**Graduation project**

*Представьте себе, что ПМ (лид, архитектор) дал вам ТЗ и некоторое время недоступен. У вас конечно есть много мыслей, для чего нужно приложение, как исправить ТЗ, дополнить его и сделать правильно. НО НЕ НАДО ИХ РЕАЛИЗОВЫВАТЬ В КОДЕ. Нужно сделать все максимально просто, удобно для доработок и очевидно (если конечно в ТЗ нет оговорок). Все свои вопросы и предложения и хотелки оформляйете отдельно (в read.me например). Если делаете что-то сложнее простейшего случая (например справочник еды)- обязательно напишите в read.me. Как и выбор стратегии кэширования.*

Совершенство достигнуто не тогда, когда нечего добавить, а тогда, когда нечего отнять

*Антуан де Сент-Экзюпери*

Design and implement a REST API using Hibernate/Spring/SpringMVC (or Spring-Boot) **without frontend**.

The task is:

Build a voting system for deciding where to have lunch.

* 2 types of users: admin and regular users
* Admin can input a restaurant and it's lunch menu of the day (2-5 items usually, just a dish name and price)
* Menu changes each day (admins do the updates)
* Users can vote on which restaurant they want to have lunch at
* Only one vote counted per user
* If user votes again the same day:
  + If it is before 11:00 we asume that he changed his mind.
  + If it is after 11:00 then it is too late, vote can't be changed

Each restaurant provides new menu each day.

As a result, provide a link to github repository. It should contain the code, README.md with API documentation and couple curl commands to test it.

[10 Best Practices for Better RESTful API](http://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/)

P.S.: Make sure everything works with latest version that is on github :)

P.P.S.: Asume that your API will be used by a frontend developer to build frontend on top of that.

**[error](https://cloud.githubusercontent.com/assets/13649199/13672935/ef09ec1e-e6e7-11e5-9f79-d1641c05cbe6.png) Рекомендации**

* 1: **читаем ТЗ ОЧЕНЬ внимательно, НЕ надо ничего своего туда домысливать и творчески изменять**
* 2: **тщательно считайте количество обращений в базу на каждый запрос. Особенно при запросах от юзеров, которых очень много! Также на сложность запросов от них, чтобы не положить базу**
* 3: **тщательно считайте количество запросов в вашем API для отображения нужной информации**
* 4: **учитывайте, что пользователей может быть ооочень много, а админов- мало**
* 5: в проекте (и тестовом задании на работу) в отличии от нашего учебного topjava оставляйте только необходимый для работы приложения код, ничего лишнего:
  + 5.1 НЕ надо делать разные профили базы и работы с ней.
  + 5.2 НЕ надо делать абстрактных контроллеров на всякий случай.
  + 5.3 НЕ надо делать классов репозиториев, если там нет ничего, кроме делегирования.
  + 5.4 Из потребностей приложения (которую надо самим придумать) реализовывать только очевидные сценарии. Те.- НИЧЕГО ЛИШНЕГО.
  + 5.5 НЕ надо все бездумно кэшировать
* 6: базу лучше взять без установки (H2 или HSQLDB)
* 7: по возможности сделать JUnit тесты
* 8: уделяйте внимание обработке ошибок
* 9: далаем REST API в соответствии с [концепцией REST](http://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/), **с учетом иерархии принадлежности объектов**
* 10: не смешивайте TO и Entity вместе. Лучше всего, если они будут независимыми друг от друга.
* 11: если приложению в объекте требуется только его id, используйте reference (как мы при сохранении еды вставляем туда юзера)
* 12: [Use for money in java app](http://stackoverflow.com/a/43051227/548473)
* 13: **Историю еды и голосований лучше сделать. Нужно различать базовые вещи, которые закладываются в архитектуру приложения и неочевидные доработки к ТЗ, которых лучше не делать.**
* 14: Еще раз про [hashCode/equals в Entity](https://stackoverflow.com/questions/5031614/the-jpa-hashcode-equals-dilemma): не делайте сравнение по всем полям!
* 15: Название пакетов, имен классов для model/to/web достаточно стандартные (нарпимер model/domain). НЕ надо придумывать своих собственных правил.
* 16: Предпочтительно использовать DATA-JPA (можно без лишней делегации, напрямую из сервиса/контроллера дергать Repository). В случае JPA позаботьтесь о своем собственном generic DAO.
* 17: Вместо ModelMatcher (который нуждается в javadoc) в тестах контроллеров можно использовать json-path иxxx.json("{...}"), патч 11\_04\_HW10\_duplicate\_datetime
* 18: На topjava мы смотрели разные варианты использования, тут делаем максимально просто. С TO многие вещи упрощаются.
* 19: Проверьте, не торчат ли из кода учебные уши topjava, типа ProfileRestController.testUTF(), AbstractServiceTest.printResult() или закомментированные JdbcTemplate. Назначение этого проекта совсем другое.
* 20: ORM работает с объектами. [В простейших случаях fk\_id как поля допустимы](https://stackoverflow.com/questions/6311776/hibernate-foreign-keys-instead-of-entities), но при этом JPA их уже никак не обрабатывает и не используйте их вместе с объектами. Ссылка на stackoverwrflow в коде обязательна!
* 21: Проверьте, станет ли код проще с @AuthenticationPrincipal (урок 11, Доступ к AuthorizedUser). С ним можно убрать из AuthorizedUser все статические методы.