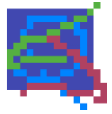


# Ιστοσελίδα Ζωολογικού Κήπου

ΟΜΑΔΙΚΗ ΕΡΓΑΣΙΑ ΓΙΑ ΤΟ ΜΑΘΗΜΑ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΔΙΑΔΙΚΤΥΟΥ (ECE\_ΓΚ802)

ΟΜΑΔΑ 30



*Το θέμα της εργασίας είναι μια ενημερωτική ιστοσελίδα για ζωολογικό κήπο, στην οποία ένας επισκέπτης μπορεί να βρει στοιχεία επικοινωνίας, τα ζώα που φιλοξενούνται και να πραγματοποιήσει αγορά εισιτηρίων. Η εφαρμογή χρησιμοποιεί εξυπηρετητή σε NodeJS, με χρήση της βιβλιοθήκης Express.*

**ΠΑΤΡΑ 2024**

ΕΥΘΥΜΙΟΣ, ΠΑΝΙΔΗΣ

[themispan2002@gmail.com](mailto:themispan2002@gmail.com), [up1084012@ac.upatras.gr](mailto:up1084012@ac.upatras.gr)

ΑΛΕΞΑΝΔΡΟΣ ΜΑΡΙΟΣ, ΤΣΑΚΙΡΙΔΗΣ

[alexandros.tsakiridis2@gmail.com](mailto:alexandros.tsakiridis2@gmail.com), [up1083879@ac.upatras.gr](mailto:up1083879@ac.upatras.gr)

ΛΕΞΕΙΣ - ΚΛΕΙΔΙΑ: web, node.js, express, html, handlebars, sqlite

## ΠΕΡΙΕΧΟΜΕΝΑ

1	ΠΕΡΙΛΗΨΗ .....	3
2	ΜΕΘΟΔΟΛΟΓΙΑ .....	3
2.1	Στόχος.....	3
2.2	Χαρακτηριστικά .....	3
2.3	Τεχνολογίες .....	4
3	ΟΡΓΑΝΩΣΗ .....	4
3.1	Επισκόπηση .....	4
3.2	Βάση Δεδομένων .....	5
3.2.1	Πίνακες.....	6
3.2.2	Δομή του ticket.....	6
3.2.3	Διαχειρίστρια κλάση .....	7
4	ΔΙΑΔΡΟΜΕΣ BROWSER .....	7
5	ΣΕΝΑΡΙΟ ΧΡΗΣΗΣ .....	8
5.1	Διεπαφή .....	8
5.2	Λειτουργία σύνδεσης.....	8
5.3	Loops στη σελίδα animals .....	9
5.4	Hashing.....	10
5.5	Η κλάση Accountant.....	10
5.6	Αυτόματη σύνδεση .....	11
6	ΧΡΟΝΟΔΙΑΓΡΑΜΜΑ.....	11
7	ΑΞΙΟΛΟΓΗΣΗ.....	12
7.1	Επέκταση login.....	12
7.2	Debug decorator .....	13
7.3	Πρόσθετα χαρακτηριστικά .....	13
7.4	Στιγμιότυπα .....	15
8	ΕΓΚΑΤΑΣΤΑΣΗ.....	20
8.1	Αποθετήριο .....	20
9	ΒΙΒΛΙΟΓΡΑΦΙΑ .....	20

## 1 ΠΕΡΙΛΗΨΗ

Η παρούσα εργασία αφορά την σχεδίαση μιας ιστοσελίδας η οποία θα εξυπηρετεί τις ανάγκες ενός ζωολογικού κήπου. Η υλοποίηση περιλαμβάνει τον σχεδιασμό σελίδων στις οποίες ο χρήστης έχει την δυνατότητα να δει γενικές πληροφορίες για τον κήπο, να δει φωτογραφίες από τα ζώα και να αγοράσει εισιτήρια, αφού περάσει από διαδικασία authentication.

## 2 ΜΕΘΟΔΟΛΟΓΙΑ

### 2.1 Στόχος

Ο κύριος στόχος ήταν η δημιουργία μιας ιστοσελίδας, στην οποία ένας πιθανός επισκέπτης μπορεί να πάρει εικόνα των ζώων που φιλοξενεί ο κήπος και να πραγματοποιήσει αγορά εισιτηρίων σε περίπτωση που θα ήθελε να τον επισκεφτεί. Προτάθηκε στην ενδιάμεση παρουσίαση και επιπλέον λειτουργικότητα από την πλευρά του διαχειριστή του κήπου, η οποία όμως είχε δευτερεύουσα προτεραιότητα και δεν ήταν απαραίτητη στο πλαίσιο της εργασίας.

### 2.2 Χαρακτηριστικά

Η γλώσσα της εφαρμογής είναι εξ ολοκλήρου στα αγγλικά και δεν υπάρχει δυνατότητα εναλλαγής γλώσσας, και η μορφοποίηση είναι σε dark theme. Η εφαρμογή έχει ένα βασικό σενάριο χρήσης, την αγορά εισιτηρίου, ενώ διαχείριση από τη μεριά του προσωπικού δεν υπάρχει, ούτε ανάθεση εργασιών σε υπαλλήλους, κτηνιάτρους ή φύλακες.

Υπάρχει πρόβλεψη να μπορεί ο χρήστης να αγοράσει πολλά εισιτήρια στην ίδια συναλλαγή. Στην περίπτωση που ανήκει σε κοινωνική ομάδα που δικαιούται έκπτωση, αλλά επιλέγεται το αντίστοιχο εισιτήριο, χωρίς να απαιτείται σκανάρισμα κάποιου δημόσιου εγγράφου που να πιστοποιεί τέτοια ιδιότητα. Η διαφοροποίηση των εισιτηρίων γίνεται με βάση την ηλικιακή ή κοινωνική ομάδα και την ημερομηνία έκδοσής τους. Δηλαδή, υπάρχει ένα εισιτήριο για όλες τις περιοχές του ζωολογικού κήπου, δεν υπάρχουν ζώνες πιο ακριβές ή πιο φτηνές.

Όταν ο χρήστης επιλέγει εισιτήρια, θεωρείται πως θα γίνει κράτησή τους για συγκεκριμένη ημέρα, χωρίς διαχωρισμό πρωινής ή απογευματινής επίσκεψης. Τα εισιτήρια που επιλέγει αποθηκεύονται στη βάση δεδομένων για μελλοντική χρήση, όπως το να μπορούν οι διαχειριστές να ελέγχουν πόσοι πελάτες θα επισκεφτούν τον κήπο στις επόμενες μέρες. Παρ' όλα αυτά, δεν υπάρχει η έννοια της διαθεσιμότητας, δηλαδή, όταν ο χρήστης επιλέγει ημερομηνία, δεν γίνεται έλεγχος για το πόσα άτομα έχουν ήδη αγοράσει εισιτήριο για την ίδια μέρα, κάτι που ασφαλώς θα γινόταν σε μια πραγματική εφαρμογή τέτοιου είδους, που δεν θα επέτρεπε να αγορασθούν παραπάνω εισιτήρια από κάποιο ανώτατο όριο.

Κατά τη διαδικασία αγοράς εισιτηρίων, ο server δημιουργεί ένα μοναδικό αναγνωριστικό συναλλαγής, το *paymentID*, το οποίο είναι τα δευτερόλεπτα που έχουν παρέλθει από το λεγόμενο Epoch, μεσάνυχτα της πρωτοχρονιάς του 1970 σε ζώνη ώρας 0 (Universal Time Coordinated). Η τιμή αυτή υπολογίζεται σε milliseconds και στρογγυλοποιείται σε ακέραιο, πράγμα που σημαίνει ότι σε κάθε δευτερόλεπτο υπάρχει διαφορετικό *paymentID*. Ταυτόχρονα, η μετατροπή του πίσω σε ημερομηνία, επιτρέπει στην εφαρμογή να βρει το timestamp του πότε αγοράστηκε.

Εδώ έχει γίνει παραδοχή ότι θα πραγματοποιείται μία το πολύ αγορά εισιτηρίων ανά δευτερόλεπτο. Σε μελλοντικό στάδιο, αυτό θα μπορούσε να αλλάξει και τα *PaymentIDs* να είναι εκφρασμένα σε milliseconds από το Epoch.

Μετά την επιλογή εισιτηρίων, το site ανακατευθύνει στην οθόνη πληρωμής, η οποία σε πραγματικές συνθήκες θα ήταν το ασφαλές περιβάλλον της τράπεζας που συνεργάζεται με τον ζωολογικό κήπο, πράγμα που σημαίνει ότι σε επίπεδο production, η συγκεκριμένη σελίδα θα αντικατασταθεί.

## 2.3 Τεχνολογίες

Για την υλοποίηση της εργασίας χρησιμοποιήθηκαν HTML, CSS, JavaScript [1] για το front end, το framework Node.js [3] για τον server, η βιβλιοθήκη ExpressJS [6] για την εφαρμογή, η Handlebars για τα HTML templates και η SQLite για την βάση δεδομένων.

Έχει χρησιμοποιηθεί το πρότυπο ECMAScript [14] που εκδόθηκε 2015, δηλαδή η έκτη έκδοση ES, με ό, τι χαρακτηριστικά έχει, για παράδειγμα την χρήση του keyword import στα αντίστοιχα statements, εν αντιθέσει με το require του CommonJS. Η χρήση του ES6 φαίνεται από τα extensions των αρχείων JavaScript του server, η οποία είναι \*.mjs. Η επικοινωνία της εφαρμογής με τον browser γίνεται με το fetch API της JavaScript. Ο ενσωματωμένος χάρτης στα footers είναι από το OpenStreetMaps [2].

Ο driver για την SQLite [4] είναι η better-sqlite3 [5], έκδοση 10, η βιβλιοθήκη για το hash είναι η bcrypt [11], έκδοση 5, το middleware για την επεξεργασία cookies είναι το cookie-parser έκδοση 1.4 [15], το render machine είναι το module express-handlebars [9] και το middleware διαχείρισης των session του χρήστη είναι το express-session [16], έκδοση 1.18. Τέλος, χρησιμοποιείται η βιβλιοθήκη fs [17] για την αποθήκευση των αρχείων PDF και η PDFKit [8] για την εξαγωγή των εισιτηρίων σε PDF.

