مدرسه (SQL)

کد شما باید روی MySQL قابل اجرا باشد.

مدرسه‌ای اطلاعات دانش‌آموزان خود را در یک جدولی در پایگاه داده قرار داده است که اطلاعات به صورت جدولی به نام `STUDNETS` می‌باشد و نام ستون‌های آن `SID` , `SNAME` , `GRADE` , `AGE` می‌باشد.ستون های `SID` ، `GRADE` و `AGE` از نوع `int` و ستون `SNAME` از نوع `varchar` میباشد.

مدیر مدرسه می‌خواهد کارهای زیر روی پایگاه داده انجام شود. از شما خواسته شده این کارها را برای مدرسه انجام دهید.

لیست کارها

۱. اضافه کردن اطلاعات زیر به پایگاه داده

SID	SNAME	GRADE	AGE
101	Mohammad	13	24
102	Ahmad	16	26
103	Saeed	19	31
104	Mina	19	34
105	Jafar	18	23
106	Ahmad	16	26

۲. به روز رسانی جدول به صورتی که کسانی که نامشان `Ahmad` است مقدار سن آنها یکی اضافه شود.

۳. تمام دانش‌آموزانی که اسم آنها `Mostafa` است حذف شوند.

۴. یافتن نام تمام دانش‌آموزانی که معدل بالاتری از تمام دانش‌آموزانی دارند که سن آنها کمتر از ۲۸ سال باشد.

روش پیاده‌سازی

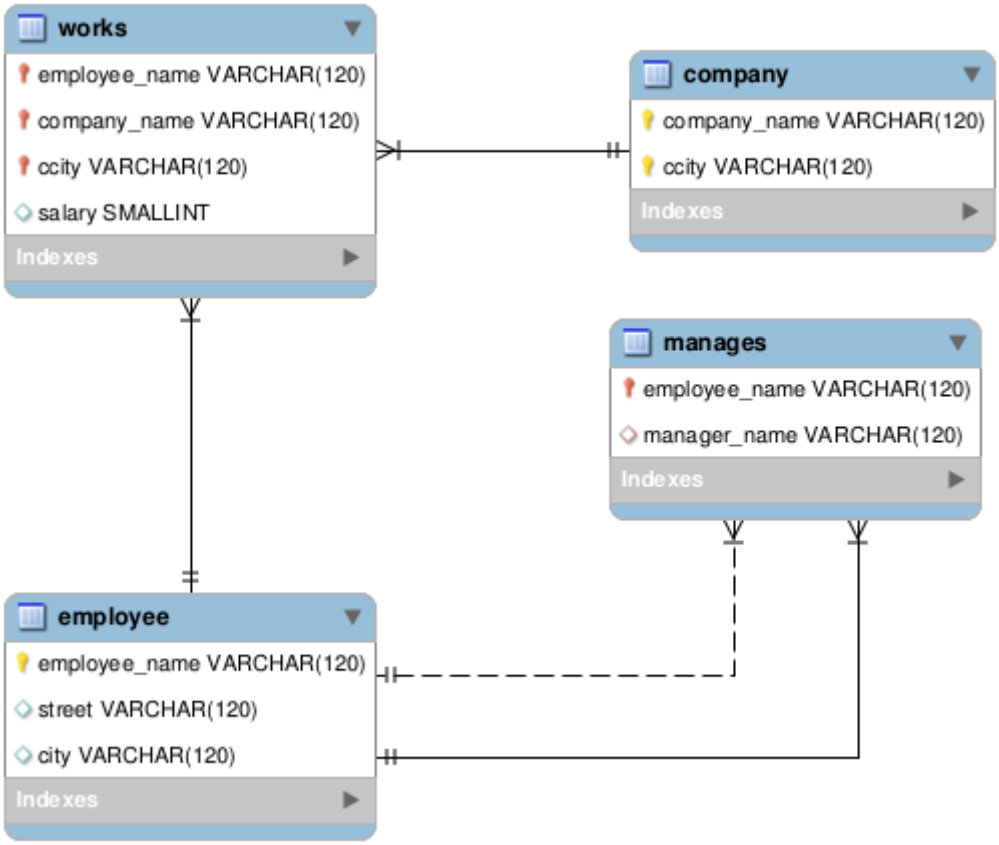
در یک فایل با نام code.sql کد خود را قرار دهید و آن را فشرده (zip) کنید و در سایت بارگذاری نمایید. کد شما باید به صورت زیر باشد(نام فایل zip مهم نیست).

```
-- Section1
   your first query here
-- Section2
   your second query here
-- Section3
   your third query here
-- Section4
   your fourth query here
```

شرکت (SQL)

کد شما باید روی MySQL قابل اجرا باشد.

