



**UANL**

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

**FCFM**

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS



Universidad Autónoma de Nuevo León  
Facultad de Ciencias Físico Matemáticas

Programación Web 2

**Documentación**

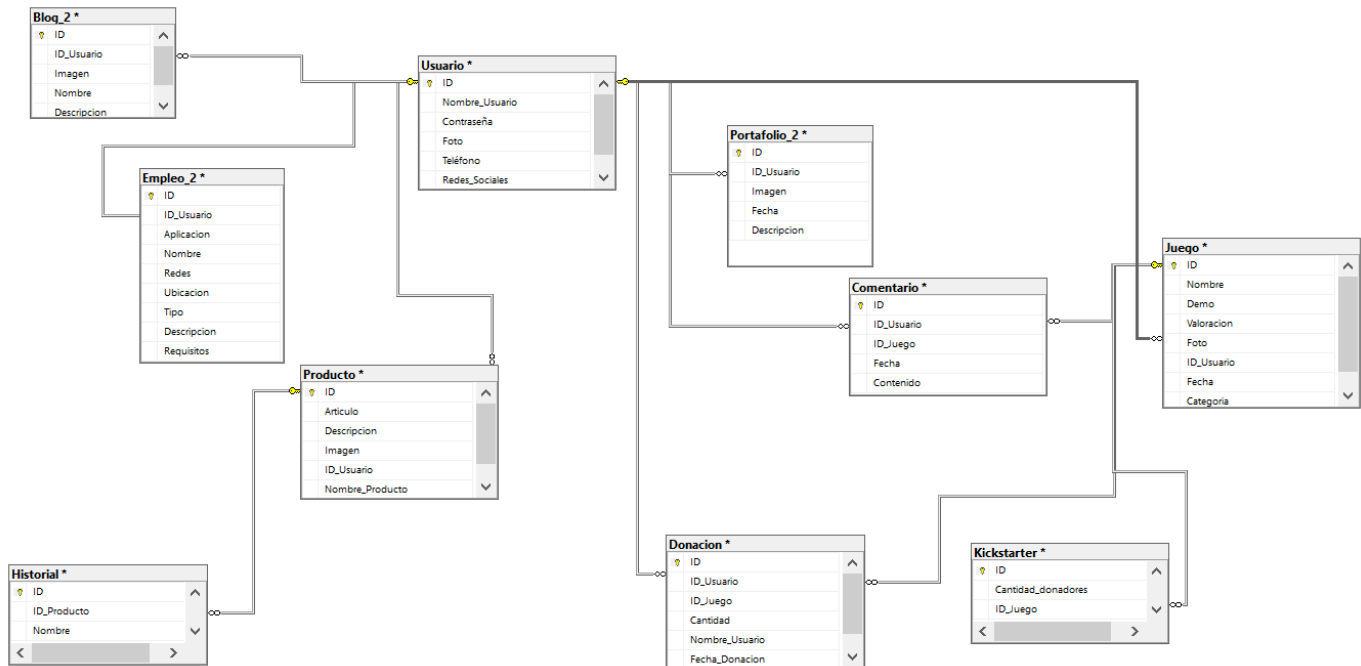
Profesor: Enrique Tolentino García

Grupo 001

Maura Aurora Silva Cantú	1834318
Erik Noé Medina Hernández	1803889

Ciudad Universitaria Semestre Enero-Junio (2022)

- Diagrama del diseño de Base de Datos



- Tabla Usuario

```
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26

exports.usuario_getall = async(req,res) =>{
  const data = await Usuario.find();
  res.send(data);
}

exports.indie_create = async(req,res)=>{
  const {body} = req;
  //validacion de informacion
  let newIndie = new Usuario(body);
  await newIndie
    .save()
    .then((newObject)=> console.log("Success!",newObject))
    .catch((err) => {
      console.error("OH NO!", err)
      res.send(err.errors);
    });
  res.send(newIndie);
};
```

**Get:** Aquí se recibe el comando para buscar información de la tabla y responde con dicha información. Desplegando cada dato de la tabla Usuario.

**Create:** Recibe el comando del para crear una nueva tabla y responde generando una nueva tabla Usuario.

```

26
27 exports.indie_update = async(req,res) => {
28   const { id } = req.params;
29   const { body } = req;
30
31   const indiedb = await Usuario.findById(id);
32
33   if(indiedb){
34     const data = await Usuario.findOneAndUpdate({_id: id}, body,{returnOriginal:false});
35     res.send({message:"Registro actualizado con exito",data})
36   }
37   else {
38     res.send({message:"El registro que intentas actualizar no existe"})
39   }
40 };
41
42 exports.indie_delete=async(req,res)=>{
43   const {id} = req.params;
44
45   await Usuario.deleteOne({_id: id});
46
47   res.send({message:"Registro eliminado"});
48 };

```

**Update:** recibe el ID de una tabla de Usuarios para buscarla y responde mostrando toda la información de dicha tabla para editarla y actualizar los datos.

**Delete:** Recibe el ID de una tabla Usuarios para buscarla, y si la encuentra, borra el usuario encontrado.

## • Tabla Juego

```

2
3 exports.juego_create = async (req,res) =>{
4   const{body} = req;
5
6   const newJuego = new Juego(body);
7
8   await newJuego.save()
9   .then((newObject)=> console.log("Success!",newObject))
10  .catch((err) => {
11    console.error("OH NO!", err)
12    res.send(err.errors);
13  });
14  res.send(newJuego);
15
16 }
17
18 exports.juego_getById = async (req,res) =>{
19   const{id} = req.params;
20
21   const data = await Juego.findById(id).populate('_usuario', 'nombre');
22
23   res.send(data);
24 }
25

```

**Get:** Aquí se recibe el ID de la tabla y responde con la información de la tabla. Despliega cada dato de la tabla Juego, además del nombre de la tabla Usuario.

**Create:** Recibe el comando del para crear una nueva tabla y responde generando una nueva tabla Juego.

```

25
26 exports.juego_update = async(req,res) => {
27   const { id } = req.params;
28   const { body } = req;
29
30   const juegodb = await Juego.findById(id);
31
32   if(juegodb){
33     const data = await Juego.findOneAndUpdate({_id: id}, body,{returnOriginal:false});
34     res.send({message:"Registro actualizado con exito",data})
35   }
36   else {
37     res.send({message:"El registro que intentas actualizar no existe"})
38   }
39 }
40
41 exports.juego_delete=async(req,res)=>{
42   const {id} = req.params;
43
44   await Juego.deleteOne({_id: id});
45
46   res.send({message:"Registro eliminado"});
47 }

```

**Update:** recibe el ID de una tabla de Juego para buscarla y responde mostrando toda la información de dicha tabla para editarla y actualizar los datos.