## 3 ΟΡΓΑΝΩΣΗ

Το μοντέλο που χρησιμοποιήθηκε είναι το Model - View – Controller [18] σε ελεύθερη ερμηνεία, όπου *model* είναι η διαχείριση και αποθήκευση των δεδομένων, *view* είναι οι “όψεις” της ιστοσελίδας, δηλαδή τα Handlebars templates και το Handlebars που τα μετατρέπει σε καθαρή html, ενώ *controller* είναι η διασύνδεση των δεδομένων του *model* με το *view*, η λήψη αποφάσεων και η κλήση άλλων λειτουργιών.

### 3.1 Επισκόπηση

Σε μια εφαρμογή του Node.js βασικό εξάρτημα είναι ο router (δρομολογητής) που αντιστοιχίζει τα αιτήματα σε κάποιο URL με κάποια αντίδραση του server, την επιστροφή κάποιου μηνύματος ή κάποιου αρχείου.

Για διευκόλυνση, οι συναρτήσεις που χρησιμοποιεί ο router είναι χωρισμένες σε δύο τμήματα, το πρώτο είναι τα routes που πραγματικά επιστρέφουν στον client κάποια σελίδα, ενώ το δεύτερο λέγεται API (Application Programming Interface) και είναι τα endpoints, όπου γίνονται τα αιτήματα από τους clients και η υποβολή δεδομένων.

Στο αρχείο *route.mjs* υπάρχουν δύο classes, η ROUTE και η API, καθεμία από τις οποίες περιέχει τις συναρτήσεις για τον router ως static methods. Μετά από το declaration της κάθε κλάσης, δηλώνονται στο αντικείμενο router οι URL διαδρομές, η μέθοδος του αιτήματος, ως επί το πλείστον GET και POST και η συνάρτηση που θα καλεί ο router όταν θα φτάνει αίτημα στο συγκεκριμένο route. Σημειώνεται ότι υπάρχει δυνατότητα να κληθούν περισσότερες συναρτήσεις από μία στη σειρά, εξ ου και η ονομασία middleware.

Στο παρασκήνιο, το αντικείμενο express που είναι ο πυρήνας της εφαρμογής, καλεί διάφορες συναρτήσεις middleware, πριν τις custom που είναι γραμμένες για το παρόν use case στις κλάσεις ROUTE και API. Περισσότερα στοιχεία για αυτό υπάρχουν στο αρχείο *server.mjs*, στο οποίο ορίζεται το configuration της εφαρμογής, εκεί φαίνεται η σειρά με την οποία θα καλούνται για κάθε αίτημα τα middleware functions για τα cookies, την ανάγνωση αρχείων JSON, την επεξεργασία των data URIs και τη ρύθμιση του session.

Όσον αφορά το model, υπάρχουν δύο βάσεις δεδομένων SQLite, η storage.sqlite για τις πληροφορίες που χρειάζονται για τους χρήστες και τα εισιτήρια και η session.sqlite που τη διαχειριζόταν το built in session middleware σε δοκιμαστικά στάδια της εφαρμογής για να αποθηκεύει πληροφορίες για τα sessions, δεν χρησιμοποιείται. Περισσότερα στοιχεία για τη βάση και την κλάση που τη διαχειρίζεται, υπάρχουν στην ομόνυμη ενότητα.

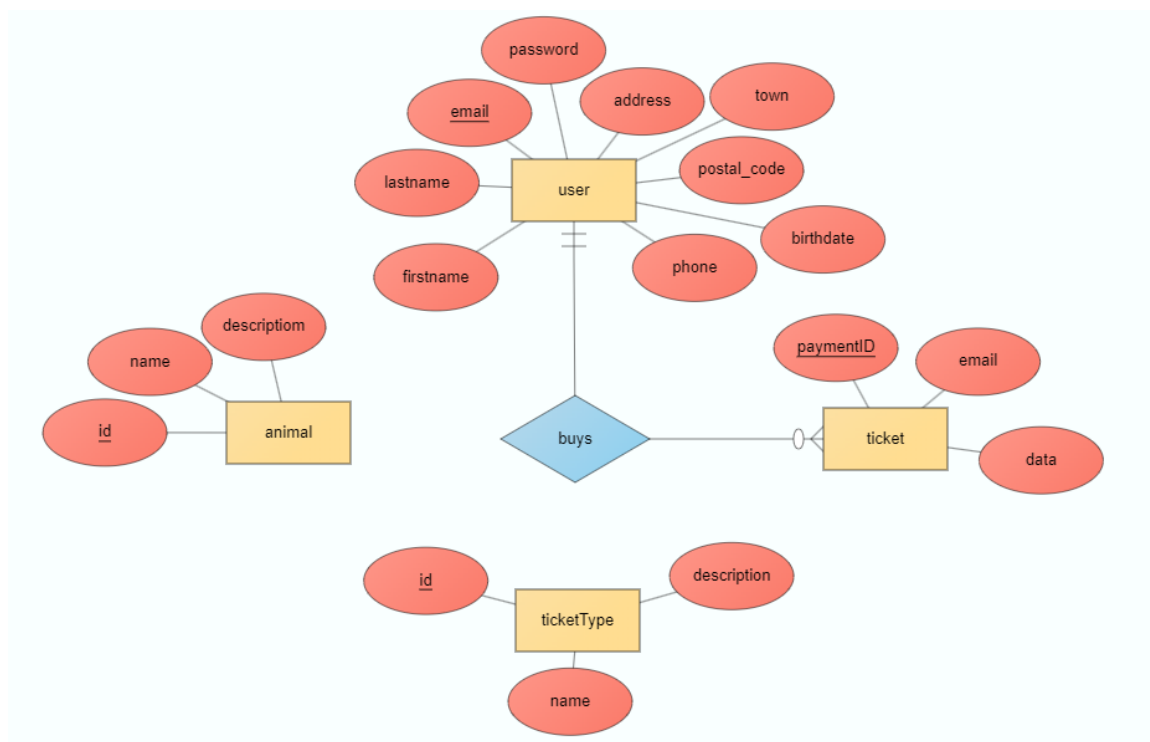
Στον φάκελο του controller δεν υπάρχουν συναρτήσεις που να πραγματοποιούν login, log out ή register, όπως θα ήταν το αναμενόμενο, αυτές οι λειτουργίες υπάρχουν απευθείας στις συναρτήσεις της κλάσης API στον router, με άλλα λόγια, σε αυτό το σημείο το Model – View - Controller pattern δεν έχει εφαρμοστεί πλήρως. Στον controller περιέχονται δύο αρχεία, το πρώτο περιέχει την κλάση *Accountant* που ρυθμίζει την πρόοδο της αγοράς εισιτηρίου, δημιουργεί *paymentID* και ελέγχει τα στοιχεία κάρτας κατά την πληρωμή, στο δεύτερο που λέγεται *invoice.mjs*, υπάρχουν συναρτήσεις που δημιουργούν ένα PDF με την απόδειξη των εισιτηρίων για να το αποθηκεύσει ο χρήστης στη συσκευή του. Περισσότερα στοιχεία για τη λειτουργία των δύο αυτών αρχείων βρίσκονται στην ενότητα που πραγματεύεται την αγορά εισιτηρίων.

### 3.2 Βάση Δεδομένων

Πίνακας 1: Πίνακες της βάσης

Table Name	Table Columns
animal	integer <b>id</b> , text <b>name</b> , text <b>description</b>
ticket	integer <b>paymentID</b> , text <b>email</b> , text <b>data</b>
ticketType	integer <b>id</b> , text <b>name</b> , text <b>description</b> , integer <b>price</b>
user	text <b>firstname</b> , text <b>lastname</b> , text <b>address</b> , text <b>town</b> , text <b>postal_code</b> , text <b>birthdate</b> , text <b>phone</b> , text <b>email</b> , text <b>password</b>

Για τον σχεδιασμό του μοντέλου οντοτήτων συσχετίσεων της βάσης δεδομένων χρησιμοποιήθηκε το ERD Maker [13]:



Εικόνα 1: Διάγραμμα Συσχετίσεων και Οντοτήτων.

*Η βάση δεν έχει πολλές συσχετίσεις μεταξύ των πινάκων της, η προσέγγιση που ακολουθήθηκε για τις λειτουργίες της παρούσας εφαρμογής και τα σενάρια χρήσης της δεν το απαιτούσαν.*

### 3.2.1 Πίνακες

Ο πίνακας *animal* αποθηκεύει πληροφορίες για τα ζώα που φιλοξενούνται στον κήπο, το όνομα του καθενός και ένα σύντομο κείμενο με βασικά στοιχεία για τις γεωγραφικές περιοχές που ζει, την εμφάνιση και τη διατροφή του. Οι εικόνες των ζώων αποθηκεύονται σε φάκελο με τα ονόματά τους, οπότε η αντιστοίχιση γίνεται εύκολα, χωρίς να αποθηκεύεται filename στον πίνακα.

Τα δεδομένα για τα ζώα της εφαρμογής, δηλαδή οι φωτογραφίες και τα ονόματά τους, πάρθηκαν από την ιστοσελίδα του Αττικού Ζωολογικού Πάρκου [7], ενώ πληροφορίες για το καθένα αντλήθηκαν από τη Wikipedia [12].

Ο πίνακας *user* αποθηκεύει στοιχεία για τον χρήστη της εφαρμογής, τα οποία υποβάλλει αυτός στην βάση, μέσω της φόρμας στη σελίδα *register*. Συγκεκριμένα, αποθηκεύονται το όνομα, το επώνυμο, το email (το οποίο είναι primary key), ο hashed κωδικός, η διεύθυνση, η πόλη, ο ταχυδρομικός κώδικας, η ημερομηνία γέννησης και το τηλέφωνό του.

Καταχρηστικά, στον πίνακα *user* αποθηκεύονται και οι συνδρομητές του newsletter, οι οποίοι δεν θεωρούνται χρήστες του site και δεν μπορούν να συνδεθούν. Ο διαχωρισμός γίνεται εύκολα, οπότε δεν απαιτείται ξεχωριστός πίνακας για το newsletter. Η κλάση που διαχειρίζεται τη βάση, ξεχωρίζει τους newsletter users επειδή στην εγγραφή τους το μοναδικό συμπληρωμένο πεδίο είναι το email, οπότε ένα query του τύπου “select \* from user where password is null” μας δίνει τους subscribers.

