Schedr - Arranging Meetups has never been this easy.

Milestone 0: Describe the problem that your application solves.

Schedr is the convenient tool for all to arrange meetings because it automatically generates the common time slot among all attendees in the midst of their hectic schedules. It provides the feature of syncing all popular calendars you own in the cloud, so that you do not have to manually key in every single unavailable timings yourself. On top of that, the app allows for rescheduling when a single attendee is unable to make it for the meeting at the last minute, it automatically sends out a new meeting time-slot to all attendees, making the whole arrangement hassle-free and convenient.

Milestone 1: Describe your application and explain how you intend to exploit the characteristics of mobile cloud computing to achieve your application's objectives, i.e. why does it make most sense to implement your application as a mobile cloud application?

Schedr is a mobile cloud app that is easily accessible on any mobile platforms, or even PCs. The online scheduler allows various meeting attendees to sort out a common time slot amongst their busy schedules, even if they are on different platforms. This is one of the main reasons why Schedr has to be developed on the mobile cloud app because of the device independence it can provide, allowing event to be scheduled even when attendees are using different devices. Another thing to be expected of Schedr as a scheduling tool would be that users need to have easy access to Schedr; thus, with the use of mobile cloud application will enable Schedr to always be readily available to the users.

Milestone 2: Describe your target users. Explain how you plan to promote your application to attract your target users.

Anyone who has tight schedules and do meetups regularly. More specifically, they are the managers who are always on-the-run meeting their clients. They are also the dentists, doctors or any other specialized professionals whose work is arranged by appointments. These people are those with tight schedules and often face issues finding a common time-slot with their clients and corporate partners.

At the beginning, we would need to raise awareness for Schedr, we need to appear at places where the busy people would find us. Depending on the budget, it could mean traditional print advertising with the newspapers that professionals read daily, near the finance sections. In this context, content marketing could be used to educate the professionals on the amount of time they can save simply by using Schedr. PR publicity materials could be done with popular online news provider, such as Mashable and TechCrunch, to feature and introduce Schedr on their websites. Next, to participate in IT conference where IT office solutions are provided to inform of all tech personnel of the corporate world the presence of Schedr, since they are the ones handling all the IT-related materials of the companies, they would be the people who can help spread the product.

When the budget is not as much, we would be able to make use of other marketing techniques. First method would be to collaborate with dentists, doctors and any other specialized professionals, to try out our appointment scheduling services with their current system. However, the sophistication of the Schedr app would be of a higherl level in order for it to be feasible.

Alternatively, we can resort to Search Engine Optimization. There will be people out there looking for such a solution to reduce their pain in arranging meetings. These people would naturally become the first batch of users who are willing to test and try out our products more than anyone else. To bridge the distance between Schedr and

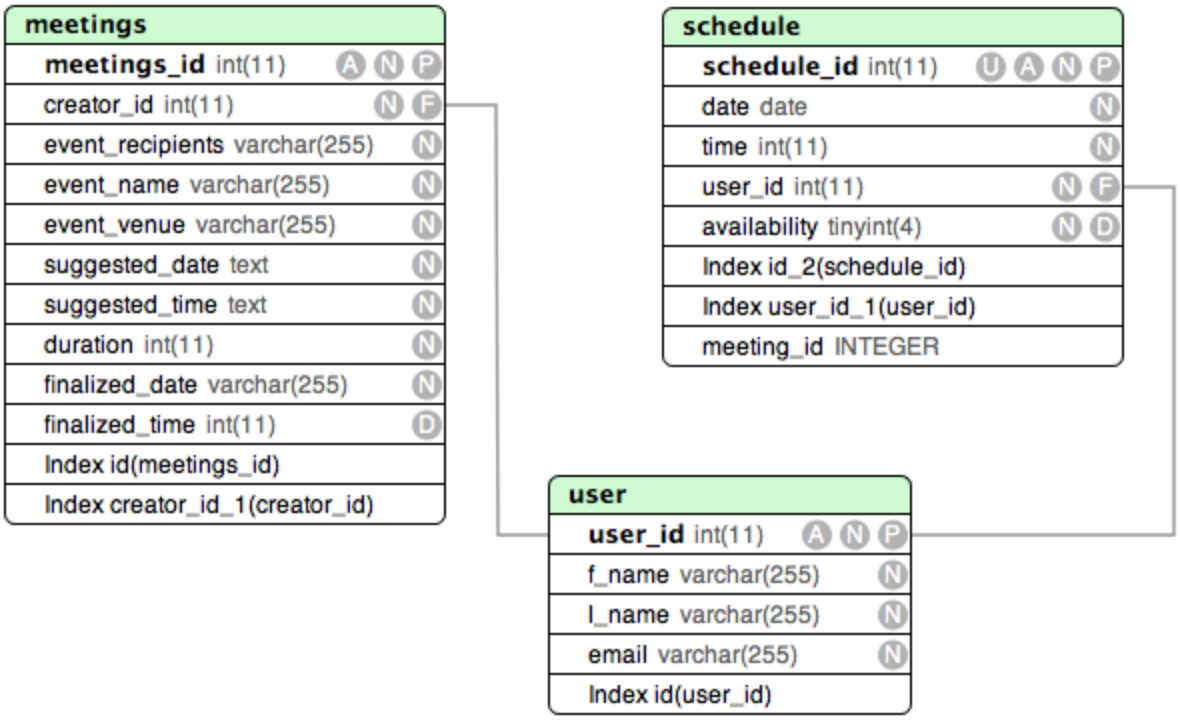
this people would mean improving our search rankings such that people know a solution (or another) has been developed and ready-to-use!

