Abilio Oliveira & James Dryden
999550263 993083613

Homestay App Specification

Types

Base classes from requirement's class diagram:

$$[Admin, Profile, File, DateRange, Allergie]$$
$$Response ::= success \mid error \mid reportAccept \mid time \mid other$$
$$ProfileType ::= student \mid host$$
$$FamilyStructure ::= couple \mid couple\ w/\ kids \mid single\ parent \mid gay\ couple \mid gay\ couple\ w/\ kids \mid other$$
$$Pet ::= cat \mid dog \mid chicken \mid birds \mid snakes \mid fish \mid other$$

Obs: Response was added to fulfill zed description requirements.

The System

HOMESTAYAPP

$$\Xi STUDENTS$$
$$\Xi HOSTS$$
$$managers : \mathbb{P}\, Admin$$
$$requests : Profile \to Profile$$
$$authorized\_requests : Profile \to Profile$$
$$accepted\_requests : Profile \to Profile$$

$$students \subseteq Profile$$
$$hosts \subseteq Profile$$
$$students \cap Hosts = \varnothing$$
$$\mathrm{dom}\, requests \subset students$$
$$\mathrm{ran}\, requests \subset hosts$$
$$requests \cap authorized\_requests = \varnothing$$
$$requests \cap accepted\_requests = \varnothing$$
$$authorized\_requests \cap accepted\_requests = \varnothing$$
$$\forall\, s : students \mid s \in \mathrm{dom}\, Requests \bullet \#(s \lhd Requests) \leq 3$$

The system must hold a schema by itself with the purpose of declaring its elements as well as its invariants.

Actors

For the purposes of this specification, we are only working with three main properties for both hosts and students, that is: Family Structure, Allergies and Pets. Those are the main points to be covered by this specification. Other properties, such as name, phone_number, etc are not as important as the ones here approached.

STUDENTS

$$students : \mathbb{P}(Profile \to (\mathbb{P}(\mathbb{N} \to FamilyStructure) \times \mathbb{P}(\mathbb{N} \to Pet) \times \mathbb{P}\, Allergie))$$

HOSTS

$hosts : \mathbb{P}(Profile \rightarrow (\mathbb{P}\, Pet \times FamilyStructure \times \mathbb{P}\, Allergie))$

Manual Selection

MANUAL

$\Xi HOMESTAYAPP$
$manual\_selection : (\mathbb{P}\, Pet \times \mathbb{P}\, FamilyStructure \times \mathbb{P}\, Allergie) \rightarrow \mathbb{P}\, Profile$

$\forall f\_pet : (\mathbb{P}\, Pet, f\_family : \mathbb{P}\, FamilyStructure, f\_allergie : \mathbb{P}\, Allergie;\ s\_hosts : \mathbb{P}\, Profile$
$\bullet manual\_selection\ f\_pet\ f\_family\ f\_allergie \ \Leftrightarrow$
$\forall h : hosts \mid$
$\quad ((\forall f\_p : f\_pet \bullet f\_p \in first(\mathrm{ran}\, h)) \vee$
$\quad (\forall f\_f : f\_family \bullet f\_f \in second(\mathrm{ran}\, h))) \wedge$
$\quad \neg(\forall f\_a : f\_allergie \bullet f\_a \in thrid(\mathrm{ran}\, h)) \bullet s\_hosts \cup h\}$

This operation goes through every host and return those hosts that 'obey' the filters.

Selection Wizard

WIZARD

$\Xi HOMESTAYAPP$
$selection\_wizard : (\mathbb{P}(\mathbb{N} \rightarrow Pet) \times \mathbb{P}(\mathbb{N} \rightarrow FamilyStructure) \times \mathbb{P}\, Allergie) \rightarrow \mathbf{seq} Profile$

$\forall f\_pet : (\mathbb{P}(\mathbb{N} \rightarrow Pet), f\_family : \mathbb{P}(\mathbb{N} \rightarrow FamilyStructure), f\_allergie : \mathbb{P}\, Allergie;\ s\_hosts : \mathbf{seq} Profile$
$\bullet\ selection\_wizard\ f\_pet\ f\_family\ f\_allergie \ \Leftrightarrow$
$\forall h : hosts \bullet order\_and\_normalize(s\_hosts \cup \{$
$\quad (\forall f\_p : f\_pets \bullet (\mathrm{ran}\, f\_p \in first(\mathrm{ran}\, h) \wedge \mathrm{dom}\, f\_p)) *$
$\quad (\forall f\_f : f\_family \bullet (\mathrm{ran}\, f\_f \in second(\mathrm{ran}\, h) \wedge \mathrm{dom}\, f\_f)) *$
$\quad (\forall f\_a : f\_allergies \bullet (\mathrm{ran}\, f\_a \in third(\mathrm{ran}\, h) \wedge 2)) \mapsto h\})$

This operation takes as input a 'family preference', a 'pet preference' and the allergies that a student is subject to. This operation outputs a **sequece** of hosts, ordered by the 'order_and_normalize function' (that we do not define here). The computation works by multiplying the preference values of the elements encountered inside each hosts 'family structure' and 'pets'. The allergiers encountered have a weight of 2.

