Implementation - Documentation of the adapted Helios structural model and databases

This document describes the adaptations to the Helios structural models, hence the implemented classes and the corresponding class diagram, as well as the adaptations to the Helios databases.

Structural model

Helios's class diagram is composed of many classes. However, figure 1 shows the most significant ones, illustrating the main components and relations in the system, including the modifications made as part of the implementation of our enhancements. Most classes, *Voter, Trustee, Tally*, etc., are subclasses of the *HeliosObject* superclass, which represents the main Helios system. In addition, the *Voter* and *Tally* classes are parts of the *Election* class, as neither voters nor trustees can exist without belonging to an election, hence the use of composition in this case. Moreover, the *Trustee* class is considered part of the *Tally* class, as trustees take part in calculating a final election tally. However, given that trustees can exist without having a (partial) tally to compute, they can exist without belonging to one. Therefore, the relationship here is aggregation. Further, the *QRCode* class is part of our implementation of the QR code verification mechanism, from which objects are created to represent the data each displayed QR code will hold.

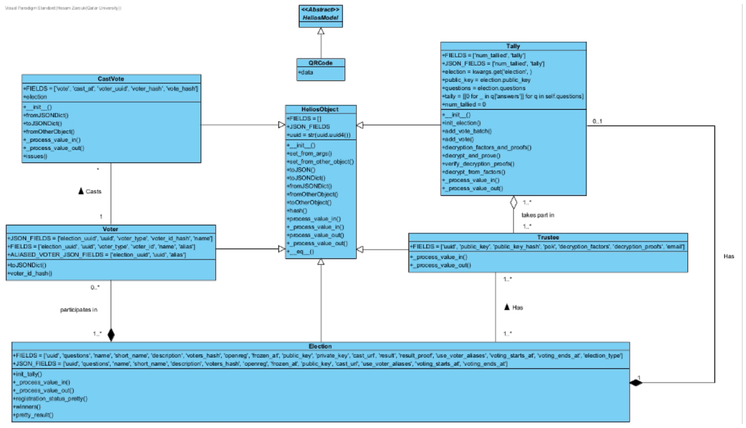

Visual Paradigm Standard(Hosam Zarouk(Qatar University))
<<Abstract>>
HeliosModel
QRCode
+data
CastVote
+FIELDS = ['vote', 'cast_at', 'voter_uuid', 'voter_hash', 'vote_hash']
+election
+__init__()
+fromJSONDict()
+toJSONDict()
+fromOtherObject()
+_process_value_in()
+_process_value_out()
+issues()
HeliosObject
+FIELDS = []
+JSON_FIELDS
+uuid = str(uuid.uuid4())
+__init__()
+set_from_args()
+set_from_other_object()
+toJSON()
+toJSONDict()
+fromJSONDict()
+fromOtherObject()
+toOtherObject()
+hash()
+process_value_in()
+_process_value_in()
+process_value_out()
+_process_value_out()
+__eq__()
Tally
+FIELDS = ['num_tallied', 'tally']
+JSON_FIELDS = ['num_tallied', 'tally']
+election = kwargs.get('election',)
+public_key = election.public_key
+questions = election.questions
+tally = [[0 for _ in q['answers']] for q in self.questions]
+num_tallied = 0
+__init__()
+init_election()
+add_vote_batch()
+add_vote()
+decryption_factors_and_proofs()
+decrypt_and_prove()
+verify_decryption_proofs()
+decrypt_from_factors()
+_process_value_in()
+_process_value_out()
0..1
*
Casts
1
Voter
+JSON_FIELDS = ['election_uuid', 'uuid', 'voter_type', 'voter_id_hash', 'name']
+FIELDS = ['election_uuid', 'uuid', 'voter_type', 'voter_id', 'name', 'alias']
+ALIASED_VOTER_JSON_FIELDS = ['election_uuid', 'uuid', 'alias']
+toJSONDict()
+voter_id_hash()
1..*
takes part in
1..*
Trustee
+FIELDS = ['uuid', 'public_key', 'public_key_hash', 'pok', 'decryption_factors', 'decryption_proofs', 'email']
+_process_value_in()
+_process_value_out()
Has
0..*
participates in
1..*
1..*
Has
1..*
Election
+FIELDS = ['uuid', 'questions', 'name', 'short_name', 'description', 'voters_hash', 'openreg', 'frozen_at', 'public_key', 'private_key', 'cast_url', 'result', 'result_proof', 'use_voter_aliases', 'voting_starts_at', 'voting_ends_at', 'election_type']
+JSON_FIELDS = ['uuid', 'questions', 'name', 'short_name', 'description', 'voters_hash', 'openreg', 'frozen_at', 'public_key', 'cast_url', 'use_voter_aliases', 'voting_starts_at', 'voting_ends_at']
+init_tally()
+_process_value_in()
+_process_value_out()
+registration_status_pretty()
+winners()
+pretty_result()
1

Fig. 1: Enhanced Helios's Class Diagram

Figure 2 illustrates the class diagram of HelioScan, our implemented QR code verification mobile application. To illustrate, the red square symbols represent private attributes and methods, and the green circles represent public attributes and methods. Since it is a relatively simple application, HelioScan contains only two main classes: *ElectionRecord* and *BallotRecord*. The *BallotRecord* object stores a ballot tracker which belongs to exactly one election, i.e., *ElectionRecord* object. In addition, each *ElectionRecord* object contains exactly one ballot tracker, i.e., *BallotRecord* object, as the application stores only the latest tracker the user has scanned for a particular election. Moreover, the *ElectionRecord* class has an attribute called *userID*, which represents the device's ID with which the user scanned the QR code. This enables the application to distinguish and display the list of elections in which the user has participated.