Ο πίνακας *ticketType* περιλαμβάνει το id, το όνομα και μια σύντομη περιγραφή του κάθε εισιτηρίου που μπορεί να εκδώσει ο κήπος για τις διάφορες ηλικιακές και κοινωνικές ομάδες.

Ο πίνακας *ticket* αποθηκεύει με κάθε επιτυχή έκδοση εισιτηρίου το id του τιμολογίου, το email του χρήστη που πραγματοποίησε τη συναλλαγή, καθώς και ένα JSON αντικείμενο σε μορφή string, το data, το οποίο φυλάσσει την ημερομηνία που έγινε η έκδοση, ποια και πόσα από το καθένα εισιτήρια αγόρασε ο αντίστοιχος χρήστης.

### 3.2.2 Δομή του ticket

Κατά τη δημιουργία της εφαρμογής, ο πίνακας *ticket* ήταν ο τελευταίος πίνακας που φτιάχτηκε, καίτοι ο *ticketType* ήταν ο πρώτος, αρχικά ως ένα Array αποθηκευμένο σαν class field της κλάσης *Database*. Για τη δομή του *ticket* υπήρχαν δύο ενδεχόμενα. Ακολουθεί παράδειγμα για τετραμελή οικογένεια:

Πρώτο ενδεχόμενο

#### ticket

<u>paymentID</u> (PK)	<u>email</u> (FK bound to user)	<u>date</u>
12345678	theiremail@domain.com	2024-05-31

#### ticketRecord

<u>paymentID</u> (FK bound to Ticket)	<u>ticket_type</u> (FK bound to ticketType)	<u>quantity</u>
12345678	1 (means child ticket)	2
12345678	2 (means adult ticket)	2

Δεύτερο Ενδεχόμενο

#### ticket

<u>paymentID</u> (PK)	<u>email</u> (FK bound to user)	<u>data</u>
12345678	theiremail@domain.com	{"date": "2024-05-31", "child": "2", "adult": "2"}

Χάριν ευκολίας, προτιμήθηκε το δεύτερο ενδεχόμενο που αποθηκεύει σε μία μόνο εγγραφή της βάσης όλες τις πληροφορίες συναλλαγής εισιτηρίου και αντικαθιστά τα πεδία του *ticketRecord* με ένα JSON string, το οποίο η JavaScript μετατρέπει σε object εύκολα. Στην πρώτη περίπτωση, θα χρειαζόταν ένα join query για να αντληθούν όλες οι πληροφορίες σχετικά με μία συναλλαγή, θα ήταν όμως πιο εύκολο να πραγματοποιηθεί ένα query που, για παράδειγμα, θα έβρισκε όλες τις συναλλαγές για συγκεκριμένη ημερομηνία που έχουν τουλάχιστον ένα παιδικό εισιτήριο. Στη δεύτερη περίπτωση που επιλέχθηκε, ένα query τέτοιου τύπου δεν μπορεί να γίνει αποκλειστικά από την SQLite, θα χρειαστεί η κλάση database να πραγματοποιήσει loop για όλα τα εισιτήρια εκείνης της μέρας και, τελικά, να κρατήσει μόνο όσα θα έχουν παιδικά εισιτήρια, κάτι που μπορεί να αποβεί πιο αργό σε μεγάλο όγκο δεδομένων.

### 3.2.3 Διαχειρίστρια κλάση

Στον φάκελο του *model* υπάρχει το αρχείο *database.mjs*, στο οποίο έχει γραφτεί η ομώνυμη κλάση, η *Database*, υπεύθυνη για τα queries, τη γραφή και την ανάγνωση δεδομένων. Ακολουθώντας τις απαιτήσεις της εφαρμογής, το μοντέλο CRUD δεν έχει εφαρμοστεί πλήρως, για παράδειγμα δεν υπάρχει λειτουργία διαγραφής χρήστη ή ακύρωσης εισιτηρίου, μετά την απομάκρυνση από το ταμείο, ουδέν λάθος αναγνωρίζεται. Επίσης, δεν υπάρχει δυνατότητα ανανέωσης στοιχείων υπάρχοντος χρήστη, παρ' ότι αυτό υπήρχε στον αρχικό σχεδιασμό.

Σε αντίθεση με τις κλάσεις ROUTE και API, οι συναρτήσεις της *Database* είναι όλες instance methods, ενώ το connection με την SQLite είναι class field. Στο τέλος του αρχείου *database* δημιουργείται ένα αντικείμενο *database*, για να γίνει import, όπου χρειαστεί. Οι μισές μέθοδοι του αντικειμένου *database* για πρακτικούς λόγους υπολογίζουν κάτι ή διαβάζουν κάποια δεδομένα από τη βάση και τα επιστρέφουν. Οι άλλες μισές μέθοδοι, για εκπαιδευτικούς λόγους, ακολουθούν τη φιλοσοφία του callback, δηλαδή δεν επιστρέφουν κάτι, παρά μόνον κάνουν κλήση σε ένα callback function με το αποτέλεσμα της εκτέλεσής τους.

Μερικές μέθοδοι που δεν δέχονται arguments, έχουν δηλωθεί ως getter methods για να καλούνται χωρίς τις παρενθέσεις, μιας και η JavaScript δίνει αυτή τη δυνατότητα (getterMethod; - nonGetterMethod();). Όπως και να 'χει, στα docstrings της κάθε μεθόδου είναι εμφανή τα arguments, τα types που δέχονται, το αν επιστρέφεται κάτι ή όχι, το αν απαιτείται callback function, και αν ναι, τι parameters αναμένεται να περαστούν στην κλήση του.

## 4 ΔΙΑΔΡΟΜΕΣ BROWSER

<u>Homepage:</u>	Η αρχική σελίδα της εφαρμογής, προβάλλει γενικές πληροφορίες για τον κήπο, έχει ένα carousel με φωτογραφίες, header και footer τα οποία είναι κοινά σε όλες τις σελίδες.
<u>Tickets:</u>	Ο χρήστης μπορεί να επιλέξει τα εισιτήρια που επιθυμεί και στη συνέχεια να πραγματοποιήσει την έκδοσή τους, εφόσον είναι συνδεδεμένος.
<u>Animals:</u>	Ο χρήστης μπορεί να δει φωτογραφίες και πληροφορίες για τα ζώα που διαθέτει ο κήπος, αυτό γίνεται μέσω ενός αναδυόμενου παραθύρου που εμφανίζεται κάθε φορά που ο χρήστης κάνει κλικ σε ένα ζώο.
<u>About:</u>	Πληροφορίες για το ποια είναι η ιστοσελίδα και οι δημιουργοί της.
<u>Dashboard:</u>	Σε αυτή τη σελίδα ένας διαχειριστής έχει διάφορες δυνατότητες, προβολή χρηστών, εισιτηρίων που έχουν αγοραστεί για συγκεκριμένη μέρα, συνδρομητές στο newsletter, αποστολή email σε αυτούς, δημιουργία σημειώσης, και άλλες λειτουργίες που δεν έχουν υλοποιηθεί.
<u>Payment:</u>	Σελίδα πληρωμής, προσβάσιμη μετά τη σελίδα επιλογής εισιτηρίων, ο router δεν την στέλνει αν δεν πληρούνται προϋποθέσεις.

## 5 ΣΕΝΑΡΙΟ ΧΡΗΣΗΣ

### 5.1 Διεπαφή

Ένας υποθετικός χρήστης μπαίνει για πρώτη φορά στην ιστοσελίδα του ζωολογικού κήπου. Έχει τη δυνατότητα να βρει στοιχεία επικοινωνίας, τηλέφωνο και διεύθυνση, ενώ μπορεί να αφήσει το δικό του email για να είναι συνδρομητής στο newsletter του κήπου. Στη σελίδα *animals* μπορεί να δει φωτογραφίες των ζώων που φιλοξενούνται στις εγκαταστάσεις του κήπου και να διαβάσει μια σύντομη περιγραφή του καθενός.

Αν το επιθυμεί, μπορεί να εγγραφεί χρήστης με το email του και στη συνέχεια να πάει στη σελίδα των εισιτηρίων. Σε περίπτωση που επιλέξει εισιτήρια και δεν είναι συνδεδεμένος, η εφαρμογή του ζητά να συνδεθεί στο αντίστοιχο παράθυρο διαλόγου. Στη συνέχεια εμφανίζεται η σελίδα πληρωμής που προσομοιάζει το περιβάλλον τράπεζας, χωρίς φυσικά να πληροί προδιαγραφές ασφαλείας ή κάποιας μορφής κρυπτογράφηση για τα στοιχεία της κάρτας του. Αυτό είναι κάτι που οι browsers αντιλαμβάνονται, οπότε αυτόματα δεν επιτρέπουν την αυτόματη συμπλήρωση των αληθινών στοιχείων κάρτας.

Για λόγους παρουσίασης, η σελίδα πληρωμής σχεδιάζει μια SVG τραπεζική κάρτα στα αριστερά και μια φόρμα στα δεξιά, με ένα κουμπί που εισάγει τυχαία στοιχεία κάρτας. Στην υποβολή, ο server πραγματοποιεί υποτιθέμενο έλεγχο των στοιχείων της κάρτας, όπως το να μην έχει παρέλθει ο μήνας λήξης της, χωρίς να γίνεται κανένας έλεγχος για την τιμή των εισιτηρίων. Φυσικά, δεν υπάρχει δυνατότητα να ελέγξει το υπόλοιπο της συγκεκριμένης κάρτας ή να το δεσμεύσει.

Αν πληρούνται τα κριτήρια του παραπάνω εικονικού ελέγχου, η συναλλαγή θεωρείται επιτυχημένη, δημιουργείται ένα αρχείο PDF με την απόδειξη της αγοράς και ο browser το προβάλλει σε νέο παράθυρο, ενώ η οθόνη πληρωμής ανακατευθύνεται σε μια άλλη που ευχαριστεί τον χρήστη για την αγορά και τον καλεί να επιστρέψει στην αρχική σελίδα.

