
00420         // Arrange: Use a non-existent client ID
00421         var nonExistentClientId = 999;
00422
00423         // Create a real logger using LoggerFactory
00424         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00425         var logger = loggerFactory.CreateLogger<BookingManager>();
00426
00427         // Create an instance of BookingManager
00428         var bookingManager = new BookingManager(logger);
00429
00430         // Act: Call RemoveClient with a non-existent client ID
00431         var result = bookingManager.RemoveClient(nonExistentClientId);
00432
00433         // Assert: Verify that the result is false
00434         Assert.False(result);
00435     }
00436

```

```

00441     [Fact]
00442     public void CreateBasicOwner_CreatesOwner_WhenValidDataIsProvided()
00443     {
00444         // Arrange: Create an owner with valid data
00445         var firstName = "Alice";
00446         var lastName = "Smith";
00447         var email = "alice.smith@example.com";
00448
00449         // Create a real logger using LoggerFactory
00450         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00451         var logger = loggerFactory.CreateLogger<BookingManager>();
00452
00453         // Create an instance of BookingManager
00454         var bookingManager = new BookingManager(logger);
00455
00456         // Act: Call CreateBasicOwner to create the owner
00457         var owner = bookingManager.CreateBasicOwner(firstName, lastName, email);
00458
00459         // Assert: Verify that the owner is created and added to the system
00460         Assert.NotNull(owner);
00461         Assert.Equal(firstName, owner.FirstName);
00462         Assert.Equal(lastName, owner.LastName);
00463         Assert.Equal(email, owner.Email);
00464     }
00465
00470     [Fact]
00471     public void CreateBasicOwner_ThrowsException_WhenEmailIsInvalid()
00472     {
00473         // Arrange: Create an owner with invalid email
00474         var firstName = "Alice";
00475         var lastName = "Smith";
00476         var invalidEmail = "invalid-email"; // Invalid email format
00477
00478         // Create a real logger using LoggerFactory
00479         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00480         var logger = loggerFactory.CreateLogger<BookingManager>();
00481
00482         // Create an instance of BookingManager
00483         var bookingManager = new BookingManager(logger);
00484
00485         // Act & Assert: Verify that an exception is thrown for invalid email
00486         var exception = Assert.Throws<OwnerCreationException>(
00487             () => bookingManager.CreateBasicOwner(firstName, lastName, invalidEmail));
00488         Assert.Contains("An error occurred while creating the owner due to invalid input",
exception.Message);
00489     }
00490
00495     [Fact]
00496     public void CreateCompleteOwner_CreatesOwner_WhenValidDataIsProvided()
00497     {
00498         // Arrange: Create an owner with full valid data
00499         var firstName = "Bob";
00500         var lastName = "Johnson";
00501         var email = "bob.johnson@example.com";
00502         var phoneNumber = "+351222333444"; // Correct phone number format
00503         var address = "123 Main St, Lisbon";
00504
00505         // Create a real logger using LoggerFactory
00506         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00507         var logger = loggerFactory.CreateLogger<BookingManager>();
00508
00509         // Create an instance of BookingManager
00510         var bookingManager = new BookingManager(logger);
00511
00512         // Act: Call CreateCompleteOwner to create the owner
00513         var owner = bookingManager.CreateCompleteOwner(firstName, lastName, email, phoneNumber,
address);
00514
00515         // Assert: Verify that the owner is created and added to the system
00516         Assert.NotNull(owner);
00517         Assert.Equal(firstName, owner.FirstName);
00518         Assert.Equal(lastName, owner.LastName);
00519         Assert.Equal(email, owner.Email);
00520         Assert.Equal(phoneNumber, owner.PhoneNumber);
00521         Assert.Equal(address, owner.Address);
00522     }
00523
00528     [Fact]
00529     public void CreateCompleteOwner_ThrowsException_WhenPhoneNumberIsInvalid()
00530     {
00531         // Arrange: Create an owner with invalid phone number
00532         var firstName = "Bob";
00533         var lastName = "Johnson";
00534         var email = "bob.johnson@example.com";
00535         var invalidPhoneNumber = "invalid-phone"; // Invalid phone number format
00536         var address = "123 Main St, Lisbon";
00537

```

```

00538         // Create a real logger using LoggerFactory
00539         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00540         var logger = loggerFactory.CreateLogger<BookingManager>();
00541
00542         // Create an instance of BookingManager
00543         var bookingManager = new BookingManager(logger);
00544
00545         // Act & Assert: Verify that an exception is thrown for invalid phone number
00546         var exception = Assert.Throws<OwnerCreationException>(
00547             () => bookingManager.CreateCompleteOwner(firstName, lastName, email, invalidPhoneNumber,
address));
00548         Assert.Contains("An error occurred while creating the owner due to invalid input",
exception.Message);
00549     }
00550
00551     [Fact]
00552     public void FindOwnerById_FindsOwner_WhenOwnerExists()
00553     {
00554         // Arrange: Create an owner and add them to the system
00555         var firstName = "Charlie";
00556         var lastName = "Brown";
00557         var email = "charlie.brown@example.com";
00558         var owner = new Owner(firstName, lastName, email);
00559
00560         // Create a real logger using LoggerFactory
00561         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00562         var logger = loggerFactory.CreateLogger<BookingManager>();
00563
00564         // Create an instance of BookingManager and add the owner to the system
00565         var bookingManager = new BookingManager(logger);
00566         bookingManager.Owners.Add(owner);
00567
00568         // Act: Call FindOwnerById to find the owner
00569         var foundOwner = bookingManager.FindOwnerById(owner.Id);
00570
00571         // Assert: Verify that the correct owner is returned
00572         Assert.NotNull(foundOwner);
00573         Assert.Equal(owner.Id, foundOwner.Id);
00574         Assert.Equal(firstName, foundOwner.FirstName);
00575         Assert.Equal(lastName, foundOwner.LastName);
00576         Assert.Equal(email, foundOwner.Email);
00577     }
00578
00579     [Fact]
00580     public void FindOwnerById_ThrowsException_WhenOwnerDoesNotExist()
00581     {
00582         // Arrange: Use a non-existent owner ID
00583         var nonExistentOwnerId = 999;
00584
00585         // Create a real logger using LoggerFactory
00586         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00587         var logger = loggerFactory.CreateLogger<BookingManager>();
00588
00589         // Create an instance of BookingManager
00590         var bookingManager = new BookingManager(logger);
00591
00592         // Act & Assert: Verify that an exception is thrown when the owner is not found
00593         var exception = Assert.Throws<ArgumentException>(() =>
bookingManager.FindOwnerById(nonExistentOwnerId));
00594         Assert.Contains("Owner with ID", exception.Message);
00595     }
00596
00597     [Fact]
00598     public void UpdateOwner_UpdatesOwner_WhenValidDataIsProvided()
00599     {
00600         // Arrange: Create an owner and add them to the system
00601         var firstName = "David";
00602         var lastName = "Clark";
00603         var email = "david.clark@example.com";
00604         var owner = new Owner(firstName, lastName, email);
00605
00606         // Create a real logger using LoggerFactory
00607         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00608         var logger = loggerFactory.CreateLogger<BookingManager>();
00609
00610         // Create an instance of BookingManager and add the owner to the system
00611         var bookingManager = new BookingManager(logger);
00612         bookingManager.Owners.Add(owner);
00613
00614         // Act: Call UpdateOwner to update the owner's details
00615         var updatedFirstName = "Davidson";
00616         var updatedLastName = "Clarkson";
00617         var updatedEmail = "david.clarkson@example.com";
00618         var result = bookingManager.UpdateOwner(owner.Id, updatedFirstName, updatedLastName,
updatedEmail);
00619
00620         // Assert: Verify that the update was successful

```

```

00633     Assert.Equal(UpdateOwnerResult.Success, result);
00634     var updatedOwner = bookingManager.FindOwnerById(owner.Id);
00635     Assert.Equal(updatedFirstName, updatedOwner.FirstName);
00636     Assert.Equal(updatedLastName, updatedOwner.LastName);
00637     Assert.Equal(updatedEmail, updatedOwner.Email);
00638 }
00639
00640 [Fact]
00641 public void UpdateOwner_ReturnsOwnerNotFound_WhenOwnerDoesNotExist()
00642 {
00643     // Arrange: Use a non-existent owner ID
00644     var nonExistentOwnerId = 999;
00645
00646     // Create a real logger using LoggerFactory
00647     var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00648     var logger = loggerFactory.CreateLogger<BookingManager>();
00649
00650     // Create an instance of BookingManager
00651     var bookingManager = new BookingManager(logger);
00652
00653     // Act: Try to update the non-existent owner
00654     var result =
00655         bookingManager.UpdateOwner(nonExistentOwnerId, "NewFirstName", "NewLastName",
00656             "new.email@example.com");
00657
00658     // Assert: Verify that the result is OwnerNotFound
00659     Assert.Equal(UpdateOwnerResult.OwnerNotFound, result);
00660 }
00661
00662 [Fact]
00663 public void UpdateOwner_ReturnsInvalidFirstName_WhenFirstNameIsValid()
00664 {
00665     // Arrange: Create an owner and add them to the system
00666     var firstName = "Emily";
00667     var lastName = "Taylor";
00668     var email = "emily.taylor@example.com";
00669     var owner = new Owner(firstName, lastName, email);
00670
00671     // Create a real logger using LoggerFactory
00672     var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00673     var logger = loggerFactory.CreateLogger<BookingManager>();
00674
00675     // Create an instance of BookingManager and add the owner to the system
00676     var bookingManager = new BookingManager(logger);
00677     bookingManager.Owners.Add(owner);
00678
00679     // Act: Try to update the owner with an invalid first name (empty string)
00680     var result = bookingManager.UpdateOwner(owner.Id, "", lastName, email);
00681
00682     // Assert: Verify that the result is InvalidFirstName
00683     Assert.Equal(UpdateOwnerResult.InvalidFirstName, result);
00684 }
00685
00686 [Fact]
00687 public void UpdateOwner_ReturnsInvalidEmail_WhenEmailIsValid()
00688 {
00689     // Arrange: Create an owner and add them to the system
00690     var firstName = "Emily";
00691     var lastName = "Taylor";
00692     var email = "emily.taylor@example.com";
00693     var owner = new Owner(firstName, lastName, email);
00694
00695     // Create a real logger using LoggerFactory
00696     var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00697     var logger = loggerFactory.CreateLogger<BookingManager>();
00698
00699     // Create an instance of BookingManager and add the owner to the system
00700     var bookingManager = new BookingManager(logger);
00701     bookingManager.Owners.Add(owner);
00702
00703     // Act: Try to update the owner with an invalid email
00704     var result = bookingManager.UpdateOwner(owner.Id, firstName, lastName, "invalid-email");
00705
00706     // Assert: Verify that the result is InvalidEmail
00707     Assert.Equal(UpdateOwnerResult.InvalidEmail, result);
00708 }
00709
00710 [Fact]
00711 public void RemoveOwner_RemovesOwner_WhenOwnerExists()
00712 {
00713     // Arrange: Create an owner and add them to the system
00714     var firstName = "James";
00715     var lastName = "Smith";
00716     var email = "james.smith@example.com";
00717     var owner = new Owner(firstName, lastName, email);
00718
00719     // Create a real logger using LoggerFactory
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733

```

```

00734         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00735         var logger = loggerFactory.CreateLogger<BookingManager>();
00736
00737         // Create an instance of BookingManager and add the owner to the system
00738         var bookingManager = new BookingManager(logger);
00739         bookingManager.Owners.Add(owner);
00740
00741         // Act: Call RemoveOwner to remove the owner
00742         var result = bookingManager.RemoveOwner(owner.Id);
00743
00744         // Assert: Verify that the result is true and the owner has been removed
00745         Assert.True(result);
00746         var removedOwner = bookingManager.Owners.FindOwnerById(owner.Id);
00747         Assert.Null(removedOwner); // Ensure the owner no longer exists in the system
00748     }
00749
00750     [Fact]
00751     public void RemoveOwner_ReturnsFalse_WhenOwnerDoesNotExist()
00752     {
00753         // Arrange: Use a non-existent owner ID
00754         var nonExistentOwnerId = 999;
00755
00756         // Create a real logger using LoggerFactory
00757         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00758         var logger = loggerFactory.CreateLogger<BookingManager>();
00759
00760         // Create an instance of BookingManager
00761         var bookingManager = new BookingManager(logger);
00762
00763         // Act: Try to remove the non-existent owner
00764         var result = bookingManager.RemoveOwner(nonExistentOwnerId);
00765
00766         // Assert: Verify that the result is false
00767         Assert.False(result);
00768     }
00769
00770     [Fact]
00771     public void CreateAccommodation_CreatesAccommodation_WhenOwnerExists()
00772     {
00773         // Arrange: Create an owner and add them to the system
00774         var firstName = "Alice";
00775         var lastName = "Johnson";
00776         var email = "alice.johnson@example.com";
00777         var owner = new Owner(firstName, lastName, email);
00778
00779         // Create a real logger using LoggerFactory
00780         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00781         var logger = loggerFactory.CreateLogger<BookingManager>();
00782
00783         // Create an instance of BookingManager and add the owner to the system
00784         var bookingManager = new BookingManager(logger);
00785         bookingManager.Owners.Add(owner);
00786
00787         // Define accommodation details
00788         var accommodationName = "Ocean View Villa";
00789         var accommodationType = AccommodationType.Villa;
00790         var accommodationAddress = "123 Beachfront Ave, Seaside, CA";
00791
00792         // Act: Call CreateAccommodation to create an accommodation
00793         var accommodation =
00794             bookingManager.CreateAccommodation(owner.Id, accommodationType, accommodationName,
00795             accommodationAddress);
00796
00797         // Assert: Verify the accommodation is created successfully
00798         Assert.NotNull(accommodation);
00799         Assert.Equal(accommodationName, accommodation.Name);
00800         Assert.Equal(accommodationAddress, accommodation.Address);
00801         Assert.Equal(owner.Id, accommodation.OwnerId);
00802     }
00803
00804     [Fact]
00805     public void CreateAccommodation_ThrowsEntityNotFoundException_WhenOwnerDoesNotExist()
00806     {
00807         // Arrange: Use a non-existent owner ID
00808         var nonExistentOwnerId = 999;
00809         var accommodationName = "Mountain Retreat";
00810         var accommodationType = AccommodationType.Cabin;
00811         var accommodationAddress = "456 Mountain Road, Summit, CA";
00812
00813         // Create a real logger using LoggerFactory
00814         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00815         var logger = loggerFactory.CreateLogger<BookingManager>();
00816
00817         // Create an instance of BookingManager
00818         var bookingManager = new BookingManager(logger);
00819
00820         // Act & Assert: Ensure that calling CreateAccommodation throws EntityNotFoundException
00821     }

```

```

00832         var exception = Assert.Throws<AccommodationCreationException>(
00833             () => bookingManager.CreateAccommodation(nonExistentOwnerId, accommodationType,
accommodationName,
00834                 accommodationAddress));
00835
00836         Assert.Equal("An error occurred while creating the accommodation due to missing owner.",
exception.Message);
00837     }
00838
00843     [Fact]
00844     public void UpdateAccommodation_UpdatesAccommodation_WhenValidDataProvided()
00845     {
00846         // Arrange: Create an accommodation and add it to the system
00847         var ownerId = 1;
00848         var accommodationType = AccommodationType.Apartment;
00849         var accommodationName = "Lake View Apartment";
00850         var accommodationAddress = "123 Lake St, Lakeside, CA";
00851         var accommodation = new Accommodation(ownerId, accommodationType, accommodationName,
accommodationAddress);
00852
00853         // Create a real logger using LoggerFactory
00854         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00855         var logger = loggerFactory.CreateLogger<BookingManager>();
00856
00857         // Create an instance of BookingManager and add the accommodation to the system
00858         var bookingManager = new BookingManager(logger);
00859         bookingManager.Accommodations.Add(accommodation);
00860
00861         // Define new details for updating
00862         var newAccommodationType = AccommodationType.Cabin;
00863         var newAccommodationName = "Mountain Retreat";
00864         var newAccommodationAddress = "456 Mountain Rd, Summit, CO";
00865
00866         // Act: Call UpdateAccommodation to update the accommodation details
00867         var result = bookingManager.UpdateAccommodation(accommodation.Id, newAccommodationType,
newAccommodationName,
00868             newAccommodationAddress);
00869
00870         // Assert: Verify that the result is Success
00871         Assert.Equal(UpdateAccommodationResult.Success, result);
00872
00873         // Assert: Verify that the accommodation details are updated
00874         var updatedAccommodation =
bookingManager.Accommodations.FindAccommodationById(accommodation.Id);
00875         Assert.NotNull(updatedAccommodation);
00876         Assert.Equal(newAccommodationType, updatedAccommodation.Type);
00877         Assert.Equal(newAccommodationName, updatedAccommodation.Name);
00878         Assert.Equal(newAccommodationAddress, updatedAccommodation.Address);
00879     }
00880
00885     [Fact]
00886     public void UpdateAccommodation_ReturnsAccommodationNotFound_WhenAccommodationDoesNotExist()
00887     {
00888         // Arrange: Use a non-existent accommodation ID
00889         var nonExistentAccommodationId = 999;
00890
00891         // Create a real logger using LoggerFactory
00892         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00893         var logger = loggerFactory.CreateLogger<BookingManager>();
00894
00895         // Create an instance of BookingManager
00896         var bookingManager = new BookingManager(logger);
00897
00898         // Act: Call UpdateAccommodation with a non-existent accommodation ID
00899         var result = bookingManager.UpdateAccommodation(nonExistentAccommodationId);
00900
00901         // Assert: Verify that the result is AccommodationNotFound
00902         Assert.Equal(UpdateAccommodationResult.AccommodationNotFound, result);
00903     }
00904
00909     [Fact]
00910     public void UpdateAccommodation_ReturnsInvalidName_WhenInvalidNameProvided()
00911     {
00912         // Arrange: Create an accommodation and add it to the system
00913         var ownerId = 1;
00914         var accommodationType = AccommodationType.Apartment;
00915         var accommodationName = "Lake View Apartment";
00916         var accommodationAddress = "123 Lake St, Lakeside, CA";
00917         var accommodation = new Accommodation(ownerId, accommodationType, accommodationName,
accommodationAddress);
00918
00919         // Create a real logger using LoggerFactory
00920         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00921         var logger = loggerFactory.CreateLogger<BookingManager>();
00922
00923         // Create an instance of BookingManager and add the accommodation to the system
00924         var bookingManager = new BookingManager(logger);

```

