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 and **README.md with API documentation and curl commands to get data for voting and vote.**

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.

Разработать и реализовать REST API с помощью Hibernate/Spring/Spring MVC (или Spring-Boot) без внешнего интерфейса.

Задача:

Постройте систему голосования, чтобы решить, где пообедать.

• 2 типа пользователей: администратор и обычные пользователи

• Администратор может занести ресторан, и его обеденное меню дня (обычно 2-5 пунктов, просто название блюда и цена)

• Меню меняется каждый день (обновления делают администраторы)

• Пользователи могут голосовать, в каком ресторане они хотят пообедать

• Только один голос на одного пользователя

• Если пользователь голосует снова в тот же день:

o если до 11:00 мы предполагаем, что он передумал.

o если это после 11:00, то слишком поздно, голосование не может быть изменено

Каждый день в ресторане предлагается новое меню.

В результате укажите ссылку на репозиторий github.

Он должен содержать код и README.md с документацией API и командами curl для получения данных для голосования и голосования.

P.S.: убедитесь, что все работает с последней версией, которая находится на GitHub :)

P.P.S.: предположим, что ваш API будет использоваться frontend-разработчика для создания фронтенда в довершение всего.

**------------------**

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

* **Сделай новый проект и добавляй туда только то что нужно! Локализация, типы ошибок, BeanMatcher, Json View, излишние делегирования и наследования - не нужны!**
* **API продумывай с точки зрения не программиста и объектов, а с точки зрения того, кто им будет пользоваться (frontend)**
* **Сначала сделай основной сценарий по ТЗ. Все остальное (если очень хочется, 3 раза подумай) - потом.**

*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 для использования со стороны клиента (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)- объязательно 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*

* 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 отличие от нашего учебного проекта 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 и сервисов**, если там нет ничего, кроме делегирования.
  + 5.4 Из потребностей приложения (которую надо самим придумать) реализовывать только очевидные сценарии. Те.- НИЧЕГО ЛИШНЕГО.
* 6: базу лучше взять без установки (H2 или HSQLDB). Ваше приложение должно сразу запуститься **ОПТИМАЛЬНО- без всяких настроек**
* 7: по возможности сделать JUnit тесты
* 8: уделяйте внимание обработке ошибок
* 9: далаем REST API в соответствии с концепцией REST, **с учетом иерархии принадлежности объектов**
  + [15 тривиальных фактов о правильной работе с протоколом HTTP](https://habrahabr.ru/company/yandex/blog/265569/)
  + [10 Best Practices for Better RESTful API](http://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/)
* 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 сделать НУЖНО! Есть 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 полям!
* 15: Название пакетов, имен классов для model/to/web достаточно стандартные (например model/domain). НЕ надо придумывать своих собственных правил.
* 16: **Используйте DATA-JPA** (можно без лишней делегации, напрямую из сервиса/контроллера дергать Repository).
* 17: В DATA-JPA 2.x используются Optional. Попробуйте работать с ними, это безопасный способ работать с null значениями.
* 18: На учебном проекте мы смотрели разные варианты использования, тут делаем максимально просто. С TO многие вещи упрощаются.
* 19: Проверьте, не торчат ли из кода учебные уши нашего проекта, 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).
* 22: Не размещайте логику приложения и преобразования в TO в слое доступа к DB
* 23: Если используете кэширование, **тщательно продумайте, что надо кэшировать (самые частые запросы)**, а что нет (большие или редкозапрашиваемые данные)!

**Попробуйте подергать свое API по всем типичным сценариям ТЗ!**

* Удобно использовать? Можно сделать проще? Например чтобы проголосовать за ресторан залогиненному юзеру достаточно restorauntId.
* Удовлетворяет ли принципам REST (см. ссылки выше)?
* Сколько раз пришлось его вызвать API для типичного сценария (нарпимер посмотреть рестораны с едой)?
* Сколько запросов к базе было сделано? Можно ли сократить (например с FETCH/Graph или через кэширование)?