### 5.2 Λειτουργία σύνδεσης

*Ακολουθεί μια αναλυτική περιγραφή της αλληλεπίδρασης του server και του client στην περίπτωση του κοινότοπου login.*

Ο χρήστης έχει την δυνατότητα να συνδεθεί στην εφαρμογή με το email και τον κωδικό του από οποιαδήποτε σελίδα και αν βρίσκεται, πατώντας το κουμπί login που βρίσκεται στο δεξί τμήμα του header, σε περίπτωση που δεν είναι ήδη συνδεδεμένος, γιατί διαφορετικά, τα Handlebars το αντικαθιστούν με ένα κουμπί για log out. Πατώντας το κουμπί εμφανίζεται το παράθυρο διαλόγου login στο οποίο ο χρήστης βάζει τα στοιχεία του. Στην υποβολή, αν δεν υπάρχει κενό πεδίο, στέλνεται ένα POST request μέσω του fetch API [1] στη διεύθυνση `api/login` που υπάρχει για αυτόν τον σκοπό.

Το request έχει για headers το Content Type και το Accept, και τα δύο με τιμή `application/json`, με την οποία ο client δηλώνει σε ποια μορφή είναι το resource που στέλνει και τι επιθυμεί ως απάντηση. Το body επομένως, είναι ένα αντικείμενο JSON το οποίο περιέχει το email, το password και το boolean `remember me`.

Η συνάρτηση middleware του server που επεξεργάζεται τα POST requests για το `api/login`, καλεί τη μέθοδο `checkUser` του `database`, περνώντας και ένα callback function με parameters το μήνυμα αποτελέσματος του ελέγχου και τον χρήστη που ενδέχεται να βρεθεί στη βάση.

Η εσωτερική λειτουργία της `checkUser` ενθυλακώνεται στην κλάση `Database`, οπότε εν προκειμένω δεν μας απασχολεί να μπούμε σε λεπτομέρειες, πέρα από το ότι καλεί με τη σειρά της μια συνάρτηση της `bcrypt` [19] που εσωτερικά μετατρέπει σε hash τον κωδικό που πληκτρολόγησε ο χρήστης και ελέγχει αν ταιριάζει με τον hash κωδικό της βάσης.



Στην περίπτωση που το callback καλείται με not null μήνυμα, στέλνεται ένα response με JSON που περιέχει ένδειξη αποτυχίας και το μήνυμα αυτό, το οποίο θα είναι “Λανθασμένος κωδικός” ή “Ο Χρήστης δεν βρέθηκε”, το front end θα διαβάσει και θα ενημερώσει τον χρήστη.

Στην περίπτωση που το callback καλείται με null μήνυμα και not null user, πραγματοποιείται σύνδεση, δηλαδή αποθηκεύεται στο υφιστάμενο session μια boolean μεταβλητή `session.signedIn` με τιμή true και μια string μεταβλητή ονόματι `session.email` με το email του χρήστη.

Αν τώρα αυτό το email είναι το email του admin (αν κάποιος συνδεθεί με το email: admin@email.com και βάλει κωδικό password) τότε γράφεται άλλη μια boolean μεταβλητή, η `session.admin`, με τιμή true, οπότε ο χρήστης αυτός έχει δικαιώματα admin. Στη συνέχεια, στέλνεται και πάλι ένα JSON που ενημερώνει το front end ότι έγινε επιτυχής σύνδεση, χωρίς κάποιο μήνυμα. Σημειώνεται ότι οι μεταβλητές που παραπάνω αποθηκεύτηκαν στο αντικείμενο session, δεν είναι προσβάσιμες στην JavaScript του front end, σε αντίθεση με τα cookies.

Φυσικά το να συγκρίνεται το email που γράφτηκε με τα emails των διαχειριστών, είναι πολύ πρόχειρη λύση, η σωστή πρακτική εδώ από τη μεριά της βάσης, θα ήταν να υπάρχει κατηγοριοποίηση για τους χρήστες, σε υπαλλήλους, πελάτες, συνδρομητές για το newsletter, διαχειριστές, φύλακες και ούτω καθεξής. Η δημιουργία ενός πίνακα `userType` είχε συμπεριληφθεί στον αρχικό σχεδιασμό της εφαρμογής αλλά τελικά έγινε εστίαση αλλού, και όχι στη σύνδεση υπαλλήλων του κήπου, παρά μόνο ενδιαφερόμενων πελατών.

Όσο γίνονται αυτά στον server, το front end έχει εκτελέσει την ασύγχρονη συνάρτηση `fetch` και περιμένει απάντηση, έχοντας τη δυνατότητα να κάνει κάτι άλλο μέχρι τότε, η έννοια της ασύγχρονης συνάρτησης είναι ότι γίνεται `awaited` και όχι `called`, δεν μπλοκάρει την εκτέλεση του προγράμματος. Χρησιμοποιήθηκε η παλαιότερη σύνταξη για ασύγχρονες συναρτήσεις, δηλαδή `chained Promises`, και όχι το `block` με τα keywords `async` και `await`, κυρίως γιατί ήταν συμβατό με τα περισσότερα παραδείγματα χρήσης του `fetch API` στο διαδίκτυο.

Όταν έρχεται η απάντηση του server, η JavaScript στον client καλεί την ασύγχρονη `json()` που επεξεργάζεται το JSON string και το κάνει object, διαβάζει το αποτέλεσμα, σε περίπτωση μηνύματος αποτυχίας το γράφει με έντονο χρώμα στο παράθυρο διαλόγου που έχει παραμείνει ανοιχτό, σε περίπτωση επιτυχούς σύνδεσης, το κλείνει και ένα άλλο ενημερωτικό παράθυρο διαλόγου ενημερώνει τον χρήστη πως συνδέθηκε.

Το κουμπί `log out` στο πάτημά του στέλνει επίσης ένα request στη διεύθυνση `api/login`, αυτή τη φορά όμως με τη μέθοδο `DELETE`, χωρίς κάποιο body. Ο router στέλνει ένα τέτοιο αίτημα σε άλλη συνάρτηση `middleware` που απλά απαντά με status code 200 [20] σε περίπτωση επιτυχούς αποσύνδεσης, ή με 400 [Bad request] σε περίπτωση αποτυχίας. Ο κωδικός 400 έχει επιλεγεί γιατί υπό κανονικές συνθήκες, ποτέ ο server δεν θα απαντήσει έτσι.

### 5.3 Loops στη σελίδα animals

Όπως προαναφέρθηκε, ο χρήστης ανά πάσα στιγμή καταλαβαίνει αν είναι συνδεδεμένος κοιτώντας το header, δεξιά της γραμμής πλοήγησης, στη μία περίπτωση φαίνονται τα κουμπιά `Login` και `Register`, στην άλλη το `Log out`. Αυτό γίνεται χάρη σε μεταβλητή του `Handlebars` που τροποποιεί το HTML του header ανάλογα, ενώ είναι κάτι που θα μπορούσε πιο σύνθετα, να κάνει και η JavaScript, αν σε κάθε φόρτωση σελίδας έστελνε ένα ακόμα request στον server για να γνωρίζει αν ο χρήστης είναι συνδεδεμένος.

Στο αρχείο `animals` αποτυπώνεται η συνύπαρξη δυνατοτήτων που παρέχει η JavaScript και το `Handlebars`. Φαίνονται δύο διαφορετικές προσεγγίσεις για τη λύση του ίδιου προβλήματος, το front end δεν γνωρίζει εκ των προτέρων πόσα θα είναι

τα αποθηκευμένα ζώα στη βάση, οπότε γίνεται ένα loop του Handlebars για κάθε element μιας λίστας με τα ονόματα και τις περιγραφές τους, που ζητά και τις εικόνες τους και την ίδια ώρα γράφει το on click EventListener για όταν πατιέται η εικόνα τους.

Παρακάτω και όταν έχουν μπει στο Document Object Model όλα τα ζώα, γίνεται ένα διπλό loop, στη Javascript αυτή τη φορά, για κάθε element της κλάσης animal, το οποίο loop δημιουργεί το εφέ που όταν γίνεται hover πάνω από μία εικόνα, σκουραίνουν οι εικόνες όλων των υπολοίπων.

Με μερικές προσθήκες, το loop που γίνεται στα Handlebars θα μπορούσε να αντικατασταθεί από JavaScript που κατά τη φόρτωση θα έκανε αίτημα με τη βοήθεια του fetch API για ένα JSON με τα στοιχεία των ζώων και θα έφτιαχνε τα πλαίσιά τους. Εκτιμήθηκε ότι αυτή η λύση ενδεχομένως θα προκαλούσε πιο αργή φόρτωση της σελίδας.

Το fetch API [1] της JavaScript χρησιμοποιείται σε διάφορα σημεία της εφαρμογής που το front end στέλνει δεδομένα στον server και ανάλογα με την απάντηση, πράττει ανάλογα. Η απάντηση του server είναι είτε κάποιο status code, είτε μήνυμα σε αρχείο JSON. Στην πρώτη περίπτωση, τα HTTP status codes που έχουν επιλεγεί [20], έχουν ειδική σημασία για την εφαρμογή, που πιθανόν να μην συνάδει με τον λόγο ύπαρξής τους. Έγινε προσπάθεια να είναι συμβατοί με τις σημασίες που έχει προσδιορίσει η IANA για τον καθέναν.

## 5.4 Hashing

Η εξωτερική βιβλιοθήκη bcrypt που παρέχει την λειτουργία hashing, αποτελεί υλοποίηση σε JavaScript του ομώνυμου αλγορίθμου που δημοσιεύτηκε στην πρώτη έκδοσή του το 1999 [11], έχει γραφτεί κατά καιρούς σε πολλές γλώσσες και έχει υπάρξει ο προεπιλεγμένος hashing αλγόριθμος για διάφορα λειτουργικά συστήματα. Όταν δημιουργεί hash για ένα string, χρησιμοποιεί το πολύ 72 bytes του, αυτό δεν σημαίνει απαραίτητα 72 χαρακτήρες, για παράδειγμα στην κωδικοποίηση UTF-8 ένας χαρακτήρας μπορεί να παρασταθεί με ένα ως τέσσερα bytes.