General Operations

Operations for both Student and Host

$$\begin{array}{l}
\textit{CreateNewProfile} \\
\hline
\Delta STUDENTS \\
\Delta HOSTS \\
type? : ProfileType \\
p? : Profile \\
family\_str\_if\_host? : FamilyStructure \\
pets\_if\_host? : \mathbb{P}\, Pet \\
family\_str\_pref\_if\_std? : \mathbb{P}(\mathbb{N} \rightarrow FamilyStructure) \\
pet\_pref\_if\_std? : \mathbb{P}\, Pet \\
allr? : \mathbb{P}\, Allergie \\
r! : Response \\
\hline
(type? = student \wedge \\
\quad students' = students \cup \{p? \mapsto (family\_str\_pref\_if\_std?,\ pet\_pref\_if\_std?,\ allr?)\} \wedge \\
\quad hosts' = hosts \wedge r! = success) \vee \\
(type? = host \wedge \\
\quad hosts' = hosts \cup \{p? \mapsto (pets\_if\_host?,\ family\_str\_if\_host?,\ allr?)\} \wedge \\
\quad students' = students \wedge r! = success)
\end{array}$$

When creating a profile the system doesn't yet know if the request is for a student profile or a host profile, that's why we need *type?*.

$$\begin{array}{l}
\textit{UpdateProfile} \\
\hline
\Delta STUDENTS \\
\Delta HOSTS \\
p? : Profile \\
family\_str\_if\_host? : FamilyStructure \\
pets\_if\_host? : \mathbb{P}\, Pet \\
family\_str\_pref\_if\_std? : \mathbb{P}(\mathbb{N} \rightarrow FamilyStructure) \\
pet\_pref\_if\_std? : \mathbb{P}\, Pet \\
allr? : \mathbb{P}\, Allergie \\
r! : Response \\
\hline
(p? \in \operatorname{dom} students \wedge \\
\quad students' = students \cup \{p? \mapsto (family\_str\_pref\_if\_std?,\ pet\_pref\_if\_std?,\ allr?)\} \wedge \\
\quad hosts' = hosts \wedge r! = success) \vee \\
(p? \in \operatorname{dom} students \wedge \\
\quad hosts' = hosts \cup \{p? \mapsto (pets\_if\_host?,\ family\_str\_if\_host?,\ allr?)\} \wedge \\
\quad students' = students \wedge r! = success) \vee \\
(p? \notin \operatorname{dom} students \wedge p? \notin \operatorname{dom} hosts\ \wedge \\
\quad students' = students \\
\quad hosts' = hosts \\
\quad r! = error)
\end{array}$$

When updating, we can tell from *p?* if its about students or hosts. Either way, this operation overwrites the data previously inserted for any information about students and/or hosts.

DeleteProfile

$$\begin{array}{l}
\Delta STUDENTS \\
\Delta HOSTS \\
\Delta HOMESTAYAPP \\
p? : Profile \\
r! : Response \\
\hline
(p? \in \operatorname{dom} students \wedge \\
\quad students' = students \setminus \{p?\} \lhd students \wedge \\
\quad requests' = requests \setminus \{p?\} \lhd requests \wedge \\
\quad authorized\_requests' = authorized\_requests \setminus \{p?\} \lhd authorized\_requests \wedge \\
\quad accepted\_requests' = accepted\_requests \setminus \{p?\} \lhd accepted\_requests \wedge \\
\quad managers' = managers \\
\quad hosts' = hosts \wedge r! = success) \vee \\
(p? \in \operatorname{dom} hosts \wedge \\
\quad hosts' = hosts \setminus \{p?\} \lhd hosts \wedge \\
\quad requests' = requests \setminus requests \rhd \{p?\} \wedge \\
\quad authorized\_requests' = authorized\_requests \setminus authorized\_requests \rhd \{p?\} \wedge \\
\quad accepted\_requests' = accepted\_requests \setminus accepted\_requests \rhd \{p?\} \wedge \\
\quad managers' = managers \\
\quad students' = students \wedge r! = success) \vee \\
(p? \notin \operatorname{dom} students \wedge p? \notin \operatorname{dom} hosts \wedge r! = error)
\end{array}$$

When deleting a profile, we need to make sure that no garbage remains on the system; thats why we make sure to remove those requests that contain such profile.

