The PropMan Project Manual

Last revised on: September 22, 2024
Rev. 1
By: Sergejs Ponomarenko

Table of Contents

1. Overview

This section provides an overall description of the project, its essence, current state and the domain of application, along with some details regarding limitations of the project's scope and potential areas of improvement.

This manual is intended for anyone who is going to revise and/or test the PropMan project.

The code is available at https://github.com/SergejsP84/PropManProject/tree/master.

The PropMan software product has been developed by the author as the graduation project in the course of his studies at Tel-Ran. The name "PropMan" is simply an abbreviation for "Property Management", whereas the actual name under which the product would operate if deployed can be chosen by its administrator during setup. The general objective was to develop a robust backend system for a specific industry; the author's choice of subject was a real estate management platform. Given the time constraints and the anticipated scope of efforts, it was decided to restrict the subject matter to property rentals. Therefore, the ultimate goal of the project was defined as the development of a backend component for a property rentals platform, somewhat akin to Airbnb.com or Booking.com.

That being the case, the functionality of the PropMan software product mimics that of the aforementioned platforms to an extent. The core business process is focused at giving potential tenants an opportunity to rent properties for relatively short time periods from managers, who in turn are provided with the functionality required in order to post their properties for rent and receive profit from such rentals. Appurtenant process descriptions and more details on the core functions are available in Section 4 "User types and system operation" - without considering scheduled service tasks, entity generation and testing facilities, the project offers 99 different user functions - a premade collection of Postman requests is included in the scope of the project, and can be used to test these endpoints.

Whereas such functionality implies processing of payments, the author had to pay particular attention to the issues of security, such as the storage of users' payment card details. However, it should be noted that the author did not intend to develop an actually functional payment mechanism, as this would obviously exceed the author's current capacity in terms of time and effort. With that in mind, the functions involving payments have been develop using "stub" endpoints, which return a simple Boolean value when triggered. Nevertheless, the respective methods are in place to extract, decrypt and furnish the required payments details, in order to enable the connection of the system to a third party payment service provider.

It should be noted that, as of the last revision date of this manual, the project contains no unit tests; the author had to remove the older ones that have become completely outdated as the structure of the project was evolving. In consideration of the applicable schedules and time constraints, the priority remains defined as streamlining the existing functionality. In lieu of that, the project can be inspected by means of:

- Launching the program from the IntelliJIDEA development environment. This would require a certain batch of preparatory actions on part of the inspecting person, which is described in Section 2 "Setup". This provides access to the functional endpoints using either
    - a premade collection of Postman requests, available for download here, or
    - the Swagger interface, accessible by means of starting the program and typing/copying "**http://localhost:8080/swagger-ui/index.html**" into the address bar of a web browser.
- Using the containerized version of the project, produced using Docker. It is still highly advisable to run the program from IDEA first, so that it could property generate basic data, which requires console input as per the current setup.

In conclusion of this introductory section, the author would like to state that there is room for improvement, of course; for instance, the code can be refactored further for great efficiency, especially when working with larger databases. Given the time constraints, however, this remains a post-defense objective. The code still contains several unused and/or discarded entity fields, methods and endpoints; this is due to the fact that the project had been reshaped several times during development. These shall also be cleaned or repurposed in the course of further post-defense refactoring. Other potential areas of improvement are identified in Section 6 "Further improvement opportunities". Any subsequent changes will be addressed in further revisions of this Manual.

2. Setup

This section lists the actions one should take in order to enable the launch of the program for testing purposes.

a) Setting the environment variables

First and foremost, the program relies on environment variables for some settings that are vital for the full functionality of the project to become operational. To ease things up, the author has prepared a *propman_env_setup.bat* file to set all of these variables at once, available for download here; however, you should set those values manually, and only run the file after these are set up. There is a total of 10 environment variables to be set.

The content of the file is as follows:

```
@echo off
SETLOCAL

REM Set environment variables temporarily (for the current session)
set "AES_SECRET_KEY=your_aes_secret_key"
set "JWT_SECRET_KEY=your_jwt_secret_key"
set "PROPMAN_DB_PASSWORD=your_db_password"
set "PROPMAN_DB_URL=your_mysql_database_address"
set "PROPMAN_DB_USERNAME=your_db_username"
set "PROPMAN_MAIL_HOST=your_mail_server"
set "PROPMAN_MAIL_PORT=your_mail_port"
set "PROPMAN_MAIL_USERNAME=your_email_address"
set "SPRING_MAIL_PASSWORD=your_spring_mail_password"
set "PROPMAN_PLATFORM_NAME=YourPlatformName"

REM Set environment variables permanently
setx AES_SECRET_KEY "your_aes_secret_key"
setx JWT_SECRET_KEY "your_jwt_secret_key"
setx PROPMAN_DB_PASSWORD "your_db_password"
setx PROPMAN_DB_URL "your_mysql_database_address"
setx PROPMAN_DB_USERNAME "your_db_username"
setx PROPMAN_MAIL_HOST "your_mail_server"
setx PROPMAN_MAIL_PORT "your_mail_port"
setx PROPMAN_MAIL_USERNAME "your_email_address"
setx SPRING_MAIL_PASSWORD "your_spring_mail_password"
setx PROPMAN_PLATFORM_NAME "YourPlatformName"

REM Display the variables to confirm they are set
echo SPRING_MAIL_PASSWORD=%SPRING_MAIL_PASSWORD%
echo AES_SECRET_KEY=%AES_SECRET_KEY%
echo JWT_SECRET_KEY=%JWT_SECRET_KEY%
echo PROPMAN_DB_PASSWORD=%PROPMAN_DB_PASSWORD%
echo PROPMAN_DB_URL=%PROPMAN_DB_URL%
echo PROPMAN_DB_USERNAME=%PROPMAN_DB_USERNAME%
echo PROPMAN_MAIL_HOST=%PROPMAN_MAIL_HOST%
echo PROPMAN_MAIL_PORT=%PROPMAN_MAIL_PORT%
echo PROPMAN_MAIL_USERNAME=%PROPMAN_MAIL_USERNAME%
echo PROPMAN_PLATFORM_NAME=%PROPMAN_PLATFORM_NAME%

ENDLOCAL
pause
```