Οι hash κωδικοί που αποθηκεύονται στη βάση έχουν μήκος 60 χαρακτήρων, οι 31 τελευταίοι είναι το hash της εισόδου, και οι 22 προηγούμενοι είναι το salt που δημιουργήθηκε. Στο πρόθεμα \$2b φαίνεται ότι η έκδοση του αλγορίθμου που χρησιμοποιείται είναι η τελευταία, διαθέσιμη από τον Φεβρουάριο του 2014. Το πρόθεμα \$10 που ακολουθεί, σημαίνει ότι οι κωδικοί έχουν γίνει hash με  $2^{10}$  κύκλους (salt rounds). Όπως εξηγείται στο documentation της βιβλιοθήκης [19][11], 10 salt rounds για έναν επεξεργαστή συχνότητας 2GHz, δηλαδή 1024 κύκλοι, σημαίνει πως το κάθε hash ολοκληρώνεται σε περίπου 100 milliseconds. Υψηλότερη τιμή salt rounds καθιστά όλο και πιο ανέφικτη μια επίθεση brute force. Τα saltRounds θα είναι το πολύ διψήφια, καθώς εκτιμάται πως με τιμή 30 το hashing θα ολοκληρωθεί μετά από δύο μέρες.

## 5.5 Η κλάση Accountant

Όταν ο χρήστης ολοκληρώνει την επιλογή εισιτηρίων στην σελίδα tickets και πατά υποβολή, η συνάρτηση *ticketsSelected* της API καλεί τη μέθοδο *accountant.generatePaymentID()* για να δημιουργηθεί το αναγνωριστικό της συναλλαγής και στη συνέχεια την *accountant.save()*, με παραμέτρους το *paymentID*, το email του χρήστη και τις πληροφορίες για τα εισιτήρια που έχει επιλέξει. Στην περίπτωση που δεν έχει συνδεθεί χρήστης, ο server θα ειδοποιήσει το front end ότι πρέπει να πραγματοποιηθεί login, οπότε θα καλέσει την *save()* μόνο με το *paymentID* και τις πληροφορίες για τα εισιτήρια.

Όταν πραγματοποιηθεί επιτυχώς το login, σε περίπτωση που στο session βρεθεί μεταβλητή *paymentID*, η συνάρτηση middleware που διαχειρίζεται το login, καταλαβαίνει ότι ο χρήστης συνδέθηκε για να αγοράσει εισιτήρια, οπότε καλεί εκ νέου την *accountant.save()*, αυτή τη φορά με παραμέτρους το *paymentID* και το email.

Η *accountant.save()* που φυλά τα δεδομένα αυτά στο array *briefcase* της κλάσης *Accountant* είναι γραμμένη ώστε να έχει διαφορετική συμπεριφορά, ανάλογα με το αν έχει κληθεί με παράμετρο ένα email και εισιτήρια, μόνο εισιτήρια ή μόνο email, καθώς στην περίπτωση που ο χρήστης επιλέγει εισιτήρια χωρίς να έχει συνδεθεί στην εφαρμογή, αυτά θα αποθηκευτούν μόνο τους στο *briefcase* της *Accountant* και το email θα προστεθεί αργότερα στην ίδια εγγραφή.

Όταν πραγματοποιηθεί πληρωμή στην σελίδα *payment*, η συνάρτηση του API που διαχειρίζεται την υποβολή της, καλεί την *accountant.checkCard()*, η οποία ελέγχει αν ο αριθμός της κάρτας είναι δεκαεξαψήφιος, αν ο κωδικός είναι τριψήφιος και αν ο μήνας λήξης της έχει παρέλθει, δηλαδή τα έγκυρα στοιχεία της κάρτας σημαίνουν επιτυχή πληρωμή. Όταν γίνει αυτό, καλείται η *accountant.invoice()*, η οποία αποθηκεύει τα στοιχεία της αγοράς στον πίνακα *ticket* της βάσης μέσω της *database.saveTicket()*, καλεί την *accountant.data()* για να δημιουργήσει ένα αντικείμενο με ορισμένο format και καλεί την *createInvoice()* του αρχείου *invoice.js* με παράμετρο το αντικείμενο αυτό. Η *createInvoice()* θα φτιάξει το PDF και θα το αποθηκεύσει σε συγκεκριμένο φάκελο, με όνομα αρχείου *[paymentID].pdf*.

Δεν υπάρχει λόγος στην ολοκλήρωση της διαδικασίας να σβήνονται οι πληροφορίες για το εισιτήριο από το *briefcase*, γιατί αυτό θα γίνει σε κάθε restart του server, έτσι και αλλιώς, τα πάντα αποθηκεύονται και στη βάση, σε περίπτωση που χρειαστούν ξανά.

Μετά την δημιουργία του PDF, καλείται η συνάρτηση *document* της API, η οποία βρίσκει στο *session* την τιμή του *paymentID* και απαντά στον client με ένα JSON που περιέχει την ένδειξη επιτυχίας και το όνομα του αρχείου. Η JavaScript στον browser ανοίγει το συγκεκριμένο αρχείο σε νέο παράθυρο και ανακατευθύνεται σε νέα οθόνη που ευχαριστεί τον χρήστη για την αγορά που έκανε.

## 5.6 Αυτόματη σύνδεση

Πέραν από την προσωρινή αποθήκευση δεδομένων σε μεταβλητές του *session*, στο εσωτερικό των κλάσεων *Database* και *Accountant* και την μόνιμη αποθήκευση στη βάση, υπάρχει η δυνατότητα να αποθηκεύεται η ίδια η σύνδεση του χρήστη, με ένα cookie στον browser.

Αν, κατά τη διάρκεια του login ο χρήστης επιλέξει το σχετικό check box, η middleware συνάρτηση που επεξεργάζεται αιτήματα στο *api/login* θα δημιουργήσει ένα cookie με value το email του χρήστη και διάρκεια ζωής τέσσερις ώρες. Όταν επιστρέψει στον client η απάντηση, το cookie αυτό θα αποθηκευτεί στη μνήμη του browser. Την επόμενη φορά που ο χρήστης θα μπει στο site, η middleware συνάρτηση *autoLogin* που καλείται κατά τη διάρκεια request σε οποιαδήποτε σελίδα της εφαρμογής, θα διαβάσει το cookie με το email, αν αυτό υπάρχει ακόμα, και θα συνδέσει τον χρήστη αυτόματα.

Η ύπαρξη της συνάρτησης αυτής θα μπορούσε να είναι ενοχλητική στο ενδεχόμενο που ο χρήστης συνδέεται με το cookie αλλά θέλει να αποσυνδεθεί με το κουμπί log out. Αν συμβεί αυτό, καλείται η συνάρτηση *database.intentionalLogout()* με παράμετρο το email αυτό και το αποθηκεύει στο array *database.cacheIntentionalLogout*. Η ύπαρξη ενός email στο array αυτό αποτρέπει την *autoLogin* να συνδέσει αυτόματα τον χρήστη, ακόμα και όταν βρίσκει το email του στα cookies.

## 6 ΧΡΟΝΟΔΙΑΓΡΑΜΜΑ

Στα μέσα Μαρτίου γράφτηκαν τα πρώτα αρχεία HTML και αποκτήθηκε μέχρι έναν βαθμό εξοικείωση με τη CSS, μέχρι το τέλος του μήνα προστέθηκε JavaScript σε διάφορα σημεία του front end.

Τον Απρίλιο ξεκίνησε η ανάπτυξη του server με την χρήση node express, η μετατροπή των HTML αρχείων σε Handlebars και η μελέτη των βιβλιοθηκών που παρέχει το npm. Το header έγινε responsive σε διαφορετικές κατηγορίες οθονών, το ίδιο και το footer, το οποίο βρέθηκε έτοιμο στο internet και μεταγράφηκε για να ταιριάζει στις ανάγκες της ιστοσελίδας.

Τον Μάιο δημιουργήθηκε η βάση δεδομένων καθώς και οι συναρτήσεις που εκτελούν queries για την αλληλεπίδραση με αυτή. Μετά από πολυήμερες προσπάθειες ρυθμίστηκαν τα sessions και λειτούργησαν τα login και log out. Πάνω στη μεταβλητή session, γράφτηκαν όσες χρειάστηκαν για να διατηρούν δεδομένα από το προηγούμενο request στο επόμενο και το σενάριο χρήσης ολοκληρώθηκε με την οθόνη πληρωμής, την δημιουργία PDF και μια τελική αναβάθμιση στην εμφάνιση των σελίδων.

Παράλληλα ολοκληρώθηκαν οι middleware συναρτήσεις της ROUTE και της API, που ορίζουν τη συμπεριφορά της εφαρμογής, οπότε οι κλάσεις αυτές μετακινήθηκαν στο νέο αρχείο router.mjs. Στη συνέχεια στον controller γράφτηκε η κλάση Accountant, βρέθηκαν στο διαδίκτυο οι συναρτήσεις για τη δημιουργία των αρχείων PDF και προσαρμόστηκαν στις ανάγκες της εφαρμογής.

## 7 ΑΞΙΟΛΟΓΗΣΗ

### 7.1 Επέκταση login

Η εργασία ήταν αφορμή για αρκετές ώρες έρευνας πάνω στη σύνταξη της JavaScript, τη σύγκριση λειτουργιών της με άλλες γλώσσες και την γενικότερη φιλοσοφία της λειτουργίας μιας μικρής web εφαρμογής. Ένα από τα τελευταία χαρακτηριστικά που προστέθηκαν ήταν το να λειτουργεί το *remember me* και να αποθηκεύει ένα cookie στον browser για αυτόματο login σε επόμενες συνεδρίες.