In terms of product development, we hope to reduce the switching costs of users from whichever scheduling method they are using. We are implementing a smooth user transition from using the Google+ calendar or even Outlook calendar to using Schedr to find the common time-slot by allowing them to sync the calendars into the app.

Milestone 3: Pick a name for your mobile cloud application. (Not graded).

Schedr

Milestone 4: Draw the database schema of your application.



Milestone 5: Design and document (at least 3) most prominent requests of your REST API. The documentation should describe the requests in terms of the triplet mentioned above. Do provide us with a brief explanation on the purpose of each request for reference. Also, explain how your API conforms to the REST principles and why you have chosen to ignore certain practices(if any).

1. POST /login/

```
{
        "fname" : "Chinab"
        "lname" : "Chugh"
        "email"  : "chinab91@gmail.com"
}
```

Return value:

```
{
        "user_id" : "2"
}
```

Data is pulled from the GCalendar using the G+ JS API and passed to the backend using /login/. It expects f_name, l_name, and email to be in a JSON post. On success, it stores the internal user id as a session for the later API calls

2. GET /specific_event/<int:meeting_id>

Return value:

```
{
	current_meeting:
	{
		creator_id: 18,
		duration: 2,
		event_name: "group meeting",
		event_recipients: "a@gmail.com,b@gmail.com,c@gmail.com",
		event_venue: "LT7A",
		finalized_date: “2013-08-16”,
		finalized_time: null,
		meetings_id: 2,
		suggested_date: "2013-08-16",
		suggested_time: "2"
	}
}
```

This API gets the specific event from the database. It then shows it to the user in the calendar page (last step where you login and take your calendar from GCal). It expects a GET parameter called meeting_id, which is used to query from the meetings table. The values returned like suggested_date and suggested_time are processed in the front end to show the calendar UI to tick/untick dates.

3. POST schedule/<int:meeting_id>

```
{
	“selected_timings”: {
				“0000_2013-09-20”,
				“0100_2013-09-20”
				}
}
```

Return value: 1 on success, 0 on failure

/schedule/ takes in a POST request of all the available free timeslots of the user after he/she has manually picked them from the calendar UI or logged in with GCal. The data is sent as a JSON and encoded with the ‘time_date’ format (ex. 0100_2013-09-27). This is then saved in the database. Following which, a function is called to check if all the people have submitted their choices for a meeting and if so, the scheduler algorithm is run to set a meeting for the recipients.

Our API conforms to the REST principle because we followed a triplet structure closely - having a request method and the relative URL, the parameters to pass it (using the JSON structure for POST params), and returned values (also using JSON or simple 1 or 0 to indicate whether is was a success or not). We have implemented some

exception handling in our REST API too. Though we don't support ALL exceptions yet, our API does return error 400(Bad Request), 401(Unauthorized) & 403(Forbidden) as needed.

Milestone 6: Tell us some of the more interesting queries (at least 3) in your application that requires database access. Provide the actual SQL queries you used.

1. existingUsers = User.query.filter_by(email = email).first()

Used in /login/ api to check if the person has logged in to the app before. If he is a first-time user, we create a new record to the database with his details as returned by G+ API.

2. newMeeting = Meetings(creator_id, event_recipients, event_name, event_venue, suggested_date, suggested_time, duration)

When a user creates a new meeting, an object of the Meetings class is created and the values from the form are saved by initializing the constructor of the class. In Flask, we have to add the object to the db session and commit it afterwards:

```
db.session.add(newMeeting)
db.session.commit()
```

3. existing_meetings = Meetings.query.all()

This returns all the existing meetings in the meeting table. We can then extract out the creator_id (person who created the event) and the recipients (which we join with the user table to get their internal user_id). We add all those existing meetings of a user to a dictionary which we jsonify to pass to the front-end to show in the 2nd screen or the admin panel

Milestone 7: Find out and explain what [QSA,L] means. Tell us about your most interesting rewrite rule.