```

00925         bookingManager.Accommodations.Add(accommodation);
00926
00927         // Define an invalid name (e.g., an empty string)
00928         var invalidAccommodationName = "";
00929
00930         // Act: Call UpdateAccommodation with an invalid accommodation name
00931         var result = bookingManager.UpdateAccommodation(accommodation.Id, accommodationType,
invalidAccommodationName);
00932
00933         // Assert: Verify that the result is InvalidName
00934         Assert.Equal(UpdateAccommodationResult.InvalidName, result);
00935     }
00936
00937     [Fact]
00942     public void UpdateAccommodation_ReturnsInvalidAddress_WhenInvalidAddressProvided()
00943     {
00944         // Arrange: Create an accommodation and add it to the system
00945         var ownerId = 1;
00946         var accommodationType = AccommodationType.Apartment;
00947         var accommodationName = "Lake View Apartment";
00948         var accommodationAddress = "123 Lake St, Lakeside, CA";
00949         var accommodation = new Accommodation(ownerId, accommodationType, accommodationName,
accommodationAddress);
00950
00951         // Create a real logger using LoggerFactory
00952         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00953         var logger = loggerFactory.CreateLogger<BookingManager>();
00954
00955         // Create an instance of BookingManager and add the accommodation to the system
00956         var bookingManager = new BookingManager(logger);
00957         bookingManager.Accommodations.Add(accommodation);
00958
00959         // Define an invalid address (e.g., an empty string)
00960         var invalidAccommodationAddress = "";
00961
00962         // Act: Call UpdateAccommodation with an invalid accommodation address
00963         var result = bookingManager.UpdateAccommodation(accommodation.Id, accommodationType,
accommodationName,
invalidAccommodationAddress);
00964
00965         // Assert: Verify that the result is InvalidAddress
00966         Assert.Equal(UpdateAccommodationResult.InvalidAddress, result);
00967     }
00968
00969     [Fact]
00975     public void RemoveAccommodation_SuccessfullyRemovesAccommodation_WhenAccommodationAndOwnerExist()
00976     {
00977         // Arrange: Create an accommodation and an owner, then add them to the system
00978         var owner = new Owner("John", "Doe", "john.doe@email.com");
00979
00980         var accommodationType = AccommodationType.Apartment;
00981         var accommodationName = "Lake View Apartment";
00982         var accommodationAddress = "123 Lake St, Lakeside, CA";
00983         var accommodation = new Accommodation(owner.Id, accommodationType, accommodationName,
accommodationAddress);
00984
00985         owner.AddAccommodation(accommodation);
00986
00987         // Create a real logger using LoggerFactory
00988         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
00989         var logger = loggerFactory.CreateLogger<BookingManager>();
00990
00991         // Create an instance of BookingManager and add the accommodation and owner to the system
00992         var bookingManager = new BookingManager(logger);
00993         bookingManager.Accommodations.Add(accommodation);
00994         bookingManager.Owners.Add(owner);
00995
00996         // Act: Call RemoveAccommodation to remove the accommodation
00997         var result = bookingManager.RemoveAccommodation(accommodation.Id);
00998
00999         // Assert: Verify that the result is Success
01000         Assert.Equal(RemoveAccommodationResult.Success, result);
01001
01002         // Assert: Verify that the accommodation is removed from the system
01003         var removedAccommodation =
bookingManager.Accommodations.FindAccommodationById(accommodation.Id);
01004         Assert.Null(removedAccommodation);
01005
01006         // Assert: Verify that the accommodation is removed from the owner's list
01007         var removedOwnerAccommodation = owner.AccommodationsOwned.Find(a => a.Id == accommodation.Id);
01008         Assert.Null(removedOwnerAccommodation);
01009     }
01010
01011     [Fact]
01016     public void RemoveAccommodation_ReturnsAccommodationNotFound_WhenAccommodationDoesNotExist()
01017     {
01018         // Arrange: Create an owner and add them to the system

```

```

01019         var owner = new Owner("John", "Doe", "john.doe@email.com");
01020
01021         // Create a real logger using LoggerFactory
01022         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
01023         var logger = loggerFactory.CreateLogger<BookingManager>();
01024
01025         // Create an instance of BookingManager
01026         var bookingManager = new BookingManager(logger);
01027         bookingManager.Owners.Add(owner);
01028
01029         // Act: Call RemoveAccommodation with a non-existent accommodation ID
01030         var result = bookingManager.RemoveAccommodation(999);
01031
01032         // Assert: Verify that the result is AccommodationNotFound
01033         Assert.Equal(RemoveAccommodationResult.AccommodationNotFound, result);
01034     }
01035
01040     [Fact]
01041     public void RemoveAccommodation_ReturnsAccommodationNotFound_WhenAccommodationCannotBeFound()
01042     {
01043         // Arrange: Create an accommodation and an owner, then add them to the system
01044         var owner = new Owner("John", "Doe", "john.doe@email.com");
01045
01046         var accommodationType = AccommodationType.Apartment;
01047         var accommodationName = "Lake View Apartment";
01048         var accommodationAddress = "123 Lake St, Lakeside, CA";
01049         var accommodation = new Accommodation(owner.Id, accommodationType, accommodationName,
accommodationAddress);
01050
01051         owner.AddAccommodation(accommodation);
01052
01053         // Create a real logger using LoggerFactory
01054         var loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());
01055         var logger = loggerFactory.CreateLogger<BookingManager>();
01056
01057         // Create an instance of BookingManager and add the accommodation and owner to the system
01058         var bookingManager = new BookingManager(logger);
01059         bookingManager.Accommodations.Add(accommodation);
01060         bookingManager.Owners.Add(owner);
01061
01062         // Act: Call RemoveAccommodation to remove accommodation from the system
01063         bookingManager.Accommodations.Remove(accommodation);
01064
01065         // Act: Call RemoveAccommodation again after the accommodation has been removed from the
system
01066         var result = bookingManager.RemoveAccommodation(accommodation.Id);
01067
01068         // Assert: Verify that the result is AccommodationRemovalFailed
01069         Assert.Equal(RemoveAccommodationResult.AccommodationNotFound, result);
01070     }
01071 }
01072 }

```

6.33 FileExtensionsTests.cs File Reference

Data Structures

- class [SmartStay.IO.Tests.Extensions.FileExtensionsTests](#)

Contains unit tests for the FileExtensions class. Tests the behavior of file-related extension methods.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.IO](#)
- namespace [SmartStay.IO.Tests](#)
- namespace [SmartStay.IO.Tests.Extensions](#)

The [SmartStay.IO.Tests.Extensions](#) namespace contains unit tests for the extension methods provided for file-related operations in the [SmartStay](#) application.

6.34 FileExtensionsTests.cs

[Go to the documentation of this file.](#)

```

00001
00010
00015 namespace SmartStay.IO.Tests.Extensions
00016 {
00017     using SmartStay.IO.Extensions;
00018     using System.IO;
00019     using Xunit;
00020
00025 public class FileExtensionsTests
00026 {
00031     [Fact]
00032     public void EnsureDirectoryExists_DirectoryDoesNotExist_CreatesDirectory()
00033     {
00034         // Arrange
00035         var testDirectory = Path.Combine(Path.GetTempPath(), "SmartStay", "NonExistentDir");
00036         var testFilePath = Path.Combine(testDirectory, "testfile.txt");
00037
00038         try
00039         {
00040             // Ensure the directory does not exist before the test
00041             if (Directory.Exists(testDirectory))
00042                 Directory.Delete(testDirectory, true);
00043
00044             // Act
00045             testFilePath.EnsureDirectoryExists();
00046
00047             // Assert
00048             Assert.True(Directory.Exists(testDirectory));
00049         }
00050         finally
00051         {
00052             // Cleanup
00053             if (Directory.Exists(testDirectory))
00054                 Directory.Delete(testDirectory, true);
00055         }
00056     }
00057
00062     [Fact]
00063     public void EnsureDirectoryExists_DirectoryExists_DoesNotThrowException()
00064     {
00065         // Arrange
00066         var testDirectory = Path.Combine(Path.GetTempPath(), "SmartStay", "ExistingDir");
00067         var testFilePath = Path.Combine(testDirectory, "testfile.txt");
00068
00069         try
00070         {
00071             // Ensure the directory exists before the test
00072             Directory.CreateDirectory(testDirectory);
00073
00074             // Act
00075             testFilePath.EnsureDirectoryExists();
00076
00077             // Assert
00078             Assert.True(Directory.Exists(testDirectory));
00079         }
00080         finally
00081         {
00082             // Cleanup
00083             if (Directory.Exists(testDirectory))
00084                 Directory.Delete(testDirectory, true);
00085         }
00086     }
00087
00092     [Theory]
00093     [InlineData("")]
00094     public void EnsureDirectoryExists_NullOrEmptyPath_DoesNotThrowException(string filePath)
00095     {
00096         // Act & Assert
00097         var exception = Record.Exception(() => filePath.EnsureDirectoryExists());
00098         Assert.Null(exception);
00099     }
00100
00105     [Fact]
00106     public void EnsureDirectoryExists_RootDirectoryPath_DoesNotThrowException()
00107     {
00108         // Arrange
00109         var rootDirectory = Path.GetPathRoot(Path.GetTempPath()); // e.g., "C:\\"
00110
00111         // Act & Assert
00112         if (rootDirectory is not null)
00113         {
00114             var exception = Record.Exception(() => rootDirectory.EnsureDirectoryExists());

```

```

00115         Assert.Null(exception);
00116     }
00117     else
00118     {
00119         throw new InvalidOperationException("Root directory path is null, which is unexpected.");
00120     }
00121 }
00122 }
00123 }

```

6.35 FileHandlerTests.cs File Reference

Data Structures

- class [SmartStay.IO.Tests.FileOperations.FileHandlerTests](#)

Contains unit tests for the FileHandler class.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.IO](#)
- namespace [SmartStay.IO.Tests](#)
- namespace [SmartStay.IO.Tests.FileOperations](#)

The [SmartStay.IO.Tests.FileOperations](#) namespace contains unit tests for file operations used within the [SmartStay](#) application.

6.36 FileHandlerTests.cs

[Go to the documentation of this file.](#)

```

00001
00010
00015 namespace SmartStay.IO.Tests.FileOperations
00016 {
00017     using SmartStay.IO.FileOperations;
00018     using System;
00019     using System.IO;
00020     using Xunit;
00021
00025 public class FileHandlerTests
00026 {
00031     [Theory]
00032     [InlineData("")]
00033     public void ReadFile_EmptyPath_ThrowsArgumentException(string filePath)
00034     {
00035         // Act & Assert
00036         var exception = Assert.Throws<ArgumentException>(() => FileHandler.ReadFile(filePath));
00037         Assert.Equal("File path cannot be null or empty.", exception.Message);
00038     }
00039
00044     [Fact]
00045     public void ReadFile_FileDoesNotExist_ThrowsFileNotFoundException()
00046     {
00047         // Arrange
00048         var filePath = "nonexistent.txt";
00049
00050         // Act & Assert
00051         var exception = Assert.Throws<FileNotFoundException>(() => FileHandler.ReadFile(filePath));
00052         Assert.Contains($"File not found: {filePath}", exception.Message);
00053     }
00054
00058     [Fact]
00059     public void ReadFile_ValidFilePath_ReturnsFileContent()
00060     {
00061         // Arrange
00062         var filePath = "test.txt";
00063         var expectedContent = "Hello, World!";
00064         File.WriteAllText(filePath, expectedContent);

```

```

00065
00066         try
00067         {
00068             // Act
00069             var content = FileHandler.ReadFile(filePath);
00070
00071             // Assert
00072             Assert.Equal(expectedContent, content);
00073         }
00074         finally
00075         {
00076             // Cleanup
00077             File.Delete(filePath);
00078         }
00079     }
00080
00081     [Theory]
00082     [InlineData("")]
00083     public void WriteFile_EmptyPath_ThrowsArgumentException(string filePath)
00084     {
00085         // Act & Assert
00086         var exception = Assert.Throws<ArgumentException>(() => FileHandler.WriteFile(filePath, "Sample
00087 Content"));
00088         Assert.Equal("File path cannot be null or empty.", exception.Message);
00089     }
00090
00091     [Fact]
00092     public void WriteFile_ValidFilePath_WritesFileContent()
00093     {
00094         // Arrange
00095         var filePath = "output.txt";
00096         var content = "Sample Content";
00097
00098         try
00099         {
00100             // Act
00101             FileHandler.WriteFile(filePath, content);
00102
00103             // Assert
00104             Assert.True(File.Exists(filePath));
00105             Assert.Equal(content, File.ReadAllText(filePath));
00106         }
00107         finally
00108         {
00109             // Cleanup
00110             File.Delete(filePath);
00111         }
00112     }
00113
00114     [Fact]
00115     public void WriteFile_NonExistentDirectory_CreatesDirectoryAndWritesFile()
00116     {
00117         // Arrange
00118         var directoryPath = Path.Combine("nonexistent", "subdir");
00119         var filePath = Path.Combine(directoryPath, "test.txt");
00120         var content = "Hello, Directory!";
00121
00122         try
00123         {
00124             // Act
00125             FileHandler.WriteFile(filePath, content);
00126
00127             // Assert
00128             Assert.True(Directory.Exists(directoryPath));
00129             Assert.True(File.Exists(filePath));
00130             Assert.Equal(content, File.ReadAllText(filePath));
00131         }
00132         finally
00133         {
00134             // Cleanup
00135             if (Directory.Exists(directoryPath))
00136                 Directory.Delete(directoryPath, true);
00137         }
00138     }
00139 }
00140
00141 }
```

6.37 PathValidatorTests.cs File Reference

Data Structures

- class [SmartStay.IO.Tests.FileOperations.PathValidatorTests](#)

Contains unit tests for the PathValidator class.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.IO](#)
- namespace [SmartStay.IO.Tests](#)
- namespace [SmartStay.IO.Tests.FileOperations](#)

The [SmartStay.IO.Tests.FileOperations](#) namespace contains unit tests for file operations used within the [SmartStay](#) application.

6.38 PathValidatorTests.cs

[Go to the documentation of this file.](#)

```
00001 using SmartStay.IO.FileOperations;
00002
00012
00017 namespace SmartStay.IO.Tests.FileOperations
00018 {
00019     using System;
00020     using System.IO;
00021     using Xunit;
00022
00026 public class PathValidatorTests
00027 {
00032     [Fact]
00033     public void FileExists_NonExistentFile_ReturnsFalse()
00034     {
00035         // Arrange
00036         var filePath = "nonexistent.txt";
00037
00038         // Act
00039         var result = PathValidator.FileExists(filePath);
00040
00041         // Assert
00042         Assert.False(result);
00043     }
00044
00049     [Fact]
00050     public void FileExists_ExistingFile_ReturnsTrue()
00051     {
00052         // Arrange
00053         var filePath = "testfile.txt";
00054         File.WriteAllText(filePath, "Sample content");
00055
00056         try
00057         {
00058             // Act
00059             var result = PathValidator.FileExists(filePath);
00060
00061             // Assert
00062             Assert.True(result);
00063         }
00064         finally
00065         {
00066             // Cleanup
00067             File.Delete(filePath);
00068         }
00069     }
00070
00075     [Theory]
00076     [InlineData("")]
00077     [InlineData(" ")]
00078     public void IsValidFileType_NullOrEmptyFilePath_ThrowsArgumentException(string filePath)
00079     {
00080         // Arrange
00081         var extension = ".txt";
00082
00083         // Act & Assert
00084         var exception = Assert.Throws<ArgumentException>(() => PathValidator.IsValidFileType(filePath,
extension));
00085         Assert.Equal("File path cannot be null or empty.", exception.Message);
00086     }
00087
```