Με δεδομένο επιπλέον χρόνο, η πρώτη προσθήκη που θα γινόταν, θα ήταν, πάλι στην κατηγορία αυτή, το χαρακτηριστικό *forgot your password*, το οποίο υπάρχει σε όλες τις εφαρμογές με διαχείριση χρηστών, συνυπάρχοντας με μια καρτέλα επεξεργασίας που ο χρήστης μπορεί να επεξεργαστεί τα στοιχεία του, να αλλάξει το email του και τον κωδικό του. Μάλιστα, στη δική μας περίπτωση η αλλαγή email θα είχε μια παραπάνω προγραμματιστική δυσκολία, καθώς νέο email θα σήμαινε νέο primary key στον πίνακα χρηστών, το οποίο πλέον δεν θα συσχετιζόταν άμεσα με συναλλαγές που έχει κάνει ο χρήστης. Ένας νέος πίνακας που θα συνέδεε παλιά και νέα emails θα ήταν μια πρώτη λύση, αν και το σωστό θα ήταν να προστεθεί id στον πίνακα *user*, και να πάρει αυτό τη θέση του πρωτεύοντος κλειδιού.

Το *forgot your password* θα ήταν ένα διακριτικό anchor στο παράθυρο διαλόγου login, το οποίο θα ζητούσε από τον χρήστη να πληκτρολογήσει λόγω χάριν το τηλέφωνό του, στη συνέχεια η εφαρμογή θα του έστελνε SMS επιβεβαίωσης, θα πατούσε ένα link γνωρίζοντας ότι είναι ασφαλές και θα ολοκλήρωνε την επαναφορά κωδικού ορίζοντας έναν νέο.

Αν παρατηρήσει κανείς τη συμπεριφορά αληθινών εφαρμογών όταν ο χρήστης πατά πως ξέχασε τον κωδικό του, θα διαπιστώσει πως συγκρατούν όλους τους κωδικούς πρόσβασης που έχει χρησιμοποιήσει στο παρελθόν, καθώς ζητούν να πληκτρολογήσει τον τελευταίο κωδικό που θυμάται. Σε επίπεδο βάσης, είναι πιθανό να έχουν έναν πίνακα *passwordHistory* που αποθηκεύει primary key emails και passwords, οπότε αν ο χρήστης εισάγει έναν κωδικό που θα διαπιστωθεί ότι χρησιμοποιούσε κάποτε, θα είναι ένδειξη ότι δεν είναι κάποιος τρίτος που προσπαθεί να παραβιάσει ξένο λογαριασμό.

Όταν σε οποιαδήποτε εφαρμογή πατήσουμε πως ξεχάσαμε τον κωδικό μας, δεν θα μας στείλει κάποιο SMS ή email που θα γράφει: ο κωδικός σας είναι αυτός. Πέρα από επισφαλές, είναι και αδύνατο να κάνει κάτι τέτοιο, καθώς ούτε η ίδια εφαρμογή γνωρίζει ποιος ακριβώς είναι ο κωδικός μας, εξάλλου κάθε φορά που προσπαθούμε να συνδεθούμε, μετατρέπει σε hash τον κωδικό που πληκτρολογήσαμε και τον συγκρίνει με το hash που έχει αποθηκεύσει στη βάση.

Αυτή είναι και μία βασική διαφορά μεταξύ της κρυπτογράφησης και του hashing, σε συνέχεια όσων ειπώθηκαν παραπάνω, μία συνάρτηση που πραγματοποιεί hashing είναι σχεδιασμένη ώστε να είναι πολύ δύσκολο να αντιστραφεί, αν όχι αδύνατο, πράγμα που σημαίνει πως ούτε ο διαχειριστής της βάσης έχει τη δυνατότητα να μετατρέψει τον hash κωδικό στον κανονικό, για να συνδεθεί στη θέση κάποιου χρήστη.

Άλλη σημαντική αναβάθμιση ασφαλείας θα ήταν ο server να αποθηκεύει, προσωρινά τουλάχιστον, τις απόπειρες σύνδεσης ανά χρήστη και αν διαπιστώνει ύποπτη συμπεριφορά, για παράδειγμα, πέντε αποτυχημένα login σε διάστημα είκοσι δευτερολέπτων, να εφαρμόζει ολοένα αυξανόμενα timeouts στις απαντήσεις του στον client, να του εμφανίζει προειδοποιήσεις και στο τέλος να στέλνει email στον χρήστη απαιτώντας την επαναφορά του κωδικού του. Εναλλακτικά, ο server θα μπορούσε να στέλνει μετά από κάθε σύνδεση ένα email που να ενημερώνει τον χρήστη ότι συνδέθηκε σε νέα συσκευή, για να μπορεί να διασταυρώσει αν ήταν αυτός που έκανε τη σύνδεση ή κάποιος υπέκλεψε τον λογαριασμό του.

Ένα άλλο χαρακτηριστικό που θα μπορούσε να προστεθεί στο θέμα του login, θα ήταν login with Gmail, login with Meta (Facebook) account, ή κάποιου είδους two-factor authentication που σε κάθε προσπάθεια σύνδεσης θα έστελνε εξαψήφιο one time password στον χρήστη μέσω SMS ή email για να επιβεβαιώσει την ταυτότητά του. Αυτό έγινε προσπάθεια να λειτουργήσει με χρήση εξωτερικής βιβλιοθήκης, η οποία δεν ρυθμίστηκε σωστά λόγω της σειράς κλήσης των middleware functions, όπως αυτή ορίζεται στο configuration του αντικειμένου express.

## 7.2 Debug decorator

Στο κομμάτι των middleware functions, οι συναρτήσεις των κλάσεων ROUTE και API κατά τη συγγραφή τους κατακλύστηκαν από γραμμές console.log(). Υπήρξε η σκέψη αυτό να γίνεται πιο κομψά, με ένα class decorator που θα έκανε loop για κάθε method της κλάσης και να της προσέθετε ένα απλό debug decorator που θα τύπωνε το όνομά της και τις παραμέτρους που πέρασαν στην κλήση της. Η ECMAScript δεν περιλαμβάνει σύνταξη με at (@) για την προσθήκη decorator, οπότε κάτι τέτοιο δεν μπόρεσε να γίνει.

Μια άλλη ιδέα θα ήταν να υπάρχει ένα custom middleware που στο configuration του express, θα δηλωνόταν ότι πρέπει να καλείται για κάθε εισερχόμενο request και να τυπώνει λόγου χάριν, το body του, όταν υπάρχει. Δυστυχώς, η express καλούσε το middleware αυτό σε κάθε request, συμπεριλαμβανομένων των GET requests για τα stylesheets ή τις εικόνες, οπότε δεν ήταν πρακτικό.

## 7.3 Πρόσθετα χαρακτηριστικά

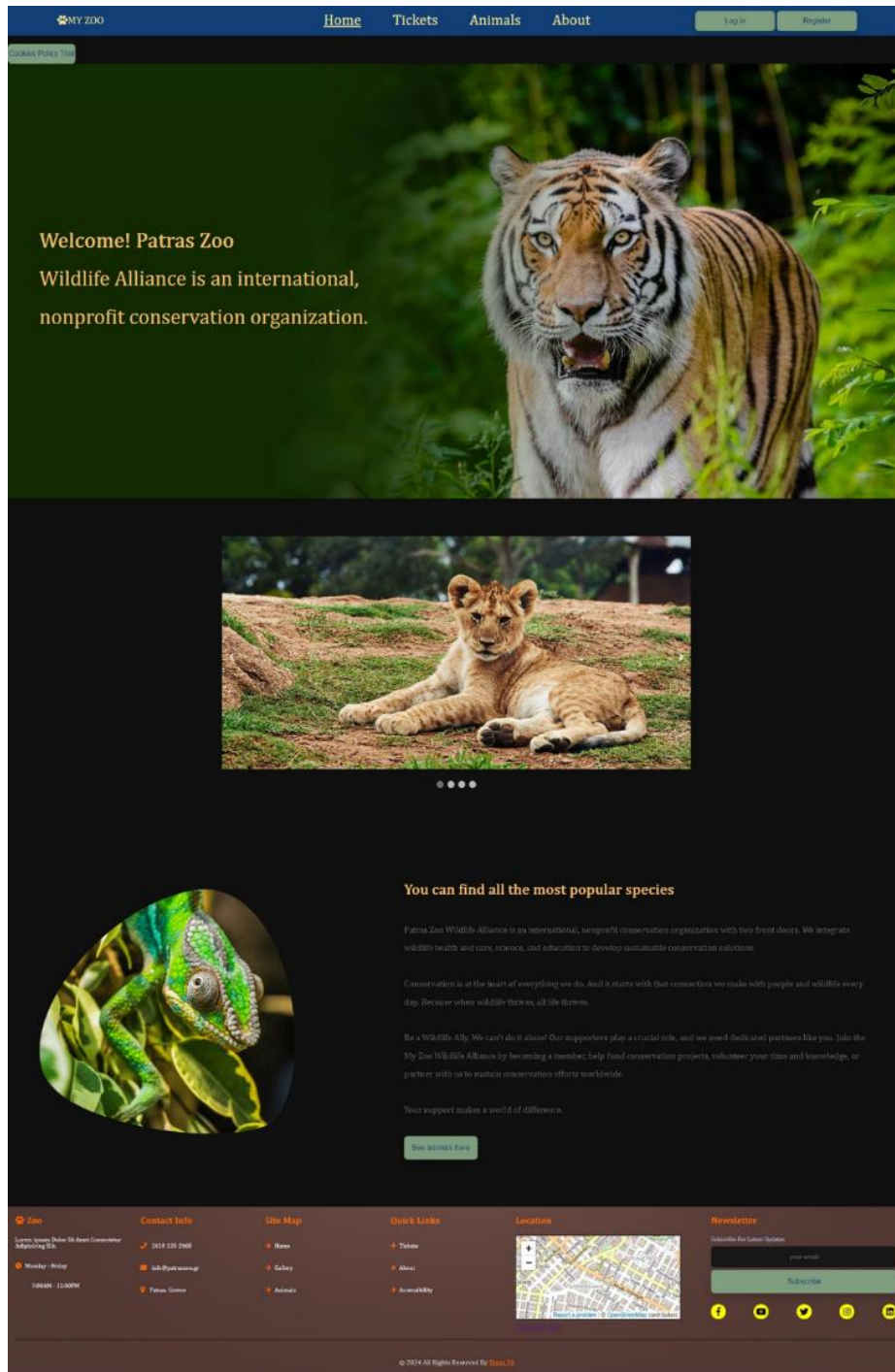
Υπό διαφορετικές συνθήκες, με καλύτερη διαχείριση χρόνου ή μεγαλύτερη προθεσμία ολοκλήρωσης της εργασίας, θα είχαν γίνει μετατροπές ώστε να υπάρχει περισσότερη συνοχή στον κώδικα της εφαρμογής, για παράδειγμα θα μεταγράφονταν κάποιες συναρτήσεις ώστε να μην επιστρέφουν, αλλά να περνούν σε callback function το αποτέλεσμα τους και το πιθανό σφάλμα. Επίσης, θα είχαν μεταφερθεί λειτουργίες και έλεγχοι που κάνουν οι middleware συναρτήσεις της ROUTE ή της API σε αντίστοιχες του controller, για να διαχωριστεί καλύτερα το τι ρόλο επιτελεί κάθε τμήμα του server, δηλαδή θα είχε τηρηθεί περισσότερο η αρχιτεκτονική Model – View – Controller.