**Delete:** Recibe el ID de una tabla Juego para buscarla, y si la encuentra, borra la publicación del encontrado.

## • Tabla Kickstarter

```

2
3 exports.kick_create = async (req,res) =>{
4   const{body} = req;
5
6   const newkick = new Kick(body);
7
8   await newkick.save()
9   .then((newObject)=> console.log("Success!",newObject))
10  .catch((err) => {
11    console.error("OH NO!", err)
12    res.send(err.errors);
13  });
14  res.send(newkick);
15 }
16
17 exports.kick_getById = async (req,res) =>{
18   const{id} = req.params;
19
20   const data = await Kick.findById(id).populate('_juego', 'nombre');
21
22   res.send(data);
23 }
24
25

```

**Get:** Aquí se recibe el ID de la tabla para buscarla y responde con la información de la tabla, además de la llave foránea de la tabla Juego mostrando el nombre de la tabla.

**Create:** Recibe el comando del para crear una nueva tabla y responde generando una nueva tabla Kick.

```

26 exports.kick_update = async(req,res) => {
27   const { id } = req.params;
28   const { body } = req;
29
30   const kickdb = await Kick.findById(id);
31
32   if(kickdb){
33     const data = await Kick.findOneAndUpdate({_id: id}, body,{returnOriginal:false});
34     res.send({message:"Registro actualizado con exito",data})
35   }
36   else {
37     res.send({message:"El registro que intentas actualizar no existe"})
38   }
39 };
40
41 exports.kick_delete = async(req,res)=>{
42   const {id} = req.params;
43
44   await Kick.deleteOne({_id: id});
45
46   res.send({message:"Registro eliminado"});
47 };

```

**Update:** recibe el ID de una tabla Kick para buscarla y responde mostrando toda la información de dicha tabla para editarla y actualizar los datos.

**Delete:** Recibe el ID de una tabla Kick para buscarla, y si la encuentra, borra la publicación del encontrado.

## • Tabla Donacion

```

2
3 exports.Donacion_create = async (req,res) =>{
4   const{body} = req;
5
6   const newDonacion = new Donacion(body);
7
8   await newDonacion.save()
9   .then((newObject)=> console.log("Success!",newObject))
10  .catch((err) => {
11    console.error("OH NO!", err)
12    res.send(err.errors);
13  });
14  res.send(newDonacion);
15 }
16
17
18 exports.donacion_getById = async (req,res) =>{
19   const{id} = req.params;
20
21   const data = await Donacion.findById(id).populate('_usuario', 'nombre');
22
23   res.send(data);
24 }
25

```

**Get:** Aquí se recibe el ID de la tabla para buscarla y responde con la información completa de la tabla, además de todos los datos de la tabla Usuario.

