ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ

ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

##### ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Συνοδευτικό πτυχιακής εργασίας

**Βάρδιες εργαζομένων σε roster**

**Έκτορας Σιμιστήρας**

**2011187**

**Επιβλέπων**

**Βασίλης Πουλόπουλος**

< Ιδιότητα >

**Σπάρτη, Σεπτέμβρης 2021**

Contents

[Ευχαριστίες 3](#_Toc82516980)

[Εισαγωγή 4](#_Toc82516981)

[Περίληψη 4](#_Toc82516982)

[Εκκίνηση 4](#_Toc82516983)

[Τεχνολογίες 4](#_Toc82516984)

[Εισαγωγή 4](#_Toc82516985)

[Αρχιτεκτονική (MVVM, OOP, Knockout, Webpack) 5](#_Toc82516986)

[Client, Service, Server 7](#_Toc82516987)

[Εισαγωγή 7](#_Toc82516988)

[Client Side 7](#_Toc82516989)

[Service 8](#_Toc82516990)

[Server 8](#_Toc82516991)

[Εφαρμογή 10](#_Toc82516992)

[Δομική Ανάλυση 10](#_Toc82516993)

[Ιστοσελίδα 12](#_Toc82516994)

[Login 13](#_Toc82516995)

[Σχεδίαση 14](#_Toc82516996)

[Διαδικασία & Λειτουργικότητα 14](#_Toc82516997)

[Dashboard 16](#_Toc82516998)

[Σχεδίαση 16](#_Toc82516999)

[Δεδομένα 18](#_Toc82517000)

[Επεξεργασία Δεδομένων 19](#_Toc82517001)

[Παρουσίαση Δεδομένων 21](#_Toc82517002)

[Λήψη Δεδομένων 22](#_Toc82517003)

[Αλλαγές Προγράμματος 24](#_Toc82517004)

[Εκτύπωση Προγράμματος 26](#_Toc82517005)

[Διαφημιστική ιστοσελίδα λειτουργιών 28](#_Toc82517006)

[Εισαγωγή 28](#_Toc82517007)

[Σχεδίαση 28](#_Toc82517008)

[Συστήματα Roster 33](#_Toc82517009)

[Εισαγωγή 33](#_Toc82517010)

[Περιγραφή 34](#_Toc82517011)

[Υποστηριζόμενοι Τύποι εργαζομένων. 34](#_Toc82517012)

[Λειτουργικοί Παράγοντες 34](#_Toc82517013)

[Περίοδος λειτουργίας 34](#_Toc82517014)

[Φόρτος Εργασίας 35](#_Toc82517015)

[Δεξιότητες εργαζομένων 35](#_Toc82517016)

[Ροές εργαζομένων 35](#_Toc82517017)

[Ανάλυση Συστημάτων 36](#_Toc82517018)

[Humanity 36](#_Toc82517019)

[Kronos 37](#_Toc82517020)

[Connecteam 38](#_Toc82517021)

# Ευχαριστίες

Θα ήθελα να ευχαριστώ θερμά τον συνάδελφο που ανέπτυξε και ανέβασε το service που χρησιμοποιήθηκε στο project δίνοντας μου την ευκαιρία να δημιουργήσω ένα όσο το δυνατόν περισσότερο ρεαλιστικό client για την εφαρμογή.

Θα ήθελα επίσης να ευχαριστήσω τον κ. Βασίλη Πουλόπουλο που δέχτηκε να αναλάβει την πτυχιακή μου εργασία καθώς και να με καθοδηγήσει κατά την διάρκεια ανάπτυξης της.

# Εισαγωγή

## Περίληψη

Η Πτυχιακή εργασία με τίτλο «Βάρδιες Εργαζομένων Σε Roster» είναι ένα Mobile First, SPA (Single Page Application) που χρησιμοποιείται για την **προβολή** και **διαχείριση** του προγράμματος εργασίας πολλαπλών χρηστών από οποιαδήποτε συσκευή (Smartphone, Tablet, Laptop, Desktop).

Οι χρήστες επισκέπτονται την ιστοσελίδα της επιχείρησης που εργάζονται **(e-port)** και από εκεί αιτούνται να συνδεθούν. Αφού η ταυτότητα τους επιβεβαιωθεί από το σύστημα αποκτούν πρόσβαση στο portal που τους επιτρέπει να προβάλλουν, εκτυπώσουν , παραμετροποιήσουν το πρόγραμμα τους ή να επισκεφτούν τον εξωτερικό σύνδεσμο που τους οδηγεί σε ένα πλήρες διαφημιστικό documentation που αφορά τις λειτουργίες ενός συστήματος roster.

Σκοπός της εργασίας επίσης είναι να αξιοποιήσουμε τις πλέον σύγχρονες τεχνολογίες προγραμματισμού παράλληλα με μια ορθή τεχνική ανάπτυξης που θα ήταν κατάλληλη για την δόμηση οποιουδήποτε project, ανεξαρτήτως μεγέθους.

# Τεχνολογίες

## Εισαγωγή

Για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε η αρχιτεκτονική προγραμματισμού **MV-VM** (Model View, View Model), Οι γλώσσες προγραμματισμού HTML, Javascript, CSS / SASS-CSS, C#, SQL τα frameworks Bootstrap, Knockout.js καθώς και η βιβλιοθήκη JQUERY.

Τέλος για την υποστήριξη της αρχιτεκτονικής που επιλέχθηκε ήταν απαραίτητη η χρήση του WEBPACK για την υποστήριξη των imports (Javascript & HTML), την σωστή ιεραρχία των dependencies και το «χτίσιμο» των output αρχείων της Javascript που τελικά συμπεριλαμβάνουμε στην HTML.

Για την υποστήριξη κάποιων λειτουργιών του Webpack καθώς και για την υλοποίηση συγκεκριμένων scripts που βοήθησαν στην διαδικασία ανάπτυξης χρησιμοποιήθηκε το NPM.

## Αρχιτεκτονική (MVVM, OOP, Knockout, Webpack)

Για την ανάπτυξη του συγκεκριμένου project αξιοποιήσαμε την τεχνική του Αντικειμενοστραφή Προγραμματισμού που βασίζεται στις κλάσεις και τα αντικείμενα (παρότι δεν αξιοποιήσαμε πλήρως την «κληρονομικότητα»).

Η αρχιτεκτονική MVVM χωρίζει ουσιαστικά το κομμάτι της λειτουργικότητας (MV) και το κομμάτι της παρουσίασης (VIEW) και μας επιτρέπει να επαναχρησιμοποιούμε και να επεξεργαζόμαστε τον κώδικα μας εύκολα κρατώντας το project καθαρό και διαχειρίσιμο

Αναλυτικότερα,

Με την βοήθεια της Knockout.js και την λειτουργία των imports για Javascript & HTML αρχεία μπορούμε να δημιουργήσουμε αντικείμενα που αποτελούνται από κλάσεις «components»

import ko from "knockout"

import template from "./view.html"

class VM{

    constructor(variable){

        //Some JS or ko variables

    }

  //Some JS functions

}

ko.components.register("example1",{

    viewModel:VM,

    template:template

});

Με αυτή την τεχνική εξασφαλίζουμε πως τα components είναι τελείως αυτόνομα όσον αφορά την λειτουργικότητα τους και μπορούν να αποθηκεύονται μόνα τους καθώς και να επαναχρησιμοποιούνται στον κώδικα όπου επιλέξουμε χρησιμοποιώντας το “data-bind” attribute που δεσμεύει η Knockout στην HTML

<div data-bind="component:{name:'example1'}"></div>

Με την βοήθεια του webpack όλα τα components θα γίνουν import στο javascript αρχείο που θα χρησιμοποιηθεί σαν **“entry point”** για την εφαρμογή και μέσω της συνάρτησης

ko.applyBindings(component);

και των **ko.observable** μεταβλητών η knockout αναλαμβάνει να παρακολουθεί δυναμικά τις αλλαγές στα δεδομένα που προβάλλονται.

Natively, η Javascript δεν μπορεί να υποστηρίξει την τεχνική των imports που απαιτείται για την δημιουργία των components, εκεί λοιπόν αναλαμβάνει το webpack και οι κατάλληλοι loaders που υποστηρίζουν την μέθοδο των imports για τα Javascript και HTML αρχεία και μας επιτρέπουν να κάνουμε bundle βιβλιοθήκες κατευθείαν από τον κώδικα μας κρατώντας σταθερό version χωρίς την βοήθεια του npm ή παραθέτοντας CDN links στην HTML καθώς και να είμαστε σίγουροι ότι διατηρείται η σωστή ιεραρχία στα dependencies του προγράμματος και τέλος να κάνουμε minimize τον κώδικα όταν ολοκληρώσουμε το project.

Ως “entry point” έχουμε ορίσει την **main.js**

Έτσι λοιπόν το webpack βλέπει όλα τα αρχεία που έχουν γίνει import και χτίζει το output εκεί που το έχουμε ορίσει στο configuration του Webpack

    entry: {

        thesis: './app/main.js'

    },

    output: {

        path: path.resolve(\_\_dirname, './dist/\_js'),

        filename: '[name].js'

    }

Για την υποστήριξη των HTML imports έχουμε φορτώσει τον κατάλληλο loader

  module: {

        rules: [{

            test: /\.html$/,

            use: [{

                loader: 'html-loader',

                options: {

                    minimize: false

                }

            }]

Ενώ έχουμε ξεχωρίσει τον κώδικα σε 2 chunks,

Ένα για τον δικό μας κώδικα που θα γίνει bundle στο **thesis.js** όπως είδαμε παραπάνω

Και ένα δεύτερο για τις βιβλιοθήκες της JQUERY και KNOCKOUT που θα γίνουν bundle σε ένα δεύτερο αρχείο, **vendors.js**

            cacheGroups: {

                commons: {

                    test: /[\\/]\_libs\\(jquery|knockout)[\\/]/,

                    name: 'vendors',

                }

            }

## Client, Service, Server

### Εισαγωγή

Κάθε εφαρμογή που αναλαμβάνει να διαχειριστεί η να προβάλει δεδομένα αποτελείται από τρία βασικά μέρη.

