

|  |
| --- |
| Interaction Humain-Machine, introduction  S2.01-S2.02, Attribution de tutorat |
| Git : <https://gitlab.univ-lille.fr/sae2.01-2.02/2022/A-G5>  Commit : |
| 12 juin  BUT Informatique 2021/2022, S2.A-G5  Alexandre Herssens  Léopold Varlamoff  Théo Franos |



**Sommaire**

I. Application finale 2

II. Méthodologie 3

A. Cahier des charges 3

B. Conception Centrée Utilisateur 3

III. Choix ergonomiques 4

A. Selon Bastien et Scapin 4

1. Guidage 4

2. Charge de travail 5

3. Contrôle explicite 5

4. Adaptabilité 5

5. Gestion des erreurs 6

6. Homogénéité et cohérence 6

7. Signifiance des codes et dénominations 7

8. Compatibilité 7

B. Autres règles et lois 7

1. Miller 7

2. Fitts 7

IV. Contribution et Bilan 8

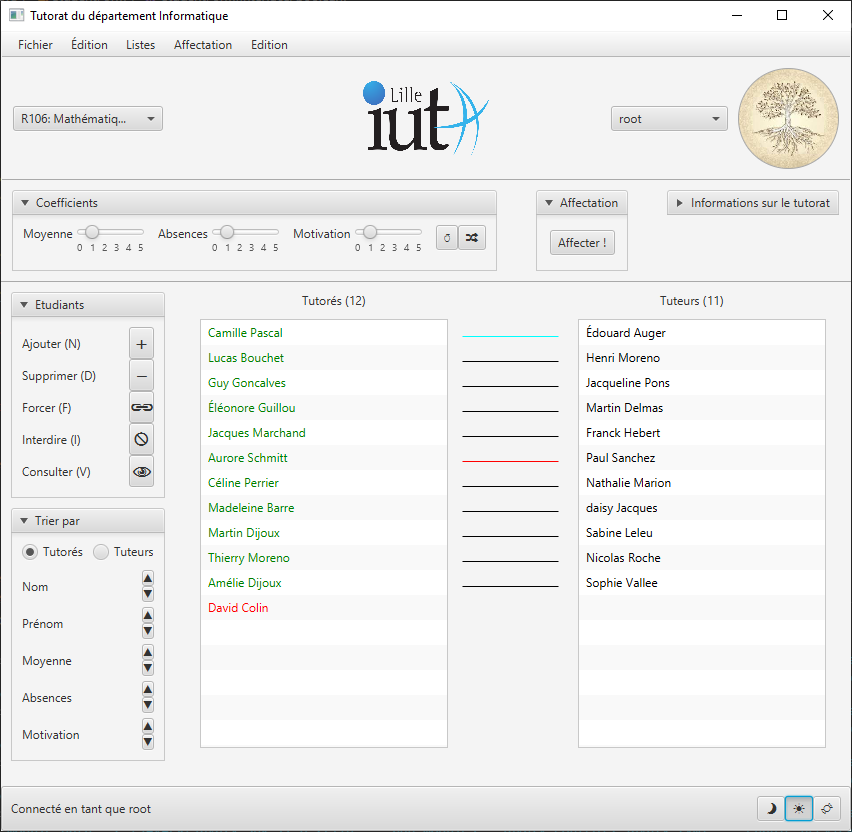
A. Alexandre 8

B. Léopold 8

C. Théo 8

V. Dépôt Git 9

# Application finale



# Méthodologie

## Cahier des charges

Ce qui nous était imposé :

|  |  |
| --- | --- |
| Contrainte | Implémentation |
| Visualisation d’une affectation | Deux listes d’étudiants, les étudiants affectés sont reliés par un trait, les étudiants dédoublés sont en violet. |
| Information sur l’affectation | Possible d’obtenir le pourcentage d’étudiants affectés, les couples d’étudiants, les affectations forcées et interdites. |
| Voir les étudiants non affectés | Les étudiants qui n’ont pas de binômes apparaissent en rouge. |
| Modifier les coefficients | Des potentiomètres permettent de modifier les coefficients. |
| Forcer une affectation | Un drag’n’drop entre deux étudiants forcera le duo (clique gauche ou menu prévu à cet effet). |
| Interdire une affectation | Un drag’n’drop entre deux étudiants interdira le duo (clique droit ou menu prévu à cet effet). |