**Create:** Recibe el comando del para crear una nueva tabla y responde generando una nueva tabla Donacion.

```

JS ComentarioController.js 25
JS DonacionController.js 26
JS EmpleadoController.js 27
JS HistorialController.js 28
JS IndieController.js 29
JS JuegoController.js 30
JS KickController.js 31
JS PortafolioController.js 32
JS ProductoController.js 33
> middleware 34
  models 35
    BlogSchema.js 36
    ComentarioSchema.js 37
    connection.js 38
    DonacionSchema.js 39
    EmpleadoSchema.js 40
    HistorialSchema.js 41
    IndieSchema.js 42
    JuegoSchema.js 43
    KickSchema.js 44
    PortafolioSchema.js 45
  46
  47
  48
  49
  50
  51
  52
  53
  54
  55
  56
  57
  58
  59
  60
  61
  62
  63
  64
  65
  66
  67
  68
  69
  70
  71
  72
  73
  74
  75
  76
  77
  78
  79
  80
  81
  82
  83
  84
  85
  86
  87
  88
  89
  90
  91
  92
  93
  94
  95
  96
  97
  98
  99
  100
  101
  102
  103
  104
  105
  106
  107
  108
  109
  110
  111
  112
  113
  114
  115
  116
  117
  118
  119
  120
  121
  122
  123
  124
  125
  126
  127
  128
  129
  130
  131
  132
  133
  134
  135
  136
  137
  138
  139
  140
  141
  142
  143
  144
  145
  146
  147
  148
  149
  150
  151
  152
  153
  154
  155
  156
  157
  158
  159
  160
  161
  162
  163
  164
  165
  166
  167
  168
  169
  170
  171
  172
  173
  174
  175
  176
  177
  178
  179
  180
  181
  182
  183
  184
  185
  186
  187
  188
  189
  190
  191
  192
  193
  194
  195
  196
  197
  198
  199
  200
  201
  202
  203
  204
  205
  206
  207
  208
  209
  210
  211
  212
  213
  214
  215
  216
  217
  218
  219
  220
  221
  222
  223
  224
  225
  226
  227
  228
  229
  230
  231
  232
  233
  234
  235
  236
  237
  238
  239
  240
  241
  242
  243
  244
  245
  246
  247
  248
  249
  250
  251
  252
  253
  254
  255
  256
  257
  258
  259
  260
  261
  262
  263
  264
  265
  266
  267
  268
  269
  270
  271
  272
  273
  274
  275
  276
  277
  278
  279
  280
  281
  282
  283
  284
  285
  286
  287
  288
  289
  290
  291
  292
  293
  294
  295
  296
  297
  298
  299
  300
  301
  302
  303
  304
  305
  306
  307
  308
  309
  310
  311
  312
  313
  314
  315
  316
  317
  318
  319
  320
  321
  322
  323
  324
  325
  326
  327
  328
  329
  330
  331
  332
  333
  334
  335
  336
  337
  338
  339
  340
  341
  342
  343
  344
  345
  346
  347
  348
  349
  350
  351
  352
  353
  354
  355
  356
  357
  358
  359
  360
  361
  362
  363
  364
  365
  366
  367
  368
  369
  370
  371
  372
  373
  374
  375
  376
  377
  378
  379
  380
  381
  382
  383
  384
  385
  386
  387
  388
  389
  390
  391
  392
  393
  394
  395
  396
  397
  398
  399
  400
  401
  402
  403
  404
  405
  406
  407
  408
  409
  410
  411
  412
  413
  414
  415
  416
  417
  418
  419
  420
  421
  422
  423
  424
  425
  426
  427
  428
  429
  430
  431
  432
  433
  434
  435
  436
  437
  438
  439
  440
  441
  442
  443
  444
  445
  446
  447
  448
  449
  450
  451
  452
  453
  454
  455
  456
  457
  458
  459
  460
  461
  462
  463
  464
  465
  466
  467
  468
  469
  470
  471
  472
  473
  474
  475
  476
  477
  478
  479
  480
  481
  482
  483
  484
  485
  486
  487
  488
  489
  490
  491
  492
  493
  494
  495
  496
  497
  498
  499
  500
  501
  502
  503
  504
  505
  506
  507
  508
  509
  510
  511
  512
  513
  514
  515
  516
  517
  518
  519
  520
  521
  522
  523
  524
  525
  526
  527
  528
  529
  530
  531
  532
  533
  534
  535
  536
  537
  538
  539
  540
  541
  542
  543
  544
  545
  546
  547
  548
  549
  550
  551
  552
  553
  554
  555
  556
  557
  558
  559
  560
  561
  562
  563
  564
  565
  566
  567
  568
  569
  570
  571
  572
  573
  574
  575
  576
  577
  578
  579
  580
  581
  582
  583
  584
  585
  586
  587
  588
  589
  590
  591
  592
  593
  594
  595
  596
  597
  598
  599
  600
  601
  602
  603
  604
  605
  606
  607
  608
  609
  610
  611
  612
  613
  614
  615
  616
  617
  618
  619
  620
  621
  622
  623
  624
  625
  626
  627
  628
  629
  630
  631
  632
  633
  634
  635
  636
  637
  638
  639
  640
  641
  642
  643
  644
  645
  646
  647
  648
  649
  650
  651
  652
  653
  654
  655
  656
  657
  658
  659
  660
  661
  662
  663
  664
  665
  666
  667
  668
  669
  670
  671
  672
  673
  674
  675
  676
  677
  678
  679
  680
  681
  682
  683
  684
  685
  686
  687
  688
  689
  690
  691
  692
  693
  694
  695
  696
  697
  698
  699
  700
  701
  702
  703
  704
  705
  706
  707
  708
  709
  710
  711
  712
  713
  714
  715
  716
  717
  718
  719
  720
  721
  722
  723
  724
  725
  726
  727
  728
  729
  730
  731
  732
  733
  734
  735
  736
  737
  738
  739
  740
  741
  742
  743
  744
  745
  746
  747
  748
  749
  750
  751
  752
  753
  754
  755
  756
  757
  758
  759
  760
  761
  762
  763
  764
  765
  766
  767
  768
  769
  770
  771
  772
  773
  774
  775
  776
  777
  778
  779
  780
  781
  782
  783
  784
  785
  786
  787
  788
  789
  790
  791
  792
  793
  794
  795
  796
  797
  798
  799
  800
  801
  802
  803
  804
  805
  806
  807
  808
  809
  810
  811
  812
  813
  814
  815
  816
  817
  818
  819
  820
  821
  822
  823
  824
  825
  826
  827
  828
  829
  830
  831
  832
  833
  834
  835
  836
  837
  838
  839
  840
  841
  842
  843
  844
  845
  846
  847
  848
  849
  850
  851
  852
  853
  854
  855
  856
  857
  858
  859
  860
  861
  862
  863
  864
  865
  866
  867
  868
  869
  870
  871
  872
  873
  874
  875
  876
  877
  878
  879
  880
  881
  882
  883
  884
  885
  886
  887
  888
  889
  890
  891
  892
  893
  894
  895
  896
  897
  898
  899
  900
  901
  902
  903
  904
  905
  906
  907
  908
  909
  910
  911
  912
  913
  914
  915
  916
  917
  918
  919
  920
  921
  922
  923
  924
  925
  926
  927
  928
  929
  930
  931
  932
  933
  934
  935
  936
  937
  938
  939
  940
  941
  942
  943
  944
  945
  946
  947
  948
  949
  950
  951
  952
  953
  954
  955
  956
  957
  958
  959
  960
  961
  962
  963
  964
  965
  966
  967
  968
  969
  970
  971
  972
  973
  974
  975
  976
  977
  978
  979
  980
  981
  982
  983
  984
  985
  986
  987
  988
  989
  990
  991
  992
  993
  994
  995
  996
  997
  998
  999
  1000

```

**Update:** recibe el ID de una tabla Donacion para buscarla y responde mostrando toda la información de dicha tabla para editarla y actualizar los datos.