Τον **Client** , που παρουσιάζει δεδομένα,

Τον **Server**, που αποθηκεύει δεδομένα,

Το **Service**, που καλείται από τον Client και αναλαμβάνει να φέρει ή/και να στείλει δεδομένα στον Server.

### Client Side

Χρησιμοποιώντας τις βασικές γλώσσες προγραμματισμού Javascript, HTML και CSS συνθέτουμε το client κομμάτι του προγράμματος που είναι υπεύθυνο για την επεξεργασία και παρουσίαση των δεδομένων. H HTML5 είναι η γλώσσα που αναλαμβάνει την οπτικοποίηση των δεδομένων, η CSS αναλαμβάνει να βελτιώσει την εμπειρία του χρήστη όχι μόνο ενισχύοντας το παρουσιαστικό αλλά εξασφαλίζοντας ότι η εφαρμογή θα προσαρμόζεται στο μέγεθος της οθόνης του χρήστη (Responsiveness) κάνοντας έτσι εφικτή την χρήση του προγράμματος από οποιαδήποτε συσκευή έχει στην διάθεση του ο τελικός χρήστης ενώ τελικά η Javascript είναι η γλώσσα που αναλαμβάνει την επεξεργασία των δεδομένων και την ανταπόκριση των λειτουργιών του προγράμματος σε πραγματικό χρόνο.  
Επιπρόσθετα έχουν χρησιμοποιηθεί δύο frameworks ,

**Α)** Για την CSS, η **Bootstrap**.

H Bootstrap είναι το πιο διαδεδομένο CSS framework για την εύκολη και γρήγορη ανάπτυξη αισθητικά όμορφων και responsive ιστοσελίδων, χρησιμοποιεί την CSS την Javascript και την PopperJS (για tooltips, popovers, dropdowns, modals) και δίνει την δυνατότητα στον χρήστη να «χτίζει» λειτουργικό, χρονοβόρο και πολύπλοκο κώδικα εισάγοντας κλάσεις κατευθείαν στην HTML.

**Β)** Για την Javascript , η **Knockout.js**

Αναφέραμε σε προηγούμενο κεφάλαιο το πως η Knockout μας παρέχει τους απαραίτητους μηχανισμούς για να χτίσουμε τα components που χρειαζόμαστε για το μοντέλο αρχιτεκτονικής που επιλέξαμε. Επιπρόσθετα η Knockout μας εξασφαλίζει ότι το περιβάλλον παρουσίασης θα παρακολουθείται διαρκώς και θα αντιδρά σε οποιαδήποτε αλλαγή συμβεί σε πραγματικό χρόνο μέσω των Observable μεταβλητών. Αυτό είναι κάτι που δεν υποστηρίζεται από την Native Javascript και κρίνεται απολύτως απαραίτητο για την ορθή ανάπτυξη της εφαρμογής μας. Για την καλύτερη λειτουργία της knockout συμπεριλάβαμε και την βιβλιοθήκη της **JQUERY.** Με την JQUERY έχουμε την δυνατότητα να δημιουργούμε pointers τα οποία δένουμε με τα instances των classes όταν επαναχρησιμοποιούμε components εξασφαλίζοντας έτσι ότι το σύστημα πάντα θα φέρνει τα σωστά δεδομένα στην παρουσίαση.

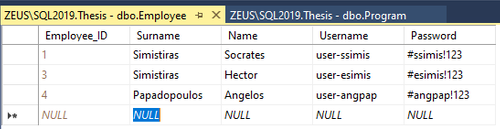
### Service

### Server

Το κομμάτι του server αποτελείται μια SQL Βάση Δεδομένων.

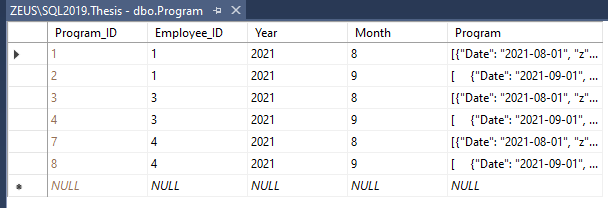
H Βάση περιέχει δύο (2) πίνακες

Α)



Ένας πίνακας για τους εργαζόμενους με τα στοιχεία πρόσβασης τους στην εφαρμογή καθώς και ένα Employee\_ID που είναι και το κύριο κλειδί. Χρησιμεύει στο να ταυτοποιηθεί ο χρήστης κατά την διάρκεια του Login.

Β)



Ένας πίνακας που αποθηκεύει το πρόγραμμα εργασίας για τους χρήστες και έχει ως ξένο κλειδί το Employee\_ID που εξυπηρετεί στην αντιστοίχιση του προγράμματος εργασίας με τον αντίστοιχο χρήστη, έτσι όταν ζητήσουμε το πρόγραμμα εργασίας για έναν χρήστη θα στείλουμε σαν παραμέτρους το ID του χρήστη και τον μήνα στον οποίο αναφερόμαστε για να μας επιστραφεί το πρόγραμμα που ζητάμε.

Τέλος υπάρχει ο κατάλληλος κώδικας για να αλλάξει τις εγγραφές στην βάση όπως αιτήθηκαν από τον εργαζόμενο όταν αυτό ζητηθεί από το service.

Graphical user interface, text

Description automatically generated

# Εφαρμογή

## Δομική Ανάλυση

Όπως αναφέρθηκε στην Εισαγωγή, η εφαρμογή αναπτύχθηκε ως SPA (Single Page Application), ένας τρόπος ανάπτυξης εφαρμογών σε μοναδικό HTML αρχείο που δεν απαιτεί redirect όταν ο χρήστης επιλέγει να αλλάξει σελίδα. Όπως αναλύσαμε σε προηγούμενο κεφάλαιο αυτή η τεχνική απαιτεί την χρήση των components που δημιουργήσαμε με την βοήθεια της Knockout.js και του Webpack.

H **index.html** περιέχει 2 μοναδικά Divisions:

1. Το **“Page Host”**

<div id="PageHost" style="position:absolute; left:0; top:0; right:0; bottom:0; " data-bind="component:{name:'page-host'}"></div>

1. To **“Printing Host”**

<div id="printingHost" style="padding:0px; margin:0px !important; outline:none;">  </div

Το Page Host είναι το division το οποίο θα φιλοξενήσει όλα τα components της εφαρμογής μας, ενώ με το Printing Host θα ασχοληθούμε σε επόμενο κεφάλαιο.

Παρατηρούμε πως το Division είναι τοποθετημένο έτσι ώστε να καλύπτει το 100% της οθόνης ανεξαρτήτους μεγέθους και περιέχει το “Data-bind” attribute το οποίο φιλοξενεί ένα component που αρχικοποιεί την σελίδα που προβάλλεται στον χρήστη. Κάθε component που περνάμε στο division αυτό αντιστοιχεί σε μία καινούργια σελίδα (e-port Site, Login, Dashboard). Για μπορούμε να έχουμε πρόσβαση στο root page από όλα τα components έτσι ώστε να επιτρέπουμε στον χρήστη να πλοηγηθεί στο σύστημα έχουμε στήσει έναν μηχανισμό στην main.js η οποία παίρνει σαν imports όλα τα components του προγράμματος άρα έχει πάντα πρόσβαση σε αυτά.

class App{

    constructor(){

        this.currentPage = ko.observable();

           }

    \_selectPage(name){

        this.currentPage(name);

    }

}

Έτσι από όλα τα components μπορούμε να καλέσουμε την \_selectPage χρησιμοποιώντας το component name που θέλουμε και να αλλάξουμε το περιεχόμενο προβολής χωρίς να χρειαστεί να κάνουμε redirect τον χρήστη σε διαφορετικό HTML αρχείο.

Την ίδια ακριβώς τεχνική έχουμε χρησιμοποιήσει και για το Dashboard το οποίο σαν root page περιέχει ένα menu όπου κάθε εγγραφή του είναι «δεμένη» με ένα component για να αλλάζει το περιεχόμενο σε ένα εσωτερικό division.

## Ιστοσελίδα

Η αρχική ιστοσελίδα που οδηγείται ο χρήστης πριν ξεκινήσει την διαδικασία πιστοποίησης του.

Όπως αναφέρθηκε, η εφαρμογή και όλα τα components της είναι “fully responsive”, δηλαδή με τον κατάλληλο CSS κώδικα εξασφαλίζουμε πως ανταποκρίνονται σε οποιοδήποτε μέγεθος συσκευής. Αρχικά σχεδιάζουμε το component για μικρές συσκευές (smartphone) και στην συνέχεια με την βοήθεια του

@media only screen and (min-width: /\*width in pixels\*/) { /\*some code\*/ }

που ενεργοποιεί τον CSS κώδικα όταν το μέγεθος της οθόνης του χρήστη είναι μέσα στο εύρος των pixel που ορίσαμε διαφοροποιούμε την σχεδίαση έτσι ώστε να είναι συμβατό και με μεγαλύτερες συσκευές.

Στην παρακάτω εικόνα μπορούμε να δούμε πως η ίδια σελίδα ανταποκρίνεται σε δύο διαφορετικές οθόνες.

|  |
| --- |
|  |
| **DESKTOP** |

|  |  |
| --- | --- |
|  | A close-up of a cell phone  Description automatically generated with medium confidence |

**MOTO G4**

Η ιστοσελίδα δεν παρέχει κάποια λειτουργικότητα στην εφαρμογή μας, εκτός από την μετάβαση στο «Login» μέσω του αντίστοιχου κουμπιού.