Ce que nous avons implémenté en plus :

|  |  |
| --- | --- |
| Contrainte | Implémentation |
| Trier les listes d’étudiants | A l’aide de bouton dans le menu de gauche, tri par ordre croissant / décroissant sur plusieurs critères. |
| Consulter les informations des étudiants | Laisser la souris sur un étudiant fera apparaître sa moyenne, niveau, etc… |
| Ajouter / supprimer des étudiants | Ajouter un nouvel étudiant manuellement ou à partir des listes du département. |
| Remplacer un étudiant | Garde l’affectation mais remplace l’étudiant |
| Maximiser un certain coefficient | Disponible dans le menu |
| Rendre les coefficients aléatoires | Bouton pour coefficients aléatoires |
| Choisir une ressource | Possibilité d’avoir plusieurs tutorats dans différentes ressources. |
| Se connecter en tant qu’enseignant | L’enseignant référent du tutorat peut se connecter. |
| Choisir le thème de l’application | Choix parmi 3 thèmes différents (boutons en bas à droite) : jour, sombre et bonbon. |

## Conception Centrée Utilisateur

Avant de commencer la conception de l’application, nous avons réalisé un profilage des utilisateurs de notre application, dans le but d’établir les attentes des personnes qui allaient utiliser notre application. Nous en sommes venus à la conclusion que l’utilisateur sera un maître de conférences de 44 ans qui utilise régulièrement des logiciels (pour des entrées de notes ou absences), mais à un niveau technologique moyen et un temps limité à consacrer à notre application. Ses tâches quotidiennes lors que l’utilisation sera :

**1.** Entrer ou importer des listes d’étudiants.

**2.** Choisir des critères d’affection ou des filtres.

**3.** Lancer le logiciel pour affecter les étudiants.

**4.** Vérifier les résultats et les informations, les modifier le cas échéant.

Après avoir créé une persona, nous avons itéré sur plusieurs prototypes.

La conception de ces prototypes était précédée d’une réévaluation des exigences des utilisateurs et d’une amélioration de la séquence d’interaction par rapport au dernier cycle. Nous faisions également des brainstormings pour pouvoir recueillir un maximum d’idées et s’assurer qu’aucun élément ne manque. Après la production de la maquette, nous l’évaluions grâce à une analyse du cahier des charges et des éléments que nous avions préparé avant la conception. Nous sollicitions également notre enseignant lorsque nous terminions un prototype, pour obtenir du feedback d’un utilisateur potentiel. Si un ou plusieurs éléments venaient à manquer, nous continuions le cycle.

# Choix ergonomiques

## Selon Bastien et Scapin

### Guidage

Nous avons tout mis en œuvre pour conseiller, orienter, informer, instruire et guider les utilisateurs de notre application.

Nous avons fait un gros travail sur l’incitation. Tout au long de l’utilisation de notre application, l’utilisateur a un moyen de visualiser où il en est, ce qui lui est possible de faire, et a à sa disposition des outils d’aide accessibles. Nous incitons également l’utilisateur lors de l’entrée de donnée, à l’aide de guide sur les valeurs acceptées et des labels qui explique ce qu’il doit entrer (par prévention à la protection contre les erreurs).

Nous pensons également que les contrôles et les menus ont été correctement groupés de façon à les distinguer des uns des autres, et que l’utilisateur peut facilement identifier la classe des items. Leur localisation est pensée pour regrouper les éléments similaires et éloigner les éléments non compatibles sémantiquement parlant, et l’encadrement et la séparation des menus avec des MenuSeparators. Nous mettons en plus de ça des distinctions visuelles pour que l’utilisateur face bien la différence entre les éléments.

Le feedback immédiat à l’utilisateur n’est par contre pas respecté pour toutes les fonctionnalités implémentées : par exemple, pour le drag’n’drop utilisé pour forcer ou interdire l’affectation de deux élèves, l’utilisateur n’aura de feedback que lorsqu’il lancera l’affectation, ce qui pourrait le faire douter quant à la réussite de son action. Il faudrait qu’il soit plus rapide.

En revanche, nous pensons avoir été limpide en ce qui concerne la lisibilité de notre interface : les éléments sont lisibles, assez espacés pour permettre une lecture facile pour tous les utilisateurs, et nous respectons les codes sémantiques classiques pour ne pas déstabiliser l’utilisateur : titres centrés, police lisible, etc…

### Charge de travail

Notre interface est conçue dans le but de réduire les temps d’interactions et la charge de travail de l’utilisateur, dans le but de pouvoir réaliser la tâche principale efficacement.

Les capacités de la mémoire à court terme étant limitées, nous avons privilégié la brièveté des informations affichées. Tous nos items sont très succincts, constitués d’un mot, voire de zéro lorsqu’une icône nous paraissait être assez parlante pour remplacer une explication. Dans les cas où des informations longues doivent être affichées, une version abrégée est toujours disponible pour réduire le temps de lecture. Nous avons également minimisé le nombre d’action nécessaire pour effectuer une tâche. Quel que soit le but de l’utilisateur, nous avons fait en sorte que cela ne requiert pas de passer par de nombreux menus ou de devoir entrer plusieurs fois les mêmes informations.