**Delete:** Recibe el ID de una tabla Donacion para buscarla, y si la encuentra, borra la publicación del encontrado.

## • Tabla Comentario

```

controllers 2
  BlogController.js 3
  ComentarioController.js 4
  DonacionController.js 5
  EmpleadoController.js 6
  HistorialController.js 7
  IndieController.js 8
  JuegoController.js 9
  KickController.js 10
  PortafolioController.js 11
  ProductoController.js 12
  middleware 13
  models 14
    BlogSchema.js 15
    ComentarioSchema.js 16
    connection.js 17
    DonacionSchema.js 18
    EmpleadoSchema.js 19
    HistorialSchema.js 20
    IndieSchema.js 21
    JuegoSchema.js 22
  23
  24
  25
  26
  27
  28
  29
  30
  31
  32
  33
  34
  35
  36
  37
  38
  39
  40
  41
  42
  43
  44
  45
  46
  47
  48
  49
  50
  51
  52
  53
  54
  55
  56
  57
  58
  59
  60
  61
  62
  63
  64
  65
  66
  67
  68
  69
  70
  71
  72
  73
  74
  75
  76
  77
  78
  79
  80
  81
  82
  83
  84
  85
  86
  87
  88
  89
  90
  91
  92
  93
  94
  95
  96
  97
  98
  99
  100
  101
  102
  103
  104
  105
  106
  107
  108
  109
  110
  111
  112
  113
  114
  115
  116
  117
  118
  119
  120
  121
  122
  123
  124
  125
  126
  127
  128
  129
  130
  131
  132
  133
  134
  135
  136
  137
  138
  139
  140
  141
  142
  143
  144
  145
  146
  147
  148
  149
  150
  151
  152
  153
  154
  155
  156
  157
  158
  159
  160
  161
  162
  163
  164
  165
  166
  167
  168
  169
  170
  171
  172
  173
  174
  175
  176
  177
  178
  179
  180
  181
  182
  183
  184
  185
  186
  187
  188
  189
  190
  191
  192
  193
  194
  195
  196
  197
  198
  199
  200
  201
  202
  203
  204
  205
  206
  207
  208
  209
  210
  211
  212
  213
  214
  215
  216
  217
  218
  219
  220
  221
  222
  223
  224
  225
  226
  227
  228
  229
  230
  231
  232
  233
  234
  235
  236
  237
  238
  239
  240
  241
  242
  243
  244
  245
  246
  247
  248
  249
  250
  251
  252
  253
  254
  255
  256
  257
  258
  259
  260
  261
  262
  263
  264
  265
  266
  267
  268
  269
  270
  271
  272
  273
  274
  275
  276
  277
  278
  279
  280
  281
  282
  283
  284
  285
  286
  287
  288
  289
  290
  291
  292
  293
  294
  295
  296
  297
  298
  299
  300
  301
  302
  303
  304
  305
  306
  307
  308
  309
  310
  311
  312
  313
  314
  315
  316
  317
  318
  319
  320
  321
  322
  323
  324
  325
  326
  327
  328
  329
  330
  331
  332
  333
  334
  335
  336
  337
  338
  339
  340
  341
  342
  343
  344
  345
  346
  347
  348
  349
  350
  351
  352
  353
  354
  355
  356
  357
  358
  359
  360
  361
  362
  363
  364
  365
  366
  367
  368
  369
  370
  371
  372
  373
  374
  375
  376
  377
  378
  379
  380
  381
  382
  383
  384
  385
  386
  387
  388
  389
  390
  391
  392
  393
  394
  395
  396
  397
  398
  399
  400
  401
  402
  403
  404
  405
  406
  407
  408
  409
  410
  411
  412
  413
  414
  415
  416
  417
  418
  419
  420
  421
  422
  423
  424
  425
  426
  427
  428
  429
  430
  431
  432
  433
  434
  435
  436
  437
  438
  439
  440
  441
  442
  443
  444
  445
  446
  447
  448
  449
  450
  451
  452
  453
  454
  455
  456
  457
  458
  459
  460
  461
  462
  463
  464
  465
  466
  467
  468
  469
  470
  471
  472
  473
  474
  475
  476
  477
  478
  479
  480
  481
  482
  483
  484
  485
  486
  487
  488
  489
  490
  491
  492
  493
  494
  495
  496
  497
  498
  499
  500
  501
  502
  503
  504
  505
  506
  507
  508
  509
  510
  511
  512
  513
  514
  515
  516
  517
  518
  519
  520
  521
  522
  523
  524
  525
  526
  527
  528
  529
  530
  531
  532
  533
  534
  535
  536
  537
  538
  539
  540
  541
  542
  543
  544
  545
  546
  547
  548
  549
  550
  551
  552
  553
  554
  555
  556
  557
  558
  559
  560
  561
  562
  563
  564
  565
  566
  567
  568
  569
  570
  571
  572
  573
  574
  575
  576
  577
  578
  579
  580
  581
  582
  583
  584
  585
  586
  587
  588
  589
  590
  591
  592
  593
  594
  595
  596
  597
  598
  599
  600
  601
  602
  603
  604
  605
  606
  607
  608
  609
  610
  611
  612
  613
  614
  615
  616
  617
  618
  619
  620
  621
  622
  623
  624
  625
  626
  627
  628
  629
  630
  631
  632
  633
  634
  635
  636
  637
  638
  639
  640
  641
  642
  643
  644
  645
  646
  647
  648
  649
  650
  651
  652
  653
  654
  655
  656
  657
  658
  659
  660
  661
  662
  663
  664
  665
  666
  667
  668
  669
  670
  671
  672
  673
  674
  675
  676
  677
  678
  679
  680
  681
  682
  683
  684
  685
  686
  687
  688
  689
  690
  691
  692
  693
  694
  695
  696
  697
  698
  699
  700
  701
  702
  703
  704
  705
  706
  707
  708
  709
  710
  711
  712
  713
  714
  715
  716
  717
  718
  719
  720
  721
  722
  723
  724
  725
  726
  727
  728
  729
  730
  731
  732
  733
  734
  735
  736
  737
  738
  739
  740
  741
  742
  743
  744
  745
  746
  747
  748
  749
  750
  751
  752
  753
  754
  755
  756
  757
  758
  759
  760
  761
  762
  763
  764
  765
  766
  767
  768
  769
  770
  771
  772
  773
  774
  775
  776
  777
  778
  779
  780
  781
  782
  783
  784
  785
  786
  787
  788
  789
  790
  791
  792
  793
  794
  795
  796
  797
  798
  799
  800
  801
  802
  803
  804
  805
  806
  807
  808
  809
  810
  811
  812
  813
  814
  815
  816
  817
  818
  819
  820
  821
  822
  823
  824
  825
  826
  827
  828
  829
  830
  831
  832
  833
  834
  835
  836
  837
  838
  839
  840
  841
  842
  843
  844
  845
  846
  847
  848
  849
  850
  851
  852
  853
  854
  855
  856
  857
  858
  859
  860
  861
  862
  863
  864
  865
  866
  867
  868
  869
  870
  871
  872
  873
  874
  875
  876
  877
  878
  879
  880
  881
  882
  883
  884
  885
  886
  887
  888
  889
  890
  891
  892
  893
  894
  895
  896
  897
  898
  899
  900
  901
  902
  903
  904
  905
  906
  907
  908
  909
  910
  911
  912
  913
  914
  915
  916
  917
  918
  919
  920
  921
  922
  923
  924
  925
  926
  927
  928
  929
  930
  931
  932
  933
  934
  935
  936
  937
  938
  939
  940
  941
  942
  943
  944
  945
  946
  947
  948
  949
  950
  951
  952
  953
  954
  955
  956
  957
  958
  959
  960
  961
  962
  963
  964
  965
  966
  967
  968
  969
  970
  971
  972
  973
  974
  975
  976
  977
  978
  979
  980
  981
  982
  983
  984
  985
  986
  987
  988
  989
  990
  991
  992
  993
  994
  995
  996
  997
  998
  999
  1000

```

**Get:** Aquí se recibe el ID de la tabla para buscarla y responde con la información de la tabla, además de la llave foránea de la tabla Juego mostrando el nombre de la tabla.