## Login

### Σχεδίαση

|  |  |
| --- | --- |
| **iPhone 8** | **Desktop** |
|  | Graphical user interface, application  Description automatically generated |

### Διαδικασία & Λειτουργικότητα

Ο χρήστης καλείται να συμπληρώσει μια φόρμα Login δίνοντας ένα όνομα χρήστη και έναν κωδικό. Πατώντας το Login η Javascript περνάει τα στοιχεία στο body ενός HTTP request με μέθοδο “POST” και τα στέλνει στο service με μια ασύγχρονη fetch συνάρτηση.

Στη συνέχεια αναμένεται η απάντηση του service το οποίο έχει συγκρίνει τα στοιχεία που δόθηκαν από τον χρήστη με τα αποθηκευμένα στοιχεία χρηστών στην βάση δεδομένων.  
Αν το promise.ok που επιστραφεί από το service είναι “true” σημαίνει πως τα δεδομένα που έδωσε ο χρήστης είναι έγκυρα και επιστρέφεται το ΙD του εργαζομένου ενώ σε διαφορετική περίπτωση «πιάνουμε» και ενημερώνουμε τον χρήστη για το error που προέκυψε.

    async \_login(un,ps) {

        let request = "https://thesis-api.azurewebsites.net/Thesis/Login"

        try {

            let options = {

                method: "POST",

                headers: {

                    "Content-Type": "application/json"

                },

                body: JSON.stringify({un,ps}) //username & password

            }

            let response = await fetch(request, options);

            if (response.ok) {  // Έλεγχος Promise

                let serviceData = await response.json();

                return serviceData; // Επιστροφή ID

            }

            else

                throw Error(response.statusText);

        } catch (e) {

            this.showLoader(false);

            alert(e.message);

        }

    }

Σε περίπτωση επιτυχίας αποθηκεύουμε το username, password και ΙD σε observable μεταβλητές σε ένα αρχείο (**Helper.js**) που παρακολουθείται από όποιο component επιλέξουμε μέσω import έτσι ώστε για κάθε χρήση του service να υπάρχει πρόσβαση στα στοιχεία που δόθηκαν κατά το login έτσι ώστε να μπορούμε να επιβεβαιώνουμε την εγκυρότητα του χρήστη μέσω του header στα επόμενα request μας όπως ορίζει η μέθοδος του Basic Authentication αλλά και να έχουμε διαθέσιμο το ID του συνδεδεμένου χρήστη που λειτουργεί σαν κλειδί στην βάση δεδομένων.

async \_onLogin(){

    this.showLoader(true);

    this.emp\_ID= await this.\_login(this.username(),this.password());

    if (this.emp\_ID!=null){

        config.user\_id=this.username(); //Απόθήκευση Username

        config.user\_pwd=this.password(); //Απόθήκευση Password

        config.employee\_id=this.emp\_ID; //Απόθήκευση ID

        this.navigateDashboard(); //Αλλαγή component

    }

}

Τέλος αλλάζουμε το περιεχόμενο του **data-bind** που φιλοξενεί το component του login και μεταφερόμαστε στο Dashboard.

navigateDashboard(){

    this.showLoader(false);

    config.dash\_body();

    this.\_PageRoot.currentPage('host-dashboard');

}

Θα πρέπει να σημειωθεί πως η μέθοδος του basic authentication σε συνδυασμό με το ΗΤΤP πρωτόκολλο δεν παρέχει επαρκή ασφάλεια δεδομένων, τα δεδομένα μεταφέρονται στο σύρμα χωρίς κρυπτογράφηση και μπορούν να αποκαλυφθούν από οποιονδήποτε επιχειρήσει να «ακούσει» την πόρτα που χρησιμοποιεί το πρόγραμμα για να επικοινωνήσει με το Service.

## Dashboard

### Σχεδίαση

Το Dashboard είναι ένα component που περιέχει πολλές λειτουργίες και έτσι κρίνεται απαραίτητο να χρησιμοποιήσουμε την τεχνική του SPA εσωτερικά για να εναλλάσσουμε τα components που προβάλλονται ανάλογα με τις επιλογές του menu.

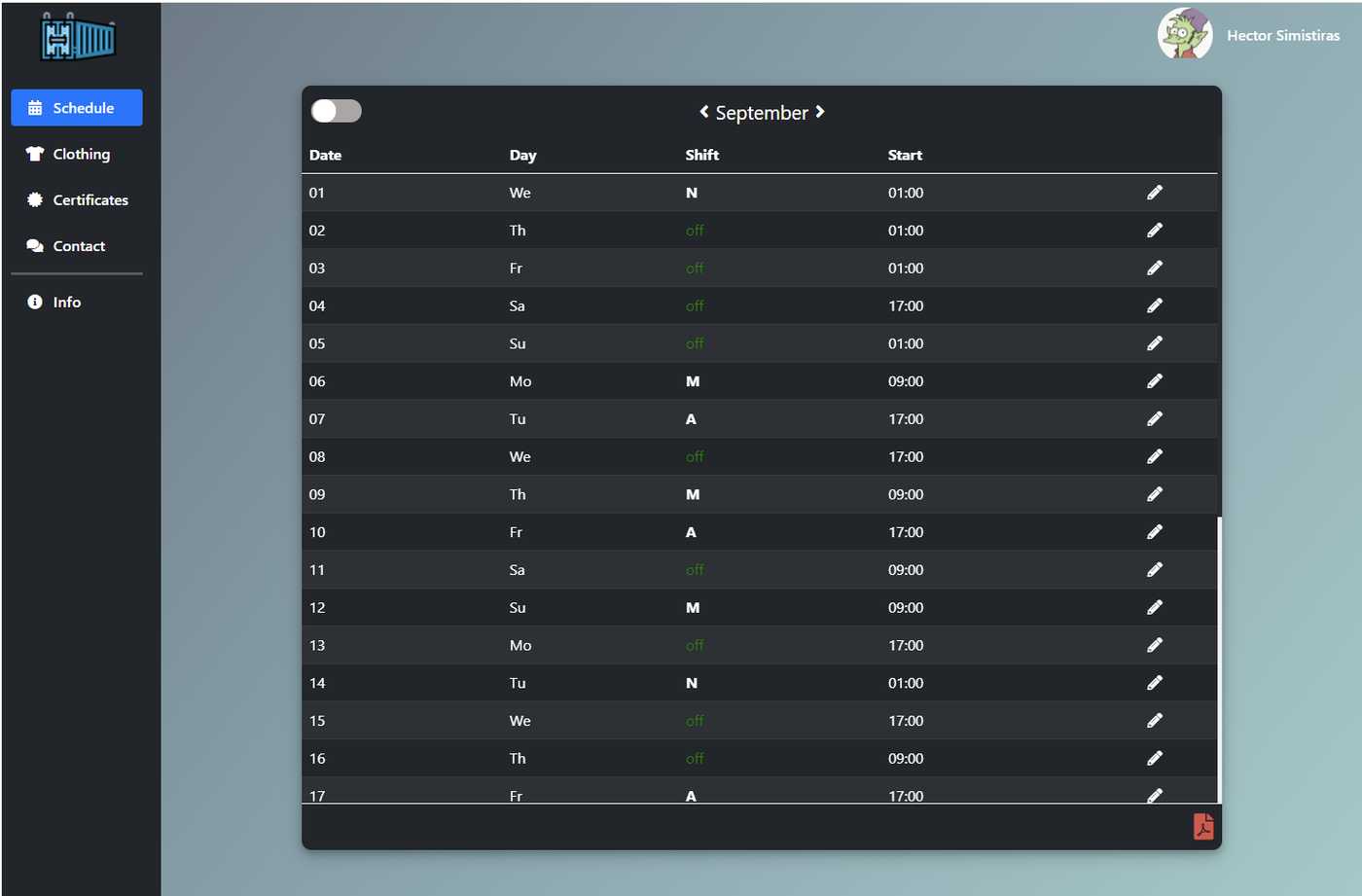
Αυτό επιτυγχάνεται δημιουργώντας τα στοιχεία της σελίδας που δεν θέλουμε να μεταβάλλονται στο αρχικό page και εσωτερικά δημιουργούμε ένα division που θα φιλοξενεί διαφορετικά components ανάλογα με την επιλογή του menu. Για κάθε link στο menu ενεργοποιούμε ένα διαφορετικό component.

Όσον αφορά την οπτικοποίηση του Dashboard, όπως και όλες οι άλλες μας σελίδας είναι πλήρως responsive όπως μπορούμε να δούμε παρακάτω

|  |  |
| --- | --- |
| A picture containing text, monitor, black, cellphone  Description automatically generated | A picture containing text, monitor, electronics  Description automatically generated |

|  |  |
| --- | --- |
| A picture containing text, monitor, electronics, screen  Description automatically generated | A picture containing text, monitor, electronics, screen  Description automatically generated |

**MOTO G4**

****

**DESKTOP**

### Δεδομένα

Τα δεδομένα που θα καλέσουμε βρίσκονται στην βάση ως ένα array από objects, όπου κάθε object αντιπροσωπεύει μία ημέρα του μήνα.

Η εγγραφή **“z”** αντιπροσωπεύει την «ζώνη» εργασίας για την συγκεκριμένη ημέρα και μπορεί να πάρει τις εξής τιμές:

* **1** Για πρωινή ζώνη
* **2** Για απογευματινή ζώνη
* **3** Για Βραδινή ζώνη

Η εγγραφή **“t”** μπορεί να πάρει τις τιμές 0 ή 1 και υποδεικνύει το εάν εργάζεται ή όχι ο χρήστης την συγκεκριμένη ημέρα

Η ημερομηνία **“Date”** φυσικά θα χρησιμεύσει στο να προσεγγίσουμε την μέρα το μήνα και το έτος στο οποίο αναφέρεται το πρόγραμμα εργασίας.