```

00092     [Fact]
00093     public void IsValidFileType_ValidExtension_ReturnsTrue()
00094     {
00095         // Arrange
00096         var filePath = "testfile.txt";
00097         var extension = ".txt";
00098
00099         // Act
00100         var result = PathValidator.IsValidFileType(filePath, extension);
00101
00102         // Assert
00103         Assert.True(result);
00104     }
00105
00110     [Fact]
00111     public void IsValidFileType_InvalidExtension_ReturnsFalse()
00112     {
00113         // Arrange
00114         var filePath = "testfile.jpg";
00115         var extension = ".txt";
00116
00117         // Act
00118         var result = PathValidator.IsValidFileType(filePath, extension);
00119
00120         // Assert
00121         Assert.False(result);
00122     }
00123
00128     [Fact]
00129     public void IsValidFileType_CaseInsensitiveExtensionComparison_ReturnsTrue()
00130     {
00131         // Arrange
00132         var filePath = "testfile.TXT";
00133         var extension = ".txt";
00134
00135         // Act
00136         var result = PathValidator.IsValidFileType(filePath, extension);
00137
00138         // Assert
00139         Assert.True(result);
00140     }
00141 }
00142 }

```

6.39 SmartStay.IO.Tests.AssemblyInfo.cs File Reference

6.40 SmartStay.IO.Tests.AssemblyInfo.cs

[Go to the documentation of this file.](#)

```

00001 //-----
00002 // <auto-generated>
00003 //     This code was generated by a tool.
00004 //     Runtime Version:4.0.30319.42000
00005 //
00006 //     Changes to this file may cause incorrect behavior and will be lost if
00007 //     the code is regenerated.
00008 // </auto-generated>
00009 //-----
00010
00011 using System;
00012 using System.Reflection;
00013
00014 [assembly: System.Reflection.AssemblyCompanyAttribute("SmartStay.IO.Tests")]
00015 [assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]
00016 [assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
00017 [assembly:
    System.Reflection.AssemblyInformationalVersionAttribute("1.0.0+c366ac03947932e5126b804e73253b4d5f5e0e8d")]
00018 [assembly: System.Reflection.AssemblyProductAttribute("SmartStay.IO.Tests")]
00019 [assembly: System.Reflection.AssemblyTitleAttribute("SmartStay.IO.Tests")]
00020 [assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
00021
00022 // Generated by the MSBuild WriteCodeFragment class.
00023

```

6.41 SmartStay.IO.Tests.GlobalUsings.g.cs File Reference

6.42 SmartStay.IO.Tests.GlobalUsings.g.cs

[Go to the documentation of this file.](#)

```
00001 // <auto-generated/>
00002 global using global::System;
00003 global using global::System.Collections.Generic;
00004 global using global::System.IO;
00005 global using global::System.Linq;
00006 global using global::System.Net.Http;
00007 global using global::System.Threading;
00008 global using global::System.Threading.Tasks;
```

6.43 SmartStay.Validation.Tests.AssemblyInfo.cs File Reference

6.44 SmartStay.Validation.Tests.AssemblyInfo.cs

[Go to the documentation of this file.](#)

```
00001 //-----
00002 // <auto-generated>
00003 //     This code was generated by a tool.
00004 //     Runtime Version:4.0.30319.42000
00005 //
00006 //     Changes to this file may cause incorrect behavior and will be lost if
00007 //     the code is regenerated.
00008 // </auto-generated>
00009 //-----
00010
00011 using System;
00012 using System.Reflection;
00013
00014 [assembly: System.Reflection.AssemblyCompanyAttribute("SmartStay.Validation.Tests")]
00015 [assembly: System.Reflection.AssemblyConfigurationAttribute("Debug")]
00016 [assembly: System.Reflection.AssemblyFileVersionAttribute("1.0.0.0")]
00017 [assembly:
    System.Reflection.AssemblyInformationalVersionAttribute("1.0.0+c366ac03947932e5126b804e73253b4d5f5e0e8d")]
00018 [assembly: System.Reflection.AssemblyProductAttribute("SmartStay.Validation.Tests")]
00019 [assembly: System.Reflection.AssemblyTitleAttribute("SmartStay.Validation.Tests")]
00020 [assembly: System.Reflection.AssemblyVersionAttribute("1.0.0.0")]
00021
00022 // Generated by the MSBuild WriteCodeFragment class.
00023
```

6.45 SmartStay.Validation.Tests.GlobalUsings.g.cs File Reference

6.46 SmartStay.Validation.Tests.GlobalUsings.g.cs

[Go to the documentation of this file.](#)

```
00001 // <auto-generated/>
00002 global using global::System;
00003 global using global::System.Collections.Generic;
00004 global using global::System.IO;
00005 global using global::System.Linq;
00006 global using global::System.Net.Http;
00007 global using global::System.Threading;
00008 global using global::System.Threading.Tasks;
00009 global using global::Xunit;
```

6.47 ValidationErrorMessageTests.cs File Reference

Data Structures

- class [SmartStay.Validation.Tests.ValidationErrorMessageTests](#)

Contains unit tests for the ValidationErrorMessage class. Ensures that error messages are correctly retrieved based on the provided error codes and that localization functions as expected.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Validation](#)
- namespace [SmartStay.Validation.Tests](#)

The SmartStay.Validation.Tests namespace contains unit tests for classes within the SmartStay.Validation namespace, ensuring correctness and reliability of validation functionalities.

6.48 ValidationErrorMessageTests.cs

[Go to the documentation of this file.](#)

```

00001
00011
00016 namespace SmartStay.Validation.Tests
00017 {
00018     using System.Globalization;
00019     using SmartStay.Validation;
00020     using Xunit;
00021
00027 public class ValidationErrorMessageTests
00028 {
00032     [Fact]
00033     public void GetErrorMessage_ValidErrorCode_ReturnsLocalizedMessage()
00034     {
00035         // Arrange
00036         var errorCode = ValidationErrorCode.InvalidName;
00037         var expectedMessage = "The provided name is invalid.";
00038
00039         // Act
00040         var result = ValidationErrorMessage.GetErrorMessage(errorCode);
00041
00042         // Assert
00043         Assert.Equal(expectedMessage, result);
00044     }
00045
00049     [Fact]
00050     public void GetErrorMessage_InvalidErrorCode_ReturnsFallbackMessage()
00051     {
00052         // Arrange
00053         var invalidErrorCode = (ValidationErrorCode)9999;
00054         var expectedFallbackMessage = "Unknown validation error.";
00055
00056         // Act
00057         var result = ValidationErrorMessage.GetErrorMessage(invalidErrorCode);
00058
00059         // Assert
00060         Assert.Equal(expectedFallbackMessage, result);
00061     }
00062
00066     [Fact]
00067     public void GetErrorMessage_SupportsLocalization()
00068     {
00069         // Arrange
00070         var errorCode = ValidationErrorCode.InvalidName;
00071         var expectedMessagePt = "O nome fornecido é inválido."; // Portuguese translation
00072         var expectedMessageEn = "The provided name is invalid."; // English translation
00073
00074         // Act
00075         CultureInfo.CurrentCulture = new CultureInfo("pt-PT");
00076         var resultPt = ValidationErrorMessage.GetErrorMessage(errorCode);
00077
00078         CultureInfo.CurrentCulture = new CultureInfo("en-US");

```

```

00079         var resultEn = ValidationErrorMessage.GetErrorMessage(errorCode);
00080
00081         // Assert
00082         Assert.Equal(expectedMessagePt, resultPt);
00083         Assert.Equal(expectedMessageEn, resultEn);
00084     }
00085 }
00086 }

```

6.49 ValidationExceptionTests.cs File Reference

Data Structures

- class [SmartStay.Validation.Tests.ValidationExceptionTests](#)

Contains unit tests for the ValidationException class. Ensures that exceptions are created with the expected error codes and messages.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Validation](#)
- namespace [SmartStay.Validation.Tests](#)

The [SmartStay.Validation.Tests](#) namespace contains unit tests for classes within the [SmartStay.Validation](#) namespace, ensuring correctness and reliability of validation functionalities.

6.50 ValidationExceptionTests.cs

[Go to the documentation of this file.](#)

```

00001
00011
00016 namespace SmartStay.Validation.Tests
00017 {
00018     using SmartStay.Validation;
00019     using Xunit;
00020
00025 public class ValidationExceptionTests
00026 {
00031     [Fact]
00032     public void Constructor_WithErrorCode_SetsErrorCodeAndMessage()
00033     {
00034         // Arrange
00035         var expectedErrorCode = ValidationErrorCode.InvalidName;
00036         var expectedErrorMessage = ValidationErrorMessage.GetErrorMessage(expectedErrorCode);
00037
00038         // Act
00039         var exception = new ValidationException(expectedErrorCode);
00040
00041         // Assert
00042         Assert.Equal(expectedErrorCode, exception.ErrorCode);
00043         Assert.Equal(expectedErrorMessage, exception.Message);
00044     }
00045
00050     [Fact]
00051     public void Constructor_WithUnknownErrorCode_UsesFallbackMessage()
00052     {
00053         // Arrange
00054         var invalidErrorCode = (ValidationErrorCode)9999; // Undefined error code
00055         var expectedFallbackMessage = "Unknown validation error.";
00056
00057         // Act
00058         var exception = new ValidationException(invalidErrorCode);
00059
00060         // Assert
00061         Assert.Equal(invalidErrorCode, exception.ErrorCode);
00062         Assert.Equal(expectedFallbackMessage, exception.Message);
00063     }
00064 }
00065 }

```

6.51 AccommodationValidatorTests.cs File Reference

Data Structures

- class [SmartStay.Validation.Tests.Validators.AccommodationValidatorTests](#)

Contains unit tests for the AccommodationValidator class. Tests the validation logic for accommodation types and IDs.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Validation](#)
- namespace [SmartStay.Validation.Tests](#)

The [SmartStay.Validation.Tests](#) namespace contains unit tests for classes within the [SmartStay.Validation](#) namespace, ensuring correctness and reliability of validation functionalities.

- namespace [SmartStay.Validation.Tests.Validators](#)

The [SmartStay.Validation.Tests.Validators](#) namespace contains unit tests for the validation logic of different fields, specifically focusing on the AccommodationValidator class for validating accommodation types and IDs.

6.52 AccommodationValidatorTests.cs

[Go to the documentation of this file.](#)

```
00001
00010
00016 namespace SmartStay.Validation.Tests.Validators
00017 {
00018     using SmartStay.Validation;
00019     using SmartStay.Validation.Validators;
00020     using SmartStay.Common.Enums;
00021     using Xunit;
00022
00027 public class AccommodationValidatorTests
00028 {
00033     [Fact]
00034     public void ValidateAccommodationType_ValidAccommodationType_ReturnsAccommodationType()
00035     {
00036         // Arrange
00037         var validAccommodationType = AccommodationType.Apartment;
00038
00039         // Act
00040         var result = AccommodationValidator.ValidateAccommodationType(validAccommodationType);
00041
00042         // Assert
00043         Assert.Equal(validAccommodationType, result);
00044     }
00045
00050     [Fact]
00051     public void ValidateAccommodationType_InvalidAccommodationType_ThrowsValidationException()
00052     {
00053         // Arrange
00054         var invalidAccommodationType = (AccommodationType)9999; // Invalid enum value
00055
00056         // Act & Assert
00057         var exception = Assert.Throws<ValidationException>(
00058             () => AccommodationValidator.ValidateAccommodationType(invalidAccommodationType));
00059         Assert.Equal(ValidationErrorCode.InvalidAccommodationType, exception.ErrorCode);
00060     }
00061
00066     [Fact]
00067     public void IsValidAccommodationType_ValidAccommodationType_ReturnsTrue()
00068     {
00069         // Arrange
00070         var validAccommodationType = AccommodationType.House;
00071
00072         // Act
00073         var result = AccommodationValidator.IsValidAccommodationType(validAccommodationType);
00074     }
00074
```

```

00075         // Assert
00076         Assert.True(result);
00077     }
00078
00083     [Fact]
00084     public void IsValidAccommodationType_InvalidAccommodationType_ReturnsFalse()
00085     {
00086         // Arrange
00087         var invalidAccommodationType = (AccommodationType)9999; // Invalid enum value
00088
00089         // Act
00090         var result = AccommodationValidator.IsValidAccommodationType(invalidAccommodationType);
00091
00092         // Assert
00093         Assert.False(result);
00094     }
00095
00100     [Fact]
00101     public void ValidateAccommodationId_ValidAccommodationId_ReturnsAccommodationId()
00102     {
00103         // Arrange
00104         var validAccommodationId = 1;
00105
00106         // Act
00107         var result = AccommodationValidator.ValidateAccommodationId(validAccommodationId);
00108
00109         // Assert
00110         Assert.Equal(validAccommodationId, result);
00111     }
00112
00117     [Fact]
00118     public void ValidateAccommodationId_InvalidAccommodationId_ThrowsValidationException()
00119     {
00120         // Arrange
00121         var invalidAccommodationId = 0; // Invalid ID
00122
00123         // Act & Assert
00124         var exception = Assert.Throws<ValidationException>(
00125             () => AccommodationValidator.ValidateAccommodationId(invalidAccommodationId));
00126         Assert.Equal(ValidationErrorCode.InvalidId, exception.ErrorCode);
00127     }
00128
00133     [Fact]
00134     public void IsValidAccommodationId_ValidAccommodationId_ReturnsTrue()
00135     {
00136         // Arrange
00137         var validAccommodationId = 10;
00138
00139         // Act
00140         var result = AccommodationValidator.IsValidAccommodationId(validAccommodationId);
00141
00142         // Assert
00143         Assert.True(result);
00144     }
00145
00150     [Fact]
00151     public void IsValidAccommodationId_InvalidAccommodationId_ReturnsFalse()
00152     {
00153         // Arrange
00154         var invalidAccommodationId = -1; // Invalid ID
00155
00156         // Act
00157         var result = AccommodationValidator.IsValidAccommodationId(invalidAccommodationId);
00158
00159         // Assert
00160         Assert.False(result);
00161     }
00162 }
00163 }

```

6.53 AddressValidatorTests.cs File Reference

Data Structures

- class [SmartStay.Validation.Tests.Validators.AddressValidatorTests](#)

Contains unit tests for the AddressValidator class. Tests the validation logic for addresses used in the SmartStay application.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Validation](#)
- namespace [SmartStay.Validation.Tests](#)

The [SmartStay.Validation.Tests](#) namespace contains unit tests for classes within the [SmartStay.Validation](#) namespace, ensuring correctness and reliability of validation functionalities.

- namespace [SmartStay.Validation.Tests.Validators](#)

The [SmartStay.Validation.Tests.Validators](#) namespace contains unit tests for the validation logic of different fields, specifically focusing on the [AccommodationValidator](#) class for validating accommodation types and IDs.

6.54 AddressValidatorTests.cs

[Go to the documentation of this file.](#)

```

00001
00010
00015 namespace SmartStay.Validation.Tests.Validators
00016 {
00017     using SmartStay.Validation;
00018     using SmartStay.Validation.Validators;
00019     using Xunit;
00020
00025 public class AddressValidatorTests
00026 {
00031     [Fact]
00032     public void ValidateAddress_ValidAddress_ReturnsAddress()
00033     {
00034         // Arrange
00035         var validAddress = "123 Main Street, Cityville";
00036
00037         // Act
00038         var result = AddressValidator.ValidateAddress(validAddress);
00039
00040         // Assert
00041         Assert.Equal(validAddress, result);
00042     }
00043
00048     [Fact]
00049     public void ValidateAddress_InvalidAddress_ThrowsValidationException()
00050     {
00051         // Arrange
00052         var invalidAddress = ""; // Empty address is invalid
00053
00054         // Act & Assert
00055         var exception = Assert.Throws<ValidationException>(() =>
AddressValidator.ValidateAddress(invalidAddress));
00056         Assert.Equal(ValidationErrorCodes.InvalidAddress, exception.ErrorCode);
00057     }
00058
00063     [Fact]
00064     public void IsValidAddress_ValidAddress_ReturnsTrue()
00065     {
00066         // Arrange
00067         var validAddress = "456 Oak Avenue, Smalltown";
00068
00069         // Act
00070         var result = AddressValidator.IsValidAddress(validAddress);
00071
00072         // Assert
00073         Assert.True(result);
00074     }
00075
00080     [Fact]
00081     public void IsValidAddress_InvalidAddress_ReturnsFalse()
00082     {
00083         // Arrange
00084         var invalidAddress = " "; // Address is only whitespace, hence invalid
00085
00086         // Act
00087         var result = AddressValidator.IsValidAddress(invalidAddress);
00088
00089         // Assert
00090         Assert.False(result);
00091     }
00092 }
00093 }

```

6.55 ClientValidatorTests.cs File Reference

Data Structures

- class [SmartStay.Validation.Tests.Validators.ClientValidatorTests](#)

Contains unit tests for the ClientValidator class. Tests the validation logic for client-related data in the [SmartStay](#) application.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Validation](#)
- namespace [SmartStay.Validation.Tests](#)

The [SmartStay.Validation.Tests](#) namespace contains unit tests for classes within the [SmartStay.Validation](#) namespace, ensuring correctness and reliability of validation functionalities.

- namespace [SmartStay.Validation.Tests.Validators](#)

The [SmartStay.Validation.Tests.Validators](#) namespace contains unit tests for the validation logic of different fields, specifically focusing on the AccommodationValidator class for validating accommodation types and IDs.

6.56 ClientValidatorTests.cs

[Go to the documentation of this file.](#)