**Create:** Recibe el comando del para crear una nueva tabla y responde generando una nueva tabla Comentario.

```

27
28 exports.Comentario_update = async(req,res) => {
29   const { id } = req.params;
30   const { body } = req;
31
32   const comentarioDb = await Comentario.findById(id);
33
34   if(comentarioDb){
35     const data = await Comentario.findOneAndUpdate({_id: id}, body,{returnOriginal:false});
36     res.send({message:"Registro actualizado con exito",data})
37   }
38   else {
39     res.send({message:"El registro que intentas actualizar no existe"})
40   }
41 };
42
43 exports.Comentario_delete = async(req,res)=>{
44   const {id} = req.params;
45
46   await Comentario.deleteOne({_id: id});
47
48   res.send({message:"Registro eliminado"});
49 };
50

```

**Update:** recibe el ID de una tabla Comentario para buscarla y responde mostrando toda la información de dicha tabla para editarla y actualizar los datos.

**Delete:** Recibe el ID de una tabla Comentario para buscarla, y si la encuentra, borra la publicación del encontrado.

## • Tabla Empleo

```

2
3 exports.Empleo_create = async (req,res) =>{
4   const{body} = req;
5
6   const newEmpleo = new Empleo(body);
7
8   await newEmpleo.save()
9   .then((newObject)=> console.log("Success!",newObject))
10  .catch((err) => {
11    console.error("OH NO!", err)
12    res.send(err.errors);
13  });
14  res.send(newEmpleo);
15
16 }
17
18 exports.Empleo_getById = async (req,res) =>{
19   const{id} = req.params;
20
21   const data = await Empleo.findById(id).populate('_usuario', 'nombre');
22
23   res.send(data);
24 }
25

```

**Get:** Aquí se recibe el ID de la tabla para buscarla y responde con la información de la tabla, además de la llave foránea de la tabla Usuario mostrando el nombre de la tabla.

**Create:** Recibe el comando del para crear una nueva tabla y responde generando una nueva tabla Empleo.

```

26 exports.Empleado_update = async(req,res) => {
27   const { id } = req.params;
28   const { body } = req;
29
30   const empleodb = await Empleo.findById(id);
31
32   if(empleodb){
33     const data = await Empleo.findOneAndUpdate({_id: id}, body,{returnOriginal:false});
34     res.send({message:"Registro actualizado con exito",data})
35   }
36   else {
37     res.send({message:"El registro que intentas actualizar no existe"})
38   }
39 };
40
41 exports.Empleado_delete = async(req,res)=>{
42   const {id} = req.params;
43
44   await Empleo.deleteOne({_id: id});
45
46   res.send({message:"Registro eliminado"});
47 };

```

**Update:** recibe el ID de una tabla Empleo para buscarla y responde mostrando toda la información de dicha tabla para editarla y actualizar los datos.

**Delete:** Recibe el ID de una tabla Empleo para buscarla, y si la encuentra, borra la publicación del encontrado.

## • Tabla Producto

```

3 exports.Producto_create = async (req,res) =>{
4   const{body} = req;
5
6   const newProducto = new Producto(body);
7
8   await newProducto.save()
9   .then((newObject)=> console.log("Success!",newObject))
10  .catch((err) => {
11    console.error("OH NO!", err)
12    res.send(err.errors);
13  });
14  res.send(newProducto);
15 }
16
17
18 exports.Producto_getById = async (req,res) =>{
19   const{id} = req.params;
20
21   const data = await Producto.findById(id).populate('_usuario', 'nombre');
22
23   res.send(data);
24 }
25

```

**Get:** Aquí se recibe el ID de la tabla para buscarla y responde con la información de la tabla, además de la llave foránea de la tabla Usuario mostrando el nombre de la tabla.

**Create:** Recibe el comando del para crear una nueva tabla y responde generando una nueva tabla Producto.



```
25
26 exports.Producto_update = async(req,res) => {
27   const { id } = req.params;
28   const { body } = req;
29
30   const productodb = await Producto.findById(id);
31
32   if(productodb){
33     const data = await Producto.findOneAndUpdate({_id: id}, body,{returnOriginal:false});
34     res.send({message:"Registro actualizado con exito",data})
35   }
36   else {
37     res.send({message:"El registro que intentas actualizar no existe"})
38   }
39 };
40
41 exports.Producto_delete = async(req,res)=>{
42   const {id} = req.params;
43
44   await Producto.deleteOne({_id: id});
45
46   res.send({message:"Registro eliminado"});
47 };
```

**Update:** recibe el ID de una tabla Producto para buscarla y responde mostrando toda la información de dicha tabla para editarla y actualizar los datos.

**Delete:** Recibe el ID de una tabla Producto para buscarla, y si la encuentra, borra la publicación del encontrado.

- Tabla Blog

```
2
3 exports.Blog_create = async (req,res) =>{
4   const{body} = req;
5
6   const newBlog = new Blog(body);
7
8   await newBlog.save()
9   .then((newObject)=> console.log("Success!",newObject))
10  .catch((err) => {
11    console.error("OH NO!", err)
12    res.send(err.errors);
13  });
14  res.send(newBlog);
15
16 }
17
18 exports.Blog_getById = async (req,res) =>{
19   const{id} = req.params;
20
21   const data = await Blog.findById(id).populate('_usuario', 'nombre');
22
23   res.send(data);
24 }
25
```

**Get:** Aquí se recibe el ID de la tabla para buscarla y responde con la información de la tabla, además de la llave foránea de la tabla Usuario mostrando el nombre de la tabla.

**Create:** Recibe el comando del para crear una nueva tabla y responde generando una nueva tabla Blog.

```

25
26 exports.Blog_update = async(req,res) => {
27   const { id } = req.params;
28   const { body } = req;
29
30   const blogdb = await Blog.findById(id);
31
32   if(blogdb){
33     const data = await Blog.findOneAndUpdate({_id: id}, body,{returnOriginal:false});
34     res.send({message:"Registro actualizado con exito",data})
35   }
36   else {
37     res.send({message:"El registro que intentas actualizar no existe"})
38   }
39 };
40
41 exports.Blog_delete = async(req,res)=>{
42   const {id} = req.params;
43
44   await Blog.deleteOne({_id: id});
45
46   res.send({message:"Registro eliminado"});
47 };

```

**Update:** recibe el ID de una tabla Blog para buscarla y responde mostrando toda la información de dicha tabla para editarla y actualizar los datos.