[

    {"Date": "2021-09-01", "z":2, "t":0 },

    {"Date": "2021-09-02", "z":3, "t":1 },

    {...},

    {...}

 ]

### Επεξεργασία Δεδομένων

Τα δεδομένα θα πρέπει να προετοιμαστούν κατάλληλα έτσι ώστε ο τελικός χρήστης να μπορεί να δει στην οθόνη του τα εξής στοιχεία :

* Τον Μήνα στον οποίο αναφέρεται το πρόγραμμα
* Την ημέρα αριθμητικά (01, 02, …)
* Την ημέρα ολογράφως (Μόνο τα δύο(2) πρώτα γράμματα)
* Την βάρδια που εργάζεται ή αν έχει ρεπό
* Την ώρα που ξεκινάει η βάρδια του

Για την υλοποίηση αυτού έχουν δημιουργηθεί δύο κλάσεις

* Η **WorkDay**

Το οποία περιέχει τα απαραίτητα «εργαλεία» που θα τροποποιήσουν τα δεδομένα που θα λάβει στον constructor.

* Η **Work\_Collection**

Η οποία αναλαμβάνει να καλέσει να δημιουργήσει τα instance της WorkDay και με την βοήθεια της knockout.js να κάνει τα τροποποιημένα δεδομένα ορατά στο σύστημα.

O Constructor της κλάσης **WorkDay** θα απομονώσει για μια ημέρα τις εγγραφές για **Date**, **“z”** , **“t”** και στην συνέχεια με τις κατάλληλες συναρτήσεις θα χρησιμοποιήσει μια μεταβλητή για να αποθηκεύσει:

**Τα νούμερα της ημέρας**

        this.day\_num = this.\_day\_num();

**Τα δύο πρώτα γράμματα της ημέρας**

        this.day\_string = this.\_day\_string();

**Την βάρδια στην οποία εργάζεται ή την ένδειξη “off” για ρεπό**

        this.shift = this.\_shift();

**Την ώρα εκκίνησης για την βάρδια**

        this.shift\_start = this.\_shift\_start();

Όταν δημιουργηθεί ένα instance για την **Work\_Collection** o constructor της θα πάρει σαν όρισμα ολόκληρο το ακατέργαστο μηνιαίο πρόγραμμα και θα κάνει τα εξής

**Πρώτον :**

Μέσω της συνάρτησης

    get\_month() {

        let month = new WorkDay(this.monthly\_schedule[0]);

        month = month.\_month();

        this.cur\_month(month);

    }

Θα δημιουργήσει ένα μοναδικό instance της **WorkDay** έτσι ώστε να πάρει τον μήνα στον οποίο αναφέρεται το πρόγραμμα και θα τον κάνει ορατό στο σύστημα

this.cur\_month = ko.observable();

**Δεύτερον :**

Μέσω της συνάρτησης

    build\_Days() {

        let entries = [];

        for (let i = 0; i < this.monthly\_schedule.length; i++) {

            entries.push( new WorkDay(this.monthly\_schedule[i]));

        }

        this.work\_list(entries);

        console.log(entries);

    }

Θα δημιουργήσει τόσα instance της **WorkDay** όσα και τα object που περιέχει το array που πήρε σαν είσοδο, δηλαδή όσες μέρες περιέχει ο συγκεκριμένος μήνας και στην συνέχεια κάθε instance θα γίνει ένα object ενός καινούργιου array που θα είναι επίσης ορατό από το σύστημα

this.work\_list = ko.observableArray([]);

Πλέον τα δεδομένα μας είναι έτοιμα για προβολή.

Υποσημείωση:

Α) Χρησιμοποιούμε τις observable μεταβλητές της knockout γιατί τα δεδομένα παρουσιάζονται δυναμικά και ενδέχεται να αλλάξουν ζωντανά.

Β) Το array με τα instance της **WorkDay** φυσικά έχει πολύ περιττή πληροφορία αλλά δεν χρειάζεται να το επεξεργαστούμε περαιτέρω μιας και η knockout όταν κληθεί στην HTML θα επιλέξει μόνο τα ορίσματα που θα ζητήσουμε χωρίς να χρειαστεί να κάνει προσπέλαση όλων των δεδομένων.

### Παρουσίαση Δεδομένων

Για την παρουσίαση των δεδομένων θα αναφερθούμε μόνο στο ουσιώδες κομμάτι του προγράμματος εργασίας.

Στο προηγούμενο κεφάλαιο μιλήσαμε για την επεξεργασία των δεδομένων στο Model-View κομμάτι του συγκεκριμένου component, η παρουσίαση φυσικά γίνεται στην HTML που είναι το View κομμάτι του component.

Έχουμε δημιουργήσει έναν πίνακα και φτιάχνοντας ένα μοναδικό row μπορούμε να δημιουργήσουμε δυναμικά όσα rows χρειαζόμαστε για την παρουσίαση του προγράμματος.

Ο κώδικας που θα το υλοποιήσει είναι ο εξής:

             <tbody>

                 <!-- ko if:$component.workCollection -->

                 <!-- ko foreach:$component.workCollection().work\_list -->

                 <tr>

                     <td data-bind="text: day\_num"></td>

                     <td data-bind="text: day\_string"></td>

                     <td data-bind="text: shift, style:dayNumStyle"></td>

                      <td data-bind="text: shift\_start"></td>

                    </tr>

                    <!-- /ko -->

                    <!-- /ko -->

                </tbody>

Βλέπουμε λοιπόν πως εντός του body του πίνακα η knockout ενεργοποιείται μέσα από τα σχόλια της HTML.

                    <!-- ko if:$component.workCollection -->

Η **“ko if”** εξετάζει αν συνθήκη που παραθέτουμε είναι αληθής και μόνο τότε υλοποιεί τον κώδικα τον οποίο περικλείει έως το closing tag

                    <!-- /ko -->

Αυτό μας εξασφαλίζει πως το περιεχόμενο του πίνακα θα φορτωθεί μόνο όταν η observable μεταβλητή που περιέχει τα δεδομένα του προγράμματος εργασίας γεμίσει αποφεύγοντας έτσι ανεπιθύμητες συμπεριφορές στην παρουσίαση.

Αμέσως μετά τον έλεγχο χρησιμοποιούμε την

<!-- ko foreach:$component.workCollection().work\_list -->

H **“ko foreach:”** χρησιμοποιείται για να προσπελάσουμε arrays και θα εκτελέσει τον κώδικα μέχρι το closing tag της για κάθε ένα όρισμα που περιέχει το array, στην δική μας περίπτωση θα τον εκτελέσει για όσες ημέρες περιέχει ο μήνας στον οποίο αναφερόμαστε.

Βλέπουμε παρακάτω πως για κάθε κελί υπάρχει ένα **“data-bind”** το οποίο αναφέρεται σαν **text** και δείχνει ένα όρισμα μέσα στο object για το οποίο εκείνη την στιγμή γίνεται προσπέλαση από την knockout. Έτσι επιτυχώς για κάθε ημέρα θα φορτωθούν τα ανάλογα δεδομένα σε κάθε κελί του πίνακα.

### Λήψη Δεδομένων

Μιλήσαμε για την επεξεργασία και παρουσίαση των δεδομένων που αφορούν το πρόγραμμα εργασίας αλλά δεν έχουμε αναφερθεί στο πως φτάνουν αυτά τα δεδομένα στο πρόγραμμα μας από το database.

Την μεταφορά των δεδομένων αναλαμβάνει το service, δεν θα ασχοληθούμε όμως με το πως γίνεται το «σερβίρισμα» αλλά με πως υλοποιούμε την κλήση του από τον client.

Για να κληθεί το πρόγραμμα έχουμε δημιουργήσει την συνάρτηση

 async \_fetchProgram(IsNext) {...}

Λόγο όγκου κώδικα θα περιοριστούμε στο να μιλήσουμε για τα βήματα που εκτελέστηκαν αντί να κάνουμε λεπτομερή ανάλυση του κώδικα.

Η **\_fetchProgram()** θα εκτελεστεί με το που περάσουμε από το Login στο Dashboard,

Το **IsNext** πρόκειται για μια Boolean μεταβλητή και σκοπός της είναι να επιλέξει ποιόν μήνα θα ζητήσει από το service. Μετά την είσοδο στην συνάρτηση θα γίνει έλεγχος, για τιμές false, undefined ή true και θα ζητηθεί το πρόγραμμα για τον τρέχον ή τον επόμενο μήνα.

Το επόμενο βήμα είναι να δημιουργηθεί το endpoint, η διεύθυνση δηλαδή την οποία θα καλέσει η fetch την συγκεκριμένη λειτουργία του service.

Στην δική μας περίπτωση το endpoint θα είναι της μορφής

<https://thesis-api.azurewebsites.net/Thesis/GetProgram/ID/Year/Month>

Παρατηρούμε ότι το ID που αποθηκεύσαμε για τον χρήστη κατα την διάρκεια του Login θα το εισάγουμε εδώ έτσι το service να βρει τον χρήστη που αναζητάμε.

Σαν **“Year”** πάντοτε θα εισάγουμε το τωρινό έτος, ενώ το **Month** θα συμπληρώνεται ανάλογα με την τιμή της **IsNext** μεταβλητής.