```

00001
00010
00015 namespace SmartStay.Validation.Tests.Validators
00016 {
00017     using SmartStay.Validation;
00018     using SmartStay.Validation.Validators;
00019     using Xunit;
00020
00025 public class ClientValidatorTests
00026 {
00031     [Fact]
00032     public void ValidateClientId_ValidId_ReturnsClientId()
00033     {
00034         // Arrange
00035         int validClientId = 123;
00036
00037         // Act
00038         var result = ClientValidator.ValidateClientId(validClientId);
00039
00040         // Assert
00041         Assert.Equal(validClientId, result);
00042     }
00043
00048     [Fact]
00049     public void ValidateClientId_InvalidId_ThrowsValidationException()
00050     {
00051         // Arrange
00052         int invalidClientId = 0; // ID cannot be zero or negative
00053
00054         // Act & Assert
00055         var exception = Assert.Throws<ValidationException>(() =>
00056             ClientValidator.ValidateClientId(invalidClientId));
00057         Assert.Equal(ValidationErrorCodes.InvalidId, exception.ErrorCode);
00058
00063     [Fact]
00064     public void IsValidClientId_ValidId_ReturnsTrue()
00065     {
00066         // Arrange
00067         int validClientId = 456;
00068
00069         // Act
00070         var result = ClientValidator.IsValidClientId(validClientId);
00071
00072         // Assert

```

```

00073         Assert.True(result);
00074     }
00075
00080     [Fact]
00081     public void IsValidClientId_InvalidId_ReturnsFalse()
00082     {
00083         // Arrange
00084         int invalidClientId = -1; // Invalid ID (negative)
00085
00086         // Act
00087         var result = ClientValidator.IsValidClientId(invalidClientId);
00088
00089         // Assert
00090         Assert.False(result);
00091     }
00092 }
00093 }

```

6.57 DateValidatorTests.cs File Reference

Data Structures

- class [SmartStay.Validation.Tests.Validators.DateValidatorTests](#)

Contains unit tests for the DateValidator class. Tests the validation logic for dates such as check-in and check-out dates.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Validation](#)
- namespace [SmartStay.Validation.Tests](#)

The [SmartStay.Validation.Tests](#) namespace contains unit tests for classes within the [SmartStay.Validation](#) namespace, ensuring correctness and reliability of validation functionalities.

- namespace [SmartStay.Validation.Tests.Validators](#)

The [SmartStay.Validation.Tests.Validators](#) namespace contains unit tests for the validation logic of different fields, specifically focusing on the AccommodationValidator class for validating accommodation types and IDs.

6.58 DateValidatorTests.cs

[Go to the documentation of this file.](#)

```

00001
00011
00016 namespace SmartStay.Validation.Tests.Validators
00017 {
00018     using SmartStay.Validation;
00019     using SmartStay.Validation.Validators;
00020     using System;
00021     using Xunit;
00022
00027 public class DateValidatorTests
00028 {
00033     [Fact]
00034     public void ValidateCheckInDate_ValidDate_ReturnsCheckInDate()
00035     {
00036         // Arrange
00037         DateTime validCheckInDate = DateTime.Today;
00038
00039         // Act
00040         var result = DateValidator.ValidateCheckInDate(validCheckInDate);
00041
00042         // Assert
00043         Assert.Equal(validCheckInDate, result);
00044     }
00045 }

```

```

00050     [Fact]
00051     public void ValidateCheckInDate_InvalidDate_ThrowsValidationException()
00052     {
00053         // Arrange
00054         DateTime invalidCheckInDate = DateTime.Today.AddDays(-1); // Past date
00055
00056         // Act & Assert
00057         var exception = Assert.Throws<ValidationException>(() =>
DateValidator.ValidateCheckInDate(invalidCheckInDate));
00058         Assert.Equal(ValidationErrorCode.InvalidDate, exception.ErrorCode);
00059     }
00060
00061     [Fact]
00062     public void ValidateCheckOutDate_ValidDateRange_ReturnsCheckOutDate()
00063     {
00064         // Arrange
00065         DateTime validCheckInDate = DateTime.Today;
00066         DateTime validCheckOutDate = DateTime.Today.AddDays(1); // Check-out after check-in
00067
00068         // Act
00069         var result = DateValidator.ValidateCheckOutDate(validCheckOutDate, validCheckInDate);
00070
00071         // Assert
00072         Assert.Equal(validCheckOutDate, result);
00073     }
00074
00075     [Fact]
00076     public void ValidateCheckOutDate_InvalidDateRange_ThrowsValidationException()
00077     {
00078         // Arrange
00079         DateTime invalidCheckInDate = DateTime.Today;
00080         DateTime invalidCheckOutDate = DateTime.Today.AddDays(-1); // Check-out before check-in
00081
00082         // Act & Assert
00083         var exception = Assert.Throws<ValidationException>(
00084             () => DateValidator.ValidateCheckOutDate(invalidCheckOutDate, invalidCheckInDate));
00085         Assert.Equal(ValidationErrorCode.InvalidDateRange, exception.ErrorCode);
00086     }
00087
00088     [Fact]
00089     public void IsValidFutureDate_ValidDate_ReturnsTrue()
00090     {
00091         // Arrange
00092         DateTime validDate = DateTime.Today;
00093
00094         // Act
00095         var result = DateValidator.IsValidFutureDate(validDate);
00096
00097         // Assert
00098         Assert.True(result);
00099     }
00100
00101     [Fact]
00102     public void IsValidFutureDate_InvalidDate_ReturnsFalse()
00103     {
00104         // Arrange
00105         DateTime invalidDate = DateTime.Today.AddDays(-1); // Past date
00106
00107         // Act
00108         var result = DateValidator.IsValidFutureDate(invalidDate);
00109
00110         // Assert
00111         Assert.False(result);
00112     }
00113
00114     [Fact]
00115     public void IsValidDateRange_ValidDateRange_ReturnsTrue()
00116     {
00117         // Arrange
00118         DateTime checkInDate = DateTime.Today;
00119         DateTime checkOutDate = DateTime.Today.AddDays(2);
00120
00121         // Act
00122         var result = DateValidator.IsValidDateRange(checkInDate, checkOutDate);
00123
00124         // Assert
00125         Assert.True(result);
00126     }
00127
00128     [Fact]
00129     public void IsValidDateRange_InvalidDateRange_ReturnsFalse()
00130     {
00131         // Arrange
00132         DateTime checkInDate = DateTime.Today.AddDays(2);
00133         DateTime checkOutDate = DateTime.Today;
00134
00135         // Act
00136     }

```

```

00160         var result = DateValidator.IsValidDateRange(checkInDate, checkOutDate);
00161
00162         // Assert
00163         Assert.False(result);
00164     }
00165 }
00166 }

```

6.59 EmailValidatorTests.cs File Reference

Data Structures

- class [SmartStay.Validation.Tests.Validators.EmailValidatorTests](#)
Contains unit tests for the EmailValidator class. Tests the validation logic for email addresses used in the [SmartStay](#) application.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Validation](#)
- namespace [SmartStay.Validation.Tests](#)
The [SmartStay.Validation.Tests](#) namespace contains unit tests for classes within the [SmartStay.Validation](#) namespace, ensuring correctness and reliability of validation functionalities.
- namespace [SmartStay.Validation.Tests.Validators](#)
The [SmartStay.Validation.Tests.Validators](#) namespace contains unit tests for the validation logic of different fields, specifically focusing on the AccommodationValidator class for validating accommodation types and IDs.

6.60 EmailValidatorTests.cs

[Go to the documentation of this file.](#)

```

00001
00010
00015 namespace SmartStay.Validation.Tests.Validators
00016 {
00017     using SmartStay.Validation;
00018     using SmartStay.Validation.Validators;
00019     using Xunit;
00020
00025 public class EmailValidatorTests
00026 {
00031     [Fact]
00032     public void ValidateEmail_ValidEmail_ReturnsEmail()
00033     {
00034         // Arrange
00035         string validEmail = "test@example.com";
00036
00037         // Act
00038         var result = EmailValidator.ValidateEmail(validEmail);
00039
00040         // Assert
00041         Assert.Equal(validEmail, result);
00042     }
00043
00048     [Fact]
00049     public void ValidateEmail_InvalidEmail_ThrowsValidationException()
00050     {
00051         // Arrange
00052         string invalidEmail = "invalid-email";
00053
00054         // Act & Assert
00055         var exception = Assert.Throws<ValidationException>(() =>
            EmailValidator.ValidateEmail(invalidEmail));
00056         Assert.Equal(ValidationErrorCodes.InvalidEmail, exception.ErrorCode);
00057     }

```

```
00058
00063 [Fact]
00064 public void IsValidEmail_ValidEmail_ReturnsTrue()
00065 {
00066     // Arrange
00067     string validEmail = "valid@example.com";
00068
00069     // Act
00070     var result = EmailValidator.IsValidEmail(validEmail);
00071
00072     // Assert
00073     Assert.True(result);
00074 }
00075
00080 [Fact]
00081 public void IsValidEmail_MissingAtSymbol_ReturnsFalse()
00082 {
00083     // Arrange
00084     string invalidEmail = "missingatsign.com";
00085
00086     // Act
00087     var result = EmailValidator.IsValidEmail(invalidEmail);
00088
00089     // Assert
00090     Assert.False(result);
00091 }
00092
00097 [Fact]
00098 public void IsValidEmail_MissingDomainExtension_ReturnsFalse()
00099 {
00100     // Arrange
00101     string invalidEmail = "missingdomain@com";
00102
00103     // Act
00104     var result = EmailValidator.IsValidEmail(invalidEmail);
00105
00106     // Assert
00107     Assert.False(result);
00108 }
00109
00114 [Fact]
00115 public void IsValidEmail_EmptyEmail_ReturnsFalse()
00116 {
00117     // Arrange
00118     string invalidEmail = string.Empty;
00119
00120     // Act
00121     var result = EmailValidator.IsValidEmail(invalidEmail);
00122
00123     // Assert
00124     Assert.False(result);
00125 }
00126
00131 [Fact]
00132 public void IsValidEmail_NullEmail_ReturnsFalse()
00133 {
00134     // Arrange
00135     string invalidEmail = null;
00136
00137     // Act
00138     var result = EmailValidator.IsValidEmail(invalidEmail);
00139
00140     // Assert
00141     Assert.False(result);
00142 }
00143
00148 [Fact]
00149 public void IsValidEmail_InvalidCharacters_ReturnsFalse()
00150 {
00151     // Arrange
00152     string invalidEmail = "invalid@ex$ample.com"; // Invalid character '$'
00153
00154     // Act
00155     var result = EmailValidator.IsValidEmail(invalidEmail);
00156
00157     // Assert
00158     Assert.False(result);
00159 }
00160 }
00161 }
```

6.61 NameValidatorTests.cs File Reference

Data Structures

- class [SmartStay.Validation.Tests.Validators.NameValidatorTests](#)

Contains unit tests for the NameValidator class. Validates both general names and accommodation names, checking correct behavior when the names are valid or invalid.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Validation](#)
- namespace [SmartStay.Validation.Tests](#)

The [SmartStay.Validation.Tests](#) namespace contains unit tests for classes within the [SmartStay.Validation](#) namespace, ensuring correctness and reliability of validation functionalities.

- namespace [SmartStay.Validation.Tests.Validators](#)

The [SmartStay.Validation.Tests.Validators](#) namespace contains unit tests for the validation logic of different fields, specifically focusing on the AccommodationValidator class for validating accommodation types and IDs.

6.62 NameValidatorTests.cs

[Go to the documentation of this file.](#)

```

00001
00011
00016 namespace SmartStay.Validation.Tests.Validators
00017 {
00018     using SmartStay.Validation;
00019     using SmartStay.Validation.Validators;
00020     using Xunit;
00021
00027 public class NameValidatorTests
00028 {
00033     [Fact]
00034     public void ValidateName_ValidName_ReturnsName()
00035     {
00036         // Arrange
00037         string validName = "Enrique Rodriguez";
00038
00039         // Act
00040         var result = NameValidator.ValidateName(validName);
00041
00042         // Assert
00043         Assert.Equal(validName, result);
00044     }
00045
00050     [Fact]
00051     public void ValidateName_InvalidName_ThrowsValidationException()
00052     {
00053         // Arrange
00054         string invalidName = ""; // Empty name is invalid
00055
00056         // Act & Assert
00057         var exception = Assert.Throws<ValidationException>(() =>
NameValidator.ValidateName(invalidName));
00058         Assert.Equal(ValidationErrorCode.InvalidName, exception.ErrorCode);
00059     }
00060
00065     [Fact]
00066     public void ValidateName_TooLongName_ThrowsValidationException()
00067     {
00068         // Arrange
00069         string tooLongName = new string('a', 51); // Exceeds max length
00070
00071         // Act & Assert
00072         var exception = Assert.Throws<ValidationException>(() =>
NameValidator.ValidateName(tooLongName));
00073         Assert.Equal(ValidationErrorCode.InvalidName, exception.ErrorCode);

```

```
00074     }
00075
00080     [Fact]
00081     public void ValidateAccommodationName_ValidName_ReturnsName()
00082     {
00083         // Arrange
00084         string validAccommodationName = "Cozy Apartment";
00085
00086         // Act
00087         var result = NameValidator.ValidateAccommodationName(validAccommodationName);
00088
00089         // Assert
00090         Assert.Equal(validAccommodationName, result);
00091     }
00092
00097     [Fact]
00098     public void ValidateAccommodationName_TooLongName_ThrowsValidationException()
00099     {
00100         // Arrange
00101         string tooLongAccommodationName = new string('b', 101); // Exceeds max length
00102
00103         // Act & Assert
00104         var exception =
00105             Assert.Throws<ValidationException>(() =>
00106                 NameValidator.ValidateAccommodationName(tooLongAccommodationName));
00107         Assert.Equal(ValidationErrorCode.InvalidAccommodationName, exception.ErrorCode);
00108     }
00113     [Fact]
00114     public void IsValidName_ValidName_ReturnsTrue()
00115     {
00116         // Arrange
00117         string validName = "Alice";
00118
00119         // Act
00120         var result = NameValidator.IsValidName(validName);
00121
00122         // Assert
00123         Assert.True(result);
00124     }
00125
00130     [Fact]
00131     public void IsValidName_InvalidName_ReturnsFalse()
00132     {
00133         // Arrange
00134         string invalidName = "";
00135
00136         // Act
00137         var result = NameValidator.IsValidName(invalidName);
00138
00139         // Assert
00140         Assert.False(result);
00141     }
00142
00147     [Fact]
00148     public void IsValidAccommodationName_ValidName_ReturnsTrue()
00149     {
00150         // Arrange
00151         string validAccommodationName = "Modern Studio";
00152
00153         // Act
00154         var result = NameValidator.IsValidAccommodationName(validAccommodationName);
00155
00156         // Assert
00157         Assert.True(result);
00158     }
00159
00164     [Fact]
00165     public void IsValidAccommodationName_InvalidName_ReturnsFalse()
00166     {
00167         // Arrange
00168         string invalidAccommodationName = null;
00169
00170         // Act
00171         var result = NameValidator.IsValidAccommodationName(invalidAccommodationName);
00172
00173         // Assert
00174         Assert.False(result);
00175     }
00176 }
00177 }
```

6.63 OwnerValidatorTests.cs File Reference

Data Structures

- class [SmartStay.Validation.Tests.Validators.OwnerValidatorTests](#)
Contains unit tests for the *OwnerValidator* class. Tests the validation logic for owner-related data used in the *SmartStay* application.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Validation](#)
- namespace [SmartStay.Validation.Tests](#)
The *SmartStay.Validation.Tests* namespace contains unit tests for classes within the *SmartStay.Validation* namespace, ensuring correctness and reliability of validation functionalities.
- namespace [SmartStay.Validation.Tests.Validators](#)
The *SmartStay.Validation.Tests.Validators* namespace contains unit tests for the validation logic of different fields, specifically focusing on the *AccommodationValidator* class for validating accommodation types and IDs.

6.64 OwnerValidatorTests.cs

[Go to the documentation of this file.](#)