Thus, in order to run the program, you would have to set the following values (please note that you are going to have to do it twice, for the current session block and for permanent setting block):

No.	Fragment to substitute with a value	Description
1.	your_aes_secret_key	An AES Secret Key. This is required as some values in the database are encrypted using the AES algorithm.
2.	your_jwt_secret_key	A JWT Secret Key. JWT tokens are used in the UserResponse DQTO, hence the necessity of this one.
3.	your_db_password	A password for the respective user of your SQL database, required in order to give the program access thereto.
4.	your_mysql_database_address	Database address, typically something like jdbc:mysql://localhost:3306/destiny_database.
5.	your_db_username	A username (login) for the respective user of your SQL database, required in order to give the program access thereto.
6.	your_mail_server	The server for the email address you are going to use as system mail
7.	your_mail_port	Your mail port (typically something like 587)
8.	your_email_address	The actual email address you are going to use as system mail
9.	your_spring_mail_password	The password to the abovementioned email address
10.	YourPlatformName	The name under which the platform shall "operate" (feel free to set any one you like)

Once you are done substituting these values, please save and close the bat file and run it. You should see the list of variables displayed.

b) Running the program from IntelliJIDEA environment

For testing purposes, however, it might be more convenient and flexible to start the program from the IntelliJIDEA development environment. Please note that you are still going to have to set the environment variables listed above in order for the program to run.

Simply pull the program from https://github.com/SergejsP84/PropManProject.git to your system, and run it. On first startup, you will be prompted in the console to change the DefaultAdmin's password and enter the DefaultAdmin's email. Please make sure to remember the password for the DefaultAdmin user, as this is designed to persist in case of database regeneration.

Afterwards, the console will ask you if you would like to generate a sample database. If you are running the program to test its functionality, it is advised that you generate such sample database: Testing is far more convenient in an already populated environment; besides, the Postman request collection, also provided for testing purposes and addressed further in this section, is designed to function with the pre-generated sample database.

Having done that, you will have the system ready for operation. One last recommended thing to do is to download the Postman request collection referred to above, which is available here. Having imported it to Postman, you should be able to start the familiarization and testing process.

c) Running a containerized version of the program

The project contains the facilities required in order to set up a containerized version thereof using Docker. Please note that it is still highly advisable to run the project from the IDEA environment at least twice first (or once, if you choose not to generate a test database at the first run or manually set the value for the NumericalConfig “ProposeSampleDatabaseCreation” in the database created after the first run). Otherwise, the console input that the program would require during these first runs may interfere with proper creation of the container. Trying to containerize the project without these initial steps would also preclude the generation of a test database.

After the initial console-based configuration is done, you may generate a Docker container using the “**docker compose up --build**” command from IDEA’s GitBash Terminal.

d) Other information

A Swagger annotation set has also been developed to provide another familiarization opportunity. To access the Swagger interface, please run the program and type http://localhost:8080/swagger-ui/index.html in the address bar of your browser. You are going to need the access credentials of your DefaultAdmin or another user with the sought role

Here are the **login and password values** for auto-generated users from the test database:

User ID	LOGIN	PASSWORD
Admins		
1.	DefaultAdmin	<The password you are going to set>
2.	AnotherAdmin	AdminPassword123
Managers		
1.	SNettleson	Ductus
2.	SUllman	HoraceDerwent
3.	CPaul	HeavensDoor
4.	GGolden	Nosferatu
5.	LitaC	SanookMakMak
Tenants		
1.	KennyM	CheatingDeath
2.	StanM	PinewoodDerby
3.	KyleB	Jewpacabra
4.	EricC	RespectMyAuth
5.	WendyT	ScienceRules
6.	ButtersS	AwGeeWhiz
7.	TolkienB	TokenOfHope
8.	CraigT	TouchTheSky
9.	TweekT	StayAwake

10.	GregoryY	ClassyTouch
11.	BebeS	BestFriend
12.	RebeccaR	RedHair
13.	ClydeD	Mysterion
14.	AnnieK	Butterfly

It should be noted here that the generated “sample” entities do not follow the usual creation course: specifically, there is no enforcement of stronger passwords (some “users” do have very simple ones, in fact), and many user entities share the same email address. An email address is a mandatory field of all user entities, Normally, the system does not allow registration of users with identical email addresses within the boundaries of the same user role, but for testing purposes, most generated users do share the same email. This has been done on purpose, for greater simplicity of the testing process. Two user email accounts have been create for testing purposes, and you can use them at the www.inbox.lv platform.

Email 1: propman_testmail1@inbox.lv
Login: propman_testmail1
Password: <please request this from the author via Discord>

Email 2: propman_testmail2@inbox.lv
Login: propman_testmail2
Password: <please request this from the author via Discord>

3. Entities and database schema

This section provides a description of entities within the project and contains a database schema to illustrate the mutual relations.