Στο θέμα των Handlebars, η σωστή χρήση τους προβλέπει την ύπαρξη στοιχείων HTML σε τρία επίπεδα, layouts, partials και views. Γράφτηκε ένα layout το οποίο χρησιμοποιούν όλα τα views για να έχουν κοινά header, footer και dialogs, όμως η ενδεικνυόμενη προσέγγιση θα ήταν το header, το footer ή τα dialogs να είναι ορισμένα σαν partials. Σε αυτή την εφαρμογή, κάτι τέτοιο θα άφηνε το layout κενό, οπότε προτιμήθηκε να οριστούν στο layout.

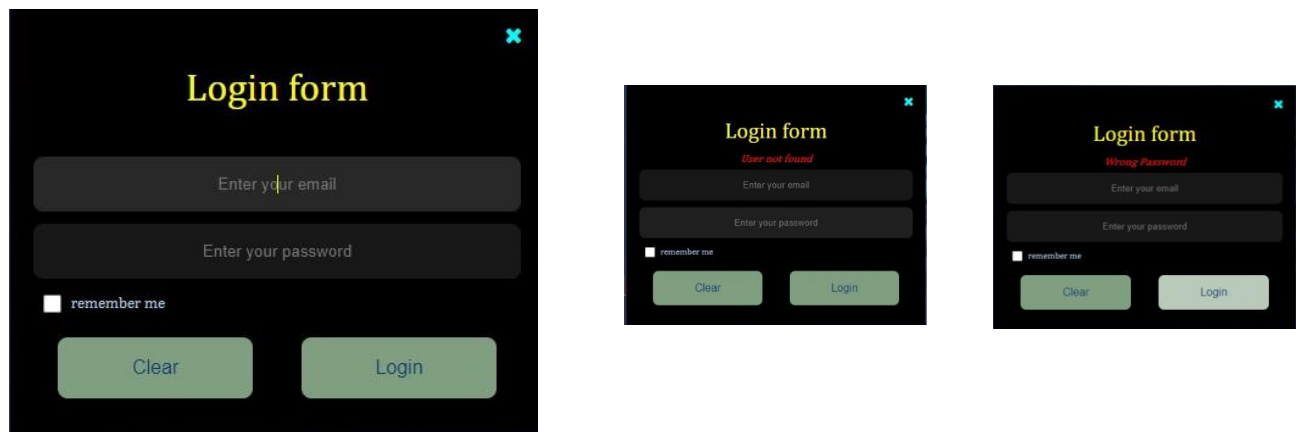
Πάλι στο κομμάτι του front end, οι συναρτήσεις JavaScript γράφτηκαν σε script tags μέσα στα αρχεία, για να είναι εύκολα προσβάσιμες, ενώ σωστή πρακτική θα ήταν να μεταφέρονταν σε ένα δικό τους αρχείο, που θα οριζόταν στην αρχή της HTML.

Αν υπήρχε πολυτέλεια χρόνου, η εφαρμογή θα είχε παραπάνω χαρακτηριστικά, λειτουργία αλλαγής γλώσσας, light και dark theme για τις σελίδες, ένα λειτουργικό administration dashboard, δυνατότητα σύνδεσης για διαφορετικών ειδών εργαζόμενους του κήπου, διαχείριση των εργασιών που θα είχαν να κάνουν κάθε μέρα, προγραμματισμός επισκέψεων και έλεγχος διαθεσιμότητας εισιτηρίων για δεδομένη μέρα για να αποτρέπεται ο συνωστισμός, διαφορετική λειτουργία της σελίδας animals ώστε να διαβάζει κείμενα από τη Wikipedia μέσω API, λειτουργικό newsletter και πραγματική αποστολή emails από noreply διεύθυνση του κήπου, προσθήκη QR code στις αποδείξεις, δημιουργία χάρτη για τις εγκαταστάσεις του κήπου, ορισμός της τοποθεσίας των ζώων πάνω στον χάρτη και μέθοδος υπολογισμού των αποστάσεων μεταξύ τους, ομαδοποίησή τους σε ζώνες ή τοποθεσίες και διαφορετικού τύπου εισιτήρια για την κάθε ζώνη, ενδεχομένως με διαφορετική τιμή, ανάλογα με τη σπανιότητα ή τη ζήτηση του καθενός, πειραματισμός με άλλες τεχνολογίες, όπως η JQuery, βιβλιοθήκες για το back end, κρυπτογράφηση, bootstrap, mongoDB για βάση δεδομένων, docker container, επιλογές για web hosting και λειτουργία σε domain.

## 7.4 Στιγμιότυπα

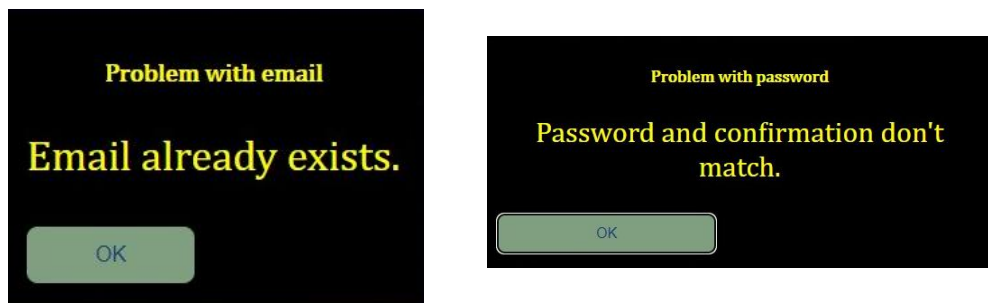


Εικόνα 2: Αρχική Σελίδα.



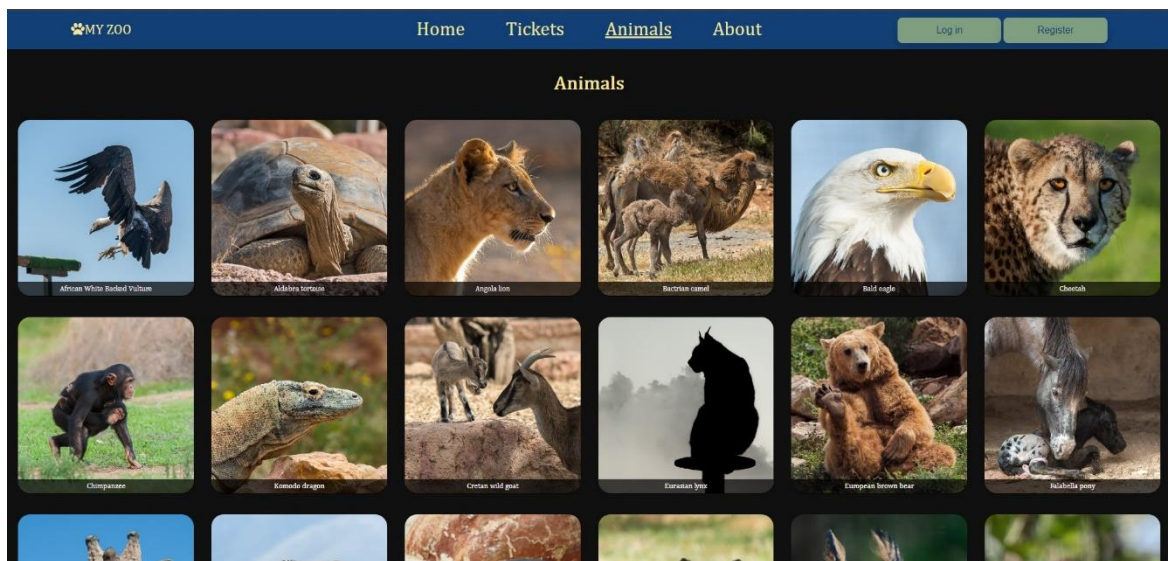
Εικόνα 3: Login Form popups and messages.

The image shows a "New user registration" form on a website with a dark blue header. The header includes the "MY ZOO" logo, navigation links for "Home", "Tickets", "Animals", and "About", and "Log in" and "Register" buttons. The registration form itself is a light gray box with the following fields: "First name", "Last name", "Address", "Street and Number", "City", "Postal Code", "Birthdate" (with a date picker showing 01/01/1905), "Contact", "Telephone", "Email", "Password", and "Confirm Password". There are "Clear" and "Register" buttons at the bottom of the form.

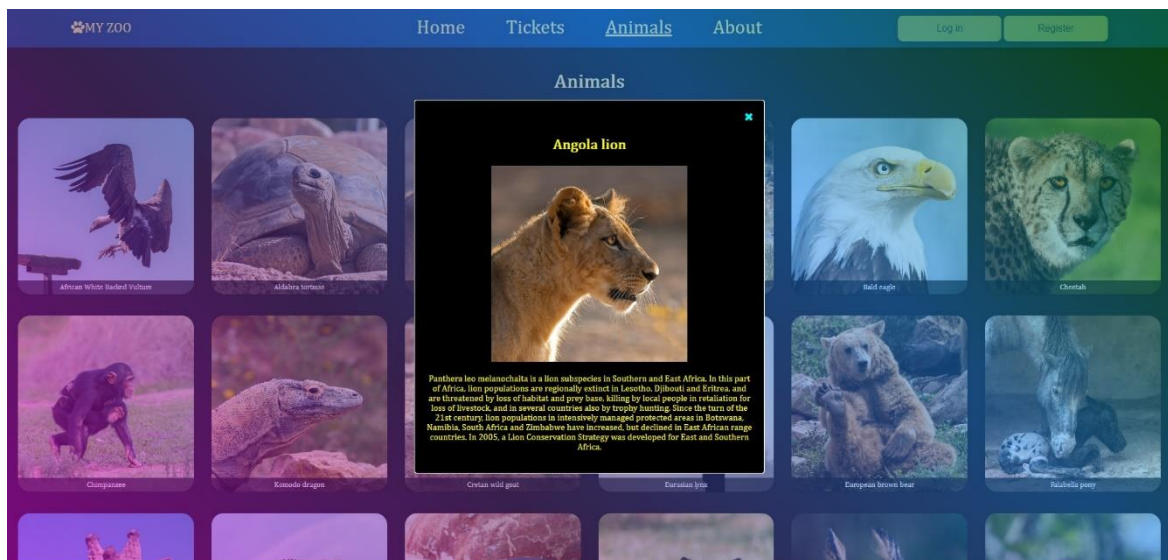


Εικόνα 4: Register Form popups and messages.