پایگاه داده ای به صورت زیر در اختیار داریم:



شمای پایگاه داده به صورت زیر است:

```
CREATE TABLE employee (
  employee_name VARCHAR (120) UNIQUE PRIMARY KEY ,
  street VARCHAR (120),
  city VARCHAR (120)
);
CREATE TABLE company (
  company_name VARCHAR (120),
  ccity VARCHAR (120),
  PRIMARY KEY (company_name, ccity)
);
```

```
CREATE TABLE works (
  employee_name VARCHAR (120),
  company_name VARCHAR (120),
  ccity VARCHAR (120),
  salary SMALLINT ,
  PRIMARY KEY (employee_name, company_name, ccity),
  FOREIGN KEY (employee_name) REFERENCES employee(employee_name)
    ON DELETE CASCADE
    ON UPDATE CASCADE ,
  FOREIGN KEY (company_name, ccity) REFERENCES company(company_name, ccity)
    ON DELETE CASCADE
    on UPDATE CASCADE
);

CREATE TABLE manages (
  employee_name VARCHAR (120) UNIQUE PRIMARY KEY ,
  manager_name VARCHAR (120),
  FOREIGN KEY (employee_name) REFERENCES employee(employee_name)
    ON DELETE CASCADE
    on UPDATE CASCADE,
  FOREIGN KEY (manager_name) REFERENCES employee(employee_name)
    ON DELETE CASCADE
    on UPDATE CASCADE
);
```

دستورات sql خود را برای پاسخ به قسمت‌های زیر بنویسید.

۱. نام و شهر شرکت‌هایی را بیابید که حقوق کارمندان آن ها بطور متوسط از میانگین حقوق کارمندان `Walker Group` بیشتر است.
۲. نام و شهر شرکت هایی که دارای بیشترین تعداد کارمندان است.
۳. نام کارمندانی که از کارمندان `Walker Group` درآمد بیشتری دارند را بیابید. نام هر کارمند را تنها یکبار در خروجی قرار دهید. (در این سوال فرض کنید یک کارمند میتواند در چندین شرکت/نهاد مشغول به کار و حقوق بگیر باشد)

روش پیاده‌سازی

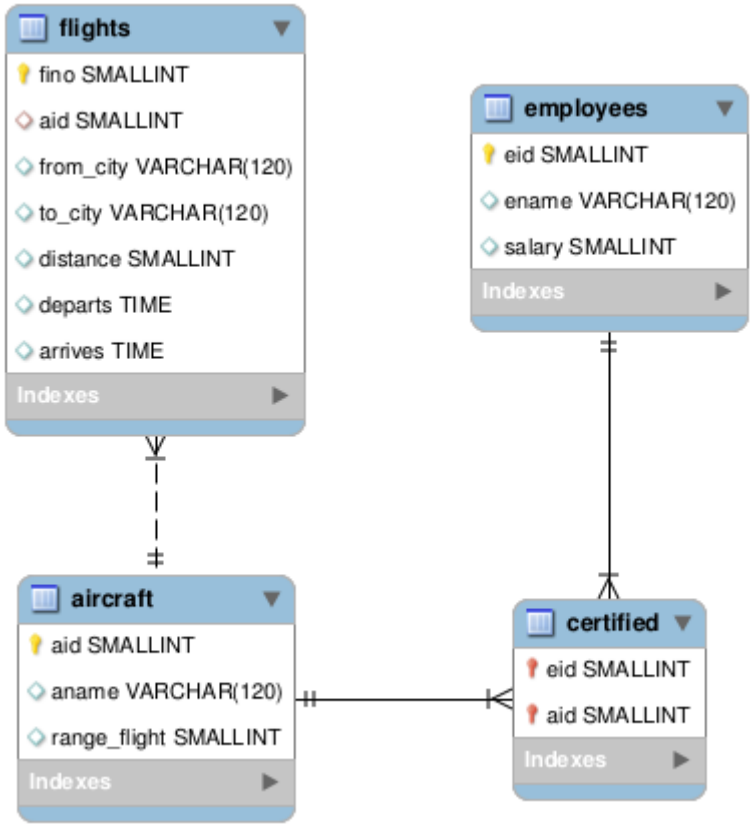
در یک فایل با نام `code.sql` کد خود را قرار دهید و آن را فشرده (`zip`) کنید و در سایت بارگذاری نمایید. کد شما باید به صورت زیر باشد(نام فایل zip مهم نیست).

```
-- Section1
   your first query here
-- Section2
   your second query here
-- Section3
   your third query here
```

هواپیما (SQL)

کد شما باید روی MySQL قابل اجرا باشد.

پایگاه داده ای با ERD زیر در اختیار داریم:



دستورات sql خود را برای پاسخ به قسمت‌های زیر بنویسید.

۱. شماره تمام پروازهایی که میتوانند توسط خلبانانی که حقوق بیشتر از `2000` است، خلبانی شوند.
۲. شماره کارمندانی که بیشترین حقوق را دریافت می‌کنند.
۳. شماره تمام شرکت‌های هواپیمایی که میتوانند به صورت مستقیم بین `Kubstad` و `South Steve` پرواز کنند.
۴. نام تمام خلبانانی(به صورت یکتا) که قابلیت پرواز با برخی از هواپیماهای `Boyer Ltd` را دارند.
۵. نام تمام خلبانانی(به صورت یکتا) که مجوز پرواز دقیقا با `5` هواپیما را دارند.