De plus, nous avons réduit la densité des informations de notre interface. L’utilisateur n’aura dans son champ de vision que les informations nécessaires à la tâche en cours, et nous avons fait en sorte qu’il n’ait pas à mémoriser des informations en trop grande quantité.

### Contrôle explicite

Les actions réalisées par l’utilisateur sont explicites : lorsqu’il ne demande aucune action, il ne se passe rien. De plus, lorsqu’il veut qu’une action soit réalisée, seule cette action est déclenchée par le système et aucune autre. Par exemple, lorsqu’un utilisateur entre un étudiant, une fenêtre lui demande de confirmer son action.

L’utilisateur a également le contrôle sur l’application. Il peut, s’il le souhaite, annuler ses actions avant que celles-ci ne prennent effet.

### Adaptabilité

Nous avons conçu notre interface pour qu’elle soit adaptée au contexte, aux besoins et aux préférences des utilisateurs.

La flexibilité est un point qui nous paraissait important et que nous avons respecté tout au long de la conception de notre application. Elle permet à l’utilisateur de réaliser une même tâche de plusieurs manières différentes pour lui donner le choix de la manière qu’il ou elle préfère. De plus, les actions peuvent être effectuées dans n’importe quel ordre, elles sont indépendantes les unes des autres, à condition qu’un tutorat soit sélectionné. L’utilisateur peut également désactiver temporairement les affichages qui ne lui sont pas utiles sur le moment.

Notre application prend également en compte l’expérience varié des utilisateurs. Notre interface est conçue pour être utilisable par tous, quel que soit le niveau technologique. Nous avons, en plus de ça, implémenté des fonctionnalités qui permettent aux utilisateurs les plus expérimentés d’avoir une utilisation plus rapide et efficace de l’application, comme par exemple des raccourcis claviers. Nous avons également pris en compte l’évolution du niveau d’un utilisateur novice : au fur et à mesure qu’il devient plus à l’aise avec l’interface, il sera guidé vers une manière plus rapide d’exécuter ses tâches du quotidien.

### Gestion des erreurs

Nous avons pris des mesures pour nous protéger des erreurs. Par exemple, les boutons qui ne devraient pas être utilisé sont grisés et inutilisables. De plus, nous avons plusieurs fenêtres d’erreurs ou de signalement qui avertit l’utilisateur s’il s’apprête à effectuer une action qui pourrait compromettre le bon fonctionnement de l’application. Nos affichages sont protégés pour éviter à l’utilisateur de les modifier et nous avons pris en compte les éventuelles erreurs qu’il pourrait commettre.

Les fenêtres d’erreurs que nous avons créées donnent des informations claires et pertinentes sur la nature de l’erreur, pourquoi elle s’est produite et comment la corriger. Ces informations sont brèves et les erreurs ne résultent en aucune action.

Dans le cas où une erreur venait quand même à survenir, nous aurions voulu implémenter un bouton « retour en arrière », mais cette fonctionnalité n’a pas vu le jour par manque de temps.

### Homogénéité et cohérence

A travers toute notre application, nous avons pris soin de garder une certaine constance. Par exemple, les mêmes actions portent le même nom partout dans l’application, les fenêtres modales de même natures (erreurs, entrées de données) sont similaires, les menus fonctionnent de la même manière, et la palette de couleur reste la même.

### Signifiance des codes et dénominations

Les noms et codes des éléments ont été choisis dans le but de refléter clairement leur fonction. Les titres sont distincts et véhiculent ce qu’ils représentent, et les groupes sont nommés explicitement pour ne pas induire l’utilisateur en erreur.

### Compatibilité

Notre interface est compatible car notre mise en page, bien que sobre et peu révolutionnaire, mais tous les contrôles sont visibles à l’écran pour une utilisation aisée, notamment pour les utilisateurs novices. De plus, des raccourcis existent pour les utilisateurs experts. L’utilisation de notre application devrait donc rester intuitive.

## Autres règles et lois

### Miller

Notre application respecte la règle du « nombre magique sept, plus ou moins deux » issue de l’article de psychologie publié par Miller. Tous nos menus et nos regroupement sémantiques ont été conçus pour ne pas dépasser 9 items, ce qui risquerait d’augmenter la charge cognitive de l’utilisateur.

### Fitts

Nous avons également fait en sorte de respecter la loi de Fitts. Pour les actions les plus importantes, et pour guider l’utilisateur lorsque c’est nécessaire, nous avons privilégié les cibles larges et proches pour faciliter un maximum le pointage.

# Contribution et Bilan

## Alexandre

## Léopold

## Théo

# Dépôt Git

Dépôt git : <https://gitlab.univ-lille.fr/sae2.01-2.02/2022/A-G5>

Numéro de commit :

SHA :

Classe principale : ihm.Interface

Packages principaux : ihm, ihm.\*

Packages de tests : ihmtests