**Delete:** Recibe el ID de una tabla Blog para buscarla, y si la encuentra, borra la publicación del encontrado.

## • Tabla Portafolio

```

2
3 exports.Portafolio_create = async (req,res) =>{
4   const{body} = req;
5
6   const newPortafolio = new Portafolio(body);
7
8   await newPortafolio.save()
9   .then((newObject)=> console.log("Success!",newObject))
10  .catch((err) => {
11    console.error("OH NO!", err)
12    res.send(err.errors);
13  });
14  res.send(newPortafolio);
15
16 }
17
18 exports.Portafolio_getById = async (req,res) =>{
19   const{id} = req.params;
20
21   const data = await Portafolio.findById(id).populate('_usuario', 'nombre');
22
23   res.send(data);
24 }
25

```

**Get:** Aquí se recibe el ID de la tabla para buscarla y responde con la información de la tabla, además de la llave foránea de la tabla Usuario mostrando el nombre de la tabla.

**Create:** Recibe el comando del para crear una nueva tabla y responde generando una nueva tabla Portafolio.

```

25
26 JS DonacionController.js
27 JS EmpleoController.js
28 JS HistorialController.js
29 JS IndieController.js
30 JS JuegoController.js
31 JS KickController.js
32 JS PortafolioController.js
33 JS ProductoController.js
34 > middleware
35 > models
36 > node_modules
37 > routes
38 JS index.js
39 {} package-lock.json
40 {} package.json
41
42 exports.Portafolio_update = async(req,res) => {
43   const { id } = req.params;
44   const { body } = req;
45
46   const portafoliodb = await Portafolio.findById(id);
47
48   if(portafoliodb){
49     const data = await Portafolio.findOneAndUpdate({ _id: id }, body,{returnOriginal:false});
50     res.send({message:"Registro actualizado con exito",data})
51   }
52   else {
53     res.send({message:"El registro que intentas actualizar no existe"})
54   }
55 }
56
57 exports.Portafolio_delete = async(req,res)=>{
58   const {id} = req.params;
59
60   await Portafolio.deleteOne({_id: id});
61
62   res.send({message:"Registro eliminado"});
63 }
64

```

**Update:** recibe el ID de una tabla Portafolio para buscarla y responde mostrando toda la información de dicha tabla para editarla y actualizar los datos.

**Delete:** Recibe el ID de una tabla Portafolio para buscarla, y si la encuentra, borra la publicación del encontrado.

## • Tabla Historial

```

4
5 controllers
6 JS BlogController.js
7 JS ComentarioController.js
8 JS DonacionController.js
9 JS EmpleoController.js
10 JS HistorialController.js
11 JS IndieController.js
12 JS JuegoController.js
13 JS KickController.js
14 JS PortafolioController.js
15 JS ProductoController.js
16 > middleware
17 > models
18 > node_modules
19 > routes
20 JS index.js
21 {} package-lock.json
22 {} package.json
23
24 exports.Historial_create = async (req,res) =>{
25   const{body} = req;
26
27   const newHistorial = new Historial(body);
28
29   await newHistorial.save()
30   .then((newObject)=> console.log("Success!",newObject))
31   .catch((err) => {
32     console.error("OH NO!", err)
33     res.send(err.errors);
34   });
35   res.send(newHistorial);
36 }
37
38 exports.Historial_getById = async (req,res) =>{
39   const{id} = req.params;
40
41   const data = await Historial.findById(id).populate('_producto', 'nombre descripcion _usuario');
42   res.send(data);
43 }
44

```

**Get:** Recibe el ID de la tabla Historial y responde mostrando la información de la tabla, que son las llaves foráneas de la tabla Producto mostrando el nombre del producto y su descripción y el nombre de la tabla Usuario.

**Create:** Recibe el comando del para crear una nueva tabla y responde generando una nueva tabla Historial.

```
JS DonacionController.js
JS EmpleoController.js
JS HistorialController.js
JS IndieController.js
JS JuegoController.js
JS KickController.js
JS PortafolioController.js
JS ProductoController.js
> middleware
> models
> node_modules
> routes
JS index.js
{} package-lock.json
{} package.json

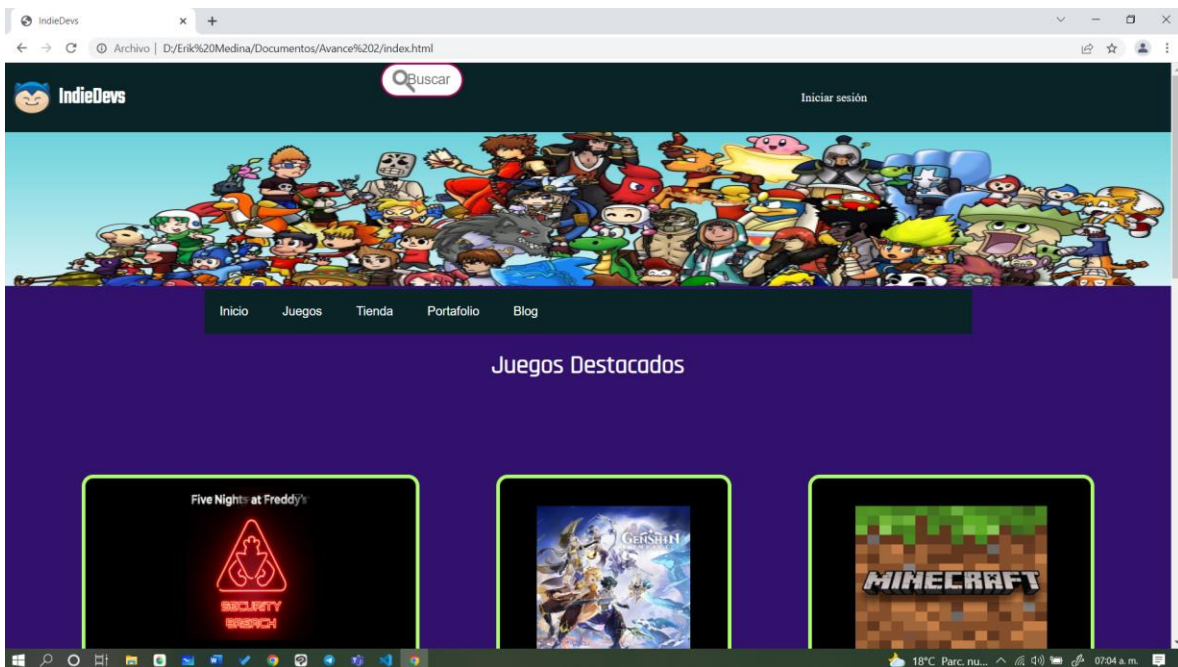
25
26 exports.Historial_update = async(req,res) => {
27   const { id } = req.params;
28   const { body } = req;
29
30   const historialdb = await Historial.findById(id);
31
32   if(historialdb){
33     const data = await Historial.findOneAndUpdate({_id: id}, body,{returnOriginal:false});
34     res.send({message:"Registro actualizado con exito",data})
35   }
36   else {
37     res.send({message:"El registro que intentas actualizar no existe"})
38   }
39 };
40
41 exports.Historial_delete = async(req,res)=>{
42   const {id} = req.params;
43
44   await Historial.deleteOne({_id: id});
45
46   res.send({message:"Registro eliminado"});
47 };
```