ViewRequests

$$\begin{array}{l}
\Xi HOMESTAYAPP \\
p? : Profile \\
r! : Response \\
\hline
(p? \in \operatorname{dom} students \wedge \\
\quad display\_request(p? \lhd requests) \wedge \\
\quad display\_request(p? \lhd authorized\_requests) \\
\quad display\_request(p? \lhd accepted\_requests) \wedge r! = success) \vee \\
(p? \in \operatorname{dom} hosts \wedge \\
\quad display\_request(requests \rhd p?) \wedge \\
\quad display\_request(authorized\_requests \rhd p?) \wedge \\
\quad display\_request(accepted\_requests \rhd p?) \wedge r! = success) \vee \\
(p? \notin \operatorname{dom} students \wedge p? \notin \operatorname{dom} hosts \wedge r! = error)
\end{array}$$

Operations related to Student

ManualRequest

$\Delta HOMESTAYAPP$
$\Xi MANUAL$
$s? : Profile$
$f\_family : \mathbb{P}\, Family$
$f\_pets : \mathbb{P}\, Pet$
$f\_allergies : \mathbb{P}\, Allegie$
$select\_hosts? : \mathbb{P}\, Profile$
$r! : Response$

$temp = manual\_selection f\_pets f\_family f\_allergies$
$requests' = requests \cup (\forall\, c : select\_hosts(temp) \bullet \{s? \mapsto c\})$
$managers' = managers$
$authorized_r equests' = authorized_r equests$

This schema allows the student to make a request based on a selection of hosts over what 'manual_selection' returns. We here introduce the function 'select_hosts' but we don't define it, since it is a interactive behavior and subject to time constraints (thus out of zed domain).

WizardRequest

$\Delta HOMESTAYAPP$
$\Xi WIZARD$
$s? : Profile$
$r! : Response$

$temp = selection\_wizard\ first(\mathrm{ran}\, s?)\ second(\mathrm{ran}\, s?)\ third(\mathrm{ran}\, s?)$
$requests' = requests \cup (\forall\, c : \mathrm{ran}(first\_three(temp)) \bullet \{s? \mapsto c\})$
$managers' = managers$
$authorized_r equests' = authorized_r equests$

This schema allows the students to make a request based on its own preferences. The function 'first_three' returns the first three elements from a sequence (not defined in this specification).

Operations related to Host

AcceptRequest

$\Delta HOMESTAYAPP$
$s? : Profile \rightarrow Profile$
$r! : Response$

$s? \in authorized\_requests$
$accepted\_requests' = accepted\_requests \cup \{s?\}$
$authorized\_requests' = authorized\_requests \setminus \{s?\}$
$requests' = requests$
$managers' = managers$
$r! = reportAccept$

Operations related to Admin

ViewRequests

$\Xi HOMESTAYAPP$
$s? : \mathbb{N}$

$$\begin{aligned}
&(s = 0 \land \\
&\quad \{\forall x : Profile \rightarrow Profile \mid x \in requests \bullet display\_requests(x)\}) \lor \\
&(s = 1 \land \\
&\quad \{\forall x : Profile \rightarrow Profile \mid x \in authorized\_requests \bullet display\_requests(x)\}) \lor \\
&(s = 2 \land \\
&\quad \{\forall x : Profile \rightarrow Profile \mid x \in accepted\_requests \bullet display\_requests(x)\})
\end{aligned}$$

The view requests schema allows the host or student to view all requests that have been authorized by the admin, accepted by the host, or that are still awaiting authorization.

AuthorizeRequest

$\Delta HOMESTAYAPP$
$s? : Profile \rightarrow Profile$
$r! : Response$

$s? \in requests$
$authorized\_requests' = authorized\_requests \cup \{s?\}$
$requests' = requests \setminus \{s?\}$
$accepted\_requests' = accepted_r equests$
$managers' = managers$
$r! = reportAccept$

This schema refers to the second stage of a request, where it gets authorized by the Admin; we need then to take a request from the request set and put it into the authorized request set, worrying to not change any other set of the main system.

ArrangeMeeting

$\Delta HOMESTAYAPP$
$s? : Profile \rightarrow Profile$
$r! : Response$

$s \in accepted\_requests$
$r! = time$

The arrange meeting schema provides the ability to take as input a request, which is a relation between a student and host, and notify both the student and host of a meeting time.