There is a total of 26 entities in the system. The entities “dwelling” in the system are as follows, and can in turn be categorized into five groups: User Entities, Rental Process Entities, Finance-Related Entities, Communication Entities and Auxiliary Entities.

I. User entities:

1) Admin

An Admin is responsible for overseeing the operation of the platform, and has access to a range of tools for changing the system’s settings, creating entities such as Refunds or Payouts that would normally be generated automatically, and otherwise intervene in the platform’s operation as the situation may require, up to and including creating, blocking and deleting Properties, Tenants or Managers. There is also a hardcoded user entity named DefaultAdmin, which cannot be deleted by normal means and has the authority to create and delete accounts of other Admins. The functions of an Admin are covered in more detail in Section 4 “User types and system operation”.

Fields of an Admin entity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity

2.	name	String	Full name of the Admin
3.	login	String	Must be unique across the system, REQUIRED
4.	password	String	REQUIRED, gets encoded after entry, password requirements NOT enforced for Admins
5.	authorities	Collection	Linked to another table, admin_authorities
6.	knownIps	List	IP addresses from which this user ever logged in
7.	email	String	The user’s email address

2) Manager
A Manager is a user who uses the platform as a means to propose and rent out his/her Property at the platform. As such, he/she is granted with the authorities to manage his/her Properties in a range of ways, confirm Booking requests made by Tenants, manage his/her own profile, including access credentials, etc. A Manager can have multiple Properties, and can also sign up as a Tenant with the same email (but a different login). As with the previous entity, the functions of a Manager are covered in more detail in Section 4 “User types and system operation”.

Fields of a Manager entity:

No.	**Field name**	**Field type**	**Comment**
1.	id	Long	ID of the entity
2.	type	ManagerType	Enumerated: can be PRIVATE or CORPORATE
3.	managerName	String	The manager’s first name / surname or company name, REQUIRED
4.	description	String	Can be used by the Manager to provide a self-description
5.	isActive	Boolean	A Manager must be active in order for his/her Properties to be available for rent
6.	joindate	Timestamp	Specifies the time when this Manager joined the platform
7.	login	String	Must be unique across the system, REQUIRED
8.	password	String	REQUIRED, must be at least 8 characters long, contain at least one uppercase letter / lowercase letter / digit, encrypted for storage
9.	properties	Set	Stores this Manager’s Properties
10.	phone	String	The Manager’s telephone number
11.	email	String	The user’s email address
12.	iban	String	The user’s IBAN. Not used in the current setup, but intended for use if the payment functionality ever gets expanded
13.	paymentCardNo	String	The user’s payment card

			number, stored in an encrypted format, keys stored in a separate file
14.	cardValidityDate	YearMonth	The user’s card validity expiration month
15.	cvv	char array	The user’s CVV, stored as a char array and further encrypted just like the payment card number
16.	confirmationToken	String	A token used to confirm the user’s email address in the course of registration
17.	expirationTime	LocalDateTime	Expiration time of the aforementioned token
18.	authorities	Collection	Linked to another table, manager_authorities
19.	knownIps	List	IP addresses from which this user ever logged in
20.	temporaryEmail	String	Used when the user wishes to change his/her email address
21.	temporaryPassword	String	Used when the user wishes to change his/her password

3) Tenant
A Tenant is a user who uses the platform for renting Properties to reside in for a specific period. Therefore, this is the user who initiates Bookings and makes TenantPayments. A Tenant can also manage his/her own profile, including access credentials, etc. Other Tenant functions include leaving Reviews, communicating with Messages, filing Claims and more. A Tenant can at the same time also be a Manager with the same email (but a different login). As with the previous entity, the functions of a Tenant are covered in more detail in Section 4 “User types and system operation”.

Fields of a Tenant entity:

No.	**Field name**	**Field type**	**Comment**
1.	id	Long	ID of the entity
2.	firstName	String	The Tenant’s first name
3.	lastName	String	The Tenant’s surname
4.	currentProperty	Property	The Property where the Tenant resides at right now
5.	isActive	Boolean	A Tenant must be active in order to make Bookings
6.	phone	String	The Tenant’s telephone number
7.	email	String	The user’s email address
8.	iban	String	The user’s IBAN. Not used in the current setup, but intended for use if the payment functionality ever gets expanded
9.	paymentCardNo	String	The user’s payment card number, stored in an encrypted format, keys stored in a

			separate file
10.	cardValidityDate	YearMonth	The user’s card validity expiration month
11.	cvv	char array	The user’s CVV, stored as a char array and further encrypted just like the payment card number
12.	rating	float	A Tenant’s rating composed of the numerous TenantRatings affixed by Managers of the Properties where this Tenant ever stayed
13.	login	String	Must be unique across the system, REQUIRED
14.	password	String	REQUIRED, must be at least 8 characters long, contain at least one uppercase letter / lowercase letter / digit, encrypted for storage
9.	leasingHistories	List	Stores a history of the Tenant’s past Bookings
10.	tenantPayments	Set	Payments made by the Tenant
11.	confirmationToken	String	A token used to confirm the user’s email address in the course of registration
12.	expirationTime	LocalDateTime	Expiration time of the aforementioned token
13.	preferredCurrency	Currency	The Currency that the Tenant likes to see the prices and payment amounts in
14.	authorities	Collection	Linked to another table, tenant_authorities
15.	knownIps	List	IP addresses from which this user ever logged in
16.	temporaryEmail	String	Used when the user wishes to change his/her email address
17.	temporaryPassword	String	Used when the user wishes to change his/her password

II. Rental process entities:

1) Property

A Property is a real estate, a property offered by a Manager via the platform and available to Tenants for rent. Properties come at a variety pf types, locations and other numerous parameters. Needless to say, this is one of the core entities within the system.