**Update:** recibe el ID de una tabla Historial para buscarla y responde mostrando toda la información de dicha tabla para editarla y actualizar los datos.

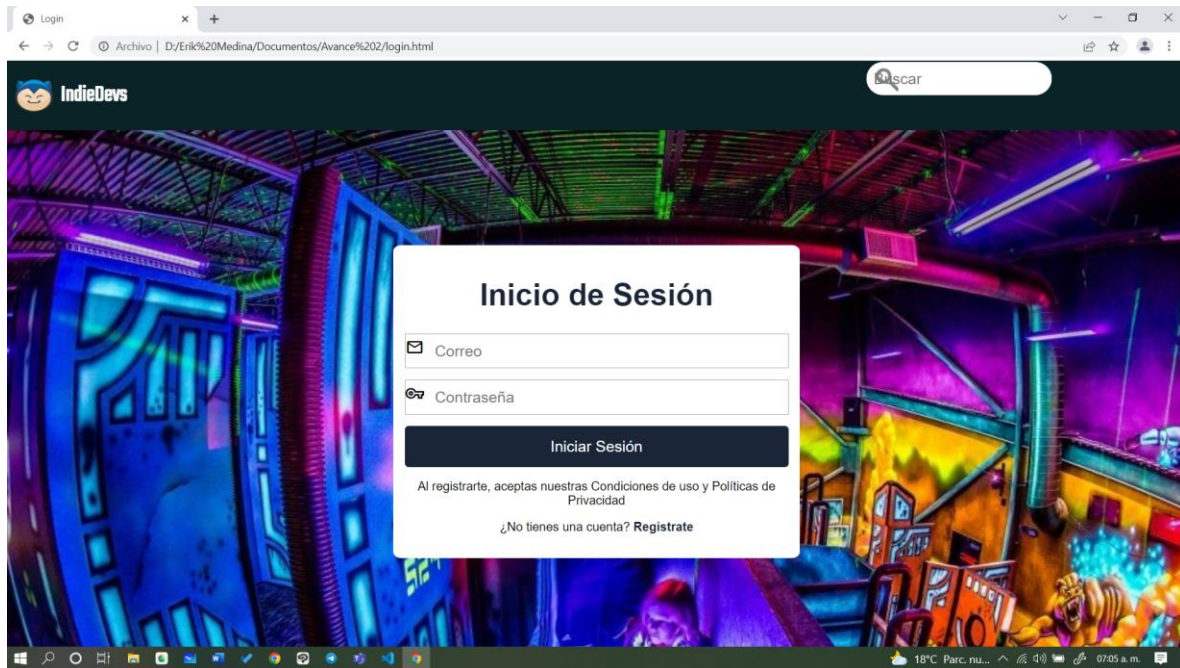
**Delete:** Recibe el ID de una tabla Historial para buscarla, y si la encuentra, borra la publicación del encontrado.

No consideramos necesarios los endpoints Update y Delete, pues no es posible o necesario editar o borrar el historial de descargas.

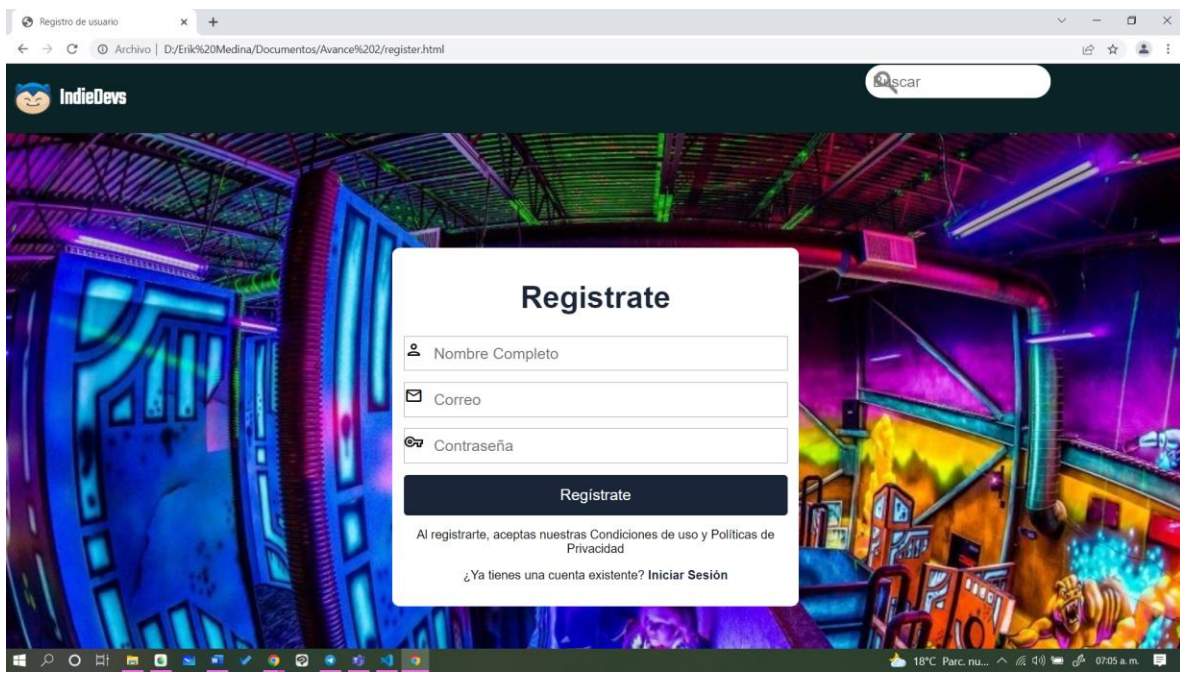
- Ventanas



En la página principal se encuentran los juegos destacados, los mejores valorados y los más recientes.