```

00001
00010
00015 namespace SmartStay.Validation.Tests.Validators
00016 {
00017     using SmartStay.Validation;
00018     using SmartStay.Validation.Validators;
00019     using Xunit;
00020
00025 public class OwnerValidatorTests
00026 {
00031     [Fact]
00032     public void ValidateOwnerId_ValidId_ReturnsId()
00033     {
00034         // Arrange
00035         int validId = 123;
00036
00037         // Act
00038         var result = OwnerValidator.ValidateOwnerId(validId);
00039
00040         // Assert
00041         Assert.Equal(validId, result);
00042     }
00043
00048     [Fact]
00049     public void ValidateOwnerId_InvalidId_ThrowsValidationException()
00050     {
00051         // Arrange
00052         int invalidId = 0;
00053
00054         // Act & Assert
00055         var exception = Assert.Throws<ValidationException>(() =>
00056             OwnerValidator.ValidateOwnerId(invalidId));
00057         Assert.Equal(ValidationErrorCodes.InvalidId, exception.ErrorCode);
00058
00063     [Fact]
00064     public void ValidateOwnerName_ValidName_ReturnsName()
00065     {
00066         // Arrange
00067         string validName = "John Doe";
00068
00069         // Act
00070         var result = OwnerValidator.ValidateOwnerName(validName);
00071
00072         // Assert

```

```

00073         Assert.Equal(validName, result);
00074     }
00075
00080     [Fact]
00081     public void ValidateOwnerName_InvalidName_ThrowsValidationException()
00082     {
00083         // Arrange
00084         string invalidName = "J";
00085
00086         // Act & Assert
00087         var exception = Assert.Throws<ValidationException>(() =>
00088             OwnerValidator.ValidateOwnerName(invalidName));
00089         Assert.Equal(ValidationErrorCode.InvalidName, exception.ErrorCode);
00090     }
00091
00095     [Fact]
00096     public void ValidateOwnerEmail_ValidEmail_ReturnsEmail()
00097     {
00098         // Arrange
00099         string validEmail = "owner@example.com";
00100
00101         // Act
00102         var result = OwnerValidator.ValidateOwnerEmail(validEmail);
00103
00104         // Assert
00105         Assert.Equal(validEmail, result);
00106     }
00107
00112     [Fact]
00113     public void ValidateOwnerEmail_InvalidEmail_ThrowsValidationException()
00114     {
00115         // Arrange
00116         string invalidEmail = "invalid-email";
00117
00118         // Act & Assert
00119         var exception = Assert.Throws<ValidationException>(() =>
00120             OwnerValidator.ValidateOwnerEmail(invalidEmail));
00121         Assert.Equal(ValidationErrorCode.InvalidEmail, exception.ErrorCode);
00122     }
00123
00127     [Fact]
00128     public void ValidateOwnerPhoneNumber_ValidPhoneNumber_ReturnsPhoneNumber()
00129     {
00130         // Arrange
00131         string validPhoneNumber = "1234567890";
00132
00133         // Act
00134         var result = OwnerValidator.ValidateOwnerPhoneNumber(validPhoneNumber);
00135
00136         // Assert
00137         Assert.Equal(validPhoneNumber, result);
00138     }
00139
00144     [Fact]
00145     public void ValidateOwnerPhoneNumber_InvalidPhoneNumber_ThrowsValidationException()
00146     {
00147         // Arrange
00148         string invalidPhoneNumber = "123";
00149
00150         // Act & Assert
00151         var exception =
00152             Assert.Throws<ValidationException>(() =>
00153                 OwnerValidator.ValidateOwnerPhoneNumber(invalidPhoneNumber));
00154         Assert.Equal(ValidationErrorCode.InvalidPhoneNumber, exception.ErrorCode);
00155     }
00156 }

```

6.65 PaymentValidatorTests.cs File Reference

Data Structures

- class [SmartStay.Validation.Tests.Validators.PaymentValidatorTests](#)

Contains unit tests for the PaymentValidator class. Tests the validation logic for payment-related data in the [SmartStay](#) application.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Validation](#)
- namespace [SmartStay.Validation.Tests](#)

The [SmartStay.Validation.Tests](#) namespace contains unit tests for classes within the [SmartStay.Validation](#) namespace, ensuring correctness and reliability of validation functionalities.

- namespace [SmartStay.Validation.Tests.Validators](#)

The [SmartStay.Validation.Tests.Validators](#) namespace contains unit tests for the validation logic of different fields, specifically focusing on the [AccommodationValidator](#) class for validating accommodation types and IDs.

6.66 PaymentValidatorTests.cs

[Go to the documentation of this file.](#)

```

00001
00010
00015 namespace SmartStay.Validation.Tests.Validators
00016 {
00017     using SmartStay.Validation;
00018     using SmartStay.Validation.Validators;
00019     using SmartStay.Common.Enums;
00020     using Xunit;
00021
00026 public class PaymentValidatorTests
00027 {
00032     [Fact]
00033     public void ValidatePrice_ValidPrice_ReturnsPrice()
00034     {
00035         // Arrange
00036         decimal validPrice = 100.00m;
00037
00038         // Act
00039         var result = PaymentValidator.ValidatePrice(validPrice);
00040
00041         // Assert
00042         Assert.Equal(validPrice, result);
00043     }
00044
00049     [Fact]
00050     public void ValidatePrice_InvalidPrice_ThrowsValidationException()
00051     {
00052         // Arrange
00053         decimal invalidPrice = -100.00m;
00054
00055         // Act & Assert
00056         var exception = Assert.Throws<ValidationException>(() =>
PaymentValidator.ValidatePrice(invalidPrice));
00057         Assert.Equal(ValidationErrorCodes.InvalidPrice, exception.ErrorCode);
00058     }
00059
00064     [Fact]
00065     public void ValidateTotalCost_ValidTotalCost_ReturnsTotalCost()
00066     {
00067         // Arrange
00068         decimal validTotalCost = 200.00m;
00069
00070         // Act
00071         var result = PaymentValidator.ValidateTotalCost(validTotalCost);
00072
00073         // Assert
00074         Assert.Equal(validTotalCost, result);
00075     }
00076
00081     [Fact]
00082     public void ValidateTotalCost_InvalidTotalCost_ThrowsValidationException()
00083     {
00084         // Arrange
00085         decimal invalidTotalCost = -200.00m;
00086
00087         // Act & Assert
00088         var exception = Assert.Throws<ValidationException>(() =>
PaymentValidator.ValidateTotalCost(invalidTotalCost));
00089         Assert.Equal(ValidationErrorCodes.InvalidTotalCost, exception.ErrorCode);
00090     }
00091

```

```

00096     [Fact]
00097     public void ValidatePaymentAmount_ValidAmount_ReturnsAmount()
00098     {
00099         // Arrange
00100         decimal validAmount = 50.00m;
00101
00102         // Act
00103         var result = PaymentValidator.ValidatePaymentAmount(validAmount);
00104
00105         // Assert
00106         Assert.Equal(validAmount, result);
00107     }
00108
00113     [Fact]
00114     public void ValidatePaymentAmount_InvalidAmount_ThrowsValidationException()
00115     {
00116         // Arrange
00117         decimal invalidAmount = -50.00m;
00118
00119         // Act & Assert
00120         var exception = Assert.Throws<ValidationException>(() =>
PaymentValidator.ValidatePaymentAmount(invalidAmount));
00121         Assert.Equal(ValidationErrorCodes.InvalidPaymentValue, exception.ErrorCode);
00122     }
00123
00128     [Fact]
00129     public void ValidatePaymentStatus_ValidStatus_ReturnsStatus()
00130     {
00131         // Arrange
00132         PaymentStatus validStatus = PaymentStatus.Completed;
00133
00134         // Act
00135         var result = PaymentValidator.ValidatePaymentStatus(validStatus);
00136
00137         // Assert
00138         Assert.Equal(validStatus, result);
00139     }
00140
00145     [Fact]
00146     public void ValidatePaymentStatus_InvalidStatus_ThrowsValidationException()
00147     {
00148         // Arrange
00149         PaymentStatus invalidStatus = (PaymentStatus)999; // Invalid enum value
00150
00151         // Act & Assert
00152         var exception = Assert.Throws<ValidationException>(() =>
PaymentValidator.ValidatePaymentStatus(invalidStatus));
00153         Assert.Equal(ValidationErrorCodes.InvalidPaymentStatus, exception.ErrorCode);
00154     }
00155
00160     [Fact]
00161     public void ValidatePaymentMethod_ValidMethod_ReturnsMethod()
00162     {
00163         // Arrange
00164         PaymentMethod validMethod = PaymentMethod.BankTransfer;
00165
00166         // Act
00167         var result = PaymentValidator.ValidatePaymentMethod(validMethod);
00168
00169         // Assert
00170         Assert.Equal(validMethod, result);
00171     }
00172
00177     [Fact]
00178     public void ValidatePaymentMethod_InvalidMethod_ThrowsValidationException()
00179     {
00180         // Arrange
00181         PaymentMethod invalidMethod = (PaymentMethod)999; // Invalid enum value
00182
00183         // Act & Assert
00184         var exception = Assert.Throws<ValidationException>(() =>
PaymentValidator.ValidatePaymentMethod(invalidMethod));
00185         Assert.Equal(ValidationErrorCodes.InvalidPaymentMethod, exception.ErrorCode);
00186     }
00187
00192     [Fact]
00193     public void ValidatePayment_ValidPayment_ReturnsPayment()
00194     {
00195         // Arrange
00196         decimal validPayment = 150.00m;
00197
00198         // Act
00199         var result = PaymentValidator.ValidatePayment(validPayment);
00200
00201         // Assert
00202         Assert.Equal(validPayment, result);
00203     }

```

```

00204
00209     [Fact]
00210     public void ValidatePayment_InvalidPayment_ThrowsValidationException()
00211     {
00212         // Arrange
00213         decimal invalidPayment = -150.00m;
00214
00215         // Act & Assert
00216         var exception = Assert.Throws<ValidationException>(() =>
PaymentValidator.ValidatePayment(invalidPayment));
00217         Assert.Equal(ValidationErrorCodes.InvalidPaymentValue, exception.ErrorCode);
00218     }
00219 }
00220 }

```

6.67 PhoneNumberValidatorTests.cs File Reference

Data Structures

- class [SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests](#)

Contains unit tests for the [PhoneNumberValidator](#) class. Tests the validation logic for phone numbers in the [SmartStay](#) application.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Validation](#)
- namespace [SmartStay.Validation.Tests](#)

The [SmartStay.Validation.Tests](#) namespace contains unit tests for classes within the [SmartStay.Validation](#) namespace, ensuring correctness and reliability of validation functionalities.

- namespace [SmartStay.Validation.Tests.Validators](#)

The [SmartStay.Validation.Tests.Validators](#) namespace contains unit tests for the validation logic of different fields, specifically focusing on the [AccommodationValidator](#) class for validating accommodation types and IDs.

6.68 PhoneNumberValidatorTests.cs

[Go to the documentation of this file.](#)

```

00001
00010
00015 namespace SmartStay.Validation.Tests.Validators
00016 {
00017     using SmartStay.Validation;
00018     using SmartStay.Validation.Validators;
00019     using Xunit;
00020
00025 public class PhoneNumberValidatorTests
00026 {
00031     [Fact]
00032     public void ValidatePhoneNumber_ValidPhoneNumber_ReturnsPhoneNumber()
00033     {
00034         // Arrange
00035         string validPhoneNumber = "+351777888999";
00036
00037         // Act
00038         var result = PhoneNumberValidator.ValidatePhoneNumber(validPhoneNumber);
00039
00040         // Assert
00041         Assert.Equal(validPhoneNumber, result);
00042     }
00043
00048     [Fact]
00049     public void ValidatePhoneNumber_InvalidPhoneNumber_ThrowsValidationException()
00050     {
00051         // Arrange

```

```

00052         string invalidPhoneNumber = "1234567890"; // Invalid due to missing international code
00053
00054         // Act & Assert
00055         var exception =
00056             Assert.Throws<ValidationException>(() =>
00057                 PhoneNumberValidator.ValidatePhoneNumber(invalidPhoneNumber));
00058         Assert.Equal(ValidationErrorCode.InvalidPhoneNumber, exception.ErrorCode);
00059     }
00060
00061     [Fact]
00062     public void IsValidPhoneNumber_ValidPhoneNumber_ReturnsTrue()
00063     {
00064         // Arrange
00065         string validPhoneNumber = "+1234567890";
00066
00067         // Act
00068         var result = PhoneNumberValidator.IsValidPhoneNumber(validPhoneNumber);
00069
00070         // Assert
00071         Assert.True(result);
00072     }
00073
00074     [Fact]
00075     public void IsValidPhoneNumber_InvalidPhoneNumber_ReturnsFalse()
00076     {
00077         // Arrange
00078         string invalidPhoneNumber = "1234567890"; // Invalid due to missing international code
00079
00080         // Act
00081         var result = PhoneNumberValidator.IsValidPhoneNumber(invalidPhoneNumber);
00082
00083         // Assert
00084         Assert.False(result);
00085     }
00086
00087     [Fact]
00088     public void ValidatePhoneNumber_EmptyPhoneNumber_ThrowsValidationException()
00089     {
00090         // Arrange
00091         string emptyPhoneNumber = "";
00092
00093         // Act & Assert
00094         var exception =
00095             Assert.Throws<ValidationException>(() =>
00096                 PhoneNumberValidator.ValidatePhoneNumber(emptyPhoneNumber));
00097         Assert.Equal(ValidationErrorCode.InvalidPhoneNumber, exception.ErrorCode);
00098     }
00099
00100     [Fact]
00101     public void ValidatePhoneNumber_NullPhoneNumber_ThrowsValidationException()
00102     {
00103         // Arrange
00104         string nullPhoneNumber = null;
00105
00106         // Act & Assert
00107         var exception =
00108             Assert.Throws<ValidationException>(() =>
00109                 PhoneNumberValidator.ValidatePhoneNumber(nullPhoneNumber));
00110         Assert.Equal(ValidationErrorCode.InvalidPhoneNumber, exception.ErrorCode);
00111     }
00112 }
00113
00114 }
00115
00116 }

```

6.69 ReservationValidatorTests.cs File Reference

Data Structures

- class [SmartStay.Validation.Tests.Validators.ReservationValidatorTests](#)

Contains unit tests for the ReservationValidator class. Tests the validation logic for reservation-related data in the SmartStay application.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Validation](#)

- namespace [SmartStay.Validation.Tests](#)

The *SmartStay.Validation.Tests* namespace contains unit tests for classes within the *SmartStay.Validation* namespace, ensuring correctness and reliability of validation functionalities.

- namespace [SmartStay.Validation.Tests.Validators](#)

The *SmartStay.Validation.Tests.Validators* namespace contains unit tests for the validation logic of different fields, specifically focusing on the *AccommodationValidator* class for validating accommodation types and IDs.

6.70 ReservationValidatorTests.cs

[Go to the documentation of this file.](#)

```

00001
00010
00015 namespace SmartStay.Validation.Tests.Validators
00016 {
00017     using SmartStay.Validation;
00018     using SmartStay.Validation.Validators;
00019     using SmartStay.Common.Enums;
00020     using Xunit;
00021
00026 public class ReservationValidatorTests
00027 {
00032     [Fact]
00033     public void ValidateReservationStatus_ValidReservationStatus_ReturnsReservationStatus ()
00034     {
00035         // Arrange
00036         var validStatus = ReservationStatus.Confirmed;
00037
00038         // Act
00039         var result = ReservationValidator.ValidateReservationStatus(validStatus);
00040
00041         // Assert
00042         Assert.Equal(validStatus, result);
00043     }
00044
00049     [Fact]
00050     public void ValidateReservationStatus_InvalidReservationStatus_ThrowsValidationException ()
00051     {
00052         // Arrange
00053         var invalidStatus = (ReservationStatus)999; // Invalid status not defined in the enum
00054
00055         // Act & Assert
00056         var exception =
00057             Assert.Throws<ValidationException>(() =>
00058                 ReservationValidator.ValidateReservationStatus(invalidStatus));
00059         Assert.Equal(ValidationErrorCode.InvalidReservationStatus, exception.ErrorCode);
00060
00065     [Fact]
00066     public void IsValidReservationStatus_ValidReservationStatus_ReturnsTrue ()
00067     {
00068         // Arrange
00069         var validStatus = ReservationStatus.Confirmed;
00070
00071         // Act
00072         var result = ReservationValidator.IsValidReservationStatus(validStatus);
00073
00074         // Assert
00075         Assert.True(result);
00076     }
00077
00082     [Fact]
00083     public void IsValidReservationStatus_InvalidReservationStatus_ReturnsFalse ()
00084     {
00085         // Arrange
00086         var invalidStatus = (ReservationStatus)999; // Invalid status
00087
00088         // Act
00089         var result = ReservationValidator.IsValidReservationStatus(invalidStatus);
00090
00091         // Assert
00092         Assert.False(result);
00093     }
00094
00099     [Fact]
00100     public void ValidateReservationId_ValidReservationId_ReturnsReservationId ()
00101     {
00102         // Arrange

```