Fields of a Property entity:

No.	**Field name**	**Field type**	**Comment**
1.	id	Long	ID of the entity
2.	manager	Manager	The Manager of this Property
3.	propertyStatus	PropertyStatus	An Enum: basically, it is Available or Blocked, as the Busy status turned out to be not

			exactly appropriate for the final setup
4.	createdAt	Timestamp	The time when the Proeprty was created. Not used in the current setup, but may come in handy if the functionality is ever expanded with promo campaigns for new Properties or something like that
5.	type	PropertyType	Enumerated: a Property can be a Room, a Hotel Room, an Apartment, a House, a Commercial property or an Other kind of real estate
6.	address	String	Would typically include the street name and the house/apartment number
7.	country	String	The country where the Property is located
8.	settlement	String	Settlement name – a city, village, etc.
9.	sizeM2	Float	The floorspace of the Property
10.	description	String	A description of the Property as assigned by its Manager
11.	rating	Float	The overall rating of the Property based on previous PropertyRatings
12.	pricePerDay	Double	Used to calculate the price of a Booking that is shorter than one week or the price of extra days when the Booking is made for longer than a week
13.	pricePerWeek	Double	Used to calculate the price of a Booking that is longer than 6 days, but does not comprise an entire month
14.	pricePerMonth	Double	Used to calculate the price of an entire 30-day period
15.	bills	Set	A set of Bills associated with the Property
16.	tenant	Tenant	The current Tenant residing at the Property
17.	photos	List	A list of links to the photos of this specific Property

2) Booking

The major element of the system, a Booking is the pivot of the entire range of business processes facilitated by the software product. It is created whenever a Tenant books a Property, and persists until this entire transaction is over one way or another.

Fields of a Booking entity:

No.	**Field name**	**Field type**	**Comment**

1.	id	Long	ID of the entity
2.	property	Property	The Property for which the Booking has been made
3.	tenantId	Long	The ID of the Tenant who made the Booking
4.	startDate	Timestamp	The date when the Booking starts… Could have used LocalDate for this purpose, but did not understand the implications back then. REQUIRED
5.	endDate	Timestamp	Same as above, but defines the end date of the Booking
6.	isPaid	Boolean	Signifies whether this particular Booking has been paid
7.	status	BookingStatus	An Enum vital for a number of processes. Can be: PENDING_APPROVAL - Booking not yet confirmed by the Property's Manager PENDING_PAYMENT - Booking approved, but not yet paid CONFIRMED - Booking paid by the Tenant CURRENT - the Tenant is staying at the Property right now CANCELLED - the Booking has been cancelled by the Tenant or automatically due to the Tenant's failure to pay OVER - the Booking has been successfully completed, and the Claim period is going on right now PROPERTY_LOCKED_BY_MANAGER - a special status for cases whenever a Manager wishes to lock a property for some time (for thatpurpose, the system creates a "stub Booking" of sorts)

3) Amenity
Any Amenity that a Property might have. These are added to the system by Admins, and assigned to Properties by their respective Managers.

Fields of an Amenity entity:

No.	**Field name**	**Field type**	**Comment**
1.	id	Long	ID of the entity
2.	description	String	REQUIRED. A short essential description of an Amenity – like “pool”, “AC”, etc.

4) LeasingHistory
Kind of like a “storage entity”. In order to avoid overloading the system with zillions of Bookings that had been completed a long time ago, a new LeasingHistory entity is created at the end of a Booking’s Claim period. The

Booking itself is removed from the system, and a record thereof is stored as a LeasingHistory.

Fields of a LeasingHistory entity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity
2.	tenant	Tenant	Stores the entire Tenant entity – an ID could be enough, but the author decided not to remake the system at this point due to time constraints
3.	propertyId	Long	This stores the ID of the property that was booked
4,	startDate	Timestamp	The start time of the underlying Booking
5,	endDate	Timestamp	The end time of the underlying Booking

5) TenantFavorites
As the name implies, this is the entity facilitating the storage and retrieval of a Tenant's favorite Properties.

Fields of a TenantFavorites entity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity
2.	tenantId	Long	Stores the ID of the Tenant whose favorite Properties it refers to
3.	favoritePropertyIDs	List	Stores the IDs of the Properties that the Tenant has marked as his/her favorites

6) EarlyTerminationRequest
This is an entity made especially for the cases when a Tenant needs to end his/her Booking prematurely. This can be accepted or denied by the respective Manager, as within this specific platform.

Fields of an EarlyTerminationRequest entity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity
2.	tenantId	Long	ID of the Tenant requesting an early termination of a Booking
3.	bookingId	Long	Stores the ID of the Booking that needs to be terminated prematurely
4.	terminationDate	LocalDateTime	Specifies when exactly does the Tenant wish to have his/her Booking terminated
5.	comment	String	A Tenant would probably like to explain the reason why he/she would like the Booking terminated prematurely. This

			field serves this purpose
6.	managersResponse	String	The Manager can either accept or deny the request; at any rate, the Manager’s response will be placed here
7.	status	ETRequestStatus	An Enum reflecting the processing stage of the request. Can be PENDING, APPROVED or DECLINED
8.	processedAt	LocalDate	Shows the date when the request has been processed

III. Finance-related entities:

1) Bill

It should be noted right away that this is NOT a bill issued to a Tenant in order for the latter to pay the rent. Instead, this entity is somewhat of a leftover from the times when the author thought it possible to develop a full-fledged system covering all aspects of property management. In the current setup, a Bill is merely any utility / mortgage / other bill that a Manager would have to pay in connection with his/her Property. Nevertheless, this does offer a degree of extra functionality, allowing Managers to add Bills that they need to pay in regard to a specific Property, and does assist in creating financial statements for Managers.