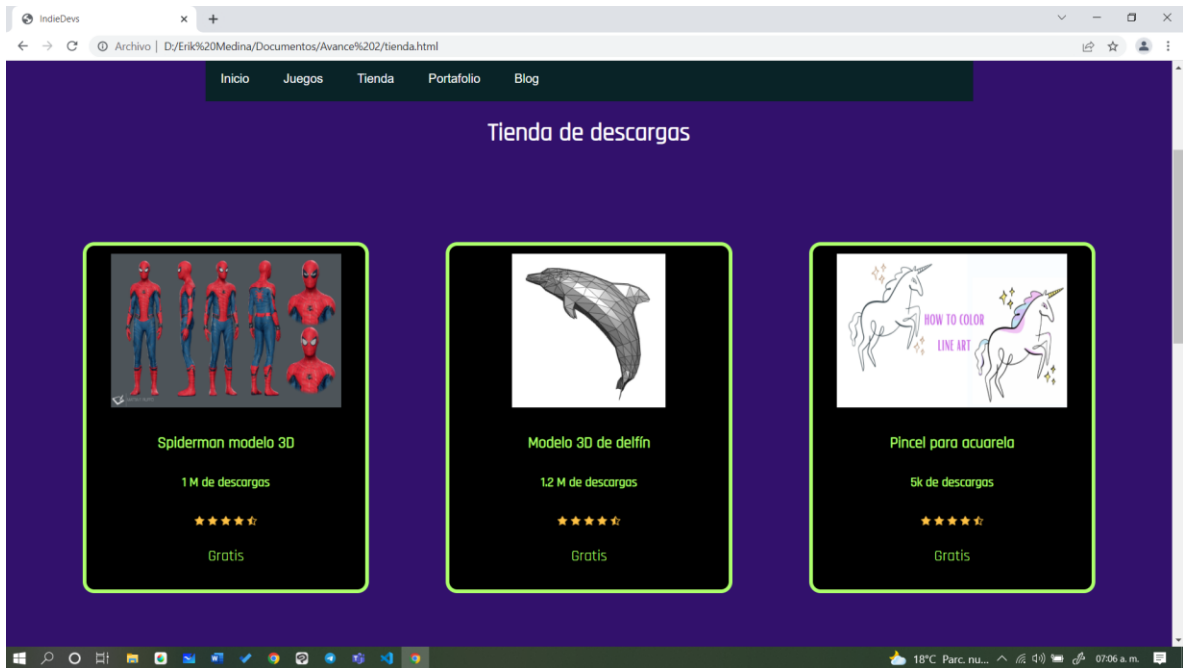


En la página del login podrás iniciar sesión.

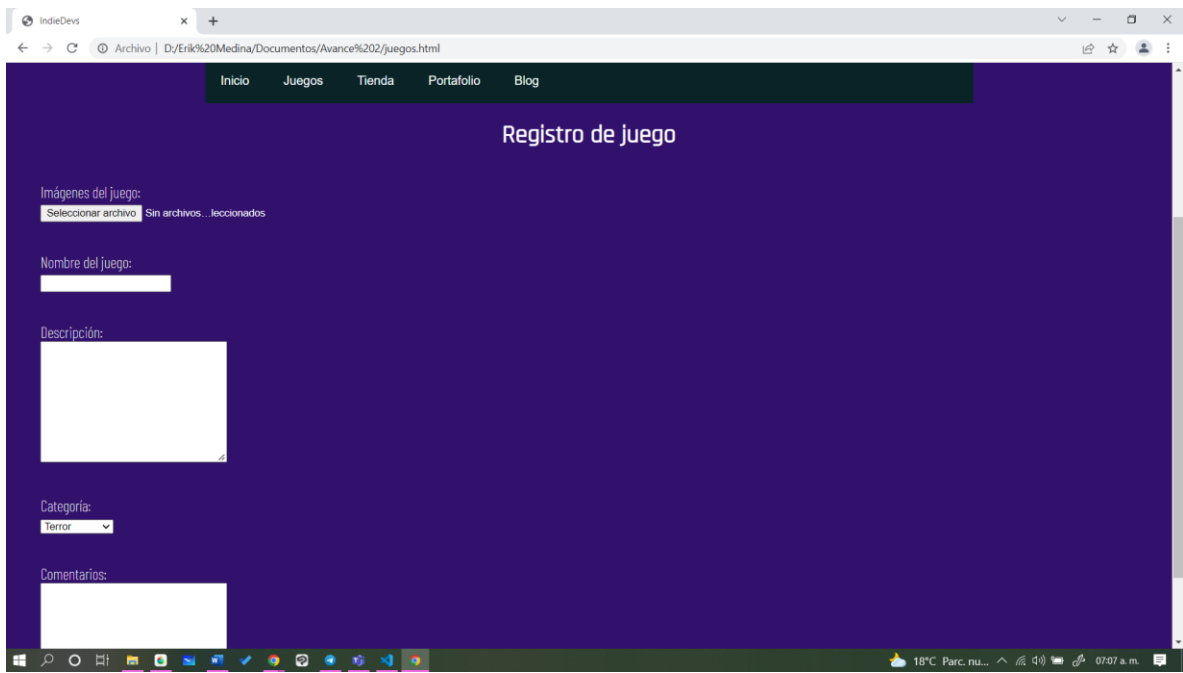


En la página del registro podrás registrarte al ingresar los datos solicitados.





En la página de la tienda podrás encontrar varios tipos de productos, ya sean modelos, brochas, assets, texturas, entre otras cosas de forma gratuita, estos serán hechos por la comunidad para la comunidad.



Después tenemos la página de crear juego, en esta básicamente agregas tu propuesta o tu juego en desarrollo, agregas imágenes, descripción, tu propuesta, mecánicas del juego, entre otras cosas, de igual manera los usuarios pueden comentar y dar su opinión respecto a tu juego, y si lo desean pueden donante a través del kickstarter.

- Colección de Postman

<https://drive.google.com/file/d/1AmoLxF-0CBT-yRTh37nltob-uGuD8qPb/view?usp=sharing>

<https://www.getpostman.com/collections/0b64fcf7ab83459980d0>