```

00103         int validId = 123;
00104
00105         // Act
00106         var result = ReservationValidator.ValidateReservationId(validId);
00107
00108         // Assert
00109         Assert.Equal(validId, result);
00110     }
00111
00112     [Fact]
00113     public void ValidateReservationId_InvalidReservationId_ThrowsValidationException()
00114     {
00115         // Arrange
00116         int invalidId = 0; // Invalid ID (less than or equal to 0)
00117
00118         // Act & Assert
00119         var exception = Assert.Throws<ValidationException>(() =>
00120             ReservationValidator.ValidateReservationId(invalidId));
00121         Assert.Equal(ValidationErrorCode.InvalidId, exception.ErrorCode);
00122     }
00123
00124     [Fact]
00125     public void IsValidReservationId_ValidReservationId_ReturnsTrue()
00126     {
00127         // Arrange
00128         int validId = 123;
00129
00130         // Act
00131         var result = ReservationValidator.IsValidReservationId(validId);
00132
00133         // Assert
00134         Assert.True(result);
00135     }
00136
00137     [Fact]
00138     public void IsValidReservationId_InvalidReservationId_ReturnsFalse()
00139     {
00140         // Arrange
00141         int invalidId = 0; // Invalid ID
00142
00143         // Act
00144         var result = ReservationValidator.IsValidReservationId(invalidId);
00145
00146         // Assert
00147         Assert.False(result);
00148     }
00149 }
00150
00151 }
00152
00153 }
00154
00155 }
00156
00157 }
00158
00159 }
00160
00161 }

```

6.71 RoomValidatorTests.cs File Reference

Data Structures

- class [SmartStay.Validation.Tests.Validators.RoomValidatorTests](#)

Contains unit tests for the RoomValidator class. Tests the validation logic for room-related data used in the [SmartStay](#) application.

Namespaces

- namespace [SmartStay](#)
- namespace [SmartStay.Validation](#)
- namespace [SmartStay.Validation.Tests](#)

The [SmartStay.Validation.Tests](#) namespace contains unit tests for classes within the [SmartStay.Validation](#) namespace, ensuring correctness and reliability of validation functionalities.

- namespace [SmartStay.Validation.Tests.Validators](#)

The [SmartStay.Validation.Tests.Validators](#) namespace contains unit tests for the validation logic of different fields, specifically focusing on the AccommodationValidator class for validating accommodation types and IDs.

6.72 RoomValidatorTests.cs

[Go to the documentation of this file.](#)

```

00001
00010
00015 namespace SmartStay.Validation.Tests.Validators
00016 {
00017     using SmartStay.Validation;
00018     using SmartStay.Validation.Validators;
00019     using SmartStay.Common.Enums;
00020     using Xunit;
00021
00026 public class RoomValidatorTests
00027 {
00032     [Fact]
00033     public void ValidateRoomType_ValidType_ReturnsRoomType()
00034     {
00035         // Arrange
00036         RoomType validRoomType = RoomType.Family;
00037
00038         // Act
00039         var result = RoomValidator.ValidateRoomType(validRoomType);
00040
00041         // Assert
00042         Assert.Equal(validRoomType, result);
00043     }
00044
00049     [Fact]
00050     public void ValidateRoomType_InvalidType_ThrowsValidationException()
00051     {
00052         // Arrange
00053         RoomType invalidRoomType = (RoomType)-1; // An invalid enum value
00054
00055         // Act & Assert
00056         var exception = Assert.Throws<ValidationException>(() =>
RoomValidator.ValidateRoomType(invalidRoomType));
00057         Assert.Equal(ValidationErrorCode.InvalidRoomType, exception.ErrorCode);
00058     }
00059
00064     [Fact]
00065     public void ValidateAvailability_ValidStatus_ReturnsAvailability()
00066     {
00067         // Arrange
00068         bool isAvailable = true;
00069
00070         // Act
00071         var result = RoomValidator.ValidateAvailability(isAvailable);
00072
00073         // Assert
00074         Assert.Equal(isAvailable, result);
00075     }
00076
00081     [Fact]
00082     public void ValidateRoomId_ValidId_ReturnsRoomId()
00083     {
00084         // Arrange
00085         int validRoomId = 101;
00086
00087         // Act
00088         var result = RoomValidator.ValidateRoomId(validRoomId);
00089
00090         // Assert
00091         Assert.Equal(validRoomId, result);
00092     }
00093
00098     [Fact]
00099     public void ValidateRoomId_InvalidId_ThrowsValidationException()
00100     {
00101         // Arrange
00102         int invalidRoomId = -1;
00103
00104         // Act & Assert
00105         var exception = Assert.Throws<ValidationException>(() =>
RoomValidator.ValidateRoomId(invalidRoomId));
00106         Assert.Equal(ValidationErrorCode.InvalidId, exception.ErrorCode);
00107     }
00108
00113     [Fact]
00114     public void IsValidRoomType_ValidType_ReturnsTrue()
00115     {
00116         // Arrange
00117         RoomType validRoomType = RoomType.Deluxe;
00118
00119         // Act
00120         var result = RoomValidator.IsValidRoomType(validRoomType);

```