Τώρα λοιπόν που έχουμε κατασκευάσει τo endpoint που θα χρησιμοποιήσουμε μπορούμε να χρησιμοποιήσουμε την fetch,

    async \_callFetchService(endpoint){

        try {

            let options = {

                method: "GET",

                credentials: "include",

                headers: {

                    Authorization: "Basic " + btoa(config.user\_id + ":" + config.user\_pwd)

                }

            }

            let response = await fetch(request, options);

            if (response.ok) {

                if (isJSON){

                    let serviceData = await response.json();

                    this.curtain\_table(false);

                    return serviceData;

                              }

                        }

            else

                throw Error(response.statusText);

        }

        catch (e) {

            this.curtain\_table(false);

            console.log(e.message);

        }

    }

Παρατηρούμε επίσης πως το request μας έχει παραλήπτη ένα διαφορετικό domain από αυτό που βρίσκεται η εφαρμογή, κάτι που δεν επιτρέπεται όταν μιλάμε για δεδομένα και όχι για εικόνες σε ένα HTTP request. Εκεί έρχεται να παίξει τον ρόλο του η πολιτική του **CORS** (Cross Origin Resource Sharing). Έτσι λοιπόν για να αποκτήσουμε πρόσβαση στα δεδομένα που ζητάμε και σύμφωνα με το Basic authentication απαιτείται να συμπεριλάβουμε ένα header στο HTTP request της μορφής:

Authorization: Basic YWRtaW46cEBzNXcwcmQ=

Δηλαδή την λέξη Basic και την btoa() κωδικοποίηση (64bit) του username+password.

Τελικά το service επιστρέφει το πρόγραμμα εργασίας σε μορφή string , και αφού χρησιμοποιήσουμε την **JSON.parse(serviceData)** για να προσπελάσουμε εύκολα τα δεδομένα μπορούμε να εκκινήσουμε την διαδικασία επεξεργασίας και προβολής των δεδομένων δημιουργώντας ένα instance της κλάσης WorkCollection.

### Αλλαγές Προγράμματος

Ο χρήστης εκτός από την προβολή του προγράμματος εργασίας του έχει το δικαίωμα να αιτηθεί τις εξής αλλαγές:

* Από ρεπό σε βάρδια
* Από βάρδια σε βάρδια
* Από βάρδια σε ρεπό

Σε οποιοδήποτε πραγματικό σενάριο οποιοδήποτε αίτημα αλλαγής θα έπρεπε να εγκριθεί από το HR της επιχείρησης, στην δική μας όμως περίπτωση καθαρά για λόγους ευκολίας οι αιτήσεις δεν χρειάζονται έγκριση και δεν υπάρχει κανένας περιορισμός στις αλλαγές εκτός από τον υποτυπώδη έλεγχο για παρελθοντικές ημερομηνίες.

Ο χρήστης πατώντας το τελευταίο κελί του πίνακα για κάθε ημέρα καλεί την συνάρτηση

    grab\_day(day\_obj) {

        this.day\_obj(day\_obj);

          let date\_current=new Date();

        if (Date.parse(date\_current)<Date.parse(day\_obj.date)) {

            let temp = day\_obj.date.split("-");

            this.selected\_day(temp[2]);

            this.selected\_workday(day\_obj);

            this.working(day\_obj.t);

            this.selected\_shift(day\_obj.z);

            this.curtain\_lift();

        }

        else {

            config.\_triggerPast();

        }

    }

η οποία με την βοήθεια της Knockout δέχεται σαν όρισμα τα data που χρησιμοποιήθηκαν για να χτιστεί η συγκεκριμένη «γραμμή» του πίνακα, δηλαδή το instance της κλάσης WorkDay που αντιστοιχεί στην ημέρα του μήνα που επιλέχθηκε. Έπειτα κάνει έλεγχο για το αν ο χρήστης επέλεξε παρελθοντική ημερομηνία ή όχι. Αν η ημερομηνία που επιλέχθηκε είναι παρελθοντική το σύστημα εμφανίζει το αντίστοιχο alert και δεν εκκινεί καμία διαδικασία αλλαγής.  
Εάν η ημερομηνία είναι έγκυρη (μελλοντική) τότε απομονώνονται τα δεδομένα που αναφέρονται στην συγκεκριμένη ημέρα και καλείται η συνάρτηση

curtain\_lift(){...}

που εμφανίζει το παράθυρο που επιλέγονται οι αλλαγές με τα στοιχεία που χρειαζόμαστε (πχ σε περίπτωση που ο εργαζόμενος έχει ρεπό την μέρα που επέλεξε να τροποποιήσει το πρόγραμμα του το κουμπί για αίτημα ρεπό δεν θα εμφανιστεί).

Εάν ο χρήστης επιθυμεί να αλλάξει την βάρδια του για την συγκεκριμένη ημέρα μέσω ένος select θα επιλέξει την βάρδια την οποία επιθυμεί, κάνοντας Submit θα καλέσει την συνάρτηση

request\_schedule\_change(condition){…}

η οποία θα αποθηκεύσει την καινούργια βάρδια στην ίδια μεταβλητή που αποθήκευσε την βάρδια στην οποία εργάζεται κατά την εκκίνηση της διαδικασίας και θα σχηματίσει το endpoint που θα χρειαστούμε για να κληθεί η μέθοδος του service που αναλαμβάνει να αλλάξει την εγγραφή στην βάση δεδομένων και πρέπει να έχει την μορφή

"https://thesis-api.azurewebsites.net/Thesis/ID/ChangeDayProgram/Date/z/num"

Όπου

**“ID”** το μοναδικό ID του χρήστη που έχουμε αποθηκεύσει κατά το login

**“Date”** είναι η ημερομηνία στην οποία αναφερόμαστε σε **ISO 8601** μορφή

**“z”** η καινούργια βάρδια που επιλέχθηκε

**“num”** παίρνει 2 τιμές , 0 ή 1 που προκύπτουν από την “condition” Boolean που περάστηκε στην συνάρτηση request\_schedule\_change(condition){…} κατά την αρχική της κλήση, η 0 αναφέρεται σε ρεπό και η 1 σε ημέρα εργασίας.

Αφού έχει σχηματιστεί το endpoint που χρειαζόμαστε για την κλήση ενεργοποιείται η

async \_callFetchService(request) {...}

η οποία καλεί το service με τον ίδιο τρόπο που αναλύσαμε στην προηγούμενη ενότητα αλλά με το καινούργιο endpoint και το service ανανεώνει την τη βάση δεδομένων με τα καινούργια στοιχεία.

Τέλος καλείται η -πλέον γνωστή μας-

\_fetchProgram(boolean);

Αφού επιλεγεί η Boolean μεταβλητή για τον μήνα στον οποίο αιτήθηκε η αλλαγή, το επικαιροποιημένο πρόγραμμα εμφανίζεται στην οθόνη του χρήστη και τέλος γίνεται trigger το notification που ενημερώνει για την επιτυχημένη αλλαγή και η διαδικασία ολοκληρώνεται

Πανομοιότυπη διαδικασία επιλέγεται και στην περίπτωση που ο χρήστης ζήτησε ρεπό μέ μοναδική διαφορά την οτι η Boolean condition που περάσαμε στην συνάρτηση

request\_schedule\_change(condition){…}

μεταφράζεται σε 0 στο endpoint.

### Εκτύπωση Προγράμματος

Ένα πολύ χρήσιμο feature σε κάθε εφαρμογή που προβάλει δεδομένα είναι η δυνατότητα του χρήστη να έχει πρόσβαση σε αυτά ακόμη και εάν βρίσκεται εκτός σύνδεσης, αυτό επιτυγχάνεται με το να παρέχεται η δυνατότητα εκτύπωσης ή απόθήκευσης των δεδομένων σε μορφή pdf.

Παρότι υπάρχουν αρκετές open source βιβλιοθήκες που μας δίνουν την δυνατότητα να επιλέξουμε τα δεδομένα που θέλουμε να δώσουμε την δυνατότητα στον τελικό χρήστη να αποθηκεύσει/εκτυπώσει αποφασίσαμε να εκμεταλλευτούμε την τεχνική του SPA και με την βοήθεια της **knockout** να δημιουργήσουμε μια μέθοδο που θα μας επιτρέπει να απομονώνουμε την πληροφορία που θέλουμε ανεξάρτητα από το που βρίσκεται στο πρόγραμμα και να την εκτυπώσουμε ή μετατρέψουμε σε αρχείο .**PDF**, παρακάτω θα αναλύσουμε την διαδικασία.

Η μέθοδος του **Windows.print()** δίνει προς εκτύπωση ή λήψη το HTML που προβάλλεται την δεδομένη στιγμή στον browser. Για να απομονώσουμε τα δεδομένα που χρειαζόμαστε χρησιμοποιήθηκε η εξής τεχνική.

Η «είσοδος» για την παρουσίαση του προγράμματος είναι η index.html όπου εντός του body υπάρχει ένα division που έχουμε ονομάσει **Page Host** το οποίο φιλοξενεί το root component.  
Για να μπορούμε να απομονώνουμε δεδομένα δημιουργήσαμε ένα δεύτερο, άδειο by default και μη ορατό division στο οποίο θα αναφερθούμε ως **Printing Host**.

