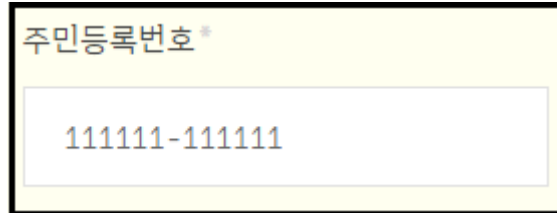


- 1) 주민등록번호
 - 구현화면



- HTML : 현재의 입력값(this.value)이 숫자(0~9)가 아닐 경우 '' (공백) 으로 대체(replace)

```
<div class="col-lg-6">
  <div class="checkout__input" id="sign_up_ssn">
    <p>주민등록번호<span>*</span></p>
    <input name="ssn" type="text" id="signUserssn" v-model="ssn" maxlength="13"
    @keyup="getSsnMask(ssn)"
    oninput="this.value = this.value.replace(/[^0-9]/g, '')"><br>
  </div>
</div>
```

- JS : 숫자를 6개 이상 입력시 자동으로 - 가 생성 (ex; 생년월일 - 주민뒷자리)
 - keyup : key 를 누울때 발생하는 이벤트

```
getSsnMask(val)
{
  let res = val;
  if(res.length === 6){
    this.ssn = res + "-";
  }
}
```

- 2) 아이디 - 유효성 체크(A-Za-z0-9)

- HTML : idValid 의 기본 stat 은 True, lidValid(=false) 일 경우, 유효하지 않는 ~ Text 가 출력

```
<div class="col-lg-6">
  <div class="checkout__input" id="sign_up_id">
    <p>아이디<span>*</span></p>
    <input name="id" type="text" class="testid" v-model="id" maxlength="20" @keyup="valid">
    <div v-if="!idValid">
      유효하지 않는 아이디 입니다.
    </div>
  </div>
</div>
```

- JS

```
computed: {
  valid() {
    if(/[A-Za-z0-9]+$/.test(this.id)){
      this.idValid = true
    }else{
      this.idValid = false
    }
  },
}
```

2-2) 아이디 - 중복 확인

- HTML : 클릭시(@click) 메소드 실행(userIdMatch)

```
<div class="col-lg-6">
  <div class="checkou__input">
    <button type="button" class="orderBtn1" @click="userIdMatch(id)">중복 확인</button>
  </div>
</div>
```

- JS

```
async userIdMatch(id){
  try {
    // console.log(this.id)
    let data = await axios.post("http://localhost:3000/finds",{
      UserId: this.id
    });
    if(this.id === data.data[0].UserId){
      alert("중복")
    }catch(err){
      alert("사용가능")
    },
  },
```

- [Async & Await](#) : 자바스크립트 비동기 처리 패턴으로 자바스크립트의 비동기적 사고 방식에서 벗어나 동기적(절차적)으로 코드를 작성할 수 있게 도와줍니다.
- data 변수를 선언해 자료를 담을 건데, axios 로 post 를 해(링크, 데이터)를 받는 것
- data.data[0].UserId 는 아래 메소드에서의 쿼리문 자체가 UserId가 중복인것을 찾아오기 때문

2-3) 아이디 - 중복확인인 연동된 메소드 살펴보기

- 여기서 데이터는 **UserId: this.id** 가 해당되며 if 문에 의해 중복 및 사용가능 여부를 호출
- **UserId**는 **UserId**가 호출된 곳에 **this.id** 를 전달함
- 우선 링크를 살펴보면 <http://localhost:3000/finds> 는 Router.js 로 연결되는 것을 의미

```
router.post( path: '/finds', PostUserId);
```

- 여기서 **PostUserId** 는 다시 Product.js 로 연결되는 것을 의미

```
export const PostUserId = (req, res) => { //req, res 형태로 나오는걸 promise "약속" 형태라 한다 async
  await 요놈들도 promise 형태로 나온다는데
  // Post
  // 사용자 아이디 중복 확인
  let query = "SELECT * FROM user WHERE UserId= ? ";
  let data = [req.body.UserId];
  // console.log(data);
  db.query(query, data, function (err, result){ // database.js에서 연동한 DB 데이터베이스에 쿼리
    던질거야.
    if(err){ //쿼리 던졌는데 에러 났다!
      console.log("Error : " + err); //에러내용 원데
      res.send(err); //화면(VUE) 에 send 할거야 err를
    }else{
      res.json(result); //JSON 형태로 보낼거야, index.js에 express.JSON 로 여기서 사용
    }
  });}
```