```
00121
00122     // Assert
00123     Assert.True(result);
00124 }
00125
00130 [Fact]
00131 public void IsValidRoomType_InvalidType_ReturnsFalse()
00132 {
00133     // Arrange
00134     RoomType invalidRoomType = (RoomType)999; // An undefined enum value
00135
00136     // Act
00137     var result = RoomValidator.IsValidRoomType(invalidRoomType);
00138
00139     // Assert
00140     Assert.False(result);
00141 }
00142
00147 [Fact]
00148 public void IsValidAvailability_AnyStatus_ReturnsTrue()
00149 {
00150     // Arrange
00151     bool anyStatus = true;
00152
00153     // Act
00154     var result = RoomValidator.IsValidAvailability(anyStatus);
00155
00156     // Assert
00157     Assert.True(result);
00158
00159     // Repeat with false
00160     result = RoomValidator.IsValidAvailability(false);
00161     Assert.True(result);
00162 }
00163 }
00164 }
```

Index

.NETCoreApp,Version=v8.0.AssemblyAttributes.cs, 96

AccommodationsTests.cs, 97, 98

AccommodationTests.cs, 83

AccommodationValidatorTests.cs, 131

Add_InvalidDateRange_ThrowsValidationException
SmartStay.Core.Tests.Repositories.ReservationsTests, 67

Add_InvalidTotalCost_ThrowsValidationException
SmartStay.Core.Tests.Repositories.ReservationsTests, 67

Add_NullReservation_ThrowsArgumentNullException
SmartStay.Core.Tests.Repositories.ReservationsTests, 67

Add_ValidAccommodation_AddsAccommodationSuccessfully
SmartStay.Core.Tests.Repositories.AccommodationsTests, 16

Add_ValidClient_AddsClientSuccessfully
SmartStay.Core.Tests.Repositories.ClientsTests, 33

Add_ValidOwner_AddsOwnerSuccessfully
SmartStay.Core.Tests.Repositories.OwnersTests, 50

Add_ValidReservation_AddsReservationSuccessfully
SmartStay.Core.Tests.Repositories.ReservationsTests, 67

AddAccommodation_ValidAccommodation_AddsSuccessfully
SmartStay.Core.Tests.Models.OwnerTests, 53

AddReservation_ValidDates_AddsReservation
SmartStay.Core.Tests.Models.RoomTests, 75

AddressValidatorTests.cs, 132, 133

BookingManagerTests.cs, 111

CalculateTotalCost_EndDateBeforeStartDate_ThrowsArgumentOutOfRangeException
SmartStay.Core.Tests.Models.RoomTests, 75

CalculateTotalCost_ValidDates_ReturnsCorrectCost
SmartStay.Core.Tests.Models.RoomTests, 75

CheckIn_StatusNotPending_ReturnsFalse
SmartStay.Core.Tests.Models.ReservationTests, 70

CheckIn_StatusPending_ChangesStatusToCheckedIn
SmartStay.Core.Tests.Models.ReservationTests, 70

CheckOut_StatusCheckedIn_ChangesStatusToCheckedOut
SmartStay.Core.Tests.Models.ReservationTests, 70

CheckOut_StatusNotCheckedIn_ReturnsFalse
SmartStay.Core.Tests.Models.ReservationTests, 71

ClientsTests.cs, 101

ClientTests.cs, 85

ClientValidatorTests.cs, 134

Constructor_AdditionalDetails_InitializesOwner
SmartStay.Core.Tests.Models.OwnerTests, 53

Constructor_BasicDetails_InitializesOwner
SmartStay.Core.Tests.Models.OwnerTests, 53

Constructor_InvalidName_ThrowsValidationException
SmartStay.Core.Tests.Models.AccommodationTests, 18

Constructor_ValidParameters_InitializesAccommodation
SmartStay.Core.Tests.Models.AccommodationTests, 18

Constructor_ValidParameters_InitializesClient
SmartStay.Core.Tests.Models.ClientTests, 36

Constructor_ValidParameters_InitializesReservation
SmartStay.Core.Tests.Models.ReservationTests, 71

Constructor_ValidParameters_InitializesRoom
SmartStay.Core.Tests.Models.RoomTests, 75

Constructor_WithErrorCode_SetsErrorCodeAndMessage
SmartStay.Validation.Tests.ValidationExceptionTests, 81

Constructor_WithUnknownErrorCode_UsesFallbackMessage
SmartStay.Validation.Tests.ValidationExceptionTests, 81

CreateAccommodation_CreatesAccommodation_WhenOwnerExists
SmartStay.Core.Tests.Services.BookingManagerTests, 26

CreateAccommodation_ThrowsEntityNotFoundException_WhenOwnerDoesNotExist
SmartStay.Core.Tests.Services.BookingManagerTests, 26

CreateBasicClient_CreatesClient_WhenValidInput
SmartStay.Core.Tests.Services.BookingManagerTests, 26

CreateBasicClient_ThrowsClientCreationException_WhenValidationFails
SmartStay.Core.Tests.Services.BookingManagerTests, 26

CreateBasicOwner_CreatesOwner_WhenValidDataIsProvided
SmartStay.Core.Tests.Services.BookingManagerTests, 26

CreateBasicOwner_ThrowsException_WhenEmailsInvalid
SmartStay.Core.Tests.Services.BookingManagerTests, 26

CreateCompleteClient_CreatesClient_WhenValidInput
SmartStay.Core.Tests.Services.BookingManagerTests, 27

CreateCompleteClient_ThrowsClientCreationException_WhenValidationFails
SmartStay.Core.Tests.Services.BookingManagerTests, 27

- SmartStay.Core.Tests.Services.BookingManagerTests, 27
- CreateCompleteOwner_CreatesOwner_WhenValidDataIsProvided, 27
- SmartStay.Core.Tests.Services.BookingManagerTests, 27
- CreateCompleteOwner_ThrowsException_WhenPhoneNumberIsInvalid, 27
- SmartStay.Core.Tests.Services.BookingManagerTests, 27
- DateValidatorTests.cs, 135
- Email_SetAndGet_ValidatesValue, 36
- SmartStay.Core.Tests.Models.ClientTests, 36
- SmartStay.Core.Tests.Models.OwnerTests, 53
- EmailValidatorTests.cs, 137
- EnsureDirectoryExists_DirectoryDoesNotExist_CreatesDirectory, 44
- SmartStay.IO.Tests.Extensions.FileExtensionsTests, 44
- EnsureDirectoryExists_DirectoryExists_DoesNotThrowException, 44
- SmartStay.IO.Tests.Extensions.FileExtensionsTests, 44
- EnsureDirectoryExists_NullOrEmptyPath_DoesNotThrowException, 44
- SmartStay.IO.Tests.Extensions.FileExtensionsTests, 44
- EnsureDirectoryExists_RootDirectoryPath_DoesNotThrowException, 44
- SmartStay.IO.Tests.Extensions.FileExtensionsTests, 44
- Export_ValidData_ExportsAccommodations, 16
- SmartStay.Core.Tests.Repositories.AccommodationsTests, 16
- Export_ValidData_ExportsClients, 33
- SmartStay.Core.Tests.Repositories.ClientsTests, 33
- Export_ValidData_ExportsOwnersWithAccommodations, 50
- SmartStay.Core.Tests.Repositories.OwnersTests, 50
- Export_ValidData_ExportsReservationsWithPayments, 68
- SmartStay.Core.Tests.Repositories.ReservationsTests, 68
- FileExists_ExistingFile_ReturnsTrue, 58
- SmartStay.IO.Tests.FileOperations.PathValidatorTests, 58
- FileExists_NonExistentFile_ReturnsFalse, 58
- SmartStay.IO.Tests.FileOperations.PathValidatorTests, 58
- FileExtensionsTests.cs, 122, 123
- FileHandlerTests.cs, 124
- FindAccommodationById_ExistingId_ReturnsAccommodation, 16
- SmartStay.Core.Tests.Repositories.AccommodationsTests, 16
- FindAccommodationById_NonExistingId_ReturnsNull, 16
- SmartStay.Core.Tests.Repositories.AccommodationsTests, 16
- FindClientById_ExistingId_ReturnsClient, 34
- SmartStay.Core.Tests.Repositories.ClientsTests, 34
- FindClientById_NonExistingId_ReturnsNull, 34
- SmartStay.Core.Tests.Repositories.ClientsTests, 34
- FindOwnerById_ExistingId_ReturnsOwner, 50
- SmartStay.Core.Tests.Repositories.OwnersTests, 50
- FindOwnerById_FindsOwner_WhenOwnerExists, 28
- SmartStay.Core.Tests.Services.BookingManagerTests, 28
- FindOwnerById_NonExistingId_ReturnsNull, 51
- SmartStay.Core.Tests.Repositories.OwnersTests, 51
- FindOwnerById_ThrowsException_WhenOwnerDoesNotExist, 28
- SmartStay.Core.Tests.Services.BookingManagerTests, 28
- FindReservationById_ExistingId_ReturnsReservation, 68
- SmartStay.Core.Tests.Repositories.ReservationsTests, 68
- FindReservationById_NonExistingId_ReturnsNull, 68
- SmartStay.Core.Tests.Repositories.ReservationsTests, 68
- FindRoomById_RoomDoesNotExist_ReturnsNull, 19
- SmartStay.Core.Tests.Models.AccommodationTests, 19
- FindRoomById_RoomExists_ReturnsRoom, 19
- SmartStay.Core.Tests.Models.AccommodationTests, 19
- FirstName_SetAndGet_ValidatesValue, 36
- SmartStay.Core.Tests.Models.ClientTests, 36
- SmartStay.Core.Tests.Models.OwnerTests, 53
- GetErrorMessage_InvalidErrorCode_ReturnsFallbackMessage, 79
- SmartStay.Validation.Tests.ValidationErrorMessageTests, 79
- GetErrorMessage_SupportsLocalization, 79
- SmartStay.Validation.Tests.ValidationErrorMessageTests, 79
- GetErrorMessage_ValidErrorCode_ReturnsLocalizedMessage, 80
- SmartStay.Validation.Tests.ValidationErrorMessageTests, 80
- Id_AutoGenerated_IsUniqueAndNonZero, 36
- SmartStay.Core.Tests.Models.ClientTests, 36
- Import_ValidData_ImportsAccommodations, 16
- SmartStay.Core.Tests.Repositories.AccommodationsTests, 16
- Import_ValidData_ImportsClients, 34
- SmartStay.Core.Tests.Repositories.ClientsTests, 34
- Import_ValidData_ImportsOwners, 51
- SmartStay.Core.Tests.Repositories.OwnersTests, 51
- Import_ValidData_ImportsReservationsWithPayments, 51

- SmartStay.Core.Tests.Repositories.ReservationsTests, 68
- IsAvailable_NoOverlap_ReturnsTrue
 - SmartStay.Core.Tests.Models.RoomTests, 75
- IsAvailable_Overlap_ReturnsFalse
 - SmartStay.Core.Tests.Models.RoomTests, 76
- IsFullyPaid_FullyPaid_ReturnsTrue
 - SmartStay.Core.Tests.Models.ReservationTests, 71
- IsValidAccommodationId_InvalidAccommodationId_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.AccommodationValidatorTests, 20
- IsValidAccommodationId_ValidAccommodationId_ReturnsTrue
 - SmartStay.Validation.Tests.Validators.AccommodationValidatorTests, 20
- IsValidAccommodationName_InvalidName_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.NameValidatorTests, 47
- IsValidAccommodationName_ValidName_ReturnsTrue
 - SmartStay.Validation.Tests.Validators.NameValidatorTests, 47
- IsValidAccommodationType_InvalidAccommodationType_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.AccommodationValidatorTests, 21
- IsValidAccommodationType_ValidAccommodationType_ReturnsTrue
 - SmartStay.Validation.Tests.Validators.AccommodationValidatorTests, 21
- IsValidAddress_InvalidAddress_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.AddressValidatorTests, 23
- IsValidAddress_ValidAddress_ReturnsTrue
 - SmartStay.Validation.Tests.Validators.AddressValidatorTests, 23
- IsValidAvailability_AnyStatus_ReturnsTrue
 - SmartStay.Validation.Tests.Validators.RoomValidatorTests, 77
- IsValidClientId_InvalidId_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.ClientValidatorTests, 38
- IsValidClientId_ValidId_ReturnsTrue
 - SmartStay.Validation.Tests.Validators.ClientValidatorTests, 38
- IsValidDateRange_InvalidDateRange_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.DateValidatorTests, 39
- IsValidDateRange_ValidDateRange_ReturnsTrue
 - SmartStay.Validation.Tests.Validators.DateValidatorTests, 39
- IsValidEmail_EmptyEmail_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.EmailValidatorTests, 42
- IsValidEmail_InvalidCharacters_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.EmailValidatorTests, 42
- IsValidEmail_MissingAtSymbol_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.EmailValidatorTests, 42
- IsValidEmail_MissingDomainExtension_ReturnsFalse
- SmartStay.Validation.Tests.Validators.EmailValidatorTests, 42
 - SmartStay.Validation.Tests.Validators.EmailValidatorTests, 42
- IsValidEmail_ValidEmail_ReturnsTrue
 - SmartStay.Validation.Tests.Validators.EmailValidatorTests, 43
- IsValidFileType_CaseInsensitiveExtensionComparison_ReturnsTrue
 - SmartStay.IO.Tests.FileOperations.PathValidatorTests, 58
- IsValidFileType_InvalidExtension_ReturnsFalse
 - SmartStay.IO.Tests.FileOperations.PathValidatorTests, 58
- IsValidFileType_NullOrEmptyFilePath_ThrowsArgumentException
 - SmartStay.IO.Tests.FileOperations.PathValidatorTests, 58
- IsValidFileType_ValidExtension_ReturnsTrue
 - SmartStay.IO.Tests.FileOperations.PathValidatorTests, 58
- IsValidFutureDate_InvalidDate_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.DateValidatorTests, 59
- IsValidFutureDate_ValidDate_ReturnsTrue
 - SmartStay.Validation.Tests.Validators.DateValidatorTests, 59
- IsValidName_InvalidName_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.NameValidatorTests, 48
- IsValidName_ValidName_ReturnsTrue
 - SmartStay.Validation.Tests.Validators.NameValidatorTests, 48
- IsValidPhoneNumber_InvalidPhoneNumber_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests, 65
- IsValidPhoneNumber_ValidPhoneNumber_ReturnsTrue
 - SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests, 65
- IsValidReservationId_InvalidReservationId_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.ReservationValidatorTests, 73
- IsValidReservationId_ValidReservationId_ReturnsTrue
 - SmartStay.Validation.Tests.Validators.ReservationValidatorTests, 73
- IsValidReservationStatus_InvalidReservationStatus_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.ReservationValidatorTests, 73
- IsValidReservationStatus_ValidReservationStatus_ReturnsTrue
 - SmartStay.Validation.Tests.Validators.ReservationValidatorTests, 73
- IsValidRoomType_InvalidType_ReturnsFalse
 - SmartStay.Validation.Tests.Validators.RoomValidatorTests, 77
- IsValidRoomType_ValidType_ReturnsTrue
 - SmartStay.Validation.Tests.Validators.RoomValidatorTests, 77
- LastAssignedId_TracksCorrectly
 - SmartStay.Core.Tests.Models.OwnerTests, 53

- Load_ValidFile_LoadsAccommodations
 - SmartStay.Core.Tests.Repositories.AccommodationsTests,SmartStay.IO.Tests.FileOperations.FileHandlerTests, 16
- Load_ValidFile_LoadsClients
 - SmartStay.Core.Tests.Repositories.ClientsTests, 34
- Load_ValidFile_LoadsOwners
 - SmartStay.Core.Tests.Repositories.OwnersTests, 51
- Load_ValidFile_LoadsReservations
 - SmartStay.Core.Tests.Repositories.ReservationsTests, 68
- MakePayment_ValidPayment_UpdatesAmountPaid
 - SmartStay.Core.Tests.Models.ReservationTests, 71
- NameValidatorTests.cs, 139
- Owner_ToString_ReturnsJson
 - SmartStay.Core.Tests.Models.AccommodationTests, 19
 - SmartStay.Core.Tests.Models.OwnerTests, 54
- OwnerId_InvalidValue_ThrowsException
 - SmartStay.Core.Tests.Models.AccommodationTests, 19
- OwnerId_SetAndGet_CorrectlySetsOwnerId
 - SmartStay.Core.Tests.Models.AccommodationTests, 19
- OwnersTests.cs, 104
- OwnerTests.cs, 87
- OwnerValidatorTests.cs, 141
- PathValidatorTests.cs, 125, 126
- Payment_InvalidAmount_ThrowsValidationException
 - SmartStay.Core.Tests.Models.PaymentTests, 59
- Payment_InvalidReservationId_ThrowsValidationException
 - SmartStay.Core.Tests.Models.PaymentTests, 59
- Payment_ToString_ReturnsJson
 - SmartStay.Core.Tests.Models.ClientTests, 37
 - SmartStay.Core.Tests.Models.PaymentTests, 60
- Payment_UniqueIds_AssignsIncrementalIds
 - SmartStay.Core.Tests.Models.PaymentTests, 60
- Payment_UpdateInvalidStatus_ThrowsValidationException
 - SmartStay.Core.Tests.Models.PaymentTests, 60
- Payment_UpdateValidStatus_UpdatesStatus
 - SmartStay.Core.Tests.Models.PaymentTests, 60
- Payment_ValidData_CreatesPayment
 - SmartStay.Core.Tests.Models.PaymentTests, 60
- PaymentTests.cs, 89, 90
- PaymentValidatorTests.cs, 142, 143
- PhoneNumber_SetAndGet_ValidatesValue
 - SmartStay.Core.Tests.Models.OwnerTests, 54
- PhoneNumberValidatorTests.cs, 145
- PreferredPaymentMethod_SetAndGet_CorrectlyAssignsValue
 - SmartStay.Core.Tests.Models.ClientTests, 37
- ReadFile_EmptyPath_ThrowsArgumentException
 - SmartStay.IO.Tests.FileOperations.FileHandlerTests, 45
- ReadFile_FileDoesNotExist_ThrowsFileNotFoundException
 - SmartStay.IO.Tests.FileOperations.FileHandlerTests, 45
- ReadFile_ValidFilePath_ReturnsFileContent
 - SmartStay.IO.Tests.FileOperations.FileHandlerTests, 46
- Remove_NonExistingAccommodation_ReturnsFalse
 - SmartStay.Core.Tests.Repositories.AccommodationsTests, 17
- Remove_NonExistingClient_ReturnsFalse
 - SmartStay.Core.Tests.Repositories.ClientsTests, 34
- Remove_NonExistingOwner_ReturnsFalse
 - SmartStay.Core.Tests.Repositories.OwnersTests, 51
- Remove_NonExistingReservation_ReturnsFalse
 - SmartStay.Core.Tests.Repositories.ReservationsTests, 69
- Remove_ValidAccommodation_RemovesAccommodationSuccessfully
 - SmartStay.Core.Tests.Repositories.AccommodationsTests, 17
- Remove_ValidClient_RemovesClientSuccessfully
 - SmartStay.Core.Tests.Repositories.ClientsTests, 34
- Remove_ValidOwner_RemovesOwnerSuccessfully
 - SmartStay.Core.Tests.Repositories.OwnersTests, 51
- Remove_ValidReservation_RemovesReservationSuccessfully
 - SmartStay.Core.Tests.Repositories.ReservationsTests, 69
- RemoveAccommodation_ReturnsAccommodationNotFound_WhenAccommodationDoesNotExist
 - SmartStay.Core.Tests.Services.BookingManagerTests, 28
- RemoveAccommodation_ReturnsAccommodationNotFound_WhenAccommodationDoesNotExist_WhenAccommodationDoesNotExist
 - SmartStay.Core.Tests.Services.BookingManagerTests, 28
- RemoveAccommodation_SuccessfullyRemovesAccommodation_WhenAccommodationDoesNotExist
 - SmartStay.Core.Tests.Services.BookingManagerTests, 29
- RemoveAccommodation_ValidAccommodation_RemovesSuccessfully
 - SmartStay.Core.Tests.Models.OwnerTests, 54
- RemoveClient_RemovesClient_WhenClientExists
 - SmartStay.Core.Tests.Services.BookingManagerTests, 29
- RemoveClient_ReturnsFalse_WhenClientDoesNotExist
 - SmartStay.Core.Tests.Services.BookingManagerTests, 29
- RemoveOwner_RemovesOwner_WhenOwnerExists
 - SmartStay.Core.Tests.Services.BookingManagerTests, 29
- RemoveOwner_ReturnsFalse_WhenOwnerDoesNotExist
 - SmartStay.Core.Tests.Services.BookingManagerTests, 29
- RemoveReservation_ExistingReservation_RemovesReservationSuccessfully
 - SmartStay.Core.Tests.Models.RoomTests, 76
- ReservationsTests.cs, 107
- ReservationTests.cs, 91, 92
- ReservationValidatorTests.cs, 146, 147

- RoomTests.cs, 94
- RoomValidatorTests.cs, 148, 149
- Save_ValidData_SavesToFile
 - SmartStay.Core.Tests.Repositories.AccommodationsTests, 17
 - SmartStay.Core.Tests.Repositories.ClientsTests, 35
 - SmartStay.Core.Tests.Repositories.OwnersTests, 51
 - SmartStay.Core.Tests.Repositories.ReservationsTests, 69
- SaveAll_CreatesFiles_WhenCalled
 - SmartStay.Core.Tests.Services.BookingManagerTests, 30
- SmartStay, 7
- SmartStay.Core, 7
- SmartStay.Core.Tests, 7
- SmartStay.Core.Tests.AssemblyInfo.cs, 97
- SmartStay.Core.Tests.GlobalUsings.g.cs, 97
- SmartStay.Core.Tests.Models, 7
- SmartStay.Core.Tests.Models.AccommodationTests, 17
 - Constructor_InvalidName_ThrowsValidationException, 18
 - Constructor_ValidParameters_InitializesAccommodation, 18
 - FindRoomById_RoomDoesNotExist_ReturnsNull, 19
 - FindRoomById_RoomExists_ReturnsRoom, 19
 - Owner_ToString_ReturnsJson, 19
 - OwnerId_InvalidValue_ThrowsException, 19
 - OwnerId_SetAndGet_CorrectlySetsOwnerId, 19
- SmartStay.Core.Tests.Models.ClientTests, 35
 - Constructor_InvalidName_ThrowsValidationException, 36
 - Constructor_ValidParameters_InitializesClient, 36
 - Email_SetAndGet_ValidatesValue, 36
 - FirstName_SetAndGet_ValidatesValue, 36