Πρώτο βήμα είναι να «τυλίξουμε» τον κώδικα που θέλουμε να προωθήσουμε προς εκτύπωση σε ένα division και με την βοήθεια της Knockout και ενός πολύ απλού function να τα αποθηκεύσουμε σε μια μεταβλητή στον constructor της κλάσης που ανήκει το template

<div data-bind="template: {afterRender: function(){$component.startWatching($element)}}">

    <!-- Data προς εκτύπωση / λήψη-->

</div>

startWatching(element){

    this.printDiv=element;

}

this.printDiv=null;

Το κουμπί/εικονίδιο/link που θα επιλέξουμε για την εκτύπωση

<a data-bind="click:print">print something!</a>

Θα καλεί μια συνάρτηση η οποία θα επιλέγει το division του **Printing Host**

Θα αδειάζει το περιεχόμενό του (μπορεί να περιέχει data από προηγούμενη εκτύπωση)

Θα παίρνει το template που αποθηκεύσαμε νωρίτερα, θα το τοποθετεί στον **Printing Host**

και τελικά θα καλεί την **window.print**

print() {

    let $ph = $("#printingHost");

    $ph.html("");

    let \_html = $(this.printDiv).html();

    $ph.html(\_html);

    setTimeout(window.print, 333);

}

Υποσημείωση: Η **setTimeout()** τοποθετείται με σκοπό να δώσει τον απαραίτητο χρόνο στον browser να φορτώσει τυχόν γραφικά που θα μεταφερθούν στον **Printing Host**.

Με την βοήθεια της CSS ορίζουμε πότε θέλουμε να γίνεται ορατό το **Printing Host** division και πότε το **Page Host**

@media screen {

  #PageHost {

    display: block; }

  #printingHost {

    display: none; } }

@media print {

  #PageHost {

    display: none !important; }

  #printingHost {

    display: block; } }

# Διαφημιστική ιστοσελίδα λειτουργιών

## Εισαγωγή

Η εργασία σχετικά με το πρόγραμμα εργασίας έχει αναπτυχθεί εμπνευσμένη από συστήματα roster που κυκλοφορούν στην αγορά, στην πραγματικότητα όμως η λειτουργικότητα του είναι υποτυπώδης σε σύγκριση με τις δυνατότητες και τα εργαλεία που προσφέρει ένα πραγματικό σύστημα roster. Έτσι λοιπόν αποφασίσαμε να καταγράψουμε τις λειτουργίες ενός πραγματικού συστήματος και να δημιουργήσουμε μια διαφημιστική ιστοσελίδα που συμπεριλαμβάνουμε σαν εξωτερικό σύνδεσμό στην εφαρμογή.

## Σχεδίαση

Η ιστοσελίδα σχεδιάστηκε για την ενημέρωση υποψήφιων αγοραστών σχετικά με τις λειτουργίες του προγράμματος. Αποτελείται από μια μοναδική σελίδα και σχεδιάστηκε με την τεχνική του **12-column grid** που υποστηρίζεται άψογα σε συνδυασμό με το απαραίτητο responsiveness από το framework για την CSS , Bootstrap.

Κάθε Section αποτελείται από ένα Grid structure Που έχει την εξής μορφή

   <section id="diaxirisi">

            <!--functions ΔΙΑΧΕΙΡΗΣΗ ΒΑΡΔΙΩΝ -->

            <div class="container-fluid bg-light">

                <div class="heading-functions pos-r ">

                    <div class="row" style="width:800px;">

                        <div class="col-1 text-center"><i class="fas fa-tasks fa-2x fa-blue mx-auto" id="fa"></i></div>

                        <div class="col-11">

                            <h4 class="mb-0 "><span>Διαχείρηση Βαρδιών</span><span onclick="rotateFn()" class="triangle"

                                    data-toggle="collapse" data-target="#toggle-diaxirisi"> </span></h4>

                        </div>

                    </div>

                </div>

                <div id="toggle-diaxirisi" class="collapse show">

                    <div class="spacer text-heavy">

                        <div class="row">

                            <div class="col-lg-4 pr-4 flex-column d-flex justify-content-between">

                                <!--BIG COLUMN 1/3-->

                                <div class="row ">

                                    <!--1/3 row-->

                                    <div class="col-1"></div>

                                    <div class="col-11">

                                        <h6 class="mb-2"></h6>

                                        <p class="mb-4 pl-2"></p>

                                        <h6 class="mb-2"> </h6>

                                        <p class="mb-4 pl-2"> </p>

                                    </div>

                                </div>

                                <div class="row">

                                    <!--2/3 row-->

                                    <div class="col-1"></div>

                                    <div class="col-11">

                                        <h6 class="mb-2"></h6>

                                        <p class="mb-4 pl-2"></p>

                                        <h6 class="mb-2"></h6>

                                        <p class="pl-2 mb-4"> </p>

                                    </div>

                                </div>

                                <div class="row ">

                                    <!--3/3 row-->

                                    <div class="col-1"></div>

                                    <div class="col-11 ">

                                        <h6 class="mb-2"></h6>

                                        <p class="mb-3 pl-2"></p>

                                    </div>

                                </div>

                            </div>

                            <!--BIG COLUMN 1/3 EAND-->

                           <div class="col-lg-4 order-lg-12">

                                <!-- BIG COLUMN 2/3-->

                                <img loading="lazy" class="img-fluid img-center mb-3 mb-lg-0 "

                                    src="./icons/swimlane-subscribe-to-news-tips.svg" alt="haha stolen from necrosoft">

                            </div>

                            <div class="col-lg-4 order-lg-4 flex-column d-flex justify-content-between">

                                <!--BIG COLUMN 3/3-->

                                <div class="row">

                                    <!--1/3 row-->

                                    <div class="col-1"></div>

                                    <div class="col-11">

                                        <h6 class="mb-2"></h6>

                                        <p class="mb-4 pl-2"></p>

                                        <h6 class="mb-2"></h6>

                                        <p class="mb-4 pl-2"></p>

                                    </div>

                                </div>

                               <div class="row">

                                    <!--2/3 row-->

                                    <div class="col-1"></div>

                                    <div class="col-11">

                                        <h6 class="mb-2"></h6>

                                        <p class="mb-4 pl-2"></p>

                                       <h6 class="mb-2"> </h6>

                                        <p class="pl-2 mb-4"> </p>

                                    </div>

                                </div>

                                <div class="row">

                                    <!--3/3 row-->

                                    <div class="col-1"></div>

                                    <div class="col-11">

                                        <h6 class="mb-2"> </h6>

                                        <p class="mb-3 pl-2"></p>

                                    </div>

                                </div>

                            </div>

                            <!--BIG COLUMN 3/3 EAND-->

                        </div>

                    </div>

                </div>

            </div>

        </section>

Παρατηρούμε πως στο section εκτός από την επικεφαλίδα αποτελείται από ένα σύστημα grid με 3 rows και 12 columns, ενώ έχει προστεθεί η δυνατότητα (πάλι μέσω της Bootstrap) να γίνεται collapse όλο το section κάτω από την επικεφαλίδα της ιστοσελίδας έτσι ώστε ο χρήστης να μπορεί να βελτιώσει την εμπειρία πλοήγησης του. Με πανομοιότυπο τρόπο δημιουργήθηκαν όλα τα section που αφορούν την ενημέρωση του χρήστη για τις λειτουργίες του roster ενώ παρακάτω θα δούμε πως αντιδράει η εφαρμογή για διαφορετικές συσκευές, δηλαδή το responsiveness.

**Graphical user interface, website

Description automatically generated**

**Desktop**

**Graphical user interface, application, Teams

Description automatically generated**

**iPad**

**A screenshot of a computer

Description automatically generated with medium confidence**

**iPad (Collapsed Sections)**

# Συστήματα Roster

## Εισαγωγή

Έχοντας πλέον ολοκληρώσει την ανάλυση της εργασίας η οποία είναι «εμπνευσμένη» από συστήματα Roster θα εξετάσουμε τις λειτουργίες που θα πρέπει να προσφέρει ένα τέτοιο σύστημα στην συνέχεια θα αναλύσουμε κάποια μοντέλα που κυκλοφορούν στην Ελληνική και όχι μόνο αγορά.

### Περιγραφή

Τα roster συστήματα έχουν σαν στόχο την οργάνωση υπάρχοντος ανθρώπινου δυναμικού, ως προς την εργασία, σε δεδομένο επιχειρησιακό περιβάλλον. Δεδομένου δε ότι τα επιχειρησιακά περιβάλλοντα διαφοροποιούνται σημαντικά μεταξύ τους, σε μία σειρά παραγόντων, έχουν ανάγκη να εφαρμόζουν διαφορετικούς τρόπους  οργάνωσης του ΑΔ.

Η κάλυψη από τα roster συστήματα, κατά το δυνατόν, των περισσοτέρων χαρακτηριστικών αυξάνει τον βαθμό πολυπλοκότητας και συνθετότητας αυτών. Έτσι προκύπτουν συστήματα roster που έχουν τη δυνατότητα να εξυπηρετούν, εν γένει, σταθερό εργασιακό περιβάλλον έως και τα πλέον σύνθετα, που έχουν στόχο την δυναμική προσαρμογή  του ΑΔ σε μια διαρκώς μεταβαλλόμενη ζήτηση.

### Υποστηριζόμενοι Τύποι εργαζομένων.

Οι τύποι εργαζομένων, που μπορεί να υποστηρίζονται από τα roster συστήματα, είναι:

Σταθερού / Κλιμακωτού Ωραρίου: ο εργαζόμενος έχει σταθερό ωράριο στη διάρκεια της οριζόμενης περιόδου από την σύμβαση. Κύρια χαρακτηριστικά:

* Εργάζεται καθορισμένες ημέρες τις εβδομάδος, όσες ορίζονται στη σύμβαση, συνήθως Δευτέρα - Παρασκευή.
* Έχει σταθερή βάρδια, Πρωινή ή Απογευματινή. Οπωσδήποτε δεν εργάζεται σε Νυχτερινή βάρδια.
* Στο Κλιμακωτό ωράριο, ένας εργαζόμενος ή μια υποομάδα μπορεί να έχει διαφορετική ώρα έναρξης στην ίδια βάρδια

Κυλιόμενου Ωραρίου: ο εργαζόμενος έχει κυλιόμενο ωράριο στη διάρκεια της οριζόμενης περιόδου από την σύμβαση. Κύρια χαρακτηριστικά:

* Εργάζεται όσες ημέρες ορίζονται από τη σύμβαση, στο διάστημα από Δευτέρα - Κυριακή.
* Εναλλάσσεται στις λειτουργούσες βάρδιες, βάσει υπάρχουσας κατανομής.