روش پیاده‌سازی

در یک فایل با نام code.sql کد خود را قرار دهید و آن را فشرده (zip) کنید و در سایت بارگذاری نمایید. کد شما باید به صورت زیر باشد(نام فایل zip مهم نیست).

```
-- Section1
   your first query here
-- Section2
   your second query here
-- Section3
   your third query here
-- Section4
   your fourth query here
-- Section5
   your fifth query here
```

زمان‌بندی (Linux)

همان‌طور که می‌دانید، یک عبارت Cron در لینوکس به صورت زیر است:

```
┌──────────── minute (0 - 59)
│ ┌──────────── hour (0 - 23)
│ │ ┌──────────── day of month (1 - 31)
│ │ │ ┌──────────── month (1 - 12)
│ │ │ │ ┌──────────── day of week (0 - 6) (Sunday to Saturday;
│ │ │ │ │                                       7 is also Sunday)
│ │ │ │ │
│ │ │ │ │
* * * * *  command to execute
```

مثلاً اگر بخواهیم دستور `some command` هر یک ساعت (در دقیقه ۱۵ هر ساعت) اجرا شود، عبارت Cron معادل آن عبارت است از:

```
15 * * * * some command
```

از شما خواسته شده تا برای هریک از موارد زیر یک عبارت Cron بنویسید.

۱. می‌خواهیم دستور `command1` هر پنج دقیقه اجرا شود.
۲. می‌خواهیم دستور `command2` هر روز ساعت ۲:۳۰ اجرا شود.
۳. می‌خواهیم دستور `command3` در روزهای ۱ ام، ۱۰ ام و ۲۰ ام هر ماه در ساعت ۷ صبح اجرا شود.
۴. می‌خواهیم دستور `command4` در روزهای جمعه در ساعت ۸ صبح و ۸ شب اجرا شود.

این ۴ cron را به ترتیب در ۴ خط اول یک فایل به نام `crons.txt` بنویسید. سپس این فایل را Zip کنید و به عنوان پاسخ ارسال کنید. دقت کنید که این فایل باید در ریشه فایل Zip باشد (در هیچ پوشه‌ای نباشد).

تست‌های پارسا (Linux)

پارسا هفته پیش به همراه مهدی در Quera یک مسابقه برنامه‌نویسی برگزار کرد. مهدی تست‌های یکی از سؤالات را به پارسا داد تا در سایت قرار دهد. تست‌ها به صورت زیر بودند:

```
tests
├── input
│   ├── 1
│   ├── 10
│   ├── 180
│   ├── 5
│   ├── 68
│   ├── 9
│   └── 900
└── output
    ├── 1
    ├── 10
    ├── 180
    ├── 5
    ├── 68
    ├── 9
    └── 900
```

اما با توجه به این که فرمت تست‌ها در Quera به این شکل نیست مهدی مجبور شد تا نام پوشه‌ها و تست‌ها را به ترتیب نام به این شکل تغییر دهد:

```
input/1    ->  in/input1.txt
input/5    ->  in/input2.txt
input/9    ->  in/input3.txt
input/10   ->  in/input4.txt
input/68   ->  in/input5.txt
input/180  ->  in/input6.txt
input/900  ->  in/input7.txt

output/1    ->  out/output1.txt
output/5    ->  out/output2.txt
output/9    ->  out/output3.txt
output/10   ->  out/output4.txt
```

```
output/68   ->  out/output5.txt
output/180  ->  out/output6.txt
output/900  ->  out/output7.txt
```

شکل نهایی تست‌ها:

```
tests
├── in
│   ├── input1.txt
│   ├── input2.txt
│   ├── input3.txt
│   ├── input4.txt
│   ├── input5.txt
│   ├── input6.txt
│   └── input7.txt
└── out
    ├── output1.txt
    ├── output2.txt
    ├── output3.txt
    ├── output4.txt
    ├── output5.txt
    ├── output6.txt
    └── output7.txt
```

به دلیل این که تست‌ها به ترتیب سختی مرتب شده‌اند، پارسا باید دقت می‌کرد که نام تست‌ها را به ترتیب درست قرار دهد.

پارسا هفته قبل این کار را به صورت دستی انجام داد و نام آن ۷ تست را درست کرد. ولی مهدی برای مسابقه این هفته، صدها تست به پارسا داده است که نام آن‌ها مانند هفته قبل نادرست است و نیاز به تغییر نام دارند. با توجه به این که دیگر روش دستی جوابگو نیست، پارسا از شما کمک خواسته است.

شما باید یک اسکریپت Bash به نام `fix.sh` بنویسد که آن را در پوشه tests قرار دهیم و با اجرای آن، نام پوشه‌ها و تست‌ها درست شود.

فایل `fix.sh` را به صورت یک فایل Zip فشرده کنید و به عنوان پاسخ آپلود کنید. دقت کنید که این فایل باید در ریشه فایل Zip باشد (در هیچ پوشه‌ای نباشد).

پردازش لاگ (Linux)

این سؤال ۶ بخش دارد. به هر تعداد از بخش‌ها که پاسخ دهید به همان میزان نمره خواهید گرفت.

به شما یک فایل لاگ (logfile) داده شده است و شما باید اطلاعاتی از این فایل استخراج کنید. در این فایل یک تعداد event به مرور زمان ثبت شده‌اند. هر event در یک خط نوشته شده است و ساختار زیر را دارد:

```
[DATETIME] - LEVEL - FILENAME - LINENUMBER - MESSAGE
```

مثال:

```
[2017-05-15T17:27:44.018903] - WARNING - apps/dashboard/plugins.py - 664 - too
many requests
```

در این ساختار، DATETIME زمان وقوع رخداد است. LEVEL سطح رخداد است. FILENAME آدرس فایلی است که رخداد در آن اتفاق افتاده است. LINENUMBER شماره خطی از فایل FILENAME است که در آن خط این رخداد نوشته شده است. MESSAGE نیز توضیح کوتاهی در مورد رخداد می‌دهد.