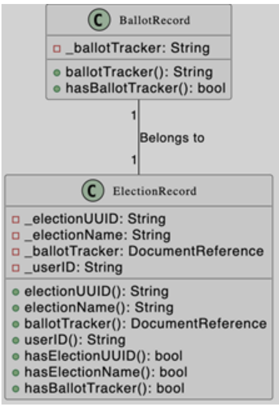


Fig. 2: HelioScan's Class Diagram

Databases

The Helios original databases, to specify the corresponding Entity-Relationship Diagram (ERD) model shown in figure 3, has not been modified by the implemented enhancements.

The HelioScan database, depicted in figure 4, consists of two entities named *ElectionRecord* and *BallotRecord*. The relationship between these entities is one-to-one, as the application is designed to store only the latest ballot tracker for each election the user participated in, and each ballot tracker belongs to exactly one election. The *ElectionRecord* comprises of the *electionUUID* (election's unique identifier) as a primary key, as well as other attributes including the *electionName*, which is used to display the names of all participated elections for a particular user. Moreover, the *userID* attribute uniquely represents each user based on their device, in order to differentiate which elections belong to which users. Additionally, the *createdAt* attribute holds a timestamp of the election record's creation, and the *ballotTracker* attribute is a reference to the particular *BallotRecord* entity that belongs to this *ElectionRecord* entity. Put simply, the *ballotTracker* attribute acts as a foreign key to the *ElectionRecord*.

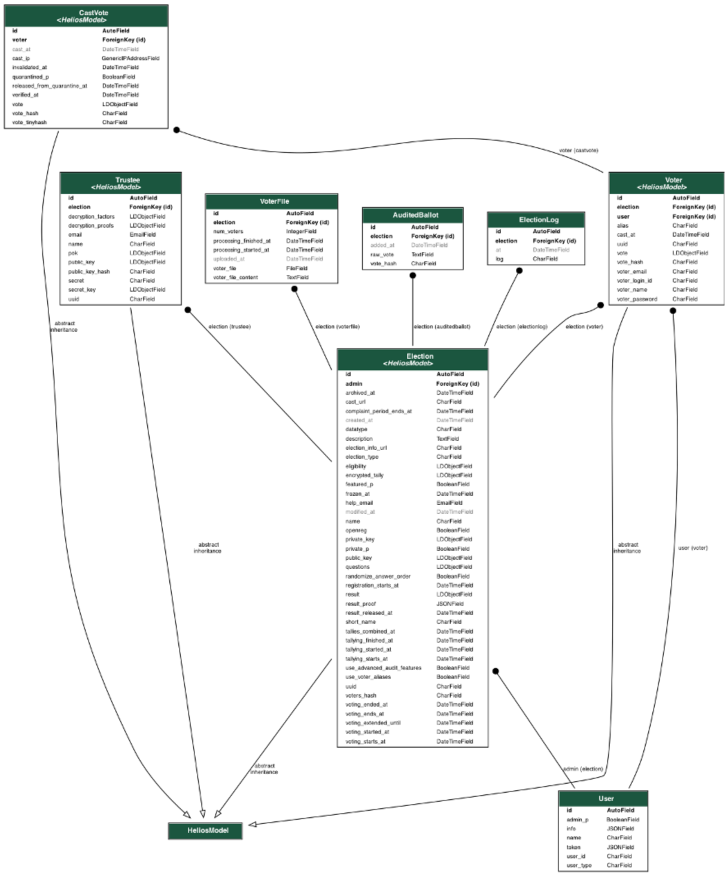


Fig. 3: An Entity-Relationship Diagram (ERD) diagram for the Helios system, derived from the original source code.

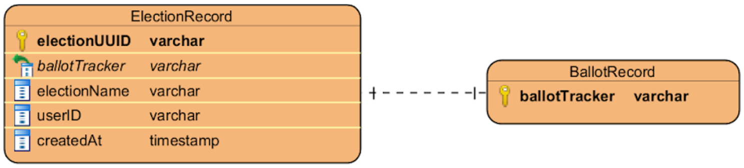


Fig. 4: An Entity-Relationship Diagram (ERD) diagram for HelioScan.

Figure 5 outlines the hierarchical representation of the third-party *Firebase* database, i.e., the verification server. At the very top of this hierarchy lies the *URL* to the database itself. Underneath this *URL*, there exists a branch called *ballot_ trackers* which holds a list of *election UUID branches*, each of which holds yet another list of *BallotTracker* branches representing all stored ballot trackers for each election. Finally, under each *BallotTracker* branch, there exists a boolean key called *verificationResult*, which dictates the correctness of the construction of this ballot tracker, if the user decides to verify their ballot.

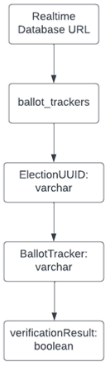
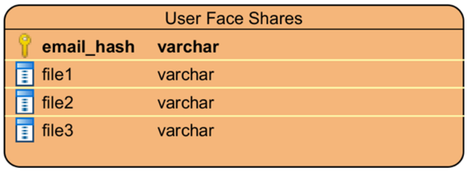


Fig. 5: A hierarchical relationship diagram for the third-party verification server.

Further, figure 6 shows the third-party authentication server's database. This database consists of only one entity, as it is used to store only part of each user's biometric data, i.e. one of the two sets of biometric shares.



User Face Shares

email_hash	**varchar**
file1	varchar
file2	varchar
file3	varchar

Fig. 6: An Entity-Relationship Diagram (ERD) diagram for the third-party authentication server.