Ανεξάρτητος εργαζόμενος: ο εργαζόμενος έχει σταθερό ή κυλιόμενο ωράριο, με αντίστοιχα χαρακτηριστικά, καλούμενος προς εργασία μέσω υπεργολάβου.

### Λειτουργικοί Παράγοντες

Μερικοί από τους σημαντικούς παράγοντες, που επηρεάζουν την οργάνωση της εργασίας, μπορεί να είναι:

* Η λειτουργίας της επιχείρησης ανά εβδομάδα: 5-θήμερη, 6-ήμερη, 7-ήμερη ή άλλη.
* Η λειτουργία βαρδιών: αριθμός βαρδιών ανά 24-ωρο
* Η διεκπεραίωση σταθερού ή κυμαινόμενου φορτίου εργασίας
* Οι δεξιότητες των εργαζομένων: διαθέτουν μία η περισσότερες εξειδικεύσεις
* Η επιχειρησιακή διάρθρωση: αξιοποίηση ΑΔ σε διαφορετικά εργασιακά κέντρα.

### Περίοδος λειτουργίας

Ο αριθμός των ημερών λειτουργίας ανά εβδομάδα, είναι ένας από τους βασικούς παράγοντες ενός roster συστήματος. Σε συνδυασμό με τα υποστηριζόμενα μοντέλα εργασίας, πλήρους απασχόλησης ή εκ περιτροπής,  και τη λειτουργία ή μη βαρδιών, καθορίζουν το πρόγραμμα εργασίας του εργαζόμενου.

Το πρόγραμμα εργασίας θα πρέπει να καλύπτει, τουλάχιστον, μία αρχική σταθερή ζήτηση για τις ημέρες λειτουργίας ανά εβδομάδα, με βάση τις συμβάσεις των εργαζομένων.

Εργασία σε βάρδιες

Η λειτουργία σε βάρδιες, είναι σημαντικός παράγοντας για την κατασκευή του προγράμματος εργασίας των εργαζομένων, θα καλύπτει την ζητούμενη κατανομή στις ζώνες εργασίας, πχ. 4/2/1, που σημαίνει 4 εργαζόμενοι στην Πρωινή βάρδια, 2 εργαζόμενοι στην Απογευματινή βάρδια και 1 εργαζόμενος στην Νυχτερινή βάρδια. Προφανώς, το πρόγραμμα εργασίας για την παραπάνω κατανομή, ικανοποιεί και οποιαδήποτε με ακέραιο πολλαπλασιαστή, δηλ. 8/4/2, 12/6/3 κοκ.

Επιπλέον, το πρόγραμμα εργασίας θα πρέπει να συμμορφώνεται με τις ισχύουσες νομικές δεσμεύσεις, όπως: ομαλή διαδοχή βαρδιών (χωρίς κόντρα βάρδια), ελεύθερη (ρεπό) Κυριακή, τουλάχιστον, μία ανά 7 εβδομάδες, ελάχιστη απόσταση μεταξύ διαδοχικών βαρδιών μεγαλύτερη των 11 ωρών, κλπ.

Πρόσθετα, το πρόγραμμα εργασίας θα πρέπει να εξασφαλίζει την ισότητα των εργαζομένων που συμμετέχουν στην ομάδα. Αυτό σημαίνει ότι όλοι οι εργαζόμενοι της ομάδας, στη διάρκεια συγκεκριμένου χρονικού διαστήματος, που είναι η περίοδος του προγράμματος, έχουν κάνει τον ίδιο αριθμό βαρδιών σε κάθε ζώνη.

### Φόρτος Εργασίας

Σε μία επιχείρηση ή σε ένα τμήμα της, ο φόρτος εργασίας που αντιμετωπίζουν μπορεί να είναι σταθερός ή να μεταβάλλεται σε επίπεδο εβδομάδας, ημέρας ή ακόμα και βάρδιας. Ο σταθερός φόρτος εργασίας καλύπτεται με εργαζόμενους που είναι σε Σταθερό / Κλιμακωτό Ωράριο ή σε Κυλιόμενο Ωράριο με σταθερή αναλογία εργαζομένων ανά βάρδια (Σταθερή Ζήτηση).

Για την αντιμετώπιση μεταβαλλόμενου φόρτου εργασίας θα πρέπει να προσδιορίζεται η εκάστοτε ζήτηση και μάλιστα ανεξάρτητα για κάθε υπάρχουσα εξειδίκευση (Κυμαινόμενη Ζήτηση). Η κυμαινόμενη ζήτηση είναι δυνατόν να μεταβάλλει τον αρχικό προγραμματισμό των εργαζομένων, πχ κλήση από ρεπό, τροποποίηση της ώρας έναρξης κ.λπ.

Ο προγραμματισμός των παραπάνω μεταβολών, απαιτεί την γρήγορη και όσο το δυνατόν αυτοματοποιημένη επικοινωνία του εκάστοτε υπεύθυνου με τον εργαζόμενο. Βοηθητικά για την κάλυψη της κυμαινόμενης ζήτησης είναι:

* Χρήση κινητού για την ενημέρωση του εργαζόμενου με το τρέχον πρόγραμμα εργασίας του και τις άδειες του
* Άμεση ενημέρωση του εργαζόμενου σχετικά με τυχόν αλλαγές (SMS, Viber, κλπ.)
* Χρήση portal για δήλωση διαθεσιμότητας και γενικότερα αυτοματοποιημένη διαχείριση αιτημάτων.

### Δεξιότητες εργαζομένων

Ένα πρόσθετο χαρακτηριστικό των roster συστημάτων είναι η δυνατότητα  να διαχειρίζονται περισσότερες από μία εξειδικεύσεις που τυχόν διαθέτει ο εργαζόμενος.

Το χαρακτηριστικό αυτό δίνει τη δυνατότητα βέλτιστης αξιοποίησης του, κάθε φορά διαθέσιμου, ανθρώπινου δυναμικού. Συγκεκριμένα, επιτρέπει την ικανοποίηση της ζήτησης με τις λιγότερες δυνατές μεταβολές.

### Ροές εργαζομένων

Ορισμένες φορές, εργαζόμενοι που ανήκουν σε διαφορετικά εργασιακά κέντρα της επιχείρησης, μπορεί να έχουν κοινές εξειδικεύσεις / δεξιότητες.

Υπάρχουν περιπτώσεις, είτε λόγω εποχικότητας ή λόγω συγκυρίας, οι εργαζόμενοι του ενός εργασιακού κέντρου να είναι σε περίσσεια  και ενός άλλου να είναι σε έλλειψη.

Όταν εμφανίζεται τέτοια περίπτωση είναι επιθυμητό να υπάρχει δυνατότητα “ροής” από το εργασιακό κέντρο, που έχει περίσσεια, προς το εργασιακό κέντρο, που έχει έλλειψη.

## Ανάλυση Συστημάτων

Θα πρέπει να λάβουμε υπόψη πως τα συστήματα roster, χωρίζονται σε δύο άτυπες κατηγορίες, high & low end.

Τα Low End συστήματα εστιάζουν σε επιχειρήσεις όπως εστιατόρια δίνοντας έμφαση στην επικοινωνία μεταξύ των εργαζομένων, προσφέροντας υπηρεσίες παρόμοιες με τα μέσα κοινωνικής δικτύωσης, δηλαδή παρέχουν πληροφορίες σχετικά με συναδέλφους, και υπηρεσίες μηνυμάτων, είτε σε προσωπικό επίπεδο είτε κοινοποιήσεις προς ομάδες ανθρώπων. Το πλεονέκτημα τέτοιων συστημάτων είναι η ευκολία που προσφέρετε στους εργαζόμενους να συνεννοηθούν σχετικά με τις βάρδιες τους καθώς και να βρίσκουν εύκολα αντικαταστάτες στην περίπτωση που επιθυμούν να λείψουν από την εργασία τους.

Τα High End συστήματα κυρίως εμβαθύνουν στο προγραμματιστικό κομμάτι, με περισσότερους και πιο πολύπλοκους υπολογισμούς που αφορούν τις εργατοώρες, τις άδειες, την τήρηση της νομοθεσίας, την εξοικονόμηση πόρων και την αύξηση της παραγωγικότητας.

Παρακάτω θα μιλήσουμε για μερικά δημοφιλή roster συστήματα που υπόκεινται και στις δύο κατηγορίες :

* **Humanity**
* **Kronos**
* **Connecteam**

### Humanity

Το Humanity είναι ένα από τα πιο διαδεδομένα roster συστήματα, υπόκειται στην κατηγορία των Low End συστημάτων και χρησιμοποιείται κυρίως σε επιχειρήσεις όπως εστιατόρια. Το Humanity επιτρέπει στους εργαζόμενους να βλέπουν με πολλές παραλλαγές το πρόγραμμα εργασίας τους, να αιτούνται άδειες και να βλέπουν τον μισθό τους σύμφωνα με τις βάρδιες που έχουν οριστεί για τον μήνα.

Το Humanity δίνει μεγάλη έμφαση στο User Experience προσφέροντας ένα όμορφο, καθαρό και εύχρηστο περιβάλλον είτε σε μορφή mobile app, είτε σε μορφή προγράμματος. Οι χρήστες του μπορούν εκτός από τις λειτουργίες που αφορούν άμεσα το πρόγραμμα και την μισθοδοσία τους να έχουν πρόσβαση σε πληροφορίες για τους συναδέλφους τους (προσωπικά μηνύματα, ενημέρωση για γενέθλια κα) και να ειδοποιούνται μέσω notification για αλλαγές ή έκδοση νέων προγραμμάτων.