- `let data = [req.body.UserId]` 에 **UserId** (key) 들어가 Value 값인 **this.id** 가 전달되는 것
- `db.query(data)` 부분에 **this.id** 가 들어가는 것 즉, **PostUserId**의 `data = this.id`

3) 비밀번호 체크 : 유효성 검사 및 동일성 체크

- HTML

```
<div class="col-lg-6">
  <div class="checkout__input">
    <p>비밀번호<span>*</span></p>
    <input name="pw" type="password" class="signup_pw" v-model="password" maxlength="16"
    @keyup="passwordValid" >
    <div v-if="!passwordValidFlag">
      유효하지 않은 비밀번호 입니다.
    </div>
  </div>
</div>
<div class="col-lg-6">
  <div class="checkout__input">
    <p>비밀번호 확인<span>*</span></p>
    <input name="pw_check" type="password" class="signup_pw_check" v-model="passwordCheck"
    maxlength="16" @keyup="passwordCheckValid">
    <div v-if="!passwordCheckFlag">
      비밀번호가 동일하지 않습니다.
    </div>
  </div>
</div>
```

- JS : 기본적인 data, return 값은 아래처럼 설정됨, 공란이거나 true 거나

```
data () {
  return {
    passwordCheck: '', passwordValidFlag: true, passwordCheckFlag: true,
  }
}
```

- JS > Method : true 면 호출되는 것 없고, false 면 v-if 의 HTML Text 가 출력

```
methods: {
  passwordValid () {
    if (/^(?=.*[a-z])(?=.*[0-9]).{8,16}$/ .test(this.password)) {
      this.passwordValidFlag = true
    } else {
      this.passwordValidFlag = false
    },
  },
  passwordCheckValid () {
    if (this.password === this.passwordCheck) {
      this.passwordCheckFlag = true
    } else {
      this.passwordCheckFlag = false
    },
  },
}
```

4) 회원가입 버튼 : ID,PW, address, callnum 등의 데이터를 받아 모두 저장

- HTML

```
<button type="button" class="orderBtn2" @click="signUpButton">회 원 가 입</button>
```

- JS

```
async signUpButton() {
  try {
    await axios.post('http://localhost:3000/create', {
      UserId: this.id, UserPw: this.password, UserName: this.name,
      UserPhNum1 : this.phone.replaceAll("-", "").substring(0,3),
      UserPhNum2 : this.phone.replaceAll("-", "").substring(3,7),
      UserPhNum3 : this.phone.replaceAll("-", "").substring(7,11),
      userEmail: this.email,
      UserSsn1 : this.ssn.replaceAll("-", "").substring(0,6),
      UserSsn2 : this.ssn.replaceAll("-", "").substring(6,13),
      Admin : 0,
      JoinDate : this.getFormatDate(this.JoinDate),
    });
    this.$router.push("/login");
    alert("회원가입 완료");
  } catch (err) {
    console.log(err)
  }
},
```

- `this.$router.push("/login");` : 회원가입버튼을 누르면 로그인("/login")페이지로 이동

4-2) 회원가입과의 연동 Method 확인

- `await axios.post('http://localhost:3000/crate'` 는 아래의 Router.js 와 product.js 로 연동되어 데이터를 전달

- Router.js 에 `JoinId` 함수와 경로 연동

```
router.post( path: '/create', joinId);
```

- product.js 의 `JoinId` 함수 : 데이터를 받으면 입력(insert)

```
export const joinId = (req,res) => {
  let query = "INSERT INTO user SET ?";
  let data = [req.body]
  db.query(query, data ,function(err, results){
    if(err) {
      console.log(err);
      res.send(err);
    } else {
      res.json(results);
    }
  }); }
```

4-3) JoinDate : 저장되기 전 메소드를 실행해 콜백

```
getFormatDate(date) {
  const YYYY = String(date.getFullYear())
  const MM = String((date.getMonth() + 1) >= 10 ? (date.getMonth() + 1): "0" + (date.getMonth() + 1))
  const dd = String(date.getDate() >= 10 ? date.getDate() : "0" + date.getDate())
  return YYYY + "." + MM + "." + dd
}
```