Fields of a Bill entity:

No.	**Field name**	**Field type**	**Comment**
1.	id	Long	ID of the entity
2.	amount	double	Pretty straightforward, this is the amount due and payable under this specific Bill
3.	currency	Currency	The currency in which the bill is
4.	property	Property	The Property that the Bill is appurtenant to.
5.	expenseCategory	String	Should allow managers to group their Bills better
6.	dueDate	Timestamp	The date by which the Bill should be paid. Yeah, I know it would have to be LocalDate, but that’s the old setup spooking the author
7.	recipient	String	The payee
8.	recipientIBAN	String	The payee’s IBAN
9.	isPaid	Boolean	Signifies whether the Bill has been paid already
10.	issuedAt	Timestamp	Date of issuance of the Bill
11.	addedAt	Timestamp	The time when the Bill has been added to the system

2) Currency

The author does admit that the Currency-related operations can still be developed to ensure a smoother process. Nevertheless, the existing functionality does consider the Currency that any monetary amount is expressed in, one being the base Currency of the system, and other facilitating viewing and paying operations for Tenants who might prefer their amounts to be specified in a Currency that is different from the base one. Has a number of redundant fields that the author has still decided to leave for possible future refactoring.

Fields of a Currency entity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity
2.	designation	String	REQUIRED. A three-letter designation of a currency, like EUR, USD, THB etc,
3.	isBaseCurrency	Boolean	Determines whether this Currency is the base Currency of the system
4.	rateToBase	Double	Currency exchange rate – 1.00 for the base Currency, respective rates for other Currencies. Can be managed by Admins
5.	bills	Set	Redundant – not used in the current setup, but retained for possible future use
6.	numericalConfigs	Set	Redundant – not used in the current setup, but retained for possible future use
7.	refunds	Set	Redundant – not used in the current setup, but retained for possible future use
8.	payouts	Set	Redundant – not used in the current setup, but retained for possible future use
9.	tenants	Set	Redundant – not used in the current setup, but retained for possible future use
10.	tenantPayments	Set	Redundant – not used in the current setup, but retained for possible future use

3) Payout

This entity is used to reflect a real-life payment that the platform would have to make to a Manager upon successful completion of a Booking. Generated automatically, these can also be added manually by Admins.

Fields of a Payout entity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity
2.	managerId	Long	REQUIRED. ID of the Manager that the Payout is due to
3.	bookingId	Long	REQUIRED. ID of the Booking that the Payout is made for
4.	amount	double	REQUIRED. Amount of the Payout
5.	createdAt	Timestamp	The time at which the Payout

			was created
6.	currency	Currency	Currency of the Payout (remains the base Currency of the system in the current setup, might be upgraded to support other currencies in the future)

4) PropertyDiscount
An entity employed by the system in order to allow Managers to set discounts (e.g. promotional or low-season) or surcharges (holiday or high-season) for the rental rice of their Properties on specific dates or during specific periods.

Fields of a PropertyDiscount entity:

No.	**Field name**	**Field type**	**Comment**
1.	id	Long	ID of the entity
2.	property	Property	The Property that the discount applies to
3.	percentage	Double	Negative for discounts, positive for surcharges. Can range from -100 (the Property is free for rent during this period) to infinity (rental fee can be raised as high as the Manager wants)
4.	periodStart	LocalDate	Beginning of the effective period of the discount
5.	periodEnd	LocalDate	End of the effective period of the discount
6.	createdAt	Timestamo	The time when the discount was added

5) Refund
In some cases, for instance, in case of cancellation in good time or a granted EarlyTerminationRequest, a Tenant may be entitled to a Refund, which is a portion of his/her original TenantPayment. This situation is facilitated through the Refund entity.

Fields of a Refund entity:

No.	**Field name**	**Field type**	**Comment**
1.	id	Long	ID of the entity
2.	tenantId	Long	REQUIRED. ID of the Tenant that the Refund is due to
3.	bookingId	Long	REQUIRED. ID of the Booking that the Refund is made for
4.	amount	double	REQUIRED. Amount of the Refund
5.	createdAt	Timestamp	The time at which the Refund was created
6.	currency	Currency	Currency of the Refund (remains the base Currency of the system in the current setup, might be upgraded to support other currencies in the future)

6) TenantPayment
A TenantPayment is produced along with a Booking to handle all aspects of a payment that a Tenant would have to make for booking this Property. The TenantPayment is received by the system from a Tenant, and is then transferred less the system commission fee to the respective Manager as a Payout.