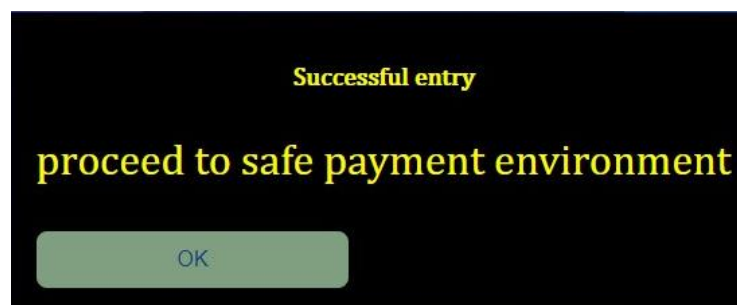
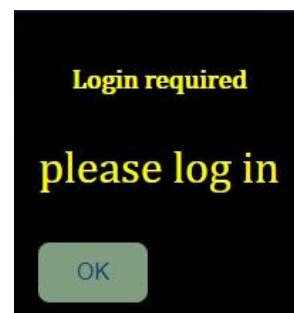
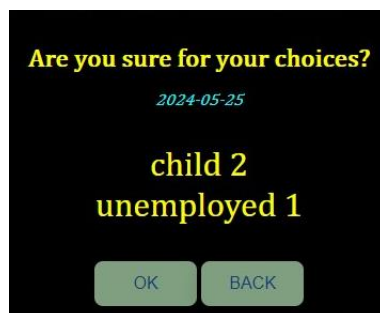
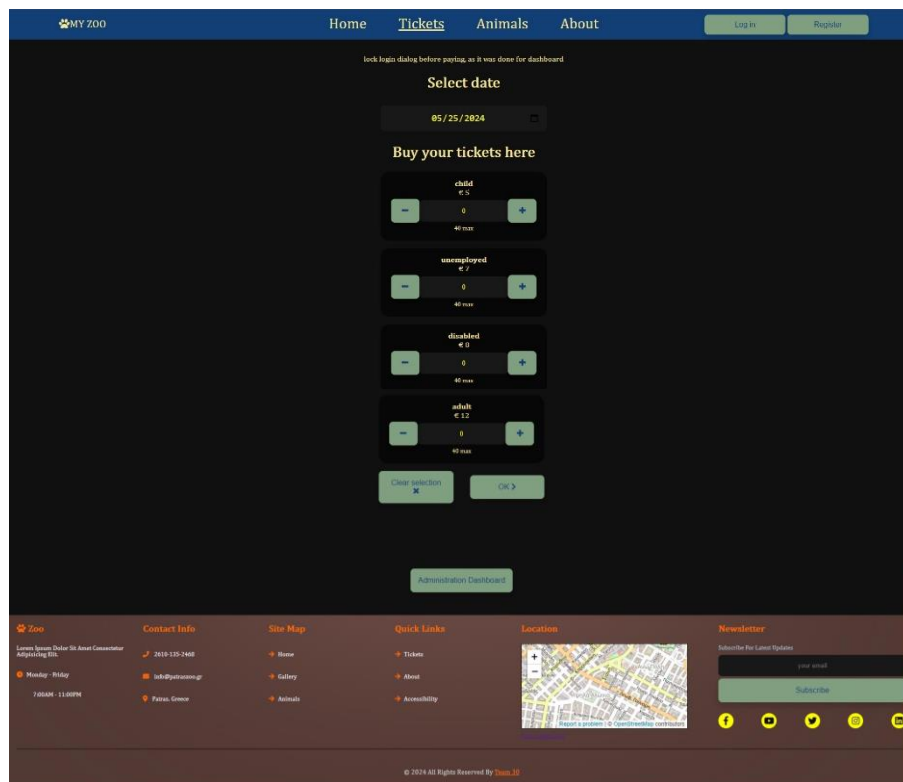




Εικόνα 5: Σελίδα Animals.



Εικόνα 6: Σελίδα Animals popup dialog.



Εικόνα 7: Σελίδα Tickets και popup messages.

MY ZOO

HomeTicketsAnimalsAbout

Log out

Payment Information

card number

0123 4567 8910 1112

cardholder name

YOUR NAME

expiration date

01/23

Name

Card Number

Expiration (mm/yy)

Security Code

Abort

Clear

Proceed

Administration Dashboard

Zoo

Learn, Inspire, Enjoy! Sit Asset Consortium  
Adopting EUs

Monday - Friday

7:00AM - 11:00PM

Contact Info

2610-135-2468

info@patraszoo.gr

Patras, Greece

Site Map

Home

Gallery

Animals


Quick Links

Tickets

About

Accessibility

Location



Report a problem | © OpenStreetMap contributors

Newsletter

Subscribe For Latest Updates

your email

Subscribe

f

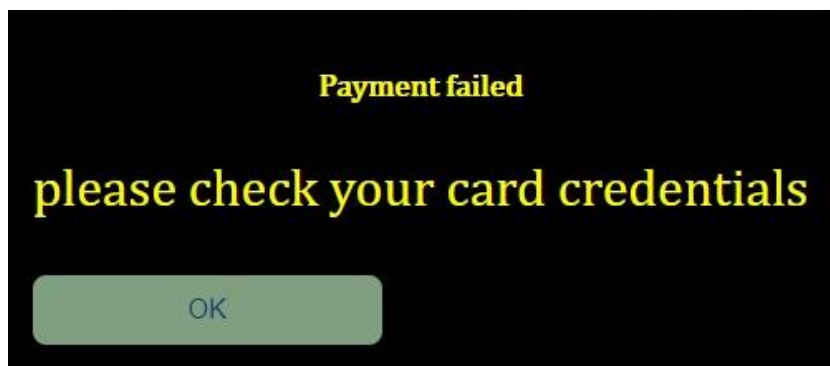
yt

tw

ig

wh

© 2024 All Rights Reserved By Team 30

A black rectangular screen with yellow text. At the top, it says "Payment failed". Below that, in a larger font, it says "please check your card credentials". At the bottom, there is a green button with the text "OK" in black.

Εικόνα 8: Σελίδα payment και message.

19

## 8 ΕΓΚΑΤΑΣΤΑΣΗ

Η εφαρμογή είναι διαθέσιμη σε GitHub repository, οι απαιτούμενες βιβλιοθήκες μπορούν να εγκατασταθούν με την εντολή `npm install`. Δεν χρησιμοποιούνται τιμές από αρχείο `.env`, έχει οριστεί το script `start` για την εκκίνηση του server.

### 8.1 Αποθετήριο

Το δημόσιο αποθετήριο του κώδικα της εφαρμογής μπορεί να βρεθεί στο [GitHub](#).

## 9 ΒΙΒΛΙΟΓΡΑΦΙΑ

- [1] HTML – CSS – JavaScript Documentation. Mozilla Developer Network. Retrieved Spring 2024 from <https://developer.mozilla.org/en-US/docs/Web>
- [2] OpenStreetMap. Retrieved May 25, 2024, from <https://www.openstreetmap.org/>
- [3] Node.js v20 Documentation. Retrieved May 25, 2024, from <https://nodejs.org/docs/latest-v20.x/api/index.html>
- [4] SQLite Documentation. Retrieved April 2024, from <https://www.sqlite.org/docs.html>
- [5] NPM. better-sqlite3 Documentation. Retrieved May 24, 2024, from <https://www.npmjs.com/package/better-sqlite3>
- [6] Express Documentation. Retrieved May 25, 2024, from <https://expressjs.com/en/4x/api.html>
- [7] Attica Park. Retrieved April 2024, from <https://www.atticapark.com/?lang=en>
- [8] PDFKit Documentation. Retrieved May 25, 2024, from <https://pdfkit.org/>
- [9] Handlebars. Handlebars Guide. Retrieved April 2024, from <https://handlebarsjs.com/guide/>
- [10] Adam Quinlan. 2024. CodePen profile. Retrieved May 20, 2024, from <https://codepen.io/quinlo>
- [11] Bcrypt password-hashing function. Retrieved May 26, 2024, from <https://en.wikipedia.org/wiki/Bcrypt>
- [12] Wikipedia: The Free Encyclopedia. Retrieved April 2024, from <https://www.wikipedia.org/>
- [13] ERD Maker. 2021. Website: <https://hci.ece.upatras.gr/erdmaker/>
- [14] ECMAScript® 2015 Language Specification. Retrieved May 26, 2024, from <https://262.ecma-international.org/6.0/>
- [15] NPM. cookie-parser Documentation. Retrieved May 26, 2024, from <https://www.npmjs.com/package/cookie-parser>
- [16] NPM. express-session Documentation. Retrieved May 26, 2024, from <https://www.npmjs.com/package/express-session>
- [17] NPM. fs Documentation. Retrieved May 26, 2024, from <https://www.npmjs.com/package/fs>
- [18] Wikipedia. MVC design pattern. Retrieved May 26, 2024, from <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
- [19] NPM. bcrypt Documentation. Retrieved May 26, 2024, from <https://www.npmjs.com/package/bcrypt>
- [20] Wikipedia. List of HTTP status codes. Retrieved May 26, 2024, from [https://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_status\\_codes](https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)