Οι διαχειριστές του προγράμματος μπορούν να δημιουργούν εύκολα σε ένα φιλικό περιβάλλον τα προγράμματα εργασίας ενώ παράλληλα το ίδιο το πρόγραμμα αναλαμβάνει πλήρη έλεγχο σχετικά με το αν κάθε εργαζόμενος έχει πάρει τις ώρες που του αναλογούν και εάν υπάρχουν conflicts βαρδιών στο πρόγραμμα που δημιουργήθηκε.

Επίσης υπάρχει πλήρης έλεγχος των εργαζομένων, κάθε διαχειριστής μπορεί να δει την ώρα που έφτασε για δουλειά κάθε εργαζόμενος, πότε πήρε και πόσο κράτησε το διάλειμμά του καθώς και πόση ώρα συνολικά άργησε μέσα στο μήνα. Μια επίσης πολύ χρήσιμη λειτουργία είναι η εκπαίδευση όπου οι διαχειριστές μπορούν να κάνουν ‘live’ σεμινάρια ή και να ανεβάζουν υλικό που θέλουν να κοινοποιήσουν σε συγκεκριμένες ομάδες εργαζομένων. Παραδείγματος χάρη σε ένα εστιατόριο ο διαχειριστής θα μπορούσε να «μοιράσει» στους μάγειρες οδηγίες για το καινούργιο πιάτο που θέλει να συμπεριληφθεί στο menu του καταστήματος συμπεριλαμβάνοντας και φωτογραφικό υλικό που αφορά την παρουσίαση.

Τέλος πολύ σημαντική κρίνεται η δυνατότητα των διαχειριστών να δημιουργούν custom reports σχετικά με τα data που θέλουν να μελετήσουν χρησιμοποιώντας τα φίλτρα που τους παρέχονται, όπως ημερομηνία/τοποθεσία/εργαζόμενοι και ειδικότητα.

Αυτή την στιγμή το “Humanity” είναι το πιο διαδεδομένο cloud based, low end roster σύστημα, ας δούμε όμως τα υπέρ και τα κατά.

|  |  |
| --- | --- |
|  |  |
|  |  |
| **Υπέρ**   1. Εύκολη επικοινωνία μεταξύ εργαζομένων αλλά και διαχειριστών 2. Εύκολη πρόσβαση σε δεδομένα 3. Κατάλληλο για προβολή μικρών δειγμάτων δεδομένων 4. Εύχρηστο και κατανοητό Interface | **Κατά**   1. Ακριβό σε σχέση με τον ανταγωνισμό 2. Περιορισμένη ανάλυση δεδομένων και αναφορών |
|  |  |

### Kronos

To Kronos υπόκειται στην κατηγορία των High End roster συστημάτων.

Είναι μια cloud LOB( Line of Business) σχεδιαστικά εφαρμογή που δίνει έμφαση στην διαχείριση των εργαζομένων και σε εργαλεία που αφορούν αναφορές, στατιστικές, τήρηση νομικών υποχρεώσεων και μισθοδοσίας με στόχο όχι μόνο την σωστή διαχείριση της επιχείρησης και του προγράμματος εργασίας αλλά και την βελτίωση της παραγωγικότητας των εργαζομένων.

Ένα από τα μεγαλύτερα πλεονεκτήματα του Kronos είναι η παρακολούθηση δεδομένων σε πραγματικό χρόνο που επιτρέπει στους διαχειριστές του συστήματος να αντιληφθούν και αντιμετωπίσουν προβλήματα χωρίς να περιμένουν τις καθιερωμένες εβδομαδιαίες ή και μηνιαίες αναφορές.

Επίσης θα πρέπει να λάβουμε υπόψη πως το Kronos έχει τη δυνατότητα να προβάλλει τα δεδομένα για πολύ συγκεκριμένες κατηγορίες και με επίσης πολύ συγκεκριμένη παραμετροποίηση. Έτσι πολλές φορές δεν χρειάζεται καν η μεσολάβηση αναλυτών για να παράγει ένας διαχειριστής τα αποτελέσματα που χρειάζεται για να λάβει τις απαραίτητες αποφάσεις. Καταλαβαίνουμε πως αυτά τα χαρακτηριστικά κάνουν το Kronos κατάλληλο για μεγάλες επιχειρήσεις, στο πελατολόγιο τους συμπεριλαμβάνονται εταιρίες όπως η (Tesla, Puma, MGM κα) ενώ χρησιμοποιείται και από μεγάλους κυβερνητικούς οργανισμούς.

Το Kronos είναι ένα πολύπλοκο σύστημα με πολλές παραμέτρους και εργαλεία, παρότι ένας εκπαιδευμένος διαχειριστής μπορεί να το αξιοποιήσει κατάλληλα για την δουλειά του παράγοντας στοχευμένα αποτελέσματα στις αναζητήσεις του η διαδικασία εκμάθησης είναι χρονοβόρα και δύσκολη και πολλές φορές χρειάζεται η πρόσληψη ειδικών για να αναλάβουν να εκπαιδεύσουν τους χρήστες του προγράμματος.

Επίσης στα αρνητικά θα συγκαταλέξουμε πως ακριβώς επειδή το Κρόνος έχει σχεδιαστεί για μεγάλες επιχειρήσεις η απομόνωση μικρών και συγκεκριμένων κομματιών πληροφορίας είναι εξαιρετικά δύσκολη.

|  |  |
| --- | --- |
| Υπερ. | Κατά. |
| 1. Διαχείριση και ανάλυση δεδομένων σε πραγματικό χρόνο 2. Μεγάλη γκάμα εργαλείων | 1. Δύσκολος χειρισμός 2. Δύσκολη απομόνωση μικρών κομματιών πληροφορίας. |

### Connecteam

Πρόκειται για ακόμη μια Low End roster εφαρμογή. Όπως είδαμε και νωρίτερα στόχος τέτοιου τύπου εφαρμογών δεν είναι η ανάλυση σύνθετων δεδομένων αλλά η εύκολη και στοχευμένη επικοινωνία μεταξύ των χρηστών και η δημιουργία προγραμμάτων εργασίας εύκολα και γρήγορα σε ένα απλό και κατανοητό Interface.

Το connecteam έχει χαρακτηριστικά που βοηθούν τους διαχειριστές να δημιουργούν προγράμματα εργασίας με απλό και γρήγορο τρόπο. Πολλές φορές σε μικρότερες ειδικά επιχειρήσεις υπάρχουν περιπτώσεις που τα εβδομαδιαία/μηνιαία προγράμματα ή ακόμα και συγκεκριμένες βάρδιες ατόμων αλλάζουν από λίγο έως καθόλου. Για αυτό τον λόγο οι διαχειριστές δημιουργούν προγράμματα εργασίας τα οποία μπορούν να αποθηκεύουν ως templates και επαναχρησιμοποιούν καθιστώντας την διαδικασία γρήγορα και αποτελεσματική.

Στην εφαρμογή υπάρχει λειτουργία επικοινωνίας παρόμοια με αυτή που προσφέρουν τα μέσα κοινωνικής δικτύωσης, σε μια εφαρμογή όμως που ο στόχος δεν είναι οι κοινωνικοποίηση αλλά η συνεννόηση μεταξύ των εργαζομένων θέλουμε η ανταλλαγή μηνυμάτων να γίνεται στοχευμένα με σκοπό την σωστή ενημέρωση χωρίς ανούσια μηνύματα και άχρηστες ειδοποιήσεις. Έτσι το connecteam δημιουργεί αυτόματα ομάδες επικοινωνίας ανάλογα με την ειδικότητα και τις βάρδιες που αναλαμβάνουν με σκοπό να εξοικονομήσει χρόνο στους χρήστες και να τους απαλλάξει από περιττές κινήσεις και μηνύματα.

Τέλος η δημιουργία προγράμματος είναι σχεδιασμένη σε ένα τέτοιο περιβάλλον όπου κατά την δημιουργία του προγράμματος εργασίας οι βάρδιες μπορούν να δημιουργηθούν εύκολα, γρήγορα , να γίνουν αντιγραφή, να μετακινηθούν με drag n drop και να προστεθούν σημειώσεις σε αυτές.

Κάνοντας λοιπόν εύκολη την επικοινωνία μεταξύ των εργαζομένων/υπευθύνων αλλά και παρέχοντας στους διαχειριστές έναν γρήγορο, εύκολο και αξιόπιστο τρόπο να δημιουργούν προγράμματα εργασίας το Connecteam εξασφαλίζει σε επιχειρήσεις έναν εύκολο και αξιόπιστο τρόπο να προγραμματίζουν την εργασίας τους καθώς και να επικοινωνούν γρήγορα και έξυπνα για οτιδήποτε αφορά την εργασία τους.

Καταλαβαίνουμε λοιπόν πως όχι μόνο στην συγκεκριμένη περίπτωση αλλά και γενικότερα τα Low End συστήματα απευθύνονται σε μικρότερες επιχειρήσεις δίνοντας τους την δυνατότητα να οργανώνουν και να σχεδιάζουν το άμεσο μέλλον με άμεσο και αποτελεσματικό τρόπο μέσω μιας εφαρμογής στην οποία δεν χρειάζεται να εκπαιδευτούν.

|  |  |
| --- | --- |
| Υπερ. | Κατά. |
| 1. Εύκολη χρήση, Φιλικό περιβάλλον 2. Στοχευμένη επικοινωνία 3. Πολύ εύκολη δημιουργία προγραμμάτων εργασίας 4. Δεν απαιτεί εκπαίδευση | 1. Περιορισμένη γκάμα εργαλείων 2. Περιορισμένη έκδοση reports |