Read this first:

1. This take-home test could be finished in 4-6 hours. Extra Google research is expected, feel free to do it.
2. There're some optional open questions, creativity is always welcomed at Tyme.
3. Take your time and try to make it as good as you can, just let us know if you need more time. Hopefully, you can learn some new things through the exam!

Challenge 1.
Your team member is going to expose these APIs, what is your feedback regarding to API design?

```
Get /users
POST /users/new
POST /users/:id/update
POST /users/:id/rename
POST /users/:id/update-timezone
DELETE /users/delete?id=:id
```

Optional: what's your personal opinion on CRUD endpoints?

Challenge 2.
What is your feedback when you review this code? Would you give this a `lgtm`?

```
public class Account {
   private int debitBalance;
   private int creditBalance;

   void increaseBalance(int amount, boolean isCredit){
      if(isCredit) {
         this.creditBalance += amount;
      } else {
         this.debitBalance += amount;
      }
   }
   //getter,setter
}
```

Challenge 3.
What performance issue related to the database can you see in the following code, and how are you going to fix it?
We have a database to store course and student on two different tables, this function is to load students belong to first 1000 courses

```
public void loadStudents(){
   List<Course> courses = loadFirst1000Courses();
   for (Course course: courses) {
      List<Student> student = loadStudentFromCourse(course);
      print(student);
```

```
    }
}
```

Challenge 4.
This piece of code is the logic behind an API for transfer money, we're using RDBMS e.g. MySQL as our persistent layer.
What database-related (2-3) issues would you find in this method?
Hint: It's pseudo code, therefore don't focus on coding syntax.

```
function transfer(fromAccId, toAccId, amount) {
    var from = findByAccountId(fromAccId);
    var to = findByAccountId(toAccId);
    if(from.balance < amount) {
        throw new BalanceErrorException("not enough balance to transfer");
    }
    updateBalance(from, balance - amount);
    updateBalance(to, balance - amount);
}
```

Challenge 5.
We found this code in our git repository, what problems related to application security can you see in the following code?

```
// A service store user credentials and authenticate user (login)
class User {
    /**
     * Use salted password
     */
    private static final String PASSWORD_SALT = "tymesalt";

    /**
     * Stored password hashed
     */
    private String hashedPassword;

    /**
     * Authenticates user against given password, return true if the password matches
     * @param password
     * @return
     */
    public boolean verifyPassword(String password){
        if (isNullOrEmpty(hashed_password) || isNullOrEmpty(password)) {
            return false;
        } else {
            return this.hashPassword(password).equals(hashedPassword);
        }
    }

    /**
     * Update password
```

```
    * @param newPassword
    */
   public void changePassword(String newPassword) {
      this.hashedPassword = this.hashPassword(newPassword);
   }

   /**
    * Generate hash from given password
    * @param password
    * @return
    */
   private String hashPassword(String password){
      //generate md5hash of a string
      return md5Hash(password + PASSWORD_SALT);
   }
}
```

Challenge 6.
Database queries are getting slow when the database size increases. What are your suggestions to improve performance over the time?
Hint: List out 4 solutions.

Challenge 7: Optional
System design
Design a high load payment system.
The problems:

- During the Black Friday, the system needs to handle 3000 concurrent requests.
- The frequency of balance/transaction inquiry is usually 5 times higher than requesting payments e.g. 2500 reads + 500 writes
- Reliability is expected, e.g. we don't want to approve any inappropriate payments.
- User experience is also important, low-latency is expected.

Resource limit:

- We have 6 servers where up to 3 servers could be used for the relational databases. One database can open 250 connections.