Fields of a TenantPaymententity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity
2.	amount	Double	Amount of the payment
3.	currency	Currency	Currency of the payment, right now, it is the base Currency as in previous cases. Further upgrades required to handle payments in multiple Currencies
4.	tenant	Tenant	The Tenant liable to make this TenantPayment
5.	managerId	Long	ID of the Manager that the respective Payout will be due to
6.	associatedPropertyId	Long	ID of the Property that the payment is being received for
7.	associatedBookingId	Long	ID of the Booking that the payment is being received for
8.	receivedFromTenant	Boolean	A flag that determines whether this payment has already been received from the respective Tenant
9.	managerPayment	Double	The amount that will have to be made to the respective Manager as a Payout
10.	feePaidToManager	Boolean	A flag specifying whether the respective amount has already been paid to the respective Manager (a TenantPayment is deemed fully completed after both fags 8 and 10 are TRUE).
11.	receiptDue	Timestamp	The time by which the TenantPayment should be received from the Tenant

IV. Communication entities:
1) Claim
Whenever a Tenant is dissatisfied with the provided service for some reason, or a Manager is willing to complain about a Tenant’s actions, this Tenant or Manager may lodge a respective Claim against his/her counterpart in this specific transaction (Booking). Claims are processed by Admins; more functionality for Claims will be added in later versions

Fields of a Claim entity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity
2.	bookingId	Long	ID of the Booking in respect of which the Claim is made
3.	description	String	This is where the claimant is supposed to describe the essence of the Claim
4.	claimantType	ClaimantType	Enum – determines whether the claimant is a Tenant or a Manager
5.	claimStatus	ClaimStatus	Enum - a Claim can be OPEN (pending review), RESOLVED (resolved), CLOSED (closed for any reason without resolving) or WITHDRAWN (if the claimant withdraws it).
6.	admitted	Boolean	Set to TRUE if the alleged party in breach admits the claim without dispute
7.	createdAt	Timestamp	The time when the Claim was lodged
8.	resolvedAt	Timestamp	The time when the Claim was resolved
9.	resolution	String	Resolution of the Claim, sort of an explanation affixed by an Admin

2) Message
A Message is an entity facilitating communication between Tenants and Managers on site. Arranged messages form a conversation.

Fields of a Message entity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity
2.	content	String	The actual text of the Message
3.	senderId	Long	ID of the sender
4.	receiverId	Long	ID of the receiver
5.	receiverType	UserType	Defines if the recipient is a Tenant or a Manager; therefore, the sender is automatically treated as the “opposite” user type
6.	sentAt	LocalDateTime	Specifies the time when the Message has been sent
7.	isRead	Boolean	Can be used as a “flag” to show whether the Message has been read

3) Review
Optionally, a Tenant can leave a Review for a Property he/she stayed at, but only after the respective Booking has been completed.

Fields of a Review entity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity
2.	tenantId	Long	ID of the Tenant leaving the Review
3.	propertyId	Long	REQUIRED: ID of the Property being reviewed
4.	text	String	Text of the Review
5.	rating	int	The Tenant's rating of the Property, on a scale of 1 to 5

4) PropertyRating
The overall rating of a Property is calculated in reliance upon these entities.

Fields of a PropertyRating entity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity
2.	tenantId	Long	ID of the Tenant who rated the Property
3.	propertyId	Long	ID of the Property that has been rated
4.	bookingId	Long	ID of the Booking after which the Tenant rated the Property
5.	rating	int	The Tenant's rating of the Property, on a scale of 1 to 5

5) TenantRating
Somewhat similar to the previous one, but serving as the basis for Tenant ratings as evaluated by their respective Managers

Fields of a TenantRating entity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity
2.	tenantId	Long	ID of the Tenant being rated
3.	bookingId	Long	ID of the Booking after which the Manager rated the Tenant
5.	rating	Integer	The Manager's rating of the Tenant, on a scale of 1 to 5

V. Auxiliary entities:
1) NumericalConfig
Initially, this entity was supposed to handle discounts and other similar matters. It was not before the project expanded to the base level of what it is now that the author understood that discounts have to be handled in a different manner, considering the specific timeframes and other factors. Nevertheless, just about that time, a need arose for the storage of various numerical data, particularly for system settings, and this already existing entity proved rather comfortable for this purpose. Thus, it has survived the restructuring of the project, and is now used to store various numerical values for the system

Fields of a NumericalConfig entity:

No.	Field name	Field type	Comment

1.	id	Long	ID of the entity
2.	name	String	Name of the NumericalConfig; used to help the system retrieve the required config from the database.
3.	value	Double	The actual value stored within the entity
4.	type	NumConfigType	Enum. Can be DISCOUNT, SURCHARGE, SYSTEM_SETTING, SWITCH or COUNTER. Also used for retrieval purposes
5.	currency	Currency	An outdated field

2) PropertyAmenity
This entity serves as a link between a Property and a specific Amenity, allowing the assignment of Amenities to a Property and retrieval of these associations. The author admits that this approach is somewhat clumsy, but this was one of the first database operation techniques ever shown to us during classes, and this entity was one of the first to be created within the system, so in order to avoid any major changes, it has been retained.

Fields of a PropertyAmenity entity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity
2.	property_id	Long	ID of the Property to which Amenities are assigned
3.	amenity_id	Long	ID of the assigned Amenity

3) TokenResetter
Another auxiliary entity. The Timestamp herein is used to determine the time during which the token exists, allowing a special scheduled task to clean the outdated ones, preventing access therewith and keeping the database table from expanding.