The default behaviour of RewriteRule is to discard any existing queries and replace it with the rewrite. Using QSA instead allows multiple queries to be combined even after the rewrite. Example:
A call to products/123?q=search would call products.php?id=123 without the QSA flag wheere it would call products.php?id=123&q=search with the QSA flag applied.

By using the L flag, we are letting the RewriteEngine know that after you have rewritten this url, do not apply any future rules. Your work here is done, carry on.

As our project uses the Python Flask framework, we are not using Apache and hence have not defied any rewrite rules. However, our REST API endpoints still use 'clean' URLs such as /login/, /event/ & /admin/

Milestone 8: Create an attractive icon and splash screen for your application. Try adding your application to the home screen to make sure that they are working properly.

Milestone 9: Style different UI components within the application using CSS in a structured way (i.e. marks will be deducted if you submit messy code). Explain why your UI design is the best possible UI for your application.

The UI is kept minimal. The sole use of two contrasting colours, green and red to show a yes/no option respectively. The UI is the best possible for a meetings app as it is build to be focused on the dates and timeslots range that the admin has indicated while creating the event. This means that users invited to logon by the admin can swiftly move through the indicated dates and choose their available times for the meeting within the timeslots range set.

The lack of the capability to view other same event users' timetable in the UI is deliberate as we feel that other scheduling applications on the market requires too much user engagement and thoughts on which timeslots to choose. Our USP is in automated scheduling and a much simpler UI affirms that aim.

Milestone 11: Explain how you will keep your client in sync with the server. Elaborate on the cases you have taken into consideration and how it will be handled.

Each time a user logs in, we sync their google calendar with our own database. Doing so lets our servers be constantly up to date with the user's schedule for effective scheduling.

Milestone 12: Compare the pros and cons of basic access authentication to other schemes such as using cookies, digest access authentication or even OAuth. Justify your choice of authentication scheme.

We have chosen to use Google+ Authentication as the primary form of authentication for our app. We initially wanted to do both user/pass auth, but then realized that most of our users will have to log in twice to use our app, once via our own auth and a 2nd time via Google+. Because of that, we decided to just use *Sign in with Google+* as our primary login mechanism and extract the user's email & name from their Google profile.

Once the user has signed in and Google has authenticated them, we check if he is a new or returning user and update our tables accordingly. Regardless, we generate a session cookie for that session and pass it along to the user.

Milestone 13: Describe 1-3 user interactions within the application and explain why those interactions help make it better.

Interaction 1: Adding any additional email fields is such that admin only adds an extra email if he requires so by pressing the “+” button making the UI less cluttered and focused for users.

Interaction 2: Calender timeslots selection in calender.html. The buttons are made 40px x 40px for easy selection on touchscreen and the background of the button changes from “cross” (deselected) to “tick” (selected) when the button is toggled to easily know which time slots are unavailable/available.

Interaction 3: On a more abstract level, the application flow takes into account the ease of communication between the platform and user as only one message is put forth to the user at every stage or transition between events.

Milestone 14: Embed Google Analytics in your application and give us a screenshot of the report. Make sure you embed the tracker at least 48 hours before submission deadline as updates are reported once per day.

Milestone 15: Identify and integrate any social network(s) with users belonging to your target group. State the social plugins you have used. Explain your choice of social network(s) and plugins. (Optional)

As of right now, we have not intergrated any direct social networking posting or promotion in our app. Presently we have integrated Google+ but that is only to get the uesr's name and google calendar. If we had more time, we would have liked to add the abiliy to write back into the user's Google Calendar too.

Milestone 16: Make use of the Geolocation API in your application. Plot it with Bing or Google Maps or even draw out possible routes for the convenience of your user. (Optional)

We are making use of HTML5 Geolocation API in our app. We are sending the latlng that the browser retrives to the Google Maps reverse geolocation API to auto fill the venue form when making a new meeting

Milestone 17: Justify your choice of framework/library by comparing it against others. Explain why the one you have chosen best fulfils your needs. (Optional)

We are using the Flask framework to write Python. The reason we choose Flask over Django, which is the most popular framework in the market, is because:
1. It is extremely lightweight (it says it is a micro-framework!) which makes is very fast for REST API calls
2. We dont need need an admin panel which django seems to be very popular for. We dont use Flask templates as we wanted to decouple the frontend and backend as much as possible. That allowed us to settle on a REST API for all communication mode
3. It is simpler and much easier to pick up because there is no magic like Django :)