۶ سطح برای رخداد وجود دارد که LEVEL می‌تواند یکی از آن‌ها باشد:

```
CRITICAL
ERROR
WARNING
INFO
DEBUG
NOTSET
```

یک نمونه از فایل لاگ:

```
[2017-05-15T17:27:44.018903] - WARNING - apps/dashboard/plugins.py - 664 - 7
plugins outdated
[2017-05-15T17:27:52.994563] - CRITICAL - apps/blog/admin.py - 123 - app
crashed
[2017-05-15T17:29:19.159658] - DEBUG - apps/dashboard/models.py - 199 - models
initialized
```

```
[2017-05-15T17:29:30.625481] - NOTSET - apps/utils/decorators.py - 536 - user type was not set
[2017-05-15T17:29:47.369583] - WARNING - apps/blog/forms.py - 90 - too many spams
[2017-05-15T17:31:19.419856] - DEBUG - apps/blog/views.py - 15 - edit page
[2017-05-15T17:31:52.815638] - WARNING - apps/blog/models.py - 910 - something wrong happened
[2017-05-15T17:33:13.965295] - DEBUG - apps/blog/admin.py - 542 - delete post
[2017-05-15T17:34:40.124876] - INFO - apps/dashboard/forms.py - 375 - blog description changed
[2017-05-15T17:35:36.002486] - DEBUG - apps/utils/decorators.py - 611 - init decorator
[2017-05-15T17:37:16.536842] - INFO - apps/dashboard/forms.py - 1381 - app updated
[2017-05-15T17:37:36.125496] - DEBUG - apps/accounts/models.py - 1336 - new user registered
[2017-05-15T17:38:57.874546] - INFO - apps/dashboard/plugins.py - 16 - plugins updated automatically
[2017-05-15T17:39:37.356897] - WARNING - apps/accounts/views.py - 308 - something is happening
[2017-05-15T17:40:52.454765] - DEBUG - apps/dashboard/views.py - 48 - new dashboard page created
```

فرض کنید فایل `log.txt` به شما داده شده است. یک اسکریپت Bash به نام `solution.sh` بنویسید که آن را در کنار `log.txt` قرار دهیم و در صورت اجرا با دستور `bash solution.sh` کارهای زیر را انجام دهد و نتایج را در ۶ فایل متنی (در همان پوشه) ذخیره کند.

۱. می‌خواهیم فقط ۸۰۰۰ رخداد اخیر را بررسی کنیم. برای این کار ۸۰۰۰ خط آخر `log.txt` را در فایل `log-recent.txt` بنویسید. در ادامه دیگر با `log.txt` کاری نداریم و از `log-recent.txt` استفاده می‌کنیم.

۲. می‌خواهیم بدانیم از هر یک از ۶ سطح، چه تعداد event داریم. سطح‌ها را به همان ترتیب داده‌شده به همراه تعداد تکرار، مانند نمونه زیر در فایل `count.txt` بنویسید.

```
CRITICAL 10
ERROR 1014
WARNING 2525
INFO 3340
```

```
DEBUG 604
NOTSET 507
```

۳. می‌خواهیم event هایی که سطح CRITICAL یا ERROR یا WARNING دارند را در یک فایل داشته باشیم. برای این کار، این event ها را با همان ترتیب اولیه در فایل `errors.txt` بنویسید.
۴. می‌خواهیم از بین event هایی که در مرحله قبل استخراج کردیم (فایل `errors.txt`)، آن‌هایی که در مسیر `apps/blog` اتفاق افتاده‌اند (FILENAME آن‌ها با `apps/blog` شروع می‌شود) را به صورت جداگانه داشته باشیم. برای این کار، این event ها را با همان ترتیب اولیه در فایل `blog-errors.txt` بنویسید.
۵. می‌خواهیم event های مرحله قبل را بر اساس FILENAME و LINENUMBER مرتب کنیم. برای این کار event های فایل `blog-errors.txt` را ابتدا بر اساس FILENAME (به ترتیب الفبایی) و در صورت مساوی بودن FILENAME بر اساس LINENUMBER (به ترتیب عددی) مرتب کنید و در فایل `blog-errors-sorted.txt` بنویسید. در صورتی که برای دو event، هر دو مقدار FILENAME و LINENUMBER برابر بودند، ترتیب اولیه آن‌ها حفظ شود.
۶. در مرحله آخر می‌خواهیم برای event های مرحله قبل، مشخص کنیم که برای هر FILENAME چه تعداد event وجود دارد. برای این کار، FILENAME ها را به همراه تعداد event های هر کدام (به ترتیب تعداد event ها، از زیاد به کم) مانند نمونه زیر در فایل `blog-errors-filecount.txt` بنویسید.

```
apps/blog/forms.py 280
apps/blog/views.py 214
apps/blog/models.py 59
apps/blog/admin.py 392
```

دقت کنید که هر FILENAME باید تنها یک بار بیاید و بین FILENAME و عدد روبروی آن یک space باشد.

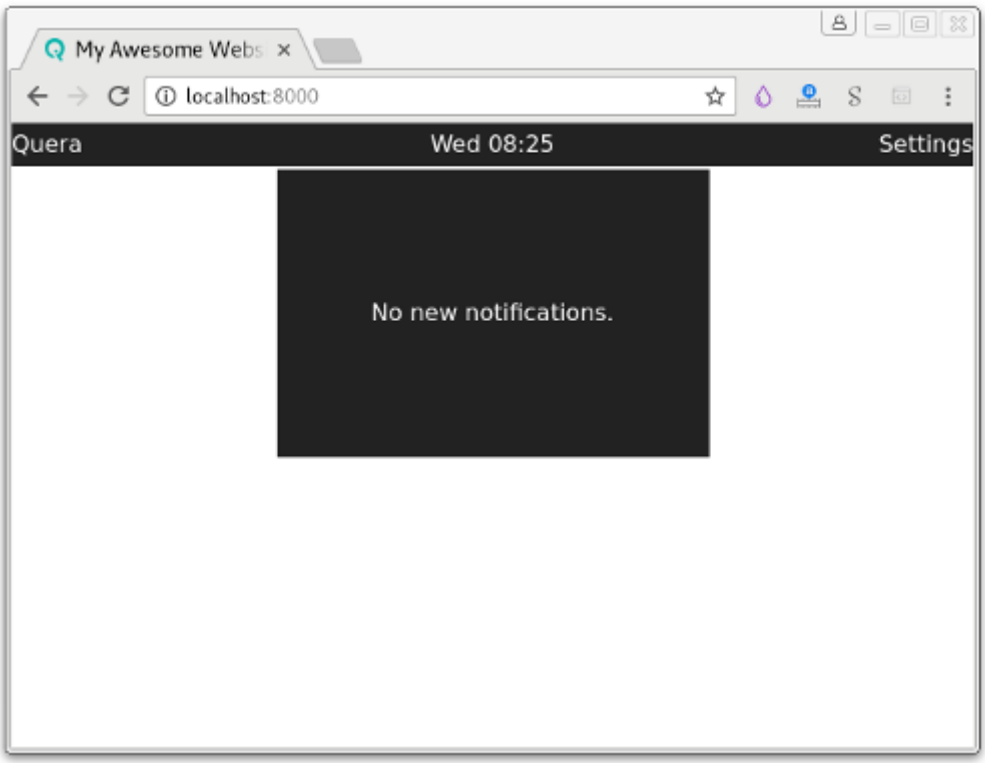
فایل `solution.sh` را به صورت یک فایل Zip فشرده کنید و به عنوان پاسخ آپلود کنید. دقت کنید که این فایل باید در ریشه فایل Zip باشد (در هیچ پوشه‌ای نباشد).

راهنمایی

دستورات grep, sort, awk, uniq, wc می‌توانند به شما کمک کنند.

گنوم (Front-end)

یکی از همکاران شما در شرکت قصد دارد با HTML و CSS طرح زیر را اجرا کند:



او کد `index.html` زیر را نوشته است و برای نوشتن CSS از شما کمک خواسته است.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>My Awesome Website</title>
    <link rel="stylesheet" href="style.css"/>
</head>
<body>

<div id="top-bar">
    <div id="title">Quera</div>
    <span id="time">Wed 08:25</span>
    <div id="settings">Settings</div>
    <div id="notifications">
```

```
        <p>
            No new notifications.
        </p>
    </div>
</div>

</body>
</html>
```

با توضیحات زیر این کار را برای او انجام دهید:

عنصر `body` نباید margin یا padding داشته باشد. رنگ تمامی متن‌ها باید `#EEE` باشد.

`#top-bar` باید از سمت بالا و راست و چپ کاملاً به لبه‌های صفحه بچسبد و ارتفاع آن باید ۳۰ پیکسل باشد. رنگ پس‌زمینه آن باید `#222` باشد.

عناصر `#title` و `#time` و `#settings` نیز باید ارتفاع ۳۰ پیکسل داشته باشند. وسط عنصر `#time` باید در وسط `#top-bar` باشد و متن درون آن نیز وسط‌چین باشد.

عنصر `#notifications` باید از نظر افقی، وسط باشد. رنگ آن `#222` باشد. عرض آن ۳۰۰ و ارتفاع آن ۲۰۰ پیکسل باشد و از بالا با `#top-bar` دو پیکسل فاصله داشته باشد. عنصر `p` درون آن باید دقیقاً در وسط قرار بگیرد.

باید نکات فوق با تغییر دادن اندازه صفحه همچنان رعایت شوند.

نکات

- کد CSS را در فایل `style.css` بنویسید و آن را در کنار `index.html` قرار دهید و به صورت Zip فشرده کنید و ارسال کنید. دقت کنید که این ۲ فایل باید در ریشه فایل Zip باشند (در هیچ پوشه‌ای نباشند).
- اجازه تغییر کد HTML را دارید ولی id های عناصر موجود را تغییر ندهید. زیرا id ها در داوری خودکار استفاده خواهند شد.

مارک‌دان (Front-end)

از شما خواسته شده تا یک ویرایشگر ساده Markdown مبتنی بر وب (با استفاده از HTML/CSS/JS) بنویسید. به عنوان نمونه، ویرایشگرهای آنلاین زیر را ببینید. اگر با Markdown آشنا نیستید، ابتدا در مورد آن جستجو و مطالعه کنید.

- http://markdownlivepreview.com/
- https://stackedit.io/editor

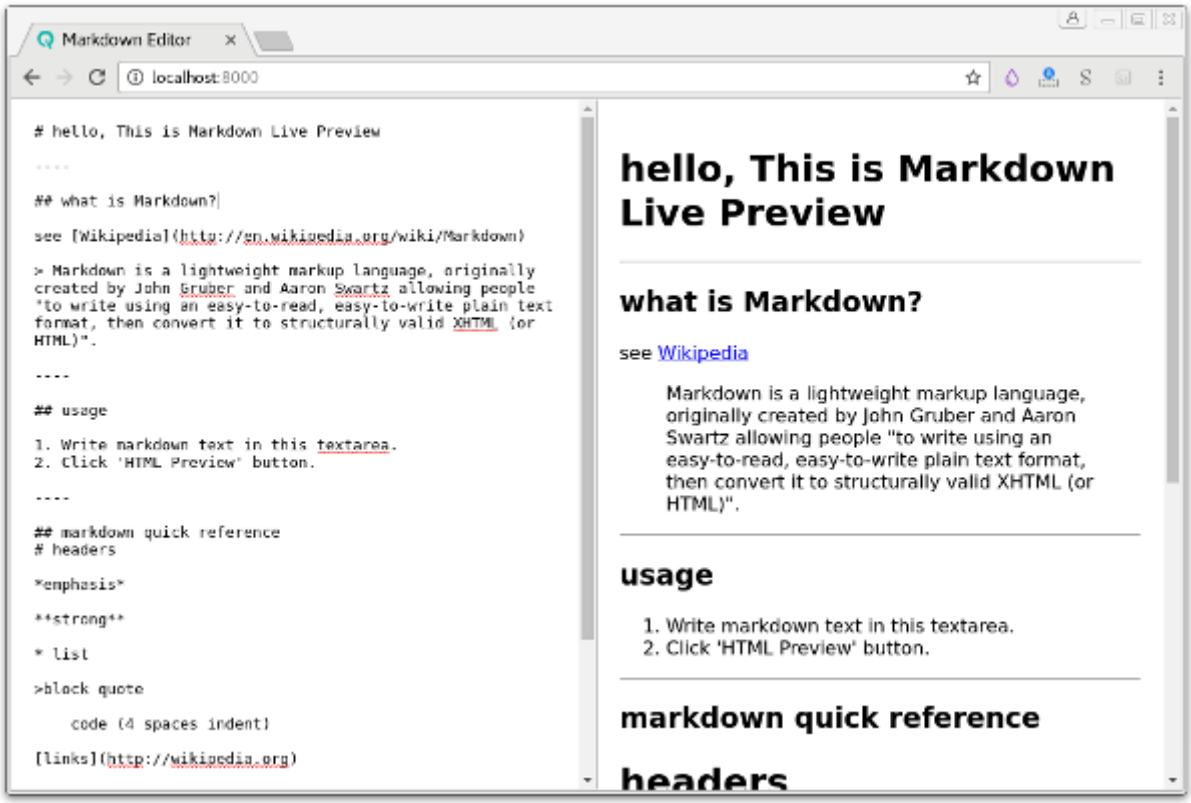
ویرایشگر شما باید ۲ بخش اصلی داشته باشد:

۱. بخش **input**: این بخش یک textarea برای نوشتن متن مارک‌دان است. id این بخش را `md-input` قرار دهید.
۲. بخش **preview**: این بخش یک div برای render و نمایش HTML متن مارک‌دان است. id این بخش را `md-preview` قرار دهید.

می‌خواهیم به محض تغییر متن بخش input، این متن render شود و HTML آن در بخش preview نمایش داده شود.

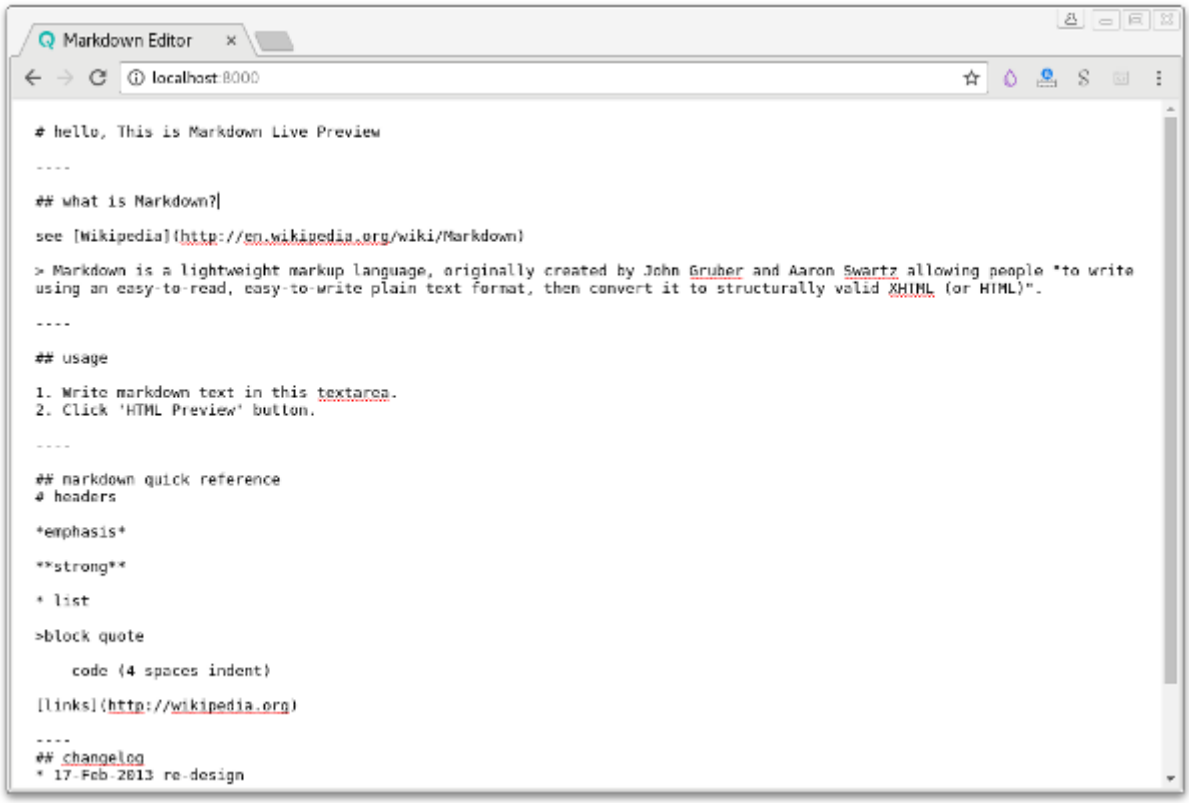
همچنین این ویرایشگر باید ۳ حالت نمایش داشته باشد که با میانبر `ctrl+m` بین این حالت‌ها به ترتیب زیر جابجا شود:

۱. حالت **split**: در این حالت بخش input در سمت چپ و بخش preview در سمت راست نمایش داده می‌شود. عرض هر بخش باید نصف عرض viewport باشد و ارتفاع هر دو بخش باید دقیقاً به اندازه ارتفاع viewport باشد. مانند تصویر زیر.

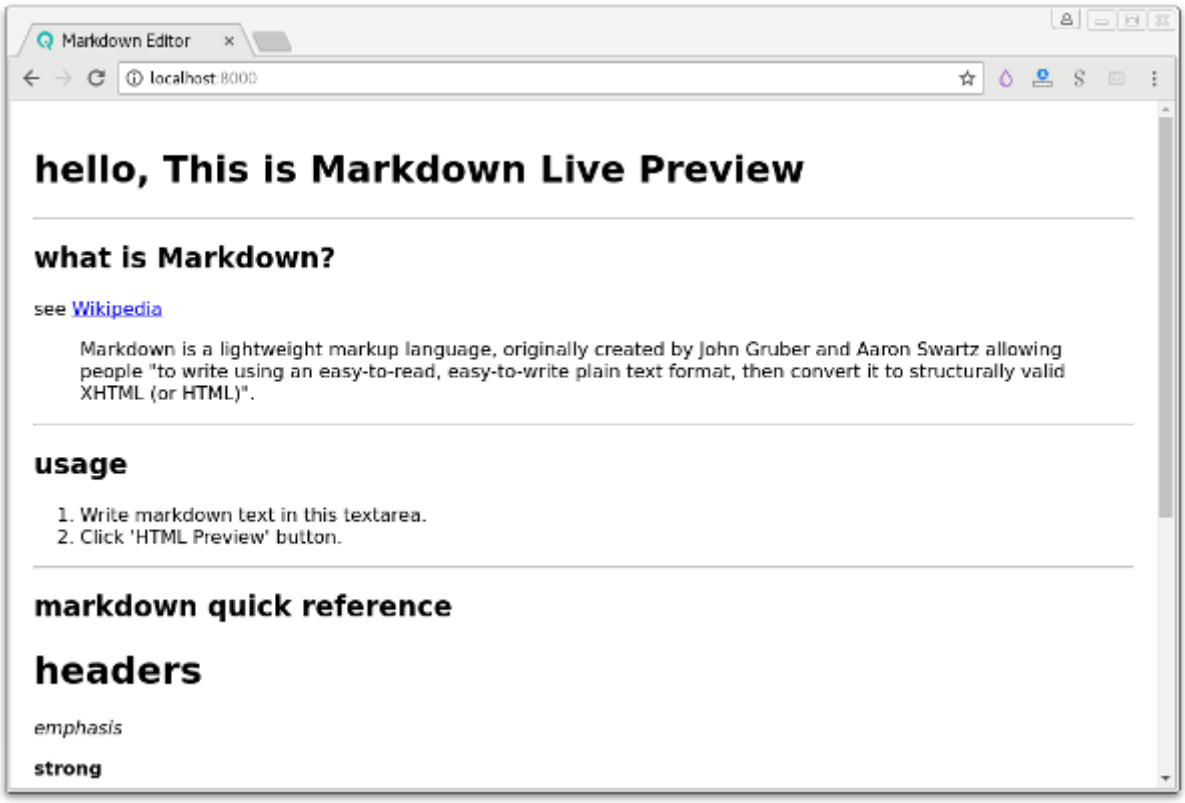


این حالت، حالت پیش‌فرض است و هربار هنگام لود شدن صفحه باید در این حالت قرار داشته باشیم.

۲. حالت **edit**: در این حالت تنها بخش input دیده می‌شود و بخش preview مخفی است (`display: none`). اندازه بخش input باید دقیقاً به اندازه viewport باشد.



۳. حالت **preview**: در این حالت تنها بخش preview دیده می‌شود و بخش input مخفی است (`display: none`). اندازه بخش preview باید دقیقاً به اندازه viewport باشد.



نکات

۱. توصیه می‌کنیم برای رندر متن مارک‌دان از کتابخانه‌های JS آماده استفاده کنید.

۲. با توجه به این که تست به صورت خودکار انجام می‌شود، دقت کنید که textarea و div، در حالت‌های مختلف، مختصات و اندازه درستی داشته باشند. مثلاً در حالت split، مختصات گوشه textarea، دقیقاً `(0, 0)` باشد، عرض آن دقیقاً نصف عرض viewport باشد و ارتفاع آن دقیقاً برابر با ارتفاع viewport باشد. و یا در حالت edit، مختصات گوشه textarea دقیقاً `(0, 0)` باشد، عرض آن دقیقاً برابر با عرض viewport باشد و ارتفاع آن دقیقاً برابر با ارتفاع viewport باشد. عرض و ارتفاع با در نظر گرفتن border و padding محاسبه می‌شود.

۳. فایل HTML اصلی باید `index.html` و در ریشه فایل Zip باشد (در هیچ پوشه‌ای نباشد). می‌توانید تعدادی فایل js و css در پوشه‌ای به نام `static` در کنار آن قرار دهید.

کتابخانه پویا (Front-end)

شرکتی که شما در آن کار می‌کنید پروژه‌ای مبتنی بر وب را برای یک کتابخانه آغاز کرده است و مقداری از کدهای سمت client را نوشته است. از شما خواسته شده تا این بخش را تکمیل کنید.

پروژه را از اینجا دانلود کنید و از حالت فشرده خارج کنید.

برای اجرا، به جای باز کردن مستقیم `index.html` با مرورگر، حتماً پوشه اصلی پروژه را به وسیله یک وب‌سرور سرو کنید. یک راه ساده برای این کار استفاده از وب‌سرور پایتون است.

با پایتون ۲:

```
cd bookstore
python -m SimpleHTTPServer 8000
```

با پایتون ۳:

```
cd bookstore
python -m http.server 8000
```

با این دستورات، پوشه پروژه بر روی پورت ۸۰۰۰ سرو می‌شود. با مراجعه به آدرس `localhost:8000` پروژه را مشاهده کنید و صفحات مختلف آن را ببینید.

همان‌طور که می‌بینید ۳ صفحه وجود دارد:

۱. صفحه اصلی که لیست کلیه کتاب‌ها در آن قرار دارد، با آدرس `http://localhost:8000/#` .
۲. صفحه مشخصات یک کتاب، با آدرس `http://localhost:8000/#book/BOOK_ID` .
۳. صفحه لیست کتاب‌های یک ژانر، با آدرس `http://localhost:8000/#genre/GENRE_NAME` .

اما مشکلی که وجود دارد این است که محتوای تمام این صفحات static است. از شما خواسته شده تا پروژه را طوری تغییر دهید که محتوای صفحات با درخواست ajax از `localhost:8000/api` دریافت شود.

ابتدا کدهای پروژه را بخوانید و با نحوه کار پروژه آشنا شوید. با باز کردن فایل `main.js` مشاهده می‌کنید که ۴ کار از شما به صورت TODO خواسته شده است. این ۴ کار را انجام دهید. سپس پروژه تکمیل‌شده را به صورت Zip فشرده کنید و به عنوان پاسخ آپلود کنید.

توضیحات API

نمونه‌هایی از پاسخ سرور API در پوشه `api` قرار داده شده است. نگاهی به فایل‌های این پوشه بیندازید. با توجه به این که پروژه را با وب‌سرور سرو کرده‌اید، با فایل‌های این پوشه، API شبیه‌سازی شده است و می‌توانید با jQuery.get یا توابع مشابه، به آدرس‌های زیر درخواست ajax بفرستید و پاسخ آن را دریافت کنید:

```
/api/get_all_books
/api/get_book/BOOK_ID
/api/get_genre_books/GENRE_NAME
```

برای آشنایی با ساختار JSON دریافتی از سرور نیز فایل‌های موجود در پوشه `api` را ببینید.

همچنین توجه کنید که وب‌سرور، نوع پاسخی که ارسال می‌کند را json تعیین نمی‌کند. بنابراین هنگام صدا زدن jQuery.get، نوع پاسخ موردانتظار (dataType) را json تعیین کنید. و یا این که خودتان متن JSON دریافتی را parse کنید.

نکته

دقت کنید که پروژه را به گونه‌ای Zip کنید که فایل `index.html` و پوشه‌های `static` و `templates` در ریشه فایل Zip قرار داشته باشند (داخل پوشه دیگری نباشند).