Fields of a TokenResetter entity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity
2.	userId	Long	ID of the user (Tenant or Manager) whose token is to be reset
3.	userType	UserType	An ENUM – this can be TENANT or MANAGER here
4.	createdAt	Timestamp	The time when the token and the respective TokenResetter entity have been created

4) NumericDataMapping
This entity contains a three-level map, which would store the SecretKeys for AES-encrypted users' payment card numbers and CVV codes at its bottom level. The mid-level would store the user type, and the upper level would contain the user's ID. Having consulted an expert from an existing developer

company, however, the author was discouraged from storing these Secret Keys in the same database with the actual encrypted values. The respective mechanisms were redeveloped as per the advice, and as a result, entities of this type are still being created – however, these are not persisted to the database. Instead, these are transformed into text lines, encrypted using the author's developed encryption mechanism (albeit rather simple, based on random shifts of ASCII values) and written into a separate file, wherefrom these are retrieved in case of necessity to use a user's payment details. This process also requires the mediation of a KeyLink entity, described further. The

Fields of a NumericDataMapping entity:

No.	**Field name**	**Field type**	**Comment**
1.	id	Long	ID of the entity
2.	map	Map<Long, Map<UserType, Map<Boolean, SecretKey>>>	The three-level storage structure described above. Level 1 Long key: user entity ID Level 2 UserType key: type of the user Level 3 Boolean key: true for a Card Number, false for a CVV code SecretKey value: secret key value

5) KeyLink
Another auxiliary entity, used to facilitate access to externally stored encrypted SecretKey values. In order to avoid overloading the storage file and mitigate the risk in case of data loss or theft, encrypted SecretKey records are limited to 100 per file. The KeyLink entity allows the program to identify the file where a particular user's SecretKeys are being stored.

Fields of a KeyLink entity:

No.	**Field name**	**Field type**	**Comment**
1.	id	Long	ID of the entity
2.	userId	Long	ID of the user (Tenant or Manager) whose keysare stored in a file
3.	userType	UserType	An ENUM – this can be TENANT or MANAGER here
4.	fileNumber	Integer	The number of the file where the record is being stored

6) PropertyLock
Whenever a Manager wishes to lock his/her Property for a specific period of time, a stub "Booking" is created to block the respective duration, effectively making the Property unavailable for rent during that period. However, inasmuch as Managers do not have the functionality to cancel Bookings, a special PropertyLock entity is required in order to allow them to lift their own imposed blocks.

Fields of a PropertyLock entity:

No.	Field name	Field type	Comment
1.	id	Long	ID of the entity
2.	bookingStubId	Long	ID of the stub Booking locking the respective Property
3.	propertyId	Long	ID of the Property in question

Apart from that, there are three tables associating authorities with users; nevertheless, inasmuch as they only serve access validation purposes, these do not participate in the business logic.

The overall diagram of the resulting database is as follows:
https://dbdiagram.io/d/PROPMAN_DATABASE-66e93f9ba0828f8aa61a0251

4. User types and system operation

This section describes the types and functions of users within the system, forming a general idea of the system's normal course of operation. These are overall descriptions; please feel free to contact the author for detailed analysis of all functions.

Users of the platform are categorized into three types: Tenants, Managers and Admins. These types are not hierarchic; each of these user types implies a certain set of available functions. Hence, the functionality is grouped into six categories:

1) General functions (mostly available to everyone, including unauthorized users)
2) Authorization functions
3) Tenant functions
4) Manager functions
5) Admin functions
6) Auxiliary functions (scheduled or startup tasks triggered without user intervention)

General functions:
Aside from authorization activities, unauthorized users have access to the following functions:

- Searching for Properties, specifying a number of criteria
- Viewing the details of a Property
- Reading Reviews for a Property
- Getting publicly available information on Property Managers
- Getting Property portfolios of a specific Manager

Authorization functions
These allow users to sign up, log in and manage their access credentials to an extent. The following access-related functionality has been implemented:

- Registration as a Tenant
- Registration as a Manager

- Logging in (a common endpoint for Tenants and Managers)
- A separate Admin login endpoint
- OTP verification procedures for Tenants, Managers and Admins
- Registration confirmation by email (for Tenants and Managers)
- Email address change confirmation
- Changing one’s password (requires the user to be logged in at the time)
- Resetting one’s password (two endpoints, one for requesting, the other one for completion)

Tenant functions
Tenant functionality is built around the process of renting the available Properties. Apart from being able to search for and view Properties and Managers (general functions), Tenants are granted the following scope of functions:

- View and edit their own profiles
- Make and cancel Bookings for Properties
- Save, retrieve and remove their favorite Properties from the favorites list
- Make Payments, view their completed and outstanding Payments
- View their Bookings
- Send / receive Messages to / from Managers
- Make Early Termination Requests
- Rate Properties
- Leave Property reviews
- Submit Claims

Manager functions
Managers are provided with an array of functions facilitating the management of their rental Properties, managing appropriate Bookings, communicating with Tenants and receiving Payouts. The exact Manager function set is as follows:

- View and edit their own profiles
- Retrieve their Bookings – all of these, just the ones for a specific Property, or any Bookings pending approval on their part
- View profiles of the Tenants who book their Properties
- Add Properties to the system
- Update their Property details
- Make their Properties unavailable for booking for a certain period, and unlock any of their previously locked Properties
- Manage Amenities appurtenant to their Properties
- Upload or remove photos of their Properties
- Remove their Properties from the system
- Set and reset discounts or surcharges for their Properties for specific periods
- Add and delete Bills incidental to their Properties, view all unpaid Bills or just the ones for a specific Property
- Generate financial statements for themselves

- Approve or decline Bookings of their Properties made by Tenants
- Accept or decline Early Termination Requests from Tenants
- Send / receive Messages to / from Tenants
- Rate Tenants staying at their Properties
- Submit Claims

Admin functions
Normally, Admins do not participate in the core business process; instead, they are provided with a range of functions required to oversee the operation of the system and perform manual intervention if necessary. All Admins can:

- Toggle the active status of any Tenant or Manager, effectively suspending or restoring the respective users
- Register new Tenants and Managers, update their information if requested to do so by the respective users or for other valid reasons, and remove these user entities if the situation so requires
- Add and remove Properties
- Add new Currencies, alter Currency exchange rates or set a new base Currency
- Add new types of Amenities to the system or remove these if required
- Edit system settings (Numerical Configs)
- View and resolve Claims
- Manually create, view and settle Payouts and Refunds
- Delete Reviews if necessary

Apart from that, there is an “uber” DefaultAdmin user entity, which cannot be deleted by normal means and, in addition to the functions described above, is also able to:

- Create and delete accounts for other Admins

Auxiliary functions
These functions mostly require no user interaction and are meant to facilitate the operation of the platform, being triggered on startup or as scheduled tasks:

- Creation of a new DefaultAdmin on first startup if none is present, and request for new DefaultAdmin details through the console
- Generation of a sample database – if instructed to do so through console input, the program will overwrite any existing database that may be there at the time, and create a new one, populated with a number of entities. This is very useful for testing. Sample databases are also easy to regenerate if need be. Inasmuch as this action removes any old database records, this action requires the user to enter the DefaultAdmin password
- Changing the status of Bookings as appropriate as of the current date
- Removal of already processed or expired Early Termination Requests
- Cancelling Bookings that have not been paid on time
- Clearing expired Tokens
- Closing Bookings that are over if no Claims are present, and converting these to Leasing Histories
- Deleting outdated Property Discounts
- Reminding Tenants of the approach of Payment due dates by email

- Reminding users of their payment card expiration dates
- Reminding Managers of guest arrivals
- Deleting outdated Property locking entities
- Removing old resolved Claims

5. Applied technologies

This section is essentially a short list of technologies used in the course of development.

Spring Boot (Web, Security, Data JPA, Thymeleaf)
Hibernate (for JPA implementation)
Jakarta (persistence, validation)
MySQL (main database, with H2 for testing)
Swagger (API documentation)
Lombok (code generation)
MapStruct (object mapping)
JWT (authentication)
Apache POI (handling Excel files)
JUnit, Hamcrest (testing framework)
Jackson: Explicit handling of JSON (with time and custom serializers).
SLF4J and Log4j: For logging
Java Cryptography Extension (JCE): For handling encryption keys
JavaMail - for mailing purposes
BCrypt for password hashing
AES for encrypting sensitive data
Docker (containerization)

6. Further improvement opportunities

The section contains suggestions regarding potential further development of this graduation project.

The author intends to develop this project for his portfolio post-defense as well, so the first focus area of any further improvement strategy would involve the addition of any components lacking at this time. Added elements would involve unit tests for all methods that implement the immediate business logic (as stated before, the previously written ones had to be removed, having become outdated as the project had been evolving).Subsequent efforts would be focused at code refactoring, in order to facilitate interaction with the database.

Should the optimized backend component be deemed fit for real-life deployment, a small team would have to be assembled in order to develop the frontend component for desktop and mobile devices, as well as to perform a full-fledged testing on every aspect of the resulting product, focused, inter alia, on the matters of efficiency and security. In case of success, the product could be proposed as a premade solution for individuals and businesses of the property rental industry, or undergo further evolution to encompass a broader range of real estate management issues.

7. Development experience

Personal impressions that the author would like to share regarding the development process.

Inasmuch as the project is intended mostly for educational purposes, natural first thing to mention would be its value for the author's own learning progress, both as an opportunity to practice any skills acquired during the previous stage of the author's studies, and as a testing ground for several technologies that the author had to learn specifically for attaining the objectives of the graduation project. In both cases, the author struggled to overcome a multitude of obstacles, which has definitely improved the author's problem-solving capabilities that would turn valuable if any similar or relatively similar problems are ever encountered in real-life coding scenarios.

The overall scope of experience thus acquired is in fact well beyond just that; aside from immediate coding proficiency, the author had a hands-on opportunity to learn the true value of exact preliminary planning and scheduling. The absence thereof at the initial stage of this specific project has resulted in a considerable loss of time and effort for the author; as a result, the development process took a much longer time than initially expected, and the project itself still had to be restructured a number of times - some less than perfect implementations therein are the legacy of these older designs, probably decreasing operation efficiency. Yet again, it should be noted here that the author would hardly be able to produce a clear and feasible development plan without the experience obtained in the course of dealing with the aforementioned issues, so this outcome, though far from perfect, will definitely have a positive impact on any future projects that the author might participate in.

Another important aspect to mention is the opportunity to practice the observation of requirements approximating those of real-life scenarios. A good instance would be the evolution of mechanisms for storing payment card data: at first, these used to be mere fields of the respective entities. Having been advised that these are sensitive data inadequate for non-encoded storage, the author implemented mechanisms for storing these data using the AES algorithm. Finally, after a comment from a formidable industry specialist blaming such approach for the insecurity of encoded values and secret keys stored in the same database, the author altered the key storage facilities to be further encrypted via a custom mechanism and stored in different files; while real-life security measures in this regard would probably require an even more sophisticated approach, this advance has a high value for the author per se, as a part of the author's education process.

Finally, of course, concluding this manual seems impossible without mentioning the experience obtained by the author as regards the issues of learning and problem solving. By the end of the development process, the author had a chance to greatly improve his solution-seeking skills, making combined use of resources like AI capabilities, available text and media materials on specific matters, and the invaluable help of professionals who helped the author in finding specific solutions or facilitated the bulk of the educational process the author had to undergo prior to the commencement of this project. Therefore, the author sees fit to end this manual with a word of

gratitude and appreciation to all Tel-Ran tutors and external counsellors who provided their assistance as the project progressed to completion.