 - Id_AutoGenerated_IsUniqueAndNonZero, 36
 - Payment_ToString_ReturnsJson, 37
 - PreferredPaymentMethod_SetAndGet_CorrectlyAssignsValue, 37
- SmartStay.Core.Tests.Models.OwnerTests, 52
 - AddAccommodation_ValidAccommodation_AddsSuccessfully, 53
 - Constructor_AdditionalDetails_InitializesOwner, 53
 - Constructor_BasicDetails_InitializesOwner, 53
 - Email_SetAndGet_ValidatesValue, 53
 - FirstName_SetAndGet_ValidatesValue, 53
 - LastAssignedId_TracksCorrectly, 53
 - Owner_ToString_ReturnsJson, 54
 - PhoneNumber_SetAndGet_ValidatesValue, 54
 - RemoveAccommodation_ValidAccommodation_RemovesSuccessfully, 54
- SmartStay.Core.Tests.Models.PaymentTests, 59
 - Payment_InvalidAmount_ThrowsValidationException, 59
 - Payment_InvalidReservationId_ThrowsValidationException, 59
 - Payment_ToString_ReturnsJson, 60
 - Payment_Uniquelds_AssignsIncrementalIds, 60
 - Payment_UpdateInvalidStatus_ThrowsValidationException, 60
 - Payment_UpdateValidStatus_UpdatesStatus, 60
 - Payment_ValidData_CreatesPayment, 60
- SmartStay.Core.Tests.Models.ReservationTests, 69
 - CheckIn_StatusNotPending_ReturnsFalse, 70
 - CheckIn_StatusPending_ChangesStatusToCheckedIn, 70
 - CheckOut_StatusCheckedIn_ChangesStatusToCheckedOut, 70
 - CheckOut_StatusNotCheckedIn_ReturnsFalse, 71
 - Constructor_ValidParameters_InitializesReservation, 71
 - IsFullyPaid_FullyPaid_ReturnsTrue, 71
 - MakePayment_ValidPayment_UpdatesAmountPaid, 71
 - Tostring_ReturnsValidJson, 71
- SmartStay.Core.Tests.Models.RoomTests, 74
 - AddReservation_ValidDates_AddsReservation, 75
 - CalculateTotalCost_EndDateBeforeStartDate_ThrowsArgumentException, 75
 - CalculateTotalCost_ValidDates_ReturnsCorrectCost, 75
 - Constructor_ValidParameters_InitializesRoom, 75
 - IsAvailable_NoOverlap_ReturnsTrue, 75
 - IsAvailable_Overlap_ReturnsFalse, 76
 - RemoveReservation_ExistingReservation_RemovesReservation, 76
 - Tostring_ReturnsValidJson, 76
- SmartStay.Core.Tests.Repositories, 9
- SmartStay.Core.Tests.Repositories.AccommodationsTests, 15
 - Add_ValidAccommodation_AddsAccommodationSuccessfully, 16
 - Export_ValidData_ExportsAccommodations, 16
 - FindAccommodationById_ExistingId_ReturnsAccommodation, 16
 - FindAccommodationById_NonExistingId_ReturnsNull, 16
 - Import_ValidData_ImportsAccommodations, 16
 - Load_ValidFile_LoadsAccommodations, 16
 - Remove_NonExistingAccommodation_ReturnsFalse, 17
 - Remove_ValidAccommodation_RemovesAccommodationSuccessfully, 17
 - Save_ValidData_SavesToFile, 17
- SmartStay.Core.Tests.Repositories.ClientsTests, 33
 - Add_ValidClient_AddsClientSuccessfully, 33
 - Export_ValidData_ExportsClients, 33
 - FindClientById_ExistingId_ReturnsClient, 34
 - FindClientById_NonExistingId_ReturnsNull, 34
 - Import_ValidData_ImportsClients, 34
 - Load_ValidFile_LoadsClients, 34
 - Remove_NonExistingClient_ReturnsFalse, 34
 - Remove_ValidClient_RemovesClientSuccessfully, 34

- Save_ValidData_SavesToFile, [35](#)
- SmartStay.Core.Tests.Repositories.OwnersTests, [49](#)
 - Add_ValidOwner_AddsOwnerSuccessfully, [50](#)
 - Export_ValidData_ExportsOwnersWithAccommodations, [50](#)
 - FindOwnerById_ExistingId_ReturnsOwner, [50](#)
 - FindOwnerById_NonExistingId_ReturnsNull, [51](#)
 - Import_ValidData_ImportsOwners, [51](#)
 - Load_ValidFile_LoadsOwners, [51](#)
 - Remove_NonExistingOwner_ReturnsFalse, [51](#)
 - Remove_ValidOwner_RemovesOwnerSuccessfully, [51](#)
 - Save_ValidData_SavesToFile, [51](#)
- SmartStay.Core.Tests.Repositories.ReservationsTests, [66](#)
 - Add_InvalidDateRange_ThrowsValidationException, [67](#)
 - Add_InvalidTotalCost_ThrowsValidationException, [67](#)
 - Add_NullReservation_ThrowsArgumentNullException, [67](#)
 - Add_ValidReservation_AddsReservationSuccessfully, [67](#)
 - Export_ValidData_ExportsReservationsWithPayments, [68](#)
 - FindReservationById_ExistingId_ReturnsReservation, [68](#)
 - FindReservationById_NonExistingId_ReturnsNull, [68](#)
 - Import_ValidData_ImportsReservationsWithPayments, [68](#)
 - Load_ValidFile_LoadsReservations, [68](#)
 - Remove_NonExistingReservation_ReturnsFalse, [69](#)
 - Remove_ValidReservation_RemovesReservationSuccessfully, [69](#)
 - Save_ValidData_SavesToFile, [69](#)
- SmartStay.Core.Tests.Services, [9](#)
- SmartStay.Core.Tests.Services.BookingManagerTests, [23](#)
 - CreateAccommodation_CreatesAccommodation_WhenOwnerExists, [26](#)
 - CreateAccommodation_ThrowsEntityNotFoundException_WhenOwnerDoesNotExist, [26](#)
 - CreateBasicClient_CreatesClient_WhenValidInput, [26](#)
 - CreateBasicClient_ThrowsClientCreationException_WhenValidationFails, [26](#)
 - CreateBasicOwner_CreatesOwner_WhenValidDataIsProvided, [26](#)
 - CreateBasicOwner_ThrowsException_WhenEmailsInvalid, [26](#)
 - CreateCompleteClient_CreatesClient_WhenValidInput, [27](#)
 - CreateCompleteClient_ThrowsClientCreationException_WhenValidationFails, [27](#)
 - CreateCompleteOwner_CreatesOwner_WhenValidDataIsProvided, [27](#)
 - CreateCompleteOwner_ThrowsException_WhenPhoneNumberIsInvalid, [27](#)
 - FindClientById_ReturnsClient_WhenClientExists, [27](#)
 - FindClientById_ThrowsArgumentException_WhenClientNotFound, [28](#)
 - FindOwnerById_FindsOwner_WhenOwnerExists, [28](#)
 - FindOwnerById_ThrowsException_WhenOwnerDoesNotExist, [28](#)
 - RemoveAccommodation_ReturnsAccommodationNotFound_WhenAccommodationDoesNotExist, [28](#)
 - RemoveAccommodation_ReturnsAccommodationNotFound_WhenAccommodationIsInvalid, [28](#)
 - RemoveAccommodation_SuccessfullyRemovesAccommodation_WhenValid, [29](#)
 - RemoveClient_RemovesClient_WhenClientExists, [29](#)
 - RemoveClient_ReturnsFalse_WhenClientDoesNotExist, [29](#)
 - RemoveOwner_RemovesOwner_WhenOwnerExists, [29](#)
 - RemoveOwner_ReturnsFalse_WhenOwnerDoesNotExist, [29](#)
 - SaveAll_CreatesFiles_WhenCalled, [30](#)
 - UpdateAccommodation_ReturnsAccommodationNotFound_WhenAccommodationDoesNotExist, [30](#)
 - UpdateAccommodation_ReturnsInvalidAddress_WhenInvalidAddressIsProvided, [30](#)
 - UpdateAccommodation_ReturnsInvalidName_WhenInvalidNameIsProvided, [30](#)
 - UpdateAccommodation_UpdatesAccommodation_WhenValidDataIsProvided, [30](#)
 - UpdateClient_ReturnsClientNotFound_WhenClientDoesNotExist, [31](#)
 - UpdateClient_ReturnsInvalidEmail_WhenEmailsInvalid, [31](#)
 - UpdateClient_ReturnsInvalidFirstName_WhenFirstNamesInvalid, [31](#)
 - UpdateClient_ReturnsInvalidLastName_WhenLastNamesInvalid, [31](#)
 - UpdateClient_UpdatesClient_WhenValidData, [31](#)
 - UpdateOwner_ReturnsInvalidEmail_WhenEmailsInvalid, [32](#)
 - UpdateOwner_ReturnsInvalidFirstName_WhenFirstNamesInvalid, [32](#)
 - UpdateOwner_ReturnsInvalidLastName_WhenLastNamesInvalid, [32](#)
 - UpdateOwner_ReturnsOwnerNotFound_WhenOwnerDoesNotExist, [32](#)
 - UpdateOwner_UpdatesOwner_WhenValidDataIsProvided, [32](#)
- SmartStay.IO, [10](#)
- SmartStay.IO.Tests, [10](#)
- SmartStay.IO.Tests.AssemblyInfo.cs, [127](#)
- SmartStay.IO.Tests.Extensions, [10](#)
- SmartStay.IO.Tests.Extensions.FileExtensionsTests, [43](#)
 - EnsureDirectoryExists_DirectoryDoesNotExist_CreatesDirectory, [44](#)
 - EnsureDirectoryExists_DirectoryExists_DoesNotThrowException, [44](#)

- 44
- EnsureDirectoryExists_NullOrEmptyPath_DoesNotThrowException, 44
- EnsureDirectoryExists_RootDirectoryPath_DoesNotThrowException, 44
- SmartStay.IO.Tests.FileOperations, 11
- SmartStay.IO.Tests.FileOperations.FileHandlerTests, 45
 - ReadFile_EmptyPath_ThrowsArgumentException, 45
 - ReadFile_FileDoesNotExist_ThrowsFileNotFoundException, 45
 - ReadFile_ValidFilePath_ReturnsFileContent, 46
 - WriteFile_EmptyPath_ThrowsArgumentException, 46
 - WriteFile_NonExistentDirectory_CreatesDirectoryAndWritesFile, 46
 - WriteFile_ValidFilePath_WritesFileContent, 46
- SmartStay.IO.Tests.FileOperations.PathValidatorTests, 57
 - FileExists_ExistingFile_ReturnsTrue, 58
 - FileExists_NonExistentFile_ReturnsFalse, 58
 - IsValidFileType_CaseInsensitiveExtensionComparison_ReturnsTrue, 58
 - IsValidFileType_InvalidExtension_ReturnsFalse, 58
 - IsValidFileType_NullOrEmptyFilePath_ThrowsArgumentException, 58
 - IsValidFileType_ValidExtension_ReturnsTrue, 58
- SmartStay.IO.Tests.GlobalUsings.g.cs, 128
- SmartStay.Validation, 11
- SmartStay.Validation.Tests, 11
- SmartStay.Validation.Tests.AssemblyInfo.cs, 128
- SmartStay.Validation.Tests.GlobalUsings.g.cs, 128
- SmartStay.Validation.Tests.ValidationErrorMessageTests, 79
 - GetErrorMessage_InvalidErrorCode_ReturnsFallbackMessage, 79
 - GetErrorMessage_SupportsLocalization, 79
 - GetErrorMessage_ValidErrorCode_ReturnsLocalizedMessage, 80
- SmartStay.Validation.Tests.ValidationExceptionTests, 80
 - Constructor_WithErrorCode_SetsErrorCodeAndMessage, 81
 - Constructor_WithUnknownErrorCode_UsesFallbackMessage, 81
- SmartStay.Validation.Tests.Validators, 12
- SmartStay.Validation.Tests.Validators.AccommodationValidatorTests, 20
 - IsValidAccommodationId_InvalidAccommodationId_ReturnsFalse, 20
 - IsValidAccommodationId_ValidAccommodationId_ReturnsTrue, 20
 - IsValidAccommodationType_InvalidAccommodationType_ReturnsFalse, 21
 - IsValidAccommodationType_ValidAccommodationType_ReturnsTrue, 21
 - ValidateAccommodationId_InvalidAccommodationId_ThrowsValidationException, 21
- SmartStay.Validation.Tests.Validators.AddressValidatorTests, 22
 - IsValidAddress_InvalidAddress_ReturnsFalse, 23
 - IsValidAddress_ValidAddress_ReturnsTrue, 23
 - ValidateAddress_InvalidAddress_ThrowsValidationException, 23
 - ValidateAddress_ValidAddress_ReturnsAddress, 23
- SmartStay.Validation.Tests.Validators.ClientValidatorTests, 37
 - IsValidClientId_InvalidId_ReturnsFalse, 38
 - IsValidClientId_ValidId_ReturnsTrue, 38
 - ValidateClientId_InvalidId_ThrowsValidationException, 38
 - ValidateClientId_ValidId_ReturnsClientId, 38
- SmartStay.Validation.Tests.Validators.DateValidatorTests, 39
 - IsValidDateRange_InvalidDateRange_ReturnsFalse, 39
 - IsValidDateRange_ValidDateRange_ReturnsTrue, 39
 - IsValidFutureDate_InvalidDate_ReturnsFalse, 40
 - IsValidFutureDate_ValidDate_ReturnsTrue, 40
 - ValidateCheckInDate_InvalidDate_ThrowsValidationException, 40
 - ValidateCheckInDate_ValidDate_ReturnsCheckInDate, 40
 - ValidateCheckOutDate_InvalidDateRange_ThrowsValidationException, 40
 - ValidateCheckOutDate_ValidDateRange_ReturnsCheckOutDate, 41
- SmartStay.Validation.Tests.Validators.EmailValidatorTests, 42
 - IsValidEmail_EmptyEmail_ReturnsFalse, 42
 - IsValidEmail_InvalidCharacters_ReturnsFalse, 42
 - IsValidEmail_MissingAtSymbol_ReturnsFalse, 42
 - IsValidEmail_MissingDomainExtension_ReturnsFalse, 42
 - IsValidEmail_NullEmail_ReturnsFalse, 42
 - IsValidEmail_ValidEmail_ReturnsTrue, 43
 - ValidateEmail_InvalidEmail_ThrowsValidationException, 43
 - ValidateEmail_ValidEmail_ReturnsEmail, 43
- SmartStay.Validation.Tests.Validators.NameValidatorTests, 45
 - IsValidAccommodationName_InvalidName_ReturnsFalse, 45
 - IsValidAccommodationName_ValidName_ReturnsTrue, 45
 - IsValidName_InvalidName_ReturnsFalse, 48
 - IsValidName_ValidName_ReturnsTrue, 48
 - ValidateAccommodationName_TooLongName_ThrowsValidationException, 48

[48](#)
 ValidateAccommodationName_ValidName_ReturnsName, [48](#)
 ValidateName_InvalidName_ThrowsValidationException, [48](#)
 ValidateName_TooLongName_ThrowsValidationException, [49](#)
 ValidateName_ValidName_ReturnsName, [49](#)
 SmartStay.Validation.Tests.Validators.OwnerValidatorTests, [54](#)
 ValidateOwnerEmail_InvalidEmail_ThrowsValidationException, [55](#)
 ValidateOwnerEmail_ValidEmail_ReturnsEmail, [55](#)
 ValidateOwnerId_InvalidId_ThrowsValidationException, [55](#)
 ValidateOwnerId_ValidId_ReturnsId, [56](#)
 ValidateOwnerName_InvalidName_ThrowsValidationException, [56](#)
 ValidateOwnerName_ValidName_ReturnsName, [56](#)
 ValidateOwnerPhoneNumber_InvalidPhoneNumber_ThrowsValidationException, [56](#)
 ValidateOwnerPhoneNumber_ValidPhoneNumber_ReturnsPhoneNumber, [56](#)
 SmartStay.Validation.Tests.Validators.PaymentValidatorTests, [61](#)
 ValidatePayment_InvalidPayment_ThrowsValidationException, [62](#)
 ValidatePayment_ValidPayment_ReturnsPayment, [62](#)
 ValidatePaymentAmount_InvalidAmount_ThrowsValidationException, [62](#)
 ValidatePaymentAmount_ValidAmount_ReturnsAmount, [62](#)
 ValidatePaymentMethod_InvalidMethod_ThrowsValidationException, [62](#)
 ValidatePaymentMethod_ValidMethod_ReturnsMethod, [63](#)
 ValidatePaymentStatus_InvalidStatus_ThrowsValidationException, [63](#)
 ValidatePaymentStatus_ValidStatus_ReturnsStatus, [63](#)
 ValidatePrice_InvalidPrice_ThrowsValidationException, [63](#)
 ValidatePrice_ValidPrice_ReturnsPrice, [63](#)
 ValidateTotalCost_InvalidTotalCost_ThrowsValidationException, [64](#)
 ValidateTotalCost_ValidTotalCost_ReturnsTotalCost, [64](#)
 SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests, [64](#)
 IsValidPhoneNumber_InvalidPhoneNumber_ReturnsFalse, [65](#)
 IsValidPhoneNumber_ValidPhoneNumber_ReturnsTrue, [65](#)
 ValidatePhoneNumber_EmptyPhoneNumber_ThrowsValidationException, [65](#)
 ValidatePhoneNumber_InvalidPhoneNumber_ThrowsValidationException, [65](#)
 ValidatePhoneNumber_NullPhoneNumber_ThrowsValidationException, [65](#)
 ValidatePhoneNumber_ValidPhoneNumber_ReturnsPhoneNumber, [66](#)
 SmartStay.Validation.Tests.Validators.ReservationValidatorTests, [72](#)
 IsValidReservationId_InvalidReservationId_ReturnsFalse, [73](#)
 IsValidReservationId_ValidReservationId_ReturnsTrue, [73](#)
 IsValidReservationStatus_InvalidReservationStatus_ReturnsFalse, [73](#)
 IsValidReservationStatus_ValidReservationStatus_ReturnsTrue, [73](#)
 ValidateReservationId_InvalidReservationId_ThrowsValidationException, [73](#)
 ValidateReservationId_ValidReservationId_ReturnsReservationId, [73](#)
 ValidateReservationStatus_InvalidReservationStatus_ThrowsValidationException, [73](#)
 ValidateReservationStatus_ValidReservationStatus_ReturnsReservationId, [73](#)
 SmartStay.Validation.Tests.Validators.RoomValidatorTests, [76](#)
 IsValidAvailability_AnyStatus_ReturnsTrue, [77](#)
 IsValidRoomType_InvalidType_ReturnsFalse, [77](#)
 IsValidRoomType_ValidType_ReturnsTrue, [77](#)
 ValidateAvailability_ValidStatus_ReturnsAvailability, [78](#)
 ValidateRoomId_InvalidId_ThrowsValidationException, [78](#)
 ValidateRoomId_ValidId_ReturnsRoomId, [78](#)
 ValidateRoomType_InvalidType_ThrowsValidationException, [78](#)
 ValidateRoomType_ValidType_ReturnsRoomType, [78](#)
 ToString_ReturnsValidJson
 SmartStay.Core.Tests.Models.ReservationTests, [71](#)
 SmartStay.Core.Tests.Models.RoomTests, [76](#)
 UpdateAccommodation_ReturnsAccommodationNotFound_WhenAccommodationNotFound, [30](#)
 SmartStay.Core.Tests.Services.BookingManagerTests, [30](#)
 UpdateAccommodation_ReturnsInvalidAddress_WhenInvalidAddressProvided, [30](#)
 SmartStay.Core.Tests.Services.BookingManagerTests, [30](#)
 UpdateAccommodation_ReturnsInvalidName_WhenInvalidNameProvided, [30](#)
 SmartStay.Core.Tests.Services.BookingManagerTests, [30](#)
 UpdateAccommodation_UpdatesAccommodation_WhenValidDataProvided, [30](#)
 SmartStay.Core.Tests.Services.BookingManagerTests, [30](#)
 UpdateClient_ReturnsClientNotFound_WhenClientDoesNotExist, [31](#)
 SmartStay.Core.Tests.Services.BookingManagerTests, [31](#)
 UpdateClient_ReturnsInvalidEmail_WhenEmailIsInvalid

- 62
- ValidatePaymentAmount_ValidAmount_ReturnsAmount SmartStay.Validation.Tests.Validators.PaymentValidatorTests, 64
- 62
- ValidatePaymentMethod_InvalidMethod_ThrowsValidationException SmartStay.Validation.Tests.Validators.PaymentValidatorTests, 64
- 62
- ValidatePaymentMethod_ValidMethod_ReturnsMethod SmartStay.Validation.Tests.Validators.PaymentValidatorTests, 63
- 63
- ValidatePaymentStatus_InvalidStatus_ThrowsValidationException SmartStay.Validation.Tests.Validators.PaymentValidatorTests, 46
- 63
- ValidatePaymentStatus_ValidStatus_ReturnsStatus SmartStay.Validation.Tests.Validators.PaymentValidatorTests, 46
- 63
- ValidatePhoneNumber_EmptyPhoneNumber_ThrowsValidationException SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests, 46
- 65
- ValidatePhoneNumber_InvalidPhoneNumber_ThrowsValidationException SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests, 65
- 65
- ValidatePhoneNumber_NullPhoneNumber_ThrowsValidationException SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests, 65
- 65
- ValidatePhoneNumber_ValidPhoneNumber_ReturnsPhoneNumber SmartStay.Validation.Tests.Validators.PhoneNumberValidatorTests, 66
- 66
- ValidatePrice_InvalidPrice_ThrowsValidationException SmartStay.Validation.Tests.Validators.PaymentValidatorTests, 63
- 63
- ValidatePrice_ValidPrice_ReturnsPrice SmartStay.Validation.Tests.Validators.PaymentValidatorTests, 63
- 63
- ValidateReservationId_InvalidReservationId_ThrowsValidationException SmartStay.Validation.Tests.Validators.ReservationValidatorTests, 73
- 73
- ValidateReservationId_ValidReservationId_ReturnsReservationId SmartStay.Validation.Tests.Validators.ReservationValidatorTests, 73
- 73
- ValidateReservationStatus_InvalidReservationStatus_ThrowsValidationException SmartStay.Validation.Tests.Validators.ReservationValidatorTests, 74
- 74
- ValidateReservationStatus_ValidReservationStatus_ReturnsReservationStatus SmartStay.Validation.Tests.Validators.ReservationValidatorTests, 74
- 74
- ValidateRoomId_InvalidId_ThrowsValidationException SmartStay.Validation.Tests.Validators.RoomValidatorTests, 78
- 78
- ValidateRoomId_ValidId_ReturnsRoomId SmartStay.Validation.Tests.Validators.RoomValidatorTests, 78
- 78
- ValidateRoomType_InvalidType_ThrowsValidationException SmartStay.Validation.Tests.Validators.RoomValidatorTests, 78
- 78
- ValidateRoomType_ValidType_ReturnsRoomType SmartStay.Validation.Tests.Validators.RoomValidatorTests, 78
- 78
- ValidateTotalCost_InvalidTotalCost_ThrowsValidationException SmartStay.Validation.Tests.Validators.PaymentValidatorTests, 64
- ValidateTotalCost_ValidTotalCost_ReturnsTotalCost SmartStay.Validation.Tests.Validators.PaymentValidatorTests, 64
- ValidationErrorMessageTests.cs, 129
- ValidationExceptionTests.cs, 130
- WriteFile_EmptyPath_ThrowsArgumentException SmartStay.IO.Tests.FileOperations.FileHandlerTests, 46
- WriteFile_NonExistentDirectory_CreatesDirectoryAndWritesFile SmartStay.IO.Tests.FileOperations.FileHandlerTests, 46
- WriteFile_ValidFilePath_WritesFileContent SmartStay.IO.Tests.FileOperations.